ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Οδηγός Μελέτης

Εισαγωγή στον Προγραμματισμό

Οι διαλέξεις του 2025–26, κεφάλαιο προς κεφάλαιο

Τμήμα Πληροφορικής και Τηλεπικοινωνιών
Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Ένα κεφάλαιο για κάθε διάλεξη: η θεωρία της διάλεξης σε κείμενο, τα κύρια σημεία, οδηγός ανάγνωσης στις σημειώσεις και στα εργαστήρια, και ασκήσεις από τις διαφάνειες, τα εργαστήρια, τις εργασίες και τα θέματα εξετάσεων.

progintro.github.io/study

Περιεχόμενα

Μέρος Α: Εργαλεία και πρώτα βήματα

Κεφάλαιο 0: Καλημέρα Κόσμε!

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να

- βρίσκετε τα εργαλεία του μαθήματος (site, Piazza, eclass, GitHub) και να ξέρετε πού να απευθύνετε μια ερώτηση·
- περιγράφετε πώς βαθμολογείται το μάθημα και γιατί οι ασκήσεις και το εργαστήριο μετράνε στην πράξη·
- ορίζετε τι είναι υπολογιστής, προγραμματισμός, πρόγραμμα και γλώσσα προγραμματισμού·
- εξηγείτε γιατί η ανθρώπινη γλώσσα δεν αρκεί για να δώσουμε εντολές σε έναν υπολογιστή (αμφισημία)·
- αναφέρετε λόγους για τους οποίους μαθαίνουμε C·
- διαβάζετε, γραμμή προς γραμμή, το πρόγραμμα Hello World και να το τρέχετε.

Προαπαιτούμενα: κανένα, αυτό είναι το πρώτο κεφάλαιο.

Χρόνος μελέτης: ~1 ώρα

Σύνοψη

Η πρώτη διάλεξη του μαθήματος «Εισαγωγή στον Προγραμματισμό» (K04) είναι κυρίως εισαγωγική: γνωρίζουμε το Τμήμα, τα διαδικαστικά του μαθήματος (ώρες, βαθμολογία, εργαστήριο, εργαλεία) και τον σκοπό του, που είναι να γίνετε junior software developers με προοπτικές. Στη συνέχεια θέτουμε τα θεμέλια: τι είναι ένας υπολογιστής, τι σημαίνει προγραμματισμός και τι είναι ένα πρόγραμμα. Βλέπουμε γιατί χρειαζόμαστε ειδικές γλώσσες προγραμματισμού (η ανθρώπινη γλώσσα είναι γεμάτη αμφισημίες) και γιατί το μάθημα διδάσκεται σε C, μια γλώσσα δημοφιλή εδώ και δεκαετίες και από τις ταχύτερες και οικονομικότερες σε ενέργεια. Κλείνουμε με το πρώτο μας πρόγραμμα, το κλασικό Hello World.

Θεωρία

§0.1 Γιατί προγραμματισμός

Η διάλεξη ξεκινά με το ερώτημα «Γιατί να ασχοληθείς με τον προγραμματισμό το 2025;» και δίνει τέσσερις ενδεικτικούς λόγους:

- **Ευελιξία**, θεματική και γεωγραφική: ο προγραμματισμός χρειάζεται σε κάθε επιστημονικό πεδίο και σε κάθε χώρα.

- **Στην αιχμή των εξελίξεων**: οι μεγάλες τεχνολογικές αλλαγές των τελευταίων δεκαετιών (προσωπικοί υπολογιστές, γονιδίωμα, διαδίκτυο) είναι σε μεγάλο βαθμό υπολογιστικές, και η πρόοδος επιταχύνεται.
- **Πνευματικά απαιτητική δουλειά**: κάθε πρόγραμμα είναι ένα μικρό πρόβλημα προς λύση.
- **Cool**.

Το Τμήμα Πληροφορικής και Τηλεπικοινωνιών του ΕΚΠΑ, σύμφωνα με τις διαφάνειες, βρίσκεται στα 250 καλύτερα πανεπιστήμια του κόσμου στην Επιστήμη Υπολογιστών (Computer Science), και το ΕΚΠΑ ήταν το πιο βελτιωμένο πανεπιστήμιο της Ευρώπης και 60 παγκοσμίως το 2024-25 σε ποιοτική εκπαίδευση (κατάταξη QS). Πάνω απ' όλα, είναι ευκαιρία να συναναστραφείτε ιδιαίτερα ταλαντούχους συμφοιτητές.

§0.2 Τα διαδικαστικά του μαθήματος

Site. Όλο το υλικό (διαφάνειες, ασκήσεις, εργαστήρια) βρίσκεται στο progintro.github.io. Το μάθημα διδάσκεται σε δύο τμήματα (άρτιοι / περιττοί αριθμοί μητρώου), με διδάσκοντες τον Θανάση Αυγερινό και τον Τάκη Σταματόπουλο· η ύλη, το εργαστήριο, οι ασκήσεις και το διαγώνισμα είναι ίδια και για τα δύο.

Ώρες. Οι διαλέξεις γίνονται Δευτέρα και Παρασκευή 9πμ–11πμ, στο Αμφιθέατρο (άρτιοι) και στην Α2 (περιττοί). Ώρες γραφείου:

Ημέρα	Ώρα	Αυγερινός	Σταματόπουλος
Δευτέρα	11πμ–12μμ	Α40	Α48
Παρασκευή	11πμ–12μμ	Α3	Α48

Βαθμολογία. Ο τελικός βαθμός υπολογίζεται ως εξής:

- πρωτοετείς: 50% Τελική Εξέταση + 30% Ασκήσεις + 20% Εργαστήριο·
- όλοι οι υπόλοιποι: 70% Τελική Εξέταση + 30% Ασκήσεις.

Ισχύει και μια **αναπροσαρμογή**: αν ο βαθμός των Ασκήσεων είναι πάνω από 3 μονάδες μεγαλύτερος από τον βαθμό της Τελικής Εξέτασης, τότε ο βαθμός των Ασκήσεων γίνεται Τελική Εξέταση + 3. Η Τελική Εξέταση είναι γραπτή με κλειστά βιβλία. Οι Ασκήσεις (εργασίες για το σπίτι) είναι προαιρετικές, αλλά η επίλυσή τους βοηθάει σημαντικά. Στο Εργαστήριο η παρουσία είναι υποχρεωτική για τους πρωτοετείς (μέχρι 2 απουσίες).

Μπορεί λοιπόν κανείς να περάσει χωρίς εργασίες και εργαστήριο; «In theory, theory and practice are the same. In practice, they are not.» Θεωρητικά ναι, στην πράξη όχι: όπως κανείς δεν μαθαίνει αναρρίχηση διαβάζοντας ένα εγχειρίδιο, έτσι και ο προγραμματισμός μαθαίνεται μόνο γράφοντας κώδικα.

Εργαστήριο. Γίνεται σε εβδομαδιαία βάση και ξεκινά την εβδομάδα της 6ης Οκτωβρίου. Γράφεστε στην εργαστηριακή ομάδα που σας ταιριάζει μέσω του eclass· αν δεν βρίσκετε θέση,

θα ανοίξουν και άλλες.

Εργαλεία του μαθήματος. Χρειάζεστε:

1. προσωπικό λογαριασμό email στο Gmail·
2. προσωπικό λογαριασμό στο GitHub, με ένα «cool αλλά και επαγγελματικό» ψευδώνυμο·
3. τον ακαδημαϊκό σας λογαριασμό, μέσω του webadm.uoa.gr·
4. εγγραφή στο Piazza, το φόρουμ επικοινωνίας του μαθήματος·
5. εγγραφή σε τμήμα εργαστηρίου στο eclass·
6. να συμπληρώσετε τη φόρμα του μαθήματος με τα στοιχεία σας.

Συνιστάται να έχετε υπολογιστή μαζί σας στο μάθημα, για να προγραμματίζετε ζωντανά.

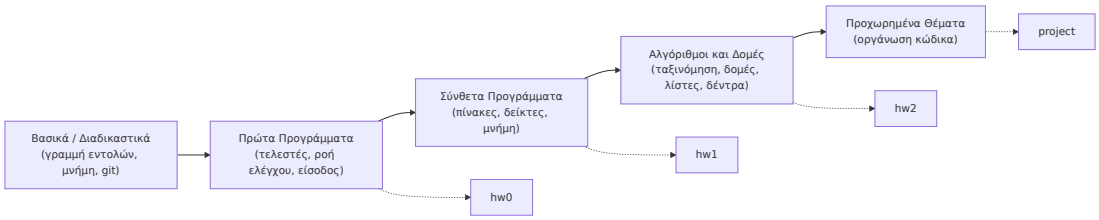
Έχω μια ερώτηση, τι κάνω; Η διάλεξη βάζει δύο επιλογές: (A) στέλνω email στον Αυγερινό ή στον Τάκη, ή (B) ελέγχω αν έχει ήδη απαντηθεί στο Piazza και, μόνο αν *δεν* έχει απαντηθεί, ποστάρω καινούρια ερώτηση. Ο δρόμος του μαθήματος είναι το (B), αφού γι' αυτό υπάρχει το Piazza: η απάντηση ωφελεί όλους και δεν επαναλαμβάνεται.

§0.3 Σκοπός και περιεχόμενο του μαθήματος

Το μάθημα έχει τρεις στόχους:

- εισαγωγή στις βασικές αρχές του προγραμματισμού και στην **αλγοριθμική σκέψη**·
- εμπειρία στη γραφή κώδικα και στην **αποσφαλμάτωση (debugging)** στη γλώσσα C·
- εξοικείωση με τα «εργαλεία της δουλειάς» (γραμμή εντολών, editor, μεταγλωττιστής, git).

Με άλλα λόγια, στο τέλος του εξαμήνου θέλουμε να είστε **junior software developers με προοπτικές**. Η ύλη χωρίζεται σε πέντε ενότητες, με τα εργαστήρια να τρέχουν παράλληλα και τις εργασίες (hw0, hw1, hw2 και ένα τελικό project) να κλείνουν τις ενότητες:

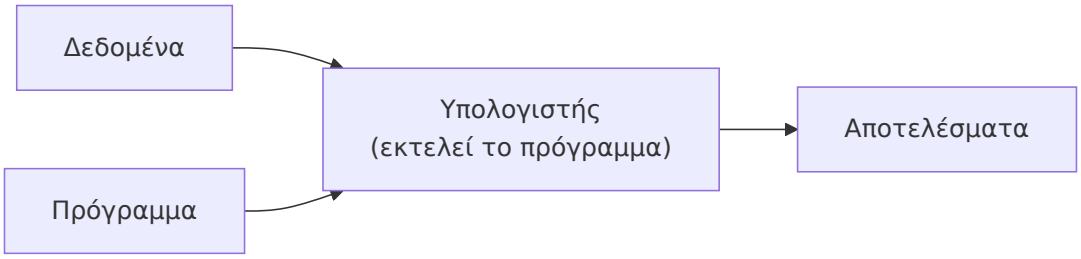


Σχήμα: οι ενότητες του μαθήματος και οι εργασίες που αντιστοιχούν σε καθεμία.

Στον χάρτη μαθημάτων του Τμήματος, το K04 είναι συνιστώμενο προαπαιτούμενο για τις Δομές Δεδομένων (K08) και τον Αντικειμενοστρεφή Προγραμματισμό (K10), και μέσω αυτών για μεγάλο μέρος του προγράμματος σπουδών (Λειτουργικά Συστήματα, Προγραμματισμός Συστήματος, Μεταγλωττιστές κ.ά.). Αν όλα αυτά ακούγονται πολλά: ένα βήμα τη φορά. Η καμπύλη Dunning-Kruger δείχνει ότι η αυτοπεποίθηση ενός αρχάριου συχνά ανεβαίνει απότομα («Peak of Mount Stupid»), πέφτει όταν συνειδητοποιεί πόσα δεν ξέρει («Valley of Despair») και μετά ανεβαίνει σταθερά όσο μεγαλώνει η πραγματική γνώση. Η απογοήτευση των πρώτων εβδομάδων είναι φυσιολογικό στάδιο, όχι ένδειξη ότι «δεν κάνετε».

§0.4 Τι είναι ο υπολογιστής

Υπολογιστής (computer) είναι μια κατασκευή που έχει την ικανότητα να επεξεργάζεται ένα σύνολο από **δεδομένα (data)** που του δίνονται και να παράγει τα απαιτούμενα **αποτελέσματα (results)**. Το *είδος* της επεξεργασίας το καθορίζει το πρόγραμμα με το οποίο τον έχουμε τροφοδοτήσει.



Σχήμα: ο υπολογιστής μετατρέπει δεδομένα σε αποτελέσματα, όπως ορίζει το πρόγραμμα.

Χρησιμοποιούμε υπολογιστές για δύο λόγους:

1. **ταχύτητα** στην επεξεργασία δεδομένων (π.χ. χρονοβόροι αριθμητικοί υπολογισμοί)·
2. **μνήμη** διαθέσιμη για αποθήκευση δεδομένων (π.χ. τραπεζικοί λογαριασμοί).

Πολλές εφαρμογές, όπως η πρόγνωση του καιρού, χρειάζονται και τα δύο ταυτόχρονα.

Οι σημειώσεις του μαθήματος συμπληρώνουν την εικόνα. Ένας υπολογιστής αποτελείται από την **κεντρική μονάδα επεξεργασίας (CPU)**, τη **μνήμη (memory)** για προσωρινή αποθήκευση δεδομένων και προγραμμάτων, τη **δευτερεύουσα μνήμη** (δίσκοι) για μόνιμη αποθήκευση και τις **μονάδες εισόδου/εξόδου** (πληκτρολόγιο, οθόνη). Όλη η πληροφορία αποθηκεύεται ως δυαδικά ψηφία, **bits** (0 και 1), οργανωμένα σε **bytes** των 8 bits. Τα προγράμματα ενός υπολογιστή είναι το **λογισμικό (software)** του, ενώ η ίδια η συσκευή είναι το **υλικό (hardware)**. Τη μνήμη και την αναπαράσταση των δεδομένων θα τις δούμε αναλυτικά στο Κεφάλαιο 2.

§0.5 Προγραμματισμός και πρόγραμμα

Προγραμματισμός (programming) είναι ο σαφής καθορισμός μιας διαδικασίας, σαν ένα σύνολο από εντολές (το πρόγραμμα), που περιγράφει λεπτομερώς τα βήματα που πρέπει να γίνουν για να επιλυθεί ένα πρόβλημα υπολογισμού.

Πρόγραμμα (program) είναι η καταγραφή της επίλυσης ενός προβλήματος. Κάθε εφαρμογή που χρησιμοποιείτε (ο browser, ένα παιχνίδι, το λειτουργικό σύστημα του κινητού σας) είναι πρόγραμμα. Όπως λέει ο Gusteau στο Ratatouille, «anyone can cook», και η διάλεξη το προσαρμόζει: *οποιοσδήποτε μπορεί να προγραμματίσει*, αλλά μόνο οι ατρόμητοι γίνονται σπουδαίοι.

Σύμφωνα με τις σημειώσεις, πίσω από κάθε πρόγραμμα βρίσκεται ένας **αλγόριθμος (algorithm)**: μια σαφώς καθορισμένη διαδικασία από εκτελέσιμα βήματα, που είναι εγγυημένο ότι θα

τερματίσει μετά από πεπερασμένο πλήθος βημάτων. Το πρόγραμμα είναι η διατύπωση ενός αλγορίθμου σε μια συγκεκριμένη γλώσσα προγραμματισμού.

§0.6 Αμφισημία και γλώσσες προγραμματισμού

Γιατί δεν δίνουμε απλώς τις οδηγίες στα ελληνικά ή στα αγγλικά; Επειδή **η ανθρώπινη γλώσσα έχει αμφισημίες (ambiguities)**. Το quiz της διάλεξης, «Ποιο είναι το πιο ψηλό βουνό του κόσμου;», έχει τρεις σωστές απαντήσεις, ανάλογα με το τι εννοούμε «ψηλό»:

Ερμηνεία του «ψηλό»	Βουνό
μεγαλύτερο υψόμετρο από τη μέση στάθμη της θάλασσας (8.848 m)	Έβερεστ
πιο μακριά από το κέντρο της Γης	Τσιμποράσο (Chimborazo)
μεγαλύτερο ύψος από τη βάση ως την κορυφή (10.210 m)	Μάουνα Κέα (Mauna Kea)

Οι αμφισημίες είναι δύο ειδών. **Λεξιλογική αμφισημία (lexical ambiguity)** έχουμε όταν μία λέξη έχει δύο ή περισσότερες σημασίες: στο «I saw her duck.» το *duck* είναι είτε «πάπια» είτε «έσκυψε». **Συντακτική αμφισημία (syntactic ambiguity)** έχουμε όταν μια πρόταση ή ακολουθία λέξεων επιδέχεται περισσότερες από μία ερμηνείες: το «The chicken is ready to eat.» σημαίνει είτε ότι το φαγητό είναι έτοιμο είτε ότι η κότα πεινάει.

Ένας υπολογιστής δεν μπορεί να «μαντέψει» τι εννοούμε. Γι' αυτό επινοήσαμε τις γλώσσες προγραμματισμού. **Γλώσσα προγραμματισμού (programming language)** είναι μια γλώσσα που μας επιτρέπει να επικοινωνούμε εντολές στον υπολογιστή. Μια καλή γλώσσα προγραμματισμού **δεν επιτρέπει αμφισημία**: κάθε πρόγραμμα έχει ακριβώς μία σημασία. Ως bonus, η γλώσσα προγραμματισμού μαζί με τον υπολογιστή μπορεί να λύσει και αμφισημίες *ανάμεσα σε προγραμματιστές*: αν διαφωνείτε για το τι κάνει ένα κομμάτι κώδικα, το τρέχετε και βλέπετε.

Οι σημειώσεις διακρίνουν τις γλώσσες **χαμηλού επιπέδου** (γλώσσα μηχανής, assembly), που είναι κοντά στον επεξεργαστή, από τις γλώσσες **υψηλού επιπέδου** (C, Python, Java κ.ά.), που είναι κοντά στον τρόπο που σκεφτόμαστε. Από μια γλώσσα υψηλού επιπέδου χρειαζόμαστε μηχανισμούς για είσοδο και έξοδο, ακολουθίες βημάτων, αποθήκευση τιμών στη μνήμη, επιλογή με βάση συνθήκη, επανάληψη και «πακετάρισμα» συχνών λειτουργιών. Όλα αυτά τα μαθαίνουμε στη C τα επόμενα κεφάλαια.

§0.7 Η γλώσσα C

Η C δημιουργήθηκε γύρω στο 1970 από τον **Dennis Ritchie**, ο οποίος μαζί με τον **Ken Thompson** δημιούργησε και το λειτουργικό σύστημα UNIX και τη γλώσσα B, την προκάτοχο της C. Τη μαθαίνουμε επειδή είναι:

- **απαραίτητη στο πρόγραμμα σπουδών**: πάνω της χτίζουν πολλά επόμενα μαθήματα·
- **δημοφιλής**: σύμφωνα με τον δείκτη TIOBE, είναι από τις πιο δημοφιλείς γλώσσες εδώ και δεκαετίες (το γράφημα της διάλεξης δείχνει την C στις πρώτες θέσεις από το 2001 ως το 2025, ενώ πρόσφατα την έχει ξεπεράσει η Python)·
- **χρησιμοποιείται παντού και (σχεδόν) για τα πάντα**: λειτουργικά συστήματα, ενσωματωμένα συστήματα, μεταγλωττιστές, βάσεις δεδομένων.

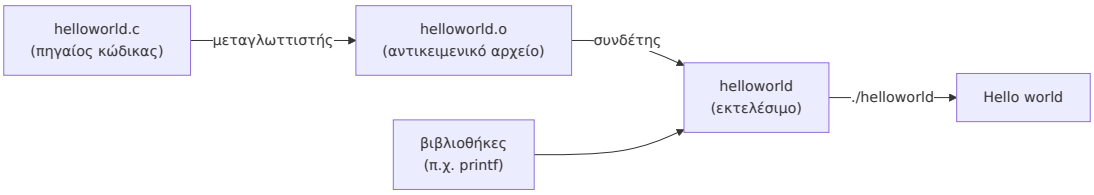
Η C είναι επίσης το «αγωνιστικό» των γλωσσών. Σε μια σύγκριση πολλών γλωσσών στα ίδια προβλήματα (βλ. «Programming Language Shootout» στο «Διάβασμα»), με τιμές κανονικοποιημένες ως προς την καλύτερη γλώσσα (1.00 = η καλύτερη):

Μέτρο	C	Rust	C++	Java	Python
Ενέργεια	1.00 (1η)	1.03	1.34	1.98	75.88
Χρόνος	1.00 (1η)	1.04	1.56	1.89	71.90
Μνήμη	1.17 (3η)	1.54	1.34	6.01	2.80

Στη μνήμη η C έρχεται τρίτη, μετά την Pascal (1.00) και τη Go (1.05). Η Python, αν και σήμερα η πιο δημοφιλής γλώσσα, είναι περίπου 72 φορές πιο αργή από τη C σε αυτή τη σύγκριση. Η C σας δίνει ταχύτητα και έλεγχο, με αντίτιμο ότι πρέπει να καταλαβαίνετε τι κάνει ο υπολογιστής από κάτω.

§0.8 Από τον πηγαίο κώδικα στο εκτελέσιμο

Ο υπολογιστής δεν εκτελεί απευθείας κώδικα C. Γράφουμε το πρόγραμμα σε ένα αρχείο κειμένου, το **πηγαίο αρχείο (source file)**, π.χ. `helloworld.c`. Ο **μεταγλωττιστής (compiler)** το μετατρέπει σε **αντικειμενικό αρχείο (object file)** σε γλώσσα μηχανής, και ο **συνδέτης (linker)** ενώνει τα αντικειμενικά αρχεία με τις **βιβλιοθήκες (libraries)** που χρειάζονται (π.χ. αυτή που περιέχει την `printf`) σε ένα **εκτελέσιμο αρχείο (executable)**, που μπορεί να τρέξει ο επεξεργαστής. Στο μάθημα χρησιμοποιούμε τον μεταγλωττιστή `gcc`, που κάνει και τα δύο βήματα με μία εντολή.



Σχήμα: μεταγλώττιση και σύνδεση ενός προγράμματος C.

Στη διάλεξη τρέξαμε το πρώτο πρόγραμμα σε έναν **online compiler**, μια ιστοσελίδα που κάνει όλα τα παραπάνω για λογαριασμό σας. Από το επόμενο κεφάλαιο (Κεφάλαιο 1) δουλεύουμε στη γραμμή εντολών, όπου καλούμε τον `gcc` μόνοι μας.

§0.9 Η δομή του Hello World

Το πρώτο πρόγραμμα που γράφει κανείς σε μια νέα γλώσσα, κατά παράδοση, τυπώνει «Hello world». Στη C μοιάζει έτσι:

```c
/* File: helloworld.c */
#include <stdio.h>

int main() {
  printf("Hello world\n");
}
```

Κάθε γραμμή έχει ρόλο:

- `/* ... */` είναι **σχόλιο (comment)**: ό,τι βρίσκεται ανάμεσα στο `/*` και στο `*/` αγνοείται από τον μεταγλωττιστή. Γράφουμε σχόλια για τους ανθρώπους που θα διαβάσουν τον κώδικα.
- `#include <stdio.h>` είναι οδηγία προς τον **προεπεξεργαστή (preprocessor)** να συμπεριλάβει εκεί τα περιεχόμενα του **αρχείου επικεφαλίδας (header file)** `stdio.h` (standard input/output), που δηλώνει τις συναρτήσεις εισόδου/εξόδου, μεταξύ αυτών και την `printf`.
- `int main() { ... }` ορίζει τη **συνάρτηση (function)** `main`. Κάθε πρόγραμμα C έχει ακριβώς μία `main`, και από εκεί ξεκινά η εκτέλεση. Οι εντολές της βρίσκονται μέσα στις αγκύλες `{ }`. Για το `int` θα μιλήσουμε σε επόμενο κεφάλαιο.
- `printf("Hello world\n");` καλεί τη συνάρτηση εξόδου `printf`, που τυπώνει τη **συμβολοσειρά (string)** ανάμεσα στα διπλά εισαγωγικά (χωρίς τα εισαγωγικά). Το `\n` είναι ειδική ακολουθία που σημαίνει **αλλαγή γραμμής (newline)**. Η εντολή τελειώνει με `;`.

Το εργαστήριο γράφει το ίδιο πρόγραμμα με μια επιπλέον γραμμή `return 0;` στο τέλος της `main`. Και οι δύο μορφές είναι σωστές· τι σημαίνει η τιμή που επιστρέφει η `main` θα το δούμε στο Κεφάλαιο 3.

Παραδείγματα

§0.10 Hello World σε online compiler

Εφαρμόζει: «Η δομή του Hello World», «Από τον πηγαίο κώδικα στο εκτελέσιμο».

Η διάλεξη προτείνει να τρέξετε το πρόγραμμα αμέσως, χωρίς να εγκαταστήσετε τίποτα, στον online compiler του Programiz (σύνδεσμος στο «Διάβασμα»). Αντιγράψτε τον κώδικα της προηγούμενης ενότητας, πατήστε «Run» και στο παράθυρο εξόδου θα δείτε:

```
Hello world
```

Δοκιμάστε μικρές αλλαγές για να δείτε τι συμβαίνει: αλλάξτε το μήνυμα, βάλτε δύο `printf` τη μία μετά την άλλη, ή αφαιρέστε το `\n` και δείτε πού πάει η επόμενη έξοδος.

§0.11 Hello World στη γραμμή εντολών

Εφαρμόζει: «Από τον πηγαίο κώδικα στο εκτελέσιμο».

Στο εργαστήριο (και από το επόμενο κεφάλαιο) γράφετε το πρόγραμμα σε ένα αρχείο με έναν κειμενογράφο, το μεταγλωττίζετε με τον `gcc` και τρέχετε το εκτελέσιμο:

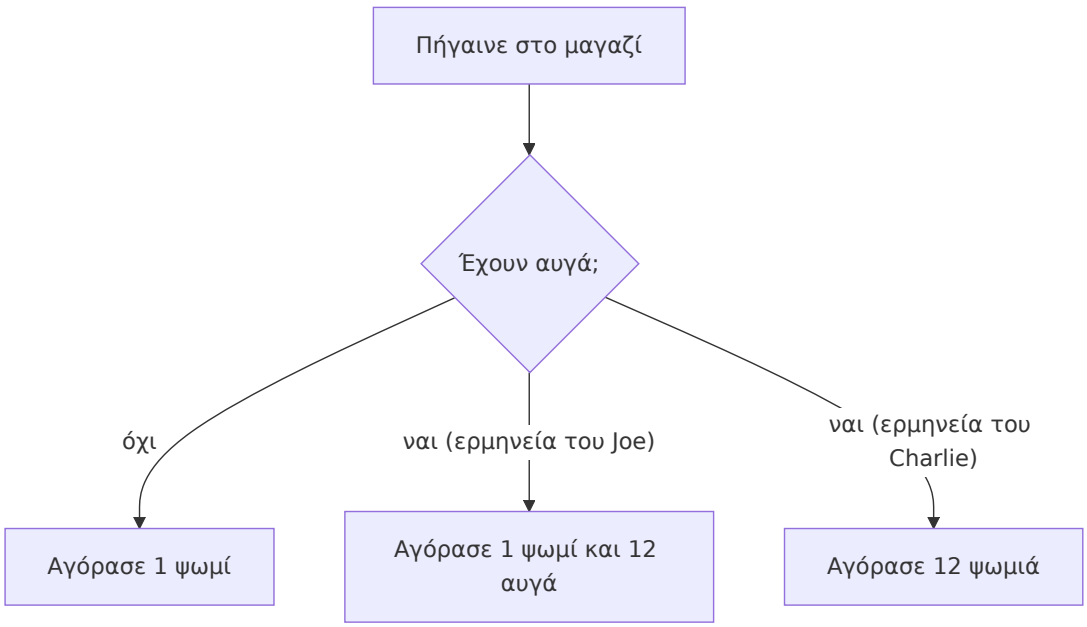
```
$ gcc -o helloworld helloworld.c
$ ./helloworld
Hello world
$
```

Η επιλογή `-o helloworld` δίνει όνομα στο εκτελέσιμο· χωρίς αυτήν, ο `gcc` το ονομάζει `a.out`. Το `./` λέει στο shell να βρει το πρόγραμμα στον τρέχοντα κατάλογο.

§0.12 Δύο αναγνώσεις μιας οδηγίας

Εφαρμόζει: «Αμφισημία και γλώσσες προγραμματισμού».

Η διάλεξη εξηγεί με ένα ανέκδοτο γιατί επινοήσαμε τις γλώσσες προγραμματισμού. Ο Joe λέει στον φίλο του, τον προγραμματιστή Charlie: «Go to the store and buy a loaf of bread. If they have eggs, buy a dozen.» Ο Charlie γυρίζει με 12 φραντζόλες ψωμί. Η φράση «buy a dozen» δεν λέει *δώδεκα από τι*:



Σχήμα: η ίδια οδηγία, δύο διαφορετικά «προγράμματα».

Σε μια γλώσσα προγραμματισμού, όπως στη C, ο προγραμματιστής θα έπρεπε να γράψει ρητά *τι* αγοράζει και *πόσο*, οπότε η αμφισημία δεν θα μπορούσε να υπάρξει.

§0.13 Πού να εξασκηθείτε

Η διάλεξη προτείνει πηγές εξάσκησης πέρα από τις ασκήσεις του μαθήματος:

- **πρακτικά προβλήματα, σε επαφή με τη βιομηχανία**: hackerrank.com, leetcode.com·
- **ανταγωνιστικός προγραμματισμός και αλγόριθμοι (ή και μαθηματικά)**: adventofcode.com, codechef.com, codewars.com, topcoder.com, usaco.org, projecteuler.net·
- **μαθήματα**: Coursera, edX, Udacity, και πολλά άλλα με μια αναζήτηση.

§0.14 Για την επόμενη φορά

Πριν από την επόμενη διάλεξη:

- διαβάστε τις σημειώσεις του κ. Σταματόπουλου μέχρι τη σελίδα 19 (βλ. «Διάβασμα»)·
- ολοκληρώστε τις εγγραφές στα εργαλεία του μαθήματος και γραφτείτε σε κάποιο εργαστήριο·
- αν θέλετε, φέρτε υπολογιστή μαζί σας·
- διαβάστε τα άρθρα για τις γλώσσες προγραμματισμού, τις γλωσσικές αμφισημίες και την ιστορία της C.

Κύρια σημεία

1. Το μάθημα στοχεύει στις βασικές αρχές του προγραμματισμού και στην αλγοριθμική σκέψη, στη γραφή και αποσφαλμάτωση κώδικα σε C και στην εξοικείωση με τα εργαλεία της δουλειάς.
2. Όλο το υλικό βρίσκεται στο progintro.github.io, και τα δύο τμήματα έχουν την ίδια ύλη, εργαστήριο, ασκήσεις και διαγώνισμα.
3. Για τους πρωτοετείς ο βαθμός είναι 50% Τελική Εξέταση + 30% Ασκήσεις + 20% Εργαστήριο, και ο βαθμός των Ασκήσεων δεν μπορεί να ξεπεράσει την Τελική Εξέταση κατά περισσότερες από 3 μονάδες.
4. Θεωρητικά μπορείτε να περάσετε χωρίς εργασίες και εργαστήριο, στην πράξη όχι: ο προγραμματισμός μαθαίνεται μόνο γράφοντας κώδικα.
5. Για ερωτήσεις ψάχνετε πρώτα στο Piazza και ποστάρετε καινούρια ερώτηση μόνο αν δεν έχει ήδη απαντηθεί.
6. Υπολογιστής είναι μια κατασκευή που επεξεργάζεται δεδομένα και παράγει αποτελέσματα· τον χρησιμοποιούμε για την ταχύτητα και τη μνήμη του.
7. Προγραμματισμός είναι ο σαφής καθορισμός μιας διαδικασίας, ως σύνολο εντολών, που λύνει ένα πρόβλημα υπολογισμού· πρόγραμμα είναι η καταγραφή αυτής της λύσης.
8. Η ανθρώπινη γλώσσα έχει λεξιλογικές και συντακτικές αμφισημίες· μια καλή γλώσσα προγραμματισμού δεν επιτρέπει καμία.

9. Μαθαίνουμε C γιατί είναι απαραίτητη στο πρόγραμμα σπουδών, δημοφιλής και χρησιμοποιείται παντού, και είναι από τις πιο γρήγορες και ενεργειακά αποδοτικές γλώσσες.
10. Ο μεταγλωττιστής και ο συνδέτης μετατρέπουν τον πηγαίο κώδικα C σε εκτελέσιμο πρόγραμμα.
11. Κάθε πρόγραμμα C έχει ακριβώς μία συνάρτηση `main`, από την οποία ξεκινά η εκτέλεση· η `printf` τυπώνει μια συμβολοσειρά και το `\n` αλλάζει γραμμή.
12. Η πρόοδος στον προγραμματισμό γίνεται ένα βήμα τη φορά· η αρχική απογοήτευση είναι φυσιολογική.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
υπολογιστής	computer	Κατασκευή που επεξεργάζεται δεδομένα και παράγει αποτελέσματα
προγραμματισμός	programming	Σαφής καθορισμός διαδικασίας που λύνει πρόβλημα υπολογισμού
πρόγραμμα	program	Η καταγραφή της επίλυσης ενός προβλήματος, ως σύνολο εντολών
αλγόριθμος	algorithm	Σαφής διαδικασία από εκτελέσιμα βήματα που τερματίζει
γλώσσα προγραμματισμού	programming language	Γλώσσα χωρίς αμφισημίες για εντολές σε υπολογιστή
αμφισημία	ambiguity	Όταν μια λέξη ή πρόταση έχει περισσότερες από μία σημασίες
λογισμικό / υλικό	software / hardware	Τα προγράμματα / η ίδια η συσκευή
αποσφαλμάτωση	debugging	Εντοπισμός και διόρθωση λαθών σε πρόγραμμα
πηγαίο αρχείο	source file	Αρχείο κειμένου με τον κώδικα, π.χ. `helloworld.c`
μεταγλωττιστής	compiler	Μετατρέπει πηγαίο κώδικα σε γλώσσα μηχανής (π.χ. `gcc`)
συνδέτης	linker	Ενώνει αντικειμενικά αρχεία και βιβλιοθήκες σε εκτελέσιμο
εκτελέσιμο	executable	Αρχείο που μπορεί να τρέξει ο επεξεργαστής
σχόλιο	comment	Κείμενο μέσα σε `/* */` που αγνοεί ο μεταγλωττιστής
αρχείο επικεφαλίδας	header file	Αρχείο με δηλώσεις, π.χ. `stdio.h`
συμβολοσειρά	string	Κείμενο ανάμεσα σε διπλά εισαγωγικά, π.χ. `"Hello world\n"`

Ελληνικά	English	Σύντομος ορισμός
αλλαγή γραμμής	newline	Ο χαρακτήρας `\n`

Διάβασμα

- **Διαφάνειες**: Διάλεξη 0, σελ. 1–36. Γιατί προγραμματισμός: σελ. 3–6· διαδικαστικά: σελ. 7–16· σκοπός και περιεχόμενο: σελ. 17–19· υπολογιστής και προγραμματισμός: σελ. 20–24· αμφισημία και γλώσσες: σελ. 25–27· η C: σελ. 28–31· Hello World: σελ. 32· εξάσκηση και επόμενη φορά: σελ. 33–34.
- **Σημειώσεις**: Οι διαφάνειες ζητούν τις σημειώσεις του κ. Σταματόπουλου μέχρι τη σελίδα 19 (K04):
  - Κεφάλαιο 0: Εισαγωγή, ολόκληρο, ιδίως «Γενικά περί υπολογιστών», «Γενικά περί προγραμματισμού υπολογιστών», «Πώς να μάθουμε να προγραμματίζουμε;», «Η δομή του υπολογιστή», «Λογισμικό και γλώσσες προγραμματισμού», «Πώς κατασκευάζουμε εκτελέσιμο πρόγραμμα;» (K04, σελ. 1–17)
  - Κεφάλαιο 1: Πρώτα προγράμματα σε C, ενότητα «Η γλώσσα προγραμματισμού C» (K04, σελ. 18), και για το Hello World η «Καλημέρα κόσμε της C» (K04, σελ. 19)
- **Εργαστήριο**: Εργαστήριο 0: Βήματα 1–2 (webmail, Piazza) για τα εργαλεία του μαθήματος, Βήμα 7 `hello.c`, Άσκηση 1 `about.c`
- **Άλλα**:
  - Site του μαθήματος: progintro.github.io· Piazza· eclass και εγγραφή σε εργαστηριακή ομάδα· webadm
  - Online compiler (Programiz)
  - TIOBE index· Programming Language Shootout (Benchmarks Game)
  - Wikipedia: Programming language, List of linguistic example sentences (γλωσσικές αμφισημίες)
  - History of C and Applications
  - Εξάσκηση: HackerRank, LeetCode, Advent of Code, CodeChef, Codewars, Topcoder, USACO, Project Euler

Συχνά λάθη

- **Email αντί για Piazza**. Ερωτήσεις για την ύλη ή τις ασκήσεις πάνε στο Piazza, αφού πρώτα ψάξετε αν έχουν ήδη απαντηθεί.
- **«Οι ασκήσεις είναι προαιρετικές, άρα τις παραλείπω»**. Χάνετε το 30% του βαθμού και, κυρίως, την εξάσκηση που χρειάζεται η Τελική Εξέταση.
- **Ξεχασμένο `;`**. `printf("Hello world\n")` χωρίς `;` δεν μεταγλωττίζεται (`expected ';'`). Διόρθωση: κάθε εντολή τελειώνει με `;`.
- **Χωρίς `#include <stdio.h>`**. Ο μεταγλωττιστής δεν ξέρει την `printf` και προειδοποιεί (ή σταματά) με μήνυμα για `implicit declaration of function 'printf'`.

- **Ξεχασμένο** `\n`. Χωρίς αλλαγή γραμμής, η επόμενη έξοδος (ή το prompt του shell) κολλάει στο τέλος του μηνύματος: `Hello world$`.
- **Λάθος εισαγωγικά**. Οι συμβολοσειρές θέλουν διπλά εισαγωγικά `"..."`· με μονά `'Hello'` ή «έξυπνα» εισαγωγικά από κειμενογράφο εγγράφων ο κώδικας δεν μεταγλωττίζεται.
- **Κεφαλαία στα ονόματα**. Η C ξεχωρίζει κεφαλαία και πεζά: `Main` ή `Printf` δεν είναι το ίδιο με `main` και `printf`.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K0.6** Έχω μια ερώτηση για το μάθημα (57% σωστές): Το 23% θα πόσταρε κατευθείαν στο Piazza με τίτλο «Ερώτηση»: το Piazza είναι το σωστό κανάλι, αλλά πρώτα ψάχνουμε αν έχει ήδη απαντηθεί, και ο τίτλος πρέπει να λέει τι ρωτάμε.
- **K0.5** Το χαρακτηριστικό της C (61% σωστές): Το 27% διάλεξε «Γρήγορη/Αποδοτική»: η ταχύτητα είναι πράγματι ένα από τα πλεονεκτήματα της C, αλλά όχι το μόνο που τονίστηκε.

Ερωτήσεις κατανόησης

- **E0.1** Ποιοι είναι οι δύο βασικοί λόγοι για τους οποίους χρησιμοποιούμε υπολογιστές;[1]
- **E0.2** Δώστε τον ορισμό του προγραμματισμού που δίνει η διάλεξη.[2]
- **E0.3** Τι διαφέρει η λεξιλογική από τη συντακτική αμφισημία; Δώστε ένα παράδειγμα από την καθεμία.[3]
- **E0.4** Γιατί δεν μπορούμε να προγραμματίσουμε έναν υπολογιστή σε φυσική γλώσσα;[4]
- **E0.5** Ποιος δημιούργησε τη C και περίπου πότε;[5]
- **E0.6** Τι κάνει η γραμμή `#include <stdio.h>` στο Hello World;[6]
- **E0.7** Από ποια συνάρτηση ξεκινά η εκτέλεση ενός προγράμματος C, και πόσες τέτοιες συναρτήσεις έχει ένα πρόγραμμα;[7]
- **E0.8** Τι θα αλλάξει στην έξοδο αν σβήσετε το `\n` από το `printf("Hello world\n");`;[8]

[1] Την ταχύτητα στην επεξεργασία δεδομένων και τη μνήμη που διαθέτουν για αποθήκευση δεδομένων.

[2] Ο σαφής καθορισμός μιας διαδικασίας, σαν ένα σύνολο από εντολές (το πρόγραμμα), που περιγράφει λεπτομερώς τα βήματα που πρέπει να γίνουν για να επιλυθεί ένα πρόβλημα υπολογισμού.

[3] Λεξιλογική: μία λέξη με πολλές σημασίες («I saw her duck.»). Συντακτική: μια πρόταση που διαβάζεται με πολλούς τρόπους («The chicken is ready to eat.»).

[4] Επειδή η φυσική γλώσσα έχει αμφισημίες και ο υπολογιστής δεν μπορεί να μαντέψει ποια ερμηνεία εννοούμε· μια γλώσσα προγραμματισμού δεν επιτρέπει αμφισημία.

[5] Ο Dennis Ritchie, γύρω στο 1970 (μαζί με τον Ken Thompson δημιούργησαν και το UNIX και τη γλώσσα B).

[6] Ζητά από τον προεπεξεργαστή να συμπεριλάβει το αρχείο επικεφαλίδας `stdio.h`, που δηλώνει τις συναρτήσεις εισόδου/εξόδου όπως η `printf`.

[7] Από τη `main`· κάθε πρόγραμμα C έχει ακριβώς μία.

[8] Δεν θα αλλάξει γραμμή μετά το μήνυμα, οπότε ό,τι τυπωθεί μετά (π.χ. το prompt του shell) θα εμφανιστεί στην ίδια γραμμή.

- **E0.9** Ποια είναι η δουλειά του μεταγλωττιστή και ποια του συνδέτη;[9]
- **E0.10** Ένας πρωτοετής έχει 4 στις Ασκήσεις (με άριστα το 10) και 9 στην Τελική Εξέταση. Ισχύει η αναπροσαρμογή;[10]

Kahoot από το αμφιθέατρο (K0.1–K0.6)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K0.1** Τι κάνει η printf: 94% σωστές απαντήσεις
- **K0.2** Εκτύπωση με νέα γραμμή: 86% σωστές απαντήσεις
- **K0.3** Η φόρμα του μαθήματος: 85% σωστές απαντήσεις
- **K0.4** Τα εργαλεία του μαθήματος: 82% σωστές απαντήσεις
- **K0.5** Το χαρακτηριστικό της C: 61% σωστές απαντήσεις
- **K0.6** Έχω μια ερώτηση για το μάθημα: 57% σωστές απαντήσεις

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A0.1–A0.7)

- **A0.1** Έχω μια ερώτηση, τι κάνω;: Διάλεξη 0: Καλημέρα Κόσμε!, διαφάνεια 16 · ★☆☆ · multiple-choice · `slides-lec00-ask-question`
- **A0.2** Ψωμί και αυγά: Διάλεξη 0: Καλημέρα Κόσμε!, διαφάνεια 26 · ★☆☆ · short-answer · `slides-lec00-bread-and-eggs`
- **A0.3** Hello World σε online compiler: Διάλεξη 0: Καλημέρα Κόσμε!, διαφάνεια 32 · ★☆☆ · tooling · `slides-lec00-hello-world`
- **A0.4** Το πιο ψηλό βουνό του κόσμου: Διάλεξη 0: Καλημέρα Κόσμε!, διαφάνεια 25 · ★☆☆ · short-answer · `slides-lec00-highest-mountain`
- **A0.5** Περνάω χωρίς εργασίες και εργαστήριο;: Διάλεξη 0: Καλημέρα Κόσμε!, διαφάνεια 11 · ★☆☆ · short-answer · `slides-lec00-pass-without-labs`
- **A0.6** Παραδείγματα προγραμμάτων: Διάλεξη 0: Καλημέρα Κόσμε!, διαφάνεια 23 · ★☆☆ · short-answer · `slides-lec00-programs-you-know`
- **A0.7** Γιατί προγραμματισμός το 2025;: Διάλεξη 0: Καλημέρα Κόσμε!, διαφάνεια 5 · ★☆☆ · short-answer · `slides-lec00-why-programming`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A1.7** Μεταγλώττιση, σύνδεση και εκτέλεση: Εργαστήριο 1, Βήμα 2 · ★☆☆ · tooling · `lab-lab01-step2-compile-link`

[9] Ο μεταγλωττιστής μετατρέπει τον πηγαίο κώδικα σε αντικειμενικό αρχείο σε γλώσσα μηχανής· ο συνδέτης ενώνει αντικειμενικά αρχεία και βιβλιοθήκες σε εκτελέσιμο.

[10] Όχι. Η αναπροσαρμογή εφαρμόζεται μόνο όταν οι Ασκήσεις ξεπερνούν την Τελική Εξέταση κατά περισσότερες από 3 μονάδες· εδώ είναι μικρότερες.

- **A2.9** Στοιχεία φοιτητή (Παλιό θέμα): Εργαστήριο 0, Άσκηση 1 · ★☆☆ · programming · `lab-lab00-about`
- **A26.1** Τα στάδια του C build process: How to Make? (προσκεκλημένη διάλεξη), διαφάνεια 5 · ★☆☆ · short-answer · `slides-lecmake-build-pipeline`

Κεφάλαιο 1: Η Γραμμή Εντολών

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να

- περιγράφετε τα επίπεδα ενός υπολογιστικού συστήματος (χρήστες, εφαρμογές, λειτουργικό σύστημα, υλικό) και τι αναλαμβάνει το λειτουργικό σύστημα·
- εξηγείτε τη διαφορά ανάμεσα σε GUI και CLI, και ανάμεσα σε kernel και shell·
- τρέχετε ένα πρόγραμμα από το shell με ορίσματα και να ξεχωρίζετε πρόγραμμα, ορίσματα και έξοδο·
- συνδέεστε σε απομακρυσμένο μηχάνημα με `ssh`·
- κινείστε στο σύστημα αρχείων και να διαχειρίζεστε αρχεία με `pwd`, `ls`, `cd`, `cat`, `mkdir`, `cp`, `rm`, `rmdir`·
- μεταγλωττίζετε και να τρέχετε ένα πρόγραμμα C με `gcc` και να εξηγείτε κάθε γραμμή του Hello World, μαζί με τον κωδικό εξόδου (`echo $?`).

Προαπαιτούμενα: Κεφάλαιο 0

Χρόνος μελέτης: ~2 ώρες

Σύνοψη

Στην πρώτη διάλεξη γράψαμε το πρώτο μας πρόγραμμα· σε αυτή μαθαίνουμε το περιβάλλον μέσα στο οποίο θα γράφουμε, θα μεταγλωττίζουμε και θα τρέχουμε όλα τα προγράμματα του μαθήματος. Ξεκινάμε από τη δομή ενός υπολογιστικού συστήματος και τον ρόλο του λειτουργικού συστήματος, γνωρίζουμε το Linux και τη γραμμή εντολών (CLI), και βλέπουμε πώς το shell τρέχει προγράμματα με ορίσματα. Μαθαίνουμε να συνδεόμαστε απομακρυσμένα με `ssh`, να κινούμαστε στο σύστημα αρχείων και να χειριζόμαστε αρχεία με τις βασικές εντολές. Τέλος, μεταγλωττίζουμε με τον `gcc` το Hello World και το αναλύουμε γραμμή προς γραμμή. Η γραμμή εντολών είναι το βασικό εργαλείο του μαθήματος: σε αυτήν θα γίνονται τα εργαστήρια, οι εργασίες και οι εξετάσεις.

Θεωρία

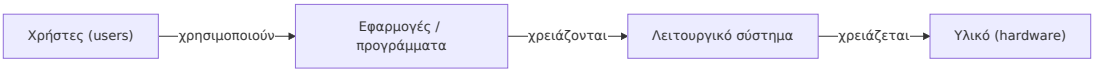
§1.1 Γιατί C;

Η διάλεξη άνοιξε με ένα ερώτημα από το Quora: γιατί τα πανεπιστήμια διδάσκουν ακόμα C και C++; Η απάντηση που δείχνουν οι διαφάνειες είναι ότι σχεδόν όλο το λογισμικό που χρησιμοποιούμε καθημερινά είναι γραμμένο σε C, C++ ή σε γλώσσες που προέρχονται από

αυτές: τα λειτουργικά συστήματα, οι browsers, οι οδηγοί (drivers) του πληκτρολογίου και του ποντικιού. Η C είναι, κατά την παρομοίωση, «το νερό» μέσα στο οποίο κολυμπάμε χωρίς να το βλέπουμε, και ένας προγραμματιστής πρέπει να ξέρει τι είναι αυτό το νερό. Όπως θα δούμε, και το Linux και τα βασικά εργαλεία της γραμμής εντολών είναι γραμμένα σε C.

§1.2 Αρχιτεκτονική υπολογιστικών συστημάτων

Ένα σύγχρονο υπολογιστικό σύστημα οργανώνεται σε επίπεδα, όπου κάθε επίπεδο στηρίζεται στο επόμενο:



Σχήμα: τα επίπεδα ενός υπολογιστικού συστήματος.

- Οι **χρήστες (users)** χρησιμοποιούν **εφαρμογές ή προγράμματα (applications / programs)**: Word, Excel, Outlook, Gmail, Netflix κ.ο.κ.
- Οι εφαρμογές χρειάζονται ένα **λειτουργικό σύστημα (operating system)**: Windows, Linux, macOS.
- Το λειτουργικό σύστημα χρειάζεται το **υλικό (hardware)**: laptop, desktop, κινητό, tablet, και περιφερειακά όπως οθόνη, ηχεία, ακουστικά, εκτυπωτής.

Μια πιο λεπτομερής εικόνα βάζει ανάμεσα στο λειτουργικό σύστημα και στο υλικό το **λογισμικό συστήματος (system software)**, όπως οι οδηγοί συσκευών, που «μιλούν» απευθείας με κάθε συσκευή. Οι εφαρμογές λέγονται αντίστοιχα **λογισμικό εφαρμογών (application software)**.

§1.3 Λειτουργικά συστήματα

Λειτουργικό σύστημα (operating system, OS) είναι το λογισμικό του υπολογιστή που είναι υπεύθυνο για:

1. τη **διαχείριση του υλικού** (των συσκευών)·
2. τη **διαχείριση των πόρων** του συστήματος: πόση μνήμη μπορεί να χρησιμοποιήσει ένα πρόγραμμα, πότε θα πάρει κύκλους στον επεξεργαστή κ.ο.κ.·
3. την **προσφορά υπηρεσιών** στα προγράμματα των χρηστών: αποθήκευση πληροφοριών στον δίσκο, μεταφορά πληροφοριών μέσω δικτύου κ.ο.κ.

Γιατί να μην τα κάνει αυτά το δικό μας πρόγραμμα; Γιατί σε έναν υπολογιστή τρέχουν ταυτόχρονα πολλά προγράμματα, που μοιράζονται την ίδια μνήμη, τον ίδιο επεξεργαστή και τον ίδιο δίσκο. Κάποιος πρέπει να μοιράζει δίκαια τους πόρους και να εμποδίζει ένα πρόγραμμα να πειράξει τα δεδομένα ενός άλλου. Επιπλέον, αν κάθε πρόγραμμα έπρεπε να ξέρει πώς δουλεύει κάθε μοντέλο δίσκου ή κάρτας δικτύου, κανείς δεν θα μπορούσε να γράψει ούτε ένα Hello World. Το λειτουργικό σύστημα κάνει αυτή τη δουλειά μία φορά, για όλους.

§1.4 Linux

Το **Linux** είναι ένα λειτουργικό σύστημα ανοιχτού κώδικα (open source), βασισμένο στο Unix. Στηρίζεται στον **Linux kernel (πυρήνα)**, που ξεκίνησε ο Linus Torvalds το 1991 και είναι γραμμένος κυρίως σε C. Είναι πολύ διαδεδομένο: τρέχει στο 90% των web servers, είναι η βάση του Android (άρα τρέχει στην πλειοψηφία των κινητών) και είναι η αποκλειστική επιλογή για υπερυπολογιστές (supercomputers). Όταν λέμε «*nix» εννοούμε όλα τα συστήματα της οικογένειας του Unix, όπως το Linux και το macOS.

§1.5 Γραφική διεπαφή και γραμμή εντολών

Υπάρχουν δύο τρόποι να αλληλεπιδράσουμε με ένα λειτουργικό σύστημα:

- Η **γραφική διεπαφή χρήστη (Graphical User Interface, GUI)**: παράθυρα, εικονίδια, ποντίκι.
- Η **διεπαφή γραμμής εντολών (Command Line Interface, CLI)**: πληκτρολογούμε εντολές ως κείμενο και διαβάζουμε την απάντηση ως κείμενο. Τη λέμε και τερματικό (terminal), κονσόλα (console) ή κέλυφος (shell). Οι όροι δεν είναι αυστηρά συνώνυμοι (οι διαφάνειες δίνουν ένα άρθρο για τις διαφορές τους), αλλά στην πράξη χρησιμοποιούνται εναλλάξ.

Στο μάθημα θα επιμείνουμε στη χρήση του CLI. Είναι πιο γρήγορο όταν το συνηθίσετε, μπορεί να αυτοματοποιηθεί, δουλεύει και σε απομακρυσμένα μηχανήματα χωρίς οθόνη, και είναι ο φυσικός χώρος όπου τρέχουν τα προγράμματα C που θα γράφουμε.

§1.6 Kernel και shell

Μια χρήσιμη αναλογία είναι ένα αμύγδαλο μέσα στο τσόφλι του. Στο κέντρο βρίσκεται ο **πυρήνας (kernel)**, το κομμάτι του λειτουργικού συστήματος που μιλάει με το υλικό. Γύρω του βρίσκεται το **κέλυφος (shell)**, το πρόγραμμα με το οποίο μιλάμε εμείς: μας δίνει τη δυνατότητα να τρέχουμε προγράμματα από τη γραμμή εντολών. Εμείς δεν αγγίζουμε ποτέ απευθείας τον πυρήνα· ζητάμε από το shell να τρέξει προγράμματα, και εκείνο ζητάει από το λειτουργικό σύστημα να τα εκτελέσει.

§1.7 Εκτέλεση προγραμμάτων από το shell

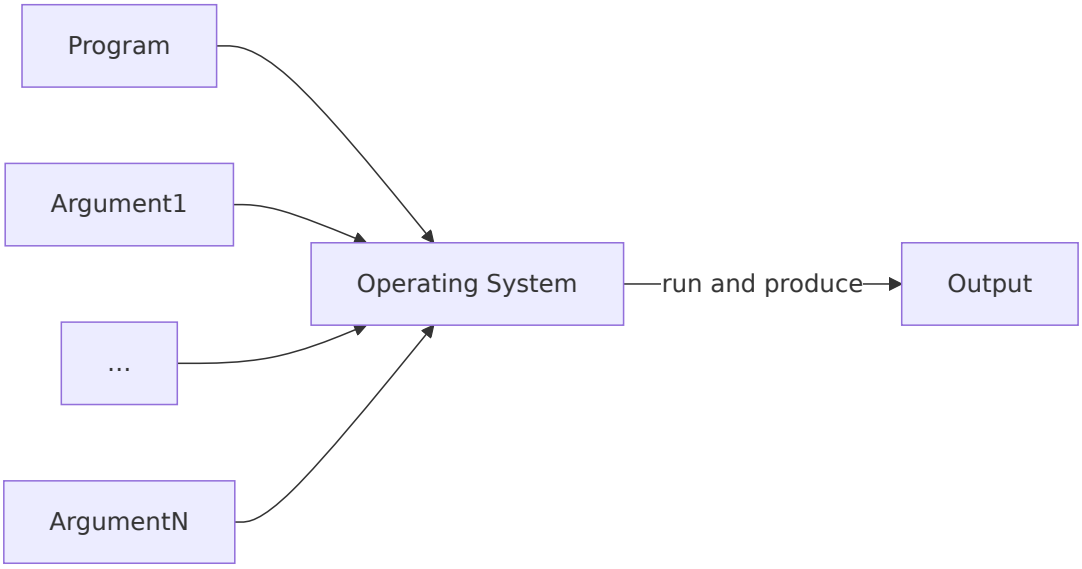
Για να τρέξουμε ένα πρόγραμμα με N ορίσματα γράφουμε:

```
Πρόγραμμα(Όρισμα0) Όρισμα1 Όρισμα2 … ΌρισμαN
```

δηλαδή το όνομα του προγράμματος και μετά τα ορίσματα, χωρισμένα με κενά. Το ίδιο το όνομα του προγράμματος μετράει ως όρισμα 0. Δύο ορισμοί:

- **Όρισμα (argument) ή δεδομένο εισόδου (input data)**: μια τιμή που μπορεί να χρησιμοποιήσει το πρόγραμμα κατά την εκτέλεσή του.
- **Δεδομένο εξόδου (output)**: η τιμή που επιστρέφει το πρόγραμμα όταν τελειώσει τον υπολογισμό του.

Για παράδειγμα, στην εντολή `/bin/echo hello good world` το πρόγραμμα είναι το `/bin/echo` και τα ορίσματα είναι τα `hello`, `good` και `world`. Το shell παραδίδει το πρόγραμμα και τα ορίσματά του στο λειτουργικό σύστημα, το οποίο το τρέχει και παράγει την έξοδο:



Σχήμα: το shell δίνει πρόγραμμα και ορίσματα στο λειτουργικό σύστημα.

Αργότερα θα δούμε πώς ένα πρόγραμμα C διαβάζει τα ορίσματα της γραμμής εντολών (`argc` / `argv`).

§1.8 Πρόσβαση στη γραμμή εντολών

Υπάρχουν τέσσερις τρόποι να αποκτήσετε γραμμή εντολών *nix, από τον πιο πρακτικό προς τον λιγότερο πρακτικό για προγραμματισμό:

1. Εγκαθιστάτε Linux (ή χρησιμοποιείτε macOS) στον υπολογιστή σας.
2. Εγκαθιστάτε **WSL 2** (Windows Subsystem for Linux) στο Windows μηχάνημά σας.
3. Συνδέεστε με `ssh` (από κονσόλα ή με το PuTTY) στα μηχανήματα του εργαστηρίου, `linuxXY.di.uoa.gr`.
4. Συνδέεστε στο Google Cloud console του προσωπικού σας λογαριασμού.

Στους τρόπους 1, 2 και 4 είστε **διαχειριστής (root)** του συστήματος: μπορείτε να εγκαταστήσετε ό,τι θέλετε. Στον τρόπο 3 είστε απλός **χρήστης (user)**.

Επειδή στη γραμμή εντολών όλα γίνονται με το πληκτρολόγιο, αξίζει να μάθετε το **τυφλό σύστημα (touch typing / blind typing)**, δηλαδή να γράφετε με όλα τα δάχτυλα χωρίς να κοιτάτε τα πλήκτρα. Οι διαφάνειες προτείνουν μερικές σελίδες εξάσκησης (βλ. «Διάβασμα»).

§1.9 Απομακρυσμένη σύνδεση με ssh

Η διάλεξη το έδειξε με μια ιστορία από τη ζωή ενός DevOps engineer: έρχεται ένα email-alert ότι σε έναν server η χρήση του επεξεργαστή ξεπέρασε το όριο. Ο server όμως βρίσκεται σε ένα data center στην άλλη άκρη του κόσμου, και δεν μπορούμε να πάμε εκεί να καθίσουμε μπροστά του. Ευτυχώς υπάρχει το «τηλεχειριστήριο»: το `ssh`.

Το **SSH (Secure Shell)** είναι ένα πρωτόκολλο δικτύου που επιτρέπει την ασφαλή απομακρυσμένη σύνδεση και διαχείριση ενός υπολογιστή ή server μέσω κρυπτογραφημένης επικοινωνίας. Με την εντολή

```
ssh user@host
```

ανοίγουμε ένα shell στο μηχάνημα `host` ως χρήστης `user`, και από εκεί και πέρα ό,τι πληκτρολογούμε εκτελείται στο απομακρυσμένο μηχάνημα. Έτσι θα συνδέεστε και στα μηχανήματα του εργαστηρίου.

§1.10 Βασικές εντολές

Κάθε γραμμή του shell ξεκινά με την **προτροπή (prompt)**, π.χ. `thanassis@linux14:~$`: ο χρήστης (`thanassis`), το μηχάνημα (`linux14`), ο τρέχων κατάλογος (`~`, ο αρχικός κατάλογος του χρήστη) και το `$`, μετά από το οποίο γράφουμε την εντολή. Μερικές εντολές για να εξερευνήσετε ένα σύστημα:

Εντολή	Τι κάνει
`who`	ποιοι χρήστες είναι συνδεδεμένοι, από ποιο τερματικό και πότε
`last`	οι πρόσφατες συνδέσεις χρηστών
`whoami`	το όνομα του τρέχοντος χρήστη
`id`	ο αριθμός χρήστη (uid) και οι ομάδες (groups) του
`users`	τα ονόματα των συνδεδεμένων χρηστών
`factor N`	αναλύει τον αριθμό `N` σε πρώτους παράγοντες
`hostname`	το όνομα του μηχανήματος
`sleep N`	περιμένει `N` δευτερόλεπτα και τελειώνει
`uptime`	ώρα, πόσο καιρό τρέχει το σύστημα, χρήστες, φορτίο
`uname -a`	πληροφορίες για τον πυρήνα και την αρχιτεκτονική
`top`	διαδραστική λίστα των διεργασιών που τρέχουν
`man`	τα εγχειρίδια (manual pages) των εντολών, π.χ. `man ls`
`ps`	οι διεργασίες (processes) που τρέχουν

Οι περισσότερες τυπώνουν κάτι και τελειώνουν. Οι `top` και `man` είναι **διαδραστικές (interactive)**: μένουν ανοιχτές μέχρι να βγείτε (με `q`). Η `uname -a` δείχνει ένα παράδειγμα **επιλογής (option)**: ορίσματα που αρχίζουν με `-` και αλλάζουν τη συμπεριφορά της εντολής.

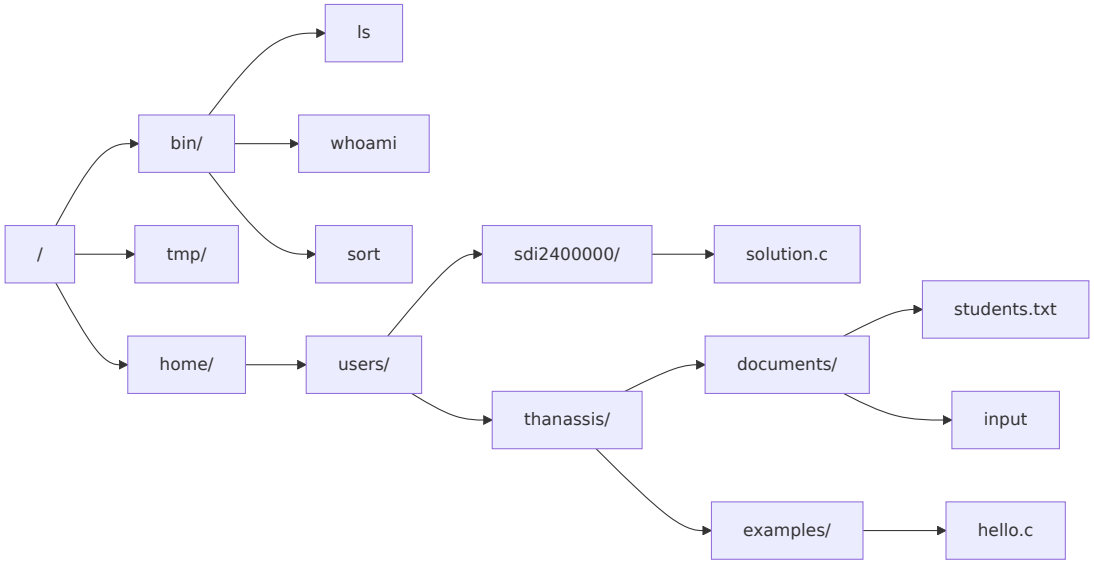
§1.11 Το σύστημα αρχείων

Αρχείο (file) είναι ένας πόρος για να καταγράφουμε δεδομένα σε έναν υπολογιστή. Συνήθως αποθηκεύεται στη δευτερεύουσα, μόνιμη μνήμη (π.χ. σκληρό δίσκο). Στο Linux σχεδόν τα πάντα είναι αρχείο (”everything is a file”): ακόμα και οι συσκευές εμφανίζονται ως αρχεία. Κάθε αρχείο:

- έχει ένα **όνομα (filename / basename)**·
- βρίσκεται μέσα σε έναν συγκεκριμένο **κατάλογο ή φάκελο (directory / folder)**·
- έχει ένα **μονοπάτι (filepath)** που καθορίζει πού βρίσκεται στο σύστημα αρχείων.

Για παράδειγμα, στο μονοπάτι `/home/users/thanassis/documents/students.txt` ο φάκελος είναι ο `/home/users/thanassis/documents` και το όνομα του αρχείου το `students.txt`. Το `.txt` στο τέλος λέγεται **επέκταση (extension)** και συνήθως περιγράφει τον τύπο του αρχείου (κείμενο, `.c` για πηγαίο κώδικα C κ.ο.κ.).

Οι κατάλογοι περιέχουν αρχεία και άλλους καταλόγους, οπότε όλο το **σύστημα αρχείων (filesystem)** σχηματίζει μια ιεραρχία, ένα δέντρο που ξεκινά από τη **ρίζα (root)** `/`:



Σχήμα: ιεραρχία συστήματος αρχείων. Οι εντολές (`ls`, `whoami`, `sort`) είναι κι αυτές αρχεία, στον κατάλογο `/bin`.

Ένα μονοπάτι που ξεκινά από `/` είναι **απόλυτο (absolute)**. Ένα μονοπάτι που δεν ξεκινά από `/` είναι **σχετικό (relative)** ως προς τον **τρέχοντα κατάλογο (current directory)**, αυτόν όπου «βρισκόμαστε». Δύο ειδικά ονόματα:

- `.` είναι ο τρέχων κατάλογος·
- `..` είναι ο **γονικός (parent)** κατάλογος, ένα επίπεδο πιο πάνω.

Το `~` είναι συντομογραφία του αρχικού καταλόγου (home directory) του χρήστη· για τον `thanassis`, `~/examples` σημαίνει `/home/users/thanassis/examples`. Προσέξτε ότι τα ονόματα

αρχείων στο Unix ξεχωρίζουν πεζά από κεφαλαία (case-sensitive): `Hello.c` και `hello.c` είναι διαφορετικά αρχεία.

§1.12 Εντολές χειρισμού αρχείων

Εντολή	Τι κάνει
`pwd`	τυπώνει τον τρέχοντα κατάλογο (print working directory)
`ls`	εμφανίζει τα αρχεία ενός καταλόγου (list)
`ls -l`	το ίδιο, με λεπτομέρειες
`cat αρχείο`	τυπώνει τα περιεχόμενα ενός αρχείου
`mkdir κατάλογος`	δημιουργεί νέο, άδειο κατάλογο (make directory)
`cd κατάλογος`	αλλάζει τον τρέχοντα κατάλογο (change directory)
`cp πηγή προορισμός`	αντιγράφει ένα αρχείο (copy)
`rm αρχείο`	διαγράφει ένα αρχείο (remove)
`rmdir κατάλογος`	διαγράφει έναν **άδειο** κατάλογο

Η `ls -l` τυπώνει για κάθε αρχείο μια γραμμή όπως `-rw-r--r-- 1 thanassis dep 62 Oct 5 16:04 hello.c`: τον τύπο και τα δικαιώματα (`-` για απλό αρχείο, `rw-r--r--`), τον ιδιοκτήτη (`thanassis`) και την ομάδα (`dep`), το μέγεθος σε bytes (`62`), την ημερομηνία τελευταίας αλλαγής και το όνομα. Τα δικαιώματα θα τα δείτε αναλυτικά στο Εργαστήριο 1.

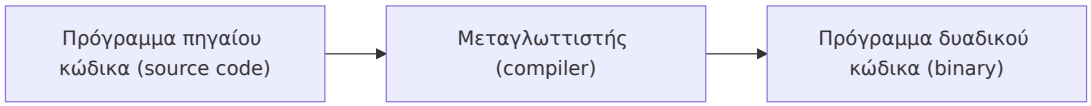
§1.13 Coreutils και άλλα εργαλεία

Οι περισσότερες διανομές Linux (distros) έρχονται με μια σουίτα βασικών βοηθητικών προγραμμάτων που λέγονται **coreutils**. Είναι ως επί το πλείστον γραμμένα σε C. Αξίζει να εξερευνήσετε μόνοι σας και άλλα, μέσα ή έξω από τα coreutils: `find`, `grep`, `sort`, `df -h`, `du -sh`, `wc -l`, `file`, `which`, `basename`, `dirname`, `ping`, `curl` κ.ο.κ. Για κάθε άγνωστη εντολή, το `man` είναι ο πρώτος σας φίλος.

Στη ζωντανή επίδειξη αναφέρθηκαν επίσης το `apt`, με το οποίο προσθέτουμε (εγκαθιστούμε) καινούργια προγράμματα σε Ubuntu / WSL, και οι **επεξεργαστές κειμένου γραμμής εντολών (command line editors)**, με τους οποίους αλλάζουμε αρχεία από την κονσόλα. Περισσότερα στα εργαστήρια.

§1.14 Μεταγλωττιστές

Μεταγλωττιστής (compiler) είναι ένα πρόγραμμα που μετατρέπει τις εντολές μιας γλώσσας προγραμματισμού σε **κώδικα μηχανής (machine code)**, ώστε να μπορεί να διαβαστεί και να τρέξει από τον υπολογιστή.



Σχήμα: από τον πηγαίο κώδικα στο εκτελέσιμο πρόγραμμα (απλοποιημένο).

Το σχήμα είναι προσεγγιστικό: λείπουν λεπτομέρειες (όπως η σύνδεση, linking) που θα προστεθούν σε επόμενες διαλέξεις. Στο μάθημα χρησιμοποιούμε τον **GNU C Compiler**, `gcc`. Η βασική χρήση του:

- `gcc hello.c` μεταγλωττίζει το `hello.c` και γράφει το εκτελέσιμο στο αρχείο `a.out`·
- `gcc -o hello hello.c` γράφει το εκτελέσιμο στο αρχείο που δίνουμε μετά το `-o`, εδώ `hello`.

Το εκτελέσιμο το τρέχουμε γράφοντας το μονοπάτι του, π.χ. `./hello`: το `./` λέει στο shell ότι το πρόγραμμα βρίσκεται στον τρέχοντα κατάλογο. Οι εντολές όπως η `ls` δεν το χρειάζονται, γιατί το shell τις βρίσκει μόνο του σε καταλόγους συστήματος όπως ο `/bin`. Η εντολή `file` δείχνει ότι το `a.out` δεν είναι κείμενο αλλά **ELF 64-bit** εκτελέσιμο για επεξεργαστή x86-64, δηλαδή κώδικας μηχανής.

§1.15 Ανατομία του Hello World

```c
/* File: helloworld.c */
#include <stdio.h>

int main() {
  printf("Hello world\n");
  return 0;
}
```

Κάθε κομμάτι έχει ρόλο:

- **Σχόλια (comments).** Το `/* File: helloworld.c */` είναι κείμενο του προγραμματιστή που συνοδεύει τον κώδικα για να τον κάνει πιο σαφή σε τρίτους ή σε εμάς τους ίδιους, ειδικά αν έχει περάσει καιρός από τότε που τον γράψαμε. Ο μεταγλωττιστής το αγνοεί. Ένα σχόλιο γράφεται ανάμεσα σε `/*` και `*/` (μπορεί να πιάνει πολλές γραμμές) ή σε μία γραμμή μετά από `//`.
- **Η οδηγία `#include` (directive).** Με το `#include <stdio.h>` ο μεταγλωττιστής συμπεριλαμβάνει τα περιεχόμενα του αρχείου `stdio.h` (standard input output) στον κώδικά μας. Το `stdio.h` περιέχει τις βασικές (standard) δηλώσεις των συναρτήσεων για εμφάνιση δεδομένων στην οθόνη (output) και εισαγωγή δεδομένων από το πληκτρολόγιο (input).
- **Η `printf`.** Είναι συνάρτηση βιβλιοθήκης, δηλωμένη στο `stdio.h` (γι' αυτό κάνουμε `#include`), και τυπώνει το αλφαριθμητικό `"Hello world\n"`. Ο χαρακτήρας `\n` δημιουργεί μια νέα γραμμή μετά το μήνυμα.

- **Η `main`.** Ένα πρόγραμμα C ορίζεται από ένα σύνολο **συναρτήσεων (functions)**, που θα τις δούμε στο Κεφάλαιο 3. Για να μπορεί να τρέξει, πρέπει να έχει **ακριβώς μία** συνάρτηση `main`: αυτή καλείται πρώτη όταν αρχίζει η εκτέλεση.
- **Το `return 0`.** Επιστρέφει την τιμή της συνάρτησης. Η τιμή που επιστρέφει η `main` είναι ο **κωδικός εξόδου (exit code)** του προγράμματος, που δείχνει αν ολοκληρώθηκε με επιτυχία: `0` σημαίνει επιτυχία, κάθε τιμή διάφορη του `0` σημαίνει ότι το πρόγραμμα **απέτυχε**. Αμέσως μετά την εκτέλεση, η εντολή `echo $?` στο shell τυπώνει τον κωδικό εξόδου.
- **Εντολές (statements).** Οι εντολές μιας συνάρτησης βρίσκονται μέσα σε άγκιστρα `{ }` και η καθεμία τελειώνει με `;` (semicolon, το σύμβολο του ελληνικού ερωτηματικού).

Προσέξτε τη διαφορά ανάμεσα σε ό,τι τυπώνει ένα πρόγραμμα (το `Hello world` της `printf`) και στον κωδικό εξόδου του (το `0` του `return`): το πρώτο το βλέπουμε στην οθόνη, το δεύτερο το διαβάζει το shell.

Παραδείγματα

§1.16 Εξερεύνηση ενός συστήματος

Εφαρμόζει τις «Βασικές εντολές». Συνδεδεμένοι στο `linux14`, τρέχουμε μία μία τις εντολές του πίνακα (το … σημαίνει ότι η έξοδος της `last` παραλείπεται):

```
thanassis@linux14:~$ who
thanassis pts/0        Oct  5 14:36 (110.220.111.247)
thanassis@linux14:~$ last
…
thanassis@linux14:~$ whoami
thanassis
thanassis@linux14:~$ id
uid=2622(thanassis) gid=1001(dep) groups=1001(dep)
thanassis@linux14:~$ users
thanassis
thanassis@linux14:~$ factor 2394872891
2394872891: 3259 734849
thanassis@linux14:~$ hostname
linux14
thanassis@linux14:~$ sleep 3
thanassis@linux14:~$ uptime
 14:40:23 up 7 days,  5:49,  3 users,  load average: 0.00, 0.00, 0.00
thanassis@linux14:~$ uname -a
linux linux14 5.4.0-163-generic #180-Ubuntu SMP Tue Sep 5 13:21:23 UTC 2023 x86_64
 ↪  x86_64 x86_64 GNU/Linux
# Interactive commands follow
```

```
thanassis@linux14:~$ top
thanassis@linux14:~$ man
thanassis@linux14:~$ ps
```

Παρατηρήστε ότι η `sleep 3` δεν τυπώνει τίποτα: απλώς η επόμενη προτροπή εμφανίζεται μετά από τρία δευτερόλεπτα. Η `factor` βρήκε ότι $2394872891 = 3259 \cdot 734849$.

§1.17 Το πρόγραμμα /bin/echo

Εφαρμόζει την «Εκτέλεση προγραμμάτων από το shell». Το `/bin/echo` είναι ένα πρόγραμμα που τυπώνει τα ορίσματά του χωρισμένα με ένα κενό:

```
$ /bin/echo hello good world
hello good world
```

Εδώ το όρισμα 0 είναι το ίδιο το πρόγραμμα, `/bin/echo`, και ακολουθούν τρία ορίσματα: `hello`, `good`, `world`. Επειδή το `/bin/echo` είναι απόλυτο μονοπάτι, το shell ξέρει ακριβώς ποιο αρχείο να τρέξει.

§1.18 Χειρισμός αρχείων βήμα βήμα

Εφαρμόζει το «Σύστημα αρχείων» και τις «Εντολές χειρισμού αρχείων». Ξεκινάμε στον κατάλογο `~/examples`, που περιέχει το `hello.c`:

```
# Directory we are currently in
thanassis@linux14:~/examples$ pwd
/home/users/thanassis/examples
# List files
thanassis@linux14:~/examples$ ls
hello.c
# List files with details
thanassis@linux14:~/examples$ ls -l
total 4
-rw-r--r-- 1 thanassis dep 62 Oct  5 16:04 hello.c
# Print contents of file
thanassis@linux14:~/examples$ cat hello.c
#include <stdio.h>

int main() {
  printf("Hello world\n");
}
# Create `foo` directory
thanassis@linux14:~/examples$ mkdir foo
thanassis@linux14:~/examples$ ls
foo  hello.c
```

```
# New directory is empty by default
thanassis@linux14:~/examples$ ls foo
```

Συνεχίζουμε μπαίνοντας στον νέο κατάλογο:

```
# Change directory to the newly created one
thanassis@linux14:~/examples$ cd foo
# List files within the directory
thanassis@linux14:~/examples/foo$ ls
# Copy file from parent `..` directory to the current one `.`
thanassis@linux14:~/examples/foo$ cp ../hello.c .
thanassis@linux14:~/examples/foo$ ls
hello.c
# Check the file contents are what we expect
thanassis@linux14:~/examples/foo$ cat hello.c
#include <stdio.h>

int main() {
  printf("Hello world\n");
}
# Remove the file
thanassis@linux14:~/examples/foo$ rm hello.c
# Go to the parent directory
thanassis@linux14:~/examples/foo$ cd ..
thanassis@linux14:~/examples$ rmdir foo
thanassis@linux14:~/examples$ ls
hello.c
```

Παρατηρήστε ότι η προτροπή αλλάζει σε `~/examples/foo$` μετά το `cd foo`, και ότι στην `cp ../hello.c .` χρησιμοποιούνται και τα δύο ειδικά ονόματα: «από τον γονικό κατάλογο, στον τρέχοντα». Ο `foo` διαγράφεται με `rmdir` μόνο αφού τον αδειάσουμε.

§1.19 Μεταγλώττιση και εκτέλεση του Hello World

Εφαρμόζει τους «Μεταγλωττιστές». Στον κατάλογο `~/examples`:

```
thanassis@linux14:~/examples$ gcc hello.c
thanassis@linux14:~/examples$ ls
a.out  hello.c
thanassis@linux14:~/examples$ ./a.out
Hello world
thanassis@linux14:~/examples$ file a.out
```

```
a.out: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked,
 ↪ interpreter /lib64/ld-linux-x86-64.so.2,
 ↪ BuildID[sha1]=52fa5999c10d767a5ff30f662346478333de74bf, for GNU/Linux 3.2.0,
 ↪ not stripped
thanassis@linux14:~/examples$ gcc -o hello hello.c
thanassis@linux14:~/examples$ ./hello
Hello world
```

Χωρίς `-o`, ο `gcc` ονομάζει το εκτελέσιμο `a.out`· με `-o hello` το ονομάζει `hello`. Αν μεταγλωττίσετε και δεύτερο πρόγραμμα χωρίς `-o`, το νέο `a.out` θα αντικαταστήσει το παλιό, γι' αυτό συνηθίζεται να δίνετε πάντα όνομα.

§1.20 Ο κωδικός εξόδου

Εφαρμόζει την «Ανατομία του Hello World». Αμέσως μετά την εκτέλεση του `./hello`, το `echo $?` τυπώνει την τιμή που επέστρεψε η `main`:

```
$ ./hello
Hello world
$ echo $?
0
```

Αν αλλάξετε το `return 0;` σε `return 1;` και ξαναμεταγλωττίσετε, το `echo $?` θα τυπώσει `1`: για το shell, το πρόγραμμα απέτυχε.

§1.21 Για την επόμενη φορά

Οι διαφάνειες ζητούν να ολοκληρώσετε τις εγγραφές στα εργαλεία του μαθήματος, να γραφτείτε σε εργαστήριο, και να εγκαταστήσετε WSL για να τρέξετε μόνοι σας όλα τα παραπάνω παραδείγματα.

Κύρια σημεία

1. Ένα υπολογιστικό σύστημα έχει επίπεδα: οι χρήστες χρησιμοποιούν εφαρμογές, οι εφαρμογές χρειάζονται το λειτουργικό σύστημα και αυτό χρειάζεται το υλικό.
2. Το λειτουργικό σύστημα διαχειρίζεται το υλικό και τους πόρους (μνήμη, επεξεργαστή) και προσφέρει υπηρεσίες (δίσκο, δίκτυο) στα προγράμματα.
3. Το Linux είναι λειτουργικό σύστημα ανοιχτού κώδικα βασισμένο στο Unix, με πυρήνα γραμμένο κυρίως σε C.
4. Στο μάθημα δουλεύουμε με τη γραμμή εντολών (CLI) και όχι με τη γραφική διεπαφή (GUI).
5. Ο kernel μιλάει με το υλικό· το shell είναι το πρόγραμμα που μας επιτρέπει να τρέχουμε προγράμματα από τη γραμμή εντολών.
6. Ένα πρόγραμμα τρέχει ως `Πρόγραμμα Όρισμα1 … ΌρισμαN`, όπου το όνομα του προγράμματος είναι το όρισμα 0.

7. Γραμμή εντολών *nix αποκτάτε με Linux / macOS, WSL 2, `ssh` στα μηχανήματα του εργαστηρίου ή Google Cloud console· μόνο στο εργαστήριο δεν είστε root.
8. Με `ssh user@host` συνδέεστε με ασφάλεια (κρυπτογραφημένα) σε απομακρυσμένο μηχάνημα και δουλεύετε σε αυτό σαν να ήσασταν μπροστά του.
9. Κάθε αρχείο έχει όνομα, βρίσκεται σε κατάλογο και προσδιορίζεται από ένα μονοπάτι στην ιεραρχία που ξεκινά από τη ρίζα `/`.
10. Το `.` είναι ο τρέχων κατάλογος, το `..` ο γονικός και το ~ ο αρχικός κατάλογος του χρήστη.
11. Οι `pwd`, `ls`, `cd`, `cat`, `mkdir`, `cp`, `rm`, `rmdir` αρκούν για τη βασική διαχείριση αρχείων· η `rm` διαγράφει οριστικά.
12. Ο μεταγλωττιστής μετατρέπει τον πηγαίο κώδικα σε κώδικα μηχανής· στο μάθημα χρησιμοποιούμε τον `gcc`.
13. Το `gcc hello.c` παράγει το `a.out`, το `gcc -o hello hello.c` το `hello`, και τα τρέχουμε με `./a.out` ή `./hello`.
14. Κάθε πρόγραμμα C έχει ακριβώς μία `main`, που εκτελείται πρώτη.
15. Η τιμή που επιστρέφει η `main` είναι ο κωδικός εξόδου: `0` για επιτυχία, οτιδήποτε άλλο για αποτυχία· τον βλέπουμε με `echo $?`.
16. Οι εντολές μιας συνάρτησης βρίσκονται μέσα σε `{ }` και καθεμία τελειώνει με `;`.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
λειτουργικό σύστημα	operating system (OS)	Λογισμικό που διαχειρίζεται υλικό και πόρους και εξυπηρετεί τα προγράμματα
υλικό	hardware	Οι φυσικές συσκευές του υπολογιστή
πυρήνας	kernel	Το κεντρικό μέρος του OS, που μιλάει με το υλικό
κέλυφος	shell	Πρόγραμμα που τρέχει τις εντολές που πληκτρολογούμε
γραφική διεπαφή χρήστη	GUI	Αλληλεπίδραση με παράθυρα και ποντίκι
διεπαφή γραμμής εντολών	CLI, terminal, console	Αλληλεπίδραση με εντολές κειμένου
όρισμα	argument	Τιμή που δίνουμε σε ένα πρόγραμμα όταν το τρέχουμε
δεδομένο εξόδου	output	Ό,τι παράγει ένα πρόγραμμα όταν τελειώσει
προτροπή	prompt	Το `user@host:dir$` πριν από κάθε εντολή
διαχειριστής	root	Χρήστης με πλήρη δικαιώματα στο σύστημα

Ελληνικά	English	Σύντομος ορισμός
αρχείο	file	Πόρος για αποθήκευση δεδομένων, συνήθως στον δίσκο
κατάλογος, φάκελος	directory, folder	Περιέχει αρχεία και άλλους καταλόγους
μονοπάτι	path	Η θέση ενός αρχείου στην ιεραρχία, π.χ. `/home/users`
επέκταση	extension	Το τέλος του ονόματος (`.txt`, `.c`) που δείχνει τον τύπο
ρίζα	root directory	Ο κατάλογος `/`, κορυφή της ιεραρχίας
τρέχων / γονικός κατάλογος	current / parent directory	`.` και `..`
μεταγλωττιστής	compiler	Μετατρέπει πηγαίο κώδικα σε κώδικα μηχανής
πηγαίος κώδικας	source code	Το πρόγραμμα όπως το γράφουμε, π.χ. `hello.c`
εκτελέσιμο, δυαδικό	executable, binary	Το πρόγραμμα σε κώδικα μηχανής, π.χ. `a.out`
σχόλιο	comment	Κείμενο για τον αναγνώστη, που ο compiler αγνοεί
οδηγία	directive	Γραμμή που αρχίζει με `#`, π.χ. `#include`
κωδικός εξόδου	exit code	Η τιμή που επιστρέφει η `main`· `0` = επιτυχία

Διάβασμα

- **Διαφάνειες**: Διάλεξη 1, σελ. 1–43. Αρχιτεκτονική και λειτουργικά συστήματα: σελ. 6–10· GUI, CLI, kernel και shell: σελ. 11–13· εκτέλεση προγραμμάτων και ορίσματα: σελ. 14–17· πρόσβαση σε γραμμή εντολών και `ssh`: σελ. 18–24· βασικές εντολές: σελ. 25· σύστημα αρχείων και εντολές αρχείων: σελ. 27–31· `gcc` και ανάλυση του Hello World: σελ. 32–40.
- **Σημειώσεις**: οι διαφάνειες ζητούν τις σελίδες 58–60 και 98–99 των σημειώσεων του κ. Σταματόπουλου (αριθμημένες σελίδες, δηλαδή Κ04, σελ. 58–60 και 98–99):
  - Κεφάλαιο 4: Συναρτήσεις, ενότητα «Δομή ενός προγράμματος C – Συναρτήσεις» (Κ04, σελ. 58–60): η `main` και η τιμή που επιστρέφει.
  - Κεφάλαιο 6: Μνήμη και συμβολοσειρές, ενότητα «Ορίσματα γραμμής εντολών» (Κ04, σελ. 98–99): πώς ένα πρόγραμμα C διαβάζει τα ορίσματά του (θα το δούμε αναλυτικά αργότερα).
  - Για το υπόβαθρο: Κεφάλαιο 0: Εισαγωγή, ενότητες «Λογισμικό και γλώσσες προγραμματισμού», «Πώς κατασκευάζουμε εκτελέσιμο πρόγραμμα;», «Η διαδικασία της μεταγλώττισης και σύνδεσης», «Προγραμματιστικά περιβάλλοντα για την C», «Παραδείγματα χρήσης του gcc» (Κ04, σελ. 11, 14–17), και Κεφάλαιο 1,

ενότητα «Καλημέρα κόσμε της C» (K04, σελ. 19).

- **Εργαστήριο**: Εργαστήριο 0: Βήματα 4–5 (`ssh`, `ls`, `cd`) και 7 (`hello.c`)· Εργαστήριο 1: Βήματα 1–3 (πλοήγηση, μεταγλώττιση, αντιγραφή και προβολή αρχείων) και 5 (`man`, `grep`, ανακατεύθυνση).
- **Άλλα**:
  - Wikipedia: Operating system, Unix, Computer file, Everything is a file, GNU Core Utilities, GNU Compiler Collection
  - Ο κώδικας του Linux kernel και των coreutils στο GitHub
  - Διαφορές terminal, console, shell και command line
  - Γιατί τα πανεπιστήμια διδάσκουν ακόμα C και C++ (Quora)
  - Τυφλό σύστημα: blindtyping.com, typingclub.com, typingstudy.com
  - Προαιρετικά: The Unix Programming Environment (PDF), Common Linux Interview Questions

Συχνά λάθη

- **Τρέξιμο χωρίς `./`**. Το `a.out` σκέτο δίνει `a.out: command not found`, γιατί το shell δεν ψάχνει στον τρέχοντα κατάλογο. Διόρθωση: `./a.out`.
- **Τρέξιμο χωρίς (ξανά)μεταγλώττιση**. Αλλάξατε το `hello.c` αλλά το `./hello` τυπώνει το παλιό μήνυμα, ή `./hello: No such file or directory`. Το εκτελέσιμο φτιάχνεται μόνο από τον `gcc`: ξανατρέξτε `gcc -o hello hello.c` μετά από κάθε αλλαγή.
- **Λάθος κατάλογος**. `cat hello.c` δίνει `No such file or directory` επειδή δεν βρίσκεστε εκεί που νομίζετε. Ελέγξτε με `pwd` και `ls`, και μετακινηθείτε με `cd`.
- **Κεφαλαία και πεζά**. `cd Examples` αποτυγχάνει όταν ο κατάλογος λέγεται `examples`: τα ονόματα στο Unix είναι case-sensitive.
- `rmdir` **σε κατάλογο με αρχεία**. Δίνει `Directory not empty`. Αδειάστε πρώτα τον κατάλογο με `rm`.
- `rm` **χωρίς προσοχή**. Δεν υπάρχει κάδος ανακύκλωσης· ένα `rm hello.c` σβήνει τον κώδικά σας οριστικά. Κρατάτε αντίγραφα (αργότερα: git).
- **Κενά μέσα σε ορίσματα**. `mkdir my work` φτιάχνει *δύο* καταλόγους, `my` και `work`, γιατί τα κενά χωρίζουν τα ορίσματα.
- **Ξεχασμένο** `;`. Αν λείπει το `;` μετά την `printf(...)`, ο `gcc` αναφέρει `error: expected ';' before 'return'` και δεν παράγει εκτελέσιμο.
- `echo $?` **αργότερα**. Αν τρέξετε άλλη εντολή ανάμεσα, το `echo $?` δείχνει τον κωδικό εξόδου εκείνης (π.χ. `0` της `ls`), όχι του προγράμματός σας.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K1.10** Το μονοπάτι ενός αρχείου (20% σωστές): Το 66% διάλεξε `/home/users/ethan/example/`,

δηλαδή τον κατάλογο (dirname) που περιέχει το αρχείο· το filepath όμως περιλαμβάνει και το όνομα του αρχείου.

- **K1.9** Εκτύπωση περιεχομένων αρχείου (29% σωστές): Το 47% διάλεξε `printf`, μπερδεύοντας τη συνάρτηση της C (και την ομώνυμη εντολή του shell), που τυπώνει το κείμενο που της δίνουμε, με μια εντολή που διαβάζει αρχεία.

Ερωτήσεις κατανόησης

- **E1.1** Ποια τρία πράγματα αναλαμβάνει ένα λειτουργικό σύστημα;[11]
- **E1.2** Ποια είναι η διαφορά ανάμεσα σε kernel και shell;[12]
- **E1.3** Στην εντολή `factor 12 35`, ποιο είναι το πρόγραμμα και πόσα ορίσματα έχει;[13]
- **E1.4** Βρίσκεστε στο `/home/users/thanassis/examples/foo`. Σε ποιον κατάλογο σας πάει το `cd ..`; Ποιο είναι το απόλυτο μονοπάτι του `../hello.c`;[14]
- **E1.5** Τι κάνει η `gcc -o prog prog.c` και πώς τρέχετε μετά το πρόγραμμα;[15]
- **E1.6** Γιατί η `rmdir foo` μπορεί να αποτύχει;[16]
- **E1.7** Τι τυπώνει το `echo $?` αμέσως μετά από ένα πρόγραμμα που τελείωσε με `return 3;`, και τι σημαίνει αυτό;[17]
- **E1.8** Πόσες συναρτήσεις `main` πρέπει να έχει ένα πρόγραμμα C;[18]

Kahoot από το αμφιθέατρο (K1.1–K1.10)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K1.1** Η καταγωγή του Linux: 87% σωστές απαντήσεις
- **K1.2** Το 2ο όρισμα του echo: 73% σωστές απαντήσεις
- **K1.3** Η εντολή man: 72% σωστές απαντήσεις
- **K1.4** Στο Linux όλα είναι...: 71% σωστές απαντήσεις
- **K1.5** Δικαιώματα υπερχρήστη: 68% σωστές απαντήσεις
- **K1.6** Το ΛΣ των web servers: 65% σωστές απαντήσεις
- **K1.7** Το basename ενός αρχείου: 53% σωστές απαντήσεις
- **K1.8** Φορτίο απομακρυσμένου server: 53% σωστές απαντήσεις
- **K1.9** Εκτύπωση περιεχομένων αρχείου: 29% σωστές απαντήσεις

[11] Διαχειρίζεται το υλικό (συσκευές), διαχειρίζεται τους πόρους (μνήμη, χρόνο επεξεργαστή) και προσφέρει υπηρεσίες στα προγράμματα (δίσκο, δίκτυο).

[12] Ο kernel είναι ο πυρήνας του λειτουργικού συστήματος και μιλάει με το υλικό. Το shell είναι το πρόγραμμα με το οποίο μιλάμε εμείς: διαβάζει τις εντολές μας και ζητά από το OS να τρέξει προγράμματα.

[13] Το πρόγραμμα είναι το `factor` (όρισμα 0) και έχει δύο ορίσματα, `12` και `35`.

[14] Στον γονικό, `/home/users/thanassis/examples`. Το `../hello.c` είναι το `/home/users/thanassis/examples/hello.c`.

[15] Μεταγλωττίζει το `prog.c` και γράφει το εκτελέσιμο στο αρχείο `prog` (αντί για `a.out`). Το τρέχετε με `./prog`.

[16] Η `rmdir` διαγράφει μόνο άδειους καταλόγους· αν ο `foo` περιέχει αρχεία, αποτυγχάνει (`Directory not empty`).

[17] Τυπώνει `3`, τον κωδικό εξόδου του προγράμματος. Επειδή είναι διάφορος του `0`, για το shell το πρόγραμμα απέτυχε.

[18] Ακριβώς μία· από αυτήν ξεκινά η εκτέλεση.

- **K1.10** Το μονοπάτι ενός αρχείου: 20% σωστές απαντήσεις

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A1.1–A1.5)

- **A1.1** Παραδείγματα για κάθε επίπεδο ενός υπολογιστικού συστήματος: Διάλεξη 1: Η Γραμμή Εντολών, διαφάνεια 7 · ★☆☆ · short-answer · `slides-lec01-architecture-examples`
- **A1.2** Πρόγραμμα και ορίσματα στο /bin/echo: Διάλεξη 1: Η Γραμμή Εντολών, διαφάνεια 15 · ★☆☆ · tooling · `slides-lec01-echo-arguments`
- **A1.3** Τρέξτε μόνοι σας τις εντολές της διάλεξης: Διάλεξη 1: Η Γραμμή Εντολών, διαφάνεια 42 · ★☆☆ · tooling · `slides-lec01-try-commands`
- **A1.4** Γιατί χρειαζόμαστε λειτουργικό σύστημα: Διάλεξη 1: Η Γραμμή Εντολών, διαφάνεια 9 · ★☆☆ · short-answer · `slides-lec01-why-os`
- **A1.5** Εξερευνήστε κι άλλα βασικά προγράμματα: Διάλεξη 1: Η Γραμμή Εντολών, διαφάνεια 31 · ★★☆ · tooling · `slides-lec01-explore-coreutils`

Εργαστήριο (A1.6–A1.10)

- **A1.6** Πλοήγηση στο σύστημα αρχείων: Εργαστήριο 1, Βήμα 1 · ★☆☆ · tooling · `lab-lab01-step1-navigation`
- **A1.7** Μεταγλώττιση, σύνδεση και εκτέλεση: Εργαστήριο 1, Βήμα 2 · ★☆☆ · tooling · `lab-lab01-step2-compile-link`
- **A1.8** Αντιγραφή, μετονομασία και προβολή αρχείων: Εργαστήριο 1, Βήμα 3 · ★☆☆ · tooling · `lab-lab01-step3-copy-view`
- **A1.9** Δικαιώματα προστασίας: Εργαστήριο 1, Βήμα 4 · ★☆☆ · tooling · `lab-lab01-step4-permissions`
- **A1.10** Αναζήτηση, ανακατεύθυνση και σωληνώσεις: Εργαστήριο 1, Βήμα 5 · ★★☆ · tooling · `lab-lab01-step5-grep-pipes`

Εργασίες (A1.11–A1.12)

- **A1.11** Ξεκινώντας με την γραμμή εντολών (bandit): Εργασία 0 (2023-24), Άσκηση 2 · ★★☆ · tooling · `hw-2023-hw0-bandit`
- **A1.12** Παιχνίδια με Κονσόλα (cmdline): Εργασία 0 (2025-26), Άσκηση 2 · ★★☆ · tooling · `hw-2025-hw0-cmdline`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A4.12** Νέο URL στο GitHub (pages): Εργασία 0 (2025-26), Άσκηση 1 · ★☆☆ · tooling · `hw-2025-hw0-pages`

- **A4.11** Το πρώτο σας repository: Εργαστήριο 1, Άσκηση 1 · ★☆☆ · tooling · `lab-lab01-info`
- **A9.14** Κωδικοποίηση και αποκωδικοποίηση κειμένου: Εργαστήριο 4, Άσκηση 4 · ★★☆ · programming · `lab-lab04-encode-decode`
- **A18.4** Ανακατεύθυνση της stderr: Διάλεξη 18, διαφάνεια 56 · ★☆☆ · tooling · `slides-lec18-stderr-redirect`

Μέρος Β: Τα θεμέλια της C

Κεφάλαιο 2: Μνήμη και Μεταβλητές

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να

- εξηγείτε τι είναι bit, byte και διεύθυνση μνήμης, και πώς διαφέρει η κύρια από τη δευτερεύουσα μνήμη·
- μετατρέπετε αριθμούς ανάμεσα στο δυαδικό, το δεκαδικό και το δεκαεξαδικό σύστημα·
- δηλώνετε μεταβλητές, να τους αναθέτετε τιμές και να εξηγείτε τι σημαίνει αυτό για τη μνήμη·
- επιλέγετε τύπο (`char`, `int`, `unsigned`, `float`, `double`, …) με βάση το εύρος τιμών του·
- αναπαριστάτε αρνητικούς ακεραίους σε συμπλήρωμα ως προς 2 και να αναγνωρίζετε την υπερχείλιση ακεραίων·
- τυπώνετε μεταβλητές με την `printf`, χρησιμοποιώντας ακολουθίες διαφυγής και προσδιοριστικά μορφοποίησης.

Προαπαιτούμενα: Κεφάλαιο 0, Κεφάλαιο 1

Χρόνος μελέτης: ~2 ώρες

Σύνοψη

Ένα πρόγραμμα δουλεύει με δεδομένα, και τα δεδομένα ζουν στη μνήμη του υπολογιστή. Σε αυτή τη διάλεξη βλέπουμε πώς οργανώνεται η μνήμη (bits, bytes, διευθύνσεις), πώς γράφουμε αριθμούς στο δυαδικό και στο δεκαεξαδικό σύστημα, και πώς η C μάς δίνει πρόσβαση στη μνήμη μέσα από **μεταβλητές**: ονόματα για κομμάτια μνήμης με συγκεκριμένο τύπο. Ο τύπος καθορίζει πόσα bytes πιάνει μια μεταβλητή και πώς ερμηνεύονται τα bits της (ως χαρακτήρας ASCII, ως ακέραιος με ή χωρίς πρόσημο, ως πραγματικός). Επειδή τα bytes είναι πεπερασμένα, και οι τιμές που χωράνε είναι πεπερασμένες: έτσι προκύπτει η υπερχείλιση ακεραίων, πηγή πραγματικών καταστροφών. Κλείνουμε με την `printf`, το βασικό μας εργαλείο για να βλέπουμε τις τιμές των μεταβλητών.

Θεωρία

§2.1 Bits και bytes

Το **bit** (binary digit, δυαδικό ψηφίο) είναι η μικρότερη μονάδα πληροφορίας σε έναν υπολογιστή: παίρνει μία από δύο τιμές, `0` ή `1`. Ένα μόνο bit λέει πολύ λίγα, γι' αυτό ο υπολογιστής ομαδοποιεί

τα bits. **Byte** είναι μια ομάδα από bits που αποθηκεύονται μαζί σε ένα κελί μνήμης. Σχεδόν πάντα ένα byte έχει 8 bits (λέγεται και octet, οκτάδα), π.χ. `00101010`.

Κάθε ένα από τα 8 bits είναι ανεξάρτητα `0` ή `1`, άρα υπάρχουν $2^8 = 256$ διαφορετικά bytes. Γενικά, με n bits φτιάχνουμε 2^n διαφορετικούς συνδυασμούς. Αυτός ο απλός κανόνας εξηγεί όλα τα εύρη τιμών που θα δούμε παρακάτω.

§2.2 Η μνήμη οργανώνεται σε bytes: διευθύνσεις

Η μνήμη είναι μια μεγάλη σειρά από bytes, το ένα μετά το άλλο: `Byte 0`, `Byte 1`, `Byte 2`, … . Το μέγεθός της μετριέται σε bytes: 1 KB (KiloByte) = 1.000 bytes, 1 MB (MegaByte) = 1.000.000 bytes, 1 GB (GigaByte) = 1.000.000.000 bytes. Οι σημειώσεις του μαθήματος χρησιμοποιούν τις δυνάμεις του 2 (1 KB = $2^{10} = 1024$ bytes κ.ο.κ.)· συναντώνται και οι δύο ορισμοί.

Η θέση ενός κελιού στη μνήμη λέγεται **διεύθυνση** (address). Μια μνήμη με N bytes έχει διευθύνσεις από 0 έως $N - 1$. Για παράδειγμα, αν στο `Byte 2` είναι αποθηκευμένο το `11100011`, λέμε ότι «στη διεύθυνση 2 υπάρχει το byte $11100011_{(2)}$». Ο επεξεργαστής διαβάζει και γράφει bytes δίνοντας τη διεύθυνσή τους. Οι διευθύνσεις θα γίνουν κεντρικές όταν φτάσουμε στους δείκτες (Κεφάλαιο 11).

§2.3 Κύρια και δευτερεύουσα μνήμη

Η **δευτερεύουσα μνήμη** (secondary memory) αποθηκεύει δεδομένα μόνιμα, σε υλικό που τα κρατάει ακόμα και χωρίς ρεύμα: σκληροί δίσκοι, flash drives (USB sticks), DVD. Η **κύρια μνήμη** (primary memory, η RAM) αποθηκεύει δεδομένα προσωρινά (χάνονται όταν σβήσει ο υπολογιστής), αλλά ο επεξεργαστής έχει άμεση πρόσβαση σε αυτά και μπορεί να τα επεξεργαστεί. Εκεί ζουν οι μεταβλητές ενός προγράμματος όσο τρέχει.

Όταν λέμε σκέτο «μνήμη», εννοούμε την κύρια μνήμη, εκτός αν διευκρινίσουμε κάτι άλλο.

§2.4 Δυαδικό, δεκαεξαδικό και οκταδικό σύστημα

Ένας **δυαδικός αριθμός** γράφεται με δύο σύμβολα, το `0` και το `1`· λέμε ότι εκφράζεται με βάση το 2, ή στο δυαδικό σύστημα, ή ότι έχει δυαδική αναπαράσταση. Όπως στο δεκαδικό, κάθε ψηφίο έχει βάρος μια δύναμη της βάσης, με το δεξιότερο ψηφίο να έχει βάρος 2^0. Έτσι

$$101010_{(2)} = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 42_{(10)}$$

Ο δείκτης (2) ή (10) δείχνει τη βάση. Αντιστρέφοντας τη διαδικασία βρίσκουμε τη δυαδική αναπαράσταση οποιουδήποτε δεκαδικού αριθμού (δείτε το παράδειγμα «Μετατροπές ανάμεσα σε βάσεις»).

Ένας **δεκαεξαδικός αριθμός** (hexadecimal) χρησιμοποιεί 16 σύμβολα: `0`–`9` και `A`–`F`, όπου `A` = 10, …, `F` = 15. Η μετατροπή γίνεται με τον ίδιο τρόπο:

$$2A_{(16)} = 2 \cdot 16^1 + 10 \cdot 16^0 = 42_{(10)}$$

Γιατί μας ενδιαφέρει το δεκαεξαδικό; Επειδή $16 = 2^4$, κάθε δεκαεξαδικό ψηφίο αντιστοιχεί ακριβώς σε 4 bits, οπότε **ένα byte γράφεται πάντα με δύο δεκαεξαδικά ψηφία** (`00` έως `FF`). Στο δεκαδικό χρειάζονται έως τρία ψηφία (`0` έως `255`), και τα ψηφία δεν αντιστοιχούν σε συγκεκριμένα bits. Για παράδειγμα, το ίδιο byte γράφεται:

Βάση	Αναπαράσταση
Δυαδικό	`0101 1111`
Δεκαεξαδικό	`5F` (`0101` = `5`, `1111` = `F`)
Δεκαδικό	`95` ($64 + 16 + 8 + 4 + 2 + 1$)

Υπάρχει και το **οκταδικό** σύστημα (octal, βάση 8, ψηφία `0`–`7`), όπου κάθε ψηφίο αντιστοιχεί σε 3 bits. Στη C και τα δύο εμφανίζονται στις σταθερές: `0x2A` είναι δεκαεξαδικό, `052` (με μηδενικό μπροστά) είναι οκταδικό.

§2.5 Μεταβλητές και δηλώσεις

Μεταβλητή (variable) είναι ένα τμήμα της μνήμης με συγκεκριμένο όνομα. Για να χρησιμοποιηθεί μια μεταβλητή στη C πρέπει πρώτα να **δηλωθεί** (declaration) με κάποιον **τύπο** (type). Η μορφή της δήλωσης είναι

```
τύπος όνομα;
```

Κάθε μέρος της δήλωσης λέει κάτι στον μεταγλωττιστή:

- ο **τύπος** λέει *πόση* μνήμη να δεσμεύσει για τα περιεχόμενα (και πώς να τα ερμηνεύει)·
- το **όνομα** κάνει τον μεταγλωττιστή να διαλέξει τη *διεύθυνση* της μνήμης όπου θα αποθηκευτεί η μεταβλητή· από εκεί και πέρα, κάθε φορά που γράφουμε το όνομα, ο μεταγλωττιστής ξέρει σε ποια bytes αναφερόμαστε.

Για παράδειγμα, με `int x;` ο μεταγλωττιστής μπορεί να δεσμεύσει 4 bytes για το `x` ξεκινώντας από το `Byte 0`, ενώ με `char c;` 1 byte, π.χ. το `Byte N-1`. Ποια ακριβώς διεύθυνση θα επιλεγεί δεν το ελέγχουμε εμείς· το σημαντικό είναι ότι το όνομα και ο τύπος αρκούν για να βρεθούν τα σωστά bytes.

§2.6 Ερμηνεία δυαδικών ψηφίων: χαρακτήρες ASCII

Τα bits στη μνήμη δεν έχουν από μόνα τους νόημα· το νόημα το δίνει ο τύπος. Τα 8 bits ενός `char` μπορούν να ερμηνευθούν ως **χαρακτήρας** κειμένου, με την αντιστοίχιση του πίνακα **ASCII**. Για παράδειγμα, ο αριθμός $67_{(10)} = 43_{(16)}$ αντιστοιχεί στο γράμμα `'C'`, το 65 στο `'A'`, το 10 στην αλλαγή γραμμής `'\n'` και το 0 στον «κενό» χαρακτήρα `'\0'`.

Ο πίνακας ASCII έχει **128** χαρακτήρες (κωδικοί 0–127), επειδή είναι κώδικας 7 bits: $2^7 = 128$. Οι πρώτοι 32 είναι χαρακτήρες ελέγχου (αλλαγή γραμμής, tab, «καμπανάκι» κ.ά.) και οι υπόλοιποι γράμματα, ψηφία και σύμβολα. Ολόκληρο τον πίνακα τον βλέπετε στο τερματικό με την εντολή `man ascii`: η `man` είναι η εντολή του Linux που μας δίνει το εγχειρίδιο (manual) για ένα πρόγραμμα ή θέμα του συστήματος.

§2.7 Αναπαράσταση ακεραίων στη μνήμη

Ο τύπος `int`, που αναπαριστά ακεραίους, πιάνει *συνήθως* 4 bytes, δηλαδή 32 bits. Ο αριθμός 42 αποθηκεύεται ως

```
00000000000000000000000000101010   (4 bytes = 32 bit)
```

και η μετατροπή στο δεκαδικό γίνεται όπως πριν, από το δεξιότερο bit: $0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 + 1 \cdot 2^3 + \cdots = 42$.

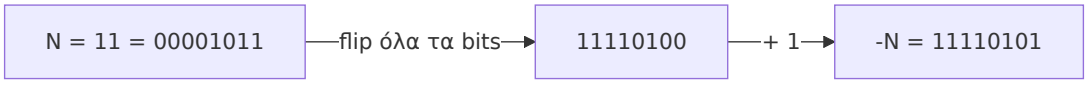
Με 32 bits αναπαριστούμε $2^{32} = 4.294.967.296$ διαφορετικούς ακεραίους. Αυτό φέρνει δύο προβλήματα:

1. **Το εύρος είναι πεπερασμένο.** Περίπου 4 δισεκατομμύρια τιμές ίσως να μην αρκούν για το πρόγραμμά μας. Ευτυχώς η C προσφέρει τύπους με μεγαλύτερο μέγεθος (π.χ. 64 bit). Το πρόβλημα είναι υπαρκτό: το πρόβλημα του 2038 οφείλεται στο ότι πολλά συστήματα μετράνε τον χρόνο σε δευτερόλεπτα σε έναν προσημασμένο ακέραιο 32 bit, που θα εξαντληθεί το 2038.
2. **Υπάρχουν και αρνητικοί ακέραιοι.** Πώς αναπαριστούμε στο δυαδικό ότι ένας αριθμός είναι αρνητικός; Η απάντηση της C (και σχεδόν όλων των επεξεργαστών) είναι το συμπλήρωμα ως προς 2.

§2.8 Συμπλήρωμα ως προς 2

Στο **συμπλήρωμα ως προς 2** (two's complement) το πρόσημο φαίνεται από το σημαντικότερο (αριστερότερο) bit του αριθμού: `0` σημαίνει θετικό (ή μηδέν), `1` σημαίνει αρνητικό. Για να πάρουμε την αναπαράσταση του $-N$ από έναν θετικό δυαδικό N:

1. αντιστρέφουμε (flip) όλα τα bits του N·
2. προσθέτουμε 1.



Σχήμα: τα δύο βήματα για το -11 σε 8 bits.

Γιατί αυτή η επιλογή; Επειδή με αυτήν η πρόσθεση δουλεύει με το ίδιο κύκλωμα για θετικούς και αρνητικούς: αν προσθέσετε `00001011` και `11110101`, παίρνετε `1 00000000`, και το κρατούμενο που περισσεύει από τα 8 bits χάνεται, οπότε μένει 0. Συνέπεια για τα εύρη: με n bits ένας

προσημασμένος ακέραιος παίρνει τιμές από -2^{n-1} έως $2^{n-1}-1$ (για 8 bits: -128 έως 127), ενώ ένας μη προσημασμένος (unsigned) από 0 έως 2^n-1 (για 8 bits: 0 έως 255).

§2.9 Οι τύποι μεταβλητών της C

Ο πίνακας συνοψίζει τους βασικούς τύπους, με τα συνηθισμένα μεγέθη που δίνουν οι διαφάνειες. Τα μεγέθη δεν είναι εγγυημένα: εξαρτώνται από τον μεταγλωττιστή και το σύστημα.

Τύπος	Συνήθως (bytes)	Εύρος τιμών	Παράδειγμα τιμής
`char`	1	-128 ... 127	`'B'`, `0x42`
`short int`	2	-32.768 ... 32.767	`42`
`int`	4	$-2.147.483.648$ ... $2.147.483.647$	`42`
`long int`	4	όπως το `int`	`42L`
`float`	4	θετικές από $1.17 \cdot 10^{-38}$ έως $3.4 \cdot 10^{38}$	`42.42F`
`double`	8	θετικές από $2.2 \cdot 10^{-308}$ έως $1.8 \cdot 10^{308}$	`42.42`
`long double`	8, 10, 12, 16		`42.42L`
`unsigned char`	1	0 ... 255	`'B'`, `0x42`
`unsigned short int`	2	0 ... 65.535	`42`
`unsigned int`	4	0 ... 4.294.967.295	`42`
`unsigned long int`	4	0 ... 4.294.967.295	`42LU`

Παρατηρήστε ότι:

- κάθε ακέραιος τύπος έχει και μια `unsigned` εκδοχή με το ίδιο μέγεθος, που δεν αναπαριστά αρνητικούς και φτάνει στο διπλάσιο θετικό όριο·
- τα εύρη βγαίνουν κατευθείαν από τον κανόνα 2^n και το συμπλήρωμα ως προς 2·
- τα γράμματα στο τέλος μιας σταθεράς ορίζουν τον τύπο της: `L` για `long`, `U` για `unsigned`, `F` για `float`· ένας πραγματικός χωρίς κατάληξη (`42.42`) είναι `double`·
- ο `long int` είναι 4 bytes σε κάποια συστήματα (π.χ. Windows) αλλά 8 bytes στο Linux 64 bit που χρησιμοποιούμε στο εργαστήριο. Ο `long long int` (δεν είναι στον πίνακα των διαφανειών) είναι συνήθως 8 bytes σε όλα.

§2.10 Ανάθεση σε μεταβλητή

Ανάθεση (assignment) είναι η αποθήκευση μιας τιμής σε μια μεταβλητή. Μπορεί να γίνει κατά τον ορισμό της (αυτό λέγεται **αρχικοποίηση**, initialization) ή αργότερα:

```
int x = 42;   // ανάθεση κατά τον ορισμό
int y;
y = 42;       // ανάθεση μετά τον ορισμό
int z = 0x2A; // η ίδια τιμή σε δεκαεξαδικό
int w = 052;  // και σε οκταδικό
```

Και οι τέσσερις μεταβλητές έχουν την ίδια τιμή: 42. Η βάση στην οποία γράφουμε μια σταθερά αφορά μόνο τον πηγαίο κώδικα· στη μνήμη αποθηκεύονται πάντα τα ίδια bits.

Τι υπάρχει σε μια μεταβλητή *πριν* την ανάθεση; Ό,τι έτυχε να βρίσκεται σε εκείνα τα bytes: ο μεταγλωττιστής δεσμεύει τη μνήμη, αλλά δεν τη «καθαρίζει». Γι' αυτό μια μεταβλητή χωρίς ανάθεση έχει απρόβλεπτη τιμή («σκουπίδια») και δεν πρέπει να τη διαβάζουμε. Μετά την `x = 42;`, τα 4 bytes της `x` γίνονται:

Byte	Πριν την ανάθεση	Μετά την ανάθεση
`Byte 0`	`00101010`	`00000000`
`Byte 1`	`01000010`	`00000000`
`Byte 2`	`11100011`	`00000000`
`Byte 3`	`11100011`	`00101010`

Η διάταξη αυτή (το λιγότερο σημαντικό byte στο τέλος) είναι η σχηματική των διαφανειών. Στους επεξεργαστές x86 που χρησιμοποιούμε, τα bytes ενός `int` αποθηκεύονται με την αντίστροφη σειρά (little endian)· το θέμα επανέρχεται στο Κεφάλαιο 12.

§2.11 Υπερχείλιση ακεραίων

Αφού ένας `int` χωράει τιμές έως 2.147.483.647, τι γίνεται αν ένας υπολογισμός βγάλει μεγαλύτερο αποτέλεσμα; Τα bits που δεν χωράνε χάνονται και το αποτέλεσμα «τυλίγεται» γύρω από το εύρος: αυτό λέγεται **υπερχείλιση ακεραίων** (integer overflow). Για παράδειγμα, $2.000.000.000 + 2.000.000.000 = 4.000.000.000$ δεν χωράει σε `int`, και το πρόγραμμα τυπώνει $4.000.000.000 - 2^{32} = -294.967.296$ (δείτε το παράδειγμα «Το πρόγραμμα overflow»). Όπως στα γνωστά σκίτσα με τα προβατάκια: μετράμε `32767` και το επόμενο είναι `-32768`.

Για προσημασμένους ακεραίους, η υπερχείλιση είναι στην πραγματικότητα **απροσδιόριστη συμπεριφορά** (undefined behavior) στη C: το «τύλιγμα» είναι αυτό που συνήθως βλέπουμε, αλλά ο μεταγλωττιστής δεν το εγγυάται. Για `unsigned` τύπους το τύλιγμα (modulo 2^n) είναι εγγυημένο. Η διόρθωση είναι πάντα η ίδια: επιλέξτε τύπο με αρκετό εύρος για τις τιμές που θα χρειαστείτε.

Η υπερχείλιση δεν είναι θεωρητικό πρόβλημα. Η πρώτη πτήση του πυραύλου Ariane 5 (1996) κατέληξε σε καταστροφή, επειδή μια πραγματική τιμή μετατράπηκε σε ακέραιο 16 bit που δεν τη χωρούσε. Το πρόβλημα του 2000 (αποθήκευση του έτους με δύο ψηφία) και το πρόβλημα του 2038 είναι συγγενικά προβλήματα αναπαράστασης.

§2.12 Ακέραιοι με συγκεκριμένο μέγεθος: sizeof και stdint.h

Είπαμε ότι ο `int` είναι *συνήθως* 4 bytes. Υπάρχουν δύο τρόποι να είμαστε βέβαιοι:

- Ο τελεστής `sizeof` δίνει το μέγεθος ενός τύπου ή μιας μεταβλητής σε bytes, π.χ. `sizeof(int)`. Θα τον δούμε αναλυτικά σε επόμενες διαλέξεις.
- Η βιβλιοθήκη `<stdint.h>` ορίζει ακέραιους τύπους με εγγυημένο πλήθος bits: `int8_t`, `uint8_t`, `int16_t`, `uint16_t`, `int32_t`, `uint32_t`, `int64_t`, `uint64_t`. Ο αριθμός είναι τα bits, και το `u` σημαίνει `unsigned`.

§2.13 Η συνάρτηση printf

Η συνάρτηση `printf()` τυπώνει δεδομένα στο αρχείο εξόδου **stdout** (standard output), που συνήθως είναι το τερματικό. Έχει μεταβλητή λίστα παραμέτρων:

- Η πρώτη παράμετρος είναι το **αλφαριθμητικό μορφοποίησης** (format string): μια ακολουθία χαρακτήρων μέσα σε διπλά εισαγωγικά `" "`, που καθορίζει πώς θα τυπωθούν τα δεδομένα.
- Οι επόμενες παράμετροι είναι προαιρετικές· αν υπάρχουν, η `printf()` χρησιμοποιεί τις τιμές τους.

Το format string μπορεί να περιέχει τρία είδη πραγμάτων: απλούς χαρακτήρες, που τυπώνονται όπως είναι· **ακολουθίες διαφυγής** (escape sequences)· και **προσδιοριστικά μορφοποίησης** (format specifiers). Για παράδειγμα, στο `printf("x = %d\n", x);` το `x =` τυπώνεται αυτούσιο, το `%d` αντικαθίσταται από την τιμή του `x` ως ακέραιος και το `\n` αλλάζει γραμμή.

§2.14 Ακολουθίες διαφυγής

Μια ακολουθία διαφυγής αποτελείται από μια ανάστροφη κεκλιμένη `\` (backslash) και έναν χαρακτήρα. Χρησιμεύει για χαρακτήρες που δεν γράφονται εύκολα μέσα σε εισαγωγικά: αλλαγή γραμμής, tab, ή τα ίδια τα `"` και `\`.

Ακολουθία	Σημασία
`\n`	Αλλαγή γραμμής, σαν το πλήκτρο Enter
`\r`	Επαναφορά του δρομέα (cursor) στην αρχή της τρέχουσας γραμμής
`\t`	Κενό ίσο με ένα tab, σαν το πλήκτρο Tab
`\\`	Ανάστροφη κεκλιμένη (backslash)
`\"`	Διπλά εισαγωγικά (double quotes)
`\xNN`	Ο χαρακτήρας με κωδικό `NN` σε δεκαεξαδικό
`\a`	Ηχητικό σήμα (bell)

Οι ακολουθίες διαφυγής είναι χαρακτήρες ASCII: στον πίνακα της `man ascii` θα δείτε το `'\n'` με κωδικό 10, το `'\t'` με 9, το `'\a'` με 7. Έτσι το `"\x43"` τυπώνει `C`.

§2.15 Προσδιοριστικά μορφοποίησης

Ένα προσδιοριστικό μορφοποίησης αποτελείται από τον χαρακτήρα `%` και έναν ή περισσότερους χαρακτήρες που λένε πώς να γίνει η μορφοποίηση. Κάθε προσδιοριστικό «καταναλώνει» με τη σειρά την επόμενη παράμετρο της `printf`.

Προσδιοριστικό	Σημασία
`%c`	Χαρακτήρας ASCII
`%d` ή `%i`	Ακέραιος (`int`)
`%u`	Μη προσημασμένος (`unsigned`) ακέραιος
`%f`	Πραγματικός (`float` ή `double`)
`%llu`	`unsigned long long int`
`%%`	Ο ίδιος ο χαρακτήρας `%`

Τα προσδιοριστικά πρέπει να ταιριάζουν με τον τύπο της τιμής: το `%d` περιμένει `int`, το `%lld` `long long`, το `%ld` `long`. Η πλήρης λίστα είναι στο εγχειρίδιο της printf· το Εργαστήριο 3 έχει έναν συνοπτικό πίνακα. Το ίδιο byte μπορεί να τυπωθεί με διαφορετικούς τρόπους: με `%c` ένας `char` που έχει τιμή 67 τυπώνεται `C`, με `%d` τυπώνεται `67`.

§2.16 Δηλώσεις πολλών μεταβλητών και δεσμευμένες λέξεις

Μεταβλητές του ίδιου τύπου μπορούν να δηλωθούν στην ίδια γραμμή, χωρισμένες με κόμμα. Οι δύο παρακάτω μορφές είναι ισοδύναμες:

```
int a;
int b;
int c;
```

```
int a, b, c;
```

Τα ονόματα μεταβλητών αποτελούνται από γράμματα, ψηφία και `_`, και δεν ξεκινούν με ψηφίο. Οι **δεσμευμένες λέξεις** (reserved keywords) της C έχουν προκαθορισμένο νόημα για τον μεταγλωττιστή και **δεν** επιτρέπεται να χρησιμοποιηθούν ως ονόματα μεταβλητών ή συναρτήσεων:

```
auto      do        goto      signed    unsigned
break     double    if        sizeof    void
case      else      int       static    volatile
char      enum      long      struct    while
const     extern    register  switch
continue  for       return    typedef
default   float     short     union
```

Πολλές από αυτές τις γνωρίσατε ήδη (`int`, `char`, `return`)· οι υπόλοιπες θα εμφανιστούν στα επόμενα κεφάλαια.

Παραδείγματα

§2.17 Μετατροπές ανάμεσα σε βάσεις

Εφαρμόζει: «Δυαδικό, δεκαεξαδικό και οκταδικό σύστημα».

Από δυαδικό σε δεκαδικό: πολλαπλασιάζουμε κάθε ψηφίο με τη δύναμη του 2 της θέσης του και προσθέτουμε: $101010_{(2)} = 32 + 8 + 2 = 42$. Με τα βάρη `128 64 32 16 8 4 2 1` γραμμένα πάνω από τα bits, το `01011111` δίνει $64 + 16 + 8 + 4 + 2 + 1 = 95$.

Από δεκαδικό σε δυαδικό: αντιστρέφουμε τη διαδικασία. Διαιρούμε διαδοχικά με το 2 και κρατάμε τα υπόλοιπα· διαβασμένα από το τελευταίο προς το πρώτο, δίνουν τα bits:

```
42 / 2 = 21  υπόλοιπο 0
21 / 2 = 10  υπόλοιπο 1
10 / 2 =  5  υπόλοιπο 0
 5 / 2 =  2  υπόλοιπο 1
 2 / 2 =  1  υπόλοιπο 0
 1 / 2 =  0  υπόλοιπο 1   ->  42 = 101010 (2)
```

Από δυαδικό σε δεκαεξαδικό και πίσω: χωρίζουμε σε τετράδες από δεξιά και μετατρέπουμε κάθε τετράδα σε ένα ψηφίο: `0101 1111` ⟶ `5F`, `0010 1010` ⟶ `2A`. Αντίστροφα, `2A` ⟶ `0010 1010`. Έτσι τα `42`, `0x2A` και `052` στη C είναι ο ίδιος αριθμός.

§2.18 Ο πίνακας ASCII στο τερματικό

Εφαρμόζει: «Ερμηνεία δυαδικών ψηφίων: χαρακτήρες ASCII».

Η διάλεξη ρώτησε: ποια εντολή του Linux μάς επιτρέπει να μάθουμε για ένα πρόγραμμα; Η απάντηση είναι η `man`, που δεν περιορίζεται σε προγράμματα:

```
man ascii
```

Ανοίγει τη σελίδα `ascii(7)` με στήλες `Oct`, `Dec`, `Hex`, `Char`. Εκεί διαβάζετε, για παράδειγμα:

```
Oct   Dec   Hex   Char
103   67    43    C
012   10    0A    LF  '\n' (new line)
```

Βγαίνετε με `q`. Το ίδιο κάνετε και για τις συναρτήσεις της C: `man 3 printf`.

§2.19 Συμπλήρωμα ως προς 2 για το −11

Εφαρμόζει: «Συμπλήρωμα ως προς 2».

Έστω $N = 11_{(10)} = 00001011_{(2)}$ σε 8 bits. Για το $-N$:

```
N           00001011
```

```
flip        11110100
+ 1       + 00000001
-N          11110101
```

Επαλήθευση: το σημαντικότερο bit είναι 1, άρα ο αριθμός είναι αρνητικός· και `00001011 + 11110101 = 1 00000000`, όπου το 9ο bit χάνεται και μένει $0 = 11 + (-11)$. Αν διαβάσουμε τα ίδια bits ως `unsigned char`, παίρνουμε $245 = 256 - 11$: τα bits είναι ίδια, αλλάζει μόνο η ερμηνεία.

§2.20 Το πρόγραμμα overflow

Εφαρμόζει: «Υπερχείλιση ακεραίων», «Οι τύποι μεταβλητών της C».

Τι θα τυπώσει το παρακάτω πρόγραμμα;

```c
#include <stdio.h>

int main() {
  printf("%d\n", 2000000000 + 2000000000);
  return 0;
}
```

```
$ ./overflow
-294967296
```

Τι συνέβη; Και οι δύο σταθερές είναι `int`, άρα και το άθροισμα υπολογίζεται ως `int`. Το σωστό αποτέλεσμα, $4.000.000.000$, ξεπερνάει το μέγιστο $2.147.483.647$, και τα 32 bits που μένουν ερμηνεύονται σε συμπλήρωμα ως προς 2 ως $4.000.000.000 - 2^{32} = -294.967.296$. Ο gcc συνήθως προειδοποιεί ήδη στη μεταγλώττιση (`integer overflow in expression of type 'int'`)· μην αγνοείτε τις προειδοποιήσεις.

Πώς το διορθώνουμε; Κάνουμε τον υπολογισμό σε τύπο 64 bit και τυπώνουμε με το αντίστοιχο προσδιοριστικό:

```c
#include <stdio.h>

int main() {
  printf("%lld\n", 2000000000LL + 2000000000LL);
  return 0;
}
```

```
$ ./overflow
4000000000
```

Η κατάληξη `LL` κάνει τις σταθερές `long long`. Αφού το αποτέλεσμα είναι θετικό, θα μπορούσαμε να χρησιμοποιήσουμε και `unsigned int` με `%u` (χωράει έως $4.294.967.295$), αλλά με πολύ μικρό περιθώριο.

§2.21 Τύποι και printf σε ένα πρόγραμμα

Εφαρμόζει: «Ερμηνεία δυαδικών ψηφίων», «Ακολουθίες διαφυγής», «Προσδιοριστικά μορφοποίησης». Το πρόγραμμα δεν είναι από τις διαφάνειες· συνδυάζει τους πίνακές τους για να τους δοκιμάσετε:

```c
#include <stdio.h>

int main() {
  char c = 'C';
  unsigned int u = 4000000000U;
  printf("c = %c (%d), x = %d\n", c, c, 0x2A);
  printf("u = %u, sizeof(int) = %zu\n", u, sizeof(int));
  printf("100%% \"done\"\tbye\\\n");
  return 0;
}
```

```
$ ./types
c = C (67), x = 42
u = 4000000000, sizeof(int) = 4
100% "done" bye\
```

Ο ίδιος `char` τυπώνεται ως γράμμα με `%c` και ως κωδικός ASCII με `%d`· το `4000000000` χωράει σε `unsigned int` αλλά όχι σε `int`· το αποτέλεσμα της `sizeof` τυπώνεται με `%zu`· στην τελευταία γραμμή, `%%` δίνει `%`, `\"` δίνει `"`, `\t` ένα tab και `\\` μια `\`.

§2.22 Συμβουλές για το εργαστήριο

- Στο Εργαστήριο 2, τυπώστε το `number` του `readint.c` *πριν* το διαβάσετε: θα δείτε ό,τι είχε η μνήμη. Η επέκταση «Υψηλή ακρίβεια» του `calc.c` ζητάει αριθμούς έως 10^{18}, που δεν χωράνε σε `int`· χρειάζεστε `long long` με `%lld`.
- Το Εργαστήριο 3 έχει στην αρχή πίνακα με τα προσδιοριστικά (`%d`, `%ld`, `%lld`, `%u`, `%lu`, `%llu`, `%f`, `%Lf`). Παράξενα αρνητικά αποτελέσματα για μεγάλους αριθμούς σημαίνουν σχεδόν πάντα υπερχείλιση.

Κύρια σημεία

1. Το bit είναι η μικρότερη μονάδα πληροφορίας (0 ή 1)· ένα byte είναι συνήθως 8 bits, και γενικά με n bits αναπαριστούμε 2^n τιμές (256 για ένα byte).
2. Η μνήμη είναι μια σειρά από bytes και η θέση κάθε byte λέγεται διεύθυνση· «μνήμη» σκέτο σημαίνει την κύρια (προσωρινή, άμεσα προσβάσιμη) και όχι τη δευτερεύουσα (μόνιμη) μνήμη.
3. Ο ίδιος αριθμός γράφεται διαφορετικά σε κάθε βάση· ένα byte γράφεται πάντα με δύο δεκαεξαδικά ψηφία, επειδή κάθε δεκαεξαδικό ψηφίο είναι 4 bits.

4. Μεταβλητή είναι ένα τμήμα της μνήμης με όνομα και πρέπει να δηλωθεί με τύπο· ο τύπος ορίζει πόση μνήμη δεσμεύεται και το όνομα οδηγεί στη διεύθυνσή της.
5. Τα bits δεν έχουν νόημα από μόνα τους: ο τύπος ορίζει αν ερμηνεύονται ως χαρακτήρας ASCII (128 χαρακτήρες, `man ascii`), ακέραιος ή πραγματικός.
6. Ο `int` είναι συνήθως 4 bytes (32 bits), άρα έχει πεπερασμένο εύρος· τα μεγέθη των τύπων είναι «συνήθη» και όχι εγγυημένα, και κάθε ακέραιος έχει `unsigned` εκδοχή.
7. Οι αρνητικοί ακέραιοι αναπαρίστανται σε συμπλήρωμα ως προς 2: αντιστρέφουμε όλα τα bits και προσθέτουμε 1· το σημαντικότερο bit δείχνει το πρόσημο.
8. Μια μεταβλητή παίρνει τιμή κατά τον ορισμό της ή αργότερα, με σταθερά σε δεκαδικό, δεκαεξαδικό (`0x2A`) ή οκταδικό (`052`)· πριν την πρώτη ανάθεση περιέχει ό,τι υπήρχε στη μνήμη.
9. Όταν ένα αποτέλεσμα δεν χωράει στον τύπο του έχουμε υπερχείλιση: το `2000000000 + 2000000000` σε `int` τυπώνει `-294967296`· η διόρθωση είναι τύπος με μεγαλύτερο εύρος.
10. Με `sizeof` μαθαίνουμε το μέγεθος ενός τύπου, και με το `<stdint.h>` δηλώνουμε ακεραίους με ακριβές πλήθος bits (`int32_t`, `uint64_t` κ.ά.).
11. Η `printf` τυπώνει στο stdout· το format string περιέχει απλούς χαρακτήρες, ακολουθίες διαφυγής (`\n`, `\t`, `\\`, `\"`) και προσδιοριστικά (`%c`, `%d`, `%u`, `%f`, `%llu`, `%%`) που πρέπει να ταιριάζουν με τους τύπους των τιμών.
12. Μεταβλητές ίδιου τύπου δηλώνονται μαζί με κόμμα (`int a, b, c;`), και οι δεσμευμένες λέξεις δεν μπορούν να γίνουν ονόματα.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
δυαδικό ψηφίο	bit (binary digit)	Η μικρότερη μονάδα πληροφορίας: 0 ή 1
byte, οκτάδα	byte, octet	Ομάδα (συνήθως 8) bits σε ένα κελί μνήμης
διεύθυνση	address	Η θέση ενός κελιού (byte) στη μνήμη
κύρια μνήμη	primary memory (RAM)	Προσωρινή μνήμη με άμεση πρόσβαση από τον επεξεργαστή
δευτερεύουσα μνήμη	secondary memory	Μόνιμη αποθήκευση: δίσκοι, flash, DVD
δυαδικό σύστημα	binary	Αρίθμηση με βάση το 2
δεκαεξαδικό σύστημα	hexadecimal	Αρίθμηση με βάση το 16 (`0`–`9`, `A`–`F`)
οκταδικό σύστημα	octal	Αρίθμηση με βάση το 8
μεταβλητή	variable	Τμήμα της μνήμης με όνομα και τύπο
δήλωση	declaration	Εισαγωγή μεταβλητής με τύπο και όνομα
τύπος	type	Πόση μνήμη πιάνει μια τιμή και πώς ερμηνεύεται

Ελληνικά	English	Σύντομος ορισμός
ανάθεση	assignment	Αποθήκευση τιμής σε μεταβλητή (`x = 42;`)
αρχικοποίηση	initialization	Ανάθεση κατά τον ορισμό (`int x = 42;`)
πίνακας ASCII	ASCII table	Αντιστοίχιση κωδικών 0–127 σε χαρακτήρες
συμπλήρωμα ως προς 2	two's complement	Αναπαράσταση αρνητικών: flip όλα τα bits και +1
μη προσημασμένος	unsigned	Ακέραιος τύπος χωρίς αρνητικές τιμές
υπερχείλιση ακεραίων	integer overflow	Αποτέλεσμα που δεν χωράει στον τύπο του
αλφαριθμητικό μορφοποίησης	format string	Η πρώτη παράμετρος της `printf`
ακολουθία διαφυγής	escape sequence	`\` και ένας χαρακτήρας, π.χ. `\n`
προσδιοριστικό μορφοποίησης	format specifier	`%` και χαρακτήρες, π.χ. `%d`
τυπική έξοδος	standard output (stdout)	Το αρχείο όπου γράφει η `printf`
δεσμευμένη λέξη	reserved keyword	Λέξη της C που δεν γίνεται όνομα (`int`, `if`, ...)

Διάβασμα

- **Διαφάνειες**: Διάλεξη 2, σελ. 1–35. Bits, bytes και διευθύνσεις: σελ. 5–8· βάσεις αρίθμησης: σελ. 9–11· δήλωση μεταβλητής: σελ. 12–15· ASCII: σελ. 16–17· αναπαράσταση ακεραίων και συμπλήρωμα ως προς 2: σελ. 18–22· τύποι: σελ. 23· ανάθεση: σελ. 24–25· υπερχείλιση και `<stdint.h>`: σελ. 26–27· `printf`: σελ. 28–30· δηλώσεις πολλών μεταβλητών και δεσμευμένες λέξεις: σελ. 31–32.
- **Σημειώσεις**: Οι διαφάνειες συνιστούν να έχετε καλύψει τις σημειώσεις του κ. Σταματόπουλου μέχρι τη σελίδα 35 και τις σελίδες 58–71 (K04). Για αυτό το κεφάλαιο ειδικά:
  - Κεφάλαιο 0: Εισαγωγή, ενότητες «Η δομή του υπολογιστή», «Η πληροφορία στον υπολογιστή» (K04, σελ. 9–10)
  - Κεφάλαιο 2: Μεταβλητές, τύποι, τελεστές και παραστάσεις, ενότητα «Μεταβλητές, σταθερές, τύποι και δηλώσεις στην C» (K04, σελ. 30–33)
  - Οι σελίδες 58–71 καλύπτουν το Κεφάλαιο 4: Συναρτήσεις μέχρι την ενότητα «Μετατροπές μεταξύ δεκαδικών και δυαδικών αριθμών», προετοιμασία για το Κεφάλαιο 3.
- **Εργαστήριο**:
  - Εργαστήριο 2: «Βήμα 3» (`readint.c`), ασκήσεις `calc.c` (με την επέκταση «Υψηλή

ακρίβεια») και `pyth.c`
  - Εργαστήριο 3: ο πίνακας προσδιοριστικών της `printf` και η άσκηση `seq.c`
- **Βιβλίο**: K&R, §2.1–2.4, όπως προτείνουν οι σημειώσεις.
- **Άλλα**:
  - Bits and bytes (Stanford CS101)
  - Wikipedia: ASCII, Escape sequences in C, Two's complement, Integer overflow
  - `printf`: tips και reference
  - Συμπλήρωμα ως προς 2: γραπτή εξήγηση και βίντεο
  - Προβλήματα αναπαράστασης: Ariane 5, Year 2000, Year 2038, xkcd 2697
  - Τερματικό: `man ascii`, `man 3 printf`

Συχνά λάθη

- **Υπερχείλιση σε `int`.** Γινόμενα ή αθροίσματα μεγάλων αριθμών βγαίνουν ξαφνικά αρνητικά (`2000000000 + 2000000000` ⟶ `-294967296`). Διόρθωση: `long long` με `%lld`, ή `unsigned long long` με `%llu`, και σταθερές με κατάληξη `LL`.
- **Μεγάλος τύπος, μικρός υπολογισμός.** `long long r = 2000000000 + 2000000000;` εξακολουθεί να υπερχειλίζει, γιατί το άθροισμα υπολογίζεται σε `int` *πριν* την ανάθεση. Διόρθωση: `2000000000LL + 2000000000`.
- **Λάθος προσδιοριστικό στην `printf`.** `printf("%d\n", x)` με `long long x` ή `printf("%d\n", 3.14)` τυπώνει σκουπίδια· ο gcc με `-Wall` προειδοποιεί (`format '%d' expects argument of type 'int'`). Διόρθωση: το προσδιοριστικό του τύπου (`%lld`, `%f`, `%zu` για `sizeof`).
- **Ανάγνωση μεταβλητής χωρίς τιμή.** `int n; printf("%d\n", n);` τυπώνει ό,τι υπήρχε στη μνήμη. Διόρθωση: αρχικοποιείτε πάντα πριν διαβάσετε.
- **Μηδενικό μπροστά από σταθερά.** `int x = 052;` δεν είναι 52 αλλά 42, γιατί είναι οκταδικό. Διόρθωση: μη γράφετε αρχικά μηδενικά σε δεκαδικές σταθερές.
- **`%` ή `\` χωρίς διαφυγή.** `printf("100%\n")` δεν τυπώνει 100%. Διόρθωση: `%%` και `\\`.
- **Δεσμευμένη λέξη ως όνομα.** `int int = 5;` ή `int new, for;` δεν μεταγλωττίζεται (`expected identifier`). Διόρθωση: άλλο όνομα.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K2.14** Ο κωδικός ASCII του 'J' (33% σωστές): Το 44% απάντησε `0x50`, κάνοντας την πρόσθεση σαν να ήταν δεκαδικό (49 + 1 = 50)· στο δεκαεξαδικό μετά το 9 έρχεται το A.
- **K2.15** Το 0xFF σε signed char (33% σωστές): Το 41% απάντησε 256, που δεν χωράει καν σε 8 bits (το `0xFF` ως unsigned είναι 255)· σε `signed char` με συμπλήρωμα ως προς 2 όλοι οι άσοι σημαίνουν αρνητικό αριθμό.
- **K2.13** Το μέγεθος του int (37% σωστές): Το 46% απάντησε 4, που είναι το συνηθισμένο

μέγεθος στα σημερινά συστήματα, αλλά το πρότυπο της C δεν το ορίζει· σε άλλες αρχιτεκτονικές ο `int` μπορεί να είναι 2 bytes.

- **K2.12** Πόσο είναι το 2^64 (38% σωστές): Το 48% απάντησε 10^{30}, υπερεκτιμώντας κατά 10 τάξεις μεγέθους· με $2^{10} \approx 10^3$, το 2^{64} είναι περίπου $1.8 \cdot 10^{19}$.
- **K2.11** Ο μετρητής του Gangnam Style (41% σωστές): Το 26% διάλεξε `int64_t`, που είναι ο τύπος στον οποίο μετέβη η Google *μετά*· ένας 64-bit ακέραιος δεν θα κόντευε ποτέ να υπερχειλίσει στα 2 δις.
- **K2.8** Πλήθος διευθύνσεων IPv4 (57% σωστές): Το 25% απάντησε 2^{16}, μετρώντας 16 bits αντί για 32 (4 bytes × 8 bits).
- **K2.7** Άθροισμα θετικών unsigned int (63% σωστές): Το 30% απάντησε «Όχι», μεταφέροντας τη συμπεριφορά του `int` (όπου η υπερχείλιση δίνει αρνητικό) στους unsigned· ένας `unsigned int` αναδιπλώνεται modulo 2^{32} και δεν γίνεται ποτέ αρνητικός (αυστηρά, μπορεί να βγει 0 ή μικρότερος από τους προσθετέους).

Ερωτήσεις κατανόησης

- **E2.1** Πόσες διαφορετικές τιμές παίρνει μια ομάδα 16 bits;[19]
- **E2.2** Μετατρέψτε το $1100100_{(2)}$ σε δεκαδικό και σε δεκαεξαδικό.[20]
- **E2.3** Τι αποφασίζει ο μεταγλωττιστής από τον τύπο και τι από το όνομα μιας δήλωσης όπως `int x;`;[21]
- **E2.4** Ποια είναι η αναπαράσταση του -1 σε 8 bits με συμπλήρωμα ως προς 2;[22]
- **E2.5** Γιατί ο `char` έχει εύρος -128 έως 127 ενώ ο `unsigned char` 0 έως 255;[23]
- **E2.6** Τι τιμή έχει η `int y = 0x10 + 010;`;[24]
- **E2.7** Τι τυπώνει η `printf("%c%c\n", 72, 0x69);`;[25]
- **E2.8** Γιατί το `long long r = 2000000000 + 2000000000;` δεν δίνει 4000000000;[26]
- **E2.9** Ποιο προσδιοριστικό χρησιμοποιείτε για `unsigned int` και ποιο για να τυπώσετε το σύμβολο `%`;[27]

Kahoot από το αμφιθέατρο (K2.1–K2.15)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

[19] $2^{16} = 65.536$, όσες και οι τιμές του `short` ή του `unsigned short`.

[20] $64 + 32 + 4 = 100_{(10)}$· σε τετράδες `0110 0100`, δηλαδή $64_{(16)}$.

[21] Από τον τύπο πόσα bytes θα δεσμεύσει (και πώς θα τα ερμηνεύει)· από το όνομα, σε ποια διεύθυνση θα αποθηκεύσει τη μεταβλητή, ώστε να τη βρίσκει κάθε φορά που τη χρησιμοποιούμε.

[22] `00000001` → flip `11111110` → +1 `11111111`.

[23] Και οι δύο έχουν 8 bits, δηλαδή 256 τιμές. Ο προσημασμένος τις μοιράζει σε αρνητικές και μη αρνητικές (-2^7 έως $2^7 - 1$), ο `unsigned` τις κρατάει όλες για μη αρνητικές (0 έως $2^8 - 1$).

[24] $16 + 8 = 24$: το `0x10` είναι δεκαεξαδικό και το `010` οκταδικό.

[25] `Hi`: 72 είναι ο κωδικός ASCII του `H` και $69_{(16)} = 105$ του `i`.

[26] Οι δύο σταθερές είναι `int`, οπότε το άθροισμα υπολογίζεται σε `int` και υπερχειλίζει πριν ανατεθεί στο `r`. Χρειάζεται `2000000000LL`.

[27] `%u` για `unsigned int` και `%%` για το `%`.

- K2.1 Bit ή byte;: 93% σωστές απαντήσεις
- K2.2 Άθροισμα θετικών int: 87% σωστές απαντήσεις
- K2.3 Κύρια και δευτερεύουσα μνήμη: 84% σωστές απαντήσεις
- K2.4 Η μορφή μιας IP διεύθυνσης: 82% σωστές απαντήσεις
- K2.5 Τιμές ενός byte: 78% σωστές απαντήσεις
- K2.6 Η ταχύτητα του Ariane 5: 70% σωστές απαντήσεις
- K2.7 Άθροισμα θετικών unsigned int: 63% σωστές απαντήσεις
- K2.8 Πλήθος διευθύνσεων IPv4: 57% σωστές απαντήσεις
- K2.9 Πόσους αριθμούς χωράει ένας uint64_t: 57% σωστές απαντήσεις
- K2.10 Πόσος χώρος για ένα κομμάτι IP: 45% σωστές απαντήσεις
- K2.11 Ο μετρητής του Gangnam Style: 41% σωστές απαντήσεις
- K2.12 Πόσο είναι το 2^64: 38% σωστές απαντήσεις
- K2.13 Το μέγεθος του int: 37% σωστές απαντήσεις
- K2.14 Ο κωδικός ASCII του 'J': 33% σωστές απαντήσεις
- K2.15 Το 0xFF σε signed char: 33% σωστές απαντήσεις

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A2.1–A2.8)

- A2.1 Πόσοι χαρακτήρες ASCII υπάρχουν: Διάλεξη 2: Μνήμη και Μεταβλητές, διαφάνεια 17 · ★☆☆ · short-answer · `slides-lec02-ascii-count`
- A2.2 Πόσα διαφορετικά bytes υπάρχουν: Διάλεξη 2: Μνήμη και Μεταβλητές, διαφάνεια 5 · ★☆☆ · short-answer · `slides-lec02-byte-values`
- A2.3 Δεκαεξαδικά ψηφία ενός byte: Διάλεξη 2: Μνήμη και Μεταβλητές, διαφάνεια 10 · ★☆☆ · short-answer · `slides-lec02-hex-digits`
- A2.4 Πόσο μεγάλος είναι ένας int: Διάλεξη 2: Μνήμη και Μεταβλητές, διαφάνεια 27 · ★☆☆ · short-answer · `slides-lec02-int-size`
- A2.5 Ακέραιοι με 32 bit: Διάλεξη 2: Μνήμη και Μεταβλητές, διαφάνεια 18 · ★☆☆ · short-answer · `slides-lec02-int32-count`
- A2.6 Η εντολή για να μάθουμε για ένα πρόγραμμα: Διάλεξη 2: Μνήμη και Μεταβλητές, διαφάνεια 16 · ★☆☆ · tooling · `slides-lec02-man`
- A2.7 Αρνητικοί αριθμοί σε συμπλήρωμα ως προς 2: Διάλεξη 2: Μνήμη και Μεταβλητές, διαφάνεια 21 · ★☆☆ · short-answer · `slides-lec02-twos-complement`
- A2.8 Υπερχείλιση ακεραίων: Διάλεξη 2: Μνήμη και Μεταβλητές, διαφάνεια 26 · ★★☆ · debug · `slides-lec02-overflow`

Εργαστήριο (A2.9–A2.11)

- A2.9 Στοιχεία φοιτητή (Παλιό θέμα): Εργαστήριο 0, Άσκηση 1 · ★☆☆ · programming · `lab-lab00-about`

- **A2.10** Υπερχείλιση ακεραίων: Εργαστήριο 0, Άσκηση 2 · ★☆☆ · short-answer · `lab-lab00-overflow`
- **A2.11** Εκτύπωση χαρακτήρων: Εργαστήριο 4, Άσκηση 1 · ★☆☆ · programming · `lab-lab04-printchar`

Θέματα εξετάσεων (A2.12)

- **A2.12** Rickroll Τριπλέτες: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #4 (Rick Astley Themed), Θέμα 3 · ★★★ · programming · `exam-2023-fall-ex4-q3`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A3.8** Η συνάρτηση about: Εξέταση Σεπτεμβρίου 2024, Θέμα 1 · ★☆☆ · programming · `exam-2024-sep-q1`
- **A3.3** Διακοπή ρεύματος και μνήμη: Διάλεξη 3: Συναρτήσεις, διαφάνεια 3 · ★☆☆ · multiple-choice · `slides-lec03-power-outage`
- **A6.23** Τυπώνοντας τον Πίνακα ASCII: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #15, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex15-q2`
- **A6.16** Μέγιστος Κοινός Διαιρέτης: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #6 (Crypto Themed), Θέμα 1 · ★☆☆ · programming · `exam-2023-fall-ex6-q1`
- **A6.18** Mystery: Εξέταση Σεπτεμβρίου 2025, Θέμα 1 · ★☆☆ · trace · `exam-2025-sep-q1`
- **A6.19** Mystery: Εξέταση Σεπτεμβρίου 2026, Θέμα 1 · ★☆☆ · trace · `exam-2026-sep-q1`
- **A6.14** Η εικασία Collatz: Εργασία 0 (2023-24), Άσκηση 3 · ★★☆ · programming · `hw-2023-hw0-collatz`
- **A6.15** Οι Ακολουθίες Aliquot: Εργασία 0 (2025-26), Άσκηση 3 · ★★☆ · programming · `hw-2025-hw0-aliquot`
- **A8.4** Αρνητικό γινόμενο σε βρόχο: Διάλεξη 8: Ροή Ελέγχου #2, διαφάνεια 2 · ★☆☆ · debug · `slides-lec08-product-overflow`
- **A9.19** Δεκαεξαδικοί: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #1 (Coreutils Themed), Θέμα 1 · ★☆☆ · programming · `exam-2023-fall-ex1-q1`
- **A9.25** ASCII Encoding: Εξέταση Ιουλίου 2024, Θέμα 1 · ★☆☆ · programming · `exam-2024-jul-q1`
- **A9.16** Το Πρόβλημα του Τρόλεϊ (trolley): Εργασία 0 (2024-25), Άσκηση 3 · ★★☆ · programming · `hw-2024-hw0-trolley`
- **A9.11** Ένα απλό κομπιουτεράκι: Εργαστήριο 2, Άσκηση 1 · ★☆☆ · programming · `lab-lab02-calc`
- **A9.12** Διάβασμα ακεραίου με scanf: Εργαστήριο 2, Βήμα 3 · ★☆☆ · short-answer · `lab-lab02-readint`
- **A9.14** Κωδικοποίηση και αποκωδικοποίηση κειμένου: Εργαστήριο 4, Άσκηση 4 · ★★☆ · programming · `lab-lab04-encode-decode`
- **A10.24** Η συνάρτηση paws: Εξέταση Ιουνίου 2026, Θέμα 2 · ★★☆ · debug · `exam-2026-jun-q2`

- **A12.15** Μέση Τιμή Τυχαίων Μεταβλητών - mean: Εξέταση Σεπτεμβρίου 2025, Θέμα 3 · ★☆☆ · programming · `exam-2025-sep-q3`
- **A12.11** Πετυχαίνοντας τον στόχο (Παλιό θέμα): Εργαστήριο 8, Άσκηση 3 · ★★☆ · programming · `lab-lab08-legolas`
- **A14.11** Η συνάρτηση dog: Εξέταση Ιανουαρίου 2025, Θέμα 2 · ★☆☆ · trace · `exam-2025-jan-q2`
- **A15.16** Κατοπτρικά Πρώτα Τετράγωνα: Εργασία 1 (2023-24), Άσκηση 2 · ★★★ · programming · `hw-2023-hw1-mirror`
- **A15.17** Παραγοντοποίηση ημιπρώτων (factor): Εργασία 1 (2024-25), Άσκηση 3 (Bonus) · ★★★ · programming · `hw-2024-hw1-factor`
- **A16.20** Αγαπήσιμοι Αριθμοί: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #0 (Valentine's Themed), Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex0-q4`
- **A16.21** Clyde Πρώτοι: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #11, Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex11-q4`
- **A16.23** Τυχεροί Αριθμοί: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #13, Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex13-q4`
- **A16.24** Οι Καλύτεροι Αριθμοί, Παμψηφεί: Κατατακτήριες Δεκεμβρίου 2024, Θέμα 2 · ★★☆ · programming · `exam-2024-dec-q2`
- **A16.18** Ο Γρίφος του Στέργιου: Bonus #0 (2025-26, προαιρετική) · ★★★ · programming · `hw-2025-bonus0-stergios`
- **A16.16** Σκαλί-σκαλί (Παλιό θέμα, Προαιρετικό): Εργαστήριο 5, Άσκηση 3 · ★★★ · programming · `lab-lab05-ladder`
- **A24.1** Ο τυχερός αριθμός σε δυαδικό: Διάλεξη 24: Προχωρημένα Θέματα, διαφάνεια 32 · ★☆☆ · trace · `slides-lec24-binary-literal`

Κεφάλαιο 3: Συναρτήσεις

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να

- επιλέγετε τον κατάλληλο τύπο ακεραίου ή πραγματικού αριθμού και να γνωρίζετε το εύρος τιμών του·
- γράφετε σταθερές σε δεκαδική, δεκαεξαδική και οκταδική μορφή·
- εξηγείτε τι είναι η υπερχείλιση ακεραίων (integer overflow) και πώς την αποφεύγετε·
- τυπώνετε τιμές με την `printf`, χρησιμοποιώντας ακολουθίες διαφυγής και προσδιοριστικά μορφοποίησης·
- διαβάζετε τα μηνύματα λάθους του `gcc`·
- εξηγείτε κάθε γραμμή του Hello World και τι σημαίνει το exit code ενός προγράμματος·
- ορίζετε και καλείτε δικές σας συναρτήσεις στη C.

Προαπαιτούμενα: Κεφάλαιο 0, Κεφάλαιο 1, Κεφάλαιο 2

Χρόνος μελέτης: ~2 ώρες

Σύνοψη

Η διάλεξη κλείνει πρώτα τους λογαριασμούς με τις μεταβλητές: τους βασικούς τύπους της C και τα εύρη τους, την ανάθεση τιμής, την υπερχείλιση ακεραίων και το τύπωμα τιμών με την `printf`. Έπειτα ξαναπιάνει το Hello World και το αναλύει γραμμή προς γραμμή: σχόλια, `#include`, `printf`, τη συνάρτηση `main` και το `return 0`. Το συμπέρασμα είναι ότι ένα πρόγραμμα C είναι μια σειρά από **συναρτήσεις**, και η εκτέλεση ξεκινά από τη `main`. Γι' αυτό το δεύτερο μισό ορίζει τη συνάρτηση, πρώτα όπως στα μαθηματικά και μετά όπως στη C (τύπος επιστροφής, όνομα, ορίσματα, σώμα, κλήση). Τέλος, με την τεχνική του pair programming, λύνουμε δύο challenges: το Πυθαγόρειο θεώρημα με συναρτήσεις και την προσέγγιση του π με τη σειρά του Leibniz.

Θεωρία

§3.1 Τύποι μεταβλητών

Κάθε μεταβλητή στη C έχει έναν **τύπο** (type), που καθορίζει πόσα bytes πιάνει στη μνήμη και ποιες τιμές μπορεί να πάρει (βλ. Κεφάλαιο 2). Τα μεγέθη δεν είναι ίδια σε κάθε υπολογιστή· ο πίνακας δίνει τα *συνηθισμένα*:

Τύπος	Bytes	Εύρος τιμών	Παράδειγμα τιμής
`char`	1	-128 ... 127	`'B'`, `0x42`
`short int`	2	-32.768 ... 32.767	`42`
`int`	4	-2.147.483.648 ... 2.147.483.647	`42`
`long int`	4	όπως το `int`	`42L`
`float`	4	μικρότερη θετική $1.17 \cdot 10^{-38}$, μεγαλύτερη $3.4 \cdot 10^{38}$	`42.42F`
`double`	8	μικρότερη θετική $2.2 \cdot 10^{-308}$, μεγαλύτερη $1.8 \cdot 10^{308}$	`42.42`
`long double`	8, 10, 12 ή 16		`42.42L`
`unsigned char`	1	0 ... 255	`'B'`, `0x42`
`unsigned short int`	2	0 ... 65535	`42`
`unsigned int`	4	0 ... 4.294.967.295	`42`
`unsigned long int`	4	0 ... 4.294.967.295	`42LU`

Οι τύποι χωρίς πρόσημο (**unsigned**) χρησιμοποιούν όλα τα bits για το μέγεθος: φτάνουν στο διπλάσιο θετικό όριο αλλά δεν έχουν αρνητικούς. Το γράμμα στο τέλος μιας σταθεράς (suffix) δίνει τον τύπο της: `L` για `long` (ή `long double`), `U` για `unsigned`, `F` για `float`· μια σταθερά με υποδιαστολή χωρίς suffix, όπως `42.42`, είναι `double`. Ο `char` είναι κι αυτός ακέραιος: ο χαρακτήρας `'B'` και το `0x42` (66) είναι η ίδια τιμή. Για τον `long int` οι σημειώσεις γράφουν «συνήθως 4, μερικές φορές 8» bytes· στα 64-bit συστήματα Linux είναι 8.

§3.2 Ανάθεση σε μεταβλητή

Ανάθεση (assignment) είναι η εγγραφή μιας τιμής σε μια μεταβλητή. Γίνεται είτε κατά τον ορισμό της (`int x = 42;`) είτε αργότερα (`int y;` και μετά `y = 42;`). Πριν από την ανάθεση, τα 4 bytes της `y` περιέχουν ό,τι είχε μείνει σε εκείνη τη θέση μνήμης: τυχαία bits («σκουπίδια»). Μετά την ανάθεση περιέχουν τη δυαδική αναπαράσταση του 42, δηλαδή `00000000 00000000 00000000 00101010`.

Την ίδια τιμή μπορείτε να τη γράψετε σε διαφορετικές βάσεις. Το πρόθεμα `0x` σημαίνει **δεκαεξαδικό** (`0x2A` $= 2 \cdot 16 + 10 = 42$), ενώ ένα σκέτο `0` μπροστά σημαίνει **οκταδικό** (`052` $= 5 \cdot 8 + 2 = 42$). Άρα `int x = 0x2A;` και `x = 052;` αναθέτουν και τα δύο 42. Προσοχή λοιπόν: το `052` *δεν* είναι το 52.

§3.3 Υπερχείλιση ακεραίων

Ένας `int` των 4 bytes χωράει μέχρι το 2.147.483.647. Η πράξη `2000000000 + 2000000000` γίνεται σε `int`· το σωστό αποτέλεσμα 4.000.000.000 δεν χωράει, και τα bits που μένουν ερμηνεύονται ως

αρνητικός αριθμός: $4.000.000.000 - 2^{32} = -294.967.296$. Αυτό είναι η **υπερχείλιση ακεραίων** (integer overflow): ο αριθμός «τυλίγεται» σαν κοντέρ, π.χ. μετά το 32.767 ενός `short` έρχεται το -32.768. Το πρόγραμμα δεν σταματά, απλώς βγάζει λάθος αποτέλεσμα, γι' αυτό το σφάλμα είναι ύπουλο. Η διόρθωση είναι υπολογισμός σε τύπο που χωράει το αποτέλεσμα, π.χ. `long long` (σταθερές με suffix `LL`) ή, για μη αρνητικά αποτελέσματα, `unsigned`.

§3.4 Ακέραιοι με συγκεκριμένη ακρίβεια

Αφού το μέγεθος ενός `int` είναι «συνήθως» 4 bytes, πώς μπορούμε να είμαστε σίγουροι;

- Ο τελεστής `sizeof` δίνει το μέγεθος ενός τύπου σε bytes, π.χ. `sizeof(int)`. Θα τον δούμε αναλυτικά σε επόμενες διαλέξεις.
- Η βιβλιοθήκη `<stdint.h>` ορίζει ακεραίους με εγγυημένο πλήθος bits: `int8_t`, `uint8_t`, `int16_t`, `uint16_t`, `int32_t`, `uint32_t`, `int64_t`, `uint64_t`. Ο αριθμός είναι τα bits και το `u` σημαίνει unsigned· π.χ. ο `uint64_t` είναι ακέραιος χωρίς πρόσημο 64 bits σε κάθε υπολογιστή.

§3.5 Η συνάρτηση printf

Η `printf()` τυπώνει δεδομένα στο αρχείο εξόδου `stdout` (standard output), δηλαδή κατά κανόνα στο τερματικό. Δέχεται **μεταβλητό πλήθος παραμέτρων**:

1. Η πρώτη είναι το **αλφαριθμητικό μορφοποίησης** (format string): μια ακολουθία χαρακτήρων μέσα σε διπλά εισαγωγικά `" "` που ορίζει *πώς* θα τυπωθούν τα δεδομένα.
2. Οι επόμενες είναι προαιρετικές: οι τιμές που θα τυπωθούν.

Το format string περιέχει τρία είδη πραγμάτων: **απλούς χαρακτήρες**, που τυπώνονται όπως είναι, **ακολουθίες διαφυγής** (escape sequences) και **προσδιοριστικά μορφοποίησης** (format specifiers). Στο `printf("x = %d\n", x);` το `x =` είναι απλοί χαρακτήρες, το `%d` η θέση της τιμής της `x` και το `\n` αλλαγή γραμμής.

§3.6 Ακολουθίες διαφυγής

Μια **ακολουθία διαφυγής** είναι μια ανάστροφη κεκλιμένη `\` (backslash) ακολουθούμενη από έναν χαρακτήρα. Αναπαριστά χαρακτήρες που δεν γράφονται εύκολα μέσα σε εισαγωγικά:

Ακολουθία	Σημασία
`\n`	Αλλαγή γραμμής, σαν το πλήκτρο Enter
`\r`	Επιστροφή του δρομέα (cursor) στην αρχή της τρέχουσας γραμμής
`\t`	Κενό ίσο με ένα tab, σαν το πλήκτρο Tab
`\\`	Τύπωμα της ανάστροφης κεκλιμένης
`\"`	Τύπωμα διπλών εισαγωγικών
`\xNN`	Ο χαρακτήρας με δεκαεξαδικό κωδικό `NN`

Ακολουθία	Σημασία
`\a`	Ηχητικό σήμα

Το `\"` χρειάζεται γιατί ένα σκέτο `"` θα έκλεινε το αλφαριθμητικό. Για τον ίδιο λόγο η `\` γράφεται διπλή.

§3.7 Προσδιοριστικά μορφοποίησης

Ένα **προσδιοριστικό μορφοποίησης** είναι ο χαρακτήρας `%` και ένας ή περισσότεροι χαρακτήρες που λένε πώς να τυπωθεί η αντίστοιχη τιμή: το πρώτο προσδιοριστικό αντιστοιχεί στην πρώτη τιμή μετά το format string, το δεύτερο στη δεύτερη, κ.ο.κ.

Προσδιοριστικό	Σημασία
`%c`	ASCII χαρακτήρας
`%d` ή `%i`	Ακέραιος
`%u`	Ακέραιος unsigned (χωρίς πρόσημο)
`%f`	Πραγματικός (`float`, και `double`)
`%llu`	`long long unsigned int`
`%%`	Ο ίδιος ο χαρακτήρας `%`

Η πλήρης λίστα (πλάτος πεδίου, ακρίβεια δεκαδικών κ.λπ.) είναι στο reference της printf. Το προσδιοριστικό πρέπει να ταιριάζει με τον τύπο της τιμής· αν τυπώσετε `double` με `%d` θα πάρετε σκουπίδια.

§3.8 Συμμετρία και μηνύματα λάθους

Στον προγραμματισμό *συνήθως* υπάρχει συμμετρία: παρενθέσεις, εισαγωγικά και άγκιστρα (curly braces) `{ }` που ανοίγουν πρέπει να κλείνουν. Τα περισσότερα συντακτικά λάθη των πρώτων εβδομάδων είναι ένα «ορφανό» σύμβολο ή ένα ξεχασμένο ερωτηματικό `;`. Όταν ο `gcc` δεν μπορεί να μεταγλωττίσει, τυπώνει ένα μήνυμα της μορφής `αρχείο:γραμμή:στήλη: error: περιγραφή`, και από κάτω τη γραμμή με ένα `^` στο σημείο που μπερδεύτηκε. **Διαβάστε το μήνυμα**: λέει ακριβώς τι περίμενε. Έχετε υπόψη ότι ο μεταγλωττιστής καταλαβαίνει το λάθος εκεί που *δεν* βρίσκει αυτό που περίμενε, οπότε το πραγματικό λάθος είναι συχνά λίγο πιο πριν (π.χ. στο τέλος της προηγούμενης γραμμής).

§3.9 Δηλώσεις πολλών μεταβλητών

Μεταβλητές του ίδιου τύπου μπορούν να δηλωθούν στην ίδια γραμμή, χωρισμένες με κόμμα: `int a; int b; int c;` γράφεται ισοδύναμα `int a, b, c;`. Μπορείτε να αρχικοποιήσετε και μερικές από αυτές: `double a = 3.0, b = 4.0;`.

§3.10 Δεσμευμένες λέξεις

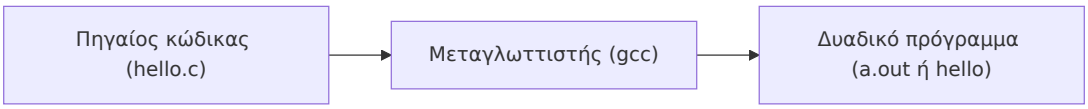
Οι **δεσμευμένες λέξεις** (reserved keywords) έχουν προκαθορισμένο νόημα για τον μεταγλωττιστή και **δεν** επιτρέπεται να χρησιμοποιηθούν ως ονόματα μεταβλητών ή συναρτήσεων:

```
auto      break     case      char      const     continue  default
do        double    else      enum      extern    float     for
goto      if        int       long      register  return    short
signed    sizeof    static    struct    switch    typedef   union
unsigned  void      volatile  while
```

Μια δήλωση όπως `int return;` δεν μεταγλωττίζεται. Τις περισσότερες θα τις συναντήσετε στα επόμενα κεφάλαια.

§3.11 Από τον πηγαίο κώδικα στο εκτελέσιμο

Μεταγλωττιστής (compiler) είναι ένα πρόγραμμα που μετατρέπει εντολές μιας γλώσσας προγραμματισμού σε κώδικα μηχανής, ώστε να μπορεί να τον διαβάσει και να τον εκτελέσει ο υπολογιστής. Στο μάθημα χρησιμοποιούμε τον GNU C Compiler, `gcc`.



Σχήμα: η μεταγλώττιση σε απλοποιημένη μορφή.

Η εικόνα είναι προσεγγιστική· λεπτομέρειες (π.χ. η σύνδεση με βιβλιοθήκες) προστίθενται σε επόμενες διαλέξεις, και μία από αυτές τη βλέπουμε ήδη στο challenge του Πυθαγορείου. Χωρίς επιλογές, ο `gcc hello.c` γράφει το εκτελέσιμο στο αρχείο `a.out`· με `-o hello` διαλέγετε εσείς το όνομα.

§3.12 Η ανατομία του Hello World

```c
/* File: helloworld.c */
#include <stdio.h>

int main() {
  printf("Hello world\n");
  return 0;
}
```

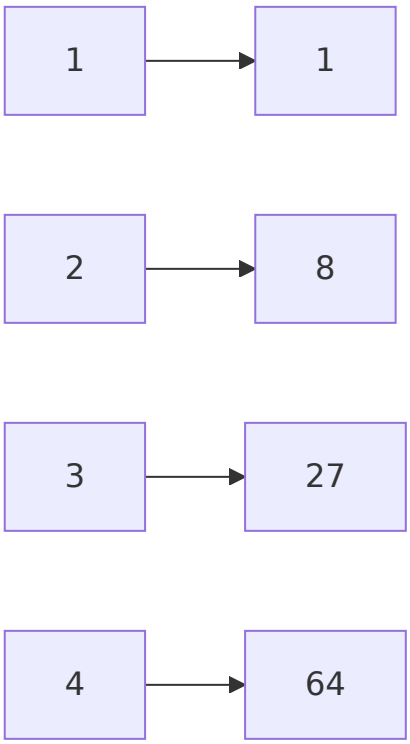
- **Σχόλια (comments)**: κείμενο του προγραμματιστή που συνοδεύει τον κώδικα για να τον κάνει πιο σαφή σε τρίτους ή σε εμάς τους ίδιους, όταν έχει περάσει καιρός. Γράφεται

ανάμεσα σε `/*` και `*/` (και σε πολλές γραμμές), ή σε μία γραμμή μετά από `//`. Ο μεταγλωττιστής τα αγνοεί.

- **Η οδηγία** `#include` (directive): το `#include <stdio.h>` συμπεριλαμβάνει στο πρόγραμμα τα περιεχόμενα του αρχείου `stdio.h` (standard input output), που περιέχει τις δηλώσεις των βασικών συναρτήσεων εξόδου (οθόνη) και εισόδου (πληκτρολόγιο).
- `printf`: συνάρτηση βιβλιοθήκης, δηλωμένη στο `stdio.h` (γι' αυτό κάνουμε `#include`). Τυπώνει το αλφαριθμητικό `"Hello world\n"`· το `\n` αλλάζει γραμμή μετά το μήνυμα.
- **Η συνάρτηση** `main`: ένα πρόγραμμα C αποτελείται από συναρτήσεις. Για να εκτελεστεί, πρέπει να έχει **ακριβώς μία** συνάρτηση `main`, που καλείται πρώτη όταν ξεκινά το πρόγραμμα.
- `return 0;`: επιστρέφει την τιμή της συνάρτησης. Η τιμή που επιστρέφει η `main` είναι και το **exit code** του προγράμματος, που δείχνει αν ολοκληρώθηκε με επιτυχία. Το `0` σημαίνει επιτυχία· κάθε άλλη τιμή σημαίνει ότι το πρόγραμμα **απέτυχε**. Στο shell το βλέπετε με `echo $?` αμέσως μετά την εκτέλεση (βλ. Κεφάλαιο 1).
- **Εντολές (statements)**: οι εντολές μιας συνάρτησης βρίσκονται μέσα σε άγκιστρα `{ }` και η καθεμία τελειώνει με `;` (semicolon, το ελληνικό ερωτηματικό).

§3.13 Συναρτήσεις στα μαθηματικά

Στα μαθηματικά, **συνάρτηση** είναι μια αντιστοίχιση ανάμεσα σε δύο σύνολα, το **σύνολο ορισμού** (domain) και το **σύνολο τιμών** (co-domain), κατά την οποία κάθε στοιχείο του συνόλου ορισμού αντιστοιχίζεται σε **ένα και μόνο** στοιχείο του συνόλου τιμών. Για παράδειγμα η $f(x) = x^3$ με $f : \mathbb{Z} \to \mathbb{Z}$:



Σχήμα: η $f(x) = x^3$ αντιστοιχίζει κάθε στοιχείο του συνόλου ορισμού (αριστερά) σε ακριβώς ένα του συνόλου τιμών (δεξιά).

Μια συνάρτηση μπορεί να έχει πολλές παραμέτρους (**πολυπαραμετρική**): $g(x, y, z) = x^2 + y^2 + z^2 + 42$ με $g : \mathbb{R} \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, ή $h(x, y) = x + y + 1$ με $h : \mathbb{R} \times \mathbb{N} \to \mathbb{R}$. Κάθε παράμετρος έχει το δικό της σύνολο. Οι παράμετροι είναι οι **είσοδοι** και η τιμή της συνάρτησης η **έξοδος**.

Αποτίμηση μιας συνάρτησης είναι ο υπολογισμός της για συγκεκριμένες τιμές: οι είσοδοι 2, 4, 8 μπαίνουν στη g και βγαίνει $g(2, 4, 8) = 2^2 + 4^2 + 8^2 + 42 = 126$.

§3.14 Συναρτήσεις στη C: ορισμός

Στη C, **συνάρτηση** είναι μια σειρά υπολογισμών που παίρνουν τις εισόδους της συνάρτησης και παράγουν την έξοδο. Όπως στα μαθηματικά, κάθε είσοδος και η έξοδος έχουν έναν **τύπο**, δηλαδή το σύνολο τιμών που μπορούν να πάρουν· ο `int`, για παράδειγμα, αντιπροσωπεύει τους ακεραίους. Η g γράφεται:

```
int g(int x, int y, int z) {
  return x * x + y * y + z * z + 42;
}
```

Ο **ορισμός συνάρτησης** (function definition) έχει τρία μέρη:

1. **Τύπος επιστροφής και όνομα**, στη μορφή τύπος όνομα: εδώ `int g`. Όποτε καλούμε

`g(...)` περιμένουμε ακέραιο αποτέλεσμα.

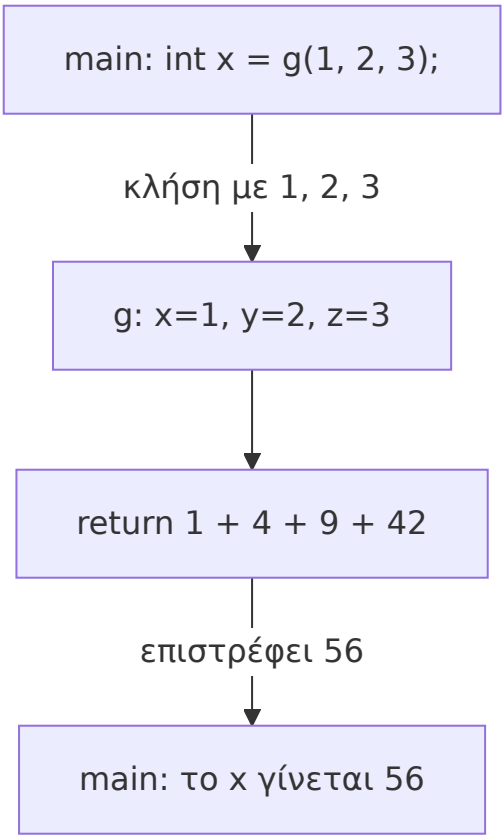
2. **Ορίσματα** (arguments) ή δεδομένα εισόδου, μέσα σε παρενθέσεις, το καθένα επίσης στη μορφή τύπος όνομα και χωρισμένα με κόμμα: `int x, int y, int z` είναι τρία ακέραια ορίσματα με ονόματα `x`, `y`, `z`. Κάθε όρισμα θέλει τον δικό του τύπο (όχι `int x, y, z`).
3. **Σώμα** (body) μέσα σε άγκιστρα `{ }`, με τον υπολογισμό της τιμής. Η εντολή `return` υπολογίζει την παράσταση και **επιστρέφει** το αποτέλεσμα σε όποιον κάλεσε τη συνάρτηση. Οι εντολές του σώματος τελειώνουν με `;`.

Η `main` είναι κι αυτή μια τέτοια συνάρτηση: τύπος επιστροφής `int`, όνομα `main`. Στη διάλεξη εμφανίζεται και με ορίσματα, `int main(int argc, char **argv)`· τι περιέχουν τα `argc` και `argv` θα το δούμε στα ορίσματα γραμμής εντολών.

§3.15 Κλήση συνάρτησης

Η **κλήση συνάρτησης** (function call) γίνεται με το όνομά της και στη συνέχεια τα ορίσματα μέσα σε παρενθέσεις, χωρισμένα με κόμμα. Η κλήση είναι παράσταση: η τιμή της είναι ό,τι επιστρέφει η συνάρτηση, και μπορεί να ανατεθεί σε μεταβλητή ή να δοθεί ως όρισμα σε άλλη συνάρτηση.

```c
int x = g(1, 2, 3);   /* x, y, z της g παίρνουν 1, 2, 3 */
```



Σχήμα: η ροή μιας κλήσης συνάρτησης και της επιστροφής της.

Προσέξτε ότι τα `x` της `main` και `x` της `g` είναι *διαφορετικές* μεταβλητές που απλώς έχουν το ίδιο όνομα· το πώς «βλέπει» κάθε συνάρτηση τις μεταβλητές της είναι θέμα εμβέλειας, που θα το δούμε αργότερα. Η συνάρτηση πρέπει να είναι ορισμένη (πιο πάνω στο αρχείο) πριν την καλέσετε, όπως η `g` πάνω από τη `main`.

§3.16 Pair programming

Το **pair programming** είναι μια τεχνική ανάπτυξης λογισμικού όπου δύο προγραμματιστές δουλεύουν μαζί σε έναν υπολογιστή. Ο ένας, ο **driver**, γράφει κώδικα· ο άλλος, ο **observer** ή **navigator**, ελέγχει (review) κάθε γραμμή καθώς γράφεται. Οι δύο αλλάζουν ρόλους συχνά. Όπως λέει ο Jeff Dean, χρειάζεται να βρείτε κάποιον συμβατό με τον τρόπο σκέψης σας, ώστε μαζί να αποτελείτε μια συμπληρωματική δύναμη. Τα δύο challenges της διάλεξης λύθηκαν έτσι, σε ζευγάρια.

Παραδείγματα

§3.17 Μεταγλώττιση και εκτέλεση του Hello World

Εφαρμόζει: «Από τον πηγαίο κώδικα στο εκτελέσιμο», «Η ανατομία του Hello World».

```
$ gcc hello.c
$ ls
a.out  hello.c
$ ./a.out
Hello world
$ file a.out
a.out: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically
linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=52fa5999c10d767a
5ff30f662346478333de74bf, for GNU/Linux 3.2.0, not stripped
$ gcc -o hello hello.c
$ ./hello
Hello world
```

Η εντολή `file` επιβεβαιώνει ότι το `a.out` είναι δυαδικό εκτελέσιμο (ELF) για επεξεργαστή x86-64, όχι κείμενο. Με `-o hello` το εκτελέσιμο παίρνει όνομα της επιλογής μας.

§3.18 Ένα ξεχασμένο ερωτηματικό

Εφαρμόζει: «Συμμετρία και μηνύματα λάθους». Αν λείπει το `;` μετά την `printf`:

```
$ gcc -o hello hello.c
hello.c: In function 'main':
```

```
hello.c:4:26: error: expected ';' before 'return'
    4 |   printf("Hello world\n")
      |                          ^
      |                          ;
    5 |   return 0;
      |   ~~~~~~
```

Το μήνυμα λέει ότι στη γραμμή 4, στήλη 26, ο μεταγλωττιστής περίμενε `;` πριν το `return`, και προτείνει μάλιστα πού να το βάλετε. Το λάθος εντοπίζεται όταν ο μεταγλωττιστής συναντά το `return` της γραμμής 5, αλλά αναφέρεται στο τέλος της γραμμής 4.

§3.19 Υπερχείλιση στην πράξη

Εφαρμόζει: «Υπερχείλιση ακεραίων», «Προσδιοριστικά μορφοποίησης».

```c
#include <stdio.h>

int main() {
  printf("%d\n", 2000000000 + 2000000000);
  return 0;
}
```

```
$ ./overflow
-294967296
```

Και οι δύο σταθερές είναι `int`, άρα και η πρόσθεση γίνεται σε `int` και υπερχειλίζει (ο `gcc` μάλιστα προειδοποιεί για integer overflow κατά τη μεταγλώττιση). Για να πάρετε 4000000000, κάντε την πράξη σε μεγαλύτερο τύπο και τυπώστε με το αντίστοιχο προσδιοριστικό, π.χ. `printf("%lld\n", 2000000000LL + 2000000000LL);`.

§3.20 Challenge #1: Πυθαγόρειο θεώρημα (pyth.c)

Εφαρμόζει: «Συναρτήσεις στη C: ορισμός», «Κλήση συνάρτησης», «Η συνάρτηση printf». Ζητείται πρόγραμμα που, με δεδομένα τα μήκη των καθέτων πλευρών ενός ορθογωνίου τριγώνου, τυπώνει το εμβαδόν και την περίμετρό του. Η λύση της διάλεξης βάζει κάθε υπολογισμό σε δική του συνάρτηση:

```c
#include <stdio.h>
#include <math.h>
/* Returns the area of a right triangle */
double compute_area(double a, double b) {
  return (a * b) / 2.0;
}
/* Returns the perimeter of a right triangle */
double compute_perimeter(double a, double b) {
  double c = sqrt(a * a + b * b);  // hypotenuse
```

```
  double perimeter = a + b + c;
  return perimeter;
}

int main(int argc, char **argv) {
  double a = 3.0, b = 4.0;
  printf("Area of triangle: %f\n", compute_area(a, b));
  printf("Triangle perimeter: %f\n", compute_perimeter(a, b));
  return 0;
}
```

```
$ gcc -o pyth pyth.c
/usr/bin/X11/ld: /tmp/cc9jUwzF.o: in function `compute_perimeter':
pyth.c:(.text+0x5b): undefined reference to `sqrt'
collect2: error: ld returned 1 exit status
$ gcc -o pyth pyth.c -lm
$ ./pyth
Area of triangle: 6.000000
Triangle perimeter: 12.000000
```

Η υποτείνουσα βγαίνει από το Πυθαγόρειο θεώρημα, $c = \sqrt{a^2 + b^2}$, με τη `sqrt` της μαθηματικής βιβλιοθήκης. Η πρώτη μεταγλώττιση αποτυγχάνει με `undefined reference to 'sqrt'`: το λάθος δεν το βγάζει ο μεταγλωττιστής αλλά ο **συνδέτης** (linker, `ld`). Το `#include <math.h>` απλώς *δηλώνει* την `sqrt`· ο κώδικάς της βρίσκεται στη μαθηματική βιβλιοθήκη, που ζητιέται ρητά με `-lm`. Αυτή είναι μία από τις «λεπτομέρειες» που λείπουν από το απλό σχήμα της μεταγλώττισης. Η `main` καλεί τις συναρτήσεις μέσα στις κλήσεις της `printf`, που τυπώνει την τιμή τους με `%f` (6 δεκαδικά). Εδώ τα 3 και 4 είναι γραμμένα στο πρόγραμμα· η ίδια άσκηση στο Εργαστήριο 2 (`pyth.c`) ζητά να διαβάζονται ως είσοδος.

§3.21 Challenge #2: Προσέγγιση του π

Εφαρμόζει: «Τύποι μεταβλητών», «Η συνάρτηση printf». Ο Leibniz έδειξε ότι

$$\frac{\pi}{4} = \sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots$$

Μπορούμε να γράψουμε πρόγραμμα που προσεγγίζει το π; Αθροίζουμε πεπερασμένο πλήθος όρων (εδώ 100000). Αντί να υπολογίζουμε το $(-1)^i$ με δύναμη, κρατάμε αριθμητή που αλλάζει πρόσημο και παρονομαστή που αυξάνει κατά 2 σε κάθε βήμα:

```
#include <stdio.h>

/* A simple way to approximate pi */
int main() {
```

```
  int i;
  double numerator = 1;
  double denominator = 1;
  double sum = 0;
  for (i = 0; i < 100000; i++) {
    sum += numerator / denominator;
    numerator = numerator * (-1);
    denominator = denominator + 2;
  }
  printf("Pi is approximately: %f\n", 4 * sum);
}
```

```
$ gcc -o pi pi.c
$ ./pi
Pi is approximately: 3.141583
```

Ο βρόχος `for` επαναλαμβάνει το σώμα του 100000 φορές (τους βρόχους τους βλέπουμε αναλυτικά στο Κεφάλαιο 6). Οι μεταβλητές είναι `double`, ώστε η διαίρεση `numerator / denominator` να δίνει πραγματικό αποτέλεσμα. Με 100000 όρους το αποτέλεσμα είναι σωστό σε 4 δεκαδικά (π = 3.14159…): η σειρά συγκλίνει αργά.

Κύρια σημεία

1. Ένα πρόγραμμα C είναι μια σειρά από συναρτήσεις.
2. Η εκτέλεση του προγράμματος ξεκινά από τη συνάρτηση `main`, που πρέπει να υπάρχει ακριβώς μία φορά.
3. Κάθε τύπος έχει συγκεκριμένο μέγεθος και εύρος τιμών· τα μεγέθη είναι «συνήθη», όχι εγγυημένα, και για εγγυημένο μέγεθος υπάρχουν οι τύποι του `<stdint.h>`.
4. Ένας υπολογισμός που ξεπερνά το εύρος του τύπου του υπερχειλίζει σιωπηλά και δίνει λάθος αποτέλεσμα· η λύση είναι μεγαλύτερος τύπος.
5. Οι σταθερές γράφονται και σε δεκαεξαδικό (`0x2A`) ή οκταδικό (`052`)· ένα αρχικό `0` αλλάζει τη βάση.
6. Η `printf` τυπώνει στο `stdout` με βάση ένα format string που περιέχει απλούς χαρακτήρες, ακολουθίες διαφυγής (`\n`, `\t`, …) και προσδιοριστικά (`%d`, `%f`, …).
7. Τα μηνύματα λάθους του `gcc` δίνουν αρχείο, γραμμή και στήλη· διαβάστε τα, και ψάξτε και λίγο πριν από το σημείο που δείχνουν.
8. Η τιμή που επιστρέφει η `main` είναι το exit code: `0` σημαίνει επιτυχία, οτιδήποτε άλλο αποτυχία, και το βλέπετε με `echo $?`.
9. Στα μαθηματικά, συνάρτηση είναι αντιστοίχιση όπου κάθε στοιχείο του συνόλου ορισμού αντιστοιχεί σε ένα και μόνο στοιχείο του συνόλου τιμών.
10. Στη C, ο ορισμός συνάρτησης έχει τύπο επιστροφής και όνομα, ορίσματα με τύπους σε παρενθέσεις, και σώμα σε `{ }` που επιστρέφει την τιμή με `return`.
11. Μια συνάρτηση καλείται με το όνομά της και τα ορίσματα σε παρενθέσεις, χωρισμένα με

κόμμα· η κλήση έχει ως τιμή ό,τι επιστρέφει η συνάρτηση.

12. Οι συναρτήσεις της `math.h`, όπως η `sqrt`, χρειάζονται `-lm` κατά τη μεταγλώττιση.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
τύπος	type	Το σύνολο τιμών και το μέγεθος μιας μεταβλητής
ανάθεση	assignment	Εγγραφή τιμής σε μεταβλητή
υπερχείλιση ακεραίων	integer overflow	Αποτέλεσμα εκτός εύρους του τύπου, που «τυλίγεται»
αλφαριθμητικό μορφοποίησης	format string	Το πρώτο όρισμα της `printf`
ακολουθία διαφυγής	escape sequence	`\` και ένας χαρακτήρας, π.χ. `\n`
προσδιοριστικό μορφοποίησης	format specifier	`%` και χαρακτήρες, π.χ. `%d`
δεσμευμένη λέξη	reserved keyword	Λέξη με προκαθορισμένο νόημα, όχι για ονόματα
μεταγλωττιστής	compiler	Μετατρέπει πηγαίο κώδικα σε κώδικα μηχανής
συνδέτης	linker	Συνδέει τον κώδικά μας με βιβλιοθήκες (π.χ. `-lm`)
οδηγία	directive	Γραμμή με `#`, π.χ. `#include`
κωδικός εξόδου	exit code	Η τιμή που επιστρέφει η `main`
συνάρτηση	function	Υπολογισμός από εισόδους σε έξοδο, με όνομα
σύνολο ορισμού / τιμών	domain / co-domain	Τα σύνολα εισόδων και εξόδων
ορισμός συνάρτησης	function definition	Τύπος, όνομα, ορίσματα και σώμα
όρισμα	argument	Δεδομένο εισόδου μιας συνάρτησης
κλήση συνάρτησης	function call	Εκτέλεση της συνάρτησης με συγκεκριμένα ορίσματα
προγραμματισμός σε ζεύγη	pair programming	Driver γράφει, navigator ελέγχει, αλλάζουν ρόλους

Διάβασμα

- **Διαφάνειες**: Διάλεξη 3, σελ. 1–47. Τύποι, ανάθεση, υπερχείλιση: σελ. 5–9· `printf`, ακολουθίες διαφυγής, προσδιοριστικά: σελ. 10–12· συμμετρία και μηνύματα λάθους: σελ. 13–14· δηλώσεις και δεσμευμένες λέξεις: σελ. 15–16· μεταγλώττιση και ανάλυση του Hello

World: σελ. 17–27· συναρτήσεις: σελ. 28–37· pair programming και challenges: σελ. 38–44.

- **Σημειώσεις**: οι διαφάνειες συνιστούν να έχετε καλύψει τις σημειώσεις του κ. Σταματόπουλου μέχρι τη σελίδα 35, και τις σελίδες 58–71. Συγκεκριμένα για αυτή τη διάλεξη:
  - Κεφάλαιο 1: Πρώτα προγράμματα σε C, ενότητες «Καλημέρα κόσμε της C» και «Πόσο είναι το π;» (K04, σελ. 19–23)
  - Κεφάλαιο 2: Μεταβλητές, τύποι, τελεστές και παραστάσεις, ενότητα «Μεταβλητές, σταθερές, τύποι και δηλώσεις στην C» (K04, σελ. 30–33)
  - Κεφάλαιο 4: Συναρτήσεις, εμβέλεια και αναδρομή, ενότητες «Δομή ενός προγράμματος C – Συναρτήσεις», «Συνάρτηση ύψωσης σε δύναμη», «Συνάρτηση υπολογισμού παραγοντικού» (K04, σελ. 58–62)· οι σελ. 63–69 («Εμβέλεια και χρόνος ζωής μεταβλητών», «Υπολογισμός παραγοντικού με αναδρομή») προετοιμάζουν επόμενα κεφάλαια.
- **Εργαστήριο**:
  - Εργαστήριο 2: άσκηση `pyth.c` (το Challenge #1 με είσοδο από τον χρήστη) και το παράρτημα «Αποσφαλμάτωση προγραμμάτων (Πράξη 1η)» για τα συντακτικά λάθη
  - Εργαστήριο 5: άσκηση `collatz.c`, ερωτήματα 1.1–1.2 (συναρτήσεις `isodd` και `collatz_it`)
- **Άλλα**:
  - Wikipedia: Integer overflow, Escape sequences in C, GNU Compiler Collection
  - `printf`: tips και reference
  - Ubuntu: The Linux command line for beginners
  - Pair programming: Wikipedia Software development, Code review· Jeff Dean quotes

Συχνά λάθη

- **Ξεχασμένο `;`**. `printf("Hello world\n")` χωρίς `;` δίνει `error: expected ';' before 'return'`. Διόρθωση: βάλτε `;` στο τέλος της εντολής που δείχνει το ^.
- **Ορφανά εισαγωγικά ή άγκιστρα**. `printf("Hello world!\n);` δίνει `missing terminating " character`· ένα `}` που λείπει δίνει σφάλμα στο τέλος του αρχείου (`expected declaration or statement at end of input`). Διόρθωση: ελέγξτε τη συμμετρία.
- **Υπερχείλιση**. Γινόμενα ή αθροίσματα μεγάλων `int` βγάζουν αρνητικούς ή παράλογους αριθμούς. Διόρθωση: `long long` ή `unsigned` και το αντίστοιχο προσδιοριστικό.
- **Αρχικό μηδέν**. `int x = 052;` δίνει 42, όχι 52: είναι οκταδικό.
- **Λάθος προσδιοριστικό**. `printf("%d\n", 3.5);` τυπώνει σκουπίδια (ο `gcc -Wall` προειδοποιεί `format '%d' expects argument of type 'int'`). Διόρθωση: `%f` για πραγματικούς.
- **Χωρίς `-lm`**. `undefined reference to 'sqrt'` από τον `ld`. Διόρθωση: `gcc -o pyth pyth.c -lm`.
- **Κοινός τύπος σε ορίσματα**. `int g(int x, y, z)` δεν μεταγλωττίζεται· κάθε όρισμα

θέλει δικό του τύπο: `int g(int x, int y, int z)`.
- **Δεσμευμένη λέξη ως όνομα**. `int for = 3;` ή `double double;` δεν μεταγλωττίζονται.
- **Χρήση μεταβλητής πριν την ανάθεση**. `int y; printf("%d\n", y);` τυπώνει ό,τι σκουπίδι υπάρχει στη μνήμη.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K3.8** Ο τύπος επιστροφής μιας συνάρτησης (28% σωστές): Το 25% επέλεξε `int`, μπερδεύοντας τον τύπο της παραμέτρου `x` με τον τύπο επιστροφής· η τιμή του `x * x` μετατρέπεται σε `double` κατά την επιστροφή.
- **K3.5** printf με %c και 42 (48% σωστές): Το 24% επέλεξε 42, σαν το %c να τυπώνει τον αριθμό· το %c τυπώνει τον χαρακτήρα με αυτόν τον κωδικό ASCII.
- **K3.3** Από τι αποτελείται ένα πρόγραμμα C (55% σωστές): Το 30% επέλεξε «μια σειρά από εντολές»· οι εντολές όμως υπάρχουν μόνο μέσα στο σώμα συναρτήσεων, και το πρόγραμμα οργανώνεται σε συναρτήσεις, με πρώτη τη `main`.

Ερωτήσεις κατανόησης

- **E3.1** Γιατί ο `unsigned char` φτάνει μέχρι το 255 ενώ ο `char` μέχρι το 127;[28]
- **E3.2** Ποια τιμή έχει η `x` μετά το `int x = 0x10;` και ποια μετά το `x = 010;`;[29]
- **E3.3** Τι θα τυπώσει το `printf("100%%\tOK\n");`;[30]
- **E3.4** Ποια τέσσερα πράγματα βλέπετε σε ένα μήνυμα λάθους του `gcc` όπως το `hello.c:4:26: error: ...`;[31]
- **E3.5** Πώς ξέρει το shell αν ένα πρόγραμμα C πέτυχε, και πώς το ελέγχετε εσείς;[32]
- **E3.6** Ονομάστε τα τρία μέρη του ορισμού `double compute_area(double a, double b) { ... }`.[33]
- **E3.7** Τι επιστρέφει η κλήση `g(0, 0, 0)` της συνάρτησης `g` της διάλεξης;[34]
- **E3.8** Γιατί το `#include <math.h>` δεν αρκεί για να χρησιμοποιήσετε την `sqrt`;[35]

[28] Ο `char` αφιερώνει ένα από τα 8 bits στο πρόσημο (εύρος -128…127), ενώ ο `unsigned char` τα χρησιμοποιεί όλα για το μέγεθος (0…255).

[29] 16 (δεκαεξαδικό 10) και 8 (οκταδικό 10).

[30] `100%`, ένα tab, `OK` και αλλαγή γραμμής· το `%%` τυπώνει ένα `%`.

[31] Το αρχείο (`hello.c`), τη γραμμή (4), τη στήλη (26) και την περιγραφή του λάθους.

[32] Από το exit code, δηλαδή την τιμή που επιστρέφει η `main` (`0` = επιτυχία). Το βλέπετε με `echo $?` αμέσως μετά την εκτέλεση.

[33] Τύπος επιστροφής και όνομα (`double compute_area`), ορίσματα (`double a, double b`), σώμα (`{ ... }`).

[34] 42, αφού $0 + 0 + 0 + 42 = 42$.

[35] Το `math.h` περιέχει μόνο τη δήλωση της `sqrt`. Ο κώδικάς της είναι στη μαθηματική βιβλιοθήκη, που πρέπει να συνδεθεί με `-lm`.

Kahoot από το αμφιθέατρο (K3.1–K3.8)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K3.1** Προσδιοριστικά %d με σειρά: 87% σωστές απαντήσεις
- **K3.2** Τιμή επιστροφής double: 58% σωστές απαντήσεις
- **K3.3** Από τι αποτελείται ένα πρόγραμμα C: 55% σωστές απαντήσεις
- **K3.4** Η ακολουθία `\r`: 48% σωστές απαντήσεις
- **K3.5** printf με %c και 42: 48% σωστές απαντήσεις
- **K3.6** Ακολουθίες διαφυγής για `\\` και `"`: 43% σωστές απαντήσεις
- **K3.7** Τα μέρη του ορισμού συνάρτησης: 31% σωστές απαντήσεις
- **K3.8** Ο τύπος επιστροφής μιας συνάρτησης: 28% σωστές απαντήσεις

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A3.1–A3.6)

- **A3.1** Τιμή μετά από κλήση συνάρτησης: Διάλεξη 3: Συναρτήσεις, διαφάνεια 37 · ★☆☆ · trace · `slides-lec03-function-call`
- **A3.2** Πόσα bytes είναι ένας int;: Διάλεξη 3: Συναρτήσεις, διαφάνεια 9 · ★☆☆ · short-answer · `slides-lec03-int-size`
- **A3.3** Διακοπή ρεύματος και μνήμη: Διάλεξη 3: Συναρτήσεις, διαφάνεια 3 · ★☆☆ · multiple-choice · `slides-lec03-power-outage`
- **A3.4** Πυθαγόρειο θεώρημα (pyth.c): Διάλεξη 3: Συναρτήσεις, διαφάνεια 40 · ★☆☆ · programming · `slides-lec03-pyth`
- **A3.5** Προσέγγιση του π με τη σειρά Leibniz: Διάλεξη 3: Συναρτήσεις, διαφάνεια 43 · ★★☆ · programming · `slides-lec03-leibniz-pi`
- **A3.6** Υπερχείλιση ακεραίων: Διάλεξη 3: Συναρτήσεις, διαφάνεια 8 · ★★☆ · trace · `slides-lec03-overflow`

Εργαστήριο (A3.7)

- **A3.7** Πυθαγόρειο θεώρημα: Εργαστήριο 2, Άσκηση 2 · ★☆☆ · programming · `lab-lab02-pyth`

Θέματα εξετάσεων (A3.8)

- **A3.8** Η συνάρτηση about: Εξέταση Σεπτεμβρίου 2024, Θέμα 1 · ★☆☆ · programming · `exam-2024-sep-q1`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A4.4** Υπολογισμός βαθμολογίας πρωτοετών: Διάλεξη 4: Git και Τελεστές, διαφάνεια 8 · ★☆☆ · programming · `slides-lec04-grade-program`
- **A4.5** Συνάρτηση max με τον τελεστή συνθήκης: Διάλεξη 4: Git και Τελεστές, διαφάνεια 31 · ★☆☆ · programming · `slides-lec04-max-conditional`
- **A4.8** Τελεστές ως συναρτήσεις: Διάλεξη 4: Git και Τελεστές, διαφάνειες 21 και 25 · ★☆☆ · short-answer · `slides-lec04-operators-as-functions`
- **A5.7** Συνάρτηση max με τον τελεστή συνθήκης: Διάλεξη 5: Τελεστές και Εντολές, διαφάνεια 13 · ★☆☆ · programming · `slides-lec05-ternary-max`
- **A6.20** Κάντο όπως ο Βιετά: Κατατακτήριες Δεκεμβρίου 2023, Θέμα 1 · ★★☆ · programming · `exam-2023-dec-q1`
- **A6.21** Ασημένιο Κλάσμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #10, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex10-q2`
- **A6.22** Κάντο όπως ο Βιετά: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #13, Θέμα 1 · ★★☆ · programming · `exam-2023-fall-ex13-q1`
- **A6.17** Προσεγγίζοντας το π: Κατατακτήριες Δεκεμβρίου 2024, Θέμα 1 · ★☆☆ · programming · `exam-2024-dec-q1`
- **A6.13** Υπολογισμός αθροίσματος σειράς: Εργαστήριο 3, Άσκηση 1 · ★★☆ · programming · `lab-lab03-seq`
- **A8.7** Η Μέθοδος Newton-Raphson: Εργασία 1 (2023-24), Άσκηση 1 · ★★☆ · programming · `hw-2023-hw1-newton`
- **A11.15** Η εικασία του Collatz: Εργαστήριο 5, Άσκηση 1 · ★★☆ · programming · `lab-lab05-collatz`
- **A11.14** Πέρασμα δεδομένων μέσω δεικτών: Εργαστήριο 6, Άσκηση 1 · ★☆☆ · programming · `lab-lab06-myprog`
- **A14.2** Αύξηση παραμέτρου μέσα σε συνάρτηση: Διάλεξη 14, διαφάνειες 11–12 · ★☆☆ · trace · `slides-lec14-local-param`
- **A15.16** Κατοπτρικά Πρώτα Τετράγωνα: Εργασία 1 (2023-24), Άσκηση 2 · ★★★ · programming · `hw-2023-hw1-mirror`
- **A23.1** Κλήση πριν από τον ορισμό: Διάλεξη 23, διαφάνειες 19-21 · ★☆☆ · debug · `slides-lec23-implicit-declaration`
- **A26.4** Compiler error ή linking error; (math.h και libm.so): How to Make? (προσκεκλημένη διάλεξη), διαφάνειες 6-8 · ★☆☆ · debug · `slides-lecmake-sqrt-errors`

Μέρος Α: Εργαλεία και πρώτα βήματα

Κεφάλαιο 4: Git και Τελεστές

> **Στόχοι**: μετά από αυτό το κεφάλαιο θα μπορείτε να εξηγείτε τι είναι το pair programming και το version control, να κάνετε τον κύκλο `clone`, `add`, `commit`, `push` του git, να κατατάσσετε και να υπολογίζετε κάθε είδος τελεστή της C, και να προβλέπετε τη σειρά υπολογισμού από την προτεραιότητα και την προσεταιριστικότητα.
>
> **Προαπαιτούμενα**: Κεφάλαιο 1, Κεφάλαιο 2, Κεφάλαιο 3
>
> **Χρόνος μελέτης**: ~2,5 ώρες

Σύνοψη

Η διάλεξη έχει τρία μέρη. Ξεκινάει με live coding σε μορφή pair programming: γράφουμε μαζί ένα πρόγραμμα που υπολογίζει τον βαθμό του μαθήματος από τα ορίσματα της γραμμής εντολών. Μετά γνωρίζουμε το **git**, το σύστημα version control με το οποίο θα γράφετε και θα υποβάλλετε όλες τις εργασίες του μαθήματος, και τον κύκλο των τεσσάρων εντολών του. Το μεγαλύτερο μέρος αφορά τους **τελεστές** της C: τι είναι τελεστής και τελεστέος, πώς κατατάσσονται, τι κάνει ο καθένας, και με ποια σειρά υπολογίζονται σε μια παράσταση. Ένα λάθος στη σειρά ή στον τύπο τους (π.χ. ακέραια διαίρεση) δίνει σιωπηλά λάθος αποτέλεσμα.

Θεωρία

§4.1 Pair programming

Pair programming είναι μια τεχνική ανάπτυξης λογισμικού στην οποία δύο προγραμματιστές δουλεύουν μαζί σε έναν υπολογιστή. Ο ένας, ο **driver**, γράφει τον κώδικα· ο άλλος, ο **observer** ή **navigator**, ελέγχει (review) κάθε γραμμή τη στιγμή που γράφεται. Οι δύο αλλάζουν ρόλους συχνά. Το όφελος είναι ότι τα λάθη πιάνονται αμέσως και ότι δύο διαφορετικοί τρόποι σκέψης συμπληρώνουν ο ένας τον άλλο. Όπως λέει ο Jeff Dean, χρειάζεται να βρείτε κάποιον «συμβατό με τον τρόπο που σκέφτεστε, ώστε οι δύο μαζί να είστε μια συμπληρωματική δύναμη». Στη διάλεξη 2–3 εθελοντές έγραψαν έτσι το πρόγραμμα βαθμολογίας (βλ. Παραδείγματα).

§4.2 Ορίσματα της main: argc, argv και atoi

Η `main` μπορεί να δεχτεί ορίσματα από τη γραμμή εντολών: `int main(int argc, char **argv)`. Η παράμετρος `argc` λέει πόσα ορίσματα περάσαμε στο πρόγραμμα, και η `argv` περιέχει το

κείμενο κάθε ορίσματος: το `argv[1]` είναι το 1ο όρισμα, το `argv[2]` το 2ο, κ.ο.κ. Το `argv[0]` είναι το ίδιο το όνομα του προγράμματος, γι' αυτό το `./grade 70 80 100` έχει `argc == 4`.

Τα ορίσματα είναι κείμενο, όχι αριθμοί. Η συνάρτηση βιβλιοθήκης `atoi` (από το `stdlib.h`) μετατρέπει ένα αλφαριθμητικό (ASCII) σε ακέραιο (integer): `atoi("70")` δίνει `70`. Αν σας ενοχλεί πόσο «μαγική» φαίνεται η `main` με τα ορίσματά της, μην ανησυχείτε: είναι από τις δυσκολότερες συναρτήσεις της C και θα χτίσουμε σταδιακά το υπόβαθρο (πίνακες, δείκτες, συμβολοσειρές) για να γίνει κατανοητή.

§4.3 Version control

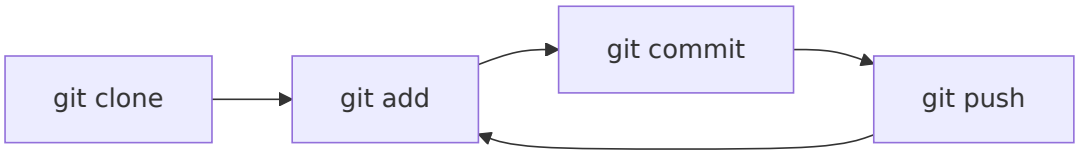
Όποιος έχει φακέλους με αρχεία `final.docx`, `final-final.docx`, `final-final-final.docx` ξέρει το πρόβλημα: η «τελική» μορφή δεν είναι ποτέ τελική. **Version control** είναι ένα σύστημα που παρακολουθεί όλες τις αλλαγές που γίνονται στον κώδικα και κρατάει ιστορικό αρχείο τους. Το θέλουμε γιατί:

- τα περισσότερα προγράμματα αλλάζουν με τον χρόνο· δεν υπάρχει «τελική μορφή»·
- κάνουμε αλλαγές και κρατάμε μόνο τα αρχεία που χρειαζόμαστε, χωρίς να χάνουμε πληροφορία·
- μπορούμε να ανατρέξουμε σε οποιαδήποτε αλλαγή έγινε, ανά πάσα στιγμή.

§4.4 Git και GitHub

Το **git** είναι το δημοφιλέστερο πρόγραμμα version control σήμερα (το χρησιμοποιεί περίπου το 90% των χρηστών/project). Την πρώτη του έκδοση την έγραψε ο Linus Torvalds το 2005. Το **GitHub** (github.com) είναι μια δημοφιλής και (για την ώρα) δωρεάν πλατφόρμα όπου αποθηκεύουμε τον κώδικά μας. Στο μάθημα θα τα χρησιμοποιήσουμε για τη γραφή **και την υποβολή** των εργασιών σας.

Ένα **repository (αποθετήριο)** είναι ένας φάκελος με αρχεία μαζί με όλο το ιστορικό τους. Υπάρχει ένα αντίγραφο στο GitHub (το **απομακρυσμένο**, remote, στον server) και ένα στον υπολογιστή σας (το **τοπικό**). Σε επίπεδο βασικής χρήσης, όση χρειάζεται το μάθημα, η δουλειά με το git είναι τέσσερα βήματα:



Σχήμα: ο κύκλος ανάπτυξης με git: ένα `clone` *στην αρχή, μετά επαναλαμβάνουμε* `add`, `commit`, `push`.

- `git clone` αντιγράφει το repository με όλα τα αρχεία σε έναν τοπικό φάκελο, που δημιουργείται αν δεν υπάρχει. Π.χ. το `git clone git@github.com:progintro/hw0-barbouni-2005.git` φτιάχνει τον φάκελο `hw0-barbouni-2005`. Γίνεται μία φορά ανά

repository.
- `git add` προσθέτει αρχεία στο **staging area** (προσωρινή λίστα) για μελλοντική αποθήκευση στο repository. Ένα καινούργιο αρχείο που δεν έχει γίνει ακόμα add είναι **untracked**: το git το βλέπει αλλά δεν το παρακολουθεί.
- `git commit` αποθηκεύει (καταχωρεί) τις αλλαγές του staging area στο **τοπικό** repository, μαζί με ένα μήνυμα που περιγράφει την αλλαγή. Κάθε commit είναι ένα σημείο του ιστορικού στο οποίο μπορείτε να επιστρέψετε.
- `git push` ανεβάζει τα τοπικά commits στον απομακρυσμένο server. Μέχρι να γίνει push, οι αλλαγές σας υπάρχουν μόνο στον υπολογιστή σας.
- `git pull` κατεβάζει αλλαγές από τον server (αν υπάρχουν). Είναι χρήσιμο όταν δουλεύουν πολλοί προγραμματιστές ή όταν δουλεύετε από πολλούς υπολογιστές, ώστε να μη χρειάζεται νέο `clone` κάθε φορά.
- `git status` δεν αλλάζει τίποτα· δείχνει αν υπάρχουν αλλαγές, ποια αρχεία είναι untracked και ποια είναι έτοιμα για commit. Τρέξτε το μετά από κάθε βήμα.

Η σειρά έχει σημασία: αν ξεχάσετε το `git add`, τα `commit` και `push` «απογειώνονται» χωρίς τα αρχεία σας. Για να δουλέψουν τα `clone`/`push` με διεύθυνση `git@github.com:...`, πρέπει να έχετε φτιάξει κλειδί SSH για τον λογαριασμό σας στο GitHub και να έχετε ρυθμίσει το git, όπως περιγράφει το Φυλλάδιο 1 του εργαστηρίου (Εργαστήριο 1, Βήμα 6).

§4.5 Τελεστής και τελεστέος

Οι **τελεστές (operators)** της C είναι σύμβολα που κάνουν μαθηματικούς, συγκριτικούς, δυαδικούς (bit), υποθετικούς ή λογικούς υπολογισμούς. Οι **τελεστέοι (operands)** είναι τα αντικείμενα (μεταβλητές, σταθερές) πάνω στα οποία δουλεύουν. Στο `x + 42` ο τελεστής είναι το `+` (πρόσθεση), ο 1ος τελεστέος η μεταβλητή `x` και ο 2ος η σταθερά `42`. Κάθε τελεστής, όπως και μια συνάρτηση, **επιστρέφει μια τιμή**.

Ο τελεστής θυμίζει συνάρτηση και οι τελεστέοι τα ορίσματά της (Κεφάλαιο 3): το πρόγραμμα παίρνει δεδομένα εισόδου και υπολογίζει μια έξοδο, αποτελείται από συναρτήσεις που παίρνουν ορίσματα και υπολογίζουν μια έξοδο, και οι συναρτήσεις χρησιμοποιούν τελεστές και τελεστέους για να υπολογίσουν μια έξοδο («it's turtles all the way down»). Αν ο `+` για ακεραίους ήταν συνάρτηση, θα είχε πρωτότυπο κάτι σαν `int plus(int a, int b)`, και ο `>` κάτι σαν `int greater(int a, int b)`.

§4.6 Κατηγοριοποίηση των τελεστών

Με βάση το **arity**, δηλαδή πόσους τελεστέους έχουν, οι τελεστές είναι:

- **μοναδιαίοι (unary)**, με έναν τελεστέο: `!x`, `~x`, `-x`, `x++`, `(double)x`·
- **δυαδικοί (binary)**, με δύο: `x + y`, `x < y`, `x && y`, `x = y`·
- **τριαδικοί (ternary)**, με τρεις: ο μοναδικός είναι ο τελεστής συνθήκης `c ? a : b`.

Με βάση τη χρήση τους χωρίζονται σε αριθμητικούς, συγκριτικούς, λογικούς, bitwise, συνθήκης, μετατροπής, ανάθεσης (και οι συγγενείς τους: αύξησης/μείωσης και παράθεσης). Τέλος, κάθε

τελεστής έχει **προτεραιότητα** και **προσεταιριστικότητα**, που ορίζουν τη σειρά υπολογισμού.

§4.7 Αριθμητικοί τελεστές

Τελεστής	Ερμηνεία	Παραδείγματα
`+`	πρόσθεση	`40 + 2` ⟶ `42`, `40.0 + 2.0` ⟶ `42.0`
`-`	αφαίρεση	`44 - 2` ⟶ `42`, `44.0 - 2.0` ⟶ `42.0`
`*`	πολλαπλασιασμός	`42 * 2` ⟶ `84`, `42.0 * 2.0` ⟶ `84.0`
`/`	διαίρεση	`85.0 / 2.0` ⟶ `42.5`, `85 / 2` ⟶ `42`
`%`	υπόλοιπο	`7 % 2` ⟶ `1`

Είναι δυαδικοί (τα `+` και `-` υπάρχουν και ως μοναδιαίοι, π.χ. `-x`). Προσέξτε τη διαίρεση: όταν **και οι δύο** τελεστέοι είναι ακέραιοι, το `/` κάνει **ακέραια διαίρεση** και δίνει το πηλίκο, πετώντας το κλασματικό μέρος (`85 / 2` είναι `42`, `3 / 4` είναι `0`). Το `%` ορίζεται μόνο για ακεραίους.

§4.8 Συγκριτικοί τελεστές

Οι **συγκριτικοί τελεστές (comparison / relational operators)** συγκρίνουν δύο τελεστέους και επιστρέφουν `0` (ψευδές) ή `1` (αληθές). Είναι δυαδικοί.

Τελεστής	Ερώτηση	Παραδείγματα
`==`	ίσοι τελεστέοι;	`42 == 0` ⟶ `0`, `42 == 42` ⟶ `1`
`!=`	διαφορετικοί;	`42 != 0` ⟶ `1`, `42 != 42` ⟶ `0`
`>`	αριστερός μεγαλύτερος;	`42 > 0` ⟶ `1`, `42 > 42` ⟶ `0`
`<`	αριστερός μικρότερος;	`42 < 0` ⟶ `0`, `42 < 44` ⟶ `1`
`>=`	μεγαλύτερος ή ίσος;	`42 >= 42` ⟶ `1`, `42 >= 43` ⟶ `0`
`<=`	μικρότερος ή ίσος;	`42 <= 42` ⟶ `1`, `42 <= 41` ⟶ `0`

Η ισότητα γράφεται με **δύο** `=`· το ένα `=` είναι ανάθεση.

§4.9 Λογικοί τελεστές

Οι **λογικοί τελεστές (logical operators)** συνδυάζουν αληθείς και ψευδείς τιμές:

Τελεστής	Ερμηνεία	Παραδείγματα
`&&`	λογική σύζευξη (AND, ∧): αληθείς και οι δύο;	`42 > 0 && 2 > 3` ⟶ `0`, `42 > 0 && 3 > 2` ⟶ `1`
`||`	λογική διάζευξη (OR, ∨): αληθής ένας από τους δύο;	`2 > 3 || 42 > 0` ⟶ `1`, `2 > 3 || 42 < 0` ⟶ `0`

Τελεστής	Ερμηνεία	Παραδείγματα
`!`	λογική άρνηση (NOT, ¬): ψευδής ο τελεστέος;	`!(42 < 0)` ⟶ `1`, `!(42 > 0)` ⟶ `0`

Οι `&&` και `||` είναι δυαδικοί, ο `!` μοναδιαίος. **Σημαντικό**: για τους λογικούς τελεστές της C, το **μηδέν σημαίνει «ψευδές»** και **οποιαδήποτε μη μηδενική τιμή σημαίνει «αληθές»**. Τα `1`, `42` και `-5` είναι όλα εξίσου αληθή, άρα `!42` είναι `0` και `-5 && 42` είναι `1`. Το αποτέλεσμα ενός λογικού τελεστή είναι πάντα `0` ή `1`.

§4.10 Bitwise τελεστές

Οι **bitwise τελεστές** δουλεύουν πάνω στα bit των **ακεραίων** αριθμών, ένα bit κάθε φορά. Είναι βολικό να τους βλέπουμε σε δεκαεξαδικό, όπου κάθε ψηφίο είναι 4 bit (`0xF` = `1111`, `0x0` = `0000`).

Τελεστής	Ερμηνεία	Παραδείγματα
`&`	σύζευξη bit (bitwise AND)	`0xFF & 0xF0` ⟶ `0xF0`, `0xFF & 0x00` ⟶ `0x00`
`|`	διάζευξη bit (bitwise OR)	`0xFF | 0xF0` ⟶ `0xFF`, `0xF0 | 0x0F` ⟶ `0xFF`
`^`	αποκλειστική διάζευξη (bitwise XOR)	`0xFF ^ 0xFF` ⟶ `0x00`, `0x00 ^ 0xFF` ⟶ `0xFF`
`~`	άρνηση bit (bitwise NOT)	`~0xF0` ⟶ `0x0F`, `~0x05` ⟶ `0xFA` (σε 8 bit)
`<<`	ολίσθηση αριστερά κατά N bit, σαν πολλαπλασιασμός με 2^N	`2 << 4` ⟶ `32`, `4 << 1` ⟶ `8`
`>>`	ολίσθηση δεξιά κατά N bit, σαν διαίρεση με 2^N	`8 >> 1` ⟶ `4`, `32 >> 4` ⟶ `2`

Ο `~` είναι μοναδιαίος, οι υπόλοιποι δυαδικοί. Τα παραδείγματα του `~` ισχύουν για ένα byte (8 bit)· σε `int` 32 bit όλα τα πάνω bit γίνονται επίσης 1, οπότε `~0xF0` είναι `0xFFFFFF0F`.

Δύο χρήσιμα μοτίβα: το `&` με μια **μάσκα** κρατάει μόνο τα bit όπου η μάσκα έχει 1 (π.χ. `x & 0x80` απομονώνει το πιο σημαντικό bit ενός byte), και το `|` **ανάβει** τα bit όπου η μάσκα έχει 1 και αφήνει τα υπόλοιπα ίδια. Μην μπερδεύετε `&` με `&&` και `|` με `||`: `2 & 1` είναι `0`, ενώ `2 && 1` είναι `1`.

§4.11 Τελεστής συνθήκης

Ο **τελεστής συνθήκης (conditional operator)** ελέγχει μια συνθήκη· αν είναι αληθής επιστρέφει την πρώτη τιμή, αλλιώς τη δεύτερη:

```
συνθήκη ? τιμή1 : τιμή2
```

Έτσι `(42 > 0) ? 8 : 16` επιστρέφει 8 και `(42 == 0) ? 8 : 16` επιστρέφει 16. Είναι ο μοναδικός **τριαδικός** τελεστής της C. Επειδή είναι παράσταση (έχει τιμή), μπαίνει όπου μπαίνει και μια τιμή: `max = (a > b) ? a : b;`.

§4.12 Τελεστής μετατροπής και σιωπηρή μετατροπή τύπων

Ο **τελεστής μετατροπής (cast operator)** αλλάζει τον τύπο ενός τελεστέου: `(τύπος)τελεστέος`. Είναι μοναδιαίος. Αυτό λέγεται και **ρητή μετατροπή τύπου (explicit type conversion)**.

- `(int)42.67` επιστρέφει `42`: τα ψηφία μετά την υποδιαστολή **αποκόπτονται**, δεν στρογγυλοποιούνται.
- `(char)67.8` επιστρέφει `'C'`: αποκοπή σε 67, που είναι ο κωδικός ASCII του `C`.
- `(double)3` επιστρέφει `3.0`.
- `(double)3/4` επιστρέφει `0.75`: το cast έχει μεγαλύτερη προτεραιότητα από το `/`, οπότε μετατρέπεται πρώτα το `3` και η διαίρεση γίνεται με πραγματικούς. Αντίθετα, `(double)(3/4)` είναι `0.0`, γιατί η ακέραια διαίρεση `3/4` δίνει ήδη `0`.

Όταν ένας τελεστής έχει τελεστέους **διαφορετικών τύπων**, ο μεταγλωττιστής προσθέτει αυτόματα μετατροπή στον «μεγαλύτερο» από τους δύο τύπους. Αυτή είναι η **σιωπηρή μετατροπή τύπων (implicit type conversion)**. Το `40 + 2.0` είναι ισοδύναμο με `(double)40 + 2.0`, δηλαδή με `40.0 + 2.0`, και δίνει `double`. Η ιεραρχία των διαφανειών, από τον «μικρότερο» στον «μεγαλύτερο» τύπο, είναι:

Σχήμα: η ιεραρχία της σιωπηρής μετατροπής· ο «μικρότερος» τύπος μετατρέπεται στον «μεγαλύτερο».

Μετατροπή γίνεται και στην ανάθεση: η τιμή μετατρέπεται στον τύπο της μεταβλητής, με απώλεια πληροφορίας αν ο τύπος δεν τη χωράει (`int i = 42.67;` δίνει `i == 42`).

§4.13 Τελεστής ανάθεσης

Ο **τελεστής ανάθεσης (assignment operator)** `=` παίρνει την τιμή του δεξιού τελεστέου (**rvalue**), την αναθέτει στη **μεταβλητή** του αριστερού τελεστέου (**lvalue**) και **την επιστρέφει**. Το `x = 42` αναθέτει `42` στο `x` και επιστρέφει `42`. Αριστερά πρέπει να υπάρχει κάτι που έχει θέση στη μνήμη (μια μεταβλητή)· το `42 = x` δεν μεταγλωττίζεται.

Επειδή η ανάθεση επιστρέφει τιμή και έχει προσεταιριστικότητα από δεξιά προς αριστερά, μπορούμε να αναθέσουμε σε πολλές μεταβλητές ταυτόχρονα: το `x = y = 42` είναι ισοδύναμο με `x = (y = 42)`.

§4.14 Συνδυαστικοί τελεστές ανάθεσης

Οι **συνδυαστικοί τελεστές ανάθεσης (compound assignment operators)** κάνουν μια πράξη και σώζουν το αποτέλεσμα με συντομογραφία:

```
τελεστέος1 op= τελεστέος2
```

όπου op οποιοσδήποτε από τους +, -, *, %, /, &, ^, |, <<, >>. Το a += b είναι ισοδύναμο με a = a + b, το a *= b με a = a * b, και ομοίως για τους άλλους. Προσοχή: ολόκληρο το δεξί μέλος υπολογίζεται πρώτα, οπότε το a *= b + 1 σημαίνει a = a * (b + 1).

§4.15 Τελεστές αύξησης και μείωσης

Ο **τελεστής αύξησης** ++ μπαίνει πριν ή μετά από το όνομα μιας μεταβλητής και την αυξάνει κατά 1· ο **τελεστής μείωσης** -- τη μειώνει κατά 1. Η θέση τους καθορίζει **ποια τιμή επιστρέφουν**:

Μορφή	Γραφή	Επιστρέφει
επιθεματική (postfix)	a++, a--	την τιμή του a **πριν** την αύξηση/μείωση
προθεματική (prefix)	++a, --a	την τιμή του a **μετά** την αύξηση/μείωση

```
a = 5;
b = a++;   /* b == 5, a == 6 */
c = ++a;   /* c == 7, a == 7 */
```

Ως αυτόνομες εντολές (a++; ή ++a;) κάνουν ακριβώς το ίδιο· η διαφορά φαίνεται μόνο όταν η τιμή τους χρησιμοποιείται μέσα σε μεγαλύτερη παράσταση.

§4.16 Τελεστής παράθεσης

Ο **τελεστής παράθεσης (comma operator)** , υπολογίζει τον πρώτο τελεστέο, αγνοεί το αποτέλεσμα και επιστρέφει τον δεύτερο: το 12, 42 επιστρέφει 42. Έχει σχετικά περιορισμένες χρήσεις και συνήθως χρησιμοποιείται με τελεστές ανάθεσης, π.χ. i = 0, j = 10 (δύο αναθέσεις σε μία παράσταση). Έχει τη **χαμηλότερη** προτεραιότητα από όλους, οπότε x = 12, 42; αναθέτει 12 στο x, ενώ x = (12, 42); αναθέτει 42.

§4.17 Προτεραιότητα και προσεταιριστικότητα

Η **προτεραιότητα (precedence)** ενός τελεστή καθορίζει τη σειρά με την οποία εκτελούνται οι υπολογισμοί. Ο πολλαπλασιασμός έχει υψηλότερη προτεραιότητα από την πρόσθεση, άρα 5 * 6 + 3 * 4 είναι (5 * 6) + (3 * 4) = 42.

Αν δύο τελεστές έχουν την ίδια προτεραιότητα, η **προσεταιριστικότητα (associativity)** καθορίζει τη σειρά: **από αριστερά προς τα δεξιά** ή το αντίστροφο. Ο πολλαπλασιασμός και η διαίρεση έχουν την ίδια προτεραιότητα και προσεταιριστικότητα από αριστερά προς τα δεξιά, άρα 90 * 50 / 100 είναι (90 * 50) / 100 = 4500 / 100 = 45. Με ακεραίους η σειρά αλλάζει το αποτέλεσμα: 50 / 100 * 90 είναι (50 / 100) * 90 = 0 * 90 = 0.

Θέση	Τελεστές	Προσεταιριστικότητα
1	`()` (παρενθέσεις, κλήση συνάρτησης), `[]`, `->`, `.`, `++ --` (επιθεματικά)	αριστερά προς δεξιά
2	`++ --` (προθεματικά), `!`, `~`, `*` (έμμεση αναφορά), `&` (διεύθυνση), `+ -` (μοναδιαία), `(τύπος)`, `sizeof`	δεξιά προς αριστερά
3	`* / %`	αριστερά προς δεξιά
4	`+ -`	αριστερά προς δεξιά
5	`<< >>`	αριστερά προς δεξιά
6	`< <= > >=`	αριστερά προς δεξιά
7	`== !=`	αριστερά προς δεξιά
8	`&`	αριστερά προς δεξιά
9	`^`	αριστερά προς δεξιά
10	`|`	αριστερά προς δεξιά
11	`&&`	αριστερά προς δεξιά
12	`||`	αριστερά προς δεξιά
13	`?:`	δεξιά προς αριστερά
14	`= += -= *= /= %= &= ^= |= <<= >>=`	δεξιά προς αριστερά
15	`,`	αριστερά προς δεξιά

Η θέση 1 έχει την υψηλότερη προτεραιότητα. Χρήσιμοι κανόνες: μοναδιαίοι πριν από αριθμητικούς, αριθμητικοί πριν από συγκριτικούς, συγκριτικοί πριν από λογικούς, και ανάθεση σχεδόν τελευταία (γι' αυτό το `x = a + b > c` αναθέτει το αποτέλεσμα της σύγκρισης). Οι τελεστές `*` (έμμεση αναφορά), `&` (διεύθυνση), `->`, `.` και `[]` θα τους δούμε με τους δείκτες και τους πίνακες. **Δεν είστε σίγουροι για την ακριβή σειρά; Χρησιμοποιήστε παρενθέσεις `()`.**

Παραδείγματα

§4.18 Πρόγραμμα υπολογισμού βαθμολογίας

Εφαρμόζει: συναρτήσεις (Κεφάλαιο 3), `argc`/`argv`/`atoi`*, αριθμητικούς τελεστές, προτεραιότητα.* Το live coding της διάλεξης. Ο βαθμός των πρωτοετών είναι 50% Τελική Εξέταση + 30% Ασκήσεις + 20% Εργαστήριο, με κάθε βαθμό ακέραιο στο $[0, 100]$. Στα μαθηματικά:

$$\text{grade}(f, h, l) = 0.5f + 0.3h + 0.2l$$

Έλεγχος ορθότητας: $\text{grade}(70, 80, 100) = 35 + 24 + 20 = 79$. Πώς το γράφουμε σε C; Μια περσινή λύση:

```c
#include <stdio.h>
#include <stdlib.h>
```

```c
// The grade function computes the final grade for first year students
// according to https://progintro.github.io
int grade(int final_exam, int homework, int lab) {
  return final_exam*50/100 + homework*30/100 + lab*20/100;
}

int main(int argc, char **argv) {
  if (argc != 4) {
    printf("Run as: grade final_exam homework lab\n");
    return 1;
  }
  int final_exam = atoi(argv[1]);
  int homework = atoi(argv[2]);
  int lab = atoi(argv[3]);
  int final_grade = grade(final_exam, homework, lab);
  printf("My grade is: %d\n", final_grade);
  return 0;
}
```

```
$ gcc -o grade grade.c
$ ./grade 70 80 100
My grade is: 79
$ ./grade 70 80
Run as: grade final_exam homework lab
```

Παρατηρήστε:

- Το `argc != 4` ελέγχει ότι δόθηκαν ακριβώς τρία ορίσματα (το `argv[0]` είναι το όνομα του προγράμματος). Αλλιώς τυπώνουμε οδηγίες χρήσης και επιστρέφουμε 1 (αποτυχία).
- Τα `argv[1]`...`argv[3]` είναι κείμενο· το `atoi` τα κάνει ακεραίους.
- Όλα είναι `int`, άρα το 50% γράφεται `*50/100` και **όχι** `*0.5` ή `*(50/100)`. Επειδή `*` και `/` έχουν ίδια προτεραιότητα και προσεταιριστικότητα από αριστερά, το `final_exam*50/100` είναι `(final_exam*50)/100`: πρώτα πολλαπλασιασμός, μετά διαίρεση. Το `50/100*final_exam` θα έδινε πάντα `0`.
- Κάθε όρος περικόπτεται χωριστά από την ακέραια διαίρεση, οπότε ο βαθμός δεν στρογγυλοποιείται: `./grade 71 81 99` δίνει `35 + 24 + 19 = 78`, ενώ ο ακριβής βαθμός είναι 78,6.

Η διάλεξη αφήνει και έναν **κανόνα αναπροσαρμογής**: αν ο βαθμός των Ασκήσεων είναι πάνω από 3 μονάδες μεγαλύτερος από τον βαθμό της Τελικής Εξέτασης, ο βαθμός των Ασκήσεων αναπροσαρμόζεται σε Τελική Εξέταση + 3. Πώς θα το εκφράσουμε; Είναι άσκηση (βλ. Ασκήσεις)· ο τελεστής συνθήκης αρκεί.

§4.19 Ο κύκλος του git βήμα προς βήμα

Εφαρμόζει: `clone`, `add`, `commit`, `push`, `pull`, `status`. Η διάλεξη κάνει τον κύκλο ζωντανά στο repository της hw0:

```
git clone git@github.com:progintro/hw0-barbouni-2005.git
cd hw0-barbouni-2005
ls                              # ένα αρχείο README.md
git status                      # καμία αλλαγή
touch command/solution.txt      # νέο αρχείο
git status                      # command/solution.txt: untracked
git add command/solution.txt
git status                      # έτοιμο για commit
git commit                      # γράψτε ένα μήνυμα για την αλλαγή
git status                      # μένει να σώσουμε και εκτός υπολογιστή
git push                        # ανέβασμα στο GitHub
```

Μετά το `push`, ελέγξτε ότι οι αλλαγές σας εμφανίζονται στη σελίδα του repository στο GitHub. Για το `pull`: κάντε μια αλλαγή σε ένα αρχείο από το web UI του GitHub και μετά τρέξτε `git pull` στον φάκελό σας· η αλλαγή εμφανίζεται τοπικά. Το ίδιο κάνει και η Άσκηση 1 του Εργαστηρίου 1 με ένα αρχείο `info.txt`.

§4.20 Κατηγορίες και «πρωτότυπα» τελεστών

Εφαρμόζει: τελεστής/τελεστέος, arity. Οι διαφάνειες ρωτούν για κάθε ομάδα «τι τύπου τελεστές είναι;» και «τι τύπο θα είχαν αν ήταν συναρτήσεις;». Οι αριθμητικοί, οι συγκριτικοί, οι `&&`/`||` και οι bitwise εκτός του `~` είναι δυαδικοί, άρα για `int` θα έμοιαζαν με `int f(int a, int b)`· οι `!`, `~` και το cast είναι μοναδιαίοι (`int f(int a)`, `double to_double(int a)`)· ο `? :` είναι τριαδικός (`int cond(int c, int a, int b)`). Για πραγματικούς τελεστέους ο ίδιος τελεστής θα ήταν άλλη «συνάρτηση» (`double plus(double a, double b)`), γι' αυτό `85 / 2` και `85.0 / 2.0` δίνουν διαφορετικό αποτέλεσμα.

§4.21 Μέγιστο με τον τελεστή συνθήκης

Εφαρμόζει: τελεστή συνθήκης, συναρτήσεις. «Πώς θα φτιάχναμε μια συνάρτηση `max`;» Αφού το `? :` επιστρέφει τιμή, αρκεί μία εντολή `return`:

```
int max(int a, int b) {
  return (a > b) ? a : b;
}
```

Κύρια σημεία

1. Στο pair programming ο driver γράφει κώδικα και ο navigator ελέγχει κάθε γραμμή· οι ρόλοι αλλάζουν συχνά.
2. Στην `main(int argc, char **argv)` το `argc` είναι το πλήθος των ορισμάτων και το `argv[1]` το 1ο όρισμα ως κείμενο· το `atoi` το κάνει ακέραιο.
3. Το version control κρατάει όλο το ιστορικό των αλλαγών· το git είναι το δημοφιλέστερο, και στο GitHub θα γράφετε και θα υποβάλλετε τις εργασίες σας.
4. Ο κύκλος του git είναι `clone` μία φορά και μετά `add`, `commit`, `push`· το `pull` φέρνει αλλαγές από τον server. Μην ξεχνάτε το `add` πριν από `commit` και `push`.
5. Οι τελεστές κάνουν υπολογισμούς πάνω σε τελεστέους και επιστρέφουν μια τιμή, όπως οι συναρτήσεις με τα ορίσματά τους· κατά arity είναι μοναδιαίοι, δυαδικοί ή τριαδικοί.
6. Η διαίρεση δύο ακεραίων είναι ακέραια διαίρεση: `85 / 2` είναι `42`.
7. Οι συγκριτικοί και οι λογικοί τελεστές επιστρέφουν `0` ή `1`· για τους λογικούς το μηδέν είναι «ψευδές» και κάθε μη μηδενική τιμή (`1`, `42`, `-5`) «αληθές».
8. Οι bitwise τελεστές δουλεύουν στα bit των ακεραίων· η ολίσθηση κατά N bit ισοδυναμεί με πολλαπλασιασμό ή διαίρεση με 2^N.
9. Το cast `(τύπος)τελεστέος` κάνει ρητή μετατροπή, και από πραγματικό σε ακέραιο αποκόπτει (δεν στρογγυλοποιεί)· σε μικτές παραστάσεις γίνεται σιωπηρή μετατροπή στον «μεγαλύτερο» τύπο.
10. Η ανάθεση αναθέτει το rvalue στο lvalue και επιστρέφει την τιμή, γι' αυτό `x = y = 42` σημαίνει `x = (y = 42)`· το `a op= b` σημαίνει `a = a op b`.
11. Το `a++` επιστρέφει την τιμή πριν την αύξηση, το `++a` την τιμή μετά.
12. Ο τελεστής παράθεσης υπολογίζει τον πρώτο τελεστέο, τον αγνοεί και επιστρέφει τον δεύτερο.
13. Η προτεραιότητα ορίζει ποιος τελεστής εφαρμόζεται πρώτος και η προσεταιριστικότητα τη σειρά για ίση προτεραιότητα· όταν δεν είστε σίγουροι, χρησιμοποιήστε παρενθέσεις.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
προγραμματισμός σε ζεύγη	pair programming	Δύο προγραμματιστές σε έναν υπολογιστή, driver και navigator
έλεγχος εκδόσεων	version control	Σύστημα που κρατάει το ιστορικό όλων των αλλαγών
αποθετήριο	repository	Φάκελος αρχείων μαζί με το ιστορικό τους
προσωρινή λίστα	staging area	Οι αλλαγές που θα μπουν στο επόμενο commit
καταχώρηση	commit	Αποθήκευση των αλλαγών στο τοπικό repository

Ελληνικά	English	Σύντομος ορισμός
τελεστής	operator	Σύμβολο που κάνει υπολογισμό και επιστρέφει τιμή
τελεστέος	operand	Μεταβλητή ή σταθερά πάνω στην οποία δουλεύει ο τελεστής
πλήθος τελεστέων	arity	Μοναδιαίος (1), δυαδικός (2), τριαδικός (3)
τελεστής μετατροπής	cast operator	`(τύπος)τελεστέος`, ρητή μετατροπή τύπου
σιωπηρή μετατροπή	implicit type conversion	Αυτόματη μετατροπή στον «μεγαλύτερο» τύπο
τελεστής ανάθεσης	assignment operator	`=`· αναθέτει και επιστρέφει την τιμή
lvalue / rvalue	lvalue / rvalue	Αριστερός (μεταβλητή) / δεξιός (τιμή) τελεστέος της ανάθεσης
επιθεματικός / προθεματικός	postfix / prefix	`a++` (παλιά τιμή) / `++a` (νέα τιμή)
τελεστής παράθεσης	comma operator	`a, b`: υπολογίζει το `a`, επιστρέφει το `b`
προτεραιότητα	precedence	Ποιος τελεστής εφαρμόζεται πρώτος
προσεταιριστικότητα	associativity	Σειρά υπολογισμού για ίση προτεραιότητα

Διάβασμα

- **Διαφάνειες**: Διάλεξη 4, σελ. 1–44. Pair programming: σελ. 5–7· πρόγραμμα βαθμολογίας και `argc`/`argv`/`atoi`: σελ. 8–10, 38· version control και git: σελ. 11–19· τελεστές: σελ. 20–37· προτεραιότητα και προσεταιριστικότητα: σελ. 39–41.
- **Σημειώσεις**: Κεφάλαιο 2: Μεταβλητές, τύποι, τελεστές και παραστάσεις, ενότητα «Εντολές αντικατάστασης, τελεστές και παραστάσεις» (K04, σελ. 34–43). Οι διαφάνειες συνιστούν να έχετε καλύψει τις σημειώσεις του κ. Σταματόπουλου μέχρι τη σελίδα 43, δηλαδή ολόκληρη αυτή την ενότητα.
- **Εργαστήριο**: Εργαστήριο 1: «Βήμα 6: Git και GitHub» (λογαριασμός, κλειδί SSH, `git config`) και «Άσκηση 1: Το πρώτο σας repository (info.txt)».
- **Βιβλίο**: K&R, §2.5–§2.12, όπως προτείνουν οι σημειώσεις.
- **Άλλα**:
  - Wikipedia (σελ. 5): Software development, Computer programmer, Source code, Code review
  - Wikipedia: Git, Arity, Comma operator, Value (lvalues και rvalues)
  - Type conversion in C, List of operators, Precedence and associativity (cppreference)
  - Προαιρετικά: Git Cheat Sheet, Git Tutorial
  - Για το pair programming: Jeff Dean quotes, Jeff and Sanjay (The New Yorker)

Συχνά λάθη

- `git commit` **χωρίς** `git add`. Το νέο αρχείο μένει untracked και δεν ανεβαίνει ποτέ. Διόρθωση: `git status`, `git add <αρχείο>`, ξανά `commit` και `push`.
- **Commit χωρίς push**. Οι αλλαγές υπάρχουν μόνο στον υπολογιστή σας και το GitHub δεν τις βλέπει· για τις εργασίες μετράει ό,τι έχει γίνει push. Ελέγξτε τη σελίδα του repository.
- `Permission denied (publickey)` **στο** `clone`/`push`. Δεν έχει προστεθεί κλειδί SSH στο GitHub. Ακολουθήστε το Βήμα 6 του Εργαστηρίου 1.
- `push` **που απορρίπτεται (**`rejected ... fetch first`**)**. Υπάρχουν στο GitHub commits που δεν έχετε τοπικά (π.χ. από το web UI). Τρέξτε πρώτα `git pull`.
- **Ακέραια διαίρεση**. `final_exam * (50/100)` ή `3 / 4` δίνουν `0`. Διόρθωση: πολλαπλασιασμός πριν από τη διαίρεση, ή ένας τελεστέος `double` (`3.0 / 4`, `(double)3 / 4`).
- **Cast στη λάθος θέση ή αναμονή στρογγυλοποίησης**. `(double)(3 / 4)` είναι `0.0` (η ακέραια διαίρεση έγινε πρώτα)· `(int)42.67` είναι `42`, όχι `43`.
- `=` **αντί για** `==`. `if (x = 42)` αναθέτει και είναι πάντα αληθές· ο gcc με `-Wall` προειδοποιεί (`suggest parentheses around assignment used as truth value`).
- `&` **αντί για** `&&` **(ή** `|` **αντί για** `||`**)**. `2 & 1` είναι `0`, `2 && 1` είναι `1`.
- **Τα παραδείγματα του** `~` **σε** `int`. `~0xF0` σε `int` είναι `0xFFFFFF0F`, όχι `0x0F`· η απάντηση των διαφανειών ισχύει για ένα byte.
- **Κόμμα χωρίς παρενθέσεις**. `x = 12, 42;` αναθέτει `12`, αφού το `,` έχει τη χαμηλότερη προτεραιότητα.
- **Υποθέσεις για την προτεραιότητα**. `x & 1 == 0` σημαίνει `x & (1 == 0)`, γιατί το `==` (θέση 7) προηγείται του `&` (θέση 8). Διόρθωση: `(x & 1) == 0`.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K4.9** Απομόνωση του πιο σημαντικού bit (14% σωστές): Το 29% διάλεξε `byte & 0xFF`: η μάσκα `0xFF` έχει και τα 8 bits στο 1, άρα κρατάει ολόκληρο το byte αντί να απομονώνει μόνο το πιο σημαντικό.
- **K4.8** 0x42 & 0xFF (18% σωστές): Το 30% απάντησε `0`, πιθανότατα επειδή μπέρδεψε το bitwise `&` με το λογικό `&&` ή υπέθεσε ότι το AND «σβήνει» τα bits· στην πραγματικότητα το AND με `0xFF` αφήνει το byte ανέπαφο.
- **K4.7** 0xbeef | 0xcafe0000 (29% σωστές): Το 29% απάντησε `0xbeefcafe`, διαβάζοντας το OR σαν «παράθεση με τη σειρά που γράφονται»· όμως το OR δεν μετακινεί bits, το καθένα μένει στη θέση του, και το `0xcafe` βρίσκεται στα πάνω 16 bits.
- **K4.5** Ποιες είναι εντολές git; (34% σωστές): Το 35% διάλεξε μόνο `git push` και `git commit`, ξεχνώντας το `git status`, που δεν αλλάζει τίποτα αλλά δείχνει ποια αρχεία έχουν αλλάξει ή είναι staged.

Ερωτήσεις κατανόησης

- **E4.1** Ποιοι είναι οι δύο ρόλοι στο pair programming και τι κάνει ο καθένας;[36]
- **E4.2** Αν τρέξετε `./grade 70 80 100`, ποια είναι η τιμή του `argc` και τι περιέχει το `argv[2]`;[37]
- **E4.3** Σε ποια κατάσταση είναι ένα αρχείο που μόλις δημιουργήσατε με `touch` και σε ποια μετά το `git add`;[38]
- **E4.4** Κάνατε `git commit` αλλά οι αλλαγές δεν φαίνονται στο GitHub. Γιατί;[39]
- **E4.5** Ποια η τιμή των `85 / 2`, `85.0 / 2`, `7 % 2` και `-5 && 42`;[40]
- **E4.6** Τι επιστρέφει το `(int)-42.67`, και γιατί όχι `-43`;[41]
- **E4.7** Με `a = 3`, ποιες είναι οι τιμές των `a` και `b` μετά το `b = a++ * 2;`;[42]
- **E4.8** Γιατί στο πρόγραμμα βαθμολογίας γράφουμε `final_exam*50/100` και όχι `50/100*final_exam`;[43]

Kahoot από το αμφιθέατρο (K4.1–K4.9)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K4.1** Ξέχασα κάτι στο git;: 95% σωστές απαντήσεις
- **K4.2** Git και GitHub: 82% σωστές απαντήσεις
- **K4.3** 0xFF | 0x42: 57% σωστές απαντήσεις
- **K4.4** 3 « 2: 55% σωστές απαντήσεις
- **K4.5** Ποιες είναι εντολές git;: 34% σωστές απαντήσεις
- **K4.6** Η σειρά των εντολών git: 34% σωστές απαντήσεις
- **K4.7** 0xbeef | 0xcafe0000: 29% σωστές απαντήσεις
- **K4.8** 0x42 & 0xFF: 18% σωστές απαντήσεις
- **K4.9** Απομόνωση του πιο σημαντικού bit: 14% σωστές απαντήσεις

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A4.1–A4.10)

- **A4.1** Η τιμή του 0xbeef | 0xcafe0000: Διάλεξη 4: Git και Τελεστές, διαφάνεια 30 · ★☆☆ · multiple-choice · `slides-lec04-cafebeef`

[36] Ο driver γράφει τον κώδικα και ο observer/navigator ελέγχει (review) κάθε γραμμή καθώς γράφεται· αλλάζουν ρόλους συχνά.

[37] `argc == 4` (το όνομα του προγράμματος και τρία ορίσματα)· το `argv[2]` είναι το κείμενο `"80"`, που το `atoi` κάνει ακέραιο `80`.

[38] Πριν: untracked (το git δεν το παρακολουθεί). Μετά το `git add`: στο staging area, έτοιμο για commit.

[39] Το `commit` αποθηκεύει μόνο στο τοπικό repository· χρειάζεται `git push` για να ανέβουν οι αλλαγές στο GitHub.

[40] `42` (ακέραια διαίρεση), `42.5` (σιωπηρή μετατροπή σε `double`), `1`, και `1` (και οι δύο μη μηδενικοί, άρα αληθείς).

[41] `-42`: το cast αποκόπτει το δεκαδικό μέρος (προς το μηδέν), δεν στρογγυλοποιεί.

[42] Το `a++` επιστρέφει την παλιά τιμή 3, άρα `b == 6`, και μετά `a == 4`.

[43] Επειδή όλα είναι `int` και `*`, `/` υπολογίζονται από αριστερά: το `final_exam*50` γίνεται πρώτο και μετά διαιρείται με 100. Το `50/100` είναι ακέραια διαίρεση που δίνει `0`, άρα όλος ο όρος θα γινόταν `0`.

- **A4.2** Ο κύκλος clone, add, commit, push, pull: Διάλεξη 4: Git και Τελεστές, διαφάνειες 15-18 · ★☆☆ · tooling · `slides-lec04-git-cycle`
- **A4.3** Κανόνας αναπροσαρμογής βαθμού ασκήσεων: Διάλεξη 4: Git και Τελεστές, διαφάνεια 8 · ★☆☆ · programming · `slides-lec04-grade-adjustment`
- **A4.4** Υπολογισμός βαθμολογίας πρωτοετών: Διάλεξη 4: Git και Τελεστές, διαφάνεια 8 · ★☆☆ · programming · `slides-lec04-grade-program`
- **A4.5** Συνάρτηση max με τον τελεστή συνθήκης: Διάλεξη 4: Git και Τελεστές, διαφάνεια 31 · ★☆☆ · programming · `slides-lec04-max-conditional`
- **A4.6** Απομόνωση του πιο σημαντικού bit: Διάλεξη 4: Git και Τελεστές, διαφάνεια 29 · ★☆☆ · multiple-choice · `slides-lec04-msb`
- **A4.7** Τι τύπου τελεστές είναι;: Διάλεξη 4: Git και Τελεστές, διαφάνειες 25-28 και 31 · ★☆☆ · short-answer · `slides-lec04-operator-arity`
- **A4.8** Τελεστές ως συναρτήσεις: Διάλεξη 4: Git και Τελεστές, διαφάνειες 21 και 25 · ★☆☆ · short-answer · `slides-lec04-operators-as-functions`
- **A4.9** Προτεραιότητα και προσεταιριστικότητα: Διάλεξη 4: Git και Τελεστές, διαφάνεια 39 · ★☆☆ · trace · `slides-lec04-precedence`
- **A4.10** Τι επιστρέφει το (double)3/4;: Διάλεξη 4: Git και Τελεστές, διαφάνεια 32 · ★★☆ · trace · `slides-lec04-cast-division`

Εργαστήριο (A4.11)

- **A4.11** Το πρώτο σας repository: Εργαστήριο 1, Άσκηση 1 · ★☆☆ · tooling · `lab-lab01-info`

Εργασίες (A4.12)

- **A4.12** Νέο URL στο GitHub (pages): Εργασία 0 (2025-26), Άσκηση 1 · ★☆☆ · tooling · `hw-2025-hw0-pages`

Θέματα εξετάσεων (A4.13)

- **A4.13** Άρτια Bits: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #8, Θέμα 1 · ★☆☆ · programming · `exam-2023-fall-ex8-q1`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A1.11** Ξεκινώντας με την γραμμή εντολών (bandit): Εργασία 0 (2023-24), Άσκηση 2 · ★★☆ · tooling · `hw-2023-hw0-bandit`
- **A1.12** Παιχνίδια με Κονσόλα (cmdline): Εργασία 0 (2025-26), Άσκηση 2 · ★★☆ · tooling · `hw-2025-hw0-cmdline`
- **A6.20** Κάντο όπως ο Βιετά: Κατατακτήριες Δεκεμβρίου 2023, Θέμα 1 · ★★☆ · programming · `exam-2023-dec-q1`

- **A6.21** Ασημένιο Κλάσμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #10, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex10-q2`
- **A6.22** Κάντο όπως ο Βιετά: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #13, Θέμα 1 · ★★☆ · programming · `exam-2023-fall-ex13-q1`
- **A6.23** Τυπώνοντας τον Πίνακα ASCII: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #15, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex15-q2`
- **A6.16** Μέγιστος Κοινός Διαιρέτης: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #6 (Crypto Themed), Θέμα 1 · ★☆☆ · programming · `exam-2023-fall-ex6-q1`
- **A6.17** Προσεγγίζοντας το π: Κατατακτήριες Δεκεμβρίου 2024, Θέμα 1 · ★☆☆ · programming · `exam-2024-dec-q1`
- **A6.25** Εύρεση Πρώτων Παραγόντων - factor: Εξέταση Ιανουαρίου 2026, Θέμα 3 · ★★☆ · programming · `exam-2026-jan-q3`
- **A6.14** Η εικασία Collatz: Εργασία 0 (2023-24), Άσκηση 3 · ★★☆ · programming · `hw-2023-hw0-collatz`
- **A8.7** Η Μέθοδος Newton-Raphson: Εργασία 1 (2023-24), Άσκηση 1 · ★★☆ · programming · `hw-2023-hw1-newton`
- **A9.30** Αποκωδικοποίηση: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #8, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex8-q2`
- **A16.19** Εαυτοί Αριθμοί: Κατατακτήριες Δεκεμβρίου 2023, Θέμα 2 · ★★☆ · programming · `exam-2023-dec-q2`
- **A16.14** Το στοιχείο χωρίς ζευγάρι: Διάλεξη 16, διαφάνεια 14 · ★★☆ · programming · `slides-lec16-single-unpaired`

Μέρος Β: Τα θεμέλια της C

Κεφάλαιο 5: Τελεστές και Εντολές

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να

- κατατάσσετε κάθε τελεστή της C κατά πλήθος τελεστέων (μοναδιαίος, δυαδικός, τριαδικός) και κατά οικογένεια (αριθμητικός, συγκριτικός, λογικός, bitwise, …)·
- υπολογίζετε με το χέρι την τιμή εκφράσεων με ακέραια διαίρεση, `%`, συγκρίσεις, λογικούς και bitwise τελεστές·
- χρησιμοποιείτε τον τελεστή συνθήκης `? :`, το cast και να προβλέπετε τις σιωπηρές μετατροπές τύπων·
- ξεχωρίζετε το `a++` από το `++a` και να ιχνηλατείτε εντολές που τα περιέχουν·
- εφαρμόζετε τους κανόνες προτεραιότητας και προσεταιριστικότητας, και να βάζετε παρενθέσεις όταν δεν είστε σίγουροι·
- διακρίνετε έκφραση από εντολή και να γράφετε απλές εντολές `if` και `if-else`.

Προαπαιτούμενα: Κεφάλαιο 2, Κεφάλαιο 3, Κεφάλαιο 4

Χρόνος μελέτης: ~2,5 ώρες

Σύνοψη

Οι τελεστές είναι τα «ρήματα» της C: με αυτούς κάνουμε πράξεις, συγκρίνουμε τιμές, συνδυάζουμε συνθήκες, πειράζουμε bits και αποθηκεύουμε αποτελέσματα σε μεταβλητές. Η διάλεξη περνάει όλες τις οικογένειες τελεστών της C (αριθμητικούς, συγκριτικούς, λογικούς, bitwise, συνθήκης, μετατροπής, ανάθεσης, αύξησης/μείωσης και παράθεσης) και εξηγεί πώς η C αποφασίζει τη σειρά των πράξεων με την προτεραιότητα και την προσεταιριστικότητα. Στη συνέχεια βλέπουμε πώς από τελεστές φτιάχνονται εκφράσεις και από εκφράσεις εντολές, και κάνουμε το πρώτο βήμα στη ροή ελέγχου με την `if` και την `if-else`. Σχεδόν κάθε γραμμή κώδικα που θα γράψετε στο μάθημα χρησιμοποιεί κάτι από αυτό το κεφάλαιο, και πολλά «περίεργα» αποτελέσματα (π.χ. ένας βαθμός που βγαίνει μικρότερος απ' ό,τι περιμένατε) εξηγούνται από αυτό.

Θεωρία

§5.1 Κατηγορίες τελεστών

Ένας **τελεστής (operator)** είναι ένα σύμβολο που εφαρμόζεται σε μία ή περισσότερες τιμές, τους **τελεστέους (operands)**, και δίνει ένα αποτέλεσμα. Τους τελεστές τους κατατάσσουμε με δύο

τρόπους:

- **Κατά πλήθος τελεστέων**: **μοναδιαίοι (unary)** με έναν τελεστέο (π.χ. `!x`, `-x`, `x++`), **δυαδικοί (binary)** με δύο (π.χ. `a + b`, `a < b`) και **τριαδικοί (ternary)** με τρεις. Η C έχει έναν μόνο τριαδικό τελεστή, τον τελεστή συνθήκης `? :`.
- **Κατά οικογένεια**: αριθμητικοί, συγκριτικοί, λογικοί, συνθήκης, bitwise, μετατροπής και ανάθεσης (μαζί με τις παραλλαγές τους).

Ένας χρήσιμος τρόπος σκέψης: ένας τελεστής συμπεριφέρεται σαν μια συνάρτηση με ειδική σύνταξη. Το `a + b` για δύο `int` μοιάζει με μια συνάρτηση `int add(int a, int b)`: παίρνει δύο ορίσματα και επιστρέφει μία τιμή.

§5.2 Αριθμητικοί τελεστές

Οι **αριθμητικοί τελεστές (arithmetic operators)** είναι δυαδικοί και δουλεύουν σε ακέραιους και σε αριθμούς κινητής υποδιαστολής:

Τελεστής	Ερμηνεία	Παραδείγματα
`+`	πρόσθεση	`40 + 2` δίνει `42`, `40.0 + 2.0` δίνει `42.0`
`-`	αφαίρεση	`44 - 2` δίνει `42`, `44.0 - 2.0` δίνει `42.0`
`*`	πολλαπλασιασμός	`42 * 2` δίνει `84`, `42.0 * 2.0` δίνει `84.0`
`/`	διαίρεση	`85.0 / 2.0` δίνει `42.5`, `85 / 2` δίνει `42`
`%`	υπόλοιπο	`7 % 2` δίνει `1`

Ο τύπος του αποτελέσματος εξαρτάται από τον τύπο των τελεστέων. Το σημαντικότερο παράδειγμα είναι η **ακέραια διαίρεση (integer division)**: όταν και οι δύο τελεστέοι του `/` είναι ακέραιοι, το αποτέλεσμα είναι ακέραιο πηλίκο και το δεκαδικό μέρος **χάνεται**. Γι' αυτό `85 / 2` είναι `42` και όχι `42.5`, και `3 / 4` είναι `0`. Ο τελεστής `%` (modulo) δίνει το υπόλοιπο της ακέραιας διαίρεσης και εφαρμόζεται μόνο σε ακέραιους. Με αρνητικούς τελεστέους η C κόβει το πηλίκο προς το μηδέν: `-7 / 2` είναι `-3` και `-7 % 2` είναι `-1`. Υπάρχουν και μοναδιαίοι `+` και `-` (π.χ. `-x`).

§5.3 Συγκριτικοί τελεστές

Οι **συγκριτικοί τελεστές (comparison ή relational operators)** συγκρίνουν δύο τελεστέους και δίνουν ακέραιο αποτέλεσμα: `1` **(αληθές)** ή `0` **(ψευδές)**. Είναι όλοι δυαδικοί:

Τελεστής	Ερμηνεία	Παραδείγματα
`==`	ίσοι τελεστέοι;	`42 == 0` δίνει `0`, `42 == 42` δίνει `1`
`!=`	διαφορετικοί τελεστέοι;	`42 != 0` δίνει `1`, `42 != 42` δίνει `0`
`>`	ο αριστερός μεγαλύτερος του δεξιού;	`42 > 0` δίνει `1`, `42 > 42` δίνει `0`
`<`	ο αριστερός μικρότερος του δεξιού;	`42 < 0` δίνει `0`, `42 < 44` δίνει `1`

Τελεστής	Ερμηνεία	Παραδείγματα
`>=`	ο αριστερός μεγαλύτερος ή ίσος;	`42 >= 42` δίνει 1, `42 >= 43` δίνει 0
`<=`	ο αριστερός μικρότερος ή ίσος;	`42 <= 42` δίνει 1, `42 <= 41` δίνει 0

Προσέξτε ότι η ισότητα γράφεται με **δύο** `=`. Το ένα `=` είναι ανάθεση.

§5.4 Λογικοί τελεστές

Οι **λογικοί τελεστές (logical operators)** συνδυάζουν αληθείς και ψευδείς τιμές:

Τελεστής	Ερμηνεία	Παραδείγματα
`&&`	λογική σύζευξη (AND, ∧): είναι και οι δύο αληθείς;	`42 > 0 && 2 > 3` δίνει 0, `42 > 0 && 3 > 2` δίνει 1
`||`	λογική διάζευξη (OR, ∨): είναι ένας από τους δύο αληθής;	`2 > 3 || 42 > 0` δίνει 1, `2 > 3 || 42 < 0` δίνει 0
`!`	λογική άρνηση (NOT, ¬): είναι ο τελεστέος ψευδής;	`!(42 < 0)` δίνει 1, `!(42 > 0)` δίνει 0

Οι `&&` και `||` είναι δυαδικοί, ο `!` μοναδιαίος. Ο πιο σημαντικός κανόνας του κεφαλαίου είναι ο εξής: **στην C το μηδέν (`0`) σημαίνει «ψευδές» και οποιαδήποτε άλλη, μη μηδενική, τιμή σημαίνει «αληθές».** Το `1`, το `42` και το `-5` είναι όλα εξίσου αληθή. Οι τελεστές *παράγουν* πάντα `0` ή `1`, αλλά *δέχονται* οποιαδήποτε τιμή· γι' αυτό το `!42` είναι `0` και το `if (n)` σημαίνει «αν το `n` δεν είναι μηδέν».

Οι `&&` και `||` υπολογίζουν πρώτα τον αριστερό τελεστέο και σταματούν αν το αποτέλεσμα έχει ήδη κριθεί (short-circuit evaluation): στο `0 && x` το `x` δεν υπολογίζεται καθόλου.

§5.5 Bitwise τελεστές

Οι **bitwise τελεστές (bitwise operators)** δουλεύουν πάνω στα μεμονωμένα bit **ακέραιων** αριθμών: εφαρμόζουν την πράξη σε κάθε θέση bit ξεχωριστά. Είναι όλοι δυαδικοί εκτός από τον `~`, που είναι μοναδιαίος.

Τελεστής	Ερμηνεία	Παραδείγματα
`&`	σύζευξη bit (bitwise AND)	`0xFF & 0xF0` δίνει `0xF0`, `0xFF & 0x00` δίνει `0x00`
`|`	διάζευξη bit (bitwise OR)	`0xFF | 0xF0` δίνει `0xFF`, `0xF0 | 0x0F` δίνει `0xFF`
`^`	αποκλειστική διάζευξη bit (bitwise XOR)	`0xFF ^ 0xFF` δίνει `0x00`, `0x00 ^ 0xFF` δίνει `0xFF`

Τελεστής	Ερμηνεία	Παραδείγματα
`~`	άρνηση bit (bitwise NOT)	`~0xF0` δίνει `0x0F`, `~0x05` δίνει `0xFA` (σε ένα byte)
`<<`	ολίσθηση αριστερά (shift left) κατά N bit, ισοδύναμο με πολλαπλασιασμό επί 2^N	`2 << 4` δίνει `32`, `4 << 1` δίνει `8`
`>>`	ολίσθηση δεξιά (shift right) κατά N bit, ισοδύναμο με διαίρεση διά 2^N	`8 >> 1` δίνει `4`, `32 >> 4` δίνει `2`

Η δεκαεξαδική γραφή (`0x…`) βολεύει εδώ, γιατί κάθε δεκαεξαδικό ψηφίο αντιστοιχεί σε ακριβώς 4 bit: `0xF0` είναι `1111 0000`. Τρεις χρήσεις που θα συναντάτε συχνά:

- **Απομόνωση bit με `&` και μάσκα (mask)**: το `x & m` κρατάει μόνο τα bit του `x` που είναι `1` και στη μάσκα `m`, και μηδενίζει τα υπόλοιπα.
- **Ενεργοποίηση bit με `|`**: το `x | m` κάνει `1` τα bit της μάσκας και αφήνει τα άλλα όπως ήταν.
- **Αντιστροφή bit με `^`**: το `x ^ m` αντιστρέφει τα bit της μάσκας.

Προσοχή: τα παραδείγματα του `~` στον πίνακα υποθέτουν τιμή ενός byte (8 bit). Σε ένα `int` 32 bit το `~0x05` είναι `0xFFFFFFFA`. Μην μπερδεύετε επίσης τα `&`/`|` (bit προς bit) με τα `&&`/`||` (λογικά, με αποτέλεσμα `0` ή `1`).

§5.6 Ο τελεστής συνθήκης

Ο **τελεστής συνθήκης (conditional operator)** ελέγχει μια συνθήκη και, αν είναι αληθής, δίνει την πρώτη τιμή, αλλιώς τη δεύτερη:

```
συνθήκη ? τιμή1 : τιμή2
```

Για παράδειγμα, `(42 > 0) ? 8 : 16` δίνει `8`, ενώ `(42 == 0) ? 8 : 16` δίνει `16`. Είναι ο μοναδικός **τριαδικός** τελεστής της C (τρεις τελεστέοι: η συνθήκη και οι δύο τιμές). Επειδή είναι *έκφραση*, μπορεί να μπει οπουδήποτε χρειάζεται μια τιμή, π.χ. στο δεξί μέλος μιας ανάθεσης ή σε ένα `return`. Υπολογίζεται μόνο η τιμή που επιλέγεται.

§5.7 Ο τελεστής μετατροπής (cast)

Ο **τελεστής μετατροπής (cast operator)** αλλάζει τον τύπο ενός τελεστέου:

```
(τύπος)τελεστέος
```

Έκφραση	Αποτέλεσμα
`(int)42.67`	`42`: τα ψηφία μετά την υποδιαστολή **αποκόπτονται**, δεν στρογγυλοποιούνται
`(char)67.8`	`'C'`: γίνεται `67`, ο κωδικός ASCII του `C`

Έκφραση	Αποτέλεσμα
`(double)3`	`3.0`
`(double)3/4`	; (δείτε τα «Παραδείγματα»)

Η μετατροπή που γράφουμε ρητά με cast λέγεται **ρητή μετατροπή τύπου (explicit type conversion)**. Το cast είναι μοναδιαίος τελεστής με υψηλή προτεραιότητα: εφαρμόζεται μόνο στον τελεστέο που ακολουθεί αμέσως, όχι σε όλη την έκφραση.

§5.8 Σιωπηρή μετατροπή τύπων

Όταν οι τελεστέοι ενός τελεστή έχουν διαφορετικούς τύπους, ο μεταγλωττιστής προσθέτει αυτόματα μια μετατροπή προς τον «μεγαλύτερο» από τους δύο τύπους. Αυτό λέγεται **σιωπηρή μετατροπή τύπων (implicit type conversion)**. Για παράδειγμα, το

```
40 + 2.0
```

είναι ισοδύναμο με `(double)40 + 2.0`, δηλαδή με `40.0 + 2.0`, και δίνει `double`. Η σειρά των τύπων, από τον «μικρότερο» στον «μεγαλύτερο», όπως τη δείχνουν οι διαφάνειες:

Σχήμα: η ιεραρχία της σιωπηρής μετατροπής· σε μικτή πράξη ο «μικρότερος» τύπος μετατρέπεται στον «μεγαλύτερο».

Η μετατροπή γίνεται **ανά τελεστή**, όχι για όλη την έκφραση μαζί. Στο `1 / 2 * 2.0` υπολογίζεται πρώτα το `1 / 2` ως ακέραια διαίρεση (`0`) και μόνο μετά μετατρέπεται σε `double`, οπότε το αποτέλεσμα είναι `0.0`.

§5.9 Ο τελεστής ανάθεσης

Ο **τελεστής ανάθεσης (assignment operator)** `=` παίρνει την τιμή του δεξιού τελεστέου (**rvalue**), την αναθέτει στη μεταβλητή του αριστερού τελεστέου (**lvalue**) και **επιστρέφει** αυτή την τιμή. Αριστερά πρέπει να υπάρχει κάτι που έχει θέση στη μνήμη (μια μεταβλητή)· το `42 = x` δεν έχει νόημα.

```
x = 42          // αναθέτει 42 στην x και επιστρέφει 42
x = y = 42      // ισοδύναμο με x = (y = 42)
```

Επειδή η ανάθεση είναι έκφραση με τιμή, μπορούμε να αναθέσουμε σε πολλές μεταβλητές ταυτόχρονα. Αυτό λειτουργεί επειδή ο `=` έχει προσεταιριστικότητα από δεξιά προς τα αριστερά: πρώτα γίνεται το `y = 42`, που επιστρέφει `42`, και αυτό ανατίθεται στο `x`.

§5.10 Συνδυαστικοί τελεστές ανάθεσης

Οι **συνδυαστικοί τελεστές ανάθεσης (compound assignment operators)** κάνουν μια πράξη και αποθηκεύουν το αποτέλεσμα με συντομογραφία:

`τελεστέος1 op= τελεστέος2`

όπου op είναι οποιοσδήποτε από τους `+`, `-`, `*`, `%`, `/`, `&`, `^`, `|`, `<<`, `>>`. Το `a += b` είναι ισοδύναμο με `a = a + b`, το `a *= b` με `a = a * b`, και ομοίως για τους υπόλοιπους. Προσοχή: το δεξί μέλος υπολογίζεται ολόκληρο πρώτα, άρα `a *= b + 1` σημαίνει `a = a * (b + 1)`.

§5.11 Τελεστές αύξησης και μείωσης

Ο **τελεστής αύξησης (increment operator)** `++` μπαίνει πριν ή μετά από το όνομα μιας μεταβλητής και την αυξάνει κατά 1· ο **τελεστής μείωσης (decrement operator)** `--` τη μειώνει κατά 1. Η θέση του έχει σημασία για την **τιμή που επιστρέφει**:

Μορφή	Όνομα	Τι επιστρέφει
`a++`	επιθεματική (postfix) αύξηση	την τιμή του a **πριν** την αύξηση
`a--`	επιθεματική (postfix) μείωση	την τιμή του a **πριν** τη μείωση
`++a`	προθεματική (prefix) αύξηση	την τιμή του a **μετά** την αύξηση
`--a`	προθεματική (prefix) μείωση	την τιμή του a **μετά** τη μείωση

Σε κάθε περίπτωση η μεταβλητή αλλάζει. Αν η έκφραση στέκεται μόνη της (`i++;`), οι δύο μορφές κάνουν το ίδιο. Η διαφορά φαίνεται όταν η τιμή χρησιμοποιείται:

```
a = 5;
b = a++;   // b = 5, a = 6
c = ++a;   // a = 7, c = 7
```

Μην αλλάζετε την ίδια μεταβλητή δύο φορές στην ίδια έκφραση (π.χ. `i = i++ + ++i;`): το αποτέλεσμα δεν ορίζεται από τη γλώσσα.

§5.12 Ο τελεστής παράθεσης (κόμμα)

Ο **τελεστής παράθεσης (comma operator)** `,` υπολογίζει τον πρώτο τελεστέο, αγνοεί το αποτέλεσμά του και επιστρέφει τον δεύτερο. Για παράδειγμα, `12, 42` δίνει `42`. Έχει σχετικά περιορισμένες χρήσεις και συνήθως χρησιμοποιείται μαζί με τελεστές ανάθεσης, για να γίνουν δύο αναθέσεις σε ένα σημείο όπου η γραμματική επιτρέπει μία έκφραση: `i = 0, j = 10`. Επειδή έχει τη χαμηλότερη προτεραιότητα από όλους τους τελεστές, το `x = 12, 42;` αναθέτει `12` στο `x`· για να πάρει το `x` την τιμή `42` χρειάζονται παρενθέσεις: `x = (12, 42);`.

Μην τον συγχέετε με τα κόμματα που χωρίζουν ορίσματα συνάρτησης (`f(a, b)`) ή μεταβλητές σε μια δήλωση (`int a, b;`): εκεί το κόμμα είναι απλώς διαχωριστικό.

§5.13 Προτεραιότητα και προσεταιριστικότητα

Η **προτεραιότητα (precedence)** ενός τελεστή καθορίζει τη σειρά με την οποία θα γίνουν οι υπολογισμοί. Ο πολλαπλασιασμός, για παράδειγμα, έχει υψηλότερη προτεραιότητα από την πρόσθεση, άρα το `5 * 6 + 3 * 4` σημαίνει `(5 * 6) + (3 * 4)`.

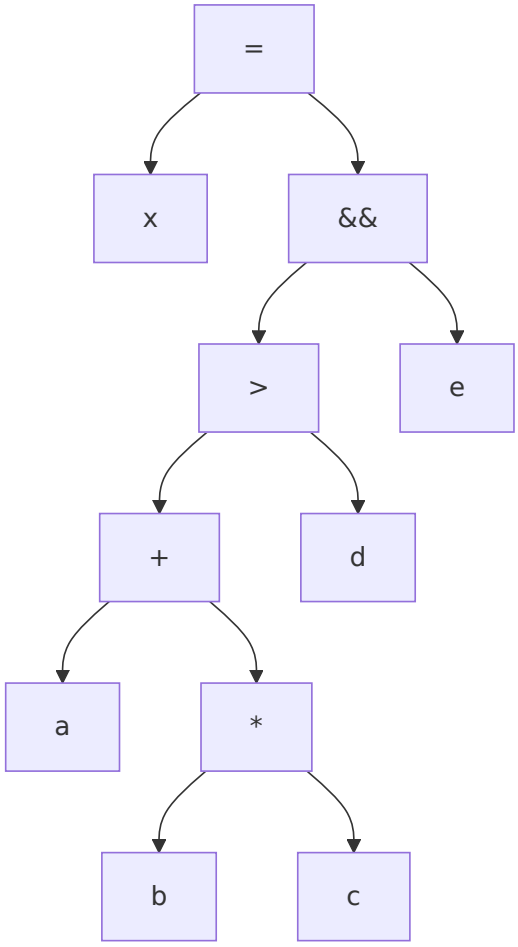
Αν δύο τελεστές έχουν την ίδια προτεραιότητα, η **προσεταιριστικότητα (associativity)** τους καθορίζει τη σειρά: από αριστερά προς τα δεξιά ή το αντίστροφο. Ο πολλαπλασιασμός και η διαίρεση έχουν την ίδια προτεραιότητα και προσεταιριστικότητα από αριστερά προς τα δεξιά, άρα το `90 * 50 / 100` σημαίνει `(90 * 50) / 100`. Με ακέραιους αυτό **δεν** είναι το ίδιο με `90 * (50 / 100)`.

Ο πλήρης πίνακας, από την υψηλότερη προτεραιότητα (1) στη χαμηλότερη (15):

Θέση	Τελεστές	Προσεταιριστικότητα
1	`()` (παρενθέσεις, κλήση συνάρτησης), `[]`, `->`, `.`, `++ --` (επιθεματικά)	αριστερά προς δεξιά
2	`++ --` (προθεματικά), `!`, `~`, `*` (έμμεση αναφορά), `&` (διεύθυνση), `+ -` (μοναδιαία), `(τύπος)` (cast), `sizeof`	δεξιά προς αριστερά
3	`*`, `/`, `%`	αριστερά προς δεξιά
4	`+`, `-` (δυαδικά)	αριστερά προς δεξιά
5	`<<`, `>>`	αριστερά προς δεξιά
6	`<`, `<=`, `>`, `>=`	αριστερά προς δεξιά
7	`==`, `!=`	αριστερά προς δεξιά
8	`&`	αριστερά προς δεξιά
9	`^`	αριστερά προς δεξιά
10	`|`	αριστερά προς δεξιά
11	`&&`	αριστερά προς δεξιά
12	`||`	αριστερά προς δεξιά
13	`?:`	δεξιά προς αριστερά
14	`=`, `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `^=`, `|=`, `<<=`, `>>=`	δεξιά προς αριστερά
15	`,`	αριστερά προς δεξιά

Δεν χρειάζεται να αποστηθίσετε όλον τον πίνακα. Κρατήστε τη γενική εικόνα: μοναδιαίοι, μετά αριθμητικοί, μετά ολισθήσεις, συγκρίσεις, bitwise, λογικοί, συνθήκη, ανάθεση και τέλος κόμμα. Δύο σημεία θέλουν προσοχή: οι bitwise `&`, `^`, `|` έχουν **χαμηλότερη** προτεραιότητα από τα `==` και `!=`, και η ανάθεση έχει χαμηλότερη προτεραιότητα από σχεδόν όλα. Ο κανόνας των διαφανειών: **δεν είμαστε σίγουροι για την ακριβή σειρά**; **Χρησιμοποιούμε παρενθέσεις `()`!**

Το παρακάτω δέντρο δείχνει πώς ομαδοποιεί ο μεταγλωττιστής την έκφραση `x = a + b * c > d && e`:



Σχήμα: οι τελεστές με υψηλότερη προτεραιότητα βρίσκονται πιο βαθιά στο δέντρο και υπολογίζονται πρώτοι.

§5.14 Εκφράσεις και εντολές

Κάθε έγκυρος συνδυασμός τελεστών και τελεστέων (μεταβλητών, σταθερών, κλήσεων συναρτήσεων) λέγεται **έκφραση (expression)**, και κάθε έκφραση έχει μια τιμή. Για παράδειγμα:

```
final_exam * 50 / 100 + homework * 30 / 100 + lab * 20 / 100
```

Μια συνάρτηση αποτελείται από εκφράσεις που συνδυάζονται και εκτελούνται με τον τρόπο που καθορίζουν κάποιες συντακτικές δομές, οι **εντολές (statements)**. Οι εντολές εκτελούνται **διαδοχικά**, **υπό συνθήκη** ή **επανειλημμένα**. Η εντολή

```
if (argc != 4) {
    ....
}
```

για παράδειγμα, εκτελεί το σώμα της μόνο υπό συνθήκη. Η διαφορά είναι ουσιαστική: μια έκφραση *υπολογίζει μια τιμή*, ενώ μια εντολή *κάνει κάτι* (και δεν έχει τιμή).

§5.15 Κατηγορίες εντολών

Οι εντολές της C χωρίζονται σε πέντε κατηγορίες:

1. **Κενές εντολές (empty / null statements)**: το σκέτο `;`, που δεν εκτελεί τίποτα (no-operation ή no-op). Το `;;;;;` είναι πέντε κενές εντολές. Η χρησιμότητά τους φαίνεται σε συνδυασμό με άλλες εντολές.
2. **Εντολές έκφρασης (expression statements)**: μια έκφραση που τελειώνει με semicolon, π.χ. `x = 4;` ή `z = ++y;`. Το semicolon (`;`) είναι διαχωριστικό εντολών: δείχνει πού τελειώνει η εντολή.
3. **Σύνθετες εντολές (compound statements) ή blocks**: μια σειρά από εντολές μέσα σε αγκύλες `{ }`, που μετράει συντακτικά ως μία εντολή.
4. **Εντολές συνθήκης (conditional statements)**: `if`, `if-else`.
5. **Εντολές επανάληψης (iteration statements)**: οι βρόχοι.

Όλες αναλύονται διεξοδικά στο Κεφάλαιο 6.

§5.16 Ροή ελέγχου: οι εντολές if και if-else

Ροή ελέγχου (control flow) είναι η σειρά με την οποία εκτελούνται οι εντολές ενός προγράμματος. Την καθορίζουν οι **εντολές ροής ελέγχου (control flow statements)** και συνήθως την οπτικοποιούμε με ένα **διάγραμμα ροής (flowchart)**.

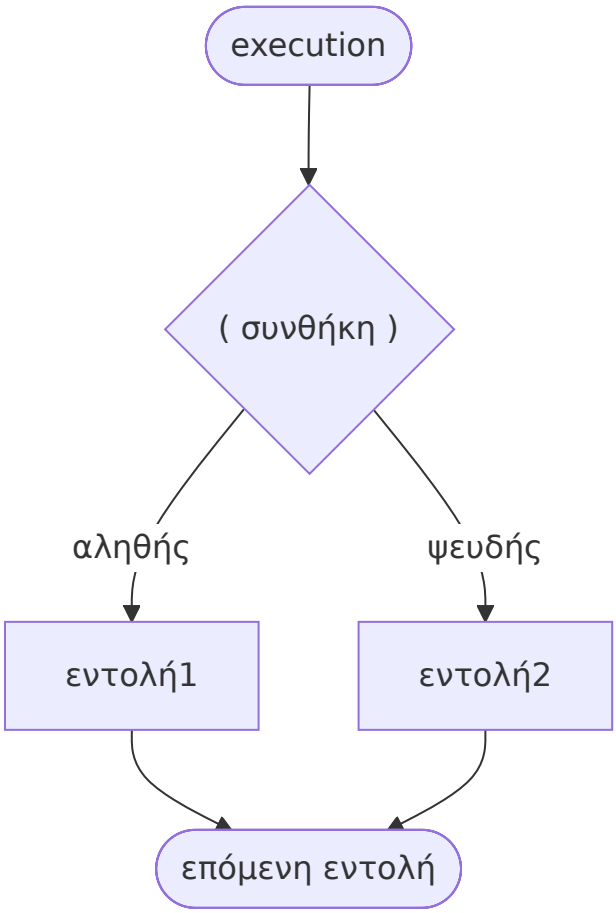
Η **εντολή** `if` εκτελεί μια εντολή μόνο αν μια λογική συνθήκη είναι αληθής, δηλαδή μη μηδενική. Οι παρενθέσεις γύρω από τη συνθήκη είναι **υποχρεωτικές**:

```
if ( συνθήκη )
    εντολή
```

Η «εντολή» μπορεί να είναι οποιασδήποτε κατηγορίας: κενή (`if (year > 1) ;`), έκφρασης (`if (year > 1) year++;`) ή block (`if (year > 1) { year++; }`).

Η **εντολή** `if-else` επεκτείνει την `if` ώστε να εκτελεί μια άλλη εντολή όταν η συνθήκη είναι ψευδής· εκτελείται ακριβώς μία από τις δύο:

```
if ( συνθήκη )
    εντολή1
else
    εντολή2
```



Σχήμα: διάγραμμα ροής της `if-else`*· χωρίς* `else`*, ο κλάδος «ψευδής» πηγαίνει κατευθείαν στην επόμενη εντολή.*

Για πολλές συνθήκες, εμφωλευμένες `if` και το πρόβλημα του dangling `else`, δείτε το Κεφάλαιο 6.

Παραδείγματα

§5.17 Απομόνωση του πιο σημαντικού bit

Εφαρμόζει: bitwise τελεστές. Πώς απομονώνουμε το πιο σημαντικό bit ενός byte; Οι επιλογές είναι `byte & 0xFF`, `byte & 0x80`, `byte | 0xFF` και `byte | 0x80`. Το πιο σημαντικό bit είναι το αριστερότερο, και η μάσκα με 1 μόνο εκεί είναι `1000 0000` = `0x80`. Το `&` μηδενίζει όλα τα bit εκτός από εκείνα της μάσκας, π.χ. `1011 0110 & 1000 0000 = 1000 0000`. Το `byte & 0xFF` επιστρέφει όλο το byte, ενώ οι επιλογές με `|` *ενεργοποιούν* bit αντί να τα απομονώνουν. Σωστή απάντηση: `byte & 0x80`, που είναι μη μηδενικό (αληθές) αν και μόνο αν το πιο σημαντικό bit είναι 1.

§5.18 Συνδυασμός τιμών με bitwise OR

Εφαρμόζει: bitwise OR, δεκαεξαδική γραφή. Ποια η τιμή της `0xbeef | 0xcafe0000`; Γράψτε τους δύο αριθμούς με 8 δεκαεξαδικά ψηφία ο καθένας:

```
  0x0000beef
| 0xcafe0000
  ----------
  0xcafebeef
```

Σε κάθε θέση ο ένας από τους δύο έχει 0, και `x | 0 = x`, οπότε τα δύο κομμάτια απλώς «κολλάνε». Από τις επιλογές `0xffffffff`, `0xcafebeef`, `0xbeefcafe`, `0xfaceb00c` σωστή είναι η `0xcafebeef`. Με αυτόν τον τρόπο (συχνά μαζί με `<<`) συνθέτουμε μια μεγάλη τιμή από μικρότερα κομμάτια.

§5.19 Μια συνάρτηση max με τον τελεστή συνθήκης

Εφαρμόζει: τελεστής συνθήκης, συναρτήσεις. Οι διαφάνειες ρωτούν πώς θα φτιάχναμε μια συνάρτηση `max`. Ο τελεστής συνθήκης την κάνει μία γραμμή:

```c
#include <stdio.h>

int max(int a, int b) {
    return (a > b) ? a : b;
}

int main(void) {
    printf("%d\n", max(42, 8));
    printf("%d\n", max(-5, 16));
    return 0;
}
```

```
$ ./max
42
16
```

§5.20 Cast και διαίρεση: (double)3/4

Εφαρμόζει: cast, σιωπηρή μετατροπή, προτεραιότητα. Πόσο κάνει `(double)3/4`; Το cast έχει μεγαλύτερη προτεραιότητα από τη διαίρεση, άρα η έκφραση σημαίνει `((double)3) / 4`. Το `3` γίνεται `3.0`, και στη μικτή πράξη `3.0 / 4` το `4` μετατρέπεται σιωπηρά σε `4.0`. Αποτέλεσμα: `0.75`. Συγκρίνετε:

Έκφραση	Τιμή	Γιατί
`3 / 4`	`0`	ακέραια διαίρεση
`(double)3 / 4`	`0.75`	ο αριθμητής είναι `double`

Έκφραση	Τιμή	Γιατί
`3 / (double)4`	`0.75`	ο παρονομαστής είναι `double`
`(double)(3 / 4)`	`0.0`	η ακέραια διαίρεση έγινε πρώτα

§5.21 Πρόγραμμα υπολογισμού βαθμολογίας

Εφαρμόζει: αριθμητικοί τελεστές, ακέραια διαίρεση, προτεραιότητα, `if`. Το πρόγραμμα των διαφανειών διαβάζει τρεις βαθμούς από τη γραμμή εντολών (με `atoi`, που μετατρέπει ένα string σε `int`) και υπολογίζει τον τελικό βαθμό με βάρη 50%, 30% και 20%. Οι διαφάνειες δείχνουν μόνο τον κώδικα των συναρτήσεων· εδώ έχουν προστεθεί τα `#include`.

```c
#include <stdio.h>
#include <stdlib.h>

int grade(int final_exam, int homework, int lab) {
    return final_exam * 50 / 100 + homework * 30 / 100 + lab * 20 / 100;
}

int main(int argc, char **argv) {
    if (argc != 4) {
        printf("Run as: grade final_exam homework lab\n");
        return 1;
    }
    int final_exam = atoi(argv[1]);
    int homework = atoi(argv[2]);
    int lab = atoi(argv[3]);
    int final_grade = grade(final_exam, homework, lab);
    printf("My grade is: %d\n", final_grade);
    return 0;
}
```

Η `if (argc != 4)` είναι μια εντολή συνθήκης: αν δεν δόθηκαν ακριβώς τρία ορίσματα (συν το όνομα του προγράμματος), τυπώνει οδηγίες και τερματίζει με κωδικό 1.

Η έκφραση στη `grade` ομαδοποιείται ως `((final_exam * 50) / 100) + ((homework * 30) / 100) + ((lab * 20) / 100)`: `*` και `/` πριν από το `+`, και από αριστερά προς τα δεξιά μεταξύ τους. Επειδή όλα είναι `int`, **κάθε** διαίρεση κόβει το δεκαδικό μέρος της. Μερικές εκτελέσεις:

```
$ ./grade
Run as: grade final_exam homework lab
$ ./grade 100 100 100
My grade is: 100
$ ./grade 85 75 95
```

```
My grade is: 83
$ ./grade 9 9 9
My grade is: 7
```

Με 85, 75 και 95 ο σωστός βαθμός είναι $42.5 + 22.5 + 19 = 84$, αλλά το πρόγραμμα βγάζει 83, γιατί το `42.5` και το `22.5` έγιναν 42 και 22. Με τρία 9 ο βαθμός θα έπρεπε να είναι 9, αλλά βγαίνει `4 + 2 + 1 = 7`. Αν αντίθετα γράφαμε `final_exam * (50 / 100)`, το `50 / 100` θα ήταν `0` και ο βαθμός θα μηδενιζόταν. Για ακριβές αποτέλεσμα θα κάναμε τους υπολογισμούς σε `double` (π.χ. `final_exam * 0.5`) και θα στρογγυλοποιούσαμε μόνο στο τέλος.

§5.22 Προτεραιότητα στην πράξη

Εφαρμόζει: προτεραιότητα, προσεταιριστικότητα. Οι δύο εκφράσεις των διαφανειών:

- `5 * 6 + 3 * 4`: οι πολλαπλασιασμοί πρώτα, `30 + 12`, άρα 42.
- `90 * 50 / 100`: ίδια προτεραιότητα, από αριστερά προς τα δεξιά, `4500 / 100`, άρα 45. Με την αντίθετη σειρά, `90 * (50 / 100)` = `90 * 0` = 0.

Αυτή ακριβώς η σειρά κάνει τη συνάρτηση `grade` να δίνει λογικά αποτελέσματα.

§5.23 Ιχνηλάτηση εντολών έκφρασης

Εφαρμόζει: αύξηση και μείωση, εντολές έκφρασης. Οι διαφάνειες ρωτούν ποιες είναι οι τιμές των `x`, `y`, `z` μετά τις εντολές `x = 4; y = 7; z = ++y; y = z - (x++); z = x - (--y);` (που δείχνουν και κλεισμένες σε `{ }`, ως σύνθετη εντολή). Φτιάξτε πίνακα με τις τρεις τιμές μετά από κάθε εντολή, θυμούμενοι ότι το `++y` δίνει τη *νέα* τιμή και το `x++` την *παλιά*· η απάντηση είναι στην ερώτηση αυτοαξιολόγησης 6.

§5.24 Μέγιστο με if-else

Εφαρμόζει: `if-else`*, τελεστής συνθήκης.* Το κλασικό παράδειγμα:

```
if (x > y)
    max = x;
else
    max = y;
```

Οι διαφάνειες ρωτούν: θα μπορούσαμε να «ζήσουμε» χωρίς `else`; Υπάρχει εναλλακτικός τρόπος να γράψουμε το παραπάνω; Τι κάνουμε όταν έχουμε πάνω από δύο συνθήκες; Χωρίς `else`, δίνουμε πρώτα στο `max` μια αρχική τιμή (`max = y;`) και τη διορθώνουμε με απλή `if`. Εναλλακτικά, ο τελεστής συνθήκης το κάνει σε μία έκφραση: `max = (x > y) ? x : y;`. Για περισσότερες συνθήκες χρειαζόμαστε εμφωλευμένες `if`, το θέμα του Κεφαλαίου 6.

Κύρια σημεία

1. Οι τελεστές κατατάσσονται σε μοναδιαίους, δυαδικούς και τριαδικούς, και σε οικογένειες: αριθμητικούς, συγκριτικούς, λογικούς, συνθήκης, bitwise, μετατροπής και ανάθεσης.
2. Η διαίρεση μεταξύ ακεραίων δίνει ακέραιο πηλίκο: `85 / 2` είναι `42` και `3 / 4` είναι `0`· ο `%` δίνει το υπόλοιπο.
3. Οι συγκριτικοί και οι λογικοί τελεστές δίνουν `0` (ψευδές) ή `1` (αληθές).
4. Στην C το μηδέν σημαίνει «ψευδές» και κάθε μη μηδενική τιμή «αληθές»· το `1`, το `42` και το `-5` είναι εξίσου αληθή.
5. Οι bitwise τελεστές δουλεύουν σε κάθε bit ακεραίων: `&` για απομόνωση με μάσκα, `|` για ενεργοποίηση, `^` για αντιστροφή, `~` για άρνηση· ολίσθηση κατά N αριστερά/δεξιά ισοδυναμεί με πολλαπλασιασμό/διαίρεση με 2^N.
6. Ο τελεστής συνθήκης `συνθήκη ? τιμή1 : τιμή2` είναι ο μόνος τριαδικός τελεστής και, ως έκφραση, μπορεί να μπει όπου χρειάζεται μια τιμή.
7. Το cast `(τύπος)τελεστέος` κάνει ρητή μετατροπή τύπου· από πραγματικό σε ακέραιο τα δεκαδικά αποκόπτονται, δεν στρογγυλοποιούνται.
8. Σε μικτές πράξεις ο μεταγλωττιστής μετατρέπει σιωπηρά τον «μικρότερο» τύπο στον «μεγαλύτερο», ξεχωριστά για κάθε τελεστή.
9. Η ανάθεση αποθηκεύει την τιμή του δεξιού μέλους (rvalue) στη μεταβλητή του αριστερού (lvalue) και επιστρέφει την τιμή, γι' αυτό γράφεται `x = y = 42`.
10. Το `a op= b` είναι συντομογραφία του `a = a op b`.
11. Το `a++` επιστρέφει την τιμή πριν την αύξηση, το `++a` την τιμή μετά την αύξηση· και τα δύο αυξάνουν το `a`.
12. Ο τελεστής παράθεσης υπολογίζει τον πρώτο τελεστέο, τον αγνοεί και επιστρέφει τον δεύτερο.
13. Η προτεραιότητα καθορίζει ποιος τελεστής εφαρμόζεται πρώτος· όταν είναι ίδια, την σειρά την ορίζει η προσεταιριστικότητα.
14. Δεν είμαστε σίγουροι για την ακριβή σειρά; Χρησιμοποιούμε παρενθέσεις `()`.
15. Έκφραση είναι κάθε έγκυρος συνδυασμός τελεστών και τελεστέων· οι εντολές συνδυάζουν εκφράσεις και εκτελούνται διαδοχικά, υπό συνθήκη ή επανειλημμένα.
16. Οι εντολές είναι κενές, έκφρασης, σύνθετες, συνθήκης ή επανάληψης· το `;` δείχνει πού τελειώνει μια εντολή.
17. Η `if` εκτελεί μια εντολή αν η συνθήκη είναι αληθής, η `if-else` επιλέγει μία από δύο· οι παρενθέσεις γύρω από τη συνθήκη είναι υποχρεωτικές.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
τελεστής	operator	Σύμβολο που εφαρμόζεται σε τιμές και δίνει αποτέλεσμα

Ελληνικά	English	Σύντομος ορισμός
τελεστέος	operand	Τιμή στην οποία εφαρμόζεται ένας τελεστής
μοναδιαίος / δυαδικός / τριαδικός	unary / binary / ternary	Τελεστής με έναν / δύο / τρεις τελεστέους
ακέραια διαίρεση	integer division	`/` μεταξύ ακεραίων, που κρατάει μόνο το πηλίκο
συγκριτικός τελεστής	comparison / relational operator	`==`, `!=`, `<`, `>`, `<=`, `>=`
λογικός τελεστής	logical operator	`&&`, `||`, `!`
bitwise τελεστής	bitwise operator	`&`, `|`, `^`, `~`, `<<`, `>>`, bit προς bit
τελεστής συνθήκης	conditional operator	`σ ? α : β`, ο μόνος τριαδικός τελεστής
τελεστής μετατροπής	cast operator	`(τύπος)x`, ρητή μετατροπή τύπου
σιωπηρή μετατροπή τύπων	implicit type conversion	Αυτόματη μετατροπή σε μικτές πράξεις
τελεστής ανάθεσης	assignment operator	`=`: αποθηκεύει και επιστρέφει μια τιμή
συνδυαστικός τελεστής ανάθεσης	compound assignment	`+=`, `-=`, …
τελεστής αύξησης / μείωσης	increment / decrement operator	`++` / `--`
προθεματικός / επιθεματικός	prefix / postfix	Πριν / μετά τη μεταβλητή (`++a` / `a++`)
τελεστής παράθεσης	comma operator	`a, b`: υπολογίζει το `a`, επιστρέφει το `b`
προτεραιότητα	precedence	Ποιος τελεστής εφαρμόζεται πρώτος
προσεταιριστικότητα	associativity	Σειρά για τελεστές ίδιας προτεραιότητας
έκφραση	expression	Συνδυασμός τελεστών και τελεστέων με τιμή
εντολή	statement	Συντακτική δομή που εκτελείται
ροή ελέγχου	control flow	Η σειρά εκτέλεσης των εντολών

Διάβασμα

- **Διαφάνειες**: Διάλεξη 5, σελ. 1–35. Κατηγορίες τελεστών: σελ. 5· αριθμητικοί, συγκριτικοί, λογικοί: σελ. 7–9· bitwise: σελ. 10–12· συνθήκης και cast: σελ. 13–15· ανάθεση, αύξηση/μείωση, κόμμα: σελ. 16–19· πρόγραμμα βαθμολογίας: σελ. 20· προτεραιότητα: σελ. 21–24· εκφράσεις και εντολές: σελ. 25–28· `if` και `if-else`: σελ.

29–33.

- **Σημειώσεις**: Κεφάλαιο 2: Μεταβλητές, τύποι, τελεστές και παραστάσεις, ενότητα «Εντολές αντικατάστασης, τελεστές και παραστάσεις» (K04, σελ. 34–43)· Κεφάλαιο 3: Η ροή του ελέγχου, ενότητες «Η ροή του ελέγχου στην C», «Εντολή `if`» (K04, σελ. 46–49). Οι διαφάνειες συνιστούν να έχετε καλύψει τις σημειώσεις του κ. Σταματόπουλου μέχρι και τη σελίδα 62.
- **Εργαστήριο**: Εργαστήριο 3: ασκήσεις `root.c` (`if...else`), `birthdate.c` (ακέραια διαίρεση και `%`), `limit.c` (αποσφαλμάτωση εκφράσεων)
- **Βιβλίο**: K&R, §2.5–2.12 (τελεστές και παραστάσεις), όπως προτείνουν οι σημειώσεις.
- **Άλλα**: Wikipedia: Comma operator, Control flow, Conditional (computer programming)

Συχνά λάθη

- **Ακέραια διαίρεση εκεί που θέλατε δεκαδικά**. `double avg = sum / n;` με `int sum, n` δίνει στρογγυλεμένο προς τα κάτω αποτέλεσμα (π.χ. `2.000000` αντί `2.500000`). Διόρθωση: `(double)sum / n`.
- **Cast στη λάθος θέση**. `(double)(3 / 4)` είναι `0.0`, γιατί η διαίρεση έγινε πριν από το cast. Διόρθωση: κάντε cast σε έναν τελεστέο, `(double)3 / 4`.
- **Περιμένετε στρογγυλοποίηση από το cast**. `(int)2.99` είναι 2, όχι 3.
- **`=` αντί για `==`**. `if (x = 0)` αναθέτει 0 και η συνθήκη είναι πάντα ψευδής· ο gcc με `-Wall` προειδοποιεί `suggest parentheses around assignment used as truth value`. Διόρθωση: `if (x == 0)`.
- **`&` αντί για `&&` (ή `|` αντί για `||`)**. `if (a & b)` με `a = 1, b = 2` είναι ψευδές (`01 & 10 = 0`), ενώ `a && b` είναι αληθές. Χρησιμοποιήστε τα λογικά για συνθήκες.
- **Bitwise με σύγκριση χωρίς παρενθέσεις**. `if (x & 1 == 0)` σημαίνει `x & (1 == 0)`, δηλαδή πάντα 0, γιατί το `==` έχει υψηλότερη προτεραιότητα από το `&`. Διόρθωση: `if ((x & 1) == 0)`.
- **Σύγχυση `a++` και `++a`**. `b = a++;` δίνει στο `b` την *παλιά* τιμή. Αν σας ενδιαφέρει η τιμή, ιχνηλατήστε με πίνακα.
- **Διπλή αλλαγή της ίδιας μεταβλητής σε μία έκφραση**. `i = i++ + 1;` ή `printf("%d %d", i++, i++);` έχουν απροσδιόριστη συμπεριφορά· ο gcc με `-Wall` λέει `operation on 'i' may be undefined`. Διόρθωση: σπάστε το σε δύο εντολές.
- **Κόμμα αντί για παρενθέσεις**. `x = 12, 42;` αφήνει `x = 12`. Αν θέλετε 42, γράψτε `x = (12, 42);`, ή καλύτερα απλώς `x = 42;`.
- **Ξεχασμένες παρενθέσεις στην `if`**. `if x > 1` δεν μεταγλωττίζεται.

Ερωτήσεις κατανόησης

- **E5.1** Τι δίνουν οι εκφράσεις `85 / 2`, `85.0 / 2`, `85 % 2` και `-7 / 2`;[44]

[44] `42` (ακέραια διαίρεση), `42.5` (σιωπηρή μετατροπή σε `double`), `1`, `-3` (το πηλίκο κόβεται προς το μηδέν).

- **E5.2** Ποια από τα `0`, `1`, `42`, `-5` θεωρούνται «αληθή» σε μια συνθήκη της C;[45]
- **E5.3** Πόσο κάνει `0xF0 | 0x0F`, `0xF0 & 0x0F` και `3 << 2`;[46]
- **E5.4** Γιατί το `(int)42.67` δίνει `42` και όχι `43`;[47]
- **E5.5** Ποια η τιμή και ο τύπος της `40 + 2.0`;[48]
- **E5.6** Ποιες είναι οι τιμές των `x`, `y`, `z` μετά τις εντολές `x = 4; y = 7; z = ++y; y = z - (x++); z = x - (--y);`;[49]
- **E5.7** Γιατί μπορούμε να γράψουμε `x = y = 42`; Με ποια σειρά γίνονται οι αναθέσεις;[50]
- **E5.8** Πόσο κάνουν `5 * 6 + 3 * 4` και `90 * 50 / 100`;[51]
- **E5.9** Τι σημαίνει `if (x & 1 == 0)` και πώς το διορθώνουμε;[52]
- **E5.10** Ποια η διαφορά ανάμεσα σε έκφραση και εντολή;[53]

Kahoot από το αμφιθέατρο (K5.1–K5.3)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K5.1** Μετατροπή double σε int: 86% σωστές απαντήσεις
- **K5.2** Ποιος αριθμός ικανοποιεί τη συνθήκη: 58% σωστές απαντήσεις
- **K5.3** x++ και ++y: 48% σωστές απαντήσεις

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A5.1–A5.7)

- **A5.1** Η τιμή της 0xbeef | 0xcafe0000: Διάλεξη 5: Τελεστές και Εντολές, διαφάνεια 12 · ★☆☆ · multiple-choice · `slides-lec05-bitwise-or-hex`
- **A5.2** Cast και διαίρεση: (double)3/4: Διάλεξη 5: Τελεστές και Εντολές, διαφάνεια 14 · ★☆☆ · trace · `slides-lec05-cast-division`
- **A5.3** Απομόνωση του πιο σημαντικού bit: Διάλεξη 5: Τελεστές και Εντολές, διαφάνεια 11 · ★☆☆ · multiple-choice · `slides-lec05-msb`
- **A5.4** Τι τύπου τελεστές είναι;: Διάλεξη 5: Τελεστές και Εντολές, διαφάνειες 7-10, 13 · ★☆☆ · short-answer · `slides-lec05-operator-arity`

[45] Τα `1`, `42` και `-5`: κάθε μη μηδενική τιμή είναι αληθής· μόνο το `0` είναι ψευδές.

[46] `0xFF`, `0x00` και `12` (ολίσθηση κατά 2 = πολλαπλασιασμός επί 4).

[47] Η μετατροπή από πραγματικό σε ακέραιο αποκόπτει τα δεκαδικά ψηφία, δεν στρογγυλοποιεί.

[48] `42.0`, τύπου `double`: το `40` μετατρέπεται σιωπηρά σε `40.0`.

[49] `z = ++y` κάνει `y = 8`, `z = 8`· `y = z - (x++)` κάνει `y = 4`, `x = 5`· `z = x - (--y)` κάνει `y = 3`, `z = 2`. Τελικά `x = 5`, `y = 3`, `z = 2`.

[50] Η ανάθεση επιστρέφει την τιμή που ανέθεσε και έχει προσεταιριστικότητα από δεξιά προς τα αριστερά: πρώτα `y = 42`, και η τιμή του (`42`) ανατίθεται στο `x`.

[51] `42` (οι πολλαπλασιασμοί πρώτα) και `45` (από αριστερά προς τα δεξιά: `4500 / 100`).

[52] Σημαίνει `x & (1 == 0)`, δηλαδή `x & 0`, που είναι πάντα `0`, γιατί το `==` έχει υψηλότερη προτεραιότητα από το `&`. Σωστά: `if ((x & 1) == 0)`.

[53] Η έκφραση είναι συνδυασμός τελεστών και τελεστέων που υπολογίζει μια τιμή· η εντολή είναι μονάδα εκτέλεσης (π.χ. μια έκφραση με `;`, ένα block, μια `if`) που δεν έχει τιμή.

- **A5.5** Οι αριθμητικοί τελεστές ως συναρτήσεις: Διάλεξη 5: Τελεστές και Εντολές, διαφάνεια 7 · ★☆☆ · short-answer · `slides-lec05-operator-as-function`
- **A5.6** Προτεραιότητα και προσεταιριστικότητα: Διάλεξη 5: Τελεστές και Εντολές, διαφάνεια 21 · ★☆☆ · trace · `slides-lec05-precedence`
- **A5.7** Συνάρτηση max με τον τελεστή συνθήκης: Διάλεξη 5: Τελεστές και Εντολές, διαφάνεια 13 · ★☆☆ · programming · `slides-lec05-ternary-max`

Εργαστήριο (A5.8)

- **A5.8** Υπολογισμός ημέρας μίας δεδομένης ημερομηνίας: Εργαστήριο 3, Άσκηση 3 · ★☆☆ · programming · `lab-lab03-birthdate`

Θέματα εξετάσεων (A5.9–A5.11)

- **A5.9** Mystery: Εξέταση Ιανουαρίου 2025, Θέμα 1 · ★☆☆ · trace · `exam-2025-jan-q1`
- **A5.10** Mystery: Εξέταση Ιανουαρίου 2026, Θέμα 1 · ★☆☆ · trace · `exam-2026-jan-q1`
- **A5.11** Η συνάρτηση mystery: Εξέταση Ιουνίου 2026, Θέμα 1 · ★☆☆ · trace · `exam-2026-jun-q1`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A2.12** Rickroll Τριπλέτες: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #4 (Rick Astley Themed), Θέμα 3 · ★★★ · programming · `exam-2023-fall-ex4-q3`
- **A3.7** Πυθαγόρειο θεώρημα: Εργαστήριο 2, Άσκηση 2 · ★☆☆ · programming · `lab-lab02-pyth`
- **A6.12** Υπολογισμός ριζών τριωνύμου: Εργαστήριο 3, Άσκηση 2 · ★★☆ · programming · `lab-lab03-root`
- **A6.2** Τιμές μετά από εντολές έκφρασης: Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 7 · ★☆☆ · trace · `slides-lec06-expression-statements`
- **A6.5** Μέγιστο με if-else και εναλλακτικές: Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 13 · ★☆☆ · short-answer · `slides-lec06-if-else-max`
- **A9.23** Μήνυμα από τον Καίσαρα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #6 (Crypto Themed), Θέμα 2 · ★☆☆ · programming · `exam-2023-fall-ex6-q2`
- **A9.31** Μετρητής λέξεων: Εξέταση Σεπτεμβρίου 2024, Θέμα 4 · ★★☆ · programming · `exam-2024-sep-q4`
- **A9.17** Επεξεργασία Ήχου (soundwave): Εργασία 1 (2025-26), Άσκηση 1 · ★★★ · programming · `hw-2025-hw1-soundwave`
- **A14.8** Η έκφραση *str++: Διάλεξη 14, διαφάνεια 34 · ★★☆ · short-answer · `slides-lec14-str-plus-plus`
- **A15.3** Πολυπλοκότητα υπολογισμού βαθμολογίας: Διάλεξη 15, διαφάνεια 21 · ★☆☆ · short-answer · `slides-lec15-complexity-grade`
- **A18.12** Κρυφό Μήνυμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #7 (Star Wars Themed), Θέμα 2 · ★☆☆ · programming · `exam-2023-fall-ex7-q2`

Κεφάλαιο 6: Εντολές και Ροή Ελέγχου

> **Στόχοι**: μετά από αυτό το κεφάλαιο θα μπορείτε να κατατάσσετε κάθε εντολή της C σε μία από πέντε κατηγορίες, να γράφετε `if`, `if-else` και αλυσίδες `else if`, να αποφεύγετε το dangling `else`, να γράφετε βρόχους `while`, `for` και `do-while`, να προβλέπετε πόσες φορές εκτελείται το σώμα τους και να διαβάζετε το διάγραμμα ροής κάθε εντολής ελέγχου.
>
> **Προαπαιτούμενα**: Κεφάλαιο 2, Κεφάλαιο 5
>
> **Χρόνος μελέτης**: ~2 ώρες

Σύνοψη

Η διάλεξη κατατάσσει τις εντολές της C σε πέντε κατηγορίες (κενές, έκφρασης, σύνθετες, συνθήκης, επανάληψης) και περνάει στη **ροή ελέγχου**: στις εντολές που αποφασίζουν *ποια* εντολή θα εκτελεστεί και *πόσες φορές*. Βλέπουμε την `if` και την `if-else`, τις εμφωλευμένες `if` και την παγίδα του dangling `else`, και τους τρεις βρόχους `while`, `for` και `do-while`, ο καθένας με το διάγραμμα ροής του. Με αυτά γράφουμε προγράμματα που παίρνουν αποφάσεις και επαναλαμβάνουν υπολογισμούς.

Θεωρία

§6.1 Κατηγορίες εντολών

Κάθε **εντολή (statement)** της C ανήκει σε μία από πέντε κατηγορίες:

#	Κατηγορία	English	Παράδειγμα
1	Κενές εντολές	null / empty statements	`;`
2	Εντολές έκφρασης	expression statements	`x = 4;`
3	Σύνθετες εντολές	compound statements / blocks	`{ x = 4; y = 7; }`
4	Εντολές συνθήκης	conditional statements	`if (x > y) max = x;`
5	Εντολές επανάληψης	iteration statements / loops	`while (i < 42) i++;`

Το semicolon (`;`) είναι το διαχωριστικό εντολών: δείχνει πού τελειώνει μια εντολή. Οι κατηγορίες 4 και 5 είναι οι εντολές ροής ελέγχου.

§6.2 Κενή εντολή, εντολή έκφρασης, σύνθετη εντολή

Η **κενή εντολή (empty / null statement)** είναι ένα σκέτο `;` και δεν κάνει τίποτα (**no-op**)· το `;;;;;` είναι πέντε κενές εντολές. Χρησιμεύει (και κρύβει παγίδες) όταν γίνεται σώμα μιας `if` ή ενός βρόχου.

Εντολή έκφρασης (expression statement) είναι μια έκφραση που τελειώνει με semicolon, π.χ. οι αναθέσεις, οι αυξήσεις και οι κλήσεις συναρτήσεων: `x = 4;`, `z = ++y;`.

Σύνθετη εντολή (compound statement) ή **block** είναι μια σειρά εντολών μέσα σε αγκύλες `{ }`. Συντακτικά ένα block μετράει ως **μία** εντολή, οπότε όπου η C ζητάει «μια εντολή» (π.χ. ως σώμα μιας `if`) μπορούμε να βάλουμε όσες θέλουμε μέσα σε αγκύλες. Μετά το `}` δεν χρειάζεται `;`, και το άδειο block `{ }` κάνει ό,τι η κενή εντολή.

§6.3 Ροή ελέγχου

Ροή ελέγχου (control flow) είναι η σειρά με την οποία εκτελούνται οι εντολές ενός προγράμματος. Την καθορίζουν οι **εντολές ροής ελέγχου (control flow statements)**, που παραλείπουν εντολές (συνθήκη) ή τις επαναλαμβάνουν (βρόχοι). Συνήθως την οπτικοποιούμε με ένα **διάγραμμα ροής (flowchart)**: ένας ρόμβος είναι ο έλεγχος μιας συνθήκης με δύο εξόδους, «αληθής» και «ψευδής», και ένα ορθογώνιο είναι μια εντολή.

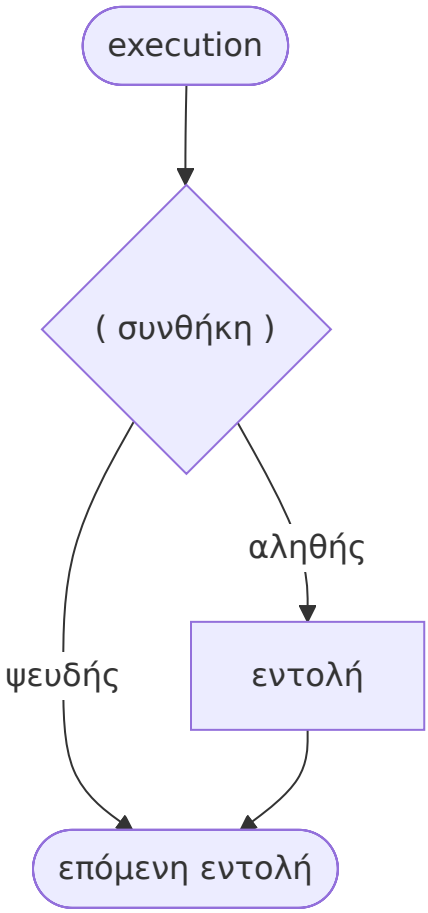
Μια συνθήκη στην C είναι απλώς μια έκφραση (Κεφάλαιο 5): τιμή **μηδέν** σημαίνει **ψευδής**, **οποιαδήποτε άλλη** τιμή σημαίνει **αληθής**. Γι' αυτό γράφουμε `while (1)` ή `if (n)`.

§6.4 Η εντολή if

Η **εντολή** `if` εκτελεί μια εντολή μόνο αν μια λογική συνθήκη είναι αληθής:

```
if ( συνθήκη )
    εντολή
```

Οι παρενθέσεις γύρω από τη συνθήκη είναι **υποχρεωτικές**. Η εντολή μπορεί να είναι οποιασδήποτε κατηγορίας: κενή (`if (year > 1) ;`, που δεν κάνει τίποτα), εντολή έκφρασης (`if (year > 1) year++;`) ή block (`if (year > 1) { year++; }`). Οι δύο τελευταίες κάνουν το ίδιο, αλλά με αγκύλες μια δεύτερη εντολή που θα προσθέσετε αργότερα μπαίνει σίγουρα μέσα στην `if`.

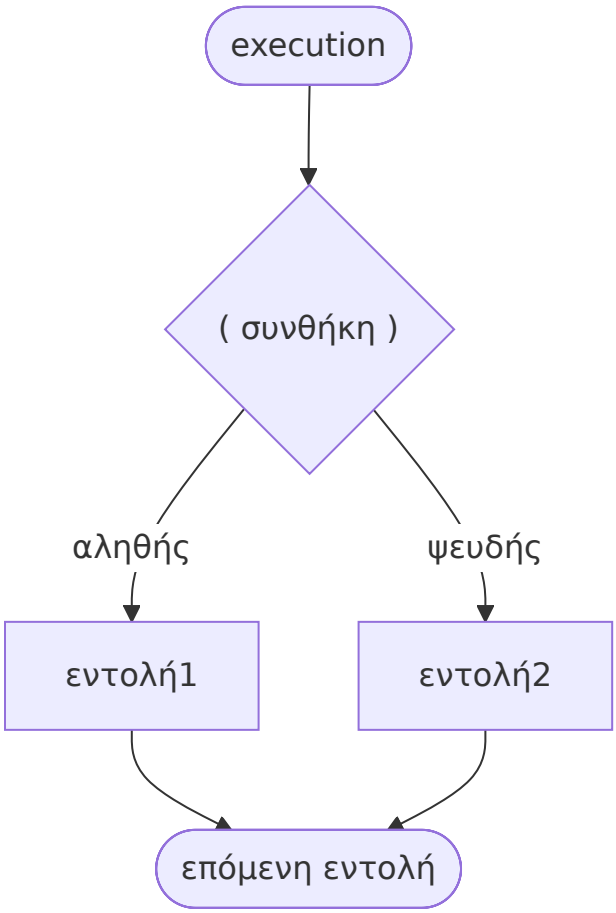


Σχήμα: διάγραμμα ροής της `if`.

§6.5 Η εντολή if-else

Η **εντολή** `if-else` εκτελεί μια *άλλη* εντολή όταν η συνθήκη είναι ψευδής. Εκτελείται ακριβώς μία από τις δύο, ποτέ και οι δύο και ποτέ καμία:

```
if ( συνθήκη )
    εντολή1
else
    εντολή2
```



Σχήμα: διάγραμμα ροής της `if-else`.

Το `else` δεν είναι απαραίτητο (το ίδιο αποτέλεσμα βγαίνει με μια αρχική ανάθεση και μια απλή `if`), αλλά δείχνει καθαρά ότι οι δύο περιπτώσεις αποκλείονται. Για απλές επιλογές τιμής υπάρχει και ο τελεστής συνθήκης `? :` του Κεφαλαίου 5. Βλ. το παράδειγμα Μέγιστο δύο αριθμών.

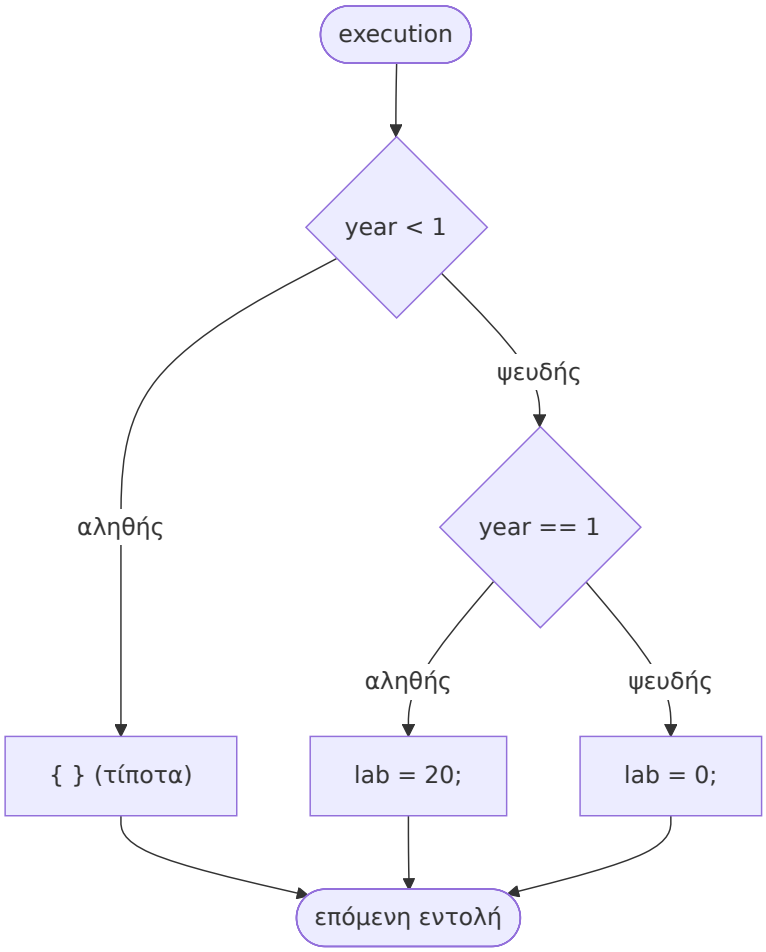
§6.6 Εμφωλευμένες if και η αλυσίδα else if

Όταν έχουμε πάνω από δύο περιπτώσεις, χρησιμοποιούμε **εμφωλευμένες εντολές `if` (nested if)**: το σώμα μιας `if` ή ενός `else` είναι άλλη `if`. Το `else if` δεν είναι ξεχωριστή λέξη-κλειδί· είναι ένα `else` του οποίου η εντολή είναι μια νέα `if`. Στο παράδειγμα των διαφανειών:

```
lab = -1;
if (year < 1)
    { /* empty block */ }
else if (year == 1)
    lab = 20;
```

```
else
    lab = 0;
```

Οι συνθήκες ελέγχονται με τη σειρά και εκτελείται το σώμα της *πρώτης* αληθούς: για `year < 1` το `lab` μένει `-1` (άδειο block), για `year == 1` γίνεται `20`, αλλιώς `0`.



Σχήμα: η αλυσίδα `else if` *ως δύο εμφωλευμένες* `if-else`.

§6.7 Το πρόβλημα του dangling else

Οι εμφωλευμένες `if` χωρίς αγκύλες μπορεί να είναι **αμφίσημες (ambiguous)**: σε ποια `if` ανήκει ένα `else`; Ο μεταγλωττιστής **αγνοεί τη στοίχιση** και ακολουθεί έναν κανόνα: **κάθε** `else` **ανήκει στην πλησιέστερη προηγούμενη** `if` **που δεν έχει ήδη** `else`. Έτσι, στο

```
if (year <= 1)
    if (year == 1)
        lab = 20;
else
```

```
    lab = 0;
```

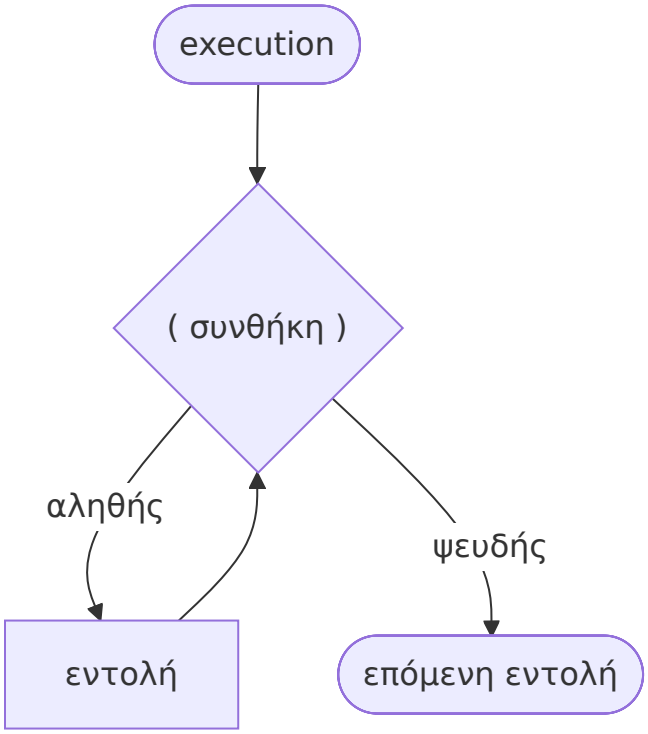
το `else` ανήκει στην εσωτερική `if (year == 1)`, παρόλο που η στοίχιση λέει το αντίθετο. Αυτό είναι το πρόβλημα του **dangling else**. Η λύση των διαφανειών: **αν δεν είμαστε σίγουροι, χρησιμοποιούμε αγκύλες `{ }` για να υποδείξουμε τα όρια των σύνθετων εντολών**. Ο gcc με `-Wall` προειδοποιεί: `suggest explicit braces to avoid ambiguous 'else' [-Wdangling-else]`. Βλ. το παράδειγμα Είναι τα δύο προγράμματα ισοδύναμα;.

§6.8 Βρόχοι: η εντολή while

Οι **βρόχοι (loops)** ή δομές επανάληψης επαναλαμβάνουν μια εντολή, το **σώμα** του βρόχου· κάθε εκτέλεσή του λέγεται **επανάληψη (iteration)**. Η **εντολή** `while` επαναλαμβάνει το σώμα **όσο** η συνθήκη είναι αληθής:

```
while ( συνθήκη )
    εντολή
```

Η συνθήκη ελέγχεται **πριν** από κάθε επανάληψη, οπότε αν είναι ψευδής από την αρχή το σώμα δεν εκτελείται **καμία** φορά. Για να τελειώσει ο βρόχος, το σώμα πρέπει να αλλάζει κάτι από το οποίο εξαρτάται η συνθήκη· αλλιώς έχουμε **ατέρμονα βρόχο (infinite loop)**.



Σχήμα: διάγραμμα ροής της `while`*· ο έλεγχος γίνεται πριν από το σώμα.*

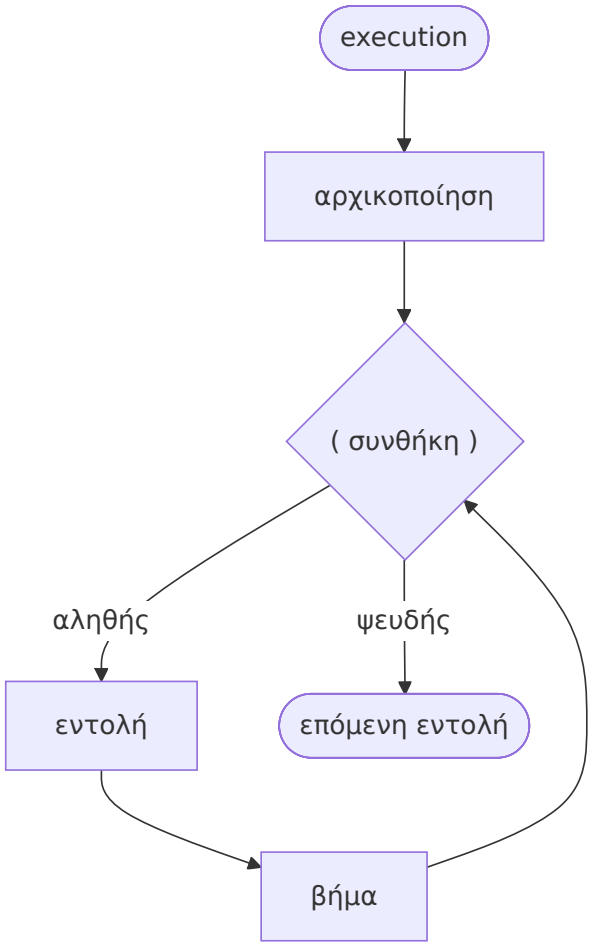
§6.9 Η εντολή for

Η **εντολή** `for` αρχικοποιεί μεταβλητές, επαναλαμβάνει μια εντολή όσο η συνθήκη είναι αληθής, και στο τέλος κάθε επανάληψης εκτελεί το **βήμα**:

```
for ( αρχικοποίηση ; συνθήκη ; βήμα )
    εντολή
```

```
αρχικοποίηση;
while ( συνθήκη ) {
    εντολή
    βήμα;
}
```

Η αρχικοποίηση εκτελείται **μία φορά**· μετά ελέγχεται η συνθήκη, εκτελείται το σώμα, εκτελείται το βήμα, και ξανά από τον έλεγχο. Όπως δείχνει η δεξιά στήλη, η `for` είναι απλώς πιο συμπαγής γραφή της `while`.



Σχήμα: διάγραμμα ροής της `for`.

Και τα τρία μέρη είναι προαιρετικά, αλλά τα δύο `;` μένουν πάντα. Συνθήκη που λείπει θεωρείται πάντα αληθής, άρα το `for (;;)` είναι η κλασική γραφή του ατέρμονα βρόχου. Ο βρόχος `for`

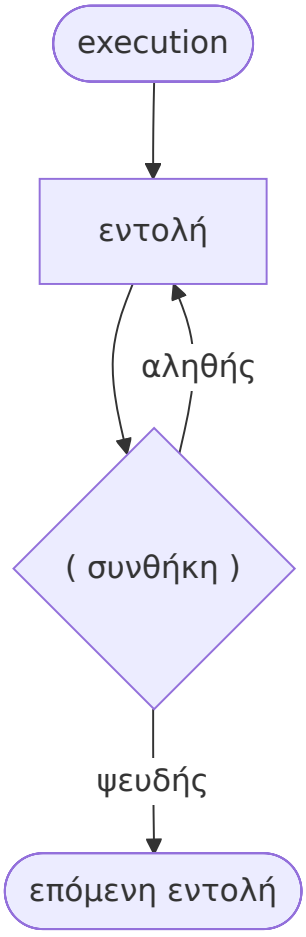
`(i = 0; i < N; i++)` εκτελείται ακριβώς `N` φορές, για `i = 0, 1, …, N-1`.

§6.10 Η εντολή do-while

Η **εντολή** `do-while` εκτελεί πρώτα μία φορά την εντολή και μετά ελέγχει τη συνθήκη για το αν χρειάζεται άλλη επανάληψη:

```
do
    εντολή
while ( συνθήκη );
```

Επειδή ο έλεγχος γίνεται στο τέλος, το σώμα εκτελείται **τουλάχιστον μία φορά** (π.χ. ζητάμε μια τιμή από τον χρήστη μέχρι να είναι έγκυρη). Το τελικό `;` είναι **υποχρεωτικό**.



Σχήμα: διάγραμμα ροής της `do-while`*· το σώμα προηγείται του ελέγχου.*

§6.11 Σύγκριση των τριών βρόχων

	`while`	`for`	`do-while`
Έλεγχος συνθήκης	πριν από κάθε επανάληψη	πριν από κάθε επανάληψη	μετά από κάθε επανάληψη
Ελάχιστες εκτελέσεις σώματος	0	0	1
Αρχικοποίηση / βήμα	; μετά τη συνθήκη	στην επικεφαλίδα	χωριστά
Κενό σώμα	λάθος	λάθος	υποχρεωτικό
Ατέρμονας βρόχος	`while (1)`	`for (;;)`	`do … while (1);`

Μετρητής με γνωστό εύρος: `for`· «όσο ισχύει κάτι»: `while`· «τουλάχιστον μία φορά»: `do-while`. Οι `switch`, `break`, `continue` έρχονται στο Κεφάλαιο 8.

Παραδείγματα

§6.12 Ιχνηλάτηση εντολών έκφρασης

Εφαρμόζει: εντολή έκφρασης, `++`/`--` από το Κεφάλαιο 5. Ποια η τιμή των `x`, `y`, `z` μετά από αυτές τις εντολές;

```
x = 4;
y = 7;
z = ++y;
y = z - (x++);
z = x - (--y);
```

Το `++y` αλλάζει πρώτα και δίνει τη νέα τιμή· το `x++` δίνει την παλιά και αλλάζει μετά:

Εντολή	`x`	`y`	`z`
`x = 4; y = 7;`	4	7	?
`z = ++y;`	4	8	8
`y = z - (x++);`	5	4	8
`z = x - (--y);`	5	3	2

§6.13 Μέγιστο δύο αριθμών

Εφαρμόζει: `if-else`. Οι διαφάνειες ρωτούν αν μπορούμε χωρίς `else`, αν υπάρχει άλλος τρόπος, και τι κάνουμε με πάνω από δύο συνθήκες. Τρεις ισοδύναμες γραφές:

```
if (x > y) max = x; else max = y;   /* 1: if-else */
max = y; if (x > y) max = x;        /* 2: χωρίς else */
max = (x > y) ? x : y;              /* 3: τελεστής συνθήκης */
```

Για περισσότερες από δύο συνθήκες: εμφωλευμένες `if`.

§6.14 Είναι τα δύο προγράμματα ισοδύναμα;

Εφαρμόζει: εμφωλευμένες `if`*, dangling else.* Το πρώτο είναι η αλυσίδα `else if` της Θεωρίας· το δεύτερο είναι:

```
lab = -1;
if (year <= 1)
    if (year == 1)
        lab = 20;
else
    lab = 0;
```

Αν το `else` ανήκε στην εξωτερική `if`, όπως υπονοεί η στοίχιση, θα ήταν ισοδύναμα. Ανήκει όμως στην εσωτερική, άρα:

`year`	1ο πρόγραμμα	2ο πρόγραμμα
0 (ή αρνητικό)	`lab = -1`	`lab = 0`
1	`lab = 20`	`lab = 20`
2 (ή μεγαλύτερο)	`lab = 0`	`lab = -1`

Δεν είναι ισοδύναμα. Για να ανήκει το `else` στην εξωτερική `if`, κλείστε την εσωτερική `if` σε αγκύλες: `if (year <= 1) { if (year == 1) lab = 20; } else lab = 0;`.

§6.15 Άθροισμα με while

Εφαρμόζει: `while`. Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>

int main() {
    int i = 0;
    int sum = 0;
    while (i < 42) {
        sum += i;
        i++;
    }
    printf("%d %d\n", i, sum);
    return 0;
```

```
}
```

Ο βρόχος σταματάει όταν `i == 42`, αφού έχει προσθέσει $0 + \cdots + 41 = 861$:

```
$ ./a.out
42 861
```

Μετά τον βρόχο το `i` έχει την τιμή που *έκανε* τη συνθήκη ψευδή (42), όχι την τελευταία για την οποία εκτελέστηκε το σώμα (41).

§6.16 Ατέρμονες βρόχοι: while (1) και for (;;)

Εφαρμόζει: κενή εντολή, `while`, `for`.

```
while (1);
for (;;) ;
```

Ατέρμονες βρόχοι με κενό σώμα (στις διαφάνειες το `for (;;)` δεν έχει σώμα· εδώ πήρε την κενή εντολή): το πρόγραμμα «κολλάει» μέχρι το Ctrl-C. Μια πάντα αληθής συνθήκη έχει νόημα σε προγράμματα που τρέχουν συνεχώς (ένας server, ένα παιχνίδι) ή που τερματίζουν από μέσα, π.χ. με `return` ή με την `break` μιας επόμενης διάλεξης.

§6.17 Το ερωτηματικό μετά το while

Εφαρμόζει: κενή εντολή, `while`. Τι κάνει το παρακάτω;

```
i = 0;
while (i < 42);
    printf("%d\n", i++);
```

Το `;` μετά το `while (i < 42)` είναι κενή εντολή και *αυτή* είναι το σώμα. Το `i` δεν αλλάζει ποτέ και το πρόγραμμα κολλάει χωρίς έξοδο· η `printf` βρίσκεται μετά τον βρόχο και δεν εκτελείται ποτέ. Ο κώδικας μεταγλωττίζεται, αφού είναι συντακτικά σωστός.

§6.18 Μέτρηση με for

Εφαρμόζει: `for`. Ο πρώτος βρόχος τυπώνει 100 φορές `Hello world`· ο δεύτερος είναι το γενικό παράδειγμα των διαφανειών:

```
int i;
for ( i = 0 ; i < 100 ; i++ )
    printf("Hello world\n");

for ( i = 0 ; i < N ; i++ ) {
    printf("%d\n", i);
}
```

- *Πόσες φορές εκτελείται το block;* `N` φορές (αν `N > 0`), για `i = 0, …, N-1`· αν `N <= 0`, καμία.

- *Θα τυπωθεί το* `N`*;* Όχι: με `i == N` η συνθήκη είναι ψευδής πριν από την `printf`.
- *Τι γίνεται αν αλλάξω αρχικοποίηση ή βήμα;* Με `i = 1` το σώμα εκτελείται `N-1` φορές· με βήμα `i += 2` τυπώνονται μόνο οι ζυγοί κάτω από `N`· με `i--` το `i` απομακρύνεται από το `N` και ο βρόχος (θεωρητικά) δεν τελειώνει ποτέ.

§6.19 do-while μέχρι το 42

Εφαρμόζει: `do-while`. Τι κάνει το παρακάτω;

```
i = 0;
do
    i++;
while (i < 42);
printf("%d\n", i);
```

Το `i` αυξάνεται και μετά ελέγχεται· όταν γίνει `42` ο έλεγχος αποτυγχάνει και τυπώνεται `42`. Με `i = 100;` στην αρχή, η `do-while` εκτελεί το σώμα μία φορά και τυπώνει `101`, ενώ ένα `while (i < 42) i++;` δεν θα το εκτελούσε καθόλου και θα έμενε `100`.

§6.20 Γρήγορος έλεγχος της λύσης με pipes και time

Συμβουλή για την άσκηση `aliquot` *της hw0.* Αντί να πληκτρολογείτε κάθε φορά την είσοδο, στείλτε τη στο πρόγραμμα με pipe (Κεφάλαιο 1):

```
echo -e "138\n0\nf\n" | ./aliquot
```

Η `echo -e` ερμηνεύει κάθε `\n` ως αλλαγή γραμμής, οπότε το `./aliquot` διαβάζει τρεις γραμμές: `138`, `0` και `f`. Με `time` μετράτε και τον χρόνο εκτέλεσης, και με `ttyplot` (ίσως χρειαστεί να το εγκαταστήσετε) σχεδιάζετε την έξοδο στο τερματικό:

```
echo -e "138\n0\nf\n" | time ./aliquot
echo -e "2856\n0\nf\n" | ./aliquot | ttyplot
```

Κύρια σημεία

1. Κάθε εντολή της C είναι κενή, έκφρασης, σύνθετη, συνθήκης ή επανάληψης.
2. Η κενή εντολή `;` δεν κάνει τίποτα· μετράει όταν γίνεται σώμα `if` ή βρόχου.
3. Μια έκφραση που τελειώνει σε `;` είναι εντολή έκφρασης.
4. Ένα block `{ }` μετράει ως μία εντολή και μπορεί να γίνει σώμα κάθε εντολής ελέγχου.
5. Ροή ελέγχου είναι η σειρά εκτέλεσης των εντολών· τη δείχνει ένα διάγραμμα ροής.
6. Η `if` εκτελεί μια εντολή μόνο αν η συνθήκη είναι μη μηδενική· οι παρενθέσεις είναι υποχρεωτικές.
7. Η `if-else` εκτελεί ακριβώς μία από δύο εντολές.
8. Για πάνω από δύο περιπτώσεις γράφουμε εμφωλευμένες `if`, συνήθως ως αλυσίδα `else if`.

9. Κάθε `else` ανήκει στην πλησιέστερη προηγούμενη `if` χωρίς `else`· η στοίχιση δεν μετράει.
10. Αν δεν είμαστε σίγουροι, χρησιμοποιούμε αγκύλες `{ }` για τα όρια των σύνθετων εντολών.
11. Η `while` ελέγχει πριν από κάθε επανάληψη, οπότε το σώμα της μπορεί να μην εκτελεστεί.
12. Ένα `;` αμέσως μετά από `while (...)` ή `for (...)` γίνεται το σώμα του βρόχου.
13. Οι `while (1)` και `for (;;)` χρησιμεύουν σε προγράμματα που τρέχουν συνεχώς.
14. Η `for` ισοδυναμεί με μια `while`· το `for (i = 0; i < N; i++)` εκτελείται `N` φορές.
15. Η `do-while` εκτελεί το σώμα τουλάχιστον μία φορά· το `;` στο τέλος είναι υποχρεωτικό.
16. Με `echo -e "..." | ./prog` δοκιμάζετε γρήγορα ένα πρόγραμμα, και με `time` το χρονομετράτε.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
εντολή	statement	Η μονάδα εκτέλεσης ενός προγράμματος C
κενή εντολή	empty / null statement	Το σκέτο `;`, που δεν κάνει τίποτα (no-op)
εντολή έκφρασης	expression statement	Μια έκφραση που τελειώνει με `;`
σύνθετη εντολή	compound statement / block	Εντολές μέσα σε `{ }`, που μετράνε ως μία
ροή ελέγχου	control flow	Η σειρά με την οποία εκτελούνται οι εντολές
διάγραμμα ροής	flowchart	Σχήμα με ρόμβους (συνθήκες) και ορθογώνια (εντολές)
εμφωλευμένες if	nested if	`if` μέσα στο σώμα άλλης `if` ή `else`
κρεμασμένο else	dangling else	Αμφισημία για το σε ποια `if` ανήκει ένα `else`
βρόχος	loop	Εντολή που επαναλαμβάνει ένα σώμα
επανάληψη	iteration	Μία εκτέλεση του σώματος του βρόχου
ατέρμονας βρόχος	infinite loop	Βρόχος που δεν τερματίζει ποτέ
αρχικοποίηση / βήμα	initialization / step	Το πρώτο και το τρίτο μέρος της `for`

Διάβασμα

- **Διαφάνειες**: Διάλεξη 6, σελ. 1–29. Pipes και `time`: σελ. 2· κατηγορίες εντολών: σελ. 5–8· ροή ελέγχου, `if`, `if-else`: σελ. 9–13· εμφωλευμένες `if` και dangling else: σελ. 14–16· `while`: σελ. 17–21· `for`: σελ. 22–24· `do-while`: σελ. 25–26.
- **Σημειώσεις**: Κεφάλαιο 3: Η ροή του ελέγχου, ενότητες «Η ροή του ελέγχου στην C», «Εντολή `if`», «Εντολές βρόχου `while`», «Εντολή βρόχου `for`» (K04, σελ. 46–49 και 52–55).

Οι διαφάνειες συνιστούν να έχετε καλύψει τις σημειώσεις του κ. Σταματόπουλου μέχρι και τη σελίδα 62, δηλαδή και το Κεφάλαιο 4: Συναρτήσεις μέχρι την ενότητα «Συνάρτηση υπολογισμού παραγοντικού».

- **Εργαστήριο**: Εργαστήριο 3: εισαγωγή για τους τρεις βρόχους και τα διαγράμματα ροής τους, ασκήσεις `seq.c` (`while`, `for`, `do...while`), `root.c` (`if...else`), `birthdate.c`, `limit.c`
- **Βιβλίο**: K&R, κεφ. 3 (σελ. 85–100), όπως προτείνουν οι σημειώσεις.
- **Άλλα**:
  - Wikipedia: Control flow, Conditional (computer programming), While loop, For loop, Dangling else
  - Για τις ακολουθίες aliquot: σχετικό άρθρο (PDF)

Συχνά λάθη

- **`;` μετά τη συνθήκη βρόχου**. `while (i < 42); printf(...);` κολλάει χωρίς έξοδο· `for (i = 0; i < N; i++);` εκτελεί το «σώμα» μία φορά, μετά τον βρόχο. Διόρθωση: σβήστε το `;`, βάλτε το σώμα σε `{ }`. Το ίδιο με `if (x > y); max = x;`.
- **Εμπιστοσύνη στη στοίχιση**. Δύο εντολές με εσοχή κάτω από `if` χωρίς αγκύλες: μόνο η πρώτη ανήκει στην `if`. Διόρθωση: `{ }`.
- **Dangling else**. Το `else` πάει στην εσωτερική `if` και βγαίνουν λάθος τιμές· ο gcc με `-Wall` λέει `suggest explicit braces to avoid ambiguous 'else'`.
- **Χωρίς παρενθέσεις**. `if x > 1` δεν μεταγλωττίζεται (`expected '(' before 'x'`).
- **Ξεχασμένο βήμα σε** `while`: ο βρόχος δεν τελειώνει. **Off-by-one**: `for (i = 0; i <= N; i++)` εκτελείται `N+1` φορές, όχι `N`.
- **Χωρίς `;` στο τέλος της** `do-while`. `do i++; while (i < 42)` δίνει `expected ';'`.
- **Ο μετρητής μετά τον βρόχο**. Μετά το `while (i < 42) { … i++; }` το `i` είναι 42.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K6.9** while(!42) (17% σωστές): Το 53% απάντησε ότι δεν είναι έγκυρη C, πιστεύοντας ότι η συνθήκη πρέπει να είναι σύγκριση· στην C κάθε ακέραια έκφραση είναι συνθήκη, και το `!42` κάνει 0.
- **K6.8** Πόσες φορές εκτελείται ένα for μέχρι N (23% σωστές): Το 34% απάντησε N, υποθέτοντας σιωπηρά ότι το `N` είναι θετικό· αν το `N` είναι 0 ή αρνητικό, το σώμα δεν εκτελείται καθόλου.
- **K6.7** while γραμμένο ως for (24% σωστές): Το 34% επέλεξε `for(; condition ; statement);`· αυτό δουλεύει μόνο αν το `statement` είναι απλή έκφραση, αφού το τρίτο μέρος της `for` δέχεται έκφραση και όχι οποιαδήποτε εντολή (π.χ. ένα μπλοκ `{…}` ή ένα `if`).

- **K6.6** if χωρίς παρενθέσεις (42% σωστές): Το 53% απάντησε True, ίσως από γλώσσες όπως η Python· στην C η συνθήκη της `if` πρέπει να είναι σε παρενθέσεις.
- **K6.4** Επαναλήψεις με βήμα 2 (57% σωστές): Το 27% επέλεξε 49 φορές, ένα κλασικό λάθος «κατά ένα» (off-by-one): ξεχνά ότι μετράει και η επανάληψη με `i = 0`.
- **K6.3** Η σύνταξη του for (59% σωστές): Το 27% επέλεξε `for(i < 100; i++;)`, βάζοντας τη συνθήκη στη θέση της αρχικοποίησης. Τα τρία μέρη του `for` είναι πάντα αρχικοποίηση; συνθήκη; βήμα, και ένα κενό μέρος αφήνει μόνο το ερωτηματικό του.

Ερωτήσεις κατανόησης

- **E6.1** Πόσες εντολές περιέχει η γραμμή `;;;;;` και τι κάνουν;[54]
- **E6.2** Γιατί ένα block `{ … }` μπορεί να σταθεί ως σώμα μιας `if`, ενώ δύο σκέτες εντολές όχι;[55]
- **E6.3** Σε ποια `if` ανήκει ένα `else` όταν υπάρχουν δύο υποψήφιες και δεν υπάρχουν αγκύλες;[56]
- **E6.4** Ποια είναι η διαφορά ανάμεσα σε `while` και `do-while`; Δώστε μια περίπτωση όπου δίνουν διαφορετικό αποτέλεσμα.[57]
- **E6.5** Γράψτε έναν βρόχο `while` ισοδύναμο με το `for (i = 0; i < 100; i++) printf("Hello world\n");`.[58]
- **E6.6** Πόσες φορές εκτελείται το σώμα του `for (i = 0; i <= N; i++);`[59]

Kahoot από το αμφιθέατρο (K6.1–K6.9)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K6.1** Απλό if-else: 91% σωστές απαντήσεις
- **K6.2** while και do-while: 85% σωστές απαντήσεις
- **K6.3** Η σύνταξη του for: 59% σωστές απαντήσεις
- **K6.4** Επαναλήψεις με βήμα 2: 57% σωστές απαντήσεις
- **K6.5** Ένα for χωρίς βήμα: 56% σωστές απαντήσεις
- **K6.6** if χωρίς παρενθέσεις: 42% σωστές απαντήσεις
- **K6.7** while γραμμένο ως for: 24% σωστές απαντήσεις
- **K6.8** Πόσες φορές εκτελείται ένα for μέχρι N: 23% σωστές απαντήσεις
- **K6.9** while(!42): 17% σωστές απαντήσεις

[54] Πέντε κενές εντολές· καμία δεν κάνει τίποτα (no-op).

[55] Η `if` δέχεται ως σώμα μία εντολή. Ένα block είναι συντακτικά μία (σύνθετη) εντολή, ενώ δύο σκέτες εντολές είναι δύο: μόνο η πρώτη ανήκει στην `if`.

[56] Στην πλησιέστερη προηγούμενη `if` που δεν έχει ήδη `else`, ανεξάρτητα από τη στοίχιση.

[57] Η `while` ελέγχει πριν από το σώμα (0 ή περισσότερες εκτελέσεις), η `do-while` μετά (τουλάχιστον μία). Με `i = 100`, το `do i++; while (i < 42);` αφήνει `i = 101`, ενώ το `while (i < 42) i++;` αφήνει `i = 100`.

[58] `i = 0; while (i < 100) { printf("Hello world\n"); i++; }`

[59] `N + 1` φορές (για `i = 0, …, N`), αν `N >= 0`.

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A6.1–A6.9)

- **A6.1** Τι τυπώνει η do-while: Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 26 · ★☆☆ · trace · `slides-lec06-do-while`
- **A6.2** Τιμές μετά από εντολές έκφρασης: Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 7 · ★☆☆ · trace · `slides-lec06-expression-statements`
- **A6.3** Πόσες φορές εκτελείται μια for: Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 24 · ★☆☆ · short-answer · `slides-lec06-for-count`
- **A6.4** Ο βρόχος for (;;): Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 23 · ★☆☆ · short-answer · `slides-lec06-for-empty`
- **A6.5** Μέγιστο με if-else και εναλλακτικές: Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 13 · ★☆☆ · short-answer · `slides-lec06-if-else-max`
- **A6.6** Ο βρόχος while (1): Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 20 · ★☆☆ · short-answer · `slides-lec06-while-forever`
- **A6.7** Άθροισμα με while: Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 19 · ★☆☆ · trace · `slides-lec06-while-sum`
- **A6.8** Το πρόβλημα του dangling else: Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 15 · ★★☆ · trace · `slides-lec06-dangling-else`
- **A6.9** Το ερωτηματικό μετά το while: Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 21 · ★★☆ · trace · `slides-lec06-while-semicolon`

Εργαστήριο (A6.10–A6.13)

- **A6.10** Αποσφαλμάτωση αθροίσματος με όριο: Εργαστήριο 3, Άσκηση 4 · ★☆☆ · debug · `lab-lab03-limit`
- **A6.11** Κατασκευή πυραμίδας: Εργαστήριο 4, Άσκηση 2 · ★☆☆ · programming · `lab-lab04-pyramid`
- **A6.12** Υπολογισμός ριζών τριωνύμου: Εργαστήριο 3, Άσκηση 2 · ★★☆ · programming · `lab-lab03-root`
- **A6.13** Υπολογισμός αθροίσματος σειράς: Εργαστήριο 3, Άσκηση 1 · ★★☆ · programming · `lab-lab03-seq`

Εργασίες (A6.14–A6.15)

- **A6.14** Η εικασία Collatz: Εργασία 0 (2023-24), Άσκηση 3 · ★★☆ · programming · `hw-2023-hw0-collatz`
- **A6.15** Οι Ακολουθίες Aliquot: Εργασία 0 (2025-26), Άσκηση 3 · ★★☆ · programming · `hw-2025-hw0-aliquot`

Θέματα εξετάσεων (A6.16–A6.25)

- **A6.16** Μέγιστος Κοινός Διαιρέτης: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #6 (Crypto Themed), Θέμα 1 · ★☆☆ · programming · `exam-2023-fall-ex6-q1`
- **A6.17** Προσεγγίζοντας το π: Κατατακτήριες Δεκεμβρίου 2024, Θέμα 1 · ★☆☆ · programming · `exam-2024-dec-q1`
- **A6.18** Mystery: Εξέταση Σεπτεμβρίου 2025, Θέμα 1 · ★☆☆ · trace · `exam-2025-sep-q1`
- **A6.19** Mystery: Εξέταση Σεπτεμβρίου 2026, Θέμα 1 · ★☆☆ · trace · `exam-2026-sep-q1`
- **A6.20** Κάντο όπως ο Βιετά: Κατατακτήριες Δεκεμβρίου 2023, Θέμα 1 · ★★☆ · programming · `exam-2023-dec-q1`
- **A6.21** Ασημένιο Κλάσμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #10, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex10-q2`
- **A6.22** Κάντο όπως ο Βιετά: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #13, Θέμα 1 · ★★☆ · programming · `exam-2023-fall-ex13-q1`
- **A6.23** Τυπώνοντας τον Πίνακα ASCII: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #15, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex15-q2`
- **A6.24** Πετυχαίνοντας τον Στόχο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #9, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex9-q2`
- **A6.25** Εύρεση Πρώτων Παραγόντων - factor: Εξέταση Ιανουαρίου 2026, Θέμα 3 · ★★☆ · programming · `exam-2026-jan-q3`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A2.11** Εκτύπωση χαρακτήρων: Εργαστήριο 4, Άσκηση 1 · ★☆☆ · programming · `lab-lab04-printchar`
- **A3.5** Προσέγγιση του π με τη σειρά Leibniz: Διάλεξη 3: Συναρτήσεις, διαφάνεια 43 · ★★☆ · programming · `slides-lec03-leibniz-pi`
- **A5.9** Mystery: Εξέταση Ιανουαρίου 2025, Θέμα 1 · ★☆☆ · trace · `exam-2025-jan-q1`
- **A5.10** Mystery: Εξέταση Ιανουαρίου 2026, Θέμα 1 · ★☆☆ · trace · `exam-2026-jan-q1`
- **A5.11** Η συνάρτηση mystery: Εξέταση Ιουνίου 2026, Θέμα 1 · ★☆☆ · trace · `exam-2026-jun-q1`
- **A5.8** Υπολογισμός ημέρας μίας δεδομένης ημερομηνίας: Εργαστήριο 3, Άσκηση 3 · ★☆☆ · programming · `lab-lab03-birthdate`
- **A9.16** Το Πρόβλημα του Τρόλεϊ (trolley): Εργασία 0 (2024-25), Άσκηση 3 · ★★☆ · programming · `hw-2024-hw0-trolley`

Κεφάλαιο 7: Επίλυση Προβλημάτων

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να

- επιλέγετε ανάμεσα σε `while`, `for` και `do-while` και να σχεδιάζετε το διάγραμμα ροής τους·
- μετατρέπετε μια περιγραφή όπως «όλοι οι τριψήφιοι άρτιοι» σε αρχική τιμή, συνθήκη και βήμα ενός βρόχου·
- διαλέγετε ανάμεσα σε έναν βρόχο με φίλτρο (`if` στο σώμα) και έναν βρόχο που πηγαίνει κατευθείαν στις τιμές που σας ενδιαφέρουν·
- γράφετε χρήσιμα σχόλια και ένα `README.md` για τις εργασίες σας·
- μορφοποιείτε τον κώδικά σας με συνέπεια, με ή χωρίς `clang-format`·
- δουλεύετε με έναν editor, το git και τη γραμμή εντολών ως ενιαία ροή εργασίας.

Προαπαιτούμενα: Κεφάλαιο 1, Κεφάλαιο 4, Κεφάλαιο 6

Χρόνος μελέτης: ~1,5 ώρα

Σύνοψη

Η διάλεξη αυτή είναι η πρώτη από τις διαλέξεις «Επίλυσης Προβλημάτων» του μαθήματος: λιγότερη νέα θεωρία και περισσότερη πρακτική. Ξεκινά με μια σύντομη ανακεφαλαίωση των τριών βρόχων της C (`while`, `for`, `do-while`) και δύο προβλήματα προθέρμανσης, που δείχνουν πώς μια φράση όπως «όλοι οι τριψήφιοι περιττοί που διαιρούνται με το 7» γίνεται αρχική τιμή, συνθήκη και βήμα ενός βρόχου. Το μεγαλύτερο μέρος της ώρας ήταν ζωντανή δουλειά με εθελοντές από το ακροατήριο, πάνω σε όσα κάνουν έναν κώδικα επαγγελματικό και όχι μόνο σωστό: σχόλια, `README`, μορφοποίηση, editors, git και γραμμή εντολών, όλα όσα θα σας ζητηθούν στις εργασίες.

Θεωρία

§7.1 Γιατί μας ενδιαφέρει η ποιότητα του λογισμικού

Η διάλεξη άνοιξε με ένα επίκαιρο γεγονός: τη διακοπή λειτουργίας της υπηρεσίας cloud AWS της Amazon τον Οκτώβριο του 2025, που επηρέασε επιχειρήσεις σε όλο τον κόσμο. Η βασική αιτία (root cause) ήταν ένα υποσύστημα που παρακολουθεί την υγεία των load balancers του δικτύου τους. Ο καθηγητής Ken Birman (Cornell) σχολίασε ότι οι προγραμματιστές πρέπει να χτίζουν λογισμικό με καλύτερη **ανοχή σε σφάλματα** (fault tolerance), δηλαδή λογισμικό που

συνεχίζει να λειτουργεί σωστά ακόμη κι όταν κάποιο κομμάτι του, ή κάποιο σύστημα από το οποίο εξαρτάται, αποτύχει.

Το δίδαγμα για εμάς είναι ότι ένα λάθος σε ένα «μικρό» κομμάτι κώδικα μπορεί να έχει τεράστιες συνέπειες. Οι σημειώσεις του μαθήματος το λένε ως εξής: ένα πρόγραμμα C πρέπει να είναι **σωστό**, **αποδοτικό**, **εύρωστο** (να μην «σκάει» για καμία είσοδο), **τεκμηριωμένο** και **ευανάγνωστο**. Τα δύο τελευταία είναι το θέμα του δεύτερου μισού της διάλεξης.

§7.2 Ανακοινώσεις: ο διαγωνισμός GRCPC

Οι διαφάνειες έδειξαν τους πίνακες αποτελεσμάτων του GRCPC για το 2023, το 2024 και το 2025. Είναι ο ελληνικός περιφερειακός διαγωνισμός προγραμματισμού του ICPC (ICPC Greece Regional Competition), όπου ομάδες φοιτητών λύνουν αλγοριθμικά προβλήματα υπό πίεση χρόνου. Η κατάταξη γίνεται με βάση πόσα προβλήματα έλυσε κάθε ομάδα και, σε ισοβαθμία, με βάση την ποινή χρόνου (penalty)· οι πίνακες δείχνουν για κάθε πρόβλημα και πόσες προσπάθειες (tries) χρειάστηκαν. Ομάδες του ΕΚΠΑ εμφανίζονται στις υψηλές θέσεις: το 2025 η «The Ancient Missiles» πήρε χρυσό μετάλλιο και η «DITide and Conquer» ασημένιο. Το μήνυμα είναι ότι οι δεξιότητες επίλυσης προβλημάτων που χτίζουμε εδώ είναι ακριβώς αυτές που χρειάζονται τέτοιοι διαγωνισμοί.

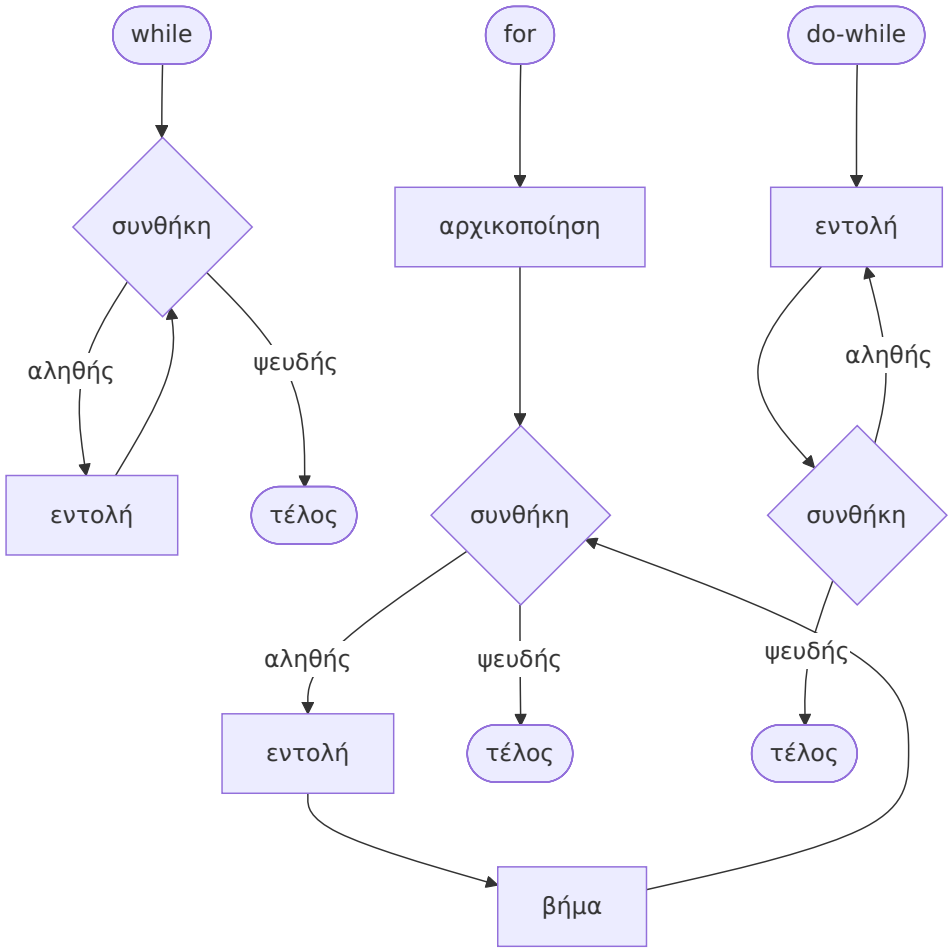
§7.3 Flow: η κατάσταση πλήρους συγκέντρωσης

Η διάλεξη όρισε το **flow** (γνωστό και ως «in the zone» ή «locked in»), όπως το περιγράφει η Wikipedia από τη θετική ψυχολογία: είναι η νοητική κατάσταση στην οποία κάποιος είναι πλήρως απορροφημένος σε μια δραστηριότητα, με ενεργητική συγκέντρωση, πλήρη εμπλοκή και απόλαυση της διαδικασίας, τόσο που αλλάζει η αίσθηση του χρόνου. Ο προγραμματισμός είναι από τις δραστηριότητες που οδηγούν εύκολα σε flow. Για να φτάσετε εκεί χρειάζεστε εργαλεία που δεν σας διακόπτουν: έναν editor που ξέρετε καλά, μια σταθερή ροή εργασίας με git και κώδικα που διαβάζεται χωρίς κόπο.

§7.4 Ανακεφαλαίωση: οι τρεις βρόχοι

Οι **δομές επανάληψης** (loops, βρόχοι) της C παρουσιάστηκαν αναλυτικά στο Κεφάλαιο 6· εδώ τις θυμόμαστε με τα διαγράμματα ροής τους.

- Η `while` επαναλαμβάνει μια εντολή όσο η λογική συνθήκη είναι αληθής. Η συνθήκη ελέγχεται *πριν* από κάθε επανάληψη, άρα το σώμα μπορεί να μην εκτελεστεί ούτε μία φορά.
- Η `for` αρχικοποιεί μεταβλητές, επαναλαμβάνει μια εντολή όσο η συνθήκη είναι αληθής και στο τέλος κάθε επανάληψης εκτελεί το βήμα: `for ( αρχικοποίηση ; συνθήκη ; βήμα ) εντολή`.
- Η `do-while` εκτελεί πρώτα μία φορά την εντολή και *μετά* ελέγχει τη συνθήκη για το αν χρειάζεται άλλη επανάληψη. Το σώμα της εκτελείται πάντα τουλάχιστον μία φορά. Προσέξτε το `;` μετά το `while ( συνθήκη )`.



Σχήμα: τα διαγράμματα ροής των `while`, `for` *και* `do-while`.

Το παράδειγμα των διαφανειών για την `for` τυπώνει 100 φορές το `Hello world`, γιατί το `i` παίρνει τις τιμές 0, 1, ..., 99:

```
int i;
for ( i = 0 ; i < 100 ; i++ )
    printf("Hello world\n");
```

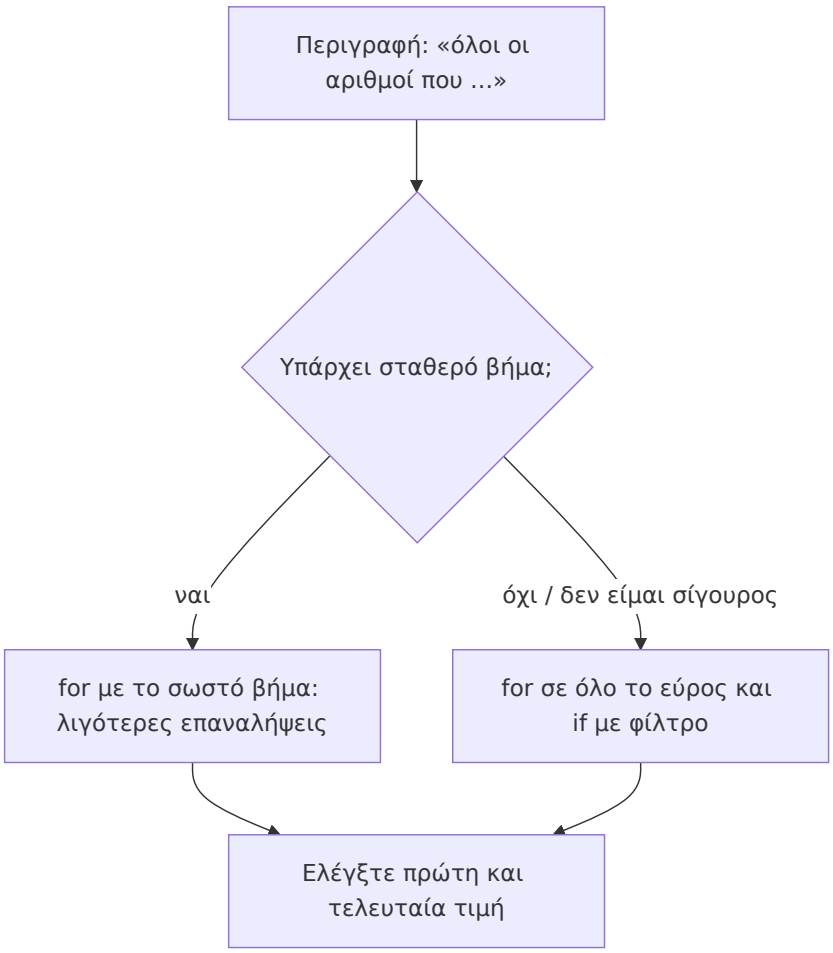
§7.5 Από την περιγραφή στον βρόχο

Τα περισσότερα προβλήματα της μορφής «κάνε κάτι για όλους τους αριθμούς που ...» λύνονται με έναν βρόχο `for`, και το δύσκολο κομμάτι είναι να μεταφράσετε την περιγραφή στα τρία μέρη του:

1. **Αρχική τιμή**: ποιος είναι ο *πρώτος* αριθμός που σας ενδιαφέρει; Αν η σειρά είναι φθίνουσα, είναι ο μεγαλύτερος.

2. **Συνθήκη**: μέχρι πού πάτε; Σκεφτείτε αν το όριο περιλαμβάνεται (`<=`, `>=`) ή όχι (`<`, `>`). Τα λάθη «κατά ένα» (off-by-one) στο όριο είναι από τα πιο συνηθισμένα.
3. **Βήμα**: πόσο απέχουν διαδοχικές τιμές; Το βήμα δεν είναι υποχρεωτικά `i++`: μπορεί να είναι `i -= 2` για φθίνουσα σειρά ή `i += 14` για να πηγαίνετε από πολλαπλάσιο σε πολλαπλάσιο.

Όταν οι τιμές που θέλετε δεν ακολουθούν απλό βήμα, ή δεν σας έρχεται αμέσως το σωστό βήμα, υπάρχει μια δεύτερη στρατηγική: διατρέχετε ένα μεγαλύτερο σύνολο και **φιλτράρετε** με μια `if` μέσα στο σώμα, για παράδειγμα με τον τελεστή υπολοίπου `%` (`i % 7 == 0` σημαίνει «το `i` διαιρείται με το 7»).



Σχήμα: δύο τρόποι να διατρέξετε τις τιμές που σας ενδιαφέρουν.

Οι δύο λύσεις δίνουν το ίδιο αποτέλεσμα, αλλά η λύση με το βήμα κάνει λιγότερες επαναλήψεις και δεν χρειάζεται `if`. Η λύση με το φίλτρο είναι συχνά πιο εύκολο να τη γράψετε σωστά με την πρώτη. Σε κάθε περίπτωση, ελέγξτε ότι η πρώτη και η τελευταία τιμή που βγάζει ο βρόχος είναι αυτές που περιμένετε.

Προσέξτε ακόμη ότι η `for` επιτρέπει πολλές αρχικοποιήσεις χωρισμένες με κόμμα: `for (product = 1, i = 100; ...)` δίνει αρχική τιμή και στον **συσσωρευτή** (accumulator) `product` και στη μεταβλητή του βρόχου. Ένας συσσωρευτής γινομένου ξεκινά από το 1 (το ουδέτερο στοιχείο του πολλαπλασιασμού), όπως ένας συσσωρευτής αθροίσματος ξεκινά από το 0.

§7.6 Σχολιασμός προγραμμάτων

Ένα **σχόλιο** (comment) είναι κείμενο μέσα στον κώδικα που ο μεταγλωττιστής αγνοεί. Στη C γράφεται ανάμεσα σε `/*` και `*/` (και μπορεί να πιάνει πολλές γραμμές) ή, από τη C99 και μετά, μετά από `//` μέχρι το τέλος της γραμμής:

```
/* File: picomp.c
   Υπολογίζει το π από τη σειρά 1/1^2 + 1/2^2 + ... */
sum = sum + current;   // πρόσθεσε τον τρέχοντα όρο
```

Τα σχόλια γράφονται για ανθρώπους: για τον βαθμολογητή, για τους συνεργάτες σας και, κυρίως, για εσάς σε έναν μήνα. Οι βασικοί κανόνες:

- **Εξηγήστε το «γιατί», όχι το «τι».** Το `i++; // αύξησε το i` δεν προσθέτει τίποτα. Ένα σχόλιο αξίζει όταν λέει κάτι που ο κώδικας δεν λέει: την ιδέα του αλγορίθμου, μια παραδοχή για την είσοδο, γιατί επιλέξατε αυτό το όριο.
- **Μη σχολιάζετε κάθε γραμμή.** Βάλτε ένα σύντομο σχόλιο στην αρχή κάθε ενότητας ή συνάρτησης που λέει τι προσπαθεί να πετύχει· τη μεγάλη εικόνα.
- **Τα σχόλια δεν διορθώνουν ασαφή κώδικα.** Αν χρειάζεστε ένα μεγάλο σχόλιο για να εξηγήσετε τι κάνει μια γραμμή, ίσως να χρειάζεται καλύτερα ονόματα μεταβλητών ή απλούστερο κώδικα.
- **Κρατήστε τα σχόλια ενημερωμένα.** Ένα σχόλιο που λέει άλλα από τον κώδικα είναι χειρότερο από κανένα σχόλιο.
- **Δώστε την πηγή.** Αν πήρατε μια ιδέα ή έναν τύπο από κάπου (βιβλίο, σελίδα, Wikipedia), βάλτε τον σύνδεσμο.

Οι σημειώσεις ορίζουν την **τεκμηρίωση** ως καλά ονόματα *μαζί με* σχόλια, «τόση όση χρειάζεται για να κάνει το πρόγραμμα κατανοητό».

§7.7 Δημιουργία README

Το `README.md` είναι το αρχείο που συνοδεύει ένα project και εξηγεί τι είναι και πώς χρησιμοποιείται. Η κατάληξη `.md` σημαίνει ότι γράφεται σε **Markdown**, μια απλή μορφή κειμένου όπου `#` ξεκινά επικεφαλίδα, `-` ξεκινά λίστα και οι τριπλές ανάποδες αποστρόφους περικλείουν κώδικα. Το GitHub εμφανίζει αυτόματα το `README.md` στην πρώτη σελίδα κάθε repository, μορφοποιημένο.

Στις εργασίες του μαθήματος το `README.md` είναι μέρος της υποβολής: για παράδειγμα η hw0 του 2025 ζητά `cmdline/README.md` με «μια σύντομη περιγραφή για τον τρόπο που λύσατε το κάθε πρόβλημα». Ένα καλό `README` για μια άσκηση:

- περιγράφει τη **λειτουργία** του προγράμματος (τι κάνει, τι είσοδο περιμένει, τι έξοδο δίνει)·
- εξηγεί τη **λογική** της λύσης, την ιδέα που δεν φαίνεται εύκολα από τον κώδικα·
- σε μεγαλύτερα projects, περιγράφει τη **δομή**: ποια αρχεία υπάρχουν και τι κάνει το καθένα·
- λέει πώς **μεταγλωττίζεται και τρέχει** το πρόγραμμα.

Η διαφορά από τα σχόλια είναι το επίπεδο: στον κώδικα βάζετε σύντομα σχόλια για κάθε ενότητα, στο `README` δίνετε την πιο εκτενή περιγραφή του συνόλου.

§7.8 Μορφοποίηση κώδικα

Ο μεταγλωττιστής δεν νοιάζεται για κενά και αλλαγές γραμμής, οι άνθρωποι όμως νοιάζονται. Κώδικας με ασυνεπή **στοίχιση** (indentation) κρύβει λάθη: μια εντολή μπορεί να *φαίνεται* μέσα σε μια `if` ενώ δεν είναι (θυμηθείτε το dangling `else` στο Κεφάλαιο 6). Οι σημειώσεις ζητούν:

- **συνεπή στοίχιση**: κάθε επίπεδο εμφώλευσης μετακινείται κατά τον ίδιο αριθμό κενών·
- γραμμές έως **80 χαρακτήρες**·
- **κενά γύρω από τους τελεστές**, όπου βοηθούν την ανάγνωση (`a = b + c;` αντί για `a=b+c;`).

Δεν χρειάζεται να τα κάνετε όλα με το χέρι. Το εργαλείο `clang-format` ξαναγράφει ένα αρχείο C σύμφωνα με ένα στυλ, και είναι εγκατεστημένο στα Linux εργαστήρια της σχολής:

```
clang-format -i -style=Google prog.c
```

Το `-i` (in place) αλλάζει το ίδιο το αρχείο και το `-style=Google` διαλέγει το Google style guide. Αν θέλετε άλλο όριο χαρακτήρων ανά γραμμή, εξάγετε το στυλ σε αρχείο, αλλάξτε το `ColumnLimit` και χρησιμοποιήστε το αρχείο:

```
clang-format -style=Google --dump-config > google.clang-format
clang-format -i -style=file:./google.clang-format prog.c
```

Το σημαντικό δεν είναι *ποιο* στυλ θα διαλέξετε αλλά να είστε **συνεπείς** μέσα σε ένα project.

§7.9 Editors

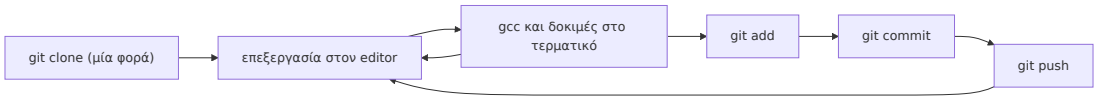
Ο **editor** (επεξεργαστής κειμένου) είναι το εργαλείο με το οποίο περνάτε τις περισσότερες ώρες ως προγραμματιστές, οπότε αξίζει να τον μάθετε καλά. Στο μάθημα έχετε δει δύο οικογένειες:

- **editors τερματικού**, όπως ο `vim` (ή ο `nano`/`pico`), που τρέχουν μέσα σε ένα `ssh` και είναι διαθέσιμοι σε κάθε μηχάνημα Linux·
- **ολοκληρωμένα περιβάλλοντα ανάπτυξης** (IDE) όπως το **VS Code**, που δίνουν χρωματισμό σύνταξης, αυτόματη μορφοποίηση, ενσωματωμένο τερματικό και debugger, και μπορούν να δουλεύουν απευθείας πάνω στον λογαριασμό σας στη σχολή (Εργαστήριο 2).

Όποιον κι αν διαλέξετε, μάθετε τις συντομεύσεις του και ρυθμίστε τον να στοιχίζει με συνέπεια.

§7.10 Git και γραμμή εντολών

Ο κώδικας, τα σχόλια και το `README` ζουν σε ένα **repository** του git, και τα εργαλεία της γραμμής εντολών δένουν τα πάντα μεταξύ τους. Ο κύκλος εργασίας που θα επαναλαμβάνετε σε κάθε άσκηση είναι:



Σχήμα: ο κύκλος εργασίας editor, τερματικό και git.

Κάντε **μικρά, συχνά commits** με μηνύματα που λένε τι αλλάξατε, και `push` ώστε η δουλειά σας να υπάρχει και στο GitHub. Αν κάτι χαλάσει, μπορείτε να γυρίσετε σε μια προηγούμενη εκδοχή που δούλευε. Οι εντολές παρουσιάστηκαν στο Κεφάλαιο 1 και στο Κεφάλαιο 4.

Παραδείγματα

§7.11 Προθέρμανση 1: τριψήφιοι άρτιοι σε φθίνουσα σειρά

Το πρόβλημα: «Θέλω να τυπώσω όλους τους τριψήφιους άρτιους σε φθίνουσα σειρά (998 996 994 ... 100). Πως;» Εφαρμόζουμε τα τρία βήματα από την ενότητα «Από την περιγραφή στον βρόχο»: ο πρώτος αριθμός είναι ο μεγαλύτερος τριψήφιος άρτιος, το 998· ο τελευταίος είναι το 100, που περιλαμβάνεται, άρα η συνθήκη είναι `i >= 100`· διαδοχικοί άρτιοι απέχουν 2 και πηγαίνουμε προς τα κάτω, άρα το βήμα είναι `i -= 2`. Η λύση των διαφανειών είναι ένας βρόχος `for` με μια μεταβλητή που μειώνεται:

```c
#include <stdio.h>

int main(int argc, char **argv) {
  int i;
  for (i = 998 ; i >= 100 ; i -= 2) {
    printf("%d\n", i);
  }
  return 0;
}
```

```
$ gcc -o even even.c
$ ./even | head -3
998
996
994
$ ./even | tail -2
102
100
```

```
$ ./even | wc -l
450
```

Με τα `head`, `tail` και `wc -l` ελέγχουμε γρήγορα τα άκρα και το πλήθος: 450 αριθμοί, όσοι είναι οι άρτιοι από το 100 μέχρι το 998.

§7.12 Προθέρμανση 2: γινόμενο των τριψήφιων περιττών πολλαπλασίων του 7

Το πρόβλημα: «Θέλω να βρω το γινόμενο όλων των τριψήφιων περιττών που διαιρούνται με το 7. Πως;» Οι διαφάνειες δίνουν δύο λύσεις, που αντιστοιχούν στις δύο στρατηγικές της ίδιας ενότητας.

Λύση με φίλτρο. Ένας βρόχος `for` με μια μεταβλητή που αυξάνεται και έλεγχο για `mod 7 = 0`:

```
for (product = 1, i = 100 ; i <= 999 ; i += 2) {
  if ( i % 7 == 0 )
    product *= i;
}
```

Προσέξτε την αρχική τιμή: το 100 είναι *άρτιος*, και με βήμα 2 το `i` περνάει μόνο από άρτιους (100, 102, ..., 998). Όπως είναι γραμμένος, ο βρόχος υπολογίζει το γινόμενο των άρτιων πολλαπλασίων του 7 (112, 126, ...). Για τους περιττούς, η αρχική τιμή πρέπει να είναι ο πρώτος τριψήφιος περιττός, `i = 101`.

Λύση με βήμα. Ο πρώτος τριψήφιος περιττός που διαιρείται με το 7 είναι το $105 = 15 \cdot 7$. Τα επόμενα πολλαπλάσια του 7 εναλλάσσονται άρτιο, περιττό, άρτιο, ..., άρα διαδοχικά *περιττά* πολλαπλάσια απέχουν $2 \cdot 7 = 14$. Έτσι δεν χρειάζεται καθόλου η `if`:

```
for (prod = 1, i = 105 ; i <= 999 ; i += 14) {
    prod *= i;
}
```

Ο βρόχος κάνει 64 επαναλήψεις (105, 119, ..., 987), ενώ η λύση με φίλτρο κάνει 450. Οι διαφάνειες δεν δείχνουν τον τύπο των `product` και `prod`. Το γινόμενο αυτών των 64 αριθμών έχει 172 δεκαδικά ψηφία, οπότε δεν χωράει σε κανέναν ακέραιο τύπο της C (ακόμη και ένας `unsigned long long` φτάνει μέχρι περίπου $1{,}8 \cdot 10^{19}$). Το παρακάτω πρόγραμμα τρέχει και τις δύο λύσεις (με διορθωμένη αρχική τιμή στην πρώτη) με `double`, που κρατά ένα προσεγγιστικό αποτέλεσμα:

```
#include <stdio.h>

int main(int argc, char **argv) {
  int i, count = 0;
  double product, prod;
  for (product = 1, i = 101; i <= 999; i += 2) {
    if (i % 7 == 0)
      product *= i;
```

```
  }
  for (prod = 1, i = 105; i <= 999; i += 14) {
    prod *= i;
    count++;
  }
  printf("product = %e\n", product);
  printf("prod    = %e (%d terms)\n", prod, count);
  return 0;
}
```

```
$ gcc -o prod prod.c
$ ./prod
product = 1.211521e+171
prod    = 1.211521e+171 (64 terms)
```

Αν αφήσετε `i = 100` στην πρώτη λύση, τα δύο αποτελέσματα διαφέρουν (`3.796882e+171` αντί για `1.211521e+171`): η σύγκριση δύο ανεξάρτητων λύσεων είναι ένας φθηνός τρόπος να πιάνετε λάθη.

§7.13 Ζωντανή επίλυση με εθελοντές

Στο δεύτερο μέρος της διάλεξης 2-3 εθελοντές συνδέθηκαν με `ssh` (ως χρήστης `ubuntu`) σε ένα κοινό μηχάνημα Ubuntu και δούλεψαν μπροστά στο ακροατήριο πάνω στα θέματα «Σχολιασμός Προγραμμάτων», «Δημιουργία README», «Μορφοποίηση Κώδικα», «Editors» και «Git, Command Line». Οι διαφάνειες δεν καταγράφουν τι ακριβώς γράφτηκε, οπότε το περιεχόμενο αυτών των θεμάτων βρίσκεται στις αντίστοιχες ενότητες της «Θεωρίας». Μια άσκηση στο ίδιο πνεύμα για εσάς:

1. πάρτε ένα πρόγραμμα από μια παλιά σας άσκηση (π.χ. το `seq.c` του Εργαστηρίου 3)·
2. περάστε το από `clang-format -i -style=Google` και δείτε τη διαφορά με `git diff`·
3. σβήστε τα σχόλια που επαναλαμβάνουν τον κώδικα και προσθέστε ένα σύντομο σχόλιο πάνω από κάθε συνάρτηση·
4. γράψτε ένα `README.md` με τη λειτουργία, τη λογική και τον τρόπο μεταγλώττισης·
5. `git add`, `git commit` με μήνυμα που περιγράφει την αλλαγή, `git push`, και δείτε πώς εμφανίζεται το `README.md` στο GitHub.

Η διάλεξη έκλεισε με ένα Kahoot (οι ερωτήσεις του δεν είναι στις διαφάνειες) και με διάβασμα για σχόλια και `README.md` (βλ. «Διάβασμα»).

Κύρια σημεία

1. Ένα σφάλμα σε ένα μικρό κομμάτι λογισμικού μπορεί να ρίξει ολόκληρες υπηρεσίες, γι' αυτό οι προγραμματιστές πρέπει να χτίζουν λογισμικό με ανοχή σε σφάλματα.

2. Η `while` ελέγχει τη συνθήκη πριν από κάθε επανάληψη, η `for` προσθέτει αρχικοποίηση και βήμα, και η `do-while` εκτελεί το σώμα τουλάχιστον μία φορά πριν ελέγξει τη συνθήκη.
3. Για να γράψετε έναν βρόχο από μια περιγραφή, προσδιορίστε ρητά την πρώτη τιμή, τη συνθήκη τερματισμού (με ή χωρίς το όριο) και το βήμα, που μπορεί να είναι αρνητικό ή μεγαλύτερο του 1.
4. Μπορείτε να διατρέξετε ένα ευρύτερο εύρος και να φιλτράρετε με `if`, ή να υπολογίσετε το σωστό βήμα και να επισκεφθείτε μόνο τις τιμές που θέλετε· το δεύτερο κάνει λιγότερες επαναλήψεις.
5. Ελέγχετε πάντα την πρώτη και την τελευταία τιμή ενός βρόχου· μια αρχική τιμή με λάθος ισοτιμία (άρτιος αντί για περιττός) αλλάζει σιωπηλά το αποτέλεσμα.
6. Ο συσσωρευτής γινομένου ξεκινά από το 1 και ο συσσωρευτής αθροίσματος από το 0, και ένα μεγάλο γινόμενο ξεπερνά γρήγορα κάθε ακέραιο τύπο.
7. Τα σχόλια εξηγούν το «γιατί» και τη μεγάλη εικόνα, όχι κάθε γραμμή, και πρέπει να συμφωνούν με τον κώδικα.
8. Το `README.md` περιγράφει τη λειτουργία, τη λογική, τη δομή και τον τρόπο εκτέλεσης ενός project, και είναι μέρος των υποβολών στις εργασίες.
9. Συνεπής στοίχιση, γραμμές έως 80 χαρακτήρες και κενά γύρω από τελεστές κάνουν τον κώδικα ευανάγνωστο· το `clang-format` τα εφαρμόζει αυτόματα.
10. Ένας editor που ξέρετε καλά, μαζί με το git και τη γραμμή εντολών, σχηματίζουν μια ροή εργασίας που σας αφήνει να συγκεντρωθείτε στο πρόβλημα (flow).

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
ανοχή σε σφάλματα	fault tolerance	Ικανότητα ενός συστήματος να λειτουργεί σωστά παρά την αποτυχία κάποιου μέρους του.
βασική αιτία	root cause	Το αρχικό σφάλμα από το οποίο ξεκίνησε μια αποτυχία.
κατάσταση ροής	flow	Κατάσταση πλήρους απορρόφησης και συγκέντρωσης σε μια δραστηριότητα.
δομή επανάληψης, βρόχος	loop	Εντολή που εκτελεί επανειλημμένα μια άλλη εντολή όσο ισχύει μια συνθήκη.
βήμα	step	Η έκφραση που αλλάζει τη μεταβλητή του βρόχου στο τέλος κάθε επανάληψης της `for`.
συσσωρευτής	accumulator	Μεταβλητή που μαζεύει ένα αποτέλεσμα (άθροισμα, γινόμενο) κατά τη διάρκεια ενός βρόχου.

Ελληνικά	English	Σύντομος ορισμός
λάθος κατά ένα	off-by-one error	Βρόχος που κάνει μία επανάληψη παραπάνω ή λιγότερο λόγω λάθους στο όριο.
σχόλιο	comment	Κείμενο μέσα στον κώδικα που αγνοεί ο μεταγλωττιστής (`/* */`, `//`).
τεκμηρίωση	documentation	Ονόματα, σχόλια και `README` που κάνουν ένα πρόγραμμα κατανοητό.
στοίχιση	indentation	Τα κενά στην αρχή κάθε γραμμής που δείχνουν το επίπεδο εμφώλευσης.
μορφοποιητής κώδικα	code formatter	Εργαλείο (π.χ. `clang-format`) που ξαναγράφει τον κώδικα σύμφωνα με ένα στυλ.
επεξεργαστής κειμένου	editor	Πρόγραμμα για τη σύνταξη του κώδικα (π.χ. `vim`, VS Code).
ολοκληρωμένο περιβάλλον ανάπτυξης	IDE	Editor με ενσωματωμένη μεταγλώττιση, τερματικό και debugger.
αποθετήριο	repository	Φάκελος που παρακολουθείται από το git, μαζί με το ιστορικό του.

Διάβασμα

- **Διαφάνειες**: Διάλεξη 7, σελ. 1–23. AWS και ανοχή σε σφάλματα: σελ. 2· GRCPC: σελ. 3–5· flow: σελ. 8· βρόχοι: σελ. 9–12· προθέρμανση: σελ. 13–18· θέματα ζωντανής επίλυσης: σελ. 19–20· επόμενη φορά: σελ. 22.
- **Σημειώσεις**: Κεφάλαιο 12: Καλές πρακτικές, συχνά λάθη και βιβλιογραφία, ενότητες «Ένα πρόγραμμα C πρέπει να είναι …» και «Συχνά προγραμματιστικά λάθη στην C» (K04, σελ. 178–182)· Κεφάλαιο 1: Πρώτα προγράμματα σε C, ενότητες «Καλημέρα κόσμε της C» (σύνταξη σχολίων) και «Πόσο είναι το π;» (πρόγραμμα με σχόλια ανά γραμμή) (K04, σελ. 19–23)· Κεφάλαιο 3: Η ροή του ελέγχου, ενότητες «Εντολές βρόχου `while`» και «Εντολή βρόχου `for`» (K04, σελ. 52–55).
- **Εργαστήριο**: Εργαστήριο 3: εισαγωγή για τους τρεις βρόχους, άσκηση `seq.c` (`while`, `for`, `do...while`) και το παράρτημα για τα λογικά λάθη (`limit.c`)· Εργαστήριο 2: «Βήμα 1: Το περιβάλλον προγραμματισμού Visual Studio Code»· Εργαστήριο 1: «Βήμα 6: Git και GitHub» και «Άσκηση 1: Το πρώτο σας repository (info.txt)».
- **Άλλα**:
  - How to write good comments (Stack Overflow Blog)
  - Οδηγοί για `README.md`: Medium, banesullivan/README, freeCodeCamp
  - Wikipedia: Flow (psychology)
  - Αποτελέσματα GRCPC: 2023, 2024, 2025

Συχνά λάθη

- **Λάθος ισοτιμία στην αρχική τιμή**. `for (i = 100; i <= 999; i += 2)` με σκοπό τους περιττούς: περνάει μόνο από άρτιους και δεν βγάζει κανένα μήνυμα λάθους. Διόρθωση: ξεκινήστε από `i = 101`, και τυπώστε τις πρώτες τιμές για έλεγχο.
- **Λάθος στο όριο (off-by-one)**. `for (i = 998; i > 100; i -= 2)` χάνει το 100. Διόρθωση: αποφασίστε ρητά αν το όριο περιλαμβάνεται και γράψτε `>=`.
- **Λάθος φορά βήματος**. `for (i = 998; i >= 100; i += 2)` δεν τερματίζει (μέχρι να υπερχειλίσει το `i`). Διόρθωση: σε φθίνουσα σειρά το βήμα μειώνει τη μεταβλητή (`i -= 2`).
- **Συσσωρευτής γινομένου από το 0**. Με `product = 0` το αποτέλεσμα είναι πάντα 0. Διόρθωση: `product = 1`.
- **Υπερχείλιση σε μεγάλο γινόμενο**. Ένα `int` ή `long` γινόμενο 64 τριψήφιων αριθμών δίνει σκουπίδια (ή αρνητικό αριθμό). Διόρθωση: εκτιμήστε το μέγεθος του αποτελέσματος πριν διαλέξετε τύπο.
- **Σχόλια που επαναλαμβάνουν τον κώδικα**. `x = x + 1; /* πρόσθεσε 1 στο x */`. Διόρθωση: σβήστε το, και γράψτε σχόλιο για την ιδέα της ενότητας.
- **Σχόλια που δεν συμφωνούν με τον κώδικα**, μετά από μια αλλαγή. Διόρθωση: ενημερώνετε τα σχόλια στο ίδιο commit με τον κώδικα.
- **Άδειο ή αδιάφορο** `README.md` σε εργασία που το ζητά. Διόρθωση: περιγράψτε λειτουργία, λογική και τρόπο εκτέλεσης, όπως ζητά η εκφώνηση.
- **Ανάμεικτη στοίχιση** (tabs και κενά, ή 2 και 4 κενά στο ίδιο αρχείο), που κάνει μια εντολή να μοιάζει μέσα σε `if` ενώ δεν είναι. Διόρθωση: `clang-format -i -style=Google prog.c` πριν από κάθε υποβολή.
- **Δουλειά που δεν έγινε ποτέ** `push`. Τα commits υπάρχουν μόνο τοπικά και η υποβολή στο GitHub είναι παλιά. Διόρθωση: `git push` και έλεγχος της σελίδας του repository.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K7.4** Καλό σχόλιο; (61% σωστές): Το 34% απάντησε True· όμως ένα σχόλιο που επαναλαμβάνει τι κάνει ο κώδικας δεν προσθέτει τίποτα, ένα καλό σχόλιο εξηγεί το «γιατί».
- **K7.3** Εργασία χωρίς σχόλια (71% σωστές): Το 25% απάντησε True, θεωρώντας τα σχόλια προαιρετικά· στο μάθημα όμως τα σχόλια και η τεκμηρίωση είναι μέρος αυτού που βαθμολογείται.

Ερωτήσεις κατανόησης

- **E7.1** Ποια είναι η διαφορά ανάμεσα σε `while` και `do-while` ως προς το πόσες φορές μπορεί να εκτελεστεί το σώμα;[60]
- **E7.2** Γράψτε την κεφαλίδα ενός βρόχου `for` που διατρέχει τα πολλαπλάσια του 5 από το 995 μέχρι και το 100, σε φθίνουσα σειρά.[61]
- **E7.3** Γιατί ο βρόχος `for (i = 100; i <= 999; i += 2)` με `if (i % 7 == 0)` δεν βρίσκει κανένα περιττό πολλαπλάσιο του 7;[62]
- **E7.4** Γιατί διαδοχικά περιττά πολλαπλάσια του 7 απέχουν 14;[63]
- **E7.5** Από ποια τιμή πρέπει να ξεκινά ένας συσσωρευτής γινομένου και γιατί;[64]
- **E7.6** Δώστε ένα παράδειγμα άχρηστου σχολίου και ένα παράδειγμα χρήσιμου σχολίου.[65]
- **E7.7** Τι πρέπει να περιέχει το `README.md` μιας άσκησης;[66]
- **E7.8** Τι κάνει η εντολή `clang-format -i -style=Google prog.c`;[67]
- **E7.9** Με ποια σειρά εκτελείτε `git commit`, `git add` και `git push` για να ανεβάσετε μια αλλαγή στο GitHub;[68]

Kahoot από το αμφιθέατρο (K7.1–K7.5)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K7.1** Αξίζει να σχολιάζω τον κώδικα;: 91% σωστές απαντήσεις
- **K7.2** Η γλώσσα του README.md: 84% σωστές απαντήσεις
- **K7.3** Εργασία χωρίς σχόλια: 71% σωστές απαντήσεις
- **K7.4** Καλό σχόλιο;: 61% σωστές απαντήσεις
- **K7.5** Πώς γράφουμε σχόλια: 46% σωστές απαντήσεις

[60] Η `while` ελέγχει τη συνθήκη πριν από το σώμα, άρα μπορεί να το εκτελέσει μηδέν φορές· η `do-while` εκτελεί το σώμα πρώτα, άρα τουλάχιστον μία φορά.

[61] `for (i = 995; i >= 100; i -= 5)`: το 995 είναι το μεγαλύτερο τριψήφιο πολλαπλάσιο του 5 και το 100 περιλαμβάνεται.

[62] Γιατί ξεκινά από άρτιο αριθμό και με βήμα 2 περνά μόνο από άρτιους· πρέπει να ξεκινά από το 101.

[63] Τα πολλαπλάσια του 7 εναλλάσσονται άρτιο, περιττό· ανάμεσα σε δύο διαδοχικά περιττά μεσολαβεί ένα άρτιο, άρα απέχουν $2 \cdot 7 = 14$.

[64] Από το 1, το ουδέτερο στοιχείο του πολλαπλασιασμού· με 0 το γινόμενο θα έμενε πάντα 0.

[65] Άχρηστο: `i++; // αύξησε το i`. Χρήσιμο: ένα σχόλιο που εξηγεί την ιδέα, π.χ. ότι το βήμα 14 επισκέπτεται μόνο τα περιττά πολλαπλάσια του 7.

[66] Τη λειτουργία του προγράμματος (είσοδος, έξοδος), τη λογική της λύσης και τον τρόπο μεταγλώττισης και εκτέλεσης· σε μεγάλα projects και τη δομή των αρχείων.

[67] Ξαναγράφει το `prog.c` στη θέση του (`-i`) με στοίχιση και κενά σύμφωνα με το στυλ της Google.

[68] `git add` (επιλογή αλλαγών), `git commit` (καταγραφή τοπικά), `git push` (αποστολή στο GitHub).

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A7.1–A7.2)

- **A7.1** Τριψήφιοι άρτιοι σε φθίνουσα σειρά: Διάλεξη 7: Επίλυση Προβλημάτων, διαφάνεια 14 · ★☆☆ · programming · `slides-lec07-even-descending`
- **A7.2** Γινόμενο τριψήφιων περιττών πολλαπλασίων του 7: Διάλεξη 7: Επίλυση Προβλημάτων, διαφάνεια 16 · ★★☆ · programming · `slides-lec07-odd-multiples-of-7`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A15.5** Πολυπλοκότητα: γινόμενο περιττών πολλαπλασίων του 7: Διάλεξη 15, διαφάνεια 13 · ★☆☆ · short-answer · `slides-lec15-complexity-odd-multiples-of-7`
- **A16.8** Γιατί εξάσκηση στην επίλυση προβλημάτων;: Διάλεξη 16, διαφάνειες 5–6 · ★☆☆ · short-answer · `slides-lec16-why-practice`

Κεφάλαιο 8: Ροή Ελέγχου #2

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να

- σχεδιάζετε έναν βρόχο είτε «φιλτράροντας» όλες τις τιμές με `if` είτε διαλέγοντας κατάλληλη αρχή και βήμα·
- αναγνωρίζετε πότε ένα γινόμενο ή άθροισμα σε βρόχο ξεπερνά τα όρια του `int`·
- σταματάτε έναν βρόχο νωρίς, με μεταβλητή-σημαία ή με `break`·
- παραλείπετε μια επανάληψη με `continue`·
- γράφετε μια `switch` αντί για μια αλυσίδα `else if`, με `case`, `default` και `break`, και ομαδοποιείτε πολλά `case`·
- εξηγείτε τι κάνει η `goto` και γιατί ο δομημένος προγραμματισμός την αποφεύγει.

Προαπαιτούμενα: Κεφάλαιο 6, Κεφάλαιο 7

Χρόνος μελέτης: ~1,5 ώρα

Σύνοψη

Στο Κεφάλαιο 6 είδαμε την `if` και τους τρεις βρόχους της C. Εδώ τους χρησιμοποιούμε σε μικρά προβλήματα με τριψήφιους αριθμούς και βλέπουμε τις υπόλοιπες δομές ελέγχου της γλώσσας. Η `break` τερματίζει αμέσως έναν βρόχο και η `continue` προχωρά στην επόμενη επανάληψη· και οι δύο αντικαθιστούν τις μεταβλητές-σημαίες που θα χρειαζόμασταν αλλιώς. Η `switch` είναι ένας πιο καθαρός τρόπος να γράψουμε μια αλυσίδα `else if` που συγκρίνει μια ακέραια τιμή με σταθερές. Τέλος, η `goto` μεταφέρει την εκτέλεση σε μια ετικέτα· υπάρχει στη γλώσσα, αλλά ο δομημένος προγραμματισμός (και αυτό το μάθημα) μας λέει να μην τη χρησιμοποιούμε.

Θεωρία

§8.1 Υπερχείλιση σε βρόχους που πολλαπλασιάζουν

Η διάλεξη ξεκίνησε με ένα ερώτημα: ένας βρόχος πολλαπλασιάζει στο `prod` όλους τους τριψήφιους περιττούς που διαιρούνται με το 7 (105, 119, ..., 987), και το πρόγραμμα τυπώνει έναν **αρνητικό** αριθμό, `-1187656959`. Τι συμβαίνει;

Ο βρόχος πολλαπλασιάζει 64 αριθμούς, ο καθένας μεγαλύτερος από 100, άρα το σωστό αποτέλεσμα ξεπερνά το 10^{128}. Ένας `int` των 32 bit χωράει μέχρι $2^{31} - 1 = 2147483647$

(περίπου $2 \cdot 10^9$), όριο που το γινόμενο ξεπερνά ήδη στον πέμπτο παράγοντα. Αυτό λέγεται **υπερχείλιση (overflow)**. Στην πράξη κρατιούνται μόνο τα χαμηλά 32 bit, η τιμή «τυλίγεται» και μπορεί να βγει αρνητική· για προσημασμένους ακεραίους όμως η C τη χαρακτηρίζει **απροσδιόριστη συμπεριφορά (undefined behavior)**, οπότε δεν βασιζόμαστε σε κανένα αποτέλεσμα. Όταν ένας βρόχος συσσωρεύει γινόμενο ή μεγάλο άθροισμα, εκτιμήστε πρώτα πόσο μεγάλο θα βγει. Ο `long long` (64 bit, περίπου έως $9 \cdot 10^{18}$) μεταθέτει το όριο, αλλά εδώ δεν αρκεί ούτε αυτός. Για τους τύπους και τα όριά τους δείτε το Κεφάλαιο 2.

§8.2 Δύο τρόποι να διατρέξουμε τις τιμές

Για το ίδιο πρόβλημα («το γινόμενο όλων των τριψήφιων περιττών που διαιρούνται με το 7») οι διαφάνειες δίνουν δύο λύσεις, που δείχνουν δύο γενικές στρατηγικές.

1. **Φιλτράρισμα**. Διατρέχουμε ένα ευρύ σύνολο τιμών και κρατάμε με μια `if` μόνο όσες ικανοποιούν τη συνθήκη (εδώ `i % 7 == 0`). Είναι απλό και δύσκολα κάνει λάθος, αλλά ελέγχει πολλές τιμές που θα απορριφθούν.
2. **Κατάλληλη αρχή και βήμα**. Σκεφτόμαστε πρώτα ποιες ακριβώς τιμές θέλουμε. Ο πρώτος τριψήφιος περιττός που διαιρείται με το 7 είναι το $105 = 15 \cdot 7$, και κάθε επόμενος απέχει $14 = 2 \cdot 7$ (το 7 κρατά τη διαιρετότητα, το 2 την περιττότητα). Άρα `i = 105; i <= 999; i += 14`, χωρίς καμία `if`.

Η δεύτερη λύση κάνει 64 επαναλήψεις αντί για 450, αλλά απαιτεί σκέψη πριν από τον κώδικα: αν η αρχή ή το βήμα είναι λάθος, ο βρόχος περνά από λάθος τιμές χωρίς κανένα σημάδι. Π.χ. στην πρώτη λύση των διαφανειών το `i` ξεκινά από το 100 με βήμα 2, οπότε περνά από τους **άρτιους** τριψήφιους (100, 102, ...)· για τους περιττούς η αρχή πρέπει να είναι 101. Ελέγχετε πάντα με το χέρι τις πρώτες τιμές που παίρνει η μεταβλητή του βρόχου. Και οι δύο βρόχοι χρησιμοποιούν τον τελεστή κόμμα στην αρχικοποίηση (`prod = 1, i = 105`) για να αρχικοποιήσουν δύο μεταβλητές.

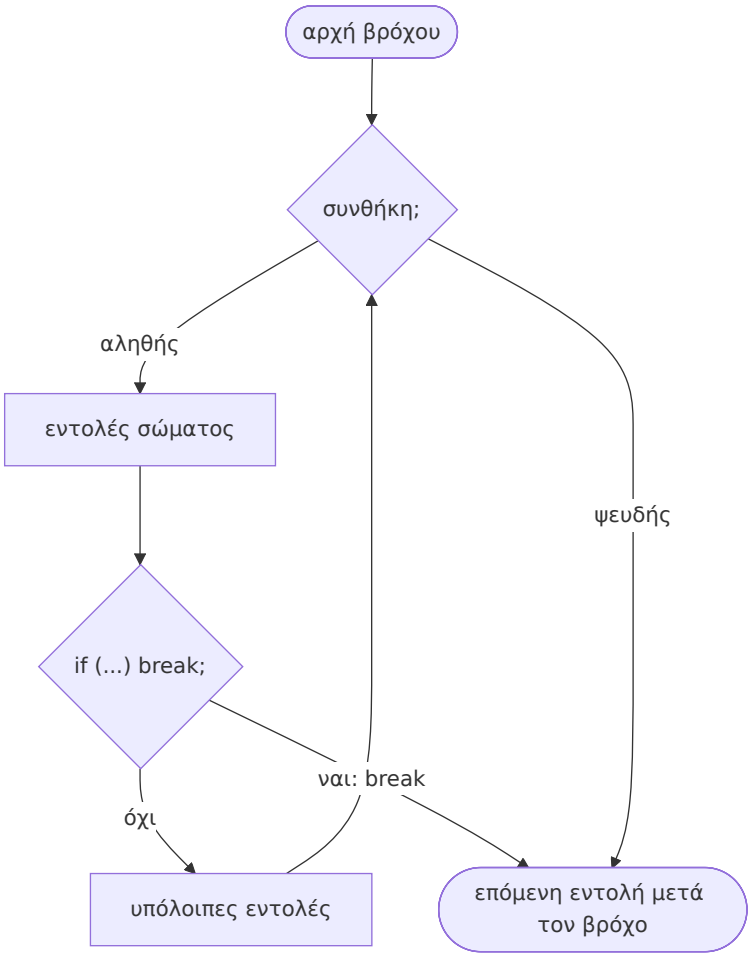
§8.3 Πρόωρος τερματισμός με μεταβλητή-σημαία

Δεύτερο πρόβλημα: *να τυπωθεί (αν υπάρχει) ο **μικρότερος** τριψήφιος που είναι πολλαπλάσιο του 2 και του 5 αλλά όχι του 4.* Ένας βρόχος από το 100 ως το 999 με τη συνθήκη `i % 2 == 0 && i % 5 == 0 && i % 4 != 0` βρίσκει τον σωστό αριθμό, αλλά δεν σταματά εκεί: τυπώνει **όλους** όσους την ικανοποιούν (110, 130, 150, ...). Το ζητούμενο όμως είναι ένας μόνο αριθμός.

Η πρώτη λύση είναι μια **μεταβλητή-σημαία (flag)**: μια ακέραια μεταβλητή `found` που ξεκινά από 0 και γίνεται 1 όταν βρούμε τον αριθμό. Την προσθέτουμε στη συνθήκη του βρόχου, `i < 1000 && !found`, οπότε μόλις γίνει 1 ο επόμενος έλεγχος αποτυγχάνει και ο βρόχος τελειώνει. Η σημαία χρησιμεύει και μετά τον βρόχο: αν έμεινε 0, δεν υπήρχε τέτοιος αριθμός (το «αν υπάρχει» της εκφώνησης). Το κόστος είναι μια επιπλέον μεταβλητή και ένας έλεγχος ακόμα (αφού εκτελεστεί πρώτα το `i++`). Η C προσφέρει έναν πιο άμεσο τρόπο για το ίδιο αποτέλεσμα.

§8.4 Η εντολή break

Η **εντολή** `break` **(break statement)** τερματίζει αμέσως τον βρόχο μέσα στον οποίο βρίσκεται. Η εκτέλεση συνεχίζει με την πρώτη εντολή **μετά** το σώμα του βρόχου· ό,τι απομένει στο σώμα, το βήμα της `for` και ο έλεγχος της συνθήκης παραλείπονται. Η `break` λειτουργεί με τον ίδιο τρόπο στη `while`, στη `do-while` και στη `for`.



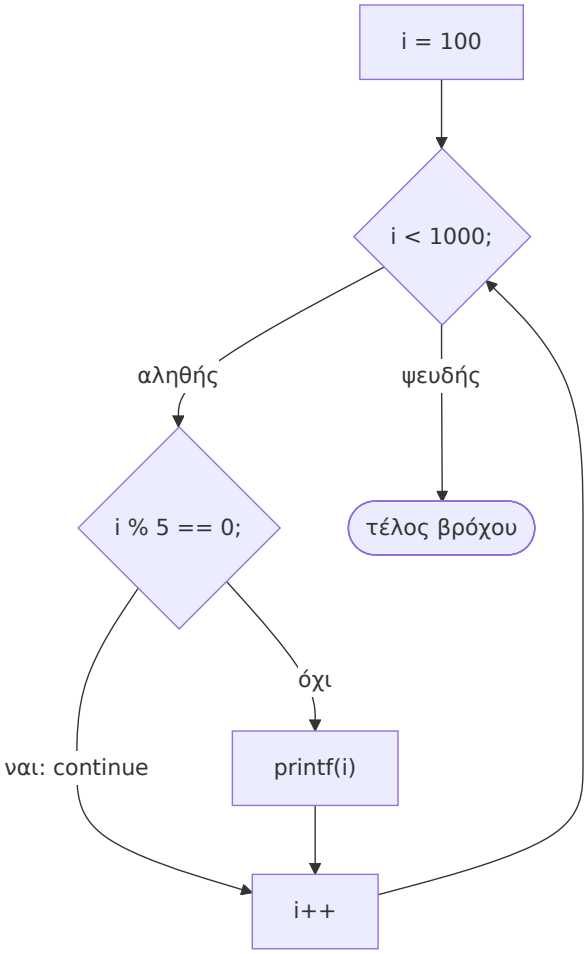
Σχήμα: η `break` *βγάζει την εκτέλεση από τον βρόχο αμέσως, χωρίς νέο έλεγχο της συνθήκης.*

Συνήθως η `break` βρίσκεται μέσα σε μια `if`: «αν βρήκαμε αυτό που ψάχνουμε, σταμάτα». Έτσι ο βρόχος έχει δύο εξόδους: την κανονική (η συνθήκη έγινε ψευδής) και την πρόωρη. Προσέξτε ότι η `break` βγαίνει μόνο από τον **πιο εσωτερικό** βρόχο (ή `switch`) που την περιέχει· σε εμφωλευμένους βρόχους ο εξωτερικός συνεχίζει κανονικά.

§8.5 Η εντολή continue

Η **εντολή** `continue` **(continue statement)** διακόπτει την **τρέχουσα επανάληψη** του βρόχου και προχωρά στην επόμενη. Οι εντολές του σώματος που ακολουθούν την `continue` παραλείπονται,

αλλά ο βρόχος δεν τελειώνει: στη `while` και στη `do-while` η εκτέλεση πηγαίνει στον έλεγχο της συνθήκης, ενώ στη `for` εκτελείται πρώτα το βήμα (π.χ. `i++`) και μετά ο έλεγχος. Όπως και η `break`, δουλεύει σε `while`, `do-while` και `for`.



Σχήμα: σε μια `for`*, η* `continue` *πηγαίνει στο βήμα* `i++` *και όχι έξω από τον βρόχο.*

Η `continue` βοηθά να «πετάμε» νωρίς τις περιπτώσεις που δεν μας ενδιαφέρουν· το ίδιο γίνεται και με μια `if` με την αντίθετη συνθήκη.

§8.6 Η εντολή switch

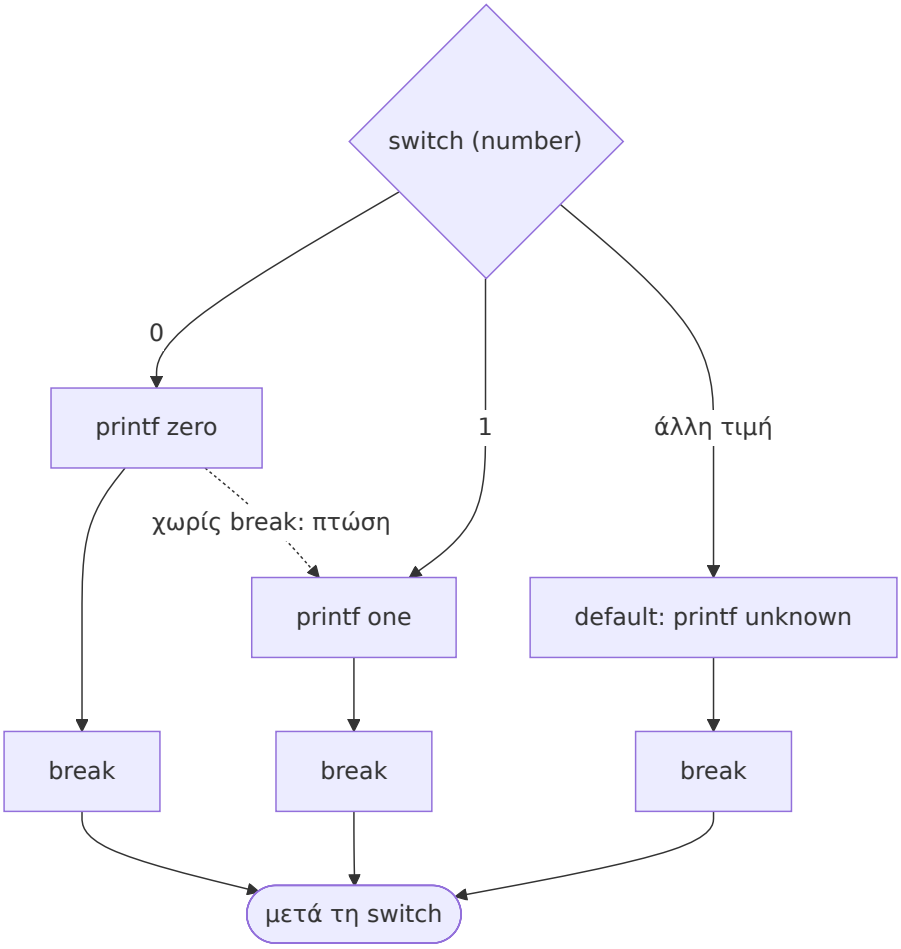
Ένα συχνό μοτίβο είναι μια αλυσίδα `else if` που συγκρίνει την **ίδια** έκφραση με διαδοχικές σταθερές: «αν `number == 0` τύπωσε `zero`, αλλιώς αν `number == 1` τύπωσε `one`, ...». Η **εντολή** `switch` **(switch statement)** είναι η εναλλακτική της δομής `if-else-if` γι' αυτή την περίπτωση: ελέγχει μία έκφραση για τις δυνατές τιμές της και χειρίζεται κάθε περίπτωση διαφορετικά. Η γενική μορφή:

```
switch (έκφραση) {
```

```
  case σταθερά1:
    εντολές
  case σταθερά2:
    εντολές
  ...
  default:
    εντολές
}
```

Υπολογίζεται μία φορά η τιμή της έκφρασης. Αν είναι ίση με τη σταθερά κάποιου `case`, η εκτέλεση **μεταπηδά** στις εντολές μετά από αυτό το `case`· αλλιώς μεταπηδά στο `default`. Το `default` είναι προαιρετικό: αν λείπει και δεν ταιριάζει κανένα `case`, η εκτέλεση συνεχίζει μετά τη `switch`. Οι εντολές ενός `case` δεν χρειάζονται αγκύλες.

Το κρίσιμο σημείο είναι ότι τα `case` είναι απλώς **σημεία εισόδου**, όχι χωριστά blocks. Αφού η εκτέλεση μπει σε ένα `case`, συνεχίζει προς τα κάτω, περνώντας και στις εντολές των επόμενων `case`, μέχρι να βρει `break` ή το τέλος της `switch`. Αυτό λέγεται **πτώση (fall-through)**. Γι' αυτό κάθε ομάδα εντολών κλείνει με `break`: η `break` ολοκληρώνει την εκτέλεση της `switch` και η ροή συνεχίζει μετά την κλειστή αγκύλη της. Είναι η ίδια `break` με αυτή των βρόχων, με δεύτερη χρήση.



Σχήμα: η `switch` *μεταπηδά στο* `case` *που ταιριάζει· χωρίς* `break` *η εκτέλεση «πέφτει» στο επόμενο.*

Το `break` στο τελευταίο `default` δεν αλλάζει τίποτα, αλλά το βάζουμε για ομοιομορφία και για ασφάλεια αν αργότερα προστεθεί `case` από κάτω.

§8.7 Πολλά case μαζί και περιορισμοί της switch

Η πτώση έχει και μια χρήσιμη εφαρμογή: **πολλά** `case` **μαζί**. Γράφουμε τα `case` το ένα κάτω από το άλλο, χωρίς εντολές ανάμεσα (`case 12: case 1: case 2:`), και βάζουμε τις κοινές εντολές και ένα `break` μετά το τελευταίο. Για τιμή 12, 1 ή 2 η εκτέλεση μπαίνει σε κάποιο από τα τρία σημεία και πέφτει στις κοινές εντολές. Εκτός από αυτή την περίπτωση, οι σημειώσεις θεωρούν κακή πρακτική να πέφτει ένα `case` στο επόμενο.

Η `switch` είναι λιγότερο γενική από την αλυσίδα `else if`:

1. Σε κάθε `case` ελέγχεται **μόνο ισότητα** (`==`). Δεν γράφονται άλλες λογικές συνθήκες (π.χ. `>`, `<`)· για διαστήματα τιμών χρειαζόμαστε `if`.

2. Η έκφραση της `switch` και η σταθερά κάθε `case` πρέπει να είναι **ακέραιες** (ένα `double` δεν επιτρέπεται, ένα `char` όπως `'a'` επιτρέπεται). Οι σημειώσεις προσθέτουν ότι στα `case` μπαίνουν σταθερές, όχι μεταβλητές, και ότι δύο `case` δεν μπορούν να έχουν την ίδια σταθερά.

§8.8 Η εντολή goto και οι ετικέτες

Η **εντολή** `goto` **(goto statement)** μεταφέρει την εκτέλεση του προγράμματος σε άλλη εντολή **της ίδιας συνάρτησης**, με την προϋπόθεση ότι η εντολή αυτή έχει μια **ετικέτα (label)**. Η ετικέτα είναι ένα όνομα ακολουθούμενο από άνω-κάτω τελεία, μπροστά από μια εντολή:

```
goto label;
...
label:
```

Μετά την `goto label;` εκτελείται η εντολή μετά το `label:`, πιο πάνω ή πιο κάτω στη συνάρτηση. Έτσι μπορούμε π.χ. να βγούμε από έναν βρόχο πηδώντας σε μια ετικέτα αμέσως μετά από αυτόν. Οι σημειώσεις αναφέρουν ως εξειδικευμένη χρήση την έξοδο από **εμφωλευμένους** βρόχους μονομιάς, που η `break` δεν κάνει. Μια ετικέτα πρέπει να ακολουθείται από εντολή· στο τέλος ενός block βάζουμε την κενή εντολή `;`.

§8.9 Δομημένος προγραμματισμός: γιατί όχι goto

Ο **δομημένος προγραμματισμός (structured programming)** φτιάχνει τα προγράμματα μόνο από ακολουθία, επιλογή (`if`, `switch`) και επανάληψη (βρόχους). Η `goto` σπάει αυτή την ιδέα: η εκτέλεση πηδά οπουδήποτε στη συνάρτηση, οπότε για να καταλάβετε μια γραμμή πρέπει να ξέρετε κάθε `goto` που οδηγεί σε αυτήν. Το αποτέλεσμα είναι δυσνόητος **κώδικας-μακαρονάδα (spaghetti code)**, χρήσιμος μόνο σε διαγωνισμούς δυσνόητου κώδικα (obfuscation contests, όπως ο IOCCC). Το επιχείρημα έγινε γνωστό με το άρθρο του Edsger Dijkstra *Go To Statement Considered Harmful*.

Η απάντηση της διάλεξης στο «να χρησιμοποιήσω goto;» είναι σκέτο **ΟΧΙ**: μπορεί κανείς να γράψει εκατομμύρια γραμμές κώδικα χωρίς να τη χρειαστεί. Στο πλαίσιο αυτού του μαθήματος, σύμφωνα με τις σημειώσεις, η χρήση της **απαγορεύεται**. Αναδιοργανώστε τη ροή με `break`, `continue`, μια σημαία ή μια συνάρτηση.

Παραδείγματα

§8.10 Το γινόμενο των τριψήφιων περιττών πολλαπλασίων του 7

Εφαρμόζει: «Δύο τρόποι να διατρέξουμε τις τιμές», «Υπερχείλιση σε βρόχους που πολλαπλασιάζουν». Οι δύο βρόχοι των διαφανειών, σε ένα πρόγραμμα:

```
#include <stdio.h>
```

```c
int main(int argc, char **argv) {
  int product, prod, i;

  // Τρόπος 1: φιλτράρισμα με if
  // (με αρχή 100 και βήμα 2 περνάμε από τους άρτιους· για περιττούς: 101)
  for (product = 1, i = 100; i <= 999; i += 2) {
    if (i % 7 == 0)
      product *= i;
  }

  // Τρόπος 2: αρχή 105 = 15 * 7, βήμα 14 = 2 * 7
  for (prod = 1, i = 105; i <= 999; i += 14) {
    prod *= i;
  }
  printf("%d\n", prod);
  return 0;
}
```

Η έξοδος που έδειξε η διάλεξη:

```
$ ./prod
-1187656959
```

Ένας αρνητικός αριθμός από γινόμενο θετικών είναι το χαρακτηριστικό σύμπτωμα της υπερχείλισης: το σωστό αποτέλεσμα έχει πάνω από 128 ψηφία. Δοκιμάστε να τυπώνετε το `prod` σε κάθε επανάληψη για να δείτε σε ποιο βήμα «σπάει». (Ο τρόπος 1, όπως είναι γραμμένος, πολλαπλασιάζει άρτιους αριθμούς και υπερχειλίζει κι αυτός.)

§8.11 Ο μικρότερος τριψήφιος: σημαία, break και goto

Εφαρμόζει: «Πρόωρος τερματισμός με μεταβλητή-σημαία», «Η εντολή break», «Η εντολή goto και οι ετικέτες». Το πρόβλημα είναι να τυπωθεί ο μικρότερος τριψήφιος που είναι πολλαπλάσιο του 2 και του 5 αλλά όχι του 4. Η πρώτη απόπειρα των διαφανειών:

```c
for (i = 100; i < 1000; i++) {
  if (i % 2 == 0 && i % 5 == 0 && i % 4 != 0) {
    printf("%d\n", i);
  }
}
```

«Υπάρχει κάποιο θέμα με αυτή την υλοποίηση;» Ναι: τυπώνει όλους τους 45 τέτοιους αριθμούς (110, 130, ..., 990), όχι μόνο τον μικρότερο. Δύο διορθώσεις, σε ένα πρόγραμμα:

```c
#include <stdio.h>
```

```c
int main(int argc, char **argv) {
  int i;

  // 1. Με μεταβλητή-σημαία στη συνθήκη
  int found = 0;
  for (i = 100; i < 1000 && !found; i++) {
    if (i % 2 == 0 && i % 5 == 0 && i % 4 != 0) {
      printf("%d\n", i);
      found = 1;
    }
  }

  // 2. Με break
  for (i = 100; i < 1000; i++) {
    if (i % 2 == 0 && i % 5 == 0 && i % 4 != 0) {
      printf("%d\n", i);
      break;
    }
  }
  return 0;
}
```

```
$ ./smallest
110
110
```

Η τρίτη εκδοχή των διαφανειών χρησιμοποιεί `goto` σε μια ετικέτα αμέσως μετά τον βρόχο. Τη δείχνουμε μόνο για να την αναγνωρίζετε· στο μάθημα δεν επιτρέπεται:

```c
for (i = 100; i < 1000; i++) {
  if (i % 2 == 0 && i % 5 == 0 && i % 4 != 0) {
    printf("%d\n", i);
    goto exit_for;
  }
}
exit_for:
  ;
```

Η έκδοση με `break` είναι η πιο σύντομη και λέει καθαρά την πρόθεση. Προσέξτε μια λεπτή διαφορά: μετά τον βρόχο με σημαία το `i` είναι 111 (έγινε ένα `i++` ακόμα), ενώ μετά την `break` είναι 110.

§8.12 Όλοι οι τριψήφιοι εκτός από τα πολλαπλάσια του 5

Εφαρμόζει: «Η εντολή continue». Όταν το `i` διαιρείται με το 5, η `continue` παραλείπει το `printf` και η `for` προχωρά στο `i++`:

```
for (i = 100; i < 1000; i++) {
  if (i % 5 == 0) {
    continue;
  }
  printf("%d\n", i);
}
```

Τυπώνονται 101, 102, 103, 104, 106, … : 720 αριθμοί συνολικά (900 τριψήφιοι μείον 180 πολλαπλάσια του 5). Ισοδύναμα θα γράφαμε `if (i % 5 != 0) printf(...)`.

§8.13 Το αγγλικό όνομα κάθε ψηφίου

Εφαρμόζει: «Η εντολή switch». Με αλυσίδα `else if`, όπως στις διαφάνειες:

```
if (number == 0)
  printf("zero\n");
else if (number == 1)
  printf("one\n");
....
else if (number == 9)
  printf("nine\n");
else
  printf("unknown\n");
```

Όλες οι συνθήκες συγκρίνουν το ίδιο `number` με σταθερές, άρα ταιριάζει η `switch`. Το πλήρες πρόγραμμα (οι διαφάνειες παραλείπουν τα `case` 2 ως 8):

```
#include <stdio.h>

int main(int argc, char **argv) {
  for (int number = -1; number <= 10; number++) {
    switch (number) {
      case 0: printf("zero\n"); break;
      case 1: printf("one\n"); break;
      case 2: printf("two\n"); break;
      case 3: printf("three\n"); break;
      case 4: printf("four\n"); break;
      case 5: printf("five\n"); break;
      case 6: printf("six\n"); break;
      case 7: printf("seven\n"); break;
      case 8: printf("eight\n"); break;
```

```
      case 9: printf("nine\n"); break;
      default: printf("unknown\n"); break;
    }
  }
  return 0;
}
```

Τυπώνει unknown, μετά zero ως nine και ξανά unknown για το 10. Αν σβήσετε το break του case 8, για number == 8 θα τυπωθούν eight και nine: η εκτέλεση πέφτει στο επόμενο case.

§8.14 Η εποχή κάθε μήνα

Εφαρμόζει: «Πολλά case μαζί και περιορισμοί της switch». Οι διαφάνειες δείχνουν χειμώνα, άνοιξη και καλοκαίρι· εδώ έχει προστεθεί και το φθινόπωρο:

```
switch (month) {
  case 12: case 1: case 2:
    printf("winter\n");
    break;
  case 3: case 4: case 5:
    printf("spring\n");
    break;
  case 6: case 7: case 8:
    printf("summer\n");
    break;
  case 9: case 10: case 11:
    printf("autumn\n");
    break;
  default:
    printf("unknown month\n");
    break;
}
```

Εδώ η switch είναι φυσική επιλογή: ο μήνας είναι ακέραιος και κάθε περίπτωση είναι ισότητα με σταθερά. Αν όμως θέλατε «χειμώνας αν month <= 2», η switch δεν μπορεί να το εκφράσει· θα χρειαζόταν if.

§8.15 Εφαρμογή στο εργαστήριο

Στο Εργαστήριο 3, η άσκηση birthdate.c βρίσκει την ημέρα της εβδομάδας από το IDAY % 7 (0 για Saturday, 1 για Sunday, ..., 6 για Friday): το ίδιο μοτίβο με το όνομα ψηφίου, άρα καλή ευκαιρία για switch.

Κύρια σημεία

1. Ένα γινόμενο ή μεγάλο άθροισμα σε βρόχο ξεπερνά εύκολα τα όρια του `int`· ένα αρνητικό αποτέλεσμα από θετικούς παράγοντες είναι σύμπτωμα υπερχείλισης.
2. Έναν βρόχο μπορούμε να τον γράψουμε φιλτράροντας τις τιμές με `if` ή διαλέγοντας αρχή και βήμα ώστε να περνά μόνο από τις τιμές που θέλουμε· στη δεύτερη περίπτωση ελέγχουμε με το χέρι τις πρώτες τιμές.
3. Όταν θέλουμε μόνο το πρώτο αποτέλεσμα, ο βρόχος πρέπει να σταματά μόλις το βρει, αλλιώς τυπώνει όλα τα αποτελέσματα.
4. Μια μεταβλητή-σημαία (`found`) στη συνθήκη του βρόχου σταματά τον βρόχο και μας λέει μετά αν βρέθηκε κάτι.
5. Η `break` τερματίζει αμέσως τον πιο εσωτερικό βρόχο (`while`, `do-while`, `for`) ή `switch` που την περιέχει.
6. Η `continue` διακόπτει την τρέχουσα επανάληψη και συνεχίζει στην επόμενη· στη `for` εκτελείται πρώτα το βήμα.
7. Η `switch` είναι εναλλακτική της αλυσίδας `if-else-if` όταν ελέγχουμε μία έκφραση για τις δυνατές τιμές της.
8. Τα `case` είναι σημεία εισόδου: χωρίς `break` η εκτέλεση συνεχίζει στο επόμενο `case`· το `break` ολοκληρώνει την εκτέλεση της `switch`.
9. Πολλά `case` γραμμένα το ένα κάτω από το άλλο μοιράζονται τις ίδιες εντολές.
10. Σε κάθε `case` ελέγχεται μόνο ισότητα (όχι `>`, `<` κ.λπ.), και η έκφραση της `switch` και οι σταθερές των `case` πρέπει να είναι ακέραιες.
11. Η `goto label;` μεταφέρει την εκτέλεση στην εντολή με ετικέτα `label:` μέσα στην ίδια συνάρτηση.
12. Δομημένος προγραμματισμός σημαίνει όχι `goto`: οδηγεί σε δυσνόητο κώδικα (spaghetti code) και στο μάθημα δεν τη χρησιμοποιούμε.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
υπερχείλιση	overflow	Αποτέλεσμα που δεν χωράει στον τύπο της μεταβλητής
απροσδιόριστη συμπεριφορά	undefined behavior	Περίπτωση όπου η C δεν ορίζει τι θα συμβεί
μεταβλητή-σημαία	flag	Μεταβλητή 0/1 που καταγράφει αν συνέβη κάτι
εντολή break	break statement	Τερματίζει αμέσως τον βρόχο ή τη `switch`
εντολή continue	continue statement	Προχωρά στην επόμενη επανάληψη του βρόχου

Ελληνικά	English	Σύντομος ορισμός
εντολή switch	switch statement	Επιλογή περίπτωσης με βάση μια ακέραια τιμή
περίπτωση	case	Σημείο εισόδου της `switch` για μια σταθερά
προεπιλογή	default	Η περίπτωση όταν δεν ταιριάζει κανένα `case`
πτώση	fall-through	Συνέχιση στο επόμενο `case` όταν λείπει το `break`
εντολή goto	goto statement	Άλμα σε εντολή με ετικέτα στην ίδια συνάρτηση
ετικέτα	label	Όνομα με `:` μπροστά από εντολή, στόχος της `goto`
δομημένος προγραμματισμός	structured programming	Προγράμματα μόνο από ακολουθία, επιλογή, επανάληψη
κώδικας-μακαρονάδα	spaghetti code	Κώδικας με μπερδεμένη ροή, δύσκολος στην ανάγνωση

Διάβασμα

- **Διαφάνειες**: Διάλεξη 8, σελ. 1–25. Υπερχείλιση στο γινόμενο: σελ. 2· παραδείγματα βρόχων και σημαία: σελ. 5–8· `break`: σελ. 9–10· `continue`: σελ. 11–12· `switch`: σελ. 13–18· `goto` και δομημένος προγραμματισμός: σελ. 19–22.
- **Σημειώσεις**: Κεφάλαιο 3: Η ροή του ελέγχου, ενότητες «Εντολή `switch`», «Εντολές `break` και `continue`», «Εντολή `goto` και ετικέτες» (K04, σελ. 50–51 και 56–57). Οι διαφάνειες συνιστούν να έχετε καλύψει τις σημειώσεις του κ. Σταματόπουλου μέχρι και τη σελίδα 71, δηλαδή και το Κεφάλαιο 4: Συναρτήσεις μέχρι την ενότητα «Μετατροπές μεταξύ δεκαδικών και δυαδικών αριθμών».
- **Εργαστήριο**: Εργαστήριο 3: ασκήσεις `seq.c` (βρόχοι), `birthdate.c` (αντιστοίχιση `IDAY % 7` σε ημέρα, κατάλληλη για `switch`)
- **Άλλα**:
  - Break and Continue (W3Schools)
  - Switch statement (GeeksforGeeks)
  - Wikipedia: Considered harmful, Spaghetti code
  - IOCCC: διαγωνισμός δυσνόητου κώδικα C

Συχνά λάθη

- **Ξεχασμένο `break` σε `case`.** Για `number == 0` τυπώνονται `zero` και `one` (και ό,τι ακολουθεί μέχρι το επόμενο `break`). Διόρθωση: κάθε ομάδα εντολών κλείνει με `break`, εκτός αν θέλετε

σκόπιμα να ομαδοποιήσετε `case`.

- **Συνθήκη ή διάστημα σε `case`.** `case month <= 2:` δεν μεταγλωττίζεται, γιατί το `case` θέλει σταθερά. Διόρθωση: απαριθμήστε τις τιμές (`case 1: case 2:`) ή γράψτε `if`.
- **`switch` σε `double` ή σε μεταβλητή στο `case`.** Ο gcc απαντά `switch quantity not an integer` ή `case label does not reduce to an integer constant`. Διόρθωση: ακέραια έκφραση και σταθερές.
- **Βρόχος που «βρίσκει» και δεν σταματά.** Ζητείται ο μικρότερος και τυπώνονται όλοι. Διόρθωση: `break` (ή σημαία) μόλις βρεθεί το πρώτο.
- **`break` για έξοδο από δύο βρόχους.** Η `break` βγαίνει μόνο από τον εσωτερικό· ο εξωτερικός συνεχίζει. Διόρθωση: σημαία που ελέγχεται και στον εξωτερικό βρόχο, ή μεταφορά των βρόχων σε συνάρτηση που κάνει `return`.
- **`continue` σε `while` πριν από το βήμα.** Σε `while (i < N) { if (...) continue; i++; }` το `i++` παραλείπεται και ο βρόχος γίνεται ατέρμονας. Διόρθωση: αυξήστε τον μετρητή πριν από την `continue`, ή χρησιμοποιήστε `for`.
- **Υπερχείλιση.** Ένα γινόμενο θετικών βγαίνει αρνητικό ή μηδέν. Διόρθωση: εκτιμήστε το μέγεθος του αποτελέσματος και επιλέξτε τύπο (`long long`).
- **Λάθος αρχή ή βήμα.** Με `i = 100; i += 2` περνάτε από άρτιους αντί για περιττούς. Διόρθωση: γράψτε στο χαρτί τις πρώτες τιμές του μετρητή.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K8.5** while(--i) με `i = 0` (17% σωστές): Το 43% απάντησε ότι θα τυπώνει @ επ' άπειρον, χάνοντας ότι το `;` είναι η (κενή) εντολή του βρόχου και ότι το `i` κάποτε υπερχειλίζει και φτάνει ξανά στο 0.
- **K8.4** for χωρίς συνθήκη με break (36% σωστές): Το 32% απάντησε ότι τελειώνει «μόνο όταν λιώσει η CPU», θεωρώντας ότι το `i` δεν θα γίνει ποτέ 0· όμως μετά τη μέγιστη τιμή του ο ακέραιος υπερχειλίζει σε αρνητικούς και φτάνει ξανά στο 0.
- **K8.3** switch και if-else (46% σωστές): Το 53% απάντησε False, επειδή τα `case` δέχονται μόνο ακέραιες σταθερές· όμως μπορούμε να κάνουμε `switch` στην ίδια τη συνθήκη (`switch (x > 3.5)`), που είναι 0 ή 1.

Ερωτήσεις κατανόησης

- **E8.1** Γιατί ένα γινόμενο θετικών ακεραίων σε βρόχο μπορεί να τυπωθεί αρνητικό;[69]
- **E8.2** Ποιες τιμές παίρνει το `i` στο `for (i = 105; i <= 999; i += 14)` και γιατί το βήμα είναι 14;[70]

[69] Επειδή το γινόμενο ξεπερνά τη μέγιστη τιμή του `int` (υπερχείλιση)· στην πράξη κρατιούνται μόνο τα χαμηλά 32 bit και το bit του προσήμου μπορεί να γίνει 1. Για προσημασμένους ακεραίους είναι απροσδιόριστη συμπεριφορά.

[70] 105, 119, 133, ..., 987 (64 τιμές, η τελευταία το $105 + 63 \cdot 14 = 987$). Το 14 είναι $2 \cdot 7$: κρατά τον αριθμό

- **E8.3** Ποια η διαφορά ανάμεσα σε `break` και `continue`;[71]
- **E8.4** Σε μια `for`, εκτελείται το βήμα (`i++`) μετά από `continue`; Μετά από `break`;[72]
- **E8.5** Τι θα τυπωθεί αν σε μια `switch` λείπουν όλα τα `break` και ταιριάξει το πρώτο `case`;[73]
- **E8.6** Πώς γράφουμε ότι οι μήνες 12, 1 και 2 έχουν την ίδια συμπεριφορά σε μια `switch`;[74]
- **E8.7** Μπορεί μια `switch` να χειριστεί τη συνθήκη `x > 100`; Γιατί;[75]
- **E8.8** Σε ποιο σημείο μπορεί να μεταφέρει την εκτέλεση μια `goto`, και γιατί την αποφεύγουμε;[76]
- **E8.9** Μετά από τον βρόχο με τη σημαία `found`, πώς ξέρουμε αν βρέθηκε αριθμός;[77]

Kahoot από το αμφιθέατρο (K8.1–K8.5)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K8.1** goto στο διαγώνισμα: 97% σωστές απαντήσεις
- **K8.2** Μόνο με break σταματά ένας βρόχος;: 94% σωστές απαντήσεις
- **K8.3** switch και if-else: 46% σωστές απαντήσεις
- **K8.4** for χωρίς συνθήκη με break: 36% σωστές απαντήσεις
- **K8.5** while(--i) με i = 0: 17% σωστές απαντήσεις

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A8.1–A8.6)

- **A8.1** Το αγγλικό όνομα κάθε ψηφίου: Διάλεξη 8: Ροή Ελέγχου #2, διαφάνεια 13 · ★☆☆ · programming · `slides-lec08-digit-names`
- **A8.2** Υπάρχει θέμα με αυτή την υλοποίηση;: Διάλεξη 8: Ροή Ελέγχου #2, διαφάνεια 7 · ★☆☆ · debug · `slides-lec08-print-all-issue`
- **A8.3** Γινόμενο τριψήφιων περιττών πολλαπλασίων του 7: Διάλεξη 8: Ροή Ελέγχου #2, διαφάνεια 5 · ★☆☆ · programming · `slides-lec08-product-multiples-of-7`
- **A8.4** Αρνητικό γινόμενο σε βρόχο: Διάλεξη 8: Ροή Ελέγχου #2, διαφάνεια 2 · ★☆☆ · debug · `slides-lec08-product-overflow`
- **A8.5** Η εποχή κάθε μήνα: Διάλεξη 8: Ροή Ελέγχου #2, διαφάνεια 17 · ★☆☆ · programming · `slides-lec08-season`

πολλαπλάσιο του 7 και περιττό.

[71] Η `break` τερματίζει ολόκληρο τον βρόχο· η `continue` τερματίζει μόνο την τρέχουσα επανάληψη και ο βρόχος συνεχίζει με την επόμενη.

[72] Μετά από `continue` ναι (μετά γίνεται ο έλεγχος της συνθήκης)· μετά από `break` όχι, ο βρόχος τελειώνει αμέσως.

[73] Οι εντολές όλων των `case` από εκεί και κάτω, μαζί με του `default`, μέχρι το τέλος της `switch` (fall-through).

[74] `case 12: case 1: case 2:` το ένα κάτω από το άλλο, και μετά τις κοινές εντολές με ένα `break` στο τέλος.

[75] Όχι. Κάθε `case` ελέγχει μόνο ισότητα με μια ακέραια σταθερά· για συγκρίσεις όπως `>` χρειάζεται `if`.

[76] Σε οποιαδήποτε εντολή με ετικέτα μέσα στην ίδια συνάρτηση. Την αποφεύγουμε γιατί κάνει τη ροή δυσνόητη (spaghetti code) και σπάει τον δομημένο προγραμματισμό· στο μάθημα απαγορεύεται.

[77] Από την τιμή της σημαίας: αν `found == 1` βρέθηκε και τυπώθηκε, αν έμεινε 0 δεν υπάρχει τέτοιος αριθμός.

- **A8.6** Ο μικρότερος τριψήφιος πολλαπλάσιο του 2 και του 5 αλλά όχι του 4: Διάλεξη 8: Ροή Ελέγχου #2, διαφάνεια 6 · ★☆☆ · programming · `slides-lec08-smallest-three-digit`

Εργασίες (A8.7)

- **A8.7** Η Μέθοδος Newton-Raphson: Εργασία 1 (2023-24), Άσκηση 1 · ★★☆ · programming · `hw-2023-hw1-newton`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A9.15** Συνεργασία (Prisoner's Dilemma): Εργασία 2 (2023-24), Άσκηση 3 · ★★☆ · programming · `hw-2023-hw2-coop`
- **A11.21** Ο Αλγόριθμος RSA (rsa): Εργασία 1 (2024-25), Άσκηση 2 · ★★★ · programming · `hw-2024-hw1-rsa`
- **A12.14** Τουρνουά: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #2 (Pokémon Themed), Θέμα 2 · ★☆☆ · programming · `exam-2023-fall-ex2-q2`
- **A12.9** Ορίσματα γραμμής εντολής: Εργαστήριο 8, Άσκηση 2 · ★☆☆ · programming · `lab-lab08-argcalc`
- **A14.20** Σπάσε το PIN: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #6 (Crypto Themed), Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex6-q3`
- **A16.4** Παλινδρομικός αριθμός: Διάλεξη 16, διαφάνεια 21 · ★☆☆ · programming · `slides-lec16-palindrome-number`
- **A16.5** Αντιστροφή ψηφίων αριθμού: Διάλεξη 16, διαφάνεια 13 · ★☆☆ · programming · `slides-lec16-reverse-digits`

Κεφάλαιο 9: Δεδομένα Εισόδου

> **Στόχοι**: μετά από αυτό το κεφάλαιο θα μπορείτε να εξηγείτε γιατί η σύγκριση `float` με `==` είναι επικίνδυνη, να ονομάζετε τις τέσσερις πηγές δεδομένων εισόδου, να διαβάζετε χαρακτήρες με `getchar` μέχρι το `EOF` και να τους γράφετε με `putchar`, να εξηγείτε τον ρόλο του buffer και του Ctrl+D, και να διαβάζετε αριθμούς με `scanf` ελέγχοντας την τιμή επιστροφής της.
>
> **Προαπαιτούμενα**: Κεφάλαιο 2, Κεφάλαιο 6, Κεφάλαιο 8
>
> **Χρόνος μελέτης**: ~2 ώρες

Σύνοψη

Ένα πρόγραμμα παίρνει δεδομένα εισόδου, τα επεξεργάζεται και παράγει δεδομένα εξόδου. Η διάλεξη ξεκινά με δύο διευκρινίσεις (γιατί ένα `float` δεν είναι «ίσο» με το `3.1` και πώς ονομάζουμε συναρτήσεις) και περνάει στις τέσσερις πηγές εισόδου: ορίσματα γραμμής εντολών, πρότυπη είσοδο (`stdin`), αρχεία και δίκτυο. Εστιάζει στην πρότυπη είσοδο: στη `getchar`, που διαβάζει έναν χαρακτήρα τη φορά και επιστρέφει `EOF` στο τέλος, στην αδελφή της `putchar`, και στη `scanf`, τη «συμμετρική» της `printf` για διάβασμα. Στην πορεία βλέπουμε τρεις κλασικές παγίδες: `char` αντί για `int` για την τιμή της `getchar`, `scanf` χωρίς `&`, και `scanf` χωρίς έλεγχο της τιμής επιστροφής. Σχεδόν κάθε εργασία του μαθήματος διαβάζει είσοδο, οπότε αυτά τα εργαλεία θα τα χρησιμοποιείτε συνέχεια.

Θεωρία

§9.1 Σύγκριση αριθμών κινητής υποδιαστολής με ==

Ο τελεστής `==` ελέγχει αν δύο τιμές είναι **ακριβώς** ίσες. Οι αριθμοί κινητής υποδιαστολής (floating point) αποθηκεύονται σε δυαδική μορφή, και οι περισσότεροι δεκαδικοί, όπως το `3.1`, **δεν αναπαρίστανται ακριβώς**: αποθηκεύεται η πλησιέστερη τιμή που χωράει. Το `float` (συνήθως 4 bytes) κρατά λιγότερα ψηφία από το `double` (8 bytes), και μια σταθερά όπως το `3.1` είναι `double`. Έτσι, μετά το `float a = 3.1;` το `a` κρατά περίπου `3.0999999046`, ενώ στο `a == 3.1` η σύγκριση γίνεται σε `double` με το `3.1000000000…`: οι τιμές διαφέρουν και η συνθήκη είναι ψευδής (δείτε το παράδειγμα).

Κανόνας: **μη συγκρίνετε αριθμούς κινητής υποδιαστολής με `==`**. Ελέγξτε αν η απόλυτη διαφορά τους είναι μικρότερη από μια μικρή ανοχή, π.χ. `fabs(a - 3.1) < 1e-6` (η `fabs` είναι στο `math.h`).

§9.2 Ονόματα: camelCase και snake_case

Για ονόματα από πολλές λέξεις υπάρχουν δύο συνηθισμένα στυλ: **camelCase**, όπου κάθε λέξη μετά την πρώτη ξεκινά με κεφαλαίο (`piApprox`), και **snake_case**, με μικρά γράμματα και κάτω παύλα (`pi_approx`). Για κώδικα C (και Python) η διάλεξη συνιστά **snake_case**: το `pi_approx` είναι (υποκειμενικά) καλύτερο από το `piApprox`. Ο μεταγλωττιστής δέχεται και τα δύο· σημασία έχει να τηρείτε ένα στυλ.

§9.3 Δεδομένα εισόδου και εξόδου

Κάθε πρόγραμμα παίρνει **δεδομένα εισόδου** (input data), τα επεξεργάζεται και παράγει **δεδομένα εξόδου** (output data).

Σχήμα: το πρόγραμμα μετατρέπει δεδομένα εισόδου σε δεδομένα εξόδου.

Τα δεδομένα εισόδου είναι **μια σειρά από χαρακτήρες (bytes)** που δίνει ο χρήστης στο πρόγραμμα. Όταν πληκτρολογείτε τον «αριθμό» `42`, το πρόγραμμα λαμβάνει δύο χαρακτήρες, `'4'` και `'2'`· η μετατροπή τους σε ακέραιο είναι δική του δουλειά.

§9.4 Οι τέσσερις πηγές εισόδου

1. **Ορίσματα** στη γραμμή εντολών (command-line arguments), π.χ. `./grade 80 100 90`.
2. **Πρότυπη είσοδος** (standard input, `stdin`): κείμενο που γράφουμε, συνήθως με το πληκτρολόγιο, ενώ το πρόγραμμα τρέχει. Είναι το θέμα αυτής της διάλεξης.
3. **Αρχεία** από το σύστημα αρχείων, όπως κάνει η `cat hello.txt` (επόμενες διαλέξεις).
4. **Δίκτυο** ή άλλες πηγές, π.χ. ένα User Interface (επόμενα εξάμηνα). Η `curl`, για παράδειγμα, διαβάζει μια σελίδα από το δίκτυο.

Με τα δύο πρώτα έχουμε καλύψει το «50%». Η πρότυπη είσοδος δεν είναι υποχρεωτικά το πληκτρολόγιο: με ανακατεύθυνση (`<`) ή σωλήνωση (`|`) το ίδιο πρόγραμμα διαβάζει από αρχείο ή από την έξοδο άλλου προγράμματος, χωρίς να το «ξέρει».

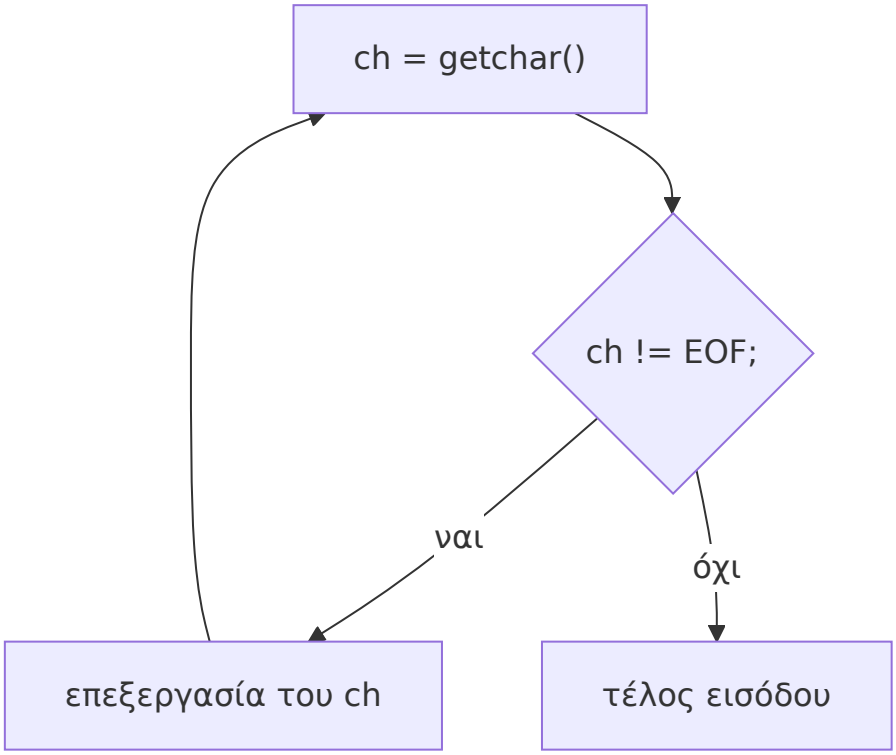
§9.5 Η συνάρτηση getchar

Η `getchar`, με μορφή `int getchar();`, ορίζεται στο `stdio.h`. Δεν παίρνει όρισμα· διαβάζει **έναν** χαρακτήρα από το `stdin`, *εάν υπάρχει*, και τον επιστρέφει ως ακέραιο (τον κωδικό ASCII του). Αν δεν υπάρχει άλλος, επιστρέφει την ειδική τιμή `EOF` (End-Of-File, `-1`). Ο τύπος επιστροφής είναι `int` και όχι `char` ώστε να χωράει, εκτός από όλους τους δυνατούς χαρακτήρες, και το `EOF`, που δεν μπερδεύεται με κανέναν από αυτούς. Για τη συμπεριφορά μιας έτοιμης συνάρτησης της βιβλιοθήκης ανοίγουμε ένα τερματικό και τρέχουμε `man getchar`.

Κάθε κλήση διαβάζει τον **επόμενο** χαρακτήρα, οπότε το βασικό μοτίβο ανάγνωσης όλης της εισόδου είναι:

```
int ch;
while ((ch = getchar()) != EOF) {
  /* επεξεργασία του ch */
}
```

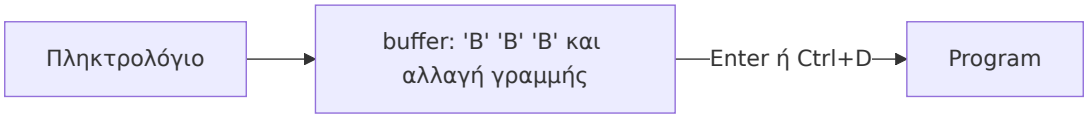
Οι παρενθέσεις γύρω από την ανάθεση είναι απαραίτητες: το `!=` έχει μεγαλύτερη προτεραιότητα από το `=`, και χωρίς αυτές το `ch` θα έπαιρνε το αποτέλεσμα της σύγκρισης (0 ή 1).



Σχήμα: ο βρόχος ανάγνωσης χαρακτήρων μέχρι το EOF.

§9.6 Buffer, Enter και Ctrl+D

Όταν πληκτρολογείτε, οι χαρακτήρες αποθηκεύονται προσωρινά σε έναν **buffer** (ενδιάμεση μνήμη) και προωθούνται στο πρόγραμμα **όταν πατήσετε Enter**, μαζί με την αλλαγή γραμμής `'\n'`. Αυτή η λειτουργία του τερματικού, με είσοδο ανά γραμμή, λέγεται canonical mode. Γι' αυτό η `getchar` «περιμένει» το Enter, και γι' αυτό η γραμμή `BBB` φτάνει ως τέσσερις χαρακτήρες: `'B'`, `'B'`, `'B'`, `'\n'`.



Σχήμα: οι χαρακτήρες περιμένουν στον buffer μέχρι το Enter ή το Ctrl+D.

Για να δηλώσετε ότι **τελείωσαν** τα δεδομένα, στο Linux πατάτε συνήθως **Ctrl+D**. Το Ctrl+D προωθεί ό,τι υπάρχει στον buffer, χωρίς αλλαγή γραμμής. Αν ο buffer είναι άδειος (στην αρχή γραμμής), το πρόγραμμα δεν παίρνει τίποτα και η `getchar` επιστρέφει `EOF`· άρα, αν έχετε γράψει κάτι χωρίς Enter, χρειάζεται δεύτερο Ctrl+D. Όταν η είσοδος έρχεται από αρχείο (`./prog < file`) ή σωλήνωση, το `EOF` έρχεται μόνο του στο τέλος των δεδομένων.

§9.7 Η συνάρτηση putchar

Η `putchar`, με μορφή `int putchar(int c);`, ορίζεται επίσης στο `stdio.h`. Παίρνει έναν χαρακτήρα, τον τυπώνει στο `stdout` και τον επιστρέφει ως ακέραιο· αν κάτι πάει στραβά στο τύπωμα, επιστρέφει `EOF`. Για να τυπώσετε το `C` γράφετε `putchar('C');`. Είναι απλούστερο υποσύνολο της `printf`: το `putchar(ch)` κάνει ό,τι και το `printf("%c", ch)`.

§9.8 Η τιμή της getchar πάει σε int, όχι σε char

Ένα `char` κρατά μόνο 256 τιμές, οπότε δεν μπορεί να ξεχωρίσει το `EOF` από κάθε πραγματικό χαρακτήρα. Στα συνηθισμένα συστήματα, όπου το `char` είναι προσημασμένο, το byte `0xff` (255) αποθηκεύεται ως `-1`, δηλαδή ίσο με το `EOF`: ένα έγκυρο byte μοιάζει με τέλος εισόδου και το πρόγραμμα σταματά πρόωρα (δείτε την `cat` με `char`). Όπου το `char` είναι απρόσημο, το `EOF` γίνεται 255 και η σύγκριση με `EOF` δεν αληθεύει ποτέ, οπότε ο βρόχος δεν τελειώνει.

Κανόνας της διάλεξης: **η μεταβλητή που δέχεται την τιμή της** `getchar` **δηλώνεται** `int`, εκτός αν είμαστε σίγουροι για το τι κάνουμε.

§9.9 Διάβασμα αριθμών χαρακτήρα-χαρακτήρα

Με τη `getchar` μπορούμε να διαβάσουμε και αριθμούς, χτίζοντας την τιμή ψηφίο-ψηφίο: αν `val` είναι η τιμή των ψηφίων ως τώρα, ένα νέο ψηφίο `d` σε βάση `base` δίνει $val \leftarrow base \cdot val + d$. Η τιμή του ψηφίου είναι `ch - '0'`, επειδή τα `'0'`…`'9'` έχουν διαδοχικούς κωδικούς ASCII (48…57), και ένας χαρακτήρας είναι έγκυρο ψηφίο σε βάση έως 10 αν `ch >= '0' && ch <= '0' + base - 1`. Αυτό κάνει η `getinteger`. Υπάρχει όμως και έτοιμος τρόπος: η `scanf`.

§9.10 Η συνάρτηση scanf

Η `scanf` ορίζεται στο header file `stdio.h` και διαβάζει δεδομένα εισόδου **πολλών τύπων** από το `stdin`, αποθηκεύοντας τις τιμές τους σε μεταβλητές. Είναι η **συμμετρική της** `printf`, για διάβασμα αντί για εκτύπωση:

```
int scanf(const char *restrict format, ...);
int printf(const char *restrict format, ...);
```

Το πρώτο όρισμα είναι μια **συμβολοσειρά μορφοποίησης** (format string) με προδιαγραφές όπως `%d` (`int`) ή `%f` (`float`)· τα `...` σημαίνουν ότι δέχεται όσα ορίσματα της περάσουμε, ένα για κάθε προδιαγραφή (θα το δούμε σε άλλο μάθημα).

Η μεγάλη διαφορά: στην `printf` δίνουμε **τιμές**, ενώ στη `scanf` τις **διευθύνσεις** των μεταβλητών, με τον τελεστή `&`: `scanf("%d", &n)`. Η C περνάει τα ορίσματα με τιμή· με `scanf("%d", n)` η `scanf` θα έπαιρνε ένα αντίγραφο της τιμής του `n` και δεν θα μπορούσε να το αλλάξει. Με το `&n` μαθαίνει τη θέση του `n` στη μνήμη και γράφει εκεί την τιμή που διάβασε. Οι διευθύνσεις είναι το θέμα του Κεφαλαίου 11.

§9.11 Κενά και πολλές τιμές

Μία κλήση μπορεί να διαβάσει πολλές τιμές: `scanf("%d %d", &n1, &n2)`. Καθώς ψάχνει για δεκαδικό ψηφίο, η `%d` **προσπερνά κενούς χαρακτήρες και αλλαγές γραμμής** πριν από τον αριθμό, οπότε η είσοδος `2  4` δίνει `n1 = 2` και `n2 = 4`, όπως και αν οι αριθμοί ήταν σε διαφορετικές γραμμές.

§9.12 Η τιμή επιστροφής της scanf

Η `scanf` επιστρέφει **πόσα δεδομένα εισόδου διάβασε** και αποθήκευσε. Αν η είσοδος τελειώσει πριν διαβάσει οτιδήποτε, επιστρέφει `EOF`. Για `scanf("%d", &n)`:

Τιμή επιστροφής	Σημασία	Το `n`
`1`	διαβάστηκε ακέραιος	έχει τη νέα τιμή
`0`	η είσοδος δεν ξεκινά με αριθμό (π.χ. `hello`)	δεν άλλαξε
`EOF`	τέλος εισόδου (π.χ. Ctrl+D)	δεν άλλαξε

Αν δεν ελέγξουμε την τιμή επιστροφής και η ανάγνωση αποτύχει, το πρόγραμμα συνεχίζει με τα «σκουπίδια» μιας μη αρχικοποιημένης μεταβλητής. Άρα ένα πρόγραμμα που δεν ελέγχει τι επέστρεψε η `scanf` **δεν είναι σωστό**. Γράφουμε, για παράδειγμα, `if (scanf("%d", &n) != 1) { printf("Invalid input\n"); return 1; }`.

Παραδείγματα

§9.13 OK ή Not OK;

Εφαρμόζει τη σύγκριση με `==`. Τι θα τυπώσει το παρακάτω πρόγραμμα;

```c
#include <stdio.h>
int main() {
    float a = 3.1;

    if(a == 3.1)
       printf("OK\n");
    else
       printf("Not OK\n");
    return 0;
}
```

```
$ ./one
Not OK
```

Το `a` κρατά το `3.1` στρογγυλεμένο σε `float`, ενώ το `3.1` της σύγκρισης είναι `double` με μεγαλύτερη ακρίβεια. Με `a == 3.1f` (σταθερά `float`) θα τυπωνόταν `OK`, αλλά η ασφαλής λύση είναι η σύγκριση με ανοχή.

§9.14 Διαβάζω και τυπώνω έναν χαρακτήρα

Εφαρμόζει τη `getchar` και τον έλεγχο για `EOF`.

```c
#include <stdio.h>
int main() {
  printf("Gimme a char: ");
  int ch = getchar();
  if (ch != EOF) {
    printf("You gave the char: %c\n", ch);
  } else {
    printf("input ended\n");
  }
  return 0;
}
```

```
$ ./chartest
Gimme a char: BBB
You gave the char: B
```

Η `getchar` διαβάζει **μόνο έναν** χαρακτήρα, το πρώτο `B`· τα `BB\n` μένουν αδιάβαστα. Αν πατήσετε αμέσως Ctrl+D, τυπώνεται `input ended`.

§9.15 Μετρητής χαρακτήρων

Διαδοχικές κλήσεις της `getchar` διαβάζουν διαδοχικούς χαρακτήρες. Τι κάνει το παρακάτω πρόγραμμα;

```c
#include <stdio.h>
int main() {
  int ch, sum = 0;
  printf("Enter characters: ");
  while( (ch = getchar()) != EOF ) {
    printf("%c", ch);
    sum++;
  }
  printf("\nTotal characters: %d\n", sum);
  return 0;
}
```

```
$ ./charcount
Enter characters: we'll always have paris
we'll always have paris
Total characters: 24
```

Τυπώνει ξανά ό,τι διαβάζει και στο τέλος το πλήθος των χαρακτήρων. Η φράση έχει 23 χαρακτήρες· ο 24ος είναι το `'\n'` του Enter. Η γραμμή επαναλαμβάνεται μόλις πατήσετε Enter (τότε φτάνει ο buffer στο πρόγραμμα) και το σύνολο τυπώνεται με το Ctrl+D.

§9.16 Μια cat με char

Εφαρμόζει τις `getchar`/`putchar` και τον κανόνα «`int`, όχι `char`». Τι πρόβλημα έχει η παρακάτω υλοποίηση της `cat`;

```c
#include <stdio.h>

int main() {
  char c;
  while((c = getchar()) != EOF)
    putchar(c);
  return 0;
}
```

```
$ echo -e "hello\xffworld" | ./cat
hello
$ echo -e "hello\xffworld" | cat
hello  world
```

Το `echo -e` βάζει το byte `0xff` ανάμεσα στις δύο λέξεις. Η πραγματική `cat` το αντιγράφει (το τερματικό το δείχνει ως). Η δική μας το αποθηκεύει στο `char c` ως `-1`, το βρίσκει ίσο με `EOF` και σταματά. Η διόρθωση είναι μία λέξη: `int c;`.

§9.17 Η συνάρτηση getinteger

Εφαρμόζει το διάβασμα αριθμών χαρακτήρα-χαρακτήρα. Θέλουμε ένα πρόγραμμα addnums που διαβάζει δύο αριθμούς και τους προσθέτει:

```
$ ./addnums
Give me a number: 40
Give me another number: 2
Total: 42
```

Ένας τρόπος είναι με τη getchar, φτιάχνοντας μια συνάρτηση getinteger που διαβάζει τους χαρακτήρες έναν-έναν και τους μετατρέπει σε αριθμό. Τι κάνει;

```
#define ERROR -1          // Return value for illegal character

int getinteger(int base) {
  int ch;                 // No need to declare ch as int - no EOF handling
  int val = 0;                                  // Initialize return value
  while ((ch = getchar()) != '\n')              // Read up to new line
    if (ch >= '0' && ch <= '0' + base - 1)      // Legal character?
      val = base * val + (ch - '0');            // Update return value
    else
      return ERROR;                             // Illegal character read
  return val;   // Everything OK - Return value of number read
}
```

Διαβάζει μέχρι την αλλαγή γραμμής· για κάθε έγκυρο ψηφίο της βάσης ενημερώνει το val, και με τον πρώτο μη έγκυρο χαρακτήρα επιστρέφει ERROR (-1). Με είσοδο 42 σε βάση 10 το val γίνεται 4 και μετά $10 \cdot 4 + 2 = 42$· με getinteger(2) και είσοδο 101 γίνεται 1, 2, 5. Το addnums προκύπτει καλώντας τη δύο φορές. Ο άλλος τρόπος για το ίδιο αποτέλεσμα είναι η scanf. Στις σημειώσεις η ίδια συνάρτηση μετατρέπει δυαδικούς σε δεκαδικούς και αντίστροφα.

§9.18 Τετράγωνο με scanf

Εφαρμόζει τη scanf. Τι κάνει το παρακάτω πρόγραμμα;

```
#include <stdio.h>
int main() {
  int n;
  printf("Gimme a number: ");
  scanf("%d", &n);
  printf("Square: %d\n", n * n);
  return 0;
}
```

Περνάμε τη **διεύθυνση** της n (&n), ώστε η scanf να μπορέσει να αναθέσει την τιμή που διάβασε,

και τυπώνουμε στο `stdout` το τετράγωνο του αριθμού που γράψαμε στο `stdin`. Τι θα γινόταν αν γράφαμε `scanf("%d", n);`;

```
$ ./scanf
Gimme a number: 3
Segmentation fault
```

Η `scanf` θα χρησιμοποιούσε την (μη αρχικοποιημένη) τιμή του `n` σαν διεύθυνση, και η εγγραφή σε μια τυχαία διεύθυνση που δεν ανήκει στο πρόγραμμα προκαλεί `Segmentation fault`. Ο `gcc -Wall` προειδοποιεί για αυτό το λάθος.

§9.19 Είναι σωστό αυτό το πρόγραμμα;

Εφαρμόζει την τιμή επιστροφής της `scanf`. Το τετράγωνο της προηγούμενης ενότητας **δεν** είναι σωστό, καθώς δεν ελέγχουμε την τιμή επιστροφής της `scanf` (`EOF` ή ίσως `0`):

```
$ ./scanf
Gimme a number: 16
Square: 256
$ ./scanf
Gimme a number: Square:
1068701481
$ ./scanf
Gimme a number: hello
Square: 1072038564
```

Στη δεύτερη εκτέλεση πατήσαμε Ctrl+D (η `scanf` επέστρεψε `EOF`), στην τρίτη γράψαμε `hello` (επέστρεψε `0`). Και στις δύο το `n` έμεινε με την τυχαία αρχική του τιμή και τυπώθηκε ένα «τετράγωνο» χωρίς νόημα.

§9.20 Δύο αριθμοί με μία scanf

Εφαρμόζει τα κενά και τις πολλές τιμές.

```c
#include <stdio.h>
int main() {
  int n1, n2;
  printf("Gimme two numbers: ");
  scanf("%d %d", &n1, &n2);
  printf("Result: %d\n", n1 * n2);
  return 0;
}
```

```
$ ./scanf2
Gimme two numbers:   2   4
Result: 8
```

Τα επιπλέον κενά δεν ενοχλούν, αφού η `%d` τα προσπερνά ψάχνοντας για ψηφίο. Ένα σωστό πρόγραμμα θα έλεγχε και εδώ ότι η `scanf` επέστρεψε 2.

§9.21 Συχνότητες γραμμάτων

Εφαρμόζει τον βρόχο ανάγνωσης μέχρι το `EOF` και «προλαβαίνει» τους πίνακες του Κεφαλαίου 10. Τι κάνει ο κώδικας;

```
int i, ch, total = 0;
int letfr[26];  // Letter occurrences and frequencies array
for (i=0 ; i < 26 ; i++)
  letfr[i] = 0;
while ((ch = getchar()) != EOF) {
  if (ch >= 'A' && ch <= 'Z') {
    letfr[ch-'A']++;            // Found upper case letter
    total++;
  }
  if (ch >= 'a' && ch <= 'z') {
    letfr[ch-'a']++;            // Found lower case letter
    total++;
  }
}
```

Μετρά πόσες φορές εμφανίζεται κάθε λατινικό γράμμα στην είσοδο, χωρίς διάκριση πεζών-κεφαλαίων. Ο πίνακας `letfr` έχει 26 μετρητές, αρχικά 0, και το `ch - 'A'` (ή `ch - 'a'`) δίνει τη θέση του γράμματος στο αλφάβητο (0 για `A`/`a`, 25 για `Z`/`z`). Το `total` μετρά όλα τα γράμματα, για να υπολογιστούν αργότερα συχνότητες. Το πλήρες πρόγραμμα, που τυπώνει και ιστόγραμμα, είναι στις σημειώσεις.

§9.22 Για το εργαστήριο και τις εργασίες

- Στο Εργαστήριο 4 γράφετε φίλτρα με `getchar`/`putchar` (`lowercase.c`, `encode.c`, `decode.c`) και τα δοκιμάζετε με σωληνώσεις, π.χ. `echo hello world | ./encode | ./decode`.
- Όταν μια εκφώνηση ορίζει τι γίνεται στο τέλος της εισόδου ή σε λάθος είσοδο, ξεχωρίστε τις τρεις περιπτώσεις της `scanf`: `1`, `0` και `EOF`. Το πρόγραμμα `trolley` της διάλεξης, π.χ., τερματίζει όταν δεν μπορεί να διαβάσει άλλο κόστος.

Κύρια σημεία

1. Οι αριθμοί κινητής υποδιαστολής δεν αναπαρίστανται ακριβώς και ένα `float` δεν είναι ίσο με την ίδια σταθερά σε `double`· δεν τους συγκρίνουμε με `==`.

2. Για ονόματα σε κώδικα C η διάλεξη συνιστά snake_case (`pi_approx`) αντί για camelCase (`piApprox`).
3. Τα δεδομένα εισόδου είναι μια σειρά από χαρακτήρες (bytes) που δίνει ο χρήστης στο πρόγραμμα, από τέσσερις πηγές: ορίσματα, πρότυπη είσοδο, αρχεία και δίκτυο.
4. Η `getchar` διαβάζει έναν χαρακτήρα από το `stdin` και τον επιστρέφει ως `int`, ή `EOF` (`-1`) αν δεν υπάρχει άλλος.
5. Η `putchar` τυπώνει έναν χαρακτήρα στο `stdout` και είναι απλούστερο υποσύνολο της `printf`.
6. Η είσοδος από το πληκτρολόγιο περνά από buffer και φτάνει στο πρόγραμμα με το Enter (μαζί με το `'\n'`)· το Ctrl+D δηλώνει στο Linux το τέλος της εισόδου.
7. Η τιμή επιστροφής της `getchar` αποθηκεύεται σε `int`, όχι σε `char`, αλλιώς το byte `0xff` μπερδεύεται με το `EOF`.
8. Για να μάθετε πώς συμπεριφέρεται μια συνάρτηση της βιβλιοθήκης, τρέξτε `man getchar`, `man scanf` κ.λπ.
9. Η `scanf` είναι η συμμετρική της `printf` για διάβασμα και παίρνει τις διευθύνσεις των μεταβλητών (`&n`)· χωρίς `&` το πρόγραμμα καταρρέει.
10. Η `scanf` επιστρέφει πόσα δεδομένα διάβασε ή `EOF`· ένα πρόγραμμα που δεν ελέγχει την τιμή επιστροφής της δεν είναι σωστό.
11. Η `%d` της `scanf` προσπερνά κενά και αλλαγές γραμμής πριν από τον αριθμό.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
δεδομένα εισόδου / εξόδου	input / output data	Ό,τι δίνεται στο πρόγραμμα / ό,τι παράγει.
πρότυπη είσοδος	standard input (`stdin`)	Η προεπιλεγμένη είσοδος, συνήθως το πληκτρολόγιο.
πρότυπη έξοδος	standard output (`stdout`)	Η προεπιλεγμένη έξοδος, συνήθως η οθόνη.
τέλος αρχείου	End-Of-File (`EOF`)	Τιμή (`-1`) που δηλώνει ότι δεν υπάρχει άλλη είσοδος.
ενδιάμεση μνήμη	buffer	Όπου περιμένουν οι χαρακτήρες πριν φτάσουν στο πρόγραμμα.
κανονική λειτουργία	canonical mode	Το τερματικό στέλνει την είσοδο ανά γραμμή.
συμβολοσειρά μορφοποίησης	format string	Το πρώτο όρισμα της `printf`/`scanf`, π.χ. `"%d %d"`.
διεύθυνση	address	Η θέση μιας μεταβλητής στη μνήμη, `&n`.
τιμή επιστροφής	return value	Η τιμή που δίνει πίσω μια συνάρτηση.

Ελληνικά	English	Σύντομος ορισμός
κινητή υποδιαστολή	floating point	Αναπαράσταση πραγματικών (`float`, `double`), κατά προσέγγιση.
camelCase / snake_case	camelCase / snake_case	Στυλ ονομάτων: `piApprox` / `pi_approx`.

Διάβασμα

- **Διαφάνειες**: Διάλεξη 9, σελ. 1–41. Σύγκριση `float` και ονόματα: σελ. 2–5· πηγές εισόδου: σελ. 8–13· `getchar`: σελ. 14–21· `putchar` και `cat`: σελ. 22–24· `getinteger`: σελ. 25–26, 38· `scanf`: σελ. 27–37· συχνότητες γραμμάτων: σελ. 39.
- **Σημειώσεις**: οι διαφάνειες παραπέμπουν στις σελίδες 28–29, 70–71, 78–79 και 86–87 των σημειώσεων του κ. Σταματόπουλου, δηλαδή στις ενότητες:
  - Κεφάλαιο 1: Πρώτα προγράμματα σε C, «Μετατροπή πεζών γραμμάτων σε κεφαλαία» (K04, σελ. 28–29)
  - Κεφάλαιο 4: Συναρτήσεις, εμβέλεια και αναδρομή, «Μετατροπές μεταξύ δεκαδικών και δυαδικών αριθμών» (K04, σελ. 70–71)
  - Κεφάλαιο 5: Δείκτες και πίνακες, «Περί ανάγνωσης ακεραίων (και όχι μόνο)» (K04, σελ. 78–79) και «Ιστόγραμμα συχνοτήτων γραμμάτων στην είσοδο» (K04, σελ. 86–87)
  - Για αναφορά: Κεφάλαιο 9: Είσοδος και έξοδος, «Είσοδος και έξοδος» (K04, σελ. 136–149), με τις `getchar`, `putchar`, `printf`, `scanf` και την ανακατεύθυνση.
- **Εργαστήριο**: Εργαστήριο 4: ασκήσεις `pyramid.c` (`putchar` και `scanf`), `lowercase.c` (βρόχος `getchar`/`putchar`), `encode.c` και `decode.c` (φίλτρα με ανακατεύθυνση και σωληνώσεις)
- **Άλλα**:
  - Αναφορά συναρτήσεων: getchar, putchar, scanf, printf, και οι σελίδες `man getchar`, `man scanf`
  - Wikipedia: Data buffer, End-of-Transmission character, Camel case, Snake case
  - POSIX: Canonical mode input processing

Συχνά λάθη

- `==` **με αριθμούς κινητής υποδιαστολής**. Το `float a = 3.1; if (a == 3.1)` τυπώνει `Not OK`. Συγκρίνετε με ανοχή: `fabs(a - 3.1) < 1e-6`.
- `char` **για την τιμή της** `getchar`. Με `char c` η `cat` σταματά στο byte `0xff` (τυπώνει μόνο `hello`) ή δεν τελειώνει ποτέ. Δηλώστε `int c`.
- **Ανάθεση χωρίς παρενθέσεις**. Το `while (ch = getchar() != EOF)` βάζει στο `ch` 1 ή 0, όχι τον χαρακτήρα. Γράψτε `while ((ch = getchar()) != EOF)`.
- **Ξεχνάτε το** `'\n'`. Ο μετρητής χαρακτήρων βγάζει 24 για φράση 23 χαρακτήρων: το Enter

είναι κι αυτό χαρακτήρας.
- **«Κόλλησε» το πρόγραμμα**. Περιμένει είσοδο. Γράψτε κάτι και Enter, ή Ctrl+D για τέλος εισόδου (δύο φορές, αν έχετε ήδη γράψει κάτι στη γραμμή).
- `scanf` **χωρίς** `&`. Το `scanf("%d", n)` δίνει `Segmentation fault`. Γράψτε `scanf("%d", &n)` και μεταγλωττίζετε με `-Wall` για να το πιάνει ο `gcc`.
- **Δεν ελέγχετε την τιμή επιστροφής της** `scanf`. Με είσοδο `hello` ή Ctrl+D τυπώνεται κάτι σαν `Square: 1072038564`. Ελέγξτε `if (scanf(...) != 1)`.
- **Μπερδεύετε το** `0` **με το** `EOF`. Ο έλεγχος `scanf(...) == EOF` μόνος του δεν πιάνει την είσοδο `hello`, για την οποία η `scanf` επιστρέφει `0`.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K9.9** Τιμή επιστροφής της scanf με λάθος είσοδο (11% σωστές): Το 42% επέλεξε `42 43`, μπερδεύοντας την τιμή επιστροφής της `scanf` με τις τιμές που αποθηκεύει στις μεταβλητές, και ξεχνώντας ότι σταματά στο `hello`.
- **K9.8** Πόσες τιμές επιστρέφει η getchar (18% σωστές): Το 64% επέλεξε 256, μετρώντας μόνο τις τιμές ενός byte και ξεχνώντας την επιπλέον τιμή `EOF`, που είναι και ο λόγος που η `getchar` επιστρέφει `int`.
- **K9.7** Μετρητής χαρακτήρων μέχρι την αλλαγή γραμμής (31% σωστές): Το 36% επέλεξε ότι θα τυπώνει 0 1 2 3 ..., θεωρώντας ότι η `printf` είναι μέσα στον βρόχο· χωρίς άγκιστρα το σώμα της `while` είναι μόνο η `charcount++;`.

Ερωτήσεις κατανόησης

- **E9.1** Γιατί η `getchar` επιστρέφει `int` και όχι `char`;[78]
- **E9.2** Ποιες είναι οι τέσσερις πηγές δεδομένων εισόδου ενός προγράμματος;[79]
- **E9.3** Γράφετε `abc` και Enter. Ποιους χαρακτήρες διαβάζει η `getchar` από τη γραμμή;[80]
- **E9.4** Γιατί η `getchar` δεν επιστρέφει αμέσως μόλις πατήσετε ένα πλήκτρο;[81]
- **E9.5** Γιατί γράφουμε `&n` στη `scanf` αλλά `n` στην `printf`;[82]
- **E9.6** Τι επιστρέφει το `scanf("%d %d", &a, &b)` με είσοδο `7 x`;[83]
- **E9.7** Τι επιστρέφει η `getinteger(10)` με είσοδο `4a2` και Enter;[84]

[78] Για να χωράει, εκτός από όλους τους δυνατούς χαρακτήρες, και την τιμή `EOF` (`-1`), που δεν πρέπει να συμπίπτει με κανέναν χαρακτήρα.

[79] Ορίσματα στη γραμμή εντολών, πρότυπη είσοδος (`stdin`), αρχεία, και δίκτυο ή άλλες πηγές.

[80] Τέσσερις: `'a'`, `'b'`, `'c'` και `'\n'`.

[81] Οι χαρακτήρες μένουν στον buffer του τερματικού και προωθούνται στο πρόγραμμα όταν πατήσετε Enter (ή Ctrl+D).

[82] Η `scanf` πρέπει να αλλάξει τη μεταβλητή, άρα χρειάζεται τη διεύθυνσή της· η `printf` μόνο διαβάζει την τιμή της.

[83] `1`: διαβάστηκε το `a = 7`, ενώ το `x` δεν είναι αριθμός, οπότε το `b` δεν άλλαξε.

[84] `ERROR`, δηλαδή `-1`, γιατί το `'a'` δεν είναι έγκυρο ψηφίο στη βάση 10.

Kahoot από το αμφιθέατρο (K9.1–K9.9)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K9.1** Σύγκριση float με ==: 94% σωστές απαντήσεις
- **K9.2** Πώς στέλνω EOF: 90% σωστές απαντήσεις
- **K9.3** Φτάνουν αμέσως τα δεδομένα;: 82% σωστές απαντήσεις
- **K9.4** Η τιμή του EOF: 79% σωστές απαντήσεις
- **K9.5** Ο τύπος επιστροφής της getchar: 66% σωστές απαντήσεις
- **K9.6** Διάβασμα ακεραίου με scanf: 53% σωστές απαντήσεις
- **K9.7** Μετρητής χαρακτήρων μέχρι την αλλαγή γραμμής: 31% σωστές απαντήσεις
- **K9.8** Πόσες τιμές επιστρέφει η getchar: 18% σωστές απαντήσεις
- **K9.9** Τιμή επιστροφής της scanf με λάθος είσοδο: 11% σωστές απαντήσεις

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A9.1–A9.10)

- **A9.1** Μετρητής χαρακτήρων με getchar: Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 20 · ★☆☆ · trace · `slides-lec09-charcount`
- **A9.2** Τι κάνει το πρόγραμμα με τη scanf;: Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 30 · ★☆☆ · trace · `slides-lec09-scanf-square`
- **A9.3** scanf με πολλά ορίσματα: Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 36–37 · ★☆☆ · trace · `slides-lec09-scanf-two`
- **A9.4** Άθροισμα δύο αριθμών από την πρότυπη είσοδο: Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 25–26 · ★★☆ · programming · `slides-lec09-addnums`
- **A9.5** Μια cat με char: Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 23 · ★★☆ · debug · `slides-lec09-cat-char`
- **A9.6** OK ή Not OK;: Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 3 · ★★☆ · trace · `slides-lec09-float-equality`
- **A9.7** Η συνάρτηση getinteger: Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 26, 38 · ★★☆ · trace · `slides-lec09-getinteger`
- **A9.8** Μέτρηση γραμμάτων στην είσοδο: Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 39 · ★★☆ · trace · `slides-lec09-letter-frequencies`
- **A9.9** scanf χωρίς &: Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 32 · ★★☆ · short-answer · `slides-lec09-scanf-no-ampersand`
- **A9.10** Είναι σωστό αυτό το πρόγραμμα;: Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 33–35 · ★★☆ · debug · `slides-lec09-scanf-return`

Εργαστήριο (A9.11–A9.14)

- **A9.11** Ένα απλό κομπιουτεράκι: Εργαστήριο 2, Άσκηση 1 · ★☆☆ · programming · `lab-lab02-calc`

- **A9.12** Διάβασμα ακεραίου με scanf: Εργαστήριο 2, Βήμα 3 · ★☆☆ · short-answer · `lab-lab02-readint`
- **A9.13** Τροποποίηση κειμένου: Εργαστήριο 4, Άσκηση 3 · ★☆☆ · programming · `lab-lab04-lowercase`
- **A9.14** Κωδικοποίηση και αποκωδικοποίηση κειμένου: Εργαστήριο 4, Άσκηση 4 · ★★☆ · programming · `lab-lab04-encode-decode`

Εργασίες (A9.15–A9.17)

- **A9.15** Συνεργασία (Prisoner's Dilemma): Εργασία 2 (2023-24), Άσκηση 3 · ★★☆ · programming · `hw-2023-hw2-coop`
- **A9.16** Το Πρόβλημα του Τρόλεϊ (trolley): Εργασία 0 (2024-25), Άσκηση 3 · ★★☆ · programming · `hw-2024-hw0-trolley`
- **A9.17** Επεξεργασία Ήχου (soundwave): Εργασία 1 (2025-26), Άσκηση 1 · ★★★ · programming · `hw-2025-hw1-soundwave`

Θέματα εξετάσεων (A9.18–A9.31)

- **A9.18** Ραβασάκι: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #0 (Valentine's Themed), Θέμα 1 · ★☆☆ · programming · `exam-2023-fall-ex0-q1`
- **A9.19** Δεκαεξαδικοί: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #1 (Coreutils Themed), Θέμα 1 · ★☆☆ · programming · `exam-2023-fall-ex1-q1`
- **A9.20** Γραμμή και Γράμμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #12, Θέμα 1 · ★☆☆ · programming · `exam-2023-fall-ex12-q1`
- **A9.21** Ορεκτικό: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #14, Θέμα 1 · ★☆☆ · programming · `exam-2023-fall-ex14-q1`
- **A9.22** Πλαγιαστά Γράμματα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #15, Θέμα 1 · ★☆☆ · programming · `exam-2023-fall-ex15-q1`
- **A9.23** Μήνυμα από τον Καίσαρα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #6 (Crypto Themed), Θέμα 2 · ★☆☆ · programming · `exam-2023-fall-ex6-q2`
- **A9.24** Ανάβεις Φωτιές: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #7 (Star Wars Themed), Θέμα 1 · ★☆☆ · programming · `exam-2023-fall-ex7-q1`
- **A9.25** ASCII Encoding: Εξέταση Ιουλίου 2024, Θέμα 1 · ★☆☆ · programming · `exam-2024-jul-q1`
- **A9.26** Αφαίρεση Μη Λατινικών Χαρακτήρων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #10, Θέμα 1 · ★★☆ · programming · `exam-2023-fall-ex10-q1`
- **A9.27** Αναζητώντας τον Blinky: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #11, Θέμα 1 · ★★☆ · programming · `exam-2023-fall-ex11-q1`
- **A9.28** Pikakίστικα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #2 (Pokémon Themed), Θέμα 1 · ★★☆ · programming · `exam-2023-fall-ex2-q1`
- **A9.29** Προσθήκη Νιφάδων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #3 (Frozen Themed), Θέμα 1 · ★★☆ · programming · `exam-2023-fall-ex3-q1`

- **A9.30** Αποκωδικοποίηση: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #8, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex8-q2`
- **A9.31** Μετρητής λέξεων: Εξέταση Σεπτεμβρίου 2024, Θέμα 4 · ★★☆ · programming · `exam-2024-sep-q4`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A3.7** Πυθαγόρειο θεώρημα: Εργαστήριο 2, Άσκηση 2 · ★☆☆ · programming · `lab-lab02-pyth`
- **A6.15** Οι Ακολουθίες Aliquot: Εργασία 0 (2025-26), Άσκηση 3 · ★★☆ · programming · `hw-2025-hw0-aliquot`
- **A6.11** Κατασκευή πυραμίδας: Εργαστήριο 4, Άσκηση 2 · ★☆☆ · programming · `lab-lab04-pyramid`
- **A10.18** Αναποδογύρισμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #5 (HP Themed), Θέμα 1 · ★☆☆ · programming · `exam-2023-fall-ex5-q1`
- **A10.1** Πρόσθεση δύο αριθμών από την είσοδο: Διάλεξη 10, διαφάνεια 5 · ★☆☆ · programming · `slides-lec10-addnums`
- **A10.11** Τι κάνει η getinteger;: Διάλεξη 10, διαφάνεια 17 · ★★☆ · trace · `slides-lec10-getinteger`
- **A12.13** Αλλαγή Τέρματος: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #13, Θέμα 2 · ★☆☆ · programming · `exam-2023-fall-ex13-q2`
- **A12.22** Το μεγαλύτερο άλμα - polevault: Εξέταση Σεπτεμβρίου 2026, Θέμα 3 · ★★☆ · programming · `exam-2026-sep-q3`
- **A13.11** Κάδρο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #9, Θέμα 4 · ★★★ · programming · `exam-2023-fall-ex9-q4`
- **A14.19** Τα Πάνω Κάτω: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #4 (Rick Astley Themed), Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex4-q2`
- **A15.7** Τι κάνει το πρόγραμμα με την getchar: Διάλεξη 15, διαφάνεια 17 · ★☆☆ · trace · `slides-lec15-getchar-count`
- **A18.3** Υποτείνουσα με scanf: Διάλεξη 18, διαφάνειες 29–30 · ★☆☆ · trace · `slides-lec18-hypotenuse`
- **A25.8** Επενδύσεις στο Χρηματιστήριο: Εξέταση Ιουνίου 2026, Θέμα 3 · ★★☆ · programming · `exam-2026-jun-q3`

Μέρος Γ: Πίνακες, δείκτες και μνήμη

Κεφάλαιο 10: Πίνακες

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να διαβάζετε αριθμούς από την είσοδο με την `scanf` και να ελέγχετε την τιμή που επιστρέφει· να δηλώνετε, να αρχικοποιείτε και να χρησιμοποιείτε μονοδιάστατους πίνακες· να υπολογίζετε πόση μνήμη πιάνει ένας πίνακας και σε ποια διεύθυνση βρίσκεται κάθε στοιχείο του· να γράφετε συναρτήσεις που παίρνουν πίνακα (μέγιστο, μέσος όρος, αναζήτηση, `atoi`)· και να εξηγείτε γιατί η πρόσβαση εκτός ορίων είναι επικίνδυνη.

Προαπαιτούμενα: Κεφάλαιο 3, Κεφάλαιο 9

Χρόνος μελέτης: ~2,5 ώρες

Σύνοψη

Η διάλεξη έχει δύο μέρη. Στο πρώτο γνωρίζουμε τη `scanf`, τη «συμμετρική» της `printf` για διάβασμα: αντί να φτιάχνουμε με το χέρι συναρτήσεις όπως η `getinteger` πάνω από την `getchar`, ζητάμε από τη βιβλιοθήκη να μετατρέψει την είσοδο σε αριθμούς, αρκεί να της δώσουμε τη *διεύθυνση* της μεταβλητής και να ελέγξουμε τι μας επέστρεψε. Στο δεύτερο μέρος εισάγουμε τους πίνακες, τη βασικότερη δομή δεδομένων σχεδόν κάθε γλώσσας: ένα όνομα για πολλές τιμές ίδιου τύπου, αποθηκευμένες σε συνεχόμενες θέσεις μνήμης και προσβάσιμες μέσω μιας θέσης (index). Με το παράδειγμα των 100 αρκουδακιών βλέπουμε γιατί χωρίς πίνακες ο κώδικας δεν κλιμακώνεται, και έπειτα γράφουμε τις πρώτες συναρτήσεις πάνω σε πίνακες. Οι πίνακες είναι η βάση για τις συμβολοσειρές, τους δείκτες και σχεδόν όλους τους αλγορίθμους του υπόλοιπου μαθήματος.

Θεωρία

§10.1 Η συνάρτηση `scanf`

Στο Κεφάλαιο 9 διαβάζαμε αριθμούς χαρακτήρα-χαρακτήρα με την `getchar` και μια δική μας συνάρτηση `getinteger` (βλ. «Παραδείγματα»). Η πρότυπη βιβλιοθήκη όμως μας δίνει έτοιμη λύση: τη `scanf`. Ορίζεται στο header file `stdio.h` και διαβάζει δεδομένα εισόδου πολλών τύπων από το `stdin` του προγράμματος, αποθηκεύοντας τις τιμές τους σε μεταβλητές.

Η δήλωσή της μοιάζει πολύ με της `printf`:

```
int scanf(const char *restrict format, ...);
int printf(const char *restrict format, ...);
```

- Το πρώτο όρισμα είναι μια **συμβολοσειρά μορφοποίησης (format string)**, όπως `"%d"` για ακέραιο ή `"%d %d"` για δύο ακεραίους.
- Τα `...` σημαίνουν ότι δέχεται όσα ορίσματα της περάσουμε (πώς γίνεται αυτό θα το δούμε σε άλλο μάθημα).

Η `printf` μετατρέπει τιμές σε κείμενο, η `scanf` κείμενο σε τιμές. Για όλες τις λεπτομέρειες τρέχουμε σε ένα τερματικό `man scanf`.

§10.2 Γιατί η `scanf` θέλει `&`

Στο `scanf("%d", &n);` δεν περνάμε την τιμή της `n` αλλά τη **διεύθυνσή (address)** της στη μνήμη, με τον τελεστή `&`. Ο λόγος: τα ορίσματα στην C περνούν με τιμή, άρα η `scanf` θα έπαιρνε ένα αντίγραφο του (αρχικοποίητου) περιεχομένου της `n` και δεν θα είχε πώς να αλλάξει την ίδια τη μεταβλητή. Με τη διεύθυνση ξέρει *πού* να γράψει την τιμή που διάβασε. (Οι διευθύνσεις και οι δείκτες είναι το θέμα του Κεφαλαίου 11· εδώ αρκεί ο κανόνας.)

Αν ξεχάσουμε το `&` και γράψουμε `scanf("%d", n);`, η `scanf` θα ερμηνεύσει τα σκουπίδια της `n` ως διεύθυνση και θα προσπαθήσει να γράψει εκεί. Το αποτέλεσμα είναι συνήθως:

```
$ ./scanf
Gimme a number: 3
Segmentation fault
```

Ο μεταγλωττιστής σάς προειδοποιεί αν μεταγλωττίζετε με `-Wall` (κάτι σαν «format '%d' expects argument of type 'int *', but argument 2 has type 'int'»).

§10.3 Η τιμή επιστροφής της `scanf`

Η `scanf` επιστρέφει έναν `int`:

- αν επιτύχει, **πόσα δεδομένα εισόδου διάβασε** (και αποθήκευσε)·
- αν η είσοδος τελειώσει πριν διαβάσει οτιδήποτε, την τιμή `EOF` (End-Of-File, δηλαδή −1)·
- αν η είσοδος δεν ταιριάζει με τη μορφοποίηση (π.χ. γράμματα αντί για αριθμό), λιγότερα από όσα ζητήσαμε, ίσως και **0**.

Ένα πρόγραμμα που αγνοεί την τιμή επιστροφής *δεν είναι σωστό*: όταν η `scanf` αποτύχει, η μεταβλητή μένει με ό,τι είχε πριν (εδώ σκουπίδια) και το πρόγραμμα συνεχίζει σαν να μην έγινε τίποτα. Ο σωστός τρόπος είναι να ελέγχουμε ότι επιστράφηκε ακριβώς ο αριθμός τιμών που ζητήσαμε:

```c
if (scanf("%d", &n) != 1) {
  printf("Invalid input\n");
  return 1;
}
```

§10.4 Πολλά ορίσματα και κενοί χαρακτήρες

Με μία κλήση μπορούμε να διαβάσουμε πολλές τιμές: `scanf("%d %d", &n1, &n2);`. Όταν η `scanf` ψάχνει για δεκαδικό ψηφίο (`%d`), **αγνοεί τους κενούς χαρακτήρες (whitespace)**, δηλαδή κενά, tabs και αλλαγές γραμμής, που προηγούνται του αριθμού. Γι' αυτό η είσοδος `2 4` διαβάζεται σωστά ως 2 και 4, όπως και το ίδιο ζευγάρι γραμμένο σε δύο γραμμές. Με δύο τιμές, η σωστή συνθήκη ελέγχου είναι `scanf(...) == 2`.

§10.5 Γιατί χρειαζόμαστε πίνακες: ο γρίφος με τα αρκουδάκια

Έστω ότι έχουμε 100 αρκουδάκια και ψάχνουμε το μεγαλύτερο. Τι κάνουμε; Τα κοιτάμε ένα-ένα, κρατώντας στο μυαλό μας το μεγαλύτερο που έχουμε δει μέχρι τώρα. Ο στόχος είναι η ελαχιστοποίηση του κόπου (ο καλός προγραμματιστής είναι «efficient», aka τεμπέλης): δεν γίνεται να βρούμε το μέγιστο χωρίς να κοιτάξουμε το καθένα, αλλά αρκεί να το κοιτάξουμε μία φορά.

Πώς το κωδικοποιούμε σε C, αν κάθε αρκουδάκι είναι ένας `int`; Με όσα ξέρουμε ως τώρα, χρειαζόμαστε 100 μεταβλητές:

```
int bear0, bear1, bear2, /* ..., */ bear99, max;
bear0 = 42; bear1 = 4; bear2 = 2; /* ... αρχικοποίηση μεταβλητών */
// find the max here: 99 σχεδόν ίδιες εντολές if
```

Ο κώδικας είναι πολύ επαναληπτικός και δεν γράφεται με βρόχο, γιατί δεν υπάρχει τρόπος να πούμε «η μεταβλητή `bear` με αριθμό `i`». Αυτό λύνουν οι πίνακες.

§10.6 Δήλωση πίνακα

Ένας **πίνακας (array)** μας επιτρέπει να χειριζόμαστε ένα σύνολο από δεδομένα ίδιου τύπου με ενιαίο και γενικό τρόπο. Στην C δηλώνεται με τη μορφή:

`τύπος όνομα[μέγεθος];`

Για παράδειγμα `int bears[100];`. Η δήλωση έχει τρία μέρη:

- ο **τύπος (type)** λέει στον μεταγλωττιστή πόση μνήμη να δεσμεύσει για κάθε στοιχείο του πίνακα·
- το **όνομα (name)** κάνει τον μεταγλωττιστή να διαλέξει τη διεύθυνση της μνήμης όπου θα αποθηκεύσει τον πίνακα·
- το **μέγεθος (size)** λέει στον μεταγλωττιστή πόσες θέσεις αυτού του τύπου να κρατήσει. Το μέγεθος είναι **στατικό**: αφού δηλωθεί, δεν αλλάζει κατά την εκτέλεση.

Πίνακες ορίζονται για όλους τους τύπους της C: `int a[1024];`, `char b[2048];`, `double c[512];`.

§10.7 Ο πίνακας στη μνήμη

Στο κάθε στοιχείο αναφερόμαστε με τη **θέση (index)** του στον πίνακα: για τον `int bears[100];` τα στοιχεία είναι τα `bears[0]`, `bears[1]`, ..., `bears[99]`. Τα στοιχεία αποθηκεύονται σε **συνεχόμενες θέσεις μνήμης**. Αν `sizeof(int) == 4` και ο μεταγλωττιστής τοποθετήσει τον πίνακα, ας πούμε, στη διεύθυνση 4, η μνήμη είναι:

Bytes	Περιεχόμενο
0–3	(κάτι άλλο)
4–7	`bears[0]`
8–11	`bears[1]`
12–15	`bears[2]`
...	...
400–403	`bears[99]`

Ο πίνακας `bears` καταλαμβάνει $4 \cdot 100 = 400$ bytes, στις διευθύνσεις 4–403. Γενικά ένας πίνακας `N` στοιχείων τύπου `T` πιάνει `N * sizeof(T)` bytes.

Επειδή τα στοιχεία είναι συνεχόμενα και ίδιου μεγέθους, η διεύθυνση κάθε στοιχείου υπολογίζεται αμέσως:

$$\text{διεύθυνση}(a[i]) = \text{αρχή του πίνακα} + i \cdot \text{sizeof}(\text{τύπος})$$

Για το `bears[2]`: $4 + 2 \cdot 4 = 12$, δηλαδή το στοιχείο ξεκινά από το 12ο byte. Αυτός ο απλός υπολογισμός είναι ο λόγος που η πρόσβαση σε οποιοδήποτε στοιχείο πίνακα κοστίζει το ίδιο, όσο μεγάλος κι αν είναι ο πίνακας, και ο λόγος που οι θέσεις ξεκινούν από το 0: η θέση είναι η *απόσταση* από την αρχή.

§10.8 Χρήση στοιχείων πίνακα

Ένας πίνακας `N` στοιχείων έχει στοιχεία με θέσεις από το **0 μέχρι το N–1**. Κάθε στοιχείο μπορεί να χρησιμοποιηθεί όπως μια μεταβλητή του ίδιου τύπου: σε εκφράσεις, αναθέσεις, τελεστές και συνθήκες. Η θέση μπορεί να είναι οποιαδήποτε ακέραια έκφραση, ακόμα και άλλο στοιχείο του πίνακα:

```
bears[4] = 42;
bears[8] = bears[2] + 42;
bears[bears[4]] = 2;        /* bears[42] = 2 */
```

Η μεγάλη δύναμη είναι ότι η θέση μπορεί να είναι μεταβλητή, οπότε ένας βρόχος `for (i = 0; i < 100; i++)` περνά από όλα τα στοιχεία με μία εντολή.

§10.9 Αρχικοποίηση πίνακα

Η σύνταξη είναι παρόμοια με την αρχικοποίηση μεταβλητής, με τις τιμές σε άγκιστρα, χωρισμένες με κόμμα:

```
int bears[100] = {
  11, 25, 26, 31, 14, 13, 19, 3, 2, 19, /* ... άλλες 90 τιμές */
};
```

Η πρώτη τιμή πάει στο `bears[0]`, η δεύτερη στο `bears[1]`, κ.ο.κ. Αν δεν αρχικοποιήσουμε έναν τοπικό πίνακα, τα στοιχεία του έχουν σκουπίδια, όπως κάθε τοπική μεταβλητή· γι' αυτό συχνά τον μηδενίζουμε με βρόχο πριν τον χρησιμοποιήσουμε.

§10.10 Πίνακας χαρακτήρων (string)

Ένας πίνακας από χαρακτήρες λέγεται και **αλφαριθμητικό** ή **συμβολοσειρά (string)**. Επειδή χρησιμοποιούνται πολύ συχνά, η C έχει αρκετές συντομεύσεις για αυτούς (θα τις δούμε στο Κεφάλαιο 14). Οι τρεις παρακάτω δηλώσεις είναι ισοδύναμες:

```
char hello[] = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd',
                '\n', '\0'};
char hello[] = {72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 10, 0};
char hello[] = "Hello World\n";
```

Η δεύτερη γράφει τους ίδιους χαρακτήρες με τους κωδικούς τους ASCII. Η τρίτη προσθέτει μόνη της στο τέλος τον **null byte** `'\0'`: τα string στην C **τερματίζονται πάντα με το null byte**, και έτσι μια συνάρτηση ξέρει πού τελειώνει η συμβολοσειρά χωρίς να της πούμε το μήκος. Όταν παραλείπουμε το μέγεθος (`[]`), ο μεταγλωττιστής το υπολογίζει από την αρχικοποίηση· εδώ 13 (12 χαρακτήρες + `'\0'`).

0	1	2	3	4	5	6	7	8	9	10	11	12
`'H'`	`'e'`	`'l'`	`'l'`	`'o'`	`' '`	`'W'`	`'o'`	`'r'`	`'l'`	`'d'`	`'\n'`	`'\0'`

§10.11 Οι πίνακες δεν ανατίθενται

Μπορούμε να αναθέσουμε τα στοιχεία ενός πίνακα σε άλλον με μία εντολή;

```
int a[3] = {1, 2, 3};
int b[3] = {4, 5, 6};
b = a;   // does-not-compile
```

Όχι, δεν επιτρέπεται στην C. Ο gcc απαντά με `error: assignment to expression with array type`. Για να αντιγράψουμε έναν πίνακα αντιγράφουμε τα στοιχεία ένα-ένα με βρόχο: `for (i =`

`0; i < 3; i++) b[i] = a[i];`. (Το γιατί θα φανεί όταν δούμε τη σχέση πινάκων και δεικτών στο Κεφάλαιο 12.)

§10.12 Πίνακες ως ορίσματα συναρτήσεων

Μια συνάρτηση μπορεί να πάρει πίνακα ως παράμετρο, π.χ. `int find_max(int bears[100])` ή `int atoi(char digits[])`. Στη δεύτερη μορφή δεν γράφουμε μέγεθος: η συνάρτηση πρέπει να ξέρει με άλλο τρόπο πού τελειώνει ο πίνακας, είτε από σταθερό μέγεθος, είτε από ξεχωριστή παράμετρο `n`, είτε, για string, από το `'\0'`. Οι σημειώσεις εξηγούν ότι στην πραγματικότητα η συνάρτηση παίρνει τη διεύθυνση του πρώτου στοιχείου, άρα δεν γίνεται αντίγραφο του πίνακα και οι αλλαγές στα στοιχεία του φαίνονται στον καλούντα· θα το δούμε αναλυτικά στα Κεφάλαια 11 και 12.

§10.13 Πού είναι χρήσιμοι οι πίνακες

Ο πίνακας είναι η βασικότερη **δομή δεδομένων (data structure)** στην πλειοψηφία των γλωσσών προγραμματισμού: μοντελοποιεί ένα σύνολο τιμών ίδιου τύπου, δεσμεύει μνήμη μαζικά με μία δήλωση και δίνει άμεση αποθήκευση, προσπέλαση και μετατροπή δεδομένων μέσω της θέσης.

§10.14 Πρόσβαση εκτός ορίων

Τι θα συμβεί αν προσπελάσουμε θέση έξω από τον πίνακα;

- **Υπερχείλιση (overflow)**: θέση μετά το τέλος, π.χ. `bears[100]`.
- **Υποχείλιση (underflow)**: θέση πριν την αρχή, π.χ. `bears[-1]`.

Η C δεν ελέγχει τα όρια: ο μεταγλωττιστής απλώς υπολογίζει «αρχή + θέση · μέγεθος» και διαβάζει ή γράφει εκεί, σε μνήμη που ανήκει σε κάτι άλλο. Το standard της γλώσσας κατατάσσει αυτή τη χρήση ως **απροσδιόριστη συμπεριφορά (undefined behavior)**: δεν υπάρχει καμία εγγύηση για το τι θα γίνει. Στην πράξη, το πρόγραμμα θα κρασάρει (`Segmentation fault`), θα αλλοιώσει σιωπηλά άλλες μεταβλητές ή, ακόμα χειρότερα, κάποιος θα το εκμεταλλευτεί για να μας «χακάρει». Είναι ευθύνη του προγραμματιστή να εξασφαλίσει ότι κάθε θέση `i` ικανοποιεί $0 \le i \le N - 1$.

Παραδείγματα

§10.15 Η `getinteger` με `getchar`

Η διάλεξη ξεκινά με το πρόβλημα «θέλω να διαβάσω δύο αριθμούς από την πρότυπη είσοδο και να τους προσθέσω»:

```
$ ./addnums
Give me a number: 40
Give me another number: 2
```

```
Total: 42
```

Με όσα ξέραμε, η λύση ήταν να διαβάζουμε τους χαρακτήρες έναν-έναν με την `getchar` και να τους μετατρέπουμε σε αριθμό (μέθοδος του Horner, βλ. Κεφάλαιο 9):

```c
#define ERROR -1        // Return value for illegal character

int getinteger(int base) {
  int ch;               // No need to declare ch as int - no EOF handling
  int val = 0;          // Initialize return value
  while ((ch = getchar()) != '\n')              // Read up to new line
    if (ch >= '0' && ch <= '0' + base - 1)      // Legal character?
      val = base * val + (ch - '0');            // Update return value
    else
      return ERROR;     // Illegal character read
  return val;           // Everything OK - Return value of number read
}
```

Η συνάρτηση διαβάζει μέχρι την αλλαγή γραμμής έναν ακέραιο γραμμένο στη βάση `base` (έως 10): κάθε νόμιμο ψηφίο «σπρώχνει» την τιμή μία θέση αριστερά (`base * val`) και προσθέτει το ψηφίο (`ch - '0'`). Με το πρώτο μη νόμιμο ψηφίο επιστρέφει `ERROR`. Προσέξτε ότι το −1 είναι ταυτόχρονα και αποτέλεσμα λάθους, οπότε δεν ξεχωρίζει από μια έγκυρη τιμή· και ότι δεν χειρίζεται το `EOF`. «Υπάρχει άλλος τρόπος;» Ναι: η `scanf`.

§10.16 Το τετράγωνο ενός αριθμού με `scanf`

Εφαρμόζει τις ενότητες «Η συνάρτηση `scanf`» και «Γιατί η `scanf` θέλει `&`».

```c
#include <stdio.h>

int main() {
  int n;
  printf("Gimme a number: ");
  scanf("%d", &n);
  printf("Square: %d\n", n * n);
  return 0;
}
```

Τυπώνει στο `stdout` το τετράγωνο του αριθμού που γράψαμε στο `stdin`. Είναι σωστό; Όχι, γιατί δεν ελέγχει την τιμή επιστροφής της `scanf`:

```
$ ./scanf
Gimme a number: 16
Square: 256
$ ./scanf
Gimme a number: Square:
```

```
1068701481
$ ./scanf
Gimme a number: hello
Square: 1072038564
```

Στη δεύτερη εκτέλεση η είσοδος έκλεισε (Ctrl-D) χωρίς αριθμό, οπότε η `scanf` επέστρεψε `EOF`· στην τρίτη το `hello` δεν είναι ακέραιος, οπότε επέστρεψε 0. Και στις δύο η `n` έμεινε αρχικοποίητη και τυπώθηκε το τετράγωνο των σκουπιδιών της.

§10.17 Δύο αριθμοί με μία `scanf`

Εφαρμόζει την ενότητα «Πολλά ορίσματα και κενοί χαρακτήρες».

```c
#include <stdio.h>

int main() {
  int n1, n2;
  printf("Gimme two numbers: ");
  scanf("%d %d", &n1, &n2);
  printf("Result: %d\n", n1 * n2);
  return 0;
}
```

```
$ ./scanf2
Gimme two numbers:   2   4
Result: 8
```

§10.18 Συχνότητες γραμμάτων με πίνακα

Πριν από τον ορισμό των πινάκων, η διάλεξη ρωτά «τι κάνει το παρακάτω πρόγραμμα;» για ένα απόσπασμα από το `histogram.c` των σημειώσεων. Εδώ είναι μέσα σε `main`, με έναν βρόχο εκτύπωσης στο τέλος που *δεν* υπάρχει στη διαφάνεια:

```c
#include <stdio.h>

int main() {
  int i, ch, total = 0;
  int letfr[26];   // Letter occurrences and frequencies array
  for (i = 0; i < 26; i++)
    letfr[i] = 0;
  while ((ch = getchar()) != EOF) {
    if (ch >= 'A' && ch <= 'Z') {
      letfr[ch - 'A']++;           // Found upper case letter
      total++;
    }
```

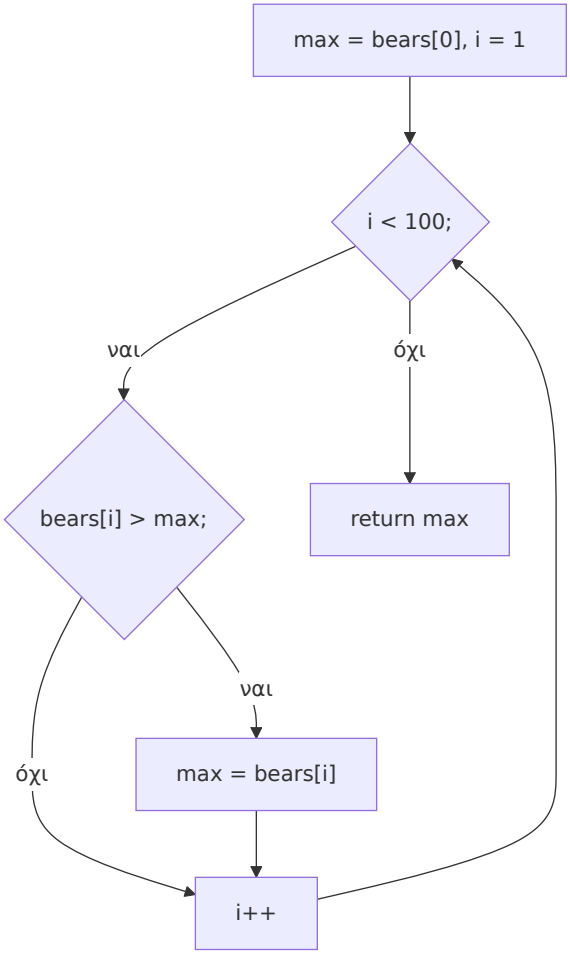
```
    if (ch >= 'a' && ch <= 'z') {
      letfr[ch - 'a']++;           // Found lower case letter
      total++;
    }
  }
  for (i = 0; i < 26; i++)         // not on the slide
    printf("%c: %d\n", 'a' + i, letfr[i]);
  printf("total: %d\n", total);
  return 0;
}
```

Ο `letfr` έχει έναν μετρητή ανά γράμμα. Η έκφραση `ch - 'A'` μετατρέπει ένα γράμμα στη θέση του (`'A'` ⟶ 0, ..., `'Z'` ⟶ 25), οπότε το `letfr[ch - 'A']++` αυξάνει τον σωστό μετρητή χωρίς 26 εντολές `if`· κεφαλαία και πεζά μετρούν μαζί. Εφαρμόζει τη «Χρήση στοιχείων πίνακα» και τον μηδενισμό με βρόχο. Το πλήρες πρόγραμμα των σημειώσεων τυπώνει και ιστόγραμμα.

§10.19 Εύρεση μέγιστου στοιχείου: `find_max`

Η λύση του γρίφου με τα αρκουδάκια: μια συνάρτηση που παίρνει πίνακα 100 στοιχείων και επιστρέφει το μέγιστο.

```
int find_max(int bears[100]) {
  int i, max = bears[0];
  for (i = 1; i < 100; i++) {
    if (bears[i] > max) max = bears[i];
  }
  return max;
}
```



Σχήμα: ο αλγόριθμος της `find_max`*: μία σάρωση, 99 συγκρίσεις.*

Ξεκινάμε με το πρώτο στοιχείο ως «μέγιστο μέχρι τώρα» (όχι με 0, που θα ήταν λάθος αν όλα τα στοιχεία ήταν αρνητικά) και κοιτάμε καθένα από τα υπόλοιπα μία φορά. Εφαρμόζει τις ενότητες «Χρήση στοιχείων πίνακα» και «Πίνακες ως ορίσματα συναρτήσεων».

§10.20 Μέσος όρος

«Θέλω μια συνάρτηση που να δέχεται έναν πίνακα 100 ακεραίων και να επιστρέφει τον μέσο όρο.»

```
int average(int grades[100]) {
  int i, sum = 0;
  for (i = 0; i < 100; i++) {
    sum += grades[i];
  }
  return sum / 100;
```

```
}
```

Το μοτίβο είναι ο συσσωρευτής: μηδενίζουμε το `sum` και προσθέτουμε κάθε στοιχείο. Προσέξτε ότι `sum / 100` είναι ακέραια διαίρεση, άρα ο μέσος όρος στρογγυλεύεται προς τα κάτω (για θετικά)· για ακριβές αποτέλεσμα θα επιστρέφαμε `double` και θα γράφαμε `sum / 100.0`.

§10.21 Αναζήτηση στοιχείου: `find`

«Θέλω μια συνάρτηση που να δέχεται έναν πίνακα 100 ακεραίων και έναν ακέραιο και να γυρνάει τη θέση του στοιχείου αν το βρήκε ή −1.»

```
int find(int haystack[100], int needle) {
  int i;
  for (i = 0; i < 100; i++) {
    if (haystack[i] == needle) {
      return i;
    }
  }
  return -1;
}
```

Αυτή είναι η **σειριακή αναζήτηση (linear search)**: ψάχνουμε τη «βελόνα» στα «άχυρα». Επιστρέφουμε αμέσως μόλις τη βρούμε, και μόνο αν ο βρόχος τελειώσει χωρίς επιτυχία επιστρέφουμε −1. Το −1 λειτουργεί ως ένδειξη αποτυχίας γιατί δεν είναι ποτέ έγκυρη θέση πίνακα. Πιο γρήγορους τρόπους αναζήτησης θα δούμε στο Κεφάλαιο 17.

§10.22 Η `atoi`: από string σε ακέραιο

«Θέλω μια συνάρτηση `atoi` που να παίρνει έναν πίνακα χαρακτήρων (μόνο ψηφία) και να επιστρέφει έναν ακέραιο.»

```
int atoi(char digits[]) {
  int result = 0;
  for (int i = 0; digits[i]; i++) {
    result = 10 * result + digits[i] - '0';
  }
  return result;
}
```

Είναι η ίδια ιδέα με την `getinteger` σε βάση 10, αλλά τώρα οι χαρακτήρες έρχονται από πίνακα αντί από την είσοδο. Η συνθήκη `digits[i]` σημαίνει `digits[i] != '\0'`: ο βρόχος σταματά στο null byte που τερματίζει το string. Για `"472"`: $0 \to 4 \to 47 \to 472$.

«Τι μπορεί να πάει στραβά με αυτήν τη συνάρτηση;» Η διάλεξη αφήνει την ερώτηση ανοιχτή. Μερικές απαντήσεις: δεν ελέγχει ότι κάθε χαρακτήρας είναι ψηφίο (για `"4a"` επιστρέφει σκουπίδια χωρίς ένδειξη λάθους)· δεν χειρίζεται πρόσημο· ένα πολύ μακρύ string υπερχειλίζει

τον `int`· και αν ο πίνακας δεν τελειώνει σε `'\0'`, ο βρόχος συνεχίζει εκτός ορίων. Επιπλέον, το όνομα `atoi` υπάρχει ήδη στο `stdlib.h`, οπότε σε πραγματικό πρόγραμμα θα διαλέγαμε άλλο όνομα.

§10.23 Συμβουλές για το εργαστήριο

Στο Εργαστήριο 6 η άσκηση `judgement.c` ζητά `max_array`, `min_array` και `sum_array` πάνω σε πίνακα 10 βαθμολογιών που διαβάζονται με `scanf`: είναι οι `find_max` και `average` αυτού του κεφαλαίου, με το μέγεθος ως παράμετρο `n` αντί για σταθερά 100. Η `sieve.c` (κόσκινο του Ερατοσθένη) χρησιμοποιεί πίνακα από 0 και 1 όπου η θέση `i` λέει αν το `i` είναι πρώτος, όπως ο `letfr` χρησιμοποιεί τη θέση για γράμμα.

Κύρια σημεία

1. Η `scanf` (από το `stdio.h`) είναι η συμμετρική της `printf`: διαβάζει από το `stdin` τιμές σύμφωνα με μια συμβολοσειρά μορφοποίησης και τις αποθηκεύει σε μεταβλητές.
2. Στη `scanf` περνάμε τη διεύθυνση της μεταβλητής (`&n`), αλλιώς δεν μπορεί να γράψει την τιμή· χωρίς `&` το πρόγραμμα συνήθως κρασάρει με `Segmentation fault`.
3. Η `scanf` επιστρέφει πόσες τιμές διάβασε, ή `EOF` αν τελείωσε η είσοδος· ένα πρόγραμμα που δεν ελέγχει αυτήν την τιμή δεν είναι σωστό.
4. Όταν ψάχνει αριθμό, η `scanf` αγνοεί κενά, tabs και αλλαγές γραμμής.
5. Ένας πίνακας δηλώνεται ως `τύπος όνομα[μέγεθος];` και κρατά `μέγεθος` τιμές του ίδιου τύπου· το μέγεθος είναι στατικό.
6. Τα στοιχεία ενός πίνακα `N` θέσεων είναι τα `a[0]` έως `a[N-1]` και καθένα χρησιμοποιείται όπως μια απλή μεταβλητή· η θέση μπορεί να είναι οποιαδήποτε ακέραια έκφραση.
7. Τα στοιχεία αποθηκεύονται σε συνεχόμενες θέσεις μνήμης, οπότε ο πίνακας πιάνει `N * sizeof(τύπος)` bytes και το `a[i]` βρίσκεται στη διεύθυνση αρχή + `i * sizeof(τύπος)`.
8. Ένας πίνακας αρχικοποιείται με λίστα τιμών σε άγκιστρα· χωρίς αρχικοποίηση, ένας τοπικός πίνακας περιέχει σκουπίδια.
9. Ένα string είναι πίνακας χαρακτήρων που τερματίζεται πάντα με το null byte `'\0'`· το `"Hello World\n"` είναι συντομογραφία για τη λίστα των χαρακτήρων του μαζί με το `'\0'`.
10. Η ανάθεση ενός πίνακα σε άλλον (`b = a;`) δεν επιτρέπεται στην C· αντιγράφουμε στοιχείο προς στοιχείο.
11. Ο πίνακας είναι η βασικότερη δομή δεδομένων: μοντελοποιεί σύνολα τιμών, δεσμεύει μνήμη μαζικά με μία δήλωση και δίνει άμεση πρόσβαση σε κάθε στοιχείο.
12. Η πρόσβαση εκτός ορίων (`bears[100]`, `bears[-1]`) είναι απροσδιόριστη συμπεριφορά: μπορεί να κρασάρει το πρόγραμμα ή να το κάνει ευάλωτο σε επιθέσεις.
13. Εύρεση μέγιστου, άθροισμα/μέσος όρος και σειριακή αναζήτηση είναι τα βασικά μοτίβα βρόχου πάνω σε πίνακα.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
πίνακας	array	Σύνολο στοιχείων ίδιου τύπου σε συνεχόμενες θέσεις μνήμης, με ένα όνομα.
θέση	index	Ο αριθμός (από 0) που επιλέγει ένα στοιχείο του πίνακα.
μέγεθος	size	Πόσα στοιχεία έχει ο πίνακας· στατικό μετά τη δήλωση.
συμβολοσειρά, αλφαριθμητικό	string	Πίνακας χαρακτήρων που τερματίζεται με `'\0'`.
null byte	null byte / null terminator	Ο χαρακτήρας `'\0'` (τιμή 0) που σημειώνει το τέλος ενός string.
συμβολοσειρά μορφοποίησης	format string	Το πρώτο όρισμα της `printf`/`scanf`, π.χ. `"%d %d"`.
διεύθυνση	address	Η θέση μιας μεταβλητής στη μνήμη· την παίρνουμε με `&`.
τέλος αρχείου	end-of-file (EOF)	Η τιμή −1 που επιστρέφουν `getchar`/`scanf` όταν τελειώσει η είσοδος.
κενοί χαρακτήρες	whitespace	Κενά, tabs και αλλαγές γραμμής.
δομή δεδομένων	data structure	Τρόπος οργάνωσης δεδομένων στη μνήμη για αποδοτική χρήση.
υπερχείλιση / υποχείλιση	overflow / underflow	Πρόσβαση μετά το τέλος / πριν την αρχή ενός πίνακα.
απροσδιόριστη συμπεριφορά	undefined behavior	Χρήση για την οποία το standard δεν εγγυάται κανένα αποτέλεσμα.
σειριακή αναζήτηση	linear search	Έλεγχος των στοιχείων ένα-ένα μέχρι να βρεθεί το ζητούμενο.

Διάβασμα

- **Διαφάνειες**: Διάλεξη 10, σελ. 1–52. `scanf` σελ. 5–16· `getinteger` και συχνότητες γραμμάτων σελ. 17–18· γρίφος με τα αρκουδάκια σελ. 20–27· δήλωση, μνήμη, χρήση και αρχικοποίηση πίνακα σελ. 28–36· `find_max` σελ. 37–38· διευθύνσεις και τύποι σελ. 39–40· strings σελ. 41· ανάθεση, χρησιμότητα, όρια σελ. 42–44· `average`, `find`, `atoi` σελ. 45–50.
- **Σημειώσεις**: η διάλεξη μαζί με την επόμενη καλύπτει τις σελίδες 73–103 των σημειώσεων του κ. Σταματόπουλου. Για αυτό το κεφάλαιο: Κεφάλαιο 5: Δείκτες και πίνακες, ενότητες

«Περί ανάγνωσης ακεραίων (και όχι μόνο)» (K04, σελ. 78–79), «Πίνακες» (K04, σελ. 80–85) και «Ιστόγραμμα συχνοτήτων γραμμάτων στην είσοδο» (K04, σελ. 86–87). Η ενότητα «Δείκτες» (K04, σελ. 72–77) ανήκει στο Κεφάλαιο 11.

- **Εργαστήριο**: Εργαστήριο 6: ασκήσεις `sieve.c`, `judgement.c`.
- **Άλλα**: Array (data type) και Array (data structure), Array programming, C Strings (w3schools), Data buffer, Memoization· `man scanf`.

Συχνά λάθη

- `scanf` **χωρίς** `&`. Το `scanf("%d", n);` μεταγλωττίζεται (με προειδοποίηση στο `-Wall`) και κρασάρει με `Segmentation fault`. Γράφετε `scanf("%d", &n);` και μεταγλωττίζετε πάντα με `-Wall`.
- **Αγνόηση της τιμής επιστροφής της** `scanf`. Με είσοδο `hello` ή κενή είσοδο το πρόγραμμα τυπώνει σκουπίδια (`Square: 1072038564`). Ελέγχετε `if (scanf("%d", &n) != 1)`.
- **Θέσεις από 1 έως N**. Ο βρόχος `for (i = 1; i <= 100; i++)` χάνει το `bears[0]` και διαβάζει το ανύπαρκτο `bears[100]`. Οι θέσεις είναι 0 έως `N-1`: `for (i = 0; i < 100; i++)`.
- **Πρόσβαση εκτός ορίων**. Γράψιμο στο `a[N]` ή στο `a[-1]` μπορεί να περάσει απαρατήρητο και να αλλοιώσει άλλες μεταβλητές, ή να κρασάρει αργότερα. Ελέγχετε κάθε θέση που προέρχεται από την είσοδο πριν τη χρησιμοποιήσετε.
- **Αρχικοποίηση μέγιστου με 0**. Με `max = 0` η `find_max` σε πίνακα με μόνο αρνητικούς επιστρέφει 0. Ξεκινάτε με `max = a[0]`.
- **Μη μηδενισμένος πίνακας μετρητών**. Αν ξεχάσετε το `letfr[i] = 0`, οι μετρητές ξεκινούν από σκουπίδια και τα αποτελέσματα είναι τυχαία.
- **Ανάθεση πίνακα**. Το `b = a;` δίνει `error: assignment to expression with array type`. Αντιγράφετε με βρόχο.
- **Ακέραια διαίρεση στον μέσο όρο**. Το `sum / 100` για άθροισμα 1050 δίνει 10, όχι 10.5. Χρησιμοποιείτε `double` και `100.0` όταν θέλετε δεκαδικά.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K10.10** Διεύθυνση στοιχείου πίνακα char (19% σωστές): Το 52% επέλεξε `1041`, μετρώντας τις θέσεις σαν να ξεκινούσαν από το 1. Αφού το `array[0]` είναι στο 1000 και κάθε `char` πιάνει 1 byte, το `array[i]` είναι στο 1000 + i.
- **K10.9** Χαρακτήρας σε συμβολοσειρά (27% σωστές): Το 47% επέλεξε «(κενό)», μετρώντας τις θέσεις από το 1. Το κενό είναι το `hello[5]`, γιατί η αρίθμηση στους πίνακες ξεκινά από το 0.
- **K10.7** Μέγεθος πίνακα int (49% σωστές): Το 35% επέλεξε `4096`, θεωρώντας ότι ένας `int`

είναι πάντα 4 bytes. Το πρότυπο της C δεν ορίζει το ακριβές μέγεθος του `int`· εξαρτάται από το σύστημα.

Ερωτήσεις κατανόησης

- **E10.1** Τι επιστρέφει η `scanf("%d %d", &a, &b)` αν η είσοδος είναι `5 x`, και τι αν η είσοδος είναι κενή;[85]
- **E10.2** Γιατί γράφουμε `scanf("%d", &n)` αλλά `printf("%d", n);`[86]
- **E10.3** Ποιες είναι η πρώτη και η τελευταία έγκυρη θέση του `double c[512];` και πόσα bytes πιάνει ο πίνακας;[87]
- **E10.4** Ο πίνακας `int a[50];` ξεκινά από τη διεύθυνση 1000 και `sizeof(int) == 4`. Σε ποια διεύθυνση ξεκινά το `a[10]`;[88]
- **E10.5** Μετά τις εντολές `bears[4] = 42; bears[bears[4]] = 2;` ποιο στοιχείο έχει την τιμή 2;[89]
- **E10.6** Πόσα στοιχεία έχει ο πίνακας `char s[] = "abc";` και ποιο είναι το τελευταίο;[90]
- **E10.7** Γιατί η `find` επιστρέφει -1 όταν δεν βρει το στοιχείο;[91]
- **E10.8** Τι επιστρέφει η `atoi` των διαφανειών για το string `"0042"`;[92]

Kahoot από το αμφιθέατρο (K10.1–K10.10)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K10.1** Πρόσβαση εκτός ορίων: 85% σωστές απαντήσεις
- **K10.2** Αντιγραφή πίνακα με ανάθεση: 84% σωστές απαντήσεις
- **K10.3** Ο δείκτης του πρώτου στοιχείου: 82% σωστές απαντήσεις
- **K10.4** Το τελευταίο στοιχείο πίνακα: 81% σωστές απαντήσεις
- **K10.5** Αρχικοποίηση με περισσότερες τιμές: 71% σωστές απαντήσεις
- **K10.6** Δείκτης μέσα σε δείκτη: 69% σωστές απαντήσεις
- **K10.7** Μέγεθος πίνακα int: 49% σωστές απαντήσεις
- **K10.8** Διεύθυνση στοιχείου πίνακα int: 47% σωστές απαντήσεις
- **K10.9** Χαρακτήρας σε συμβολοσειρά: 27% σωστές απαντήσεις
- **K10.10** Διεύθυνση στοιχείου πίνακα char: 19% σωστές απαντήσεις

[85] Για `5 x` επιστρέφει 1 (διάβασε μόνο το `a`)· για κενή είσοδο επιστρέφει `EOF` (-1).

[86] Η `printf` χρειάζεται μόνο την τιμή, ενώ η `scanf` πρέπει να γράψει στη μεταβλητή, άρα χρειάζεται τη διεύθυνσή της.

[87] `c[0]` και `c[511]`· $512 \cdot 8 = 4096$ bytes (με `sizeof(double) == 8`).

[88] $1000 + 10 \cdot 4 = 1040$.

[89] Το `bears[42]`, γιατί η θέση `bears[4]` αποτιμάται πρώτα σε 42.

[90] 4 στοιχεία: `'a'`, `'b'`, `'c'` και το τελευταίο είναι το `'\0'`.

[91] Γιατί κάθε έγκυρη θέση είναι από 0 έως 99, οπότε το -1 δεν μπερδεύεται με επιτυχημένο αποτέλεσμα.

[92] Το 42: τα αρχικά μηδενικά αφήνουν το `result` στο 0.

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A10.1–A10.14)

- **A10.1** Πρόσθεση δύο αριθμών από την είσοδο: Διάλεξη 10, διαφάνεια 5 · ★☆☆ · programming · `slides-lec10-addnums`
- **A10.2** Ανάθεση πίνακα σε πίνακα: Διάλεξη 10, διαφάνεια 42 · ★☆☆ · short-answer · `slides-lec10-array-assign`
- **A10.3** Ποιος πίνακας πιάνει περισσότερη μνήμη;: Διάλεξη 10, διαφάνεια 40 · ★☆☆ · short-answer · `slides-lec10-array-memory`
- **A10.4** Μέσος όρος πίνακα: Διάλεξη 10, διαφάνειες 45-46 · ★☆☆ · programming · `slides-lec10-average`
- **A10.5** Αναζήτηση στοιχείου σε πίνακα: Διάλεξη 10, διαφάνειες 47-48 · ★☆☆ · programming · `slides-lec10-find`
- **A10.6** Το μεγαλύτερο αρκουδάκι: Διάλεξη 10, διαφάνειες 21-26 και 37 · ★☆☆ · programming · `slides-lec10-find-max`
- **A10.7** Πρόσβαση εκτός ορίων πίνακα: Διάλεξη 10, διαφάνεια 44 · ★☆☆ · short-answer · `slides-lec10-out-of-bounds`
- **A10.8** Τι κάνει το πρόγραμμα με τη scanf;: Διάλεξη 10, διαφάνειες 9-10 · ★☆☆ · trace · `slides-lec10-scanf-square`
- **A10.9** scanf με πολλά ορίσματα: Διάλεξη 10, διαφάνειες 15-16 · ★☆☆ · trace · `slides-lec10-scanf-two-numbers`
- **A10.10** Η συνάρτηση atoi: Διάλεξη 10, διαφάνειες 49-50 · ★★☆ · programming · `slides-lec10-atoi`
- **A10.11** Τι κάνει η getinteger;: Διάλεξη 10, διαφάνεια 17 · ★★☆ · trace · `slides-lec10-getinteger`
- **A10.12** Τι κάνει ο πίνακας letfr;: Διάλεξη 10, διαφάνεια 18 · ★★☆ · trace · `slides-lec10-letter-frequency`
- **A10.13** Είναι σωστό αυτό το πρόγραμμα;: Διάλεξη 10, διαφάνειες 12-14 · ★★☆ · debug · `slides-lec10-scanf-correct`
- **A10.14** scanf χωρίς &: Διάλεξη 10, διαφάνεια 11 · ★★☆ · debug · `slides-lec10-scanf-no-ampersand`

Εργαστήριο (A10.15–A10.17)

- **A10.15** Πίνακες και συναρτήσεις: Εργαστήριο 6, Άσκηση 4 · ★☆☆ · programming · `lab-lab06-judgement`
- **A10.16** Το κόσκινο του Ερατοσθένη: Εργαστήριο 6, Άσκηση 2 · ★☆☆ · programming · `lab-lab06-sieve`
- **A10.17** Εντοπισμός σφάλματος μνήμης με τον gdb: Εργαστήριο 7, Άσκηση 6 · ★☆☆ · debug · `lab-lab07-my_prog`

Θέματα εξετάσεων (A10.18–A10.24)

- **A10.18** Αναποδογύρισμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #5 (HP Themed), Θέμα 1 · ★☆☆ · programming · `exam-2023-fall-ex5-q1`
- **A10.19** Η συνάρτηση cons: Εξέταση Σεπτεμβρίου 2026, Θέμα 2 · ★☆☆ · trace · `exam-2026-sep-q2`
- **A10.20** Τοποθέτηση του Pacman: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #11, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex11-q2`
- **A10.21** Φορτωμένο Έλκηθρο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #3 (Frozen Themed), Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex3-q2`
- **A10.22** Αναγράμματα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #4 (Rick Astley Themed), Θέμα 1 · ★★☆ · programming · `exam-2023-fall-ex4-q1`
- **A10.23** Μαγικό Ζευγάρι: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #5 (HP Themed), Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex5-q2`
- **A10.24** Η συνάρτηση paws: Εξέταση Ιουνίου 2026, Θέμα 2 · ★★☆ · debug · `exam-2026-jun-q2`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A9.20** Γραμμή και Γράμμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #12, Θέμα 1 · ★☆☆ · programming · `exam-2023-fall-ex12-q1`
- **A11.11** Η δική μας atoi: Διάλεξη 11, διαφάνειες 41–42 · ★★☆ · programming · `slides-lec11-atoi`
- **A11.4** Μέσος όρος πίνακα 100 ακεραίων: Διάλεξη 11, διαφάνειες 37–38 · ★☆☆ · programming · `slides-lec11-average`
- **A11.8** Θέση στοιχείου σε πίνακα ή -1: Διάλεξη 11, διαφάνειες 39–40 · ★☆☆ · programming · `slides-lec11-find`
- **A12.20** Στατιστικές: Εξέταση Ιουλίου 2024, Θέμα 2 · ★★☆ · programming · `exam-2024-jul-q2`
- **A12.21** Κινούμενος Μέσος Όρος - sma: Εξέταση Ιανουαρίου 2025, Θέμα 3 · ★★☆ · programming · `exam-2025-jan-q3`
- **A14.15** Προσεγγίζοντας την Λέξη: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #12, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex12-q2`
- **A15.18** Δίδυμοι Πρώτοι: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #7 (Star Wars Themed), Θέμα 3 · ★★★ · programming · `exam-2023-fall-ex7-q3`
- **A15.1** Πολυπλοκότητα της atoi: Διάλεξη 15, διαφάνεια 15 · ★☆☆ · short-answer · `slides-lec15-complexity-atoi`
- **A16.22** Εαυτοί Αριθμοί: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #12, Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex12-q4`
- **A16.1** Μέσος όρος πίνακα και πολυπλοκότητα: Διάλεξη 16, διαφάνειες 8–9 · ★☆☆ · programming · `slides-lec16-average-complexity`
- **A16.2** Αναζήτηση σε πίνακα και πολυπλοκότητα: Διάλεξη 16, διαφάνειες 10 και 15 · ★☆☆ · programming · `slides-lec16-find-complexity`

Κεφάλαιο 11: Δείκτες και Αναδρομή

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να εξηγείτε τι είναι η διεύθυνση μιας μεταβλητής και να τη βρίσκετε με τον τελεστή `&`· να δηλώνετε, να αρχικοποιείτε και να χρησιμοποιείτε δείκτες (`*`)· να ξέρετε πότε ένας δείκτης είναι `NULL` ή μη έγκυρος· να κάνετε αριθμητική δεικτών και να εξηγείτε γιατί το `ptr[n]` είναι το ίδιο με το `*(ptr + n)`· να γράφετε συναρτήσεις που δέχονται πίνακες· και να γράφετε μια απλή αναδρομική συνάρτηση με βάση τερματισμού.

Προαπαιτούμενα: Κεφάλαιο 2, Κεφάλαιο 3, Κεφάλαιο 10

Χρόνος μελέτης: ~3 ώρες

Σύνοψη

Η διάλεξη αυτή εισάγει έναν νέο τύπο, τον **δείκτη (pointer)**: μια μεταβλητή που δεν κρατά έναν αριθμό ή έναν χαρακτήρα, αλλά τη διεύθυνση μιας άλλης μεταβλητής στη μνήμη. Ξεκινάμε από το πώς η μνήμη οργανώνεται σε bytes με διευθύνσεις, βλέπουμε πώς βρίσκουμε τη διεύθυνση μιας μεταβλητής με το `&` και πώς φτάνουμε από τη διεύθυνση πίσω στη μεταβλητή με το `*`. Έπειτα μαθαίνουμε τις πράξεις που επιτρέπονται σε δείκτες και ανακαλύπτουμε ότι η γνωστή μας σύνταξη `a[i]` των πινάκων είναι στην πραγματικότητα αριθμητική δεικτών. Με αυτά γράφουμε συναρτήσεις που δουλεύουν πάνω σε πίνακες (μέσος όρος, αναζήτηση, μια δική μας `atoi`). Η διάλεξη κλείνει με την **αναδρομή (recursion)**, όπου μια συνάρτηση καλεί τον εαυτό της, με παράδειγμα το παραγοντικό. Οι δείκτες είναι το θεμέλιο για όλο το υπόλοιπο μάθημα: δυναμική μνήμη, συμβολοσειρές, λίστες και δέντρα.

Θεωρία

§11.1 Η μνήμη και οι διευθύνσεις

Όπως είδαμε στο Κεφάλαιο 2, η μνήμη του υπολογιστή είναι μια μεγάλη σειρά από **bytes** (1 KB = 1.000 bytes, 1 MB = 1.000.000 bytes, 1 GB = 1.000.000.000 bytes). Μια μνήμη με N bytes έχει τα bytes 0, 1, …, N−1. Η θέση ενός κελιού στη μνήμη λέγεται **διεύθυνση (address)**: για παράδειγμα, «στη διεύθυνση 2 υπάρχει το byte 11100011 (δυαδικό)».

Μια **μεταβλητή (variable)** είναι ένα τμήμα της μνήμης με συγκεκριμένο όνομα, και για να χρησιμοποιηθεί πρέπει να έχει **δηλωθεί** με κάποιον **τύπο (type)**. Στη δήλωση `int x;` ο τύπος λέει στον μεταγλωττιστή *πόση* μνήμη να δεσμεύσει (π.χ. 4 bytes για έναν `int`), και το όνομα τον κάνει να διαλέξει *πού*, δηλαδή σε ποια διεύθυνση, θα αποθηκευτεί η μεταβλητή.

Η **ανάθεση (assignment)** μπορεί να γίνει κατά τον ορισμό (`int x = 42;`), αργότερα (`int x;` και μετά `x = 42;`) ή με δεκαεξαδική σταθερά (`int x = 0x2A;`, το ίδιο 42). Πριν από την ανάθεση, τα bytes της `x` έχουν ό,τι «σκουπίδια» έτυχε να υπάρχουν εκεί· μετά, κρατούν την αναπαράσταση του 42 (`00101010` στο δυαδικό).

§11.2 Ο τελεστής &: η διεύθυνση μιας μεταβλητής

Τη διεύθυνση μιας μεταβλητής τη βρίσκουμε με τον μοναδιαίο τελεστή `&` (ampersand). Αν η `x` τοποθετήθηκε στα bytes 100–103, τότε το `&x` είναι 100: η διεύθυνση του *πρώτου* byte της.

Διεύθυνση	Περιεχόμενο
100	`00000000`
101	`00000000`
102	`00000000`
103	`00101010`

Η `int x = 42;` *στα bytes 100–103, όπως τη σχεδιάζουν οι διαφάνειες.*

Η διεύθυνση είναι πάντα ένας **ακέραιος** αριθμός, με όσα bits αποφασίσει ο μεταγλωττιστής: 32 σε συστήματα 32 bit (`gcc -m32`), 64 σε συστήματα 64 bit. Και οι διευθύνσεις **μπορούν να αλλάζουν από εκτέλεση σε εκτέλεση**: το 100 ισχύει για μία εκτέλεση του προγράμματος, όχι για πάντα.

Οι διαφάνειες σχεδιάζουν τον ακέραιο με το σημαντικό του μέρος στο πρώτο byte και την τιμή 42 στο τελευταίο. Η πραγματική σειρά των bytes εξαρτάται από τον επεξεργαστή (endianness) και τη βλέπουμε στο Κεφάλαιο 12.

§11.3 Ο τύπος δείκτη

Πέρα από τους βασικούς τύπους `int`, `char`, `double`, προσθέτουμε έναν νέο τύπο για να αποθηκεύουμε διευθύνσεις. Ένας **δείκτης (pointer)** είναι μια μεταβλητή που περιέχει τη διεύθυνση μνήμης ενός δεδομένου συγκεκριμένου τύπου. Η γενική μορφή της δήλωσης είναι:

```
τύπος * όνομα;      // π.χ.
int * pointer;
```

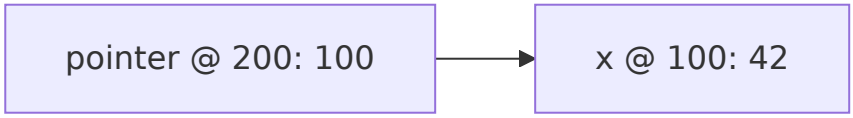
Ο τύπος τύπος `*` λέει στον μεταγλωττιστή ότι η διεύθυνση που θα αποθηκεύει ο δείκτης είναι για δεδομένα τύπου τύπος: ο `int *` δείχνει σε `int`, ο `char *` σε `char`. Το όνομα είναι, όπως πάντα, η μεταβλητή που κρατά την τιμή του δείκτη, και ο μεταγλωττιστής επιλέγει πού θα αποθηκευτεί. Τα κενά γύρω από το `*` δεν έχουν σημασία: `int *p`, `int * p` και `int* p` είναι το ίδιο. Σε δήλωση πολλών μεταβλητών όμως το `*` ανήκει στο όνομα: `int *pa = &a, *pb = &b;` δηλώνει δύο δείκτες.

§11.4 Αρχικοποίηση ενός δείκτη

Αρχικοποιούμε έναν δείκτη με τη διεύθυνση μιας μεταβλητής του σωστού τύπου:

```
int x = 42;
int *pointer = &x;
```

Τώρα λέμε ότι ο `pointer` **δείχνει (points to)** στη μεταβλητή `x`. Ο ίδιος ο δείκτης είναι μια μεταβλητή με δική της θέση στη μνήμη. Αν η `x` είναι στη διεύθυνση 100 και ο `pointer` στη 200, τότε το περιεχόμενο του `pointer` είναι 100 και το `&pointer` είναι 200· το `printf("%d, %d\n", pointer, &pointer);` των διαφανειών τυπώνει `100, 200` (το σωστό format για διευθύνσεις είναι το `%p`, βλ. παρακάτω):



Σχήμα: ο δείκτης (στη διεύθυνση 200) κρατά την τιμή 100, τη διεύθυνση της `x`.

Το ίδιο γίνεται με κάθε τύπο: με `char c = 42; char *pointer = &c;` και τη `c` στη διεύθυνση 150, ο `pointer` περιέχει 150. Η διαφορά είναι ότι ο `char` πιάνει 1 byte, ενώ ο `int` 4.

§11.5 Ο τελεστής `sizeof`

Ο τελεστής `sizeof` υπολογίζει πόσα bytes πιάνει στη μνήμη ένας τύπος ή μια μεταβλητή. Το αποτέλεσμα έχει τύπο `size_t` και το σωστό format για την `printf` είναι το `%zu` (το `%d` συνήθως «δουλεύει», αλλά ο `gcc -Wall` προειδοποιεί):

```
printf("int size: %zu\n", sizeof(int));   // int size: 4
```

Πόσο μεγάλος είναι ένας δείκτης; Όσο μια διεύθυνση, **ανεξάρτητα από τον τύπο στον οποίο δείχνει**. Σε σύστημα 64 bit οι `int *`, `char *` και `double *` πιάνουν όλοι 8 bytes (σε 32 bit, 4). Ο τύπος του δείκτη δεν αλλάζει το μέγεθός του, αλλάζει το πώς ερμηνεύεται η μνήμη όπου δείχνει.

§11.6 Η ειδική τιμή `NULL`

Όταν θέλουμε να δηλώσουμε ότι ένας δείκτης **δεν δείχνει σε κάποια μεταβλητή**, του αναθέτουμε την τιμή `NULL` (τη διεύθυνση 0), και μπορούμε να την ελέγξουμε:

```
int * ipointer = NULL;
if (ipointer == NULL) {
  printf("pointer does not point anywhere\n");
}
```

Δεν υπάρχει περίπτωση να βρίσκεται μια μεταβλητή στη διεύθυνση 0; Θεωρητικά ναι, πρακτικά όχι: τα συστήματα κρατούν αυτή τη διεύθυνση εκτός χρήσης ακριβώς για να

σημαίνει «πουθενά». Ο Tony Hoare, που εισήγαγε τις null αναφορές, την ονόμασε «the billion dollar mistake», γιατί η χρήση ενός `NULL` δείκτη σαν να ήταν έγκυρος είναι μία από τις πιο συχνές αιτίες σφαλμάτων.

§11.7 Αποαναφορά: ο τελεστής *

Ο δείκτης δείχνει σε μια μεταβλητή· μπορούμε να φτάσουμε στη μεταβλητή έχοντας μόνο τη διεύθυνσή της; Ναι: με τον μοναδιαίο τελεστή `*`, που λέγεται **αποαναφορά (dereference)**. Το `*pointer` είναι η μεταβλητή στην οποία δείχνει ο `pointer`:

```
int x = 42;
int *pointer = &x;
printf("%d\n", *pointer);   // 42
```

Η χρήση του `*pointer` είναι **ισοδύναμη με τη χρήση της μεταβλητής** `x`, και για διάβασμα και για γράψιμο: το `*pointer = 7;` αλλάζει την `x`. Προσέξτε ότι το `*` έχει δύο ρόλους: στη δήλωση (`int *p`) λέει «ο `p` είναι δείκτης», ενώ σε μια έκφραση (`*p`) λέει «πήγαινε εκεί που δείχνει ο `p`».

§11.8 Μη έγκυροι δείκτες

Για να χρησιμοποιήσουμε το περιεχόμενο της διεύθυνσης όπου δείχνει ένας δείκτης, η διεύθυνση πρέπει πρώτα να **υπάρχει**:

```
int *pointer;
printf("%d\n", *pointer);   // μη αρχικοποιημένος δείκτης!
```

Αυτό κατά πάσα πιθανότητα θα τερματίσει με **segmentation fault**: ο `pointer` δεν αρχικοποιήθηκε, άρα περιέχει μια τυχαία τιμή που δεν είναι έγκυρη διεύθυνση μνήμης. Το ίδιο συμβαίνει με αποαναφορά ενός `NULL` δείκτη. Κανόνας: κάθε δείκτης είτε δείχνει σε κάτι έγκυρο, είτε είναι `NULL` και ελέγχεται πριν χρησιμοποιηθεί.

§11.9 Οι τελεστές * και & είναι συμπληρωματικοί

Ο `*` δίνει τη μεταβλητή σε μια διεύθυνση, ο `&` τη διεύθυνση μιας μεταβλητής. Είναι λοιπόν **αντίστροφοι**: όταν εφαρμόζονται σε έγκυρους δείκτες αλληλοαναιρούνται. Με `int x = 42; int *pointer = &x;` τα `pointer`, `&*pointer` και `*&pointer` έχουν όλα την ίδια τιμή. Για να τυπώσουμε μια διεύθυνση χρησιμοποιούμε το format `%p`, που τη δείχνει στο δεκαεξαδικό:

```
$ ./pointer_size
0x7ffcf94870cc 0x7ffcf94870cc 0x7ffcf94870cc
```

§11.10 Πράξεις με δείκτες

Σε δείκτες επιτρέπονται μόνο οι εξής πράξεις:

- πρόσθεση ή αφαίρεση ακεραίου σε/από δείκτη (και τα `++`, `--`, `+=`, `-=`)·

- αφαίρεση δύο δεικτών·
- σύγκριση δύο δεικτών ή σύγκριση με το 0 (`NULL`).

Το κρίσιμο σημείο είναι η πρόσθεση. **Η πρόσθεση ενός ακεραίου αυξάνει τη διεύθυνση κατά το μέγεθος του τύπου όπου δείχνει ο δείκτης, πολλαπλασιασμένο με τον ακέραιο**. Για τύπος `*pointer`; το `pointer += N` μεταφέρει τη διεύθυνση κατά `N * sizeof(τύπος)` bytes:

```
int *ipointer;    ipointer += 2;   // + 2 * sizeof(int)    = 8 bytes
char *cpointer;   cpointer += 2;   // + 2 * sizeof(char)   = 2 bytes
double *dpointer; dpointer += 2;   // + 2 * sizeof(double) = 16 bytes
```

Δηλαδή ο δείκτης μετακινείται κατά N **στοιχεία**, όχι κατά N bytes. Έτσι, αν ο `int *pointer` δείχνει στη 100, το `++pointer` τον κάνει 104· αν ο `char *pointer` δείχνει στη 150, το `--pointer` τον κάνει 149. Η αφαίρεση δύο δεικτών δίνει αντίστοιχα πόσα στοιχεία απέχουν, και έχει νόημα, όπως και η σύγκρισή τους, μόνο όταν δείχνουν στο ίδιο μπλοκ μνήμης, π.χ. στον ίδιο πίνακα (σημειώσεις, «Δείκτες»).

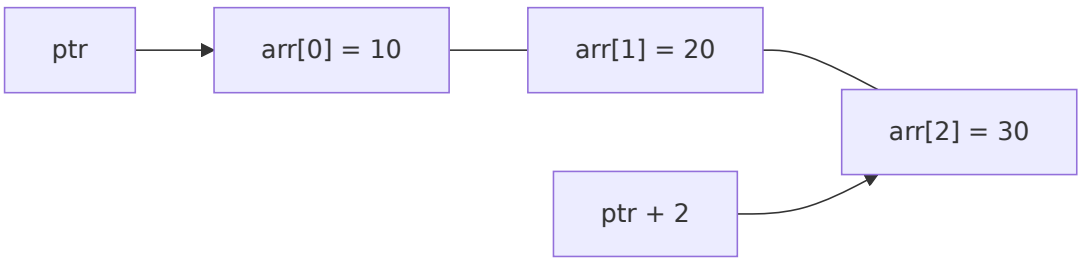
Γιατί να μετακινήσουμε έναν δείκτη; Η μνήμη δίπλα σε μια απλή μεταβλητή δεν μας ανήκει. Έχει νόημα όταν ο δείκτης δείχνει μέσα σε **συνεχόμενες θέσεις ίδιου τύπου**, δηλαδή σε έναν πίνακα.

§11.11 Δείκτες και πίνακες: `*(ptr + n)` και `ptr[n]`

Τα στοιχεία ενός πίνακα είναι σε συνεχόμενες θέσεις (Κεφάλαιο 10). Αν `ptr = &arr[0]`, τότε το `ptr + 2` δείχνει στο `arr[2]` και το `*(ptr + 2)` είναι το ίδιο το `arr[2]`. Η έκφραση `*(ptr + n)`, «υπολόγισε τη διεύθυνση n στοιχεία μετά τον δείκτη και δώσε μου τη μεταβλητή εκεί», είναι τόσο συχνή που η C έχει συντομογραφία:

```
*(ptr + n)   // ≡ ptr[n]
*(ptr + 3)   // ≡ ptr[3]
```

Μας θυμίζει κάτι; Είναι ακριβώς η σύνταξη αναφοράς σε στοιχείο πίνακα. Και πράγματι, το όνομα ενός πίνακα συμπεριφέρεται σαν δείκτης **κολλημένος** στο πρώτο του στοιχείο: για `int a[100];` το `a` είναι το `&a[0]`, το `a[i]` είναι το `*(a + i)` και το `a + i` είναι το `&a[i]`.



Σχήμα: ο `ptr` δείχνει στο `arr[0]`· το `ptr + 2` δείχνει δύο στοιχεία (8 bytes) πιο μετά, στο `arr[2]`.

Γι' αυτό μια συνάρτηση που δέχεται πίνακα στην πραγματικότητα δέχεται τη διεύθυνση του

πρώτου στοιχείου: η παράμετρος `int grades[100]` είναι ισοδύναμη με `int *grades`, και ο πίνακας δεν αντιγράφεται (σημειώσεις, «Πίνακες»).

§11.12 Διαφορές πινάκων και δεικτών

Παρόλο που η προσπέλαση στοιχείων είναι ίδια και ο πίνακας είναι ουσιαστικά δείκτης στο πρώτο στοιχείο, πίνακας και δείκτης **δεν** είναι το ίδιο πράγμα. Με `int a[100]; int *ptr;`:

	Πίνακας `a`	Δείκτης `ptr`
Αλλαγή διεύθυνσης	Δεν γίνεται: `a = ptr;` και `a++` είναι λάθη	Επιτρέπεται: `ptr = a;`, `ptr++`
Τι δημιουργεί η δήλωση	Θέσεις για τα στοιχεία (100 `int`)	Θέση για μία διεύθυνση
`sizeof`	Όλος ο πίνακας: `100 * sizeof(int)`	Μία διεύθυνση (8 σε 64 bit)
`&`	`&a` έχει τη διεύθυνση του πρώτου στοιχείου (`&a[0]`)	`&ptr` είναι η διεύθυνση του ίδιου του δείκτη

§11.13 Αναδρομή

Αναδρομή (recursion) είναι η μέθοδος κατά την οποία μια συνάρτηση **καλεί τον εαυτό της** για να λύσει ένα υποπρόβλημα του αρχικού προβλήματος, έως ότου φτάσει σε μια βάση τερματισμού, όπου η αναδρομή σταματά. Έχει δύο βασικά στοιχεία:

1. **Βασική περίπτωση τερματισμού (base case)**: μια συνθήκη που καθορίζει πότε θα σταματήσει η αναδρομή, και η απάντηση δίνεται απευθείας.
2. **Αναδρομική περίπτωση (recursive case)**: το τμήμα του κώδικα όπου η συνάρτηση καλεί τον εαυτό της με ένα «μικρότερο» πρόβλημα, που πλησιάζει τη βάση.

Το κλασικό παράδειγμα είναι το **παραγοντικό (factorial)**: το $n!$ ενός φυσικού αριθμού είναι το γινόμενο όλων των θετικών ακεραίων μικρότερων ή ίσων του n. Ο μαθηματικός του ορισμός είναι ήδη αναδρομικός:

$$n! = 1 \text{ αν } n = 0, \qquad n! = n \times (n-1)! \text{ αν } n > 0$$

και μεταφράζεται στην C σχεδόν λέξη προς λέξη:

```c
int factorial(int number) {
  if (number == 0) return 1;                  // base case
  else return number * factorial(number - 1); // recursive case
}
```

Η αναδρομική υλοποίηση είναι πολύ κοντά στον ορισμό, κι αυτό κάνει «εύκολο» τον έλεγχο ορθότητας: αρκεί να ελέγξουμε ότι η βάση είναι σωστή και ότι κάθε βήμα εφαρμόζει σωστά τον κανόνα. Κάθε κλήση περιμένει το αποτέλεσμα της επόμενης, και ο πολλαπλασιασμός γίνεται στην «επιστροφή»:

Σχήμα: η αλυσίδα κλήσεων του `factorial(5)`*· οι τιμές επιστρέφονται από δεξιά προς τα αριστερά: 1, 1, 2, 6, 24, 120.*

Η αναδρομή πρέπει να τερματίζει. Αν η βάση λείπει ή δεν φτάνεται ποτέ, η συνάρτηση καλεί τον εαυτό της ξανά και ξανά. Κάθε κλήση που δεν έχει επιστρέψει πιάνει χώρο στη μνήμη (στη στοίβα, βλ. Κεφάλαιο 13), οπότε κάποια στιγμή η μνήμη αυτή εξαντλείται και το πρόγραμμα καταρρέει, συνήθως με segmentation fault. Μην ξεχνάτε λοιπόν να γράφετε σωστά base cases, που να καλύπτουν *όλες* τις εισόδους.

Παραδείγματα

§11.14 Τι τυπώνει: αναθέσεις μέσω δεικτών

Εφαρμογή της αποαναφοράς (`*`) για γράψιμο και διάβασμα:

```c
#include <stdio.h>
int main() {
  int a = 100, b = 200, c;
  int *ptr_a = &a, *ptr_b = &b, *ptr_c = &c;
  *ptr_c = a;
  *ptr_a = b;
  *ptr_b = *ptr_c;
  printf("%d %d %d", a, b, c);
  return 0;
}
```

Ακολουθούμε ποια μεταβλητή αλλάζει σε κάθε γραμμή: `*ptr_c = a` είναι `c = 100`· `*ptr_a = b` είναι `a = 200`· `*ptr_b = *ptr_c` είναι `b = c`, δηλαδή `b = 100`. Τυπώνει `200 100 100` (χωρίς αλλαγή γραμμής, αφού λείπει το `\n`). Οι δείκτες εδώ λειτουργούν απλώς ως δεύτερα ονόματα για τις `a`, `b`, `c`.

§11.15 Αναφορά σε στοιχεία πίνακα μέσω δείκτη

Εφαρμογή της αριθμητικής δεικτών σε πίνακα:

```c
#include <stdio.h>
int main() {
```

```
  int *ptr, arr[] = {10, 20, 30};
  ptr = &arr[0];
  printf("mem[%p], %d\n", ptr, *ptr);
  ptr += 2;
  printf("mem[%p], %d\n", ptr, *ptr);
  return 0;
}
```

```
$ ./test
mem[0xffa35e60], 10
mem[0xffa35e68], 30
```

Το `ptr += 2` πρόσθεσε `2 * sizeof(int) = 8` στη διεύθυνση (`0x...60` ⟶ `0x...68`) και ο δείκτης δείχνει πια στο `arr[2]`. Οι ίδιες οι διευθύνσεις θα είναι διαφορετικές στον δικό σας υπολογιστή· η διαφορά 8 όχι.

§11.16 Πώς τυπώνω μόνο το `"World\n"`;

Από την προηγούμενη διάλεξη έχουμε μια συμβολοσειρά, και θέλουμε να τυπώσουμε μόνο το δεύτερο μισό της:

```
char hello[] = "Hello World\n";
char *world = &hello[6];
printf("%s", world);          // World
```

Το `%s` τυπώνει χαρακτήρες από τη διεύθυνση που του δίνουμε μέχρι το `'\0'`. Το `&hello[6]` (ισοδύναμα `hello + 6`) είναι η διεύθυνση του `'W'`, άρα τυπώνεται `World` και η αλλαγή γραμμής. Δεν χρειάστηκε καμία αντιγραφή: ο `world` απλώς δείχνει μέσα στον ίδιο πίνακα.

§11.17 Μέσος όρος ενός πίνακα 100 ακεραίων

Μια συνάρτηση που δέχεται πίνακα 100 ακεραίων και επιστρέφει τον μέσο όρο:

```
int average(int grades[100]) {
  int i, sum = 0;
  for (i = 0; i < 100; i++) {
    sum += grades[i];
  }
  return sum / 100;
}
```

Η συνάρτηση παίρνει τη διεύθυνση του πίνακα (όχι αντίγραφο) και διατρέχει τα στοιχεία με `grades[i]`. Προσέξτε ότι το `sum / 100` είναι ακέραια διαίρεση: ο μέσος όρος στρογγυλεύεται προς τα κάτω. Για ακρίβεια θα επιστρέφαμε `double` και θα γράφαμε `sum / 100.0`.

§11.18 Αναζήτηση στοιχείου σε πίνακα

Μια συνάρτηση που δέχεται πίνακα 100 ακεραίων και έναν ακέραιο, και επιστρέφει τη θέση του στοιχείου αν το βρει, αλλιώς `-1`:

```
int find(int haystack[100], int needle) {
  int i;
  for (i = 0; i < 100; i++) {
    if (haystack[i] == needle) {
      return i;
    }
  }
  return -1;
}
```

Το `return i` μέσα στον βρόχο τερματίζει αμέσως τη συνάρτηση στην πρώτη εμφάνιση. Αν ο βρόχος τελειώσει χωρίς να βρει τίποτα, φτάνουμε στο `return -1`. Το `-1` είναι ασφαλής «σημαία αποτυχίας», γιατί δεν είναι ποτέ έγκυρη θέση πίνακα.

§11.19 Η δική μας `atoi`

Μια συνάρτηση που παίρνει πίνακα χαρακτήρων (μόνο ψηφία) και επιστρέφει τον ακέραιο που αναπαριστά:

```
int atoi(char digits[]) {
  int result = 0;
  for (int i = 0; digits[i]; i++) {
    result = 10 * result + digits[i] - '0';
  }
  return result;
}
```

Η συνθήκη `digits[i]` σταματά στο `'\0'` (τιμή 0) στο τέλος της συμβολοσειράς. Το `digits[i] - '0'` μετατρέπει τον χαρακτήρα ψηφίου στην τιμή του (`'7' - '0' == 7`), και το `10 * result +` «σπρώχνει» τα ψηφία που έχουμε ήδη μία θέση αριστερά: για `"123"` το `result` γίνεται 1, 12, 123.

Τι μπορεί να πάει στραβά; Η συνάρτηση υποθέτει πολλά: αν η είσοδος έχει χαρακτήρα που δεν είναι ψηφίο (π.χ. `"12a"` ή `"-5"`) υπολογίζει σκουπίδια χωρίς να το αναφέρει· αν ο αριθμός δεν χωρά σε `int` έχουμε υπερχείλιση· και αν ο πίνακας δεν τελειώνει σε `'\0'`, ο βρόχος διαβάζει εκτός ορίων. Επίσης το όνομα `atoi` συγκρούεται με τη συνάρτηση της `stdlib.h`, οπότε σε πραγματικό πρόγραμμα θα διαλέγαμε άλλο όνομα.

§11.20 Παραγοντικό από τη γραμμή εντολών

Εφαρμογή της αναδρομής: ολόκληρο το πρόγραμμα, που διαβάζει τον αριθμό από τη γραμμή εντολών (τα `argc`/`argv` τα εξηγούμε στο Κεφάλαιο 12):

```c
#include <stdio.h>
#include <stdlib.h>
// Compute the factorial of a number using the recursive
// formula.
int factorial(int number) {
  if (number == 0) return 1;
  else return number * factorial(number - 1);
}
int main(int argc, char **argv) {
  if (argc != 2) {
    printf("Program needs to be called as `./prog number`\n");
    return 1;
  }
  int number = atoi(argv[1]);
  printf("%d! = %d\n", number, factorial(number));
  return 0;
}
```

Οι διαφάνειες θέτουν τρεις ερωτήσεις για αυτό το πρόγραμμα:

- **Για κάποιους αριθμούς το αποτέλεσμα είναι αρνητικό. Τι συμβαίνει;**

  ```
  $ ./fact 20
  20! = -2102132736
  ```

  Είναι **υπερχείλιση ακεραίου**: ο `int` των 32 bit φτάνει μέχρι 2.147.483.647. Το $12! = 479.001.600$ χωρά, αλλά ήδη το $13!$ δεν χωρά, και από εκεί και πέρα τα αποτελέσματα είναι λάθος (το αρνητικό πρόσημο είναι απλώς το πιο εμφανές σύμπτωμα).

- **Πόσες αναδρομικές κλήσεις κάνει το `factorial(5)`; Και το `factorial(N)`;** Το `factorial(5)` καλεί τα `factorial(4)`, ..., `factorial(0)`: 5 αναδρομικές κλήσεις (6 κλήσεις συνολικά). Γενικά το `factorial(N)` κάνει N αναδρομικές κλήσεις.

- **Τι θα συμβεί αν δώσουμε αρνητικό αριθμό;** Το `number` δεν φτάνει ποτέ στο 0 (-1, -2, -3, ...), άρα η βάση δεν ενεργοποιείται ποτέ και η αναδρομή δεν τερματίζει, μέχρι να εξαντληθεί η στοίβα. Η διόρθωση είναι να ελέγχουμε την είσοδο ή να γράψουμε τη βάση ως `number <= 0`.

Για εξάσκηση στην αναδρομή, το Εργαστήριο 5 έχει την ακολουθία Collatz και τους αριθμούς Fibonacci.

Κύρια σημεία

1. Η μνήμη είναι μια σειρά από bytes, και η θέση κάθε byte λέγεται διεύθυνση· η διεύθυνση μιας μεταβλητής είναι η διεύθυνση του πρώτου της byte.
2. Ο τελεστής `&` δίνει τη διεύθυνση μιας μεταβλητής· η διεύθυνση είναι ακέραιος και μπορεί να αλλάζει από εκτέλεση σε εκτέλεση.
3. Ένας δείκτης (`τύπος *όνομα`) είναι μεταβλητή που κρατά τη διεύθυνση ενός δεδομένου του συγκεκριμένου τύπου.
4. Όλοι οι δείκτες έχουν το ίδιο μέγεθος, όσο μια διεύθυνση (8 bytes σε 64 bit), ανεξάρτητα από τον τύπο στον οποίο δείχνουν.
5. Ο τελεστής `*` (αποαναφορά) δίνει τη μεταβλητή όπου δείχνει ο δείκτης· οι `*` και `&` είναι αντίστροφοι, και οι διευθύνσεις τυπώνονται με `%p`.
6. Το `NULL` σημαίνει «πουθενά»· αποαναφορά ενός `NULL` ή μη αρχικοποιημένου δείκτη οδηγεί συνήθως σε segmentation fault.
7. Σε δείκτες επιτρέπονται πρόσθεση/αφαίρεση ακεραίου, αφαίρεση δύο δεικτών και σύγκριση· το `p + N` προχωρά κατά `N * sizeof(τύπος)` bytes, δηλαδή N στοιχεία.
8. Το `*(ptr + n)` γράφεται συντομότερα `ptr[n]`, και το όνομα ενός πίνακα είναι δείκτης κολλημένος στο πρώτο του στοιχείο.
9. Ένας πίνακας δεν μπορεί να αλλάξει διεύθυνση, και τα `sizeof`, `&` δίνουν άλλα αποτελέσματα σε πίνακα και σε δείκτη.
10. Μια συνάρτηση που δέχεται πίνακα δουλεύει πάνω στον ίδιο πίνακα, μέσω της διεύθυνσής του.
11. Αναδρομή είναι μια συνάρτηση που καλεί τον εαυτό της· χρειάζεται base case και recursive case που πλησιάζει τη βάση, και η αναδρομική υλοποίηση του παραγοντικού ακολουθεί κατά λέξη τον μαθηματικό ορισμό.
12. Η αναδρομή πρέπει να τερματίζει: χωρίς σωστή βάση για όλες τις εισόδους, το πρόγραμμα καταρρέει.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
διεύθυνση	address	Η θέση ενός byte στη μνήμη· ένας ακέραιος.
δείκτης	pointer	Μεταβλητή που κρατά τη διεύθυνση ενός δεδομένου.
δείχνει σε	points to	Ο δείκτης κρατά τη διεύθυνση της μεταβλητής.
αποαναφορά	dereference	Πρόσβαση με `*` στη μεταβλητή όπου δείχνει ο δείκτης.
κενός δείκτης	null pointer (`NULL`)	Δείκτης με τιμή 0 που δεν δείχνει πουθενά.

Ελληνικά	English	Σύντομος ορισμός
μη έγκυρος δείκτης	invalid pointer	Δείκτης που δεν κρατά έγκυρη διεύθυνση.
σφάλμα κατάτμησης	segmentation fault	Τερματισμός από πρόσβαση σε μη επιτρεπτή μνήμη.
αριθμητική δεικτών	pointer arithmetic	Το `p + n` δείχνει `n` στοιχεία (όχι bytes) μετά.
αναδρομή	recursion	Μια συνάρτηση καλεί τον εαυτό της.
βασική περίπτωση	base case	Η συνθήκη όπου η αναδρομή σταματά.
αναδρομική περίπτωση	recursive case	Το σημείο όπου η συνάρτηση καλεί τον εαυτό της.
παραγοντικό	factorial	$n! = 1 \cdot 2 \cdots n$, με $0! = 1$.
υπερχείλιση	overflow	Αποτέλεσμα που δεν χωρά στον τύπο του.

Διάβασμα

- **Διαφάνειες**: Διάλεξη 11, σελ. 1–50. Μνήμη και διευθύνσεις σελ. 5–11· δείκτες, `sizeof` και `NULL` σελ. 12–19· αποαναφορά και μη έγκυροι δείκτες σελ. 20–24· πράξεις με δείκτες σελ. 25–28· δείκτες και πίνακες σελ. 29–35· παραδείγματα με πίνακες σελ. 36–42· αναδρομή σελ. 43–48.
- **Σημειώσεις**: η διάλεξη μαζί με την επόμενη καλύπτει τις σελίδες 73–103 των σημειώσεων του κ. Σταματόπουλου:
  - Κεφάλαιο 5: Δείκτες και πίνακες, ενότητες «Δείκτες» (K04, σελ. 72–77) και «Πίνακες» (K04, σελ. 80–85).
  - Κεφάλαιο 4: Συναρτήσεις, εμβέλεια και αναδρομή, ενότητες «Συνάρτηση υπολογισμού παραγοντικού» (K04, σελ. 62) και «Υπολογισμός παραγοντικού με αναδρομή» (K04, σελ. 69).
- **Εργαστήριο**:
  - Εργαστήριο 6: ασκήσεις `myprog.c` (πέρασμα δεδομένων μέσω δεικτών), `pointers.c` (πίνακες και αριθμητική δεικτών), `judgement.c` (πίνακες και συναρτήσεις).
  - Εργαστήριο 5: ασκήσεις `collatz.c` και `fib.c` (αναδρομή).
- **Άλλα**:
  - Tutorials για pointers: W3Schools, Tutorialspoint, GeeksforGeeks, Programiz.
  - Tony Hoare, «Null References: The Billion Dollar Mistake» (ομιλία).

Συχνά λάθη

- **Αποαναφορά μη αρχικοποιημένου δείκτη.** `int *p; *p = 5;` ⟶ `Segmentation fault` (ή, χειρότερα, σιωπηλή αλλοίωση άλλης μνήμης). Αρχικοποιείτε κάθε δείκτη, έστω με `NULL`,

και ελέγχετε πριν τον χρησιμοποιήσετε.

- **Διευθύνσεις με %d.** `printf("%d", ptr)` δίνει warning (`format '%d' expects argument of type 'int'`) και σε 64 bit τυπώνει μόνο το μισό της διεύθυνσης. Χρησιμοποιήστε `%p`.
- **Ένα * για πολλές μεταβλητές.** Στο `int *p, q;` μόνο ο `p` είναι δείκτης, ο `q` είναι `int`. Γράψτε `int *p, *q;`.
- **Σύγχυση του * της δήλωσης με το * της έκφρασης.** Το `int *p = &x;` αρχικοποιεί τον `p`, όχι το `*p`· αργότερα γράφετε `p = &x;`, όχι `*p = &x;`.
- **Αριθμητική δεικτών σε bytes.** Για `int *p` στη διεύθυνση 100, το `p + 1` είναι 104, όχι 101.
- **Ανάθεση σε πίνακα.** `int a[100]; a = ptr;` ⟶ `assignment to expression with array type`. Ένας πίνακας δεν αλλάζει διεύθυνση· χρησιμοποιήστε δείκτη.
- **Ακέραια διαίρεση στον μέσο όρο.** `sum / 100` με `int` χάνει το δεκαδικό μέρος. Γράψτε `sum / 100.0` και επιστρέψτε `double`.
- **Αναδρομή χωρίς (σωστό) base case.** `factorial(-1)` με βάση `number == 0` δεν τερματίζει ⟶ segmentation fault. Βεβαιωθείτε ότι κάθε είσοδος φτάνει στη βάση.
- **Υπερχείλιση στο παραγοντικό.** Από το `13!` και πάνω ο `int` υπερχειλίζει (`20! = -2102132736`). Ελέγξτε το εύρος της εισόδου ή χρησιμοποιήστε μεγαλύτερο τύπο.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K11.13** Άθροισμα δύο δεικτών (28% σωστές): Το 28% επέλεξε `166` και το 22% `84`, θεωρώντας ότι δύο δείκτες προστίθενται σαν ακέραιοι. Η C επιτρέπει δείκτη ± ακέραιο και διαφορά δύο δεικτών, όχι όμως άθροισμα δεικτών.
- **K11.12** Δείκτης συν 4 σε δεκαεξαδικό (31% σωστές): Το 32% επέλεξε `0x4C` και το 22% `0x46`. Οι δεύτεροι ξέχασαν ότι το `+ 4` σημαίνει 4 × `sizeof(int)` bytes, ενώ οι πρώτοι πρόσθεσαν σωστά 16 bytes, αλλά μπέρδεψαν τη δεκαεξαδική πρόσθεση (16 = `0x10`).
- **K11.9** Αύξηση του ίδιου του δείκτη (37% σωστές): Το 25% επέλεξε `42`, θεωρώντας ότι το `ptr++` αυξάνει την τιμή του `x`. Στην πραγματικότητα αλλάζει τη διεύθυνση που κρατά ο δείκτης, όχι την τιμή στην οποία δείχνει.
- **K11.8** Μέγεθος δεικτών (46% σωστές): Το 49% απάντησε False, μπερδεύοντας το μέγεθος του δείκτη με το μέγεθος του τύπου στον οποίο δείχνει. Κάθε δείκτης κρατά μια διεύθυνση, άρα όλοι έχουν το ίδιο μέγεθος (8 bytes σε 64-bit συστήματα).
- **K11.6** Διεύθυνση και ακέραιος (50% σωστές): Το 47% απάντησε False, ίσως επειδή η διεύθυνση έχει τύπο δείκτη ή επειδή τη βλέπουμε σε δεκαεξαδικό. Όμως κάθε διεύθυνση είναι ο αύξων αριθμός ενός byte στη μνήμη, δηλαδή ακέραιος.
- **K11.4** Η τιμή του NULL (65% σωστές): Το 28% απάντησε False, θεωρώντας το `NULL` ειδική τιμή χωρίς αριθμητική αναπαράσταση. Στην πράξη το `NULL` είναι η διεύθυνση 0.
- **K11.3** *(ptr + 4) και ptr[4] (67% σωστές): Το 26% απάντησε False, θεωρώντας ότι οι αγκύλες είναι κάτι διαφορετικό από την αριθμητική δεικτών. Στην C το `ptr[n]` ορίζεται ακριβώς ως `*(ptr + n)`.

Ερωτήσεις κατανόησης

- **E11.1** Τι είναι η διεύθυνση μιας μεταβλητής `int` που πιάνει τα bytes 100–103;[93]
- **E11.2** Με `int x = 42; int *p = &x;`, τι είναι τα `p`, `*p` και `&p`;[94]
- **E11.3** Πόσα bytes πιάνουν ένας `char *` και ένας `double *` σε σύστημα 64 bit;[95]
- **E11.4** Γιατί το `int *p; printf("%d", *p);` πιθανότατα καταρρέει;[96]
- **E11.5** Ο `double *d` δείχνει στη διεύθυνση 1000. Πού δείχνει το `d + 3`;[97]
- **E11.6** Γράψτε το `arr[4]` με αριθμητική δεικτών.[98]
- **E11.7** Για `int a[100]; int *ptr = a;`, ποιο είναι το `sizeof(a)` και ποιο το `sizeof(ptr)` αν `sizeof(int) == 4` σε 64 bit;[99]
- **E11.8** Ποια είναι τα δύο στοιχεία κάθε αναδρομικής συνάρτησης;[100]
- **E11.9** Πόσες φορές καλείται συνολικά η `factorial` για `factorial(3)`;[101]

Kahoot από το αμφιθέατρο (K11.1–K11.14)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K11.1** Τερματισμός αναδρομής: 87% σωστές απαντήσεις
- **K11.2** Πίνακας και δείκτης: 71% σωστές απαντήσεις
- **K11.3** *(ptr + 4) και ptr[4]: 67% σωστές απαντήσεις
- **K11.4** Η τιμή του NULL: 65% σωστές απαντήσεις
- **K11.5** Η τιμή του ptr + 1: 61% σωστές απαντήσεις
- **K11.6** Διεύθυνση και ακέραιος: 50% σωστές απαντήσεις
- **K11.7** Αλλαγή μεταβλητής και δείκτης: 48% σωστές απαντήσεις
- **K11.8** Μέγεθος δεικτών: 46% σωστές απαντήσεις
- **K11.9** Αύξηση του ίδιου του δείκτη: 37% σωστές απαντήσεις
- **K11.10** Η έκφραση *&x: 35% σωστές απαντήσεις
- **K11.11** Αύξηση μέσω δείκτη: 34% σωστές απαντήσεις
- **K11.12** Δείκτης συν 4 σε δεκαεξαδικό: 31% σωστές απαντήσεις
- **K11.13** Άθροισμα δύο δεικτών: 28% σωστές απαντήσεις
- **K11.14** Δείκτης στη μέση πίνακα: 12% σωστές απαντήσεις

[93] Η διεύθυνση του πρώτου byte της, 100.

[94] `p` είναι η διεύθυνση της `x`· `*p` είναι η ίδια η `x` (42)· `&p` είναι η διεύθυνση του δείκτη `p`.

[95] 8 bytes ο καθένας: ο δείκτης κρατά μια διεύθυνση, ανεξάρτητα από τον τύπο.

[96] Ο `p` δεν αρχικοποιήθηκε, άρα δεν περιέχει έγκυρη διεύθυνση· η αποαναφορά του δίνει συνήθως segmentation fault.

[97] Στη $1000 + 3 \cdot 8 = 1024$.

[98] `*(arr + 4)`.

[99] `sizeof(a)` είναι 400, `sizeof(ptr)` είναι 8.

[100] Η βασική περίπτωση τερματισμού (base case) και η αναδρομική περίπτωση (recursive case).

[101] 4 φορές: `factorial(3)`, `(2)`, `(1)`, `(0)`, δηλαδή 3 αναδρομικές κλήσεις.

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A11.1–A11.13)

- **A11.1** Πρόσβαση σε μεταβλητή μόνο μέσω της διεύθυνσής της: Διάλεξη 11, διαφάνεια 20 · ★☆☆ · short-answer · `slides-lec11-access-by-address`
- **A11.2** Αναφορά σε στοιχεία πίνακα μέσω δείκτη: Διάλεξη 11, διαφάνειες 29–30 · ★☆☆ · trace · `slides-lec11-array-pointer-trace`
- **A11.3** Αναθέσεις μέσω δεικτών: Διάλεξη 11, διαφάνεια 24 · ★☆☆ · trace · `slides-lec11-assign-through-pointers`
- **A11.4** Μέσος όρος πίνακα 100 ακεραίων: Διάλεξη 11, διαφάνειες 37–38 · ★☆☆ · programming · `slides-lec11-average`
- **A11.5** Το παραγοντικό αναδρομικά σε C: Διάλεξη 11, διαφάνειες 44–45 · ★☆☆ · programming · `slides-lec11-factorial`
- **A11.6** Πόσες αναδρομικές κλήσεις κάνει το factorial: Διάλεξη 11, διαφάνεια 46 · ★☆☆ · short-answer · `slides-lec11-factorial-calls`
- **A11.7** Αρνητικό παραγοντικό: Διάλεξη 11, διαφάνεια 46 · ★☆☆ · short-answer · `slides-lec11-factorial-overflow`
- **A11.8** Θέση στοιχείου σε πίνακα ή -1: Διάλεξη 11, διαφάνειες 39–40 · ★☆☆ · programming · `slides-lec11-find`
- **A11.9** Το μέγεθος δεικτών διαφορετικών τύπων: Διάλεξη 11, διαφάνεια 18 · ★☆☆ · trace · `slides-lec11-pointer-sizes`
- **A11.10** Πώς τυπώνω μόνο το World: Διάλεξη 11, διαφάνειες 34–35 · ★☆☆ · programming · `slides-lec11-print-world`
- **A11.11** Η δική μας atoi: Διάλεξη 11, διαφάνειες 41–42 · ★★☆ · programming · `slides-lec11-atoi`
- **A11.12** Τι μπορεί να πάει στραβά με την atoi: Διάλεξη 11, διαφάνεια 42 · ★★☆ · debug · `slides-lec11-atoi-pitfalls`
- **A11.13** Αρνητικό όρισμα στο αναδρομικό παραγοντικό: Διάλεξη 11, διαφάνεια 46 · ★★☆ · short-answer · `slides-lec11-factorial-negative`

Εργαστήριο (A11.14–A11.18)

- **A11.14** Πέρασμα δεδομένων μέσω δεικτών: Εργαστήριο 6, Άσκηση 1 · ★☆☆ · programming · `lab-lab06-myprog`
- **A11.15** Η εικασία του Collatz: Εργαστήριο 5, Άσκηση 1 · ★★☆ · programming · `lab-lab05-collatz`
- **A11.16** Η ακολουθία Fibonacci: Εργαστήριο 5, Άσκηση 2 · ★★☆ · programming · `lab-lab05-fib`
- **A11.17** Πίνακες και αριθμητική δεικτών: Εργαστήριο 6, Άσκηση 3 · ★★☆ · trace · `lab-lab06-pointers`
- **A11.18** Υπολογισμός μισθών: Εργαστήριο 8, Άσκηση 4 · ★★☆ · debug · `lab-lab08-wages`

Εργασίες (A11.19–A11.21)

- **A11.19** Ο Αλγόριθμος του Ευκλείδη (gcd): Εργασία 1 (2024-25), Άσκηση 1 · ★★☆ · programming · `hw-2024-hw1-gcd`
- **A11.20** Άψογα Τετράγωνα (Bonus): Εργασία 1 (2023-24), Άσκηση 3 (Bonus) · ★★★ · programming · `hw-2023-hw1-flawless`
- **A11.21** Ο Αλγόριθμος RSA (rsa): Εργασία 1 (2024-25), Άσκηση 2 · ★★★ · programming · `hw-2024-hw1-rsa`

Θέματα εξετάσεων (A11.22)

- **A11.22** Ο Στέργιος Ξαναχτυπά: Εξέταση Ιανουαρίου 2026, Θέμα 6 · ★☆☆ · trace · `exam-2026-jan-q6`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A10.15** Πίνακες και συναρτήσεις: Εργαστήριο 6, Άσκηση 4 · ★☆☆ · programming · `lab-lab06-judgement`
- **A10.14** scanf χωρίς &: Διάλεξη 10, διαφάνεια 11 · ★★☆ · debug · `slides-lec10-scanf-no-ampersand`
- **A12.7** Περιεχόμενα του x μετά από βρόχο με δείκτη: Διάλεξη 12, διαφάνεια 25 · ★★☆ · trace · `slides-lec12-pointer-copy-loop`
- **A13.2** Γιατί η αναδρομή πρέπει να τελειώνει;: Διάλεξη 13, διαφάνεια 27 · ★☆☆ · short-answer · `slides-lec13-infinite-recursion`
- **A14.10** Το Δικό σου Chatbot (jason): Εργασία 2 (2024-25), Άσκηση 3 · ★★★ · programming · `hw-2024-hw2-jason`
- **A15.10** Πολυπλοκότητα του αναδρομικού παραγοντικού: Διάλεξη 15, διαφάνεια 25 · ★★☆ · short-answer · `slides-lec15-complexity-factorial`
- **A15.11** Πολυπλοκότητα του αναδρομικού Fibonacci: Διάλεξη 15, διαφάνεια 27 · ★★☆ · short-answer · `slides-lec15-complexity-fibonacci`
- **A16.18** Ο Γρίφος του Στέργιου: Bonus #0 (2025-26, προαιρετική) · ★★★ · programming · `hw-2025-bonus0-stergios`
- **A16.16** Σκαλί-σκαλί (Παλιό θέμα, Προαιρετικό): Εργαστήριο 5, Άσκηση 3 · ★★★ · programming · `lab-lab05-ladder`
- **A16.9** Η atoi και τι μπορεί να πάει στραβά: Διάλεξη 16, διαφάνειες 16–17 · ★★☆ · programming · `slides-lec16-atoi`
- **A16.11** Αποδοτική αναδρομική Fibonacci: Διάλεξη 16, διαφάνεια 26 · ★★☆ · programming · `slides-lec16-fibonacci-efficient`
- **A16.3** Η συνάρτηση get_two_chars: Διάλεξη 16, διαφάνεια 20 · ★☆☆ · programming · `slides-lec16-get-two-chars`
- **A16.7** Η συνάρτηση swap: Διάλεξη 16, διαφάνειες 22–23 · ★☆☆ · programming · `slides-lec16-swap`
- **A22.19** Reverse Inorder Traversal: Εξέταση Ιουλίου 2024, Θέμα 4 · ★★☆ · programming ·

`exam-2024-jul-q4`

- **A22.14** Δυαδικά δένδρα: Εργαστήριο 9, Άσκηση 4 · ★★☆ · programming · `lab-lab09-tree`
- **A25.5** Σκαλί-Σκαλί: Κατατακτήριες Δεκεμβρίου 2023, Θέμα 3 · ★★☆ · programming · `exam-2023-dec-q3`
- **A25.7** Ανεβαίνοντας Επίπεδο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #2 (Pokémon Themed), Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex2-q4`
- **A25.12** Λύσε τον Λαβύρινθο: Κατατακτήριες Δεκεμβρίου 2024, Θέμα 4 · ★★★ · programming · `exam-2024-dec-q4`
- **A25.14** Γεμίζοντας με χρώμα: Εξέταση Σεπτεμβρίου 2024, Θέμα 6 · ★★★ · programming · `exam-2024-sep-q6`
- **A25.4** Ανελκυστήρες για Ανυπόμονους και Ανυπόμονες (elevate): Εργασία 2 (2025-26), Άσκηση 1 · ★★★ · programming · `hw-2025-hw2-elevate`

Κεφάλαιο 12: Δείκτες και Πίνακες

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να χρησιμοποιείτε πολυδιάστατους πίνακες και να υπολογίζετε πού βρίσκεται στη μνήμη το `a[i][j]`· να χειρίζεστε δείκτες σε δείκτες, πίνακες από δείκτες και το `argv`· να δεσμεύετε δυναμικούς πίνακες με `malloc`· και να εξηγείτε τι είναι το endianness.

Προαπαιτούμενα: Κεφάλαιο 10, Κεφάλαιο 11

Χρόνος μελέτης: ~2,5 ώρες

Σύνοψη

Η διάλεξη πηγαίνει τους πίνακες και τους δείκτες ένα βήμα πιο πέρα. Οι δισδιάστατοι πίνακες της C είναι πίνακες από πίνακες, αποθηκευμένοι γραμμή-γραμμή σε συνεχόμενη μνήμη, οπότε η θέση του `a[i][j]` βγαίνει με έναν απλό τύπο. Οι δείκτες σε δείκτες και οι πίνακες από δείκτες (με κορυφαίο παράδειγμα το `argv`) προσθέτουν ένα επίπεδο έμμεσης αναφοράς. Η `malloc` δίνει πίνακες με μέγεθος που αποφασίζεται κατά την εκτέλεση, και το endianness εξηγεί τη σειρά των bytes ενός ακεραίου. Όλα αυτά στηρίζουν τις δομές δεδομένων του δεύτερου μισού του μαθήματος.

Θεωρία

§12.1 Ο πίνακας στη μνήμη

Ένας **πίνακας (array)** κρατά δεδομένα ίδιου τύπου (Κεφάλαιο 10). Στη δήλωση `int bears[100];` ο **τύπος (type)** ορίζει πόση μνήμη παίρνει κάθε στοιχείο, το **όνομα (name)** κάνει τον μεταγλωττιστή να διαλέξει μια διεύθυνση για τον πίνακα, και το **μέγεθος (size)** πόσα στοιχεία θα κρατήσει. Το μέγεθος είναι στατικό: δεν αλλάζει κατά την εκτέλεση. Στα στοιχεία αναφερόμαστε με τη **θέση (index)** τους, `bears[0]` έως `bears[99]`, και αυτά κάθονται σε συνεχόμενες θέσεις. Με `sizeof(int) == 4` και αρχή στη διεύθυνση 4:

Bytes	0–3	4–7	8–11	…	400–403
Περιεχόμενο	(άλλο)	`bears[0]`	`bears[1]`	…	`bears[99]`

Ο πίνακας πιάνει $4 \cdot 100 = 400$ bytes (4–403), και το `bears[i]` βρίσκεται στη διεύθυνση «αρχή + `i * sizeof(int)`».

§12.2 Δισδιάστατοι πίνακες

Εικόνες, επιστημονικά και οικονομικά δεδομένα ή μια σκακιέρα έχουν φυσικά περισσότερες **διαστάσεις (dimensions)**. Ένας **δισδιάστατος πίνακας (two-dimensional array)** είναι πίνακας από (υπο)πίνακες: τύπος όνομα`[γραμμές][στήλες];`. Οι **γραμμές (rows)** είναι πόσους υποπίνακες έχει (1η διάσταση) και οι **στήλες (columns)** πόσα στοιχεία έχει ο καθένας (2η διάσταση), άρα συνολικά γραμμές × στήλες στοιχεία. Το `a[i][j]` είναι το στοιχείο της γραμμής `i` και της στήλης `j`, με μέτρηση από το 0, και χρησιμοποιείται όπως μια απλή μεταβλητή. Η αρχικοποίηση δίνει μία λίστα σε αγκύλες ανά γραμμή (εδώ η γραμμή 0 είναι 1, 4, 7, 10 και η γραμμή 1 είναι 3, 6, 9, 12):

```
int array[2][4] = {
    {1, 4, 7, 10},
    {3, 6, 9, 12},
};
```

Σε αρχικοποίηση μπορεί να παραλειφθεί μόνο το μέγεθος της **πρώτης** διάστασης (`char arr[][2] = {{'a','b'},{'c','d'}};`, από τις σημειώσεις)· ο λόγος φαίνεται στον υπολογισμό διευθύνσεων παρακάτω.

§12.3 Αποθήκευση κατά γραμμές και sizeof

Η μνήμη είναι μονοδιάστατη, γι' αυτό η C αποθηκεύει τον πίνακα **κατά γραμμές (row-major order)**: όλη η γραμμή 0, αμέσως μετά η γραμμή 1. Για τον παραπάνω `array`, με αρχή στο 100 (κάθε στοιχείο πιάνει 4 bytes):

Bytes	100	104	108	112	116	120	124	128
Τιμή	1	4	7	10	3	6	9	12
Στοιχείο	`[0][0]`	`[0][1]`	`[0][2]`	`[0][3]`	`[1][0]`	`[1][1]`	`[1][2]`	`[1][3]`

Αφού ο πίνακας είναι πίνακας από πίνακες, το `array[0]` είναι από μόνο του ένας πίνακας 4 ακεραίων (bytes 100–115) και το `array[1]` ο επόμενος (116–131). Γι' αυτό `sizeof(array[0])` και `sizeof(array[1])` τυπώνουν 16 και `sizeof(array)` 32 (οι διαφάνειες τυπώνουν με `%d`· για `size_t` το σωστό είναι `%zu`), και `sizeof(array) / sizeof(array[0])` δίνει το πλήθος των γραμμών.

§12.4 Υπολογισμός διεύθυνσης στοιχείου

Για `int array[X][Y];`, για να φτάσουμε στο `a[i][j]` προσπερνάμε `i` γραμμές των `Y` ακεραίων και μετά `j` ακεραίους:

```
&a[i][j] = StartAddressOfArray + i * Y * sizeof(int) + j * sizeof(int)
IndexOfElementIJ = i * Y + j
```

Ο δεύτερος είναι η θέση σε έναν νοητό μονοδιάστατο πίνακα `X * Y` ακεραίων. Και οι δύο χρειάζονται **μόνο τον αριθμό `Y` των στηλών**. Γι' αυτό η πρώτη διάσταση μπορεί να παραλειφθεί, και μια συνάρτηση που δέχεται δισδιάστατο πίνακα γράφει την παράμετρο π.χ. `int matr[][12]` (σημειώσεις, «Πολυδιάστατοι πίνακες»). Έλεγχος με τον πίνακα παραπάνω: το `a[1][2]` είναι στο $100 + 1 \cdot 4 \cdot 4 + 2 \cdot 4 = 124$, όπου πράγματι βρίσκεται το 9.

§12.5 Πίνακες περισσότερων διαστάσεων

Πίνακες τριών ή περισσότερων διαστάσεων λειτουργούν με τον ίδιο τρόπο, με περισσότερα ζεύγη αγκυλών (π.χ. `int rubiksCube[6][3][3];`), και είναι πάλι συνεχόμενοι στη μνήμη: για `[X][Y][Z]` η θέση του `[i][j][k]` είναι `i * Y * Z + j * Z + k`. Η διάλεξη ρωτά: **είναι απαραίτητοι οι πολυδιάστατοι πίνακες;** Όχι· μπορούμε να κρατήσουμε τα ίδια δεδομένα σε μονοδιάστατο πίνακα και να υπολογίζουμε μόνοι μας τη θέση. Είναι όμως βολικοί, γιατί τον λογαριασμό τον κάνει ο μεταγλωττιστής.

§12.6 Δείκτες: υπενθύμιση

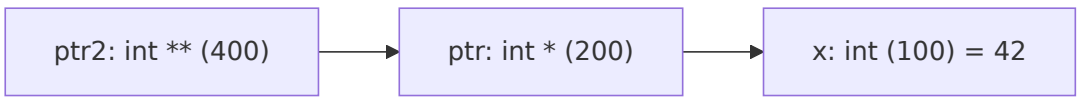
Ένας **δείκτης (pointer)** κρατά μια διεύθυνση μνήμης (Κεφάλαιο 11). Ο μοναδιαίος `*` κάνει **αποαναφορά (dereference)**: μετά το `int *pointer = &x;` το `*pointer` είναι ισοδύναμο με το `x`, για ανάγνωση και για εγγραφή (αν το `x` είναι στη διεύθυνση 100, ο `pointer` περιέχει την τιμή 100).

Στην **αριθμητική δεικτών (pointer arithmetic)**, αν ο `p` δείχνει σε `int`, ο `p + 2` δείχνει δύο ακεραίους (όχι bytes) πιο μετά, το `*(p + 2)` ισοδυναμεί με `p[2]` και το `p++` πάει στο επόμενο στοιχείο. Το όνομα ενός πίνακα σε έκφραση μετατρέπεται σε δείκτη στο πρώτο του στοιχείο, άρα το `p = x;` κάνει τον `p` να δείχνει στο `x[0]`.

§12.7 Δείκτης σε δείκτη

Ένας δείκτης έχει κι αυτός διεύθυνση, που μπορεί να αποθηκευτεί σε άλλον δείκτη. Ο **δείκτης σε δείκτη (pointer to pointer)** δηλώνεται με δύο αστερίσκους: μετά από `int x = 42; int *ptr = &x; int **ptr2 = &ptr;` ο `ptr2` είναι δείκτης σε `int *` (διαβάστε τον τύπο από δεξιά). Με τις διευθύνσεις της διαφάνειας:

Μεταβλητή	Τύπος	Διεύθυνση	Τιμή
`x`	`int`	100	42
`ptr`	`int *`	200	100
`ptr2`	`int **`	400	200



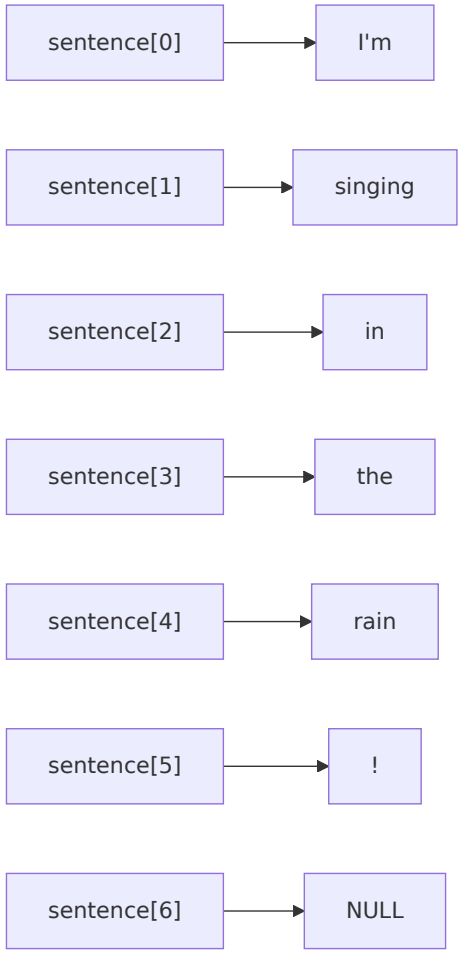
Σχήμα: κάθε `*` ακολουθεί ένα βέλος· `*ptr2` είναι ο `ptr` (100), `**ptr2` ο `x` (42).

Η διάλεξη ρωτά τι τυπώνει το `printf("%p %d", *ptr2, **ptr2);`: μια διεύθυνση (σε δεκαεξαδικό, διαφορετική σε κάθε εκτέλεση) και το 42. Το ίδιο γενικεύεται σε `int ***`. **Γιατί έχει σημασία**; Δέντρα και γράφοι είναι κόμβοι που δείχνουν σε άλλους κόμβους (Κεφάλαιο 21), και το `argv` και οι δυναμικοί δισδιάστατοι πίνακες έχουν τύπους `char **` και `int **`.

§12.8 Πίνακες από δείκτες

Οι δείκτες μπαίνουν και σε πίνακες: ο **πίνακας από δείκτες (array of pointers)** `int *ptr[3];` έχει τρία στοιχεία τύπου `int *`. Το `*ptr[2]` σημαίνει `*(ptr[2])`, γιατί οι αγκύλες έχουν μεγαλύτερη προτεραιότητα από τον μοναδιαίο `*`.

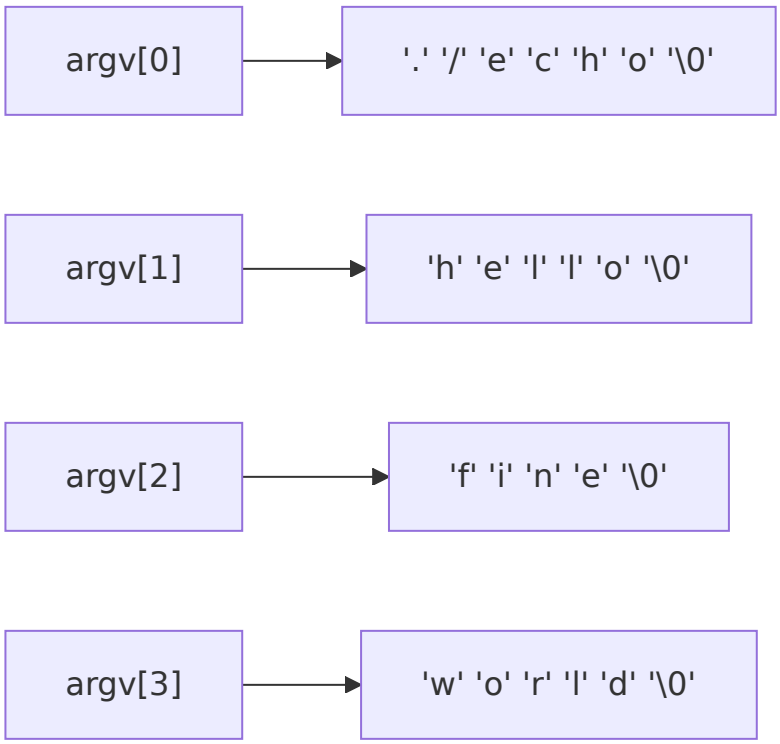
Η συνηθέστερη χρήση είναι ένας πίνακας συμβολοσειρών, `char *sentence[]`: κάθε στοιχείο δείχνει στον πρώτο χαρακτήρα μιας συμβολοσειράς, και σε αντίθεση με έναν δισδιάστατο πίνακα `char` οι συμβολοσειρές μπορούν να έχουν διαφορετικά μήκη. Συχνά το τελευταίο στοιχείο είναι ο **κενός δείκτης (null pointer)** `NULL`, που λειτουργεί ως «φρουρός»: ένας βρόχος με συνθήκη `sentence[i]` σταματά εκεί χωρίς να ξέρει το πλήθος.



Σχήμα: πίνακας από δείκτες σε συμβολοσειρές διαφορετικού μήκους, με `NULL` *στο τέλος.*

§12.9 Το argv

Το πιο γνωστό παράδειγμα πίνακα από δείκτες είναι η παράμετρος της `main` για τα **ορίσματα γραμμής εντολών (command-line arguments)**: `int main(int argc, char * argv[])`. Το `argc` είναι το πλήθος των ορισμάτων **μαζί με το όνομα του προγράμματος**, και το `argv[i]` δείχνει στη συμβολοσειρά του `i`-οστού ορίσματος. Για `./echo hello fine world` έχουμε `argc = 4`:



Σχήμα: το `argv` *για* `./echo hello fine world` *(*`argc = 4`*).*

Τα πραγματικά ορίσματα ξεκινούν από το `argv[1]`, και το `argv[argc]` είναι `NULL` (σημειώσεις, «Ορίσματα γραμμής εντολών»). Επειδή ένας πίνακας περνά σε συνάρτηση ως δείκτης στο πρώτο του στοιχείο, η παράμετρος γράφεται ισοδύναμα `char **argv`.

§12.10 Δυναμικοί πίνακες με malloc

Όταν το μέγεθος γίνεται γνωστό μόνο κατά την εκτέλεση (π.χ. διαβάζεται από την είσοδο), χρησιμοποιούμε **δυναμικούς πίνακες (dynamic arrays)**:

```
τύπος * όνομα = malloc(μέγεθος * sizeof(τύπος));
```

Η `malloc` (από το `stdlib.h`) δεσμεύει μνήμη και επιστρέφει τη διεύθυνσή της, που την κρατά ένας δείκτης στον τύπο των στοιχείων. Το όρισμα είναι σε **bytes**, γι' αυτό πολλαπλασιάζουμε το πλήθος με `sizeof(τύπος)`. Το `int * array = malloc(N * sizeof(int));` δημιουργεί πίνακα `N` ακεραίων, που χάρη στην αριθμητική δεικτών χρησιμοποιείται κανονικά, από `array[0]` έως `array[N - 1]`. Η μνήμη αυτή βρίσκεται στον **σωρό (heap)**, το θέμα του Κεφαλαίου 13.

Από τις σημειώσεις: η `malloc` δεν αρχικοποιεί τη μνήμη (οι τιμές 1, 3, 3, 7 της διαφάνειας είναι ενδεικτικές)· αν αποτύχει επιστρέφει `NULL`, οπότε ελέγχουμε πάντα το αποτέλεσμα· και ό,τι δεσμεύσαμε το αποδεσμεύουμε με `free`. Το `sizeof` ενός δείκτη είναι το μέγεθος μιας διεύθυνσης (8 bytes σε σύστημα 64 bit), όχι του μπλοκ, οπότε το πλήθος των στοιχείων το κρατάμε σε

μεταβλητή. Ένας δυναμικός δισδιάστατος πίνακας `N × M` φτιάχνεται με έναν `int **` που δείχνει σε `N` δείκτες γραμμών, με κάθε γραμμή από δική της `malloc`, ή με έναν μονοδιάστατο πίνακα `N * M` και τη θέση `i * M + j`.

§12.11 Endianness

Ένας `int` αποτελείται από πολλά bytes. **Endianness** λέμε τη σειρά με την οποία αποθηκεύονται: **little endian** από το μικρότερο (λιγότερο σημαντικό) byte προς το μεγαλύτερο, **big endian** αντίστροφα. Για τα διαδοχικά bytes `68 65 6c 6c` (ASCII `'h' 'e' 'l' 'l'`):

Μηχάνημα	byte +0	byte +1	byte +2	byte +3	Ακέραιος
little endian	`68`	`65`	`6c`	`6c`	`0x6c6c6568`
big endian	`68`	`65`	`6c`	`6c`	`0x68656c6c`

Οι x86 και οι περισσότεροι ARM είναι little endian (τα σχήματα μνήμης των διαφανειών για τους δείκτες είναι σχεδιασμένα big endian, που διαβάζεται ευκολότερα). Το endianness μετράει σε δυαδικά δεδομένα, π.χ. κεφαλίδες αρχείων εικόνας και ήχου.

Παραδείγματα

§12.12 Η σκακιέρα

Εφαρμόζει τους «Δισδιάστατους πίνακες»: κάθε κελί είναι ένας `char` (κεφαλαία για τον έναν παίκτη, πεζά για τον άλλο, κενό για άδειο τετράγωνο).

```c
#include <stdio.h>

int main() {
    char chessBoard[8][8] = {
        {'R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R'},
        {'P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'},
        {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
        {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
        {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
        {' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '},
        {'p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'},
        {'r', 'n', 'b', 'q', 'k', 'b', 'n', 'r'}
    };
    chessBoard[1][4] = ' ';
    chessBoard[3][4] = 'p';
    return 0;
}
```

Οι δύο αναθέσεις κάνουν την κίνηση e2–e4 (γραμμή 1 → γραμμή 3, στήλη 4). Με τη σύμβαση του πίνακα το πιόνι θα έπρεπε να είναι `'P'`· η διαφάνεια γράφει `'p'`. Για εκτύπωση, ο εξωτερικός βρόχος διατρέχει τις γραμμές και ο εσωτερικός τις στήλες.

§12.13 Ο κύβος του Rubik

Εφαρμόζει τους «Πίνακες περισσότερων διαστάσεων»: 6 έδρες × 3 × 3 = 54 ακέραιοι.

```
int rubiksCube[6][3][3] = {
    {{0, 0, 0}, {0, 0, 0}, {0, 0, 0}}, // Face 1 (e.g., Red)
    {{1, 1, 1}, {1, 1, 1}, {1, 1, 1}}, // Face 2 (e.g., Green)
    {{2, 2, 2}, {2, 2, 2}, {2, 2, 2}}, // Face 3 (e.g., Blue)
    {{3, 3, 3}, {3, 3, 3}, {3, 3, 3}}, // Face 4 (e.g., Yellow)
    {{4, 4, 4}, {4, 4, 4}, {4, 4, 4}}, // Face 5 (e.g., Orange)
    {{5, 5, 5}, {5, 5, 5}, {5, 5, 5}}  // Face 6 (e.g., White)
};
```

§12.14 Βρόχος με αριθμητική δεικτών

Εφαρμόζει τη «Δείκτες: υπενθύμιση». Ποια είναι τα περιεχόμενα του `x` μετά την εκτέλεση;

```
int * p;
int x[] = {5, 7, 2, 3, 6, 0, 1, 4};
p = x;
while (*p = *(p+2))
  p++;
```

Η συνθήκη είναι **ανάθεση**: αντιγράφει στο `*p` την τιμή δύο θέσεις πιο μετά, και αφού η τιμή μιας ανάθεσης είναι η τιμή που ανατέθηκε, ο βρόχος σταματά μόλις αντιγραφεί ένα 0. Ο πίνακας αλλάζει όσο τρέχει ο βρόχος, οπότε ιχνηλατήστε βήμα-βήμα. Για όσους δυσκολεύονται με τους δείκτες, η διάλεξη προτείνει το βίντεο των βοηθών (δείτε το Διάβασμα).

§12.15 Πίνακας από δείκτες σε ακεραίους

Εφαρμόζει τους «Πίνακες από δείκτες»:

```
#include <stdio.h>
int main() {
  int *ptr[3], a = 100, b = 200, c = 300;
  ptr[0] = &a;
  ptr[1] = &b;
  ptr[2] = &c;
  printf("%d %d %d\n", *ptr[2], *ptr[1], *ptr[0]);
  return 0;
}
```

```
$ ./example
300 200 100
```

§12.16 Μια πρόταση ως πίνακας από συμβολοσειρές

Εφαρμόζει τους «Πίνακες από δείκτες» με φρουρό `NULL`:

```c
#include <stdio.h>
int main() {
  char *sentence[] = {
      "I'm", "singing", "in", "the", "rain", "!", NULL
  };
  int i;
  for(i = 0 ; sentence[i]; i++) {
      printf("%s\n", sentence[i]);
  }
  return 0;
}
```

```
$ ./sentence
I'm
singing
in
the
rain
!
```

§12.17 Ένα απλό echo

Εφαρμόζει «Το argv»: τυπώνει όλα τα ορίσματα, μαζί με το όνομα του προγράμματος. Για ορίσματα-αριθμούς (π.χ. `argcalc.c` του εργαστηρίου 8) χρειάζεται η `atoi`.

```c
#include <stdio.h>
int main(int argc, char * argv[]) {
  int i;
  for(i = 0 ; i < argc ; i++) {
    printf("%s\n", argv[i]);
  }
  return 0;
}
```

```
$ ./echo hello fine world
./echo
hello
fine
```

```
world
```

§12.18 Πόση μνήμη δεσμεύει η malloc

Εφαρμόζει τους «Δυναμικούς πίνακες». Πόση μνήμη δεσμεύει κάθε κλήση και τι τυπώνει το `printf`;

```c
int * nums = malloc(100 * sizeof(int));
double * coeffs = malloc(100 * sizeof(double));
char * str = malloc(100 * sizeof(char));
printf("%zu %zu %zu\n", sizeof(nums), sizeof(coeffs), sizeof(str));
```

```
$ ./dynamic
8 8 8
```

Κάθε κλήση δεσμεύει `100 * sizeof(τύπος)` bytes, αλλά το `printf` τυπώνει το μέγεθος των ίδιων των δεικτών: 8 bytes ο καθένας σε σύστημα 64 bit. Για την άσκηση `array.c` του εργαστηρίου 7: διαβάστε το `N`, δεσμεύστε, ελέγξτε για `NULL`, και στο τέλος `free`.

§12.19 Βλέποντας τα bytes ενός ακεραίου

Εφαρμόζει το «Endianness». Μετατρέπουμε (cast) τη διεύθυνση του `x` σε `char *`, που προχωρά ένα byte τη φορά. Τι θα τυπώσει;

```c
#include <stdio.h>
int main() {
  int x = 42;
  char * bytes = (char*)&x;
  int i;
  for(i = 0; i < sizeof(int) / sizeof(char); i++)
     printf("%02x\n", bytes[i]);
  return 0;
}
```

```
$ ./int
2a
00
00
00
```

Το 42 είναι `0x0000002a`· πρώτο τυπώνεται το λιγότερο σημαντικό byte, άρα το μηχάνημα είναι little endian. Επειδή το `char` είναι συνήθως προσημασμένο, ένα byte `0x80` θα έβγαινε `ffffff80` με `%02x`· με `unsigned char *` αυτό δεν συμβαίνει.

Κύρια σημεία

1. Ένας δισδιάστατος πίνακας είναι πίνακας από πίνακες, τύπος `όνομα[γραμμές][στήλες]`, αρχικοποιείται με μία λίστα ανά γραμμή, και το `a[i][j]` είναι μια απλή μεταβλητή.
2. Αποθηκεύεται κατά γραμμές σε συνεχόμενη μνήμη· `sizeof(array[0])` είναι μία γραμμή, `sizeof(array)` όλος ο πίνακας.
3. Για `int array[X][Y]` η θέση του `a[i][j]` είναι `i * Y + j`: χρειάζεται μόνο το `Y`, γι' αυτό μόνο η πρώτη διάσταση μπορεί να παραλειφθεί.
4. Πίνακες περισσότερων διαστάσεων λειτουργούν ίδια· δεν είναι απαραίτητοι, αλλά απλοποιούν τον κώδικα.
5. Το `*pointer` ισοδυναμεί με τη μεταβλητή όπου δείχνει ο δείκτης· ένας `int **` κρατά τη διεύθυνση ενός δείκτη, και κάθε `*` ακολουθεί ένα επίπεδο.
6. Οι δείκτες μπαίνουν σε πίνακες· ένας πίνακας `char *` με `NULL` στο τέλος κρατά συμβολοσειρές διαφορετικού μήκους.
7. Το `argv` είναι πίνακας από `char *` (ισοδύναμα `char **`) με `argc` στοιχεία, και το `argv[0]` είναι το όνομα του προγράμματος.
8. Το `malloc(μέγεθος * sizeof(τύπος))` φτιάχνει στον σωρό πίνακα με μέγεθος που αποφασίζεται κατά την εκτέλεση· το `sizeof` του δείκτη είναι 8, όχι το μέγεθος του μπλοκ.
9. Endianness είναι η σειρά των bytes ενός ακεραίου: στο little endian πρώτο είναι το λιγότερο σημαντικό· με έναν `char *` το διαπιστώνουμε.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
πίνακας / θέση	array / index	Στοιχεία ίδιου τύπου σε συνεχόμενη μνήμη / ο `i` στο `a[i]`.
δισδιάστατος πίνακας	two-dimensional array	Πίνακας από πίνακες, `a[γραμμές][στήλες]`.
αποθήκευση κατά γραμμές	row-major order	Οι γραμμές αποθηκεύονται η μία μετά την άλλη.
αποαναφορά	dereference	Πρόσβαση με `*` εκεί όπου δείχνει ένας δείκτης.
δείκτης σε δείκτη	pointer to pointer	Δείκτης που κρατά διεύθυνση δείκτη, π.χ. `int **`.
πίνακας από δείκτες	array of pointers	Πίνακας με στοιχεία τύπου δείκτη, π.χ. `char *s[5]`.
κενός δείκτης	null pointer (`NULL`)	Δείκτης που δεν δείχνει πουθενά· συχνά σημαδεύει το τέλος.
ορίσματα γραμμής εντολών	command-line arguments	Οι λέξεις της κλήσης, στα `argc`/`argv`.

Ελληνικά	English	Σύντομος ορισμός
δυναμικός πίνακας	dynamic array	Πίνακας με μέγεθος που αποφασίζεται κατά την εκτέλεση.
σωρός	heap	Η περιοχή μνήμης από την οποία δεσμεύει η `malloc`.
σειρά bytes	endianness	Η σειρά αποθήκευσης των bytes ενός ακεραίου.

Διάβασμα

- **Διαφάνειες**: Διάλεξη 12, σελ. 1–45: μονοδιάστατοι πίνακες 5–10· δισδιάστατοι 11–22· πολυδιάστατοι 23· δείκτες σε δείκτες 24–28· πίνακες από δείκτες και `argv` 29–35· `malloc` 36–40· endianness 41–43.
- **Σημειώσεις**: η διάλεξη ζητά να διαβάσετε τις σελ. 73–103:
  - Κεφάλαιο 5: «Δείκτες» (K04, σελ. 72–77), «Περί ανάγνωσης ακεραίων (και όχι μόνο)» (78–79), «Πίνακες» (80–85), «Ιστόγραμμα συχνοτήτων γραμμάτων στην είσοδο» (86–87).
  - Κεφάλαιο 6: «Δυναμική δέσμευση μνήμης» (K04, σελ. 88–92), «Συμβολοσειρές» (93–96), «Πίνακες δεικτών και δείκτες σε δείκτες» (97), «Ορίσματα γραμμής εντολών» (98–99), «Πολυδιάστατοι πίνακες» (100–102), «Αρχικοποίηση πινάκων» (103).
- **Εργαστήριο**: Εργαστήριο 6: `pointers.c`· Εργαστήριο 7: `twodim.c`, `array.c`, `mines.c`· Εργαστήριο 8: `argcalc.c`.
- **Άλλα**: Endianness (Wikipedia)· βίντεο «Εισαγωγή στους Pointers» από τους βοηθούς του μαθήματος· `man 3 malloc`.

Συχνά λάθη

- `a[i, j]` **αντί για** `a[i][j]`. Το κόμμα είναι τελεστής, άρα σημαίνει `a[j]`.
- **Μπέρδεμα γραμμών και στηλών**. Στο `int a[2][4]` το `a[3][1]` είναι εκτός ορίων (`Segmentation fault`): ο πρώτος δείκτης είναι η γραμμή.
- **Παράλειψη της δεύτερης διάστασης**. Το `int m[][] = {{1, 2}, {3, 4}};` δίνει `array type has incomplete element type`. Γράψτε `int m[][2]`.
- `sizeof` **δείκτη ως μέγεθος πίνακα** (8, όχι 400) ή `sizeof` **με** `%d` (`format '%d' expects argument of type 'int'`): κρατήστε το πλήθος σε μεταβλητή και τυπώνετε το `sizeof` με `%zu`.
- **Ξεχασμένο** `sizeof` **ή** `stdlib.h` **στη** `malloc`. Το `malloc(100)` χωρά μόνο 25 `int`· χωρίς `#include <stdlib.h>` ο `gcc` λέει `implicit declaration of function 'malloc'`.
- **Χωρίς έλεγχο για** `NULL` **ή αρχικοποίηση**. Η `malloc` μπορεί να αποτύχει και δεν μηδενίζει τη μνήμη: ελέγξτε `if (array == NULL)` και γράψτε πριν διαβάσετε.
- `argv[1]` **χωρίς έλεγχο του** `argc`. Χωρίς όρισμα το `argv[1]` είναι `NULL` και το

`atoi(argv[1])` δίνει `Segmentation fault`. Ελέγξτε `if (argc < 2)`· και θυμηθείτε ότι για `./prog a b c` το `argc` είναι 4.

- **Πίνακας από δείκτες χωρίς φρουρό** `NULL`. Το `for (i = 0; sentence[i]; i++)` βγαίνει έξω από τον πίνακα.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K12.9** Διεύθυνση στοιχείου δισδιάστατου πίνακα (33% σωστές): Το 29% επέλεξε `155`: μέτρησαν σωστά ότι προηγούνται 55 στοιχεία, αλλά ξέχασαν να τα πολλαπλασιάσουν με `sizeof(int)` για να τα κάνουν bytes.
- **K12.8** Χαρακτήρας από πίνακα συμβολοσειρών (38% σωστές): Το 23% επέλεξε `"fine"` και άλλο 22% `'e'`: οι πρώτοι σταμάτησαν στο `strings[1]` αγνοώντας τον δεύτερο δείκτη, οι δεύτεροι μέτρησαν τις θέσεις από το 1.
- **K12.7** sizeof μιας γραμμής (42% σωστές): Το 26% επέλεξε `1`, θεωρώντας ότι το `map[5]` είναι ένας χαρακτήρας. Στην πραγματικότητα το `map[5]` είναι ολόκληρη η 6η γραμμή, ένας πίνακας 10 `char`.
- **K12.6** Ο τύπος του argv (44% σωστές): Το 34% επέλεξε «Ένας δείκτης σε πίνακες από χαρακτήρες», διαβάζοντας τη δήλωση ανάποδα. Οι αγκύλες δένουν πιο ισχυρά από το `*`, άρα το `argv` είναι πρώτα πίνακας, και τα στοιχεία του είναι `char *`.

Ερωτήσεις κατανόησης

- **E12.1** Πόσα bytes έχει ο `double m[3][5];` (`sizeof(double) == 8`), και πού βρίσκεται το `a[2][3]` του `int a[10][20];` αν αυτός ξεκινά στο 1000;[102]
- **E12.2** Γιατί ο αριθμός των γραμμών δεν εμφανίζεται στον τύπο της διεύθυνσης του `a[i][j]`;[103]
- **E12.3** Για `./prog one two`, ποιο είναι το `argc` και τι περιέχει το `argv[0]`;[104]
- **E12.4** Τι δεσμεύει το `malloc(100 * sizeof(double))` και πόσο είναι το `sizeof` του δείκτη που το κρατά;[105]
- **E12.5** Με ποια σειρά βρίσκονται τα bytes του `0x12345678` σε little endian;[106]

Kahoot από το αμφιθέατρο (K12.1–K12.9)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

[102] 120 bytes· $1000 + 2 \cdot 20 \cdot 4 + 3 \cdot 4 = 1172$.

[103] Για να φτάσουμε στη γραμμή `i` προσπερνάμε `i` γραμμές μήκους `Y`.

[104] `argc` είναι 3 και το `argv[0]` δείχνει στο `"./prog"`.

[105] 800 bytes στον σωρό· ο δείκτης είναι 8 bytes σε σύστημα 64 bit.

[106] 78, 56, 34, 12.

- **K12.1** Διαστάσεις πίνακα: 93% σωστές απαντήσεις
- **K12.2** Στατικοί και δυναμικοί πίνακες: 92% σωστές απαντήσεις
- **K12.3** Πλήθος στοιχείων δισδιάστατου πίνακα: 81% σωστές απαντήσεις
- **K12.4** sizeof δισδιάστατου πίνακα: 68% σωστές απαντήσεις
- **K12.5** Το τελευταίο στοιχείο δισδιάστατου πίνακα: 67% σωστές απαντήσεις
- **K12.6** Ο τύπος του argv: 44% σωστές απαντήσεις
- **K12.7** sizeof μιας γραμμής: 42% σωστές απαντήσεις
- **K12.8** Χαρακτήρας από πίνακα συμβολοσειρών: 38% σωστές απαντήσεις
- **K12.9** Διεύθυνση στοιχείου δισδιάστατου πίνακα: 33% σωστές απαντήσεις

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A12.1–A12.7)

- **A12.1** Στοιχείο δισδιάστατου πίνακα: Διάλεξη 12, διαφάνεια 14 · ★☆☆ · trace · `slides-lec12-2d-element`
- **A12.2** Πίνακας από δείκτες: Διάλεξη 12, διαφάνεια 29 · ★☆☆ · trace · `slides-lec12-array-of-pointers`
- **A12.3** Δείκτης σε δείκτη: Διάλεξη 12, διαφάνεια 27 · ★☆☆ · trace · `slides-lec12-pointer-to-pointer`
- **A12.4** Παράδειγμα endianness: Διάλεξη 12, διαφάνεια 42 · ★★☆ · trace · `slides-lec12-endianness`
- **A12.5** Πόση μνήμη δεσμεύει η malloc και τι λέει το sizeof: Διάλεξη 12, διαφάνειες 39–40 · ★★☆ · short-answer · `slides-lec12-malloc-sizeof`
- **A12.6** Είναι απαραίτητοι οι πολυδιάστατοι πίνακες;: Διάλεξη 12, διαφάνεια 23 · ★★☆ · short-answer · `slides-lec12-multidim-necessary`
- **A12.7** Περιεχόμενα του x μετά από βρόχο με δείκτη: Διάλεξη 12, διαφάνεια 25 · ★★☆ · trace · `slides-lec12-pointer-copy-loop`

Εργαστήριο (A12.8–A12.11)

- **A12.8** Δυναμική δέσμευση μνήμης για μονοδιάστατο πίνακα: Εργαστήριο 7, Άσκηση 2 · ★☆☆ · programming · `lab-lab07-array`
- **A12.9** Ορίσματα γραμμής εντολής: Εργαστήριο 8, Άσκηση 2 · ★☆☆ · programming · `lab-lab08-argcalc`
- **A12.10** Δισδιάστατοι πίνακες: Εργαστήριο 7, Άσκηση 1 · ★★☆ · programming · `lab-lab07-twodim`
- **A12.11** Πετυχαίνοντας τον στόχο (Παλιό θέμα): Εργαστήριο 8, Άσκηση 3 · ★★☆ · programming · `lab-lab08-legolas`

Εργασίες (A12.12)

- **A12.12** FauxtoShop: περιστροφή εικόνας BMP: Εργασία 2 (2023-24), Άσκηση 1 · ★★★ · programming · `hw-2023-hw2-fauxtoshop`

Θέματα εξετάσεων (A12.13–A12.22)

- **A12.13** Αλλαγή Τέρματος: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #13, Θέμα 2 · ★☆☆ · programming · `exam-2023-fall-ex13-q2`
- **A12.14** Τουρνουά: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #2 (Pokémon Themed), Θέμα 2 · ★☆☆ · programming · `exam-2023-fall-ex2-q2`
- **A12.15** Μέση Τιμή Τυχαίων Μεταβλητών - mean: Εξέταση Σεπτεμβρίου 2025, Θέμα 3 · ★☆☆ · programming · `exam-2025-sep-q3`
- **A12.16** Περιστροφή Πίνακα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #1 (Coreutils Themed), Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex1-q3`
- **A12.17** Κινήσεις σε Πλέγμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #11, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex11-q3`
- **A12.18** Πολλαπλασιασμός Πινάκων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #8, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex8-q3`
- **A12.19** Πολύτιμοι Πίνακες: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #9, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex9-q3`
- **A12.20** Στατιστικές: Εξέταση Ιουλίου 2024, Θέμα 2 · ★★☆ · programming · `exam-2024-jul-q2`
- **A12.21** Κινούμενος Μέσος Όρος - sma: Εξέταση Ιανουαρίου 2025, Θέμα 3 · ★★☆ · programming · `exam-2025-jan-q3`
- **A12.22** Το μεγαλύτερο άλμα - polevault: Εξέταση Σεπτεμβρίου 2026, Θέμα 3 · ★★☆ · programming · `exam-2026-sep-q3`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A4.13** Άρτια Bits: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #8, Θέμα 1 · ★☆☆ · programming · `exam-2023-fall-ex8-q1`
- **A6.24** Πετυχαίνοντας τον Στόχο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #9, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex9-q2`
- **A9.17** Επεξεργασία Ήχου (soundwave): Εργασία 1 (2025-26), Άσκηση 1 · ★★★ · programming · `hw-2025-hw1-soundwave`
- **A10.20** Τοποθέτηση του Pacman: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #11, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex11-q2`
- **A10.21** Φορτωμένο Έλκηθρο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #3 (Frozen Themed), Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex3-q2`
- **A10.23** Μαγικό Ζευγάρι: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #5 (HP Themed), Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex5-q2`
- **A11.19** Ο Αλγόριθμος του Ευκλείδη (gcd): Εργασία 1 (2024-25), Άσκηση 1 · ★★☆ ·

programming · hw-2024-hw1-gcd
- A11.21 Ο Αλγόριθμος RSA (rsa): Εργασία 1 (2024-25), Άσκηση 2 · ★★★ · programming · hw-2024-hw1-rsa
- A11.17 Πίνακες και αριθμητική δεικτών: Εργαστήριο 6, Άσκηση 3 · ★★☆ · trace · lab-lab06-pointers
- A13.8 Δυναμική δέσμευση μνήμης για δισδιάστατο πίνακα: Εργαστήριο 7, Άσκηση 3 · ★★☆ · programming · lab-lab07-mines
- A13.9 Κινήσεις σε πλέγμα (Παλιό θέμα): Εργαστήριο 7, Άσκηση 5 · ★★★ · programming · lab-lab07-pacman
- A13.4 Δυναμικός δισδιάστατος πίνακας MxN: Διάλεξη 13, διαφάνειες 56–58 · ★★☆ · programming · slides-lec13-dynamic-2d
- A14.13 Ταιριαστές Καρδιές: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #0 (Valentine's Themed), Θέμα 2 · ★★☆ · programming · exam-2023-fall-ex0-q2
- A14.14 Εντοπισμός Διπλών Ορισμάτων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #1 (Coreutils Themed), Θέμα 2 · ★★☆ · programming · exam-2023-fall-ex1-q2
- A14.22 Μεταμορφώσιμες Προτάσεις: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #5 (HP Themed), Θέμα 4 · ★★★ · programming · exam-2023-fall-ex5-q4
- A14.12 Η συνάρτηση transform: Εξέταση Σεπτεμβρίου 2025, Θέμα 2 · ★☆☆ · trace · exam-2025-sep-q2
- A14.9 Επεξεργασία συμβολοσειρών: Εργαστήριο 8, Άσκηση 1 · ★★☆ · programming · lab-lab08-string
- A14.1 Είναι το πρώτο όρισμα ”--boo”;: Διάλεξη 14, διαφάνεια 42 · ★☆☆ · programming · slides-lec14-check-boo
- A15.2 Πολυπλοκότητα εύρεσης μέγιστου σε πίνακα N x N: Διάλεξη 15, διαφάνεια 23 · ★☆☆ · short-answer · slides-lec15-complexity-find-max-2d
- A15.4 Πολυπλοκότητα δυναμικού πίνακα με malloc: Διάλεξη 15, διαφάνεια 19 · ★☆☆ · short-answer · slides-lec15-complexity-malloc
- A16.10 char *array[], char **array και char array[10][10]: Διάλεξη 16, διαφάνεια 19 · ★★☆ · short-answer · slides-lec16-char-pointer-arrays
- A16.6 yes αν ένα στοιχείο υπάρχει σε δισδιάστατο πίνακα: Διάλεξη 16, διαφάνεια 24 · ★☆☆ · programming · slides-lec16-search-2d
- A17.11 Πλησιάζοντας στον Στόχο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #10, Θέμα 4 · ★★☆ · programming · exam-2023-fall-ex10-q4
- A17.13 Η συνάρτηση compute: Εξέταση Ιανουαρίου 2026, Θέμα 2 · ★★☆ · trace · exam-2026-jan-q2
- A18.17 Αλλαγή Μεγέθους: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #4 (Rick Astley Themed), Θέμα 4 · ★★☆ · programming · exam-2023-fall-ex4-q4
- A18.13 Κόψιμο Αρχείων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #7 (Star Wars Themed), Θέμα 4 · ★☆☆ · programming · exam-2023-fall-ex7-q4
- A18.11 Προβλέποντας το Μέλλον (future): Εργασία 2 (2024-25), Άσκηση 1 · ★★☆ · programming · hw-2024-hw2-future
- A18.7 Μέτρηση στατιστικών αρχείων: Εργαστήριο 10, Άσκηση 4 · ★☆☆ · programming ·

`lab-lab10-count`

- **A18.8** Σύγκριση αρχείων: Εργαστήριο 10, Άσκηση 3 · ★☆☆ · programming · `lab-lab10-filediff`
- **A18.10** Αρχεία κειμένου: Εργαστήριο 10, Άσκηση 1 · ★★☆ · programming · `lab-lab10-more`
- **A18.1** Μέγεθος δισδιάστατου πίνακα και μιας γραμμής του: Διάλεξη 18, διαφάνεια 2 · ★☆☆ · short-answer · `slides-lec18-2d-sizeof`
- **A18.5** Ακέραιοι από αρχείο κειμένου με fread: Διάλεξη 18, διαφάνειες 50–51 · ★★☆ · trace · `slides-lec18-fread-int`
- **A18.6** Μια λέξη σε char[7] με scanf: Διάλεξη 18, διαφάνειες 31–35 · ★★☆ · debug · `slides-lec18-scanf-string`
- **A22.15** Zoomba: συντομότερη διαδρομή σε δωμάτιο: Εργασία 3 (2023-24), Άσκηση 1 · ★★★ · programming · `hw-2023-hw3-zoomba`
- **A25.12** Λύσε τον Λαβύρινθο: Κατατακτήριες Δεκεμβρίου 2024, Θέμα 4 · ★★★ · programming · `exam-2024-dec-q4`
- **A25.14** Γεμίζοντας με χρώμα: Εξέταση Σεπτεμβρίου 2024, Θέμα 6 · ★★★ · programming · `exam-2024-sep-q6`
- **A25.15** Το Καλό το Μονοπάτι - path: Εξέταση Σεπτεμβρίου 2025, Θέμα 5 · ★★★ · programming · `exam-2025-sep-q5`
- **A25.16** Περικύκλωση - encirclement: Εξέταση Ιανουαρίου 2026, Θέμα 5 · ★★★ · programming · `exam-2026-jan-q5`

Κεφάλαιο 13: Μνήμη

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να εξηγείτε τι είναι το endianness και να το διαπιστώνετε με έναν `char *`· να ξεχωρίζετε τη στοίβα από τον σωρό και να λέτε τι αποθηκεύεται στην καθεμία· να σχεδιάζετε τα stack frames μιας σειράς κλήσεων συναρτήσεων· να εξηγείτε γιατί η ατέρμονη αναδρομή και οι τεράστιοι τοπικοί πίνακες δίνουν `Segmentation fault`· και να δεσμεύετε, να μεγαλώνετε και να αποδεσμεύετε μνήμη στον σωρό με `malloc`, `calloc`, `realloc` και `free`, και για δισδιάστατους πίνακες.

Προαπαιτούμενα: Κεφάλαιο 3, Κεφάλαιο 11, Κεφάλαιο 12

Χρόνος μελέτης: ~3 ώρες

Σύνοψη

Μέχρι τώρα οι μεταβλητές μας «απλώς υπήρχαν»: τις δηλώναμε και ο μεταγλωττιστής φρόντιζε για τη μνήμη τους. Η διάλεξη αυτή ανοίγει το καπό και δείχνει πού ακριβώς ζουν τα δεδομένα ενός προγράμματος C. Ξεκινά με το endianness, δηλαδή τη σειρά των bytes ενός ακεραίου, και περνά στις κατηγορίες μνήμης: τη στοίβα (stack), όπου κάθε κλήση συνάρτησης παίρνει το δικό της stack frame, και τον σωρό (heap), από τον οποίο δεσμεύουμε μνήμη όσο τρέχει το πρόγραμμα. Βλέπουμε γιατί η στοίβα έχει όριο (και γιατί μια αναδρομή χωρίς τέλος «σκάει»), και μαθαίνουμε όλο τον κύκλο ζωής της δυναμικής μνήμης: `malloc`/`calloc`, έλεγχος για `NULL`, `realloc` και `free`. Η διαχείριση μνήμης είναι η βάση για τις δυναμικές δομές δεδομένων (λίστες, δέντρα) του δεύτερου μισού του μαθήματος και απαραίτητη για την Εργασία #1 και την εξέταση.

Θεωρία

§13.1 Endianness

Ένας `int` πιάνει πολλά bytes (συνήθως 4), οπότε πρέπει να αποφασιστεί με ποια σειρά μπαίνουν στη μνήμη. **Endianness** λέμε αυτόν τον τρόπο αποθήκευσης: στο **little endian** τα bytes αποθηκεύονται από το μικρότερο (λιγότερο σημαντικό) προς το μεγαλύτερο, στο **big endian** από το μεγαλύτερο προς το μικρότερο.

Αν στη μνήμη υπάρχουν διαδοχικά τα bytes `68 65 6c 6c` (ASCII `'h' 'e' 'l' 'l'`) και τα διαβάσουμε ως έναν ακέραιο:

Μηχάνημα	byte +0	byte +1	byte +2	byte +3	Ακέραιος
little endian	`68`	`65`	`6c`	`6c`	`0x6c6c6568`
big endian	`68`	`65`	`6c`	`6c`	`0x68656c6c`

Στο little endian το πρώτο byte στη μνήμη είναι το μικρότερο byte του αριθμού. Οι υπολογιστές x86 που χρησιμοποιούμε είναι little endian· το επιβεβαιώνουμε στο παράδειγμα «Βλέποντας τα bytes του 42». Το θέμα το είδατε πρώτη φορά στο Κεφάλαιο 12.

§13.2 Η μνήμη οργανώνεται σε bytes

Υπενθύμιση: η μνήμη είναι μια σειρά από N bytes αριθμημένα από 0 έως $N - 1$, και κάθε byte έχει 8 bits. Το μέγεθός της μετράμε σε bytes: 1 KB (KiloByte) = 1.000 bytes, 1 MB (MegaByte) = 1.000.000 bytes, 1 GB (GigaByte) = 1.000.000.000 bytes. Ο αριθμός ενός byte είναι η **διεύθυνσή** του (address), αυτή που κρατά ένας pointer.

§13.3 Κατηγορίες μνήμης

Ένα πρόγραμμα C χρησιμοποιεί τρεις κατηγορίες μνήμης:

1. τη **στοίβα** (stack), για τις τοπικές μεταβλητές των συναρτήσεων·
2. τον **σωρό** (heap), για τη μνήμη που ζητάμε ρητά με `malloc`·
3. την **παγκόσμια / στατική μνήμη** (global / static memory), για μεταβλητές που ζουν όσο όλο το πρόγραμμα.

Η διάλεξη καλύπτει τις δύο πρώτες και αφήνει την τρίτη για αργότερα. Σύμφωνα με τις σημειώσεις, εκεί φυλάσσονται οι εξωτερικές (καθολικές) μεταβλητές και οι τοπικές `static`, θέμα που συνδέεται με την εμβέλεια (Κεφάλαιο 14).

§13.4 Η στοίβα (stack)

Η **στοίβα** είναι μια **συνεχόμενη** περιοχή της μνήμης όπου προστίθενται και αφαιρούνται στοιχεία με σειρά **Last-In-First-Out (LIFO)**: το τελευταίο στοιχείο που προστέθηκε είναι το πρώτο που θα αφαιρεθεί, όπως σε μια στοίβα από πιάτα. Δεν μπορούμε να βγάλουμε ένα πιάτο από τη μέση.

Στο σχήμα των διαφανειών η στοίβα ξεκινά από το τέλος της μνήμης (byte 32000) και μεγαλώνει προς τις μικρότερες διευθύνσεις. Μετά το `char a = 61; char b = 62; char c = 63;` και, λίγο αργότερα, το `char d = 64;`:

Διεύθυνση	Περιεχόμενο	Μεταβλητή
31996	(ελεύθερο)	
31997	64	`d` (μπήκε τελευταία)

Διεύθυνση	Περιεχόμενο	Μεταβλητή
31998	63	c
31999	62	b
32000	61	a (μπήκε πρώτη)

Όταν τα d και c πάψουν να χρειάζονται, φεύγουν με την αντίστροφη σειρά: πρώτα το d, μετά το c, και μένουν τα a, b. Η στοίβα λοιπόν μεγαλώνει και μικραίνει μόνο από την κορυφή της, και γι' αυτό η διαχείρισή της είναι απλή και γρήγορη.

§13.5 Τι αποθηκεύεται στη στοίβα: το stack frame

Σε κάθε κλήση μιας συνάρτησης μπαίνουν στη στοίβα:

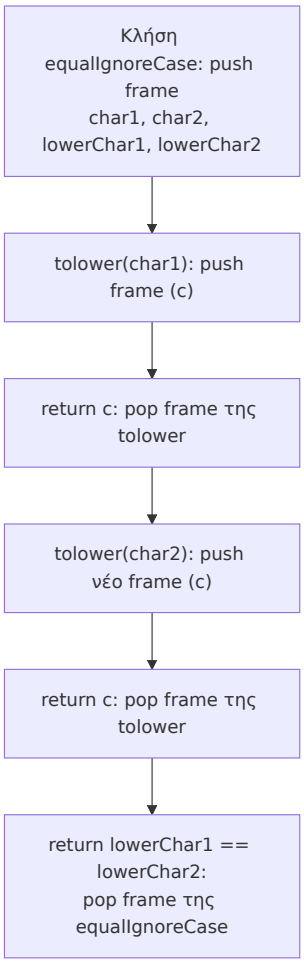
1. οι **τοπικές μεταβλητές** που ορίζονται μέσα στη συνάρτηση·
2. τα **ορίσματα** που περνάμε στην κλήση·
3. **προσωρινά δεδομένα** (συνήθως μερικά bytes) που αποθηκεύει ο μεταγλωττιστής για τη συγκεκριμένη κλήση.

Ο χώρος αυτός δεσμεύεται σε **κάθε** κλήση και λέγεται **διάγραμμα ενεργοποίησης** (activation record ή **stack frame**) της συνάρτησης. Για τη συνάρτηση

```
int equalIgnoreCase(char char1, char char2) {
    char lowerChar1 = tolower(char1);
    char lowerChar2 = tolower(char2);
    return lowerChar1 == lowerChar2;
}
```

το frame περιέχει (από κάτω προς τα πάνω) τα `char1`, `char2`, ένα προσωρινό `CompTmp1`, τα `lowerChar1`, `lowerChar2` και ένα προσωρινό `CompTmp2`. Τα ονόματα `CompTmp` είναι σχηματικά· το τι ακριβώς κρατά ο μεταγλωττιστής εκεί δεν μας απασχολεί.

Όταν η `equalIgnoreCase` καλεί την `tolower`, ένα **νέο frame** μπαίνει πάνω από το δικό της, με το όρισμα `c` και τα προσωρινά της (`CompTmp3`–`CompTmp5`). Όταν εκτελεστεί η `return` της `tolower`, το frame της αφαιρείται ολόκληρο, και η εκτέλεση συνεχίζει στην `equalIgnoreCase`. Η δεύτερη κλήση `tolower(char2)` φτιάχνει ξανά ένα καινούργιο frame στην ίδια θέση, και η `return` της `equalIgnoreCase` αφαιρεί και το δικό της. Έτσι οι τοπικές μεταβλητές μιας συνάρτησης υπάρχουν μόνο όσο εκτελείται η κλήση της.



Σχήμα: η ζωή των stack frames κατά την εκτέλεση της `equalIgnoreCase`.

§13.6 Γιατί η αναδρομή πρέπει να τελειώνει

Στο Κεφάλαιο 11 είπαμε ότι η αναδρομή πρέπει να έχει βάση. Τώρα φαίνεται το γιατί: κάθε αναδρομική κλήση προσθέτει ένα νέο frame στη στοίβα, και τα frames αφαιρούνται μόνο όταν οι κλήσεις επιστρέψουν. Μια αναδρομή χωρίς τέλος γεμίζει τη στοίβα μέχρι να ξεπεράσει το όριό της, και το λειτουργικό σύστημα τερματίζει το πρόγραμμα με `Segmentation fault` (αυτό λέγεται και stack overflow). Ισχύει και το αντίστροφο: επειδή κάθε κλήση έχει το δικό της frame, οι μεταβλητές διαφορετικών κλήσεων της ίδιας αναδρομικής συνάρτησης δεν μπερδεύονται.

§13.7 Το μέγεθος της στοίβας

Στα περισσότερα συστήματα η στοίβα περιορίζεται σε μερικά megabytes (MBs), επειδή δεν περιμένουμε εκατομμύρια εμφωλευμένες κλήσεις ή πολύ μεγάλα τοπικά δεδομένα. Το όριο το βλέπουμε με την εντολή `ulimit -s`:

```
$ ulimit -s
8192
```

Η τιμή είναι σε KB, δηλαδή 8 MB. Ένας τοπικός πίνακας μεγαλύτερος από αυτό (π.χ. `char bomb[9000000];`, 9 MB) δεν χωρά στη στοίβα και το πρόγραμμα κρασάρει πριν καν τυπώσει κάτι. Με `ulimit -s unlimited` το όριο αίρεται, αλλά είναι γενικά **κακή πρακτική** το πρόγραμμά μας να στηρίζεται σε unlimited stack: στον υπολογιστή που θα το τρέξει κάποιος άλλος (ή ο αυτόματος βαθμολογητής) το όριο θα είναι το συνηθισμένο. Για μεγάλους πίνακες η λύση είναι ο σωρός.

§13.8 Ο σωρός (heap)

Ο **σωρός** είναι ένα σύνολο από τοποθεσίες μνήμης σε τυχαία σειρά, σαν ένας σωρός ρούχα: τα κομμάτια του δεν δεσμεύονται και δεν ελευθερώνονται με σειρά LIFO, αλλά όποτε και όπου τα ζητήσουμε. Σε αντίθεση με τη στοίβα, ο σωρός μπορεί να δεσμεύσει **ολόκληρη τη διαθέσιμη μνήμη**, άρα εκεί χωρούν οι μεγάλοι πίνακες. Το τίμημα είναι ότι τη μνήμη του σωρού τη διαχειριζόμαστε εμείς: τη ζητάμε ρητά και την επιστρέφουμε ρητά.

	Στοίβα (stack)	Σωρός (heap)
Τι κρατά	τοπικές μεταβλητές, ορίσματα, προσωρινά	ό,τι δεσμεύουμε με `malloc`
Οργάνωση	συνεχόμενη, LIFO	τοποθεσίες σε τυχαία σειρά
Μέγεθος	λίγα MB (`ulimit -s`)	έως όλη τη διαθέσιμη μνήμη
Ποιος αποδεσμεύει	αυτόματα, με την `return`	εμείς, με `free`

§13.9 Δυναμικοί πίνακες με malloc

Με τη βοήθεια των pointers φτιάχνουμε **δυναμικούς** πίνακες: πίνακες των οποίων το μέγεθος καθορίζεται την ώρα που τρέχει το πρόγραμμα. Η γενική μορφή είναι

```
τύπος * όνομα = malloc(μέγεθος * sizeof(τύπος));
```

- `τύπος * όνομα`: δήλωση pointer σε δεδομένα τύπος, ο τύπος κάθε στοιχείου.
- `malloc` (από το `stdlib.h`): δεσμεύει μνήμη **στον σωρό** και επιστρέφει τη διεύθυνση όπου θα βάλουμε τα στοιχεία.
- `μέγεθος * sizeof(τύπος)`: ο αριθμός των **bytes** που πρέπει να δεσμευτούν για να χωρέσουν `μέγεθος` στοιχεία.

Μετά το `int * array = malloc(4 * sizeof(int));` ο pointer `array` ζει στη στοίβα (είναι τοπική μεταβλητή) και δείχνει σε 4 συνεχόμενους ακεραίους στον σωρό, τα `array[0]` έως `array[3]`. Με `N` αντί για 4 έχουμε πίνακα `N` ακεραίων, τα `array[0]` έως `array[N - 1]`.

Οι τιμές 1, 3, 3, 7 που δείχνουν οι διαφάνειες μέσα στο μπλοκ είναι ενδεικτικές: η `malloc` **δεν αρχικοποιεί** τη μνήμη. Και το `sizeof` του pointer δίνει το μέγεθος μιας διεύθυνσης (8 bytes

σε σύστημα 64 bit), όχι του μπλοκ, οπότε το πλήθος των στοιχείων το κρατάμε σε δική μας μεταβλητή.

§13.10 Οι τύποι void και void *

Ο τύπος `void` («κενό») δηλώνει το κενό σύνολο, έναν τύπο που δεν μπορεί να έχει τιμές. Μια συνάρτηση με τύπο επιστροφής `void` δεν επιστρέφει τίποτα (η `return;` απλώς τερματίζει). Δήλωση μεταβλητής `void a;` **δεν** επιτρέπεται.

Ο τύπος `void *` («δείκτης σε κενό») είναι ένας pointer σε «κάτι», δηλαδή απλώς μια διεύθυνση χωρίς πληροφορία για το τι βρίσκεται εκεί. Γι' αυτό τον επιστρέφουν οι συναρτήσεις δέσμευσης:

```
void *malloc(size_t size);
void *calloc(size_t nmemb, size_t size);
```

Το αποτέλεσμα ανατίθεται σε `int *`, `double *`, `char *`, ανάλογα με τη χρήση· όλοι οι pointers έχουν το ίδιο μέγεθος. Στη C η μετατροπή από `void *` γίνεται αυτόματα, οπότε cast (`(int *) malloc(...)`) δεν χρειάζεται. Η `calloc` κάνει ό,τι και η `malloc` για `nmemb` στοιχεία μεγέθους `size` το καθένα, αλλά επιπλέον **μηδενίζει** όλη τη μνήμη πριν επιστρέψει τον pointer.

§13.11 Ελέγχουμε ΠΑΝΤΑ το αποτέλεσμα της malloc

Αν δεν υπάρχει αρκετή μνήμη, η `malloc` επιστρέφει τον `NULL` pointer. Αν συνεχίσουμε σαν να μη συνέβη τίποτα, η πρώτη εγγραφή στον «πίνακα» γίνεται στη διεύθυνση `NULL` και το πρόγραμμα κρασάρει με `Segmentation fault`. Γι' αυτό μετά από κάθε `malloc` (και `calloc`, `realloc`) ελέγχουμε την τιμή επιστροφής:

```
int * array = malloc(4 * sizeof(int));
if (!array) {          // ισοδύναμα: if (array == NULL)
    fprintf(stderr, "array allocation failed\n");
    exit(1);
}
```

Το μήνυμα λάθους πάει στο `stderr` και η `exit(1)` (από το `stdlib.h`) τερματίζει το πρόγραμμα με κωδικό αποτυχίας. Εναλλακτικά, η `perror("array allocation")` τυπώνει στο `stderr` το μήνυμά μας μαζί με την περιγραφή του λάθους του συστήματος. Σε μια συνάρτηση που δεν θέλει να τερματίσει όλο το πρόγραμμα, ο χειρισμός («fail gracefully») μπορεί να είναι μια `return` με κωδικό λάθους.

§13.12 Αποδέσμευση μνήμης με free

Η μνήμη που δεσμεύτηκε με `malloc`/`calloc` απελευθερώνεται με τη συνάρτηση `free` (πάλι από το `stdlib.h`):

```
void free(void *ptr);
```

Ως μόνο όρισμα παίρνει τον pointer στη μνήμη που δεσμεύτηκε αρχικά. Φροντίζουμε **κάθε κλήση `malloc` να συνοδεύεται από μία `free`**. Αν δεν απελευθερώσουμε μνήμη που δεν χρειαζόμαστε πια, έχουμε **διαρροή μνήμης** (memory leak): το πρόγραμμα κρατά όλο και περισσότερη μνήμη που δεν μπορεί να ξαναχρησιμοποιήσει.

Μετά το `free(array);` τα 4 κελιά του σωρού επιστρέφουν στο σύστημα, αλλά ο pointer `array` στη στοίβα **εξακολουθεί να κρατά την ίδια διεύθυνση**: δείχνει πλέον σε μνήμη που δεν είναι δική μας (dangling pointer). Δύο πράγματα **απαγορεύονται**:

- **χρήση μετά την αποδέσμευση** (use after free), π.χ. `array[0] = 4;` μετά το `free(array);`·
- **διπλή αποδέσμευση** (double free), π.χ. δεύτερο `free(array);`.

Και στις δύο περιπτώσεις, στην καλύτερη το πρόγραμμα θα κρασάρει, και στη χειρότερη μπορεί να έχουμε **security vulnerability**, δηλαδή κενό ασφαλείας που εκμεταλλεύεται κάποιος επιτιθέμενος. Το χειρότερο είναι ότι συχνά το πρόγραμμα φαίνεται να δουλεύει.

§13.13 Αλλαγή μεγέθους με realloc

Όσο τρέχει το πρόγραμμα μπορεί να χρειαστούμε περισσότερη (ή λιγότερη) μνήμη. Με την `realloc` προσπαθούμε να αλλάξουμε το μέγεθος ενός δυναμικού πίνακα:

```
array = realloc(array, 8192 * sizeof(int));   // από 4 σε 8192 ακεραίους
if (!array) {...}
```

- Τα **περιεχόμενα διατηρούνται**: τα 1, 3, 3, 7 είναι ακόμη στις θέσεις 0–3.
- Η `realloc` **μπορεί να αλλάξει τη διεύθυνση** του μπλοκ μέσα στον σωρό (αν δεν χωρά να μεγαλώσει εκεί που είναι, το αντιγράφει αλλού). Γι' αυτό αναθέτουμε το αποτέλεσμα ξανά στον pointer και δεν κρατάμε αντίγραφα της παλιάς διεύθυνσης.
- Ισχύει ο ίδιος έλεγχος: **πάντα** ελέγχουμε για `NULL` αποτέλεσμα.

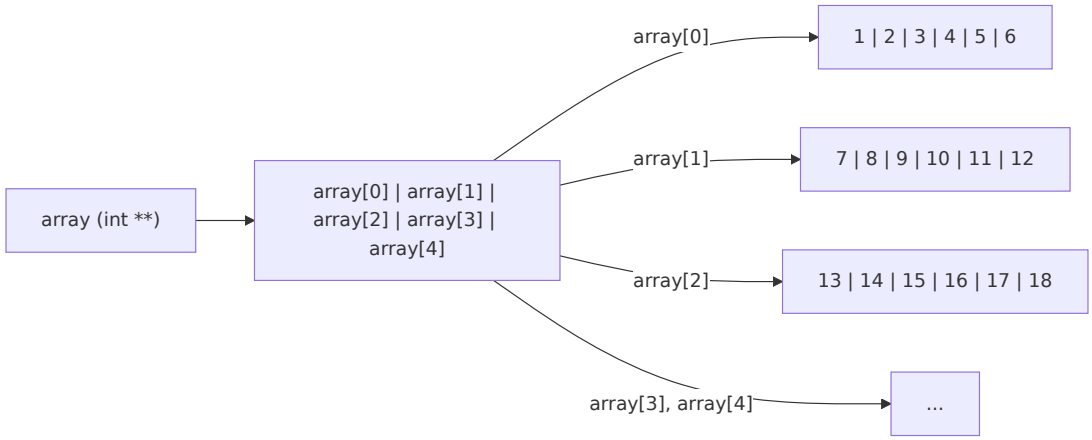
Στο τέλος, μία `free(array);` αποδεσμεύει το (νέο) μπλοκ.

§13.14 Δυναμικοί δισδιάστατοι πίνακες

Για έναν πίνακα ακεραίων `M × N` με διαστάσεις γνωστές μόνο την ώρα της εκτέλεσης, δεσμεύουμε πρώτα έναν πίνακα από `M` pointers (`int *`), έναν για κάθε γραμμή, και μετά, για κάθε γραμμή, έναν πίνακα από `N` ακεραίους:

```
int ** array = malloc(M * sizeof(int*));
for (int i = 0; i < M; i++)
    array[i] = malloc(N * sizeof(int));   // + έλεγχος για NULL σε κάθε κλήση
```

Το `array` είναι `int **` (δείκτης σε δείκτη, Κεφάλαιο 12). Το `array[i]` είναι ο pointer της γραμμής `i` και το `array[i][j]` το στοιχείο της στήλης `j` σε αυτήν, ακριβώς όπως σε έναν στατικό δισδιάστατο πίνακα. Η διαφορά είναι ότι οι γραμμές **δεν** είναι πια συνεχόμενες στη μνήμη: η καθεμία είναι ένα ξεχωριστό μπλοκ στον σωρό.



Σχήμα: δυναμικός πίνακας με `M = 5`, `N = 6`· *το* `array[1][3]` *είναι 10 και το* `array[2][2]` *είναι 15.*

Η αποδέσμευση γίνεται με την **αντίστροφη** σειρά: **πρώτα** οι υποπίνακες (οι γραμμές) και **μετά** ο πίνακας των pointers. Αν κάνουμε πρώτα `free(array)`, δεν μπορούμε πια να διαβάσουμε τα `array[i]` για να τα αποδεσμεύσουμε (θα ήταν use after free).

```
for (int i = 0; i < M; i++)
    free(array[i]);
free(array);
```

Παραδείγματα

§13.15 Βλέποντας τα bytes του 42

Εφαρμόζει το «Endianness». Τι θα τυπώσει το παρακάτω;

```
#include <stdio.h>
int main() {
  int x = 42;
  char * bytes = (char*)&x;
  int i;
  for(i = 0; i < sizeof(int) / sizeof(char); i++)
     printf("%02x\n", bytes[i]);
  return 0;
}
```

```
$ ./int
2a
00
00
00
```

Ο pointer `bytes` δείχνει στο πρώτο byte του `x`, και επειδή `sizeof(char) == 1` προχωρά ένα byte τη φορά. Το 42 είναι `0x0000002a`· πρώτο στη μνήμη βρίσκεται το λιγότερο σημαντικό byte `2a`, άρα ο υπολογιστής είναι little endian (σε big endian θα βλέπαμε `00 00 00 2a`).

§13.16 Τα frames της equalIgnoreCase

Εφαρμόζει το «Τι αποθηκεύεται στη στοίβα». Οι διαφάνειες γράφουν μόνες τους μια `tolower` (αντί για αυτή του `ctype.h`) για να φαίνεται και το frame της. Ακολουθήστε την εκτέλεση και σχεδιάστε τη στοίβα σε κάθε βήμα (το `gcc` ίσως προειδοποιήσει για σύγκρουση με την ενσωματωμένη `tolower`):

```c
#include <stdio.h>

char tolower(char c) {
    if (c >= 'A' && c <= 'Z')
        return c + ('a' - 'A');
    return c;
}

int equalIgnoreCase(char char1, char char2) {
    char lowerChar1 = tolower(char1);
    char lowerChar2 = tolower(char2);
    return lowerChar1 == lowerChar2;
}

int main() {
    printf("%d\n", equalIgnoreCase('Q', 'q'));
    return 0;
}
```

Το πρόγραμμα τυπώνει `1`. Η στοίβα σε κάθε βήμα είναι αυτή του σχήματος της «Θεωρίας».

§13.17 Ατέρμονη αναδρομή

Εφαρμόζει το «Γιατί η αναδρομή πρέπει να τελειώνει».

```c
void recurse() {
  recurse();
}

int main() {
  recurse();
  return 0;
}
```

```
$ gcc -o rec rec.c
$ ./rec
Segmentation fault
```

Κάθε κλήση της `recurse` βάζει ένα frame στη στοίβα και καμία δεν επιστρέφει, οπότε η στοίβα ξεπερνά τα 8 MB της.

§13.18 Ένας πίνακας-βόμβα στη στοίβα

Εφαρμόζει το «Μέγεθος της στοίβας». Τι θα κάνει το ακόλουθο πρόγραμμα;

```c
#include <stdio.h>

int main() {
  char bomb[9000000];
  printf("Hello World\n");
  return 0;
}
```

```
$ gcc -o hello hello.c
$ ulimit -s
8192
$ ./hello
Segmentation fault
$ ulimit -s unlimited
$ ulimit -s
unlimited
$ ./hello
Hello World
```

Ο τοπικός πίνακας των 9 MB δεν χωρά στα 8 MB της στοίβας, γι' αυτό δεν τυπώνεται καν το `Hello World`. Με unlimited stack δουλεύει, αλλά όπως είπαμε δεν στηριζόμαστε σε αυτό.

§13.19 Ο ίδιος πίνακας στον σωρό

Εφαρμόζει τον «Σωρό» και τον «Έλεγχο της malloc». Με `char *bomb = malloc(sizeof(char) * 9000000);` στη θέση του τοπικού πίνακα το πρόγραμμα τυπώνει `Hello World`: τα 9 MB στον σωρό δεν είναι πρόβλημα. Αν ζητήσουμε `900000000000L` bytes (900 GB, περισσότερα από τη μνήμη του υπολογιστή), **πάλι** τυπώνει `Hello World`: η `malloc` αποτυγχάνει ήσυχα και επιστρέφει `NULL`, που απλώς δεν το χρησιμοποιούμε. Μόλις όμως προσθέσουμε `bomb[0] = 'A';`, το πρόγραμμα γράφει στη διεύθυνση `NULL`:

```
$ ./heap
Segmentation fault
```

Η σωστή εκδοχή ελέγχει το αποτέλεσμα και βγαίνει με κατανοητό μήνυμα:

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
  char *bomb = malloc(sizeof(char) * 900000000000L);
  if (!bomb) {
    fprintf(stderr, "array allocation failed\n");
    exit(1);
  }
  bomb[0] = 'a';
  printf("Hello World\n");
  free(bomb);
  return 0;
}
```

(Οι διαφάνειες παραλείπουν το `free(bomb);`· εδώ το προσθέσαμε, αφού κάθε `malloc` συνοδεύεται από μία `free`.)

§13.20 Δυναμικός πίνακας M × N

Εφαρμόζει τους «Δυναμικούς δισδιάστατους πίνακες», με τις `perror` της σελ. 64. Γεμίζουμε τον πίνακα με 1, 2, 3, … όπως στο σχήμα και τυπώνουμε τα δύο στοιχεία που σημειώνει η διάλεξη:

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int M = 5, N = 6;
    int ** array = malloc(M * sizeof(int*));
    if (!array) {
        perror("array allocation");
        exit(1);
    }
    for (int i = 0; i < M; i++) {
        array[i] = malloc(N * sizeof(int));
        if (!array[i]) {
            perror("array[i] failed");
            exit(1);
        }
        for (int j = 0; j < N; j++)
            array[i][j] = i * N + j + 1;
    }
    printf("%d %d\n", array[1][3], array[2][2]);
    for (int i = 0; i < M; i++)
        free(array[i]);
```

```
    free(array);
    return 0;
}
```

```
$ ./dyn2d
10 15
```

Αυτό ακριβώς χρειάζεστε στην άσκηση `mines.c` του εργαστηρίου 7, όπου οι διαστάσεις διαβάζονται από την είσοδο. Στην τελική εξέταση θα χρειαστεί να διαχειριστείτε δεδομένα που το μέγεθός τους δεν ξέρετε εκ των προτέρων.

§13.21 Κουίζ: ένας πίνακας 5 × 5 μέσα σε συνάρτηση

Το κουίζ στο τέλος της διάλεξης δείχνει τρεις διαδοχικές εκδοχές μιας `void bar()` που φτιάχνει δυναμικά πίνακα 5 × 5 (δείτε την άσκηση «Κουίζ: πίνακας 5x5 σε συνάρτηση»). Η πρώτη δεν ελέγχει για `NULL` και δεν αποδεσμεύει τίποτα. Η δεύτερη ελέγχει κάθε `malloc` και κάνει `return;` σε αποτυχία. Η τρίτη προσθέτει στο τέλος τις `free` με τη σωστή σειρά. Ακόμη και η τρίτη έχει ένα κενό: αν αποτύχει η `malloc` της γραμμής 3, η `return;` αφήνει δεσμευμένα το `array` και τις γραμμές 0–2. Κάθε έξοδος από μια συνάρτηση πρέπει να αποδεσμεύει ό,τι έχει δεσμευτεί ως εκείνη τη στιγμή.

Κύρια σημεία

1. Endianness είναι η σειρά των bytes ενός ακεραίου στη μνήμη: στο little endian πρώτο είναι το λιγότερο σημαντικό byte, στο big endian το περισσότερο σημαντικό.
2. Ένα πρόγραμμα C έχει τρεις κατηγορίες μνήμης: τη στοίβα, τον σωρό και την παγκόσμια/στατική μνήμη.
3. Η στοίβα είναι συνεχόμενη περιοχή μνήμης με σειρά LIFO: ό,τι μπήκε τελευταίο βγαίνει πρώτο.
4. Σε κάθε κλήση συνάρτησης δεσμεύεται στη στοίβα ένα stack frame (activation record) με τις τοπικές μεταβλητές, τα ορίσματα και προσωρινά δεδομένα του μεταγλωττιστή.
5. Το frame αφαιρείται όταν η συνάρτηση επιστρέψει, γι' αυτό οι τοπικές μεταβλητές ζουν μόνο όσο εκτελείται η κλήση.
6. Η αναδρομή πρέπει να τελειώνει, γιατί κάθε κλήση προσθέτει ένα frame· μια ατέρμονη αναδρομή γεμίζει τη στοίβα και δίνει `Segmentation fault`.
7. Η στοίβα είναι λίγα MB (`ulimit -s`, συνήθως 8192 KB)· μεγάλοι τοπικοί πίνακες δεν χωρούν, και είναι κακή πρακτική να στηριζόμαστε σε `ulimit -s unlimited`.
8. Ο σωρός είναι σύνολο τοποθεσιών μνήμης σε τυχαία σειρά και μπορεί να δεσμεύσει όλη τη διαθέσιμη μνήμη.
9. Το `τύπος * όνομα = malloc(μέγεθος * sizeof(τύπος));` φτιάχνει στον σωρό πίνακα με μέγεθος που αποφασίζεται κατά την εκτέλεση· χρειάζεται `#include <stdlib.h>`.
10. Οι `malloc`/`calloc` επιστρέφουν `void *`, μια διεύθυνση που ανατίθεται σε pointer οποιουδήποτε τύπου· η `calloc` επιπλέον μηδενίζει τη μνήμη.

11. Ελέγχουμε ΠΑΝΤΑ το αποτέλεσμα των `malloc`/`calloc`/`realloc` για `NULL`, αλλιώς η πρώτη χρήση δίνει `Segmentation fault`.
12. Κάθε `malloc` συνοδεύεται από μία `free`· μνήμη που δεν αποδεσμεύεται είναι διαρροή μνήμης (memory leak).
13. Η χρήση μνήμης μετά την `free` και η διπλή `free` απαγορεύονται: στην καλύτερη κρασάρουν, στη χειρότερη ανοίγουν κενό ασφαλείας.
14. Η `realloc` αλλάζει το μέγεθος ενός δυναμικού πίνακα, διατηρεί τα περιεχόμενα και μπορεί να αλλάξει τη διεύθυνσή του.
15. Ένας δυναμικός πίνακας `M × N` είναι ένας `int **` προς `M` pointers γραμμών, με κάθε γραμμή από δική της `malloc`· αποδεσμεύουμε πρώτα τις γραμμές και μετά τον πίνακα των pointers.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
σειρά bytes	endianness	Η σειρά αποθήκευσης των bytes ενός ακεραίου.
στοίβα	stack	Συνεχόμενη μνήμη LIFO για τοπικές μεταβλητές και ορίσματα.
διάγραμμα ενεργοποίησης	activation record / stack frame	Ο χώρος στη στοίβα για μία κλήση συνάρτησης.
υπερχείλιση στοίβας	stack overflow	Η στοίβα ξεπερνά το όριό της, π.χ. από ατέρμονη αναδρομή.
σωρός	heap	Μνήμη που δεσμεύουμε και αποδεσμεύουμε ρητά.
παγκόσμια / στατική μνήμη	global / static memory	Μνήμη για μεταβλητές που ζουν όσο το πρόγραμμα.
δυναμικός πίνακας	dynamic array	Πίνακας με μέγεθος που αποφασίζεται κατά την εκτέλεση.
κενός τύπος	`void`	Τύπος χωρίς τιμές· π.χ. συνάρτηση που δεν επιστρέφει τίποτα.
δείκτης σε κενό	`void *`	Pointer σε «κάτι»: σκέτη διεύθυνση.
κενός δείκτης	null pointer (`NULL`)	Pointer που δεν δείχνει πουθενά· η `malloc` τον δίνει σε αποτυχία.
διαρροή μνήμης	memory leak	Μνήμη που δεσμεύτηκε και δεν αποδεσμεύτηκε ποτέ.
χρήση μετά την αποδέσμευση	use after free	Πρόσβαση σε μνήμη μετά την `free` της.
διπλή αποδέσμευση	double free	Δεύτερη `free` στον ίδιο pointer.

Διάβασμα

- **Διαφάνειες**: Διάλεξη 13, σελ. 1–64: endianness σελ. 5–7· κατηγορίες μνήμης σελ. 8–9· στοίβα και stack frames σελ. 10–26· αναδρομή και μέγεθος στοίβας σελ. 27–31· σωρός και `malloc` σελ. 32–40· `void` και `void *` σελ. 41–42· έλεγχος για `NULL` σελ. 43–46· `free` σελ. 47–51· `realloc` σελ. 52–55· δυναμικοί δισδιάστατοι πίνακες σελ. 56–58 και 64· κουίζ σελ. 61–63.
- **Σημειώσεις**:
  - Κεφάλαιο 6: Δυναμική μνήμη, συμβολοσειρές και πολυδιάστατοι πίνακες, ενότητες «Δυναμική δέσμευση μνήμης» (K04, σελ. 88–92) και «Πολυδιάστατοι πίνακες» (K04, σελ. 100–102, για τη δυναμική δέσμευση με `int **`).
  - Κεφάλαιο 4: Συναρτήσεις, εμβέλεια και αναδρομή, ενότητα «Εμβέλεια και χρόνος ζωής μεταβλητών» (K04, σελ. 63–68), για τη στοίβα, τη στατική μνήμη και την αναδρομή.
- **Εργαστήριο**: Εργαστήριο 7: ασκήσεις `array.c`, `mines.c`, και το παράρτημα «Σφάλματα διαχείρισης μνήμης» με την άσκηση `my_prog.c` (εντοπισμός σφάλματος μνήμης με τον `gdb`).
- **Άλλα** (οι σύνδεσμοι της σελ. 59):
  - Memory management and heap (Wikipedia).
  - Stack memory (Wikipedia).
  - Activation records (Coding Ninjas).
  - Dynamic memory allocation in C (Wikipedia).
  - Memory leaks (Wikipedia).
  - Data segment και bss section (Wikipedia· για άλλη φορά).
  - Endianness (Wikipedia).
  - `man 3 malloc` (καλύπτει `malloc`, `calloc`, `realloc`, `free`).

Συχνά λάθη

- **Μεγάλος τοπικός πίνακας**. Το `char bomb[9000000];` ή ένα `int grid[5000][5000];` μέσα σε συνάρτηση δίνει `Segmentation fault` πριν την πρώτη εντολή. Δεσμεύστε τον με `malloc`.
- **Αναδρομή χωρίς βάση ή με βάση που δεν φτάνεται.** Σύμπτωμα: `Segmentation fault` μετά από πολλές κλήσεις. Ελέγξτε ότι κάθε κλήση πλησιάζει τη βάση.
- **Χωρίς έλεγχο για** `NULL`. Όταν η `malloc` αποτύχει, το `bomb[0] = 'A';` κρασάρει. Γράφετε πάντα `if (!p) { ... }` αμέσως μετά.
- **Ξεχασμένο** `#include <stdlib.h>`. Ο `gcc` λέει `implicit declaration of function 'malloc'` (ή `'exit'`, `'free'`).
- `sizeof(int)` **αντί για** `sizeof(int*)` **στον πίνακα των γραμμών.** Σε σύστημα 64 bit ένας pointer είναι 8 bytes και ένας `int` 4, οπότε δεσμεύεται η μισή μνήμη. Για `int **` γράφετε `malloc(M * sizeof(int*))`.
- **Ανάγνωση μνήμης της** `malloc` **χωρίς αρχικοποίηση.** Τυπώνονται «σκουπίδια».

Αρχικοποιήστε, ή χρησιμοποιήστε `calloc` αν θέλετε μηδενικά.

- **Διαρροή μνήμης**. Μια `malloc` χωρίς αντίστοιχη `free`, ή μια πρόωρη `return` που προσπερνά τις `free` (όπως στο κουίζ). Ελέγξτε κάθε δρόμο εξόδου.
- **Χρήση μετά την `free` ή διπλή `free`**. Η glibc συχνά σταματά το πρόγραμμα με `free()`: `double free detected` ή `double free or corruption`· άλλες φορές το πρόγραμμα «δουλεύει» τυχαία. Μετά την `free` μην αγγίζετε τον pointer.
- **Λάθος σειρά αποδέσμευσης σε δισδιάστατο πίνακα**. Το `free(array);` πριν από τα `free(array[i]);` διαβάζει αποδεσμευμένη μνήμη. Πρώτα οι γραμμές, μετά ο πίνακας.
- **Παλιά διεύθυνση μετά τη `realloc`**. Ένας δεύτερος pointer που κρατούσε την παλιά διεύθυνση μπορεί να δείχνει πλέον σε αποδεσμευμένη μνήμη. Χρησιμοποιείτε μόνο ό,τι επέστρεψε η `realloc`. Επιπλέον, αν η `realloc` αποτύχει επιστρέφει `NULL` αλλά το παλιό μπλοκ μένει δεσμευμένο, οπότε το `array = realloc(array, ...)` το «χάνει»· σε πρόγραμμα που δεν τερματίζει αμέσως, κρατήστε το αποτέλεσμα πρώτα σε προσωρινό pointer.
- `void a;`. Δεν μεταγλωττίζεται (`variable or field 'a' declared void`): `void` δεν είναι τύπος μεταβλητής, μόνο `void *` είναι.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K13.11** Βάθος αναδρομής μέχρι να γεμίσει η στοίβα (20% σωστές): Το 48% επέλεξε `~8`, θεωρώντας ότι το 8192 είναι bytes. Το `ulimit -s` μετράει σε KB, άρα η στοίβα είναι 8 MB.
- **K13.10** malloc, calloc, realloc και free (27% σωστές): Περίπου οι μισοί (50%) συμπεριέλαβαν ότι αποδεσμεύουμε μνήμη της στοίβας με την `free`. Η `free` είναι μόνο για μνήμη του σωρού· οι τοπικές μεταβλητές αποδεσμεύονται αυτόματα όταν επιστρέφει η συνάρτηση.

Ερωτήσεις κατανόησης

- **E13.1** Τι σημαίνει LIFO και γιατί ταιριάζει στις κλήσεις συναρτήσεων;[107]
- **E13.2** Ποια τρία είδη δεδομένων μπαίνουν σε ένα stack frame;[108]
- **E13.3** Γιατί ο `char bomb[9000000];` κρασάρει ως τοπική μεταβλητή, ενώ ο ίδιος πίνακας με `malloc` όχι;[109]
- **E13.4** Τι είναι διαρροή μνήμης και πώς την αποφεύγουμε;[110]
- **E13.5** Μετά το `free(array);` τι τιμή έχει ο `array`;[111]

[107] Last-In-First-Out: βγαίνει πρώτο ό,τι μπήκε τελευταίο. Η συνάρτηση που κλήθηκε τελευταία είναι η πρώτη που επιστρέφει.

[108] Οι τοπικές μεταβλητές, τα ορίσματα και προσωρινά δεδομένα του μεταγλωττιστή.

[109] Η στοίβα είναι περιορισμένη (συνήθως 8 MB) ενώ ο σωρός μπορεί να δεσμεύσει όλη τη διαθέσιμη μνήμη.

[110] Μνήμη που δεσμεύτηκε και δεν αποδεσμεύτηκε· την αποφεύγουμε με μία `free` για κάθε `malloc`, σε κάθε δρόμο εξόδου.

[111] Την ίδια διεύθυνση με πριν, που όμως δεν είναι πια δική μας (dangling pointer).

- **E13.6** Γιατί μετά τη `realloc` αναθέτουμε το αποτέλεσμα ξανά στον pointer;[112]
- **E13.7** Πόσες κλήσεις `malloc` και πόσες `free` χρειάζεται ένας δυναμικός πίνακας `M × N` με `int **`;[113]

Kahoot από το αμφιθέατρο (K13.1–K13.11)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K13.1** Διπλό free: 92% σωστές απαντήσεις
- **K13.2** Κατηγορίες μνήμης: 87% σωστές απαντήσεις
- **K13.3** Αφαίρεση από στοίβα: 80% σωστές απαντήσεις
- **K13.4** Είναι συνεχόμενος ο σωρός;: 75% σωστές απαντήσεις
- **K13.5** Πάντα πετυχαίνει η malloc;: 70% σωστές απαντήσεις
- **K13.6** Αποδέσμευση από τον σωρό: 67% σωστές απαντήσεις
- **K13.7** Δέσμευση 10^9 double: 66% σωστές απαντήσεις
- **K13.8** Πού βάζω έναν μεγάλο πίνακα: 65% σωστές απαντήσεις
- **K13.9** malloc για πίνακα 10000 int: 60% σωστές απαντήσεις
- **K13.10** malloc, calloc, realloc και free: 27% σωστές απαντήσεις
- **K13.11** Βάθος αναδρομής μέχρι να γεμίσει η στοίβα: 20% σωστές απαντήσεις

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A13.1–A13.7)

- **A13.1** Τι γίνεται μετά την free;: Διάλεξη 13, διαφάνειες 47–51 · ★☆☆ · short-answer · `slides-lec13-after-free`
- **A13.2** Γιατί η αναδρομή πρέπει να τελειώνει;: Διάλεξη 13, διαφάνεια 27 · ★☆☆ · short-answer · `slides-lec13-infinite-recursion`
- **A13.3** Τα stack frames της equalIgnoreCase: Διάλεξη 13, διαφάνειες 20–26 · ★☆☆ · short-answer · `slides-lec13-stack-frames`
- **A13.4** Δυναμικός δισδιάστατος πίνακας MxN: Διάλεξη 13, διαφάνειες 56–58 · ★★☆ · programming · `slides-lec13-dynamic-2d`
- **A13.5** Ένας τεράστιος πίνακας στον σωρό: Διάλεξη 13, διαφάνειες 43–46 · ★★☆ · trace · `slides-lec13-heap-bomb`
- **A13.6** Κουίζ: πίνακας 5x5 σε συνάρτηση: Διάλεξη 13, διαφάνειες 61–63 · ★★☆ · debug · `slides-lec13-quiz-5x5`
- **A13.7** Ένας τεράστιος πίνακας στη στοίβα: Διάλεξη 13, διαφάνειες 28–31 · ★★☆ · trace · `slides-lec13-stack-bomb`

[112] Γιατί η `realloc` μπορεί να μετακινήσει το μπλοκ σε άλλη διεύθυνση.
[113] $M + 1$ κλήσεις `malloc` και $M + 1$ κλήσεις `free`.

Εργαστήριο (A13.8–A13.9)

- **A13.8** Δυναμική δέσμευση μνήμης για δισδιάστατο πίνακα: Εργαστήριο 7, Άσκηση 3 · ★★☆ · programming · `lab-lab07-mines`
- **A13.9** Κινήσεις σε πλέγμα (Παλιό θέμα): Εργαστήριο 7, Άσκηση 5 · ★★★ · programming · `lab-lab07-pacman`

Θέματα εξετάσεων (A13.10–A13.11)

- **A13.10** Debugging: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #1 (Coreutils Themed), Θέμα 4 · ★★☆ · debug · `exam-2023-fall-ex1-q4`
- **A13.11** Κάδρο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #9, Θέμα 4 · ★★★ · programming · `exam-2023-fall-ex9-q4`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A11.22** Ο Στέργιος Ξαναχτυπά: Εξέταση Ιανουαρίου 2026, Θέμα 6 · ★☆☆ · trace · `exam-2026-jan-q6`
- **A12.16** Περιστροφή Πίνακα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #1 (Coreutils Themed), Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex1-q3`
- **A12.13** Αλλαγή Τέρματος: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #13, Θέμα 2 · ★☆☆ · programming · `exam-2023-fall-ex13-q2`
- **A12.18** Πολλαπλασιασμός Πινάκων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #8, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex8-q3`
- **A12.19** Πολύτιμοι Πίνακες: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #9, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex9-q3`
- **A12.12** FauxtoShop: περιστροφή εικόνας BMP: Εργασία 2 (2023-24), Άσκηση 1 · ★★★ · programming · `hw-2023-hw2-fauxtoshop`
- **A12.8** Δυναμική δέσμευση μνήμης για μονοδιάστατο πίνακα: Εργαστήριο 7, Άσκηση 2 · ★☆☆ · programming · `lab-lab07-array`
- **A12.4** Παράδειγμα endianness: Διάλεξη 12, διαφάνεια 42 · ★★☆ · trace · `slides-lec12-endianness`
- **A12.5** Πόση μνήμη δεσμεύει η malloc και τι λέει το sizeof: Διάλεξη 12, διαφάνειες 39–40 · ★★☆ · short-answer · `slides-lec12-malloc-sizeof`
- **A14.11** Η συνάρτηση dog: Εξέταση Ιανουαρίου 2025, Θέμα 2 · ★☆☆ · trace · `exam-2025-jan-q2`
- **A14.21** Συνένωση Αλφαριθμητικών - join: Εξέταση Ιανουαρίου 2025, Θέμα 5 · ★★☆ · programming · `exam-2025-jan-q5`
- **A15.18** Δίδυμοι Πρώτοι: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #7 (Star Wars Themed), Θέμα 3 · ★★★ · programming · `exam-2023-fall-ex7-q3`
- **A16.26** Μετρώντας τα Αστέρια - stars: Εξέταση Ιανουαρίου 2025, Θέμα 6 · ★★★ · programming · `exam-2025-jan-q6`
- **A16.17** Χτίζοντας έναν χιονάνθρωπο (Παλιό θέμα): Εργαστήριο 7, Άσκηση 4 · ★★★ ·

programming · `lab-lab07-olaf`

- **A16.13** Δισδιάστατος πίνακας στον σωρό: Διάλεξη 16, διαφάνεια 25 · ★★☆ · programming · `slides-lec16-heap-2d-array`
- **A17.13** Η συνάρτηση compute: Εξέταση Ιανουαρίου 2026, Θέμα 2 · ★★☆ · trace · `exam-2026-jan-q2`
- **A18.20** Καλύτερο Ταίριασμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #0 (Valentine's Themed), Θέμα 3 · ★★★ · programming · `exam-2023-fall-ex0-q3`
- **A18.15** Κρυμμένο Μήνυμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #10, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex10-q3`
- **A18.22** Μίνι Βάση Δεδομένων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #2 (Pokémon Themed), Θέμα 3 · ★★★ · programming · `exam-2023-fall-ex2-q3`
- **A18.19** Ταξινόμηση Αρχείων Καταγραφής: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #6 (Crypto Themed), Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex6-q4`
- **A18.23** Ταξινόμηση Πακέτων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #8, Θέμα 4 · ★★★ · programming · `exam-2023-fall-ex8-q4`
- **A18.11** Προβλέποντας το Μέλλον (future): Εργασία 2 (2024-25), Άσκηση 1 · ★★☆ · programming · `hw-2024-hw2-future`
- **A19.10** Δομές και δείκτες: Εργαστήριο 9, Άσκηση 2 · ★★☆ · programming · `lab-lab09-person`
- **A21.8** Αντιστροφή λίστας: Εξέταση Σεπτεμβρίου 2024, Θέμα 5 · ★★☆ · programming · `exam-2024-sep-q5`
- **A21.9** Μεσαίο Στοιχείο Λίστας: Εξέταση Σεπτεμβρίου 2025, Θέμα 4 · ★★☆ · programming · `exam-2025-sep-q4`
- **A21.10** Ν-οστό Στοιχείο Λίστας: Εξέταση Σεπτεμβρίου 2026, Θέμα 4 · ★★☆ · programming · `exam-2026-sep-q4`
- **A22.15** Zoomba: συντομότερη διαδρομή σε δωμάτιο: Εργασία 3 (2023-24), Άσκηση 1 · ★★★ · programming · `hw-2023-hw3-zoomba`
- **A25.9** Το Νερό Νεράκι: Κατατακτήριες Δεκεμβρίου 2023, Θέμα 4 · ★★★ · programming · `exam-2023-dec-q4`
- **A25.11** Η Τριπλέτα Στόχος: Κατατακτήριες Δεκεμβρίου 2024, Θέμα 3 · ★★★ · programming · `exam-2024-dec-q3`
- **A25.17** Η Μεγαλύτερη Χωρητικότητα - capacity: Εξέταση Σεπτεμβρίου 2026, Θέμα 5 · ★★★ · programming · `exam-2026-sep-q5`

Κεφάλαιο 14: Εμβέλεια, Μνήμη και Συμβολοσειρές

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να βρίσκετε την εμβέλεια κάθε μεταβλητής, να ξεχωρίζετε τοπικές, παγκόσμιες και στατικές μεταβλητές και να λέτε σε ποια μνήμη ζει η καθεμία· να εξηγείτε την επισκίαση· και να χρησιμοποιείτε (και να υλοποιείτε) τις `strlen`, `strcmp`, `strcpy` και `strcat`, γνωρίζοντας τους κινδύνους τους.

Προαπαιτούμενα: Κεφάλαιο 10, Κεφάλαιο 12, Κεφάλαιο 13

Χρόνος μελέτης: ~2 ώρες

Σύνοψη

Η διάλεξη κλείνει την εικόνα της μνήμης ενός προγράμματος C. Ξεκινά από την εμβέλεια: από πού είναι ορατή μια μεταβλητή, ανάλογα με το πού δηλώθηκε, και τι γίνεται όταν δύο μεταβλητές έχουν το ίδιο όνομα. Έπειτα προσθέτει την τρίτη κατηγορία μνήμης, την παγκόσμια / στατική, όπου ζουν οι παγκόσμιες και οι `static` μεταβλητές και τα string literals, και δείχνει με το `nm` πώς τα καταγράφει ο μεταγλωττιστής. Τέλος, περνά στις συμβολοσειρές και στις βασικές συναρτήσεις της `string.h`, με πιθανές υλοποιήσεις τους. Είναι καθημερινά εργαλεία για κάθε πρόγραμμα που δουλεύει με κείμενο ή με ορίσματα γραμμής εντολών.

Θεωρία

§14.1 Εμβέλεια μεταβλητής

Εμβέλεια (scope) μιας μεταβλητής λέγεται το μέρος του προγράμματος όπου η μεταβλητή είναι ορατή / προσβάσιμη με το όνομά της. Την εμβέλεια την καθορίζει **το σημείο της δήλωσης**. Υπάρχουν δύο είδη μεταβλητών και εμβέλειας: οι **τοπικές μεταβλητές** (local variables), που δηλώνονται μέσα σε συνάρτηση, και οι **παγκόσμιες μεταβλητές** (global variables), που δηλώνονται έξω από κάθε συνάρτηση.

§14.2 Τοπικές μεταβλητές

Μια **τοπική μεταβλητή** δηλώνεται μέσα στο σώμα μιας συνάρτησης. Η εμβέλειά της αρχίζει **από το σημείο της δήλωσης** και φτάνει **ως το τέλος του block εντολών** (`{ … }`) όπου ορίστηκε: πριν από τη δήλωση το όνομα δεν υπάρχει ακόμη, μετά το `}` δεν υπάρχει πια. Οι τοπικές μεταβλητές αποθηκεύονται στη **στοίβα** (stack), στο frame της κλήσης (Κεφάλαιο 13).

Οι **παράμετροι** είναι κι αυτές τοπικές μεταβλητές, με εμβέλεια όλο το σώμα της συνάρτησης· αρχικοποιούνται κατά την κλήση με τις τιμές των ορισμάτων.

Αφού μια τοπική μεταβλητή δεν είναι ορατή σε άλλη συνάρτηση, **μπορούμε να χρησιμοποιούμε το ίδιο όνομα σε διαφορετικές συναρτήσεις**: κάθε δήλωση είναι άλλη μεταβλητή, σε άλλη θέση μνήμης. Για τον ίδιο λόγο, το `foo(i)` δίνει στη `foo` **αντίγραφο της τιμής** του `i`· ό,τι κι αν κάνει η `foo` στην παράμετρό της, το `i` του καλούντος δεν αλλάζει.

§14.3 Παγκόσμιες μεταβλητές

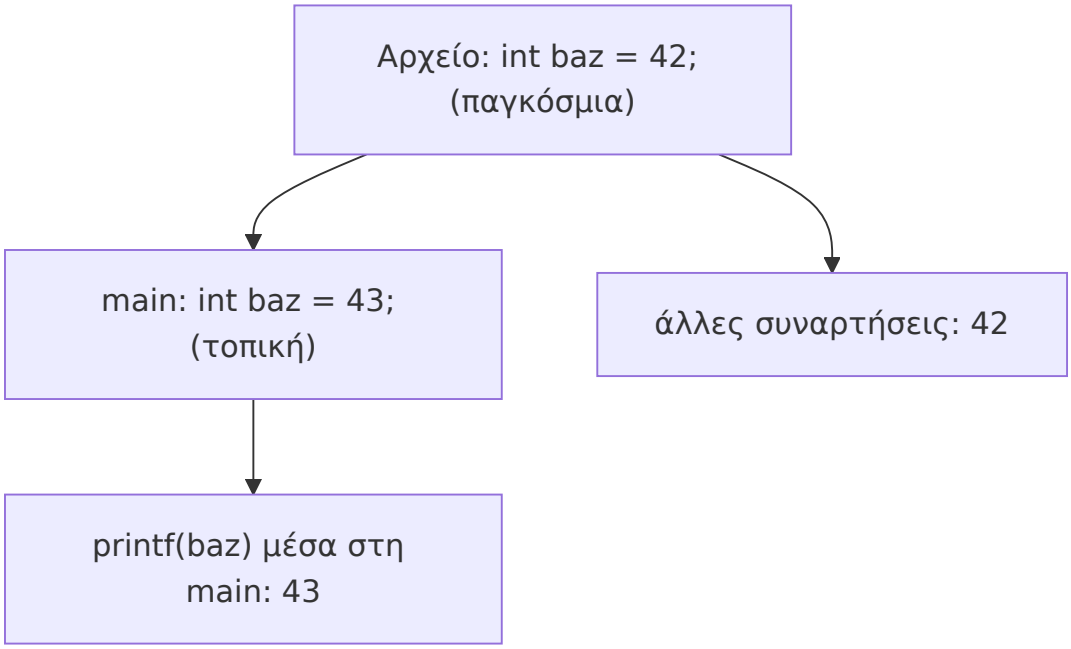
Μια **παγκόσμια μεταβλητή** δηλώνεται έξω από οποιαδήποτε συνάρτηση. Η εμβέλειά της είναι **από το σημείο της δήλωσης ως το τέλος του αρχείου**: κάθε συνάρτηση που ορίζεται πιο κάτω μπορεί να τη διαβάσει και να την αλλάξει.

```
int baz = 42;
void foo() { printf("baz: %d\n", baz); }   /* βλέπει την baz */
void bar() { baz++; }                       /* και αυτή */
```

Οι σημειώσεις προσθέτουν δύο πράγματα. Οι παγκόσμιες μεταβλητές είναι εύκολος δρόμος επικοινωνίας μεταξύ συναρτήσεων, αλλά η εκτεταμένη χρήση τους κάνει τα προγράμματα δυσανάγνωστα και τις συναρτήσεις λιγότερο γενικής χρήσης. Και οι μη αρχικοποιημένες παγκόσμιες ξεκινούν με 0, ενώ οι μη αρχικοποιημένες τοπικές έχουν απροσδιόριστη τιμή.

§14.4 Επισκίαση μεταβλητής

Όταν μια δήλωση μεταβλητής βρίσκεται **μέσα στην εμβέλεια άλλης μεταβλητής με το ίδιο όνομα**, η εσωτερική μεταβλητή **επισκιάζει** (shadows) την εξωτερική: μέσα στο εσωτερικό block το όνομα αναφέρεται στην εσωτερική. Η εξωτερική δεν αλλάζει και ξαναγίνεται ορατή όταν κλείσει το block. Το ίδιο ισχύει για εμφωλευμένα blocks μιας συνάρτησης: ισχύει πάντα η πιο «κοντινή» δήλωση.



Σχήμα: η τοπική `baz` *της* `main` *επισκιάζει την παγκόσμια μόνο μέσα στη* `main`.

Η επισκίαση είναι νόμιμη C αλλά συχνή πηγή λαθών: νομίζουμε ότι αλλάζουμε μια μεταβλητή και αλλάζουμε άλλη. Ο `gcc` προειδοποιεί γι' αυτήν με `-Wshadow`.

§14.5 Στατικές μεταβλητές

Μια μεταβλητή λέγεται **στατική** (static) όταν **διατηρεί την τιμή της ανάμεσα σε κλήσεις συναρτήσεων, μέχρι το τέλος του προγράμματος**. Τη δηλώνουμε μέσα σε συνάρτηση με τη λέξη `static`, π.χ. `static int counter = 0;`. Η εμβέλειά της είναι όπως μιας τοπικής (μόνο η συνάρτηση τη βλέπει), ο **χρόνος ζωής** της όμως είναι όλο το πρόγραμμα: δεν ζει στη στοίβα, οπότε δεν χάνεται όταν επιστρέφει η συνάρτηση. **Η αρχικοποίηση στη δήλωση γίνεται μόνο μία φορά**, πριν από την πρώτη κλήση· κάθε επόμενη κλήση βρίσκει την τιμή που άφησε η προηγούμενη.

Προσοχή: αν η ανάθεση γραφτεί **έξω** από τη δήλωση (`static int counter;` και μετά `counter = 0;`), είναι απλή εντολή που εκτελείται σε κάθε κλήση, και η τιμή δεν διατηρείται. Οι σημειώσεις αναφέρουν και μια δεύτερη σημασία: το `static` μπροστά σε παγκόσμια μεταβλητή περιορίζει την εμβέλειά της στο αρχείο όπου ορίζεται.

§14.6 Η παγκόσμια / στατική μνήμη

Ένα πρόγραμμα C έχει τρεις κατηγορίες μνήμης: τη **στοίβα** (stack), τον **σωρό** (heap) και την **παγκόσμια / στατική μνήμη** (global / static memory). Η τελευταία είναι ένα μέρος της μνήμης του προγράμματος, ξεχωριστό από τη στοίβα και τον σωρό, όπου αποθηκεύονται οι

παγκόσμιες μεταβλητές, οι στατικές μεταβλητές και δεδομένα του προγράμματος (ακόμη και ο ίδιος ο κώδικας). **Αρχικοποιείται όταν ξεκινά το πρόγραμμα και αποδεσμεύεται όταν τερματίσει**. Έχει υποκατηγορίες (data, code, `.bss`), που η διάλεξη αφήνει για άλλο μάθημα.

Κατηγορία	Τι κρατά	Πότε δεσμεύεται / αποδεσμεύεται
στοίβα	τοπικές μεταβλητές, παράμετροι	στην κλήση / στην επιστροφή
σωρός	ό,τι ζητάμε με `malloc`	με `malloc` / με `free`
παγκόσμια / στατική	παγκόσμιες, `static`, string literals, κώδικας	στην αρχή / στο τέλος του προγράμματος

§14.7 Σύμβολα, .bss και data

Κάθε αρχείο C μπορεί να ορίζει τις δικές του παγκόσμιες μεταβλητές, και ο μεταγλωττιστής **σώζει τα ονόματά τους ως σύμβολα** (symbols) στο αρχείο αντικειμένου (`.o`). Το εργαλείο `nm` τα εμφανίζει με ένα γράμμα για το είδος τους: `T` για κώδικα (συναρτήσεις), `B` για το `.bss` (μεταβλητές χωρίς αρχικοποίηση ή με 0), `D` για το data (μεταβλητές αρχικοποιημένες με μη μηδενικές τιμές). Το μικρό γράμμα σημαίνει σύμβολο τοπικό στο αρχείο: η `static int counter` της `foo` εμφανίζεται ως `b counter.0`.

Η διαφορά `.bss` και data φαίνεται στο μέγεθος του αρχείου. Για το `.bss` αρκεί να γραφτεί **πόσος χώρος χρειάζεται**· ο χώρος δίνεται γεμάτος μηδενικά όταν ξεκινά το πρόγραμμα. Αν όμως η μεταβλητή αρχικοποιηθεί με **μη μηδενικές τιμές**, ο μεταγλωττιστής αποθηκεύει **στο αρχείο** όλο τον χώρο και όλες τις τιμές. Έτσι ένας τεράστιος παγκόσμιος πίνακας χωρίς αρχικοποίηση δεν μεγαλώνει το `.o`, ενώ με `= {1}` το κάνει εκατοντάδες MB.

§14.8 String literals στη στατική μνήμη

Οι συμβολοσειρές σε εισαγωγικά (**string literals**) που **δεν** χρησιμοποιούνται για την αρχικοποίηση ενός τοπικού πίνακα αποθηκεύονται κι αυτές στη στατική μνήμη:

```
char *str1 = "Hello World";   /* δείκτης σε string της στατικής μνήμης */
char str2[] = "Hello World";  /* τοπικός πίνακας 12 char, στη στοίβα */
```

Ο `str1` είναι δείκτης που δείχνει σε ένα literal της στατικής μνήμης· δύο ίδια literals μπορεί να είναι **το ίδιο αντίγραφο**, στην ίδια διεύθυνση. Στον `str2` οι χαρακτήρες **αντιγράφονται** σε νέο πίνακα στη στοίβα, με δική του διεύθυνση. Οι σημειώσεις προσθέτουν ότι τα literals δεν επιτρέπεται να αλλάξουν: τους χαρακτήρες του `str2` μπορούμε να τους αλλάξουμε, αυτούς όπου δείχνει ο `str1` όχι.

§14.9 Συμβολοσειρές

Ένας πίνακας από χαρακτήρες λέγεται **αλφαριθμητικό** ή **συμβολοσειρά** (string). Οι τρεις δηλώσεις που είδαμε στο Κεφάλαιο 10 είναι ισοδύναμες:

```
char hello[] = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd',
                '\n', '\0'};
char hello[] = {72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100, 10, 0};
char hello[] = "Hello World\n";
```

Τα string **τερματίζονται πάντα με το null byte** `'\0'`. Η μορφή με τα εισαγωγικά το προσθέτει μόνη της, οπότε ο πίνακας έχει 13 θέσεις για 12 χαρακτήρες. Το null byte είναι ο μόνος τρόπος να ξέρει μια συνάρτηση πού τελειώνει ένα string: όλες οι συναρτήσεις παρακάτω **διατρέχουν τους χαρακτήρες μέχρι να βρουν** `'\0'`. Σε ένα string αναφερόμαστε και με δείκτη `char *` στον πρώτο χαρακτήρα του· έτσι περνά σε συναρτήσεις.

§14.10 Οι συναρτήσεις της string.h

Οι συναρτήσεις βιβλιοθήκης για συμβολοσειρές δηλώνονται στο `string.h` (`#include <string.h>`):

```
size_t strlen(const char *s);
int strcmp(const char *s1, const char *s2);
char *strcpy(char *dst, const char *src);
char *strcat(char *dst, const char *src);
```

- Η `strlen` επιστρέφει το πλήθος των χαρακτήρων **χωρίς να μετρά το null byte**: το `strlen("hello world")` είναι 11. Επιστρέφει `size_t` (μη προσημασμένο), που τυπώνεται με `%zu`. Το μήκος δεν είναι αποθηκευμένο πουθενά· η `strlen` διατρέχει όλο το string.
- Η `strcmp` ελέγχει αν δύο συμβολοσειρές A και B είναι ίσες. Αν είναι, επιστρέφει **0**· αλλιώς **θετική** τιμή αν A > B, **αρνητική** αν A < B. Συγκρίνει χαρακτήρα προς χαρακτήρα με βάση τους κωδικούς ASCII, ως την πρώτη διαφορά (λεξικογραφική σειρά). Εγγυημένο είναι μόνο το πρόσημο. Δύο string **δεν** συγκρίνονται με `==`: αυτό συγκρίνει διευθύνσεις.
- Η `strcpy` **αντιγράφει** το string του 2ου ορίσματος στο 1ο, μαζί με το null byte, και επιστρέφει τον προορισμό. Έτσι «αναθέτουμε» string, αφού οι πίνακες δεν ανατίθενται με `=`.
- Η `strcat` **συνενώνει** (concatenates) δύο συμβολοσειρές στην πρώτη: αρχίζει την αντιγραφή του 2ου ορίσματος **από το τέλος (το null byte) του πρώτου**.

Η `strcpy` είναι ιδιαίτερα επικίνδυνη (θέμα ασφάλειας!): γράφει στη μνήμη **χωρίς να ελέγχει αν υπάρχει αρκετός χώρος**. Αν ο προορισμός είναι μικρότερος, γράφει πάνω σε ό,τι ακολουθεί (buffer overflow). Τον έλεγχο πρέπει να τον κάνουμε εμείς, και ελαχιστοποιούμε τη χρήση της. Η `strncpy` (`man strncpy`) δέχεται και μέγιστο πλήθος χαρακτήρων: λίγο καλύτερη, αλλά πάλι κακή επιλογή, γιατί μπορεί να αφήσει τον προορισμό χωρίς null byte. **Η `strcat` έχει ακριβώς τα ίδια προβλήματα**: ελέγχουμε πάντα ότι ο πίνακας του 1ου ορίσματος χωρά και τα δύο string και το null byte.

Η διάλεξη αφήνει για μελέτη και τις `strdup`, `strchr` και `strtok` (δείτε «Διάβασμα»)· την `strtok` τη χρησιμοποιεί το Εργαστήριο 8.

§14.11 Η έκφραση *str++

Οι υλοποιήσεις των συναρτήσεων αυτών γράφονται συνήθως με εκφράσεις όπως το `*str++`. Ο μεταθεματικός (postfix) `++` έχει **μεγαλύτερη προτεραιότητα** από το `*`, οπότε το `*str++` σημαίνει `*(str++)`: δίνει τον χαρακτήρα όπου δείχνει ο `str` **τώρα**, και μετά **ο δείκτης** προχωρά στον επόμενο. Αντίθετα, το `(*str)++` αυξάνει **τον χαρακτήρα** και ο δείκτης μένει στη θέση του. Η προτεραιότητα των τελεστών (Κεφάλαιο 5) μετράει λοιπόν, και οι παρενθέσεις αλλάζουν το νόημα.

Έκφραση	Τιμή	Τι αλλάζει
`*str++`, `*(str++)`	ο τρέχων χαρακτήρας	ο δείκτης `str` προχωρά
`(*str)++`	ο τρέχων χαρακτήρας	ο χαρακτήρας αυξάνεται κατά 1

Παραδείγματα

§14.12 Η εμβέλεια μέσα στη foo

Εφαρμογή της «Τοπικές μεταβλητές».

```
void foo(int bar) {
  printf("%d\n", bar);
  int i = 42;
  printf("%d\n", i);
  int j;
  for (j = 0; j < i; j++)
    printf("%d %d\n", bar, i * j);
}
```

Η εμβέλεια του `bar` είναι όλο το σώμα (αρχικοποιείται κατά την κλήση)· του `i`, από το `int i = 42;` ως το `}` (στην πρώτη `printf` δεν υπάρχει ακόμη)· του `j`, από το `int j;` ως το `}`.

§14.13 Το ίδιο όνομα σε δύο συναρτήσεις

Εφαρμογή της «Τοπικές μεταβλητές». Τι επιστρέφει αυτό το πρόγραμμα;

```
void foo() {
  int i = 43;
}
int main() {
  int i = 42;
```

```
  foo();
  return i;
}
```

```
$ gcc -o local local.c
$ ./local
$ echo $?
42
```

Το `i` της `foo` είναι άλλη μεταβλητή, στο frame της `foo`· το 43 δεν αγγίζει το `i` της `main`. Το ίδιο ισχύει με όρισμα: στο `local2.c` η `foo` γίνεται `void foo(int i) { i++; }` και η `main` καλεί `foo(i);`. Το `./local2; echo $?` τυπώνει πάλι 42: η παράμετρος πήρε αντίγραφο του 42 και έγινε 43, το `i` της `main` όχι.

§14.14 Μια παγκόσμια και μια τοπική baz

Εφαρμογή της «Επισκίαση μεταβλητής». Τι θα τυπώσει;

```
#include <stdio.h>
int baz = 42;
int main() {
  int baz = 43;
  printf("baz: %d\n", baz);
  return 0;
}
```

```
$ ./shadow
baz: 43
```

Η παγκόσμια `baz` έχει εμβέλεια όλο το αρχείο, αλλά μέσα στη `main` την επισκιάζει η τοπική.

§14.15 Ένας μετρητής με static

Εφαρμογή της «Στατικές μεταβλητές». Τι θα τυπώσει;

```
#include <stdio.h>
void foo() {
  static int counter = 0; int i = 0;
  counter++; i++;
  printf("%d %d\n", counter, i);
}
int main() {
  foo(); foo(); foo();
  return 0;
}
```

```
$ ./static
1 1
2 1
3 1
```

Το `counter` αρχικοποιείται μία φορά και μετρά τις κλήσεις· το `i` ξαναγίνεται 0 σε κάθε κλήση. Αν η πρώτη γραμμή της `foo` γίνει `static int counter; int i = 0; counter = 0;`, το `counter = 0` εκτελείται σε κάθε κλήση και το πρόγραμμα τυπώνει `1 1` τρεις φορές.

§14.16 Παράδειγμα με κάθε μνήμη

Εφαρμογή της «Η παγκόσμια / στατική μνήμη».

```
#include <stdlib.h>
char global[3] = {42, 24, 55};
void foo(char c1, char c2) {
  char c3 = 63;
  char * tmp = malloc(3 * sizeof(char));
  tmp[0] = 42; tmp[1] = 255; tmp[2] = 255;
}
int main() {
  foo(61, 62);
  return 0;
}
```

Κατά την εκτέλεση της `foo` (οι διευθύνσεις είναι ενδεικτικές):

Περιοχή	Περιεχόμενο
παγκόσμια / στατική	`global`: 42, 24, 55 (bytes 1–3)
σωρός	το μπλοκ της `malloc`: 42, 255, 255
στοίβα	`c1` = 61 (byte 32000), `c2` = 62 (31999), `c3` = 63 (31998)

Ο `global` υπάρχει σε όλη την εκτέλεση. Τα `c1`, `c2`, `c3` (και ο ίδιος ο δείκτης `tmp`) ζουν στη στοίβα μόνο όσο τρέχει η `foo`. Το μπλοκ του σωρού μένει δεσμευμένο και μετά την επιστροφή, αφού δεν καλείται `free`, και όταν χαθεί ο `tmp` δεν μπορούμε πια να το ελευθερώσουμε.

§14.17 Ο πίνακας των 400 MB

Εφαρμογή της «Σύμβολα, .bss και data». Το `memory.c`:

```
int global_buffer[10];
void foo() {
  static int counter = 0;
  counter++;
```

```
}
int main() {
  foo();
  return 0;
}
```

```
$ gcc -c memory.c
$ nm memory.o
0000000000000028 b counter.0
0000000000000000 T foo
0000000000000000 B global_buffer
0000000000000016 T main
$ du -sh memory.o
4.0K    memory.o
```

Οι `foo` και `main` είναι κώδικας (`T`), ο `global_buffer` και ο `counter` είναι στο `.bss`. Με `int global_buffer[100000000];` (400.000.000 bytes) το `nm` δείχνει τον `counter` στο `0000000017d78400`, δηλαδή 400.000.000, αμέσως μετά τον πίνακα, αλλά το `memory.o` μένει **4.0K**: γράφεται μόνο το μέγεθος. Με `int global_buffer[100000000] = {1};` όμως:

```
$ gcc -c memory.c
$ nm memory.o
0000000000000000 b counter.0
0000000000000000 T foo
0000000000000000 D global_buffer
0000000000000016 T main
$ du -sh memory.o
382M    memory.o
```

Ο πίνακας πέρασε στο data (`D`) και ο μεταγλωττιστής έγραψε στο αρχείο όλα τα 400 MB (382 MiB). Οι Συχνές Ερωτήσεις του μαθήματος συνδέουν το σφάλμα `section size is larger than file size` με τέτοιους πίνακες: αν τον θέλετε με μηδενικά, μην τον αρχικοποιείτε, απλώς δηλώστε τον παγκόσμιο.

§14.18 Δείκτες και πίνακες με το ίδιο literal

Εφαρμογή της «String literals στη στατική μνήμη».

```
#include <stdio.h>
int main() {
  char * str1 = "Hello World", * str2 = "Hello World";
  printf("%p %p\n", str1, str2);
  return 0;
}
```

```
$ gcc -o const const.c
```

```
$ ./const
0x5588197ac004 0x5588197ac004
```

Οι δύο δείκτες δείχνουν στο ίδιο αντίγραφο του literal. Με πίνακες, `char str1[] = "Hello World", str2[] = "Hello World";`, η έξοδος γίνεται:

```
$ ./const
0x7ffc2576ccc4 0x7ffc2576ccb8
```

Δύο διευθύνσεις στη στοίβα (`0x7ffc…`), σε απόσταση 12 bytes: όσο ένας πίνακας των 11 χαρακτήρων και του null byte.

§14.19 Το μήκος του πρώτου ορίσματος

Εφαρμογή της «Οι συναρτήσεις της string.h».

```c
#include <stdio.h>
#include <string.h>
int main(int argc, char ** argv) {
  if (argc != 2) return 1;
  printf("Arg1 length: %zu\n", strlen(argv[1]));
  return 0;
}
```

```
$ gcc -o strlen strlen.c
$ ./strlen "hello world"
Arg1 length: 11
```

Τα εισαγωγικά κάνουν το `hello world` ένα όρισμα· το κενό μετράει, το null byte όχι.

§14.20 Η strcmp από τη γραμμή εντολών

Εφαρμογή της «Οι συναρτήσεις της string.h».

```c
#include <stdio.h>
#include <string.h>
int main(int argc, char ** argv) {
  if (argc != 3) return 1;
  printf("strcmp(arg1, arg2): %d\n",
         strcmp(argv[1], argv[2]));
  return 0;
}
```

```
$ ./strcmp foo bar
strcmp(arg1, arg2): 4
$ ./strcmp foo foo
strcmp(arg1, arg2): 0
```

```
$ ./strcmp f bar
strcmp(arg1, arg2): 4
$ ./strcmp bar f
strcmp(arg1, arg2): -4
$ ./strcmp bar c
strcmp(arg1, arg2): -1
$ ./strcmp bar ba
strcmp(arg1, arg2): 114
```

Στο `foo/bar` η πρώτη διαφορά είναι `'f' - 'b'` = 102 − 98 = 4. Στο `bar/ba` το `ba` τελειώνει πρώτο, οπότε συγκρίνεται το `'r'` (114) με το `'\0'` (0). Η σειρά είναι λεξικογραφική: το `f` είναι «μεγαλύτερο» από το `bar` παρότι είναι πιο κοντό.

§14.21 Αντιγραφή και συνένωση

Εφαρμογή της «Οι συναρτήσεις της string.h».

```c
#include <stdio.h>
#include <string.h>
int main(int argc, char ** argv) {
  char hello[16] = "Hello!";
  char world[16];
  strcpy(world, hello);
  printf("world: %s\n", world);
  return 0;
}
```

```
$ ./strcpy
world: Hello!
```

Το πρόγραμμα της `strcat` έχει στο σώμα του:

```c
char hello[16] = "Hello ";
char world[16] = "World!";
strcat(hello, world);
printf("concat: %s\n", hello);
```

```
$ ./strcat
concat: Hello World!
```

Ο χώρος φτάνει: το `"Hello World!"` θέλει 12 χαρακτήρες και το null byte, 13 από τα 16 bytes του `hello`. Με `char hello[8]` η `strcat` θα έγραφε έξω από τον πίνακα.

§14.22 Πιθανές υλοποιήσεις

Εφαρμογή της «Η έκφραση *str++». Οι υλοποιήσεις των διαφανειών (η βιβλιοθήκη δηλώνει τα ορίσματα που δεν αλλάζουν ως `const char *`):

```c
size_t strlen(char * str) {
  size_t length = 0;
  while (*str++) length++;
  return length;
}

int strcmp(char * str1, char * str2) {
  while (*str1 && (*str1 == *str2)) {
    str1++;
    str2++;
  }
  return *str1 - *str2;
}

char * strcpy(char * dst, char * src) {
  char * original_dst = dst;
  while (*src) *dst++ = *src++;
  *dst = '\0';
  return original_dst;
}

char *strcat(char *dst, char *src) {
  char *original_dst = dst;
  while (*dst) dst++;
  while (*src) *dst++ = *src++;
  *dst = '\0';
  return original_dst;
}
```

- `strlen`: το `*str++` διαβάζει τον τρέχοντα χαρακτήρα και προχωρά τον δείκτη· ο βρόχος σταματά στο `'\0'` (ψευδές) χωρίς να το μετρήσει.
- `strcmp`: προχωρά όσο οι χαρακτήρες είναι ίσοι και το πρώτο string δεν τελείωσε, και επιστρέφει τη διαφορά των πρώτων διαφορετικών χαρακτήρων (0 αν τελείωσαν μαζί).
- `strcpy`: κρατά την αρχική τιμή του `dst`, γιατί ο βρόχος τον μετακινεί, αντιγράφει ως το `'\0'` και γράφει το null byte ρητά. Κανένας έλεγχος χώρου.
- `strcat`: ο πρώτος βρόχος φτάνει στο null byte του `dst`· από εκεί συνεχίζει όπως η `strcpy`.

§14.23 Είναι παλινδρομικό;

Εφαρμογή της «Οι συναρτήσεις της string.h». Ένα string είναι παλινδρομικό αν διαβάζεται ίδιο και από τις δύο μεριές. Η λύση των διαφανειών:

```c
int isPalindrome(char *str) {
    int left = 0;
```

```
    int right = strlen(str) - 1;
    while (left < right) {
        if (str[left] != str[right]) return 0;
        left++;
        right--;
    }
    return 1;
}
```

Δύο δείκτες θέσης ξεκινούν από τις άκρες και πλησιάζουν· με την πρώτη διαφορά η απάντηση είναι «όχι», κι αν συναντηθούν, «ναι». Το `strlen(str) - 1` είναι ο τελευταίος ορατός χαρακτήρας, πριν από το null byte. Η διάλεξη θέτει και την ερώτηση «πώς ελέγχω αν το πρώτο όρισμα του προγράμματός μου είναι `"--boo"`;», που τη βρίσκετε στις «Ασκήσεις».

Κύρια σημεία

1. Εμβέλεια είναι το μέρος του προγράμματος όπου μια μεταβλητή είναι ορατή, και την καθορίζει το σημείο της δήλωσης.
2. Μια τοπική μεταβλητή (και μια παράμετρος) είναι ορατή από τη δήλωσή της ως το τέλος του block της και ζει στη στοίβα· με το ίδιο όνομα σε άλλη συνάρτηση είναι άλλη μεταβλητή.
3. Μια παγκόσμια μεταβλητή δηλώνεται έξω από συναρτήσεις και είναι ορατή από τη δήλωσή της ως το τέλος του αρχείου.
4. Μια εσωτερική δήλωση με το ίδιο όνομα επισκιάζει την εξωτερική μέσα στο block της.
5. Μια `static` μεταβλητή συνάρτησης αρχικοποιείται μία φορά και κρατά την τιμή της ανάμεσα στις κλήσεις, εκτός αν της αναθέσουμε τιμή σε ξεχωριστή εντολή.
6. Υπάρχουν τρεις κατηγορίες μνήμης, η στοίβα, ο σωρός και η παγκόσμια / στατική μνήμη, που ζει από την αρχή ως το τέλος του προγράμματος.
7. Οι παγκόσμιες και οι στατικές μεταβλητές γίνονται σύμβολα του αρχείου αντικειμένου (`nm`), και μόνο όσες έχουν μη μηδενικές αρχικές τιμές πιάνουν χώρο στο αρχείο.
8. Τα string literals που δεν αρχικοποιούν τοπικό πίνακα ζουν στη στατική μνήμη, ενώ ο `char s[] = "…"` είναι δικό του αντίγραφο στη στοίβα.
9. Ένα string είναι πίνακας χαρακτήρων που τελειώνει πάντα με το null byte `'\0'`.
10. Η `strlen` μετρά χωρίς το null byte, η `strcmp` επιστρέφει 0, θετικό ή αρνητικό, η `strcpy` αντιγράφει και η `strcat` συνενώνει.
11. Το `*str++` σημαίνει `*(str++)`, όχι `(*str)++`: η προτεραιότητα των τελεστών μετράει.
12. Οι `strcpy` και `strcat` δεν ελέγχουν αν χωρά το αποτέλεσμα, οπότε τον έλεγχο τον κάνουμε πάντα εμείς.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
εμβέλεια	scope	Το μέρος του προγράμματος όπου ένα όνομα είναι ορατό.
τοπική μεταβλητή	local variable	Δηλώνεται σε συνάρτηση· ορατή ως το τέλος του block.
παγκόσμια μεταβλητή	global variable	Δηλώνεται έξω από συναρτήσεις· ορατή ως το τέλος του αρχείου.
επισκίαση	shadowing	Εσωτερική δήλωση με ίδιο όνομα κρύβει την εξωτερική.
στατική μεταβλητή	static variable	Κρατά την τιμή της ανάμεσα σε κλήσεις, ως το τέλος του προγράμματος.
χρόνος ζωής	lifetime	Το διάστημα της εκτέλεσης όπου υπάρχει η μεταβλητή.
παγκόσμια / στατική μνήμη	global / static memory	Μνήμη που ζει όσο το πρόγραμμα.
σύμβολο	symbol	Όνομα συνάρτησης ή μεταβλητής σε αρχείο αντικειμένου.
σταθερή συμβολοσειρά	string literal	Κείμενο σε εισαγωγικά μέσα στον κώδικα.
συμβολοσειρά / αλφαριθμητικό	string	Πίνακας `char` που τελειώνει με `'\0'`.
null byte	null byte	Ο χαρακτήρας `'\0'` που σημαδεύει το τέλος.
συνένωση	concatenation	Προσάρτηση ενός string στο τέλος άλλου.
υπερχείλιση buffer	buffer overflow	Εγγραφή πέρα από το τέλος ενός πίνακα.

Διάβασμα

- **Διαφάνειες**: Διάλεξη 14, σελ. 1–46: εμβέλεια, τοπικές και παγκόσμιες μεταβλητές 5–13· επισκίαση 14–16· στατικές μεταβλητές 17–19· κατηγορίες μνήμης και στατική μνήμη 20–27· συμβολοσειρές και `string.h` 28–41· ερωτήσεις (`--boo`, παλινδρομικό) 42–44.
- **Σημειώσεις**: η διάλεξη λέει ότι κάλυψε τις σελ. 63–68, 74–76 και 93–96 των διαφανειών του κ. Σταματόπουλου:
  - Κεφάλαιο 4: Συναρτήσεις, εμβέλεια και αναδρομή, ενότητα «Εμβέλεια και χρόνος ζωής μεταβλητών» (K04, σελ. 63–68)·
  - Κεφάλαιο 5: Δείκτες και πίνακες, ενότητα «Δείκτες» (K04, σελ. 74–76)·
  - Κεφάλαιο 6: Δυναμική μνήμη, συμβολοσειρές και πολυδιάστατοι πίνακες, ενότητα «Συμβολοσειρές» (K04, σελ. 93–96).

- **Εργαστήριο**: Εργαστήριο 8: άσκηση `string.c` (`strcpy`, `mystrlen`, `mystrcat`, `strcmp`, `strcat`, `strtok`) και `argcalc.c` (ορίσματα γραμμής εντολών ως string).
- **Άλλα**: Static variable (Wikipedia) και Static variables in C· Data segment και .bss· Variable shadowing· Why strcpy and strncpy are not safe to use· strdup, strchr, strtok· `man strncpy`, `man nm`.

Συχνά λάθη

- **Αναμονή ότι η συνάρτηση αλλάζει τη μεταβλητή του καλούντος**. Το `foo(i)` με `i++` μέσα στη `foo` αφήνει το `i` της `main` ίδιο.
- **Χρήση μεταβλητής έξω από το block της**. Ένα `int k` δηλωμένο μέσα σε `for` ή `if` δεν υπάρχει μετά το `}`: `error: 'k' undeclared`.
- **Άθελη επισκίαση**. Μια τοπική με το όνομα μιας παγκόσμιας κάνει τις αλλαγές να «χάνονται». Μεταγλωττίστε με `-Wshadow` και δώστε άλλα ονόματα.
- `static` **με ανάθεση σε χωριστή εντολή**. Το `static int c; c = 0;` μηδενίζει τον μετρητή σε κάθε κλήση· βάλτε την αρχικοποίηση στη δήλωση.
- **Τεράστιος αρχικοποιημένος παγκόσμιος πίνακας**. Το `= {1}` σε πίνακα 400 MB δίνει αρχείο 382 MB ή `section size is larger than file size`.
- **Αλλαγή ενός string literal**. Το `char *s = "abc"; s[0] = 'x';` μπορεί να δώσει `Segmentation fault`· για string που αλλάζει γράψτε `char s[] = "abc";`.
- **Χωρίς χώρο για το null byte**. Το `char s[5] = "hello";` δεν έχει θέση για το `'\0'`, και η `strlen` ή η `printf("%s")` διαβάζουν πέρα από τον πίνακα. Ένα string θέλει `strlen + 1` bytes.
- **Σύγκριση string με** `==`**, ή** `if (strcmp(a, b))` **για ισότητα**. Το πρώτο συγκρίνει διευθύνσεις· το δεύτερο είναι αληθές όταν **διαφέρουν**. Γράψτε `strcmp(a, b) == 0`.
- `strcpy`/`strcat` **σε μικρό πίνακα**. Γράφουν πέρα από το τέλος χωρίς προειδοποίηση (buffer overflow, π.χ. `*** stack smashing detected ***`)· ελέγξτε τα μήκη πριν.
- `strlen` **με** `%d`. Ο `gcc -Wall` λέει `format '%d' expects argument of type 'int'`· η `strlen` επιστρέφει `size_t`, που τυπώνεται με `%zu`.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K14.8** Πού αποθηκεύονται οι τοπικές μεταβλητές (23% σωστές): Το 55% απάντησε «Στην στοίβα», ξεχνώντας τις τοπικές `static` μεταβλητές, που έχουν τοπική εμβέλεια αλλά ζουν στη στατική μνήμη για όλη την εκτέλεση.
- **K14.7** Η strcat με τον εαυτό της (24% σωστές): Το 34% επέλεξε bananabananabananabanana και το 24% bananabanana, κοιτάζοντας μόνο τη λογική της συνένωσης: ο πίνακας έχει 10 bytes, οπότε ήδη η πρώτη `strcat` γράφει εκτός ορίων, και η επικάλυψη πηγής και προορισμού είναι από μόνη της απροσδιόριστη συμπεριφορά.

- **K14.6** strcpy από literal σε πίνακα (32% σωστές): Το 24% απάντησε ότι θα κρασάρει, μπερδεύοντας τους ρόλους: η `strcpy` γράφει στον πίνακα `s2`, που είναι εγγράψιμος και χωράει τα 6 bytes του `"World"`· το literal `s1` μόνο διαβάζεται.
- **K14.5** Η τιμή της strcmp για ίσες συμβολοσειρές (48% σωστές): Το 30% επέλεξε 1, διαβάζοντας την `strcmp` σαν έλεγχο ισότητας που επιστρέφει «αληθές» όταν οι συμβολοσειρές είναι ίδιες.
- **K14.4** Πού αποθηκεύεται το char str[] (49% σωστές): Το 31% απάντησε «Στην στοίβα», υποθέτοντας σιωπηρά ότι η δήλωση είναι μέσα σε συνάρτηση· αν είναι παγκόσμια, ο πίνακας βρίσκεται στη στατική μνήμη.
- **K14.2** Δύο είδη εμβέλειας (63% σωστές): Το 31% απάντησε False, πιθανόν θεωρώντας τις `static` μεταβλητές τρίτο είδος εμβέλειας· η `static` αλλάζει τον χρόνο ζωής και τη μνήμη, όχι το από πού είναι ορατή μια τοπική μεταβλητή.

Ερωτήσεις κατανόησης

- **E14.1** Ποια είναι η εμβέλεια μιας παραμέτρου συνάρτησης και ποια μιας παγκόσμιας μεταβλητής;[114]
- **E14.2** Σε ποια μνήμη ζει μια `static int` μεταβλητή μιας συνάρτησης, και γιατί δεν χάνεται η τιμή της;[115]
- **E14.3** Γιατί ο παγκόσμιος `int a[100000000];` δεν μεγαλώνει το `.o`, ενώ ο `int a[100000000] = {1};` το κάνει 382 MB;[116]
- **E14.4** Τι επιστρέφουν τα `strlen("abc")` και `sizeof("abc")`;[117]
- **E14.5** Ποια είναι η διαφορά ανάμεσα στο `*p++` και το `(*p)++`;[118]
- **E14.6** Με ποια συνθήκη ελέγχουμε αν δύο string `a` και `b` είναι ίσα;[119]

Kahoot από το αμφιθέατρο (K14.1–K14.8)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K14.1** Πού δηλώνονται οι παγκόσμιες μεταβλητές: 93% σωστές απαντήσεις
- **K14.2** Δύο είδη εμβέλειας: 63% σωστές απαντήσεις
- **K14.3** Διαδοχικές strcat: 60% σωστές απαντήσεις
- **K14.4** Πού αποθηκεύεται το char str[]: 49% σωστές απαντήσεις
- **K14.5** Η τιμή της strcmp για ίσες συμβολοσειρές: 48% σωστές απαντήσεις
- **K14.6** strcpy από literal σε πίνακα: 32% σωστές απαντήσεις

[114] Η παράμετρος: όλο το σώμα της συνάρτησης. Η παγκόσμια: από τη δήλωσή της ως το τέλος του αρχείου.

[115] Στην παγκόσμια / στατική μνήμη, που υπάρχει από την αρχή ως το τέλος του προγράμματος, όχι στο frame της στοίβας.

[116] Ο πρώτος είναι στο `.bss`: γράφεται μόνο το μέγεθος και ο χώρος γεμίζει με μηδενικά στην εκκίνηση. Ο δεύτερος είναι στο data: όλες οι τιμές γράφονται στο αρχείο.

[117] 3 και 4: η `strlen` δεν μετρά το `'\0'`, ο πίνακας όμως το περιέχει.

[118] Το `*p++` δίνει την τιμή όπου δείχνει ο `p` και προχωρά τον δείκτη· το `(*p)++` αυξάνει την τιμή και ο δείκτης μένει.

[119] `strcmp(a, b) == 0`· το `a == b` συγκρίνει διευθύνσεις.

- **K14.7** Η strcat με τον εαυτό της: 24% σωστές απαντήσεις
- **K14.8** Πού αποθηκεύονται οι τοπικές μεταβλητές: 23% σωστές απαντήσεις

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A14.1–A14.8)

- **A14.1** Είναι το πρώτο όρισμα "--boo";: Διάλεξη 14, διαφάνεια 42 · ★☆☆ · programming · `slides-lec14-check-boo`
- **A14.2** Αύξηση παραμέτρου μέσα σε συνάρτηση: Διάλεξη 14, διαφάνειες 11–12 · ★☆☆ · trace · `slides-lec14-local-param`
- **A14.3** Τοπική μεταβλητή με το ίδιο όνομα σε δύο συναρτήσεις: Διάλεξη 14, διαφάνειες 9–10 · ★☆☆ · trace · `slides-lec14-local-same-name`
- **A14.4** Παλινδρομικό string: Διάλεξη 14, διαφάνειες 43–44 · ★☆☆ · programming · `slides-lec14-palindrome`
- **A14.5** Επισκίαση παγκόσμιας μεταβλητής: Διάλεξη 14, διαφάνειες 14–16 · ★☆☆ · trace · `slides-lec14-shadowing`
- **A14.6** Στατική και τοπική μεταβλητή σε τρεις κλήσεις: Διάλεξη 14, διαφάνειες 17–18 · ★☆☆ · trace · `slides-lec14-static-counter`
- **A14.7** Στατική μεταβλητή με ανάθεση εκτός δήλωσης: Διάλεξη 14, διαφάνεια 19 · ★★☆ · trace · `slides-lec14-static-assign`
- **A14.8** Η έκφραση *str++: Διάλεξη 14, διαφάνεια 34 · ★★☆ · short-answer · `slides-lec14-str-plus-plus`

Εργαστήριο (A14.9)

- **A14.9** Επεξεργασία συμβολοσειρών: Εργαστήριο 8, Άσκηση 1 · ★★☆ · programming · `lab-lab08-string`

Εργασίες (A14.10)

- **A14.10** Το Δικό σου Chatbot (jason): Εργασία 2 (2024-25), Άσκηση 3 · ★★★ · programming · `hw-2024-hw2-jason`

Θέματα εξετάσεων (A14.11–A14.22)

- **A14.11** Η συνάρτηση dog: Εξέταση Ιανουαρίου 2025, Θέμα 2 · ★☆☆ · trace · `exam-2025-jan-q2`
- **A14.12** Η συνάρτηση transform: Εξέταση Σεπτεμβρίου 2025, Θέμα 2 · ★☆☆ · trace · `exam-2025-sep-q2`
- **A14.13** Ταιριαστές Καρδιές: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #0 (Valentine's Themed), Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex0-q2`

- **A14.14** Εντοπισμός Διπλών Ορισμάτων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #1 (Coreutils Themed), Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex1-q2`
- **A14.15** Προσεγγίζοντας την Λέξη: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #12, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex12-q2`
- **A14.16** Κρεμάλα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #12, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex12-q3`
- **A14.17** Δυνατότητες: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #15, Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex15-q4`
- **A14.18** Εύρεση Λέξεων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #3 (Frozen Themed), Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex3-q3`
- **A14.19** Τα Πάνω Κάτω: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #4 (Rick Astley Themed), Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex4-q2`
- **A14.20** Σπάσε το PIN: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #6 (Crypto Themed), Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex6-q3`
- **A14.21** Συνένωση Αλφαριθμητικών - join: Εξέταση Ιανουαρίου 2025, Θέμα 5 · ★★☆ · programming · `exam-2025-jan-q5`
- **A14.22** Μεταμορφώσιμες Προτάσεις: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #5 (HP Themed), Θέμα 4 · ★★★ · programming · `exam-2023-fall-ex5-q4`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A9.27** Αναζητώντας τον Blinky: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #11, Θέμα 1 · ★★☆ · programming · `exam-2023-fall-ex11-q1`
- **A10.22** Αναγράμματα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #4 (Rick Astley Themed), Θέμα 1 · ★★☆ · programming · `exam-2023-fall-ex4-q1`
- **A11.16** Η ακολουθία Fibonacci: Εργαστήριο 5, Άσκηση 2 · ★★☆ · programming · `lab-lab05-fib`
- **A13.11** Κάδρο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #9, Θέμα 4 · ★★★ · programming · `exam-2023-fall-ex9-q4`
- **A15.14** Πολυπλοκότητα της strcmp: Διάλεξη 15, διαφάνεια 31 · ★★☆ · short-answer · `slides-lec15-complexity-strcmp`
- **A15.6** Πολυπλοκότητα της strlen: Διάλεξη 15, διαφάνεια 29 · ★☆☆ · short-answer · `slides-lec15-complexity-strlen`
- **A18.15** Κρυμμένο Μήνυμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #10, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex10-q3`
- **A18.18** Επιλογή: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #5 (HP Themed), Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex5-q3`
- **A18.19** Ταξινόμηση Αρχείων Καταγραφής: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #6 (Crypto Themed), Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex6-q4`
- **A18.23** Ταξινόμηση Πακέτων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #8, Θέμα 4 · ★★★ · programming · `exam-2023-fall-ex8-q4`
- **A19.12** World Cup 2026: Εξέταση Ιουνίου 2026, Θέμα 4 · ★★★ · programming · `exam-2026-jun-q4`

- **A19.10** Δομές και δείκτες: Εργαστήριο 9, Άσκηση 2 · ★★☆ · programming · `lab-lab09-person`
- **A23.5** Σπάστε το πρόγραμμά σας σε αρθρώματα: Εργαστήριο 10, Άσκηση 5 · ★★☆ · tooling · `lab-lab10-more-modules`
- **A25.2** DNA Matching: Εργασία 2 (2023-24), Άσκηση 2 · ★★★ · programming · `hw-2023-hw2-dna`
- **A25.3** Το Καλύτερο GPS (jabbamaps): Εργασία 2 (2024-25), Άσκηση 2 · ★★★ · programming · `hw-2024-hw2-jabbamaps`

Κεφάλαιο 15: Πολυπλοκότητα και Προεπεξεργασ

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να εξηγείτε τι μετρά η χρονική και η χωρική πολυπλοκότητα ενός αλγορίθμου· να διαβάζετε τους συμβολισμούς O, Ω και Θ και να κατατάσσετε τις συνηθισμένες κλάσεις πολυπλοκότητας· να εκτιμάτε την πολυπλοκότητα βρόχων, εμφωλευμένων βρόχων και αναδρομικών συναρτήσεων· και να χρησιμοποιείτε τον προεπεξεργαστή (`#include`, `#define`, `#if`, `#ifdef`, `gcc -E`, `-D`).

Προαπαιτούμενα: Κεφάλαιο 0, Κεφάλαιο 11, Κεφάλαιο 12, Κεφάλαιο 14

Χρόνος μελέτης: ~2 ώρες

Σύνοψη

Η διάλεξη έχει δύο μέρη. Στο πρώτο μαθαίνουμε να συγκρίνουμε αλγορίθμους με την **πολυπλοκότητα**: πώς μεγαλώνει ο χρόνος εκτέλεσης και η μνήμη που χρειάζονται καθώς μεγαλώνει το μέγεθος του προβλήματος. Ο συμβολισμός Big-O μάς επιτρέπει να αγνοούμε σταθερές και λεπτομέρειες του μηχανήματος και να κρατάμε μόνο τον ρυθμό αύξησης. Στη συνέχεια εφαρμόζουμε την ιδέα σε προγράμματα που έχουμε ήδη δει (βρόχοι, `atoi`, `strlen`, `strcmp`, αναδρομή) και βλέπουμε ότι ένα καλύτερο σκεπτικό μπορεί να ρίξει την πολυπλοκότητα από $O(n)$ σε $O(\sqrt{n})$. Στο δεύτερο μέρος γνωρίζουμε τον **προεπεξεργαστή**, το πρώτο στάδιο του `gcc`, που μετασχηματίζει τον πηγαίο κώδικα πριν από τη μεταγλώττιση με τις οδηγίες `#include`, `#define` και τις οδηγίες μεταγλώττισης υπό συνθήκη.

Θεωρία

§15.1 Τι είναι η πολυπλοκότητα

Για να συγκρίνουμε δύο αυτοκίνητα κοιτάμε μετρήσιμα χαρακτηριστικά: τελική ταχύτητα, επιτάχυνση, κατανάλωση. Για να συγκρίνουμε δύο αλγορίθμους χρειαζόμαστε επίσης ένα μέτρο. Η **πολυπλοκότητα (complexity)** είναι ένα μέτρο εκτίμησης της απόδοσης ενός αλγορίθμου *ως συνάρτηση του μεγέθους του προβλήματος* που λύνει. Δύο είναι οι βασικές μετρικές:

1. **Χρονική πολυπλοκότητα (time complexity)**: πόσο χρόνο χρειάζεται η εκτέλεση.
2. **Χωρική πολυπλοκότητα (space complexity)**: πόση μνήμη απαιτείται.

Αν n είναι το μέγεθος του προβλήματος (το πλήθος των στοιχείων ενός πίνακα, το μήκος μιας συμβολοσειράς, η τιμή ενός αριθμού), θέλουμε να εκφράσουμε τον χρόνο εκτέλεσης ως $t = f(n)$.

Ο χρόνος σε δευτερόλεπτα δεν είναι καλό μέτρο: το ίδιο πρόγραμμα κάνει ένα λεπτό σε έναν αργό υπολογιστή και δευτερόλεπτα σε έναν γρήγορο. Γι' αυτό μετράμε **βήματα** (πόσες φορές εκτελείται ένας βρόχος, πόσες κλήσεις γίνονται) και μας ενδιαφέρει πώς αυξάνονται όταν αυξάνεται το n, όχι ο ακριβής αριθμός τους. Η θεωρία της πολυπλοκότητας είναι ολόκληρος κλάδος της πληροφορικής· ένας από τους πιο γνωστούς ερευνητές του είναι ο Χρίστος Παπαδημητρίου, συν-συγγραφέας και του κόμικ *Logicomix*.

§15.2 Ο συμβολισμός Big-O, Ω και Θ

Οι κλάσεις πολυπλοκότητας ορίζονται με τρεις συμβολισμούς. Για δύο συναρτήσεις f και g του μεγέθους n:

- **Άνω όριο**, O **(Big-O)**:

$$g = O(f) \iff \exists c. \exists n_0. \forall n > n_0. \quad g(n) < c \cdot f(n)$$

- **Κάτω όριο**, Ω:

$$g = \Omega(f) \iff \exists c. \exists n_0. \forall n > n_0. \quad g(n) > c \cdot f(n)$$

- **Τάξη μεγέθους**, Θ:

$$g = \Theta(f) \iff \exists c_1, c_2. \exists n_0. \forall n > n_0. \quad c_1 \cdot f(n) < g(n) < c_2 \cdot f(n)$$

Διαβάστε τον ορισμό του O ως εξής: *από κάποιο σημείο n_0 και μετά*, η g δεν ξεπερνά ποτέ ένα σταθερό πολλαπλάσιο της f. Πριν από το n_0 μπορεί να συμβαίνει οτιδήποτε· οι μικρές είσοδοι δεν μετράνε. Η σταθερά c «απορροφά» τους σταθερούς παράγοντες: ένας βρόχος που κάνει $n/2$ ή $3n + 5$ βήματα είναι και οι δύο $O(n)$. Το Ω λέει το αντίστροφο (η g μεγαλώνει *τουλάχιστον* όσο η f), και το Θ ότι ισχύουν και τα δύο, δηλαδή η g μεγαλώνει *ακριβώς* με τον ρυθμό της f. Στην πράξη, όταν λέμε «ο αλγόριθμος είναι $O(n)$», εννοούμε συνήθως το πιο σφικτό άνω όριο που ξέρουμε.

Η γραφική παράσταση των διαφανειών το δείχνει: οι καμπύλες $f(x)$ και $c \cdot g(x)$ διασταυρώνονται αρκετές φορές, αλλά μετά από ένα σημείο x_0 η $c \cdot g(x)$ μένει πάντα πάνω από την $f(x)$, άρα $f(x) = O(g(x))$.

§15.3 Διάταξη των κλάσεων πολυπλοκότητας

Οι συνηθισμένες κλάσεις, από την ταχύτερη στην πιο αργή, είναι:

$$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!) < O(n^n)$$

Κλάση	Όνομα	Παράδειγμα από τη διάλεξη
$O(1)$	σταθερή (constant)	υπολογισμός βαθμού με έναν τύπο
$O(\log n)$	λογαριθμική (logarithmic)	κατοπτρικός αριθμός (ψηφία του n)
$O(\sqrt{n})$	τετραγωνική ρίζα	άθροισμα τέλειων τετραγώνων, 2η εκδοχή
$O(n)$	γραμμική (linear)	`strlen`, `atoi`, παραγοντικό
$O(n \log n)$	log-linear	(καλές ταξινομήσεις, Κεφάλαιο 17)
$O(n^2)$	τετραγωνική (quadratic)	μέγιστο σε πίνακα $n \times n$
$O(n^3)$	κυβική, πολυωνυμική	τρεις εμφωλευμένοι βρόχοι
$O(2^n)$	εκθετική (exponential)	αναδρομικό Fibonacci
$O(n!)$	παραγοντική	όλες οι διατάξεις n στοιχείων

Οι γραφικές παραστάσεις των διαφανειών δείχνουν πόσο γρήγορα απομακρύνονται οι καμπύλες: για $n = 100$ η $\log_2 n$ είναι περίπου 7 και η $\sqrt{n}$ είναι 10, ενώ η n^2 είναι 10.000 και η 2^n έχει 31 ψηφία. Οι αλγόριθμοι μέχρι το πολυωνυμικό επίπεδο είναι συνήθως πρακτικοί· οι εκθετικοί και οι παραγοντικοί γίνονται άχρηστοι ήδη για μερικές δεκάδες στοιχεία.

§15.4 Πώς εκτιμούμε την πολυπλοκότητα ενός προγράμματος

Μερικοί απλοί κανόνες καλύπτουν όλα τα παραδείγματα της διάλεξης:

- **Σειρά εντολών χωρίς βρόχους**: $O(1)$, όσες εντολές κι αν είναι.
- **Ένας βρόχος που τρέχει n φορές** με $O(1)$ σώμα: $O(n)$. Το βήμα δεν αλλάζει την κλάση: `i += 2` κάνει $n/2$ επαναλήψεις, που είναι πάλι $O(n)$.
- **Εμφωλευμένοι βρόχοι**: οι επαναλήψεις πολλαπλασιάζονται· δύο βρόχοι των n δίνουν $O(n^2)$.
- **Βρόχος που διαιρεί** την τιμή με μια σταθερά σε κάθε βήμα (π.χ. `n /= 10`): τρέχει τόσες φορές όσα τα ψηφία, $O(\log n)$.
- **Κλήση συνάρτησης μέσα σε βρόχο**: πολλαπλασιάστε το κόστος της κλήσης με τις επαναλήψεις.
- **Αναδρομή**: μετράμε τις κλήσεις. Μία κλήση ανά επίπεδο με βάθος n δίνει $O(n)$· δύο κλήσεις ανά επίπεδο δίνουν δέντρο κλήσεων με έως 2^n κόμβους.

Για τη **μνήμη** μετράμε ό,τι μεγαλώνει με το n: έναν πίνακα n θέσεων (π.χ. με `malloc`) και, στην αναδρομή, τα πλαίσια στη **στοίβα (stack)**: κάθε ενεργή κλήση κρατά τις δικές της τοπικές μεταβλητές, οπότε βάθος αναδρομής n σημαίνει $O(n)$ μνήμη. Σταθερό πλήθος μεταβλητών είναι $O(1)$ χώρος.

Προσέξτε **ως προς τι** μετράτε: στην `atoi` το n είναι το πλήθος των ψηφίων, στην `mirror` η *τιμή* του αριθμού, στην `strcmp` τα μήκη δύο συμβολοσειρών.

§15.5 Μεταγλωττιστής και προεπεξεργαστής

Μεταγλωττιστής (compiler) είναι ένα πρόγραμμα που μετατρέπει εντολές μιας γλώσσας προγραμματισμού σε κώδικα μηχανής, ώστε να μπορεί να τον διαβάσει και να τον τρέξει ο υπολογιστής (Κεφάλαιο 0). Με την εντολή `gcc hello.c -o hello` ο πηγαίος κώδικας (source code) `hello.c` γίνεται δυαδικό πρόγραμμα (binary program) `hello`.

Ο `gcc` δεν δουλεύει σε ένα βήμα: εσωτερικά περνά τον κώδικα από διαδοχικά στάδια. Το πρώτο είναι ο **προεπεξεργαστής (preprocessor)**, ένα υποσύστημα του μεταγλωττιστή που καλείται αυτόματα πριν από την πραγματική μεταγλώττιση. Παίρνει ένα αρχείο C και παράγει ένα *άλλο* αρχείο C, εφαρμόζοντας τις οδηγίες που απευθύνονται σε αυτόν. Δουλεύει μόνο με κείμενο: δεν ξέρει τι είναι μεταβλητή ή τύπος.

Σχήμα: ο προεπεξεργαστής είναι το πρώτο στάδιο μέσα στον `gcc`.

Την έξοδο του προεπεξεργαστή τη βλέπουμε με την επιλογή `-E` του `gcc` ή τρέχοντας απευθείας το πρόγραμμα του προεπεξεργαστή, `cpp`:

```
gcc -E hello.c -o processed.c
cpp hello.c -o processed.c
```

Το `processed.c` είναι ο **προεπεξεργασμένος πηγαίος κώδικας (preprocessed source code)**: αυτό που πραγματικά μεταγλωττίζεται. Οι γραμμές που αρχίζουν με `#` στην έξοδο (π.χ. `# 1 "hello.c"`) είναι σημειώσεις για τον μεταγλωττιστή σχετικά με το από ποιο αρχείο και ποια γραμμή προήλθε κάθε κομμάτι.

§15.6 Οδηγίες προεπεξεργαστή

Όλες οι **οδηγίες προς τον προεπεξεργαστή (preprocessor directives)** αρχίζουν με το σύμβολο `#`. Οι πιο συνηθισμένες είναι:

1. `#include`: εισαγωγή αρχείου.
2. `#define`: ορισμός μακροεντολών.
3. `#if`, `#else`, `#elif`, `#endif`: μεταγλώττιση υπό συνθήκη.
4. `#ifdef`, `#ifndef`: έλεγχος αν έχει οριστεί μια μακροεντολή.

Δεν τελειώνουν με `;`, γιατί δεν είναι εντολές της C.

§15.7 Η οδηγία `#include`

Η `#include <file.h>` εισάγει *τα περιεχόμενα* του αρχείου `file.h` στο σημείο όπου γράφτηκε η οδηγία, σαν να τα είχαμε αντιγράψει εκεί. Τα αρχεία αυτά είναι συνήθως **αρχεία**

επικεφαλίδας (header files) με δηλώσεις συναρτήσεων, σταθερές και μακροεντολές· για παράδειγμα, το `stdio.h` δηλώνει την `printf`.

Πού βρίσκονται; Η μορφή με `< >` τα ψάχνει σε προκαθορισμένους φακέλους του λειτουργικού (π.χ. `/usr/include`) ή σε φακέλους που δίνουμε με το όρισμα `-I` του `gcc`. Η δεύτερη μορφή, `#include "file.h"`, ψάχνει *πρώτα* στον φάκελο όπου βρίσκεται το πηγαίο αρχείο και μετά στους ίδιους φακέλους με τη μορφή `< >`. Η πρώτη μορφή είναι για τις βιβλιοθήκες του συστήματος, η δεύτερη για τα δικά μας αρχεία. Η οργάνωση ενός προγράμματος σε πολλά αρχεία έρχεται στο Κεφάλαιο 23.

§15.8 Η οδηγία `#define` και οι μακροεντολές

Η `#define` ορίζει **μακροεντολές (macros)**: ονόματα που ο προεπεξεργαστής αντικαθιστά με το κείμενο που τους αντιστοιχίσαμε, όπου τα βρει ως ξεχωριστή λέξη (όχι μέσα σε `"..."` ή σε σχόλια). Δύο χρήσεις:

1. **Ορισμός σταθεράς**: `#define TRUE 1`. Κάθε `TRUE` γίνεται `1`.
2. **Ορισμός υπολογισμού (μακροεντολή με παραμέτρους)**:

```
#define MAX(A, B) ((A) > (B) ? (A) : (B))
```

Το `MAX(x + 1, y)` γίνεται `((x + 1) > (y) ? (x + 1) : (y))`. Μοιάζει με κλήση συνάρτησης, αλλά δεν είναι: δεν υπάρχουν τύποι ούτε κλήση, μόνο **αντικατάσταση κειμένου πριν από τη μεταγλώττιση**. Γι' αυτό:

- Βάζουμε **παρενθέσεις** γύρω από κάθε παράμετρο και γύρω από όλο το κείμενο. Χωρίς αυτές οι τελεστές του κώδικα γύρω από τη μακροεντολή «μπερδεύονται» με τους δικούς της λόγω προτεραιότητας (δείτε το παράδειγμα με το `PROD`).
- Μια παράμετρος που εμφανίζεται δύο φορές στο κείμενο υπολογίζεται δύο φορές: το `MAX(i++, j++)` αυξάνει μία από τις δύο μεταβλητές δύο φορές.

Η τιμή μιας μακροεντολής μπορεί να δοθεί και **από τη γραμμή εντολών**, με τη σύνταξη `-DMACRO=VALUE` του `gcc`· το σκέτο `-DMACRO` την ορίζει με τιμή `1`. Έτσι ένα πρόγραμμα αλλάζει συμπεριφορά χωρίς να αλλάξει ούτε γράμμα του κώδικα.

§15.9 Μεταγλώττιση υπό συνθήκη: `#if`, `#else`, `#endif`

Οι οδηγίες `#if` / `#else` / `#endif` μοιάζουν με το `if` της C, αλλά δρουν **στο επίπεδο του κώδικα**: δεν αποφασίζουν τι θα εκτελεστεί, αλλά *ποιες γραμμές θα μείνουν* στο αρχείο που θα μεταγλωττιστεί. Ό,τι βρίσκεται ανάμεσα σε `#if` και `#endif` κρατιέται μόνο αν η ακέραια παράσταση μετά το `#if` είναι μη μηδενική· αλλιώς κρατιέται το τμήμα του `#else`. Το `#elif` είναι συντομογραφία για `#else` που περιέχει άλλο `#if`. Η παράσταση υπολογίζεται από τον προεπεξεργαστή, άρα πρέπει να περιέχει σταθερές και μακροεντολές, όχι μεταβλητές του προγράμματος.

Μια συνηθισμένη χρήση είναι το `#if 0 ... #endif`, που «σβήνει» προσωρινά ένα κομμάτι κώδικα χωρίς να το σβήσουμε πραγματικά.

§15.10 Οι οδηγίες `#ifdef` και `#ifndef`

Με το `#ifdef NAME` ελέγχουμε αν η μακροεντολή `NAME` *έχει οριστεί* (με `#define` ή με `-DNAME`), ανεξάρτητα από την τιμή της· το `#ifndef NAME` ελέγχει το αντίθετο. Ισοδύναμα γράφεται `#if defined(NAME)` και `#if !defined(NAME)`. Τυπικές χρήσεις:

- **Μηνύματα αποσφαλμάτωσης (debugging)** που ενεργοποιούνται με `-DDEBUG` και εξαφανίζονται εντελώς από το τελικό πρόγραμμα όταν δεν το δίνουμε.
- **Include guards**: ένα αρχείο επικεφαλίδας τυλίγεται σε `#ifndef NAME_H` / `#define NAME_H` / `#endif`, ώστε τα περιεχόμενά του να συμπεριληφθούν μόνο μία φορά (δείτε το Παράρτημα του Εργαστηρίου 10).

Παραδείγματα

§15.11 Γινόμενο των περιττών πολλαπλασίων του 7

Εφαρμόζει: «Πώς εκτιμούμε την πολυπλοκότητα». Θέλουμε το γινόμενο όλων των περιττών από το 1 μέχρι το `N` που διαιρούνται με το 7 (το πρόβλημα του Κεφαλαίου 7). Ένας βρόχος με μια μεταβλητή που αυξάνεται κατά 2 και έλεγχο για `% 7 == 0`:

```
for (product = 1, i = 1; i < N; i += 2) {
  if (i % 7 == 0)
    product *= i;
}
```

Χρόνος: $O(N)$, **χώρος**: $O(1)$. Ο βρόχος κάνει περίπου $N/2$ επαναλήψεις, και η σταθερά $1/2$ δεν αλλάζει την κλάση. Ο χώρος είναι δύο μεταβλητές, όσο μεγάλο κι αν είναι το `N`. (Αν το «μέχρι το `N`» περιλαμβάνει το `N`, η συνθήκη πρέπει να είναι `i <= N`.)

§15.12 Η `atoi`

Εφαρμόζει: έναν βρόχο ανά στοιχείο. Η συνάρτηση του Κεφαλαίου 10 μετατρέπει έναν πίνακα χαρακτήρων (μόνο ψηφία) σε ακέραιο:

```
int atoi(char digits[]) {
  int result = 0;
  for (int i = 0; digits[i]; i++) {
    result = 10 * result + digits[i] - '0';
  }
  return result;
}
```

Χρόνος: $O(N)$, **χώρος**: $O(1)$, όπου N το πλήθος των ψηφίων: μία επανάληψη ανά ψηφίο και μόνο δύο τοπικές μεταβλητές. (Το όνομα `atoi` συμπίπτει με τη συνάρτηση της `stdlib.h`· εδώ είναι δική μας υλοποίηση.)

§15.13 Μέτρηση χαρακτήρων με `getchar`

Εφαρμόζει: βρόχος ανά χαρακτήρα εισόδου. Διαδοχικές κλήσεις της `getchar()` διαβάζουν διαδοχικούς χαρακτήρες (Κεφάλαιο 9). Το πρόγραμμα ξανατυπώνει μια γραμμή και μετρά τους χαρακτήρες της:

```c
#include <stdio.h>

int main() {
  int ch, sum = 0;
  printf("Enter characters: ");
  while ((ch = getchar()) != '\n' && ch != EOF) {
    printf("%c", ch);
    sum++;
  }
  printf("\nTotal characters: %d\n", sum);
  return 0;
}
```

```
$ echo hello | ./count
Enter characters: hello
Total characters: 5
```

Χρόνος: $O(N)$, **χώρος**: $O(1)$, με N το μήκος της γραμμής: το πρόγραμμα *δεν* αποθηκεύει τη γραμμή, κρατά μόνο τον τρέχοντα χαρακτήρα και τον μετρητή.

§15.14 Δυναμικός πίνακας με `malloc`

Εφαρμόζει: χωρική πολυπλοκότητα. Με τους δείκτες μπορούμε να φτιάξουμε **δυναμικούς πίνακες**, το μέγεθος των οποίων αποφασίζεται τη στιγμή που τρέχει το πρόγραμμα (Κεφάλαιο 12, Κεφάλαιο 13):

```c
int *array = malloc(N * sizeof(int));
for (int i = 0; i < N; i++)
  array[i] = i * i;
```

Χρόνος: $O(N)$, **χώρος**: $O(N)$. Εδώ ο χώρος μεγαλώνει με το `N`, γιατί δεσμεύουμε `N` ακεραίους. (Σε πλήρες πρόγραμμα ελέγχουμε αν η `malloc` επέστρεψε `NULL` και καλούμε `free` στο τέλος.)

§15.15 Υπολογισμός βαθμολογίας

Εφαρμόζει: $O(1)$. Το γνωστό μας παράδειγμα από τα Κεφάλαια 4 και 5:

```c
// Compute grades using the class formula
int grade(int final_exam, int homework, int lab, int year) {
  if (year <= 1) {
    return final_exam * 50 / 100 + homework * 30 / 100 + lab * 20 / 100;
  } else {
    return final_exam * 70 / 100 + homework * 30 / 100;
  }
}
```

Χρόνος: $O(1)$, **χώρος**: $O(1)$. Δεν υπάρχει βρόχος ούτε αναδρομή· ο αριθμός των πράξεων είναι ο ίδιος για κάθε είσοδο.

§15.16 Μέγιστο στοιχείο σε πίνακα N × N

Εφαρμόζει: εμφωλευμένοι βρόχοι.

```c
int find_max(int **matrix, size_t n) {
  int i, j, max = -1;
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
      if (matrix[i][j] > max) max = matrix[i][j];
    }
  }
  return max;
}
```

Χρόνος: $O(n^2)$, **χώρος**: $O(1)$. Ο εσωτερικός βρόχος τρέχει n φορές για κάθε μία από τις n επαναλήψεις του εξωτερικού: $n \cdot n$ συγκρίσεις. Προσέξτε ότι η αρχική τιμή `max = -1` δουλεύει μόνο αν όλα τα στοιχεία είναι μη αρνητικά· η γενική λύση είναι `max = matrix[0][0]`.

§15.17 Παραγοντικό και Fibonacci

Εφαρμόζει: πολυπλοκότητα αναδρομής (Κεφάλαιο 11).

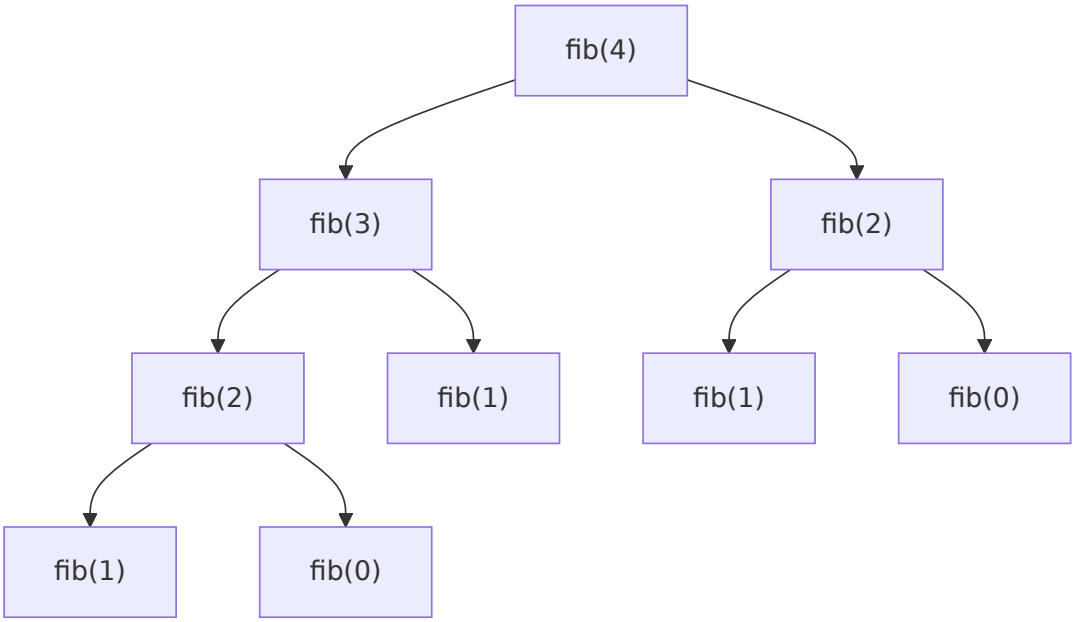
```c
int factorial(int n) {
  if (n == 0) return 1;
  return n * factorial(n - 1);
}
```

Χρόνος: $O(n)$, **χώρος**: $O(n)$. Το `factorial(n)` κάνει μία αλυσίδα $n+1$ κλήσεων. Ο χώρος *δεν* είναι $O(1)$: πριν επιστρέψει η πρώτη κλήση, και οι $n+1$ κλήσεις είναι ενεργές ταυτόχρονα, η καθεμία με το δικό της πλαίσιο στη στοίβα.

```c
int fib(int n) {
  if (n == 0 || n == 1) return 1;
  return fib(n - 1) + fib(n - 2);
```

```
}
```

Χρόνος: $O(2^n)$, **χώρος**: $O(n)$. Κάθε κλήση κάνει *δύο* αναδρομικές κλήσεις, οπότε οι κλήσεις σχηματίζουν δέντρο που σχεδόν διπλασιάζεται σε κάθε επίπεδο, και οι ίδιες τιμές υπολογίζονται ξανά και ξανά:



Σχήμα: το δέντρο κλήσεων του `fib(4)`*· το* `fib(2)` *υπολογίζεται δύο φορές.*

Η μνήμη όμως μένει $O(n)$: σε κάθε στιγμή είναι ενεργές μόνο οι κλήσεις ενός μονοπατιού από τη ρίζα ως ένα φύλλο, και το μακρύτερο μονοπάτι έχει μήκος n.

§15.18 `strlen` και `strcmp`

Εφαρμόζει: βρόχος ανά χαρακτήρα. Μια πιθανή υλοποίηση της `strlen` (Κεφάλαιο 14):

```
size_t strlen(char *str) {
  size_t length = 0;
  while (*str++) length++;
  return length;
}
```

Χρόνος: $O(n)$, **χώρος**: $O(1)$, όπου n το μήκος της συμβολοσειράς. Η C δεν αποθηκεύει πουθενά το μήκος· για να το βρει, η `strlen` πρέπει να διατρέξει όλη τη συμβολοσειρά μέχρι το `'\0'`.

```
int strcmp(char *str1, char *str2) {
  while (*str1 && (*str1 == *str2)) {
    str1++;
    str2++;
```

```
  }
  return *str1 - *str2;
}
```

Χρόνος: $O(\min(m, n))$, **χώρος**: $O(1)$, για συμβολοσειρές μήκους n και m. Ο βρόχος σταματά στην πρώτη διαφορά ή στο τέλος της συντομότερης, άρα δεν μπορεί να κάνει περισσότερα βήματα από το μικρότερο μήκος.

§15.19 Άθροισμα τέλειων τετραγώνων

Εφαρμόζει: κλήση μέσα σε βρόχο και βελτίωση πολυπλοκότητας. Θέλουμε το άθροισμα των τέλειων τετραγώνων στο διάστημα `[low, high]`, με τα όρια από τη γραμμή εντολών. Πρώτη εκδοχή: ελέγχουμε κάθε αριθμό.

```
int isPerfectSquare(int num) {
  int root = sqrt(num);
  return root * root == num;
}
// ... στη main:
int low = atoi(argv[1]);
int high = atoi(argv[2]);
int i, sum = 0;
for (i = low; i <= high; i++) {
  if (isPerfectSquare(i))
    sum += i;
}
```

Χρόνος: $O(n)$, **χώρος**: $O(1)$, όπου n το πλήθος των αριθμών του διαστήματος (η `sqrt` μετράει εδώ ως ένα βήμα). Δεύτερη εκδοχή: αντί να ψάχνουμε ποιοι αριθμοί είναι τετράγωνα, *παράγουμε* απευθείας τα τετράγωνα i^2 για i από $\sqrt{low}$ ως $\sqrt{high}$:

```
int low = atoi(argv[1]);
int high = atoi(argv[2]);
int i, sum = 0;
for (i = sqrt(low); i <= sqrt(high); i++)
  sum += i * i;
```

Χρόνος: $O(\sqrt{n})$, **χώρος**: $O(1)$. Για διάστημα ενός εκατομμυρίου αριθμών ο βρόχος κάνει περίπου χίλιες επαναλήψεις αντί για ένα εκατομμύριο. Το δίδαγμα: η μεγαλύτερη βελτίωση έρχεται από καλύτερο σκεπτικό, όχι από «γρηγορότερο» κώδικα.

Προσοχή σε μια λεπτομέρεια: η ανάθεση `i = sqrt(low)` κόβει το δεκαδικό μέρος. Αν το `low` δεν είναι τέλειο τετράγωνο, π.χ. `low = 5`, το `i` ξεκινά από το 2 και το άθροισμα περιλαμβάνει το 4, που είναι εκτός διαστήματος. Η σωστή αρχή είναι το μικρότερο `i` με `i * i >= low`. Και οι δύο εκδοχές χρειάζονται `#include <math.h>` και μεταγλώττιση με `-lm`.

§15.20 Κατοπτρικός αριθμός

Εφαρμόζει: $O(\log n)$. Η `mirror` επιστρέφει τον αριθμό με τα ψηφία ανάποδα (`1234` ⟶ `4321`):

```
int mirror(int n) {
  int result = 0, tmp;
  while (n > 0) {
    tmp = n % 10;
    result = 10 * result + tmp;
    n /= 10;
  }
  return result;
}
```

Χρόνος: $O(\log n)$, **χώρος**: $O(1)$. Κάθε επανάληψη διαιρεί το `n` με το 10, οπότε ο βρόχος τρέχει τόσες φορές όσα είναι τα ψηφία του n, δηλαδή περίπου $\log_{10} n$. Εδώ το «μέγεθος» είναι η *τιμή* του n, όχι το πλήθος των ψηφίων του.

Η διάλεξη κλείνει το μέρος αυτό με ένα ανοιχτό ερώτημα: τι χρονική πολυπλοκότητα έχει ο έλεγχος αν ένας αριθμός είναι πρώτος; Σκεφτείτε μέχρι πού χρειάζεται να δοκιμάσετε διαιρέτες (δείτε την αντίστοιχη άσκηση).

§15.21 Η μακροεντολή `PROD`

Εφαρμόζει: «Η οδηγία `#define`*»,* `gcc -E`. Τι επιστρέφει το παρακάτω πρόγραμμα;

```
#define PROD 2*5
int main() {
  return 20 / PROD;
}
```

Η πρώτη σκέψη είναι $20/10 = 2$. Ο προεπεξεργαστής όμως αντικαθιστά *κείμενο*:

```
$ cpp prod.c
# 0 "prod.c"
# 0 "<built-in>"
# 0 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 0 "<command-line>" 2
# 1 "prod.c"


int main() {
  return 20 / 2*5;
}
$ gcc -o prod prod.c
```

```
$ ./prod
$ echo $?
50
```

Το `20 / 2*5` υπολογίζεται από αριστερά προς τα δεξιά: `(20 / 2) * 5 = 50`. Με `#define PROD (2*5)` το αποτέλεσμα θα ήταν 2. Την τιμή που επιστρέφει η `main` τη βλέπουμε με `echo $?` (Κεφάλαιο 1).

Αν σβήσουμε τη γραμμή `#define` και δώσουμε την τιμή από τη γραμμή εντολών:

```
$ gcc -DPROD=10 -o prod prod.c
$ ./prod
$ echo $?
2
```

§15.22 `#if 0` / `#else`

Εφαρμόζει: «Μεταγλώττιση υπό συνθήκη». Σε τι θα προεπεξεργαστεί το πρόγραμμα;

```
int main() {
#if 0
  return 42;
#else
  return 1;
#endif
}
```

```
$ gcc -E example.c
# 0 "example.c"
# 0 "<built-in>"
# 0 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 0 "<command-line>" 2
# 1 "example.c"
int main() {



  return 1;

}
```

Το `return 42;` δεν υπάρχει καν στον κώδικα που φτάνει στον μεταγλωττιστή· στη θέση των οδηγιών και του κομματιού που αφαιρέθηκε μένουν κενές γραμμές, ώστε οι αριθμοί γραμμών στα μηνύματα λάθους να ταιριάζουν με το αρχικό αρχείο.

§15.23 Μηνύματα αποσφαλμάτωσης με `#ifdef DEBUG`

Εφαρμόζει: «Οι οδηγίες `#ifdef` *και* `#ifndef`*».*

```
#define DEBUG
#ifdef DEBUG
  printf("debugging is on\n");
#else
  printf("debugging is off\n");
#endif
#ifndef DEBUG
  printf("optimizations are on\n");
#endif
```

Με το `#define DEBUG` στην αρχή τυπώνεται μόνο `debugging is on`. Αν σβήσουμε αυτή τη γραμμή, τυπώνονται `debugging is off` και `optimizations are on`. Αντί να αλλάζουμε τον κώδικα, μπορούμε να αφήσουμε έξω το `#define DEBUG` και να μεταγλωττίζουμε με `gcc -DDEBUG` όταν θέλουμε τα μηνύματα.

Κύρια σημεία

1. Η πολυπλοκότητα μετρά την απόδοση ενός αλγορίθμου ως συνάρτηση του μεγέθους n του προβλήματος, σε χρόνο εκτέλεσης και σε χώρο μνήμης.
2. $g = O(f)$ σημαίνει ότι από κάποιο n_0 και μετά $g(n) < c \cdot f(n)$· το O είναι άνω όριο, το Ω κάτω όριο και το Θ τάξη μεγέθους.
3. Οι σταθερές και οι όροι χαμηλότερης τάξης δεν μετράνε: $n/2$ και $3n + 5$ είναι και τα δύο $O(n)$.
4. Από την ταχύτερη στην πιο αργή: $O(1), O(\log n), O(\sqrt{n}), O(n), O(n \log n), O(n^2), O(n^3), O(2^n), O(n!), O(n^n)$.
5. Ένας βρόχος n επαναλήψεων είναι $O(n)$, δύο εμφωλευμένοι $O(n^2)$, ένας βρόχος που διαιρεί με 10 είναι $O(\log n)$.
6. Η αναδρομή κοστίζει και σε μνήμη: βάθος n σημαίνει $O(n)$ χώρο στη στοίβα· το αναδρομικό Fibonacci είναι $O(2^n)$ σε χρόνο.
7. Η `strlen` είναι $O(n)$ και η `strcmp` $O(\min(m, n))$.
8. Ένα καλύτερο σκεπτικό αλλάζει την κλάση: το άθροισμα τέλειων τετραγώνων πέφτει από $O(n)$ σε $O(\sqrt{n})$.
9. Ο προεπεξεργαστής είναι το πρώτο στάδιο του `gcc`· μετασχηματίζει το κείμενο του προγράμματος πριν από τη μεταγλώττιση, και την έξοδό του τη βλέπουμε με `gcc -E` ή `cpp`.
10. Οι οδηγίες του αρχίζουν με `#`: `#include`, `#define`, `#if`/`#else`/`#elif`/ `#endif`, `#ifdef`/`#ifndef`.
11. Το `#include <...>` ψάχνει στους φακέλους του συστήματος και του `-I`· το `#include "..."` πρώτα στον φάκελο του πηγαίου αρχείου.
12. Μια μακροεντολή αντικαθίσταται ως κείμενο, γι' αυτό χρειάζεται παρενθέσεις: με

`#define PROD 2*5` το `20 / PROD` κάνει 50.

13. Με `-DMACRO=VALUE` δίνουμε τιμή σε μακροεντολή από τη γραμμή εντολών, και με `#ifdef` / `#ifndef` ενεργοποιούμε κώδικα, π.χ. μηνύματα αποσφαλμάτωσης.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
πολυπλοκότητα	complexity	Μέτρο απόδοσης ενός αλγορίθμου ως συνάρτηση του μεγέθους του προβλήματος.
χρονική / χωρική πολυπλοκότητα	time / space complexity	Πώς αυξάνεται ο χρόνος / η μνήμη με το n.
άνω όριο	upper bound, Big-O	$g = O(f)$: η g δεν ξεπερνά τη $c \cdot f$ για μεγάλα n.
κάτω όριο	lower bound, Ω	$g = \Omega(f)$: η g είναι τουλάχιστον $c \cdot f$ για μεγάλα n.
τάξη μεγέθους	order of growth, Θ	Ισχύουν ταυτόχρονα O και Ω.
σταθερή / λογαριθμική / γραμμική	constant / logarithmic / linear	$O(1)$ / $O(\log n)$ / $O(n)$.
τετραγωνική / εκθετική	quadratic / exponential	$O(n^2)$ / $O(2^n)$.
μεταγλωττιστής	compiler	Μετατρέπει πηγαίο κώδικα σε κώδικα μηχανής.
προεπεξεργαστής	preprocessor	Πρώτο στάδιο του μεταγλωττιστή· μετασχηματίζει το κείμενο του κώδικα.
οδηγία προεπεξεργαστή	preprocessor directive	Γραμμή που αρχίζει με `#`, π.χ. `#include`.
αρχείο επικεφαλίδας	header file	Αρχείο `.h` με δηλώσεις, που εισάγεται με `#include`.
μακροεντολή	macro	Όνομα (με ή χωρίς παραμέτρους) που αντικαθίσταται με κείμενο.
μεταγλώττιση υπό συνθήκη	conditional compilation	Κράτημα ή αφαίρεση κώδικα με `#if` / `#ifdef`.

Διάβασμα

- **Διαφάνειες**: Διάλεξη 15, σελ. 1–54: πολυπλοκότητα και Big-O 5–12· παραδείγματα ανάλυσης 13–39· μεταγλωττιστές και προεπεξεργαστής 40–43· οδηγίες 44–52.
- **Σημειώσεις**: η διάλεξη ζητά τις σελ. 154–159 και 161 των διαφανειών του κ. Σταματόπουλου:

  - Κεφάλαιο 10: Ο προεπεξεργαστής της C, ενότητα «Ο προεπεξεργαστής της C» (K04, σελ. 154–159).
  - Κεφάλαιο 11: Ταξινόμηση και αναζήτηση, ενότητα «Ταξινόμηση πινάκων», οι παράγραφοι για την πολυπλοκότητα χρόνου και τον συμβολισμό O (K04, σελ. 161).
- **Εργαστήριο**: Εργαστήριο 6: `sieve.c` (μέγεθος πίνακα με `#define`, πρώτοι αριθμοί)· Εργαστήριο 10: Παράρτημα «Οργάνωση προγράμματος σε πολλαπλά αρχεία» (αρχεία επικεφαλίδας, include guards).
- **Άλλα**: Big O notation και C preprocessor (Wikipedia)· `man cpp`, `man gcc` (επιλογές `-E`, `-D`, `-I`).

Συχνά λάθη

- **Μέτρηση του χρόνου σε δευτερόλεπτα αντί για βήματα**. «Τρέχει σε 0,1 s, άρα είναι γρήγορο» δεν λέει τίποτα για n δέκα φορές μεγαλύτερο. Μετρήστε πώς αυξάνονται οι επαναλήψεις με το n.
- **«Ο βρόχος με `i += 2` είναι $O(n/2)$»**. Οι σταθερές παραλείπονται: είναι $O(n)$.
- **Ξεχνάτε τη στοίβα στην αναδρομή**. Το `factorial` δεν έχει πίνακα, αλλά χρειάζεται $O(n)$ μνήμη για τις n ενεργές κλήσεις.
- **Κλήση $O(n)$ μέσα σε βρόχο**. `for (i = 0; i < strlen(s); i++)` καλεί την `strlen` σε κάθε επανάληψη, άρα γίνεται $O(n^2)$. Υπολογίστε το μήκος μία φορά πριν από τον βρόχο.
- **Μακροεντολή χωρίς παρενθέσεις**. `#define PROD 2*5` και `20 / PROD` δίνει 50, όχι 2· `#define SQUARE(X) X * X` και `SQUARE(a + 1)` δίνει `a + 1 * a + 1`. Γράψτε `(2*5)` και `((X) * (X))`.
- **Παράμετρος με παρενέργεια σε μακροεντολή**. `MAX(i++, j++)` αυξάνει μία μεταβλητή δύο φορές. Μην περνάτε `++`, `--` ή κλήσεις συναρτήσεων σε μακροεντολές.
- **`;` στο τέλος του `#define`**. `#define N 10;` κάνει το `int a[N];` να γίνει `int a[10;];` και ο `gcc` βγάζει συντακτικό λάθος σε γραμμή που φαίνεται σωστή. Κοιτάξτε την έξοδο του `gcc -E`.
- `undefined reference to 'sqrt'`. Οι συναρτήσεις της `math.h` χρειάζονται `-lm` στη μεταγλώττιση: `gcc squares.c -o squares -lm`.
- **Το `i = sqrt(low)` κόβει προς τα κάτω** και προσθέτει τετράγωνο εκτός διαστήματος όταν το `low` δεν είναι τέλειο τετράγωνο (π.χ. `low = 5` προσθέτει το 4).
- `#include "stdio.h"` **αντί για** `<stdio.h>` (δουλεύει, αλλά ψάχνει πρώτα στον φάκελό σας) ή `#include <myfile.h>` για δικό σας αρχείο (`fatal error: myfile.h: No such file or directory`, εκτός αν δώσετε `-I.`).

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K15.8** Πολυπλοκότητα O(n² + log n) (36% σωστές): Το 60% απάντησε False, θεωρώντας

ότι το O(·) πρέπει να περιέχει έναν μόνο όρο· η έκφραση είναι σωστή, απλώς ισοδυναμεί με O(n²), γιατί ο κυρίαρχος όρος είναι το n².

- **K15.6** Πολυπλοκότητα μέτρησης ψηφίων (38% σωστές): Το 45% επέλεξε O(n), μπερδεύοντας την τιμή του n με το μέγεθος της εισόδου: ο αλγόριθμος κάνει μία επανάληψη ανά ψηφίο, όχι ανά μονάδα της τιμής.
- **K15.7** Η έξοδος του προεπεξεργαστή (38% σωστές): Το 30% απάντησε «Εξαρτάται», χωρίς να έχει ξεκαθαρίσει τη θέση του προεπεξεργαστή στη σειρά προεπεξεργασία → μεταγλώττιση → συμβολομετάφραση → σύνδεση: ο προεπεξεργαστής κάνει πάντα μόνο αντικαταστάσεις κειμένου.
- **K15.5** Πολυπλοκότητα μέτρησης bit (43% σωστές): Το 32% επέλεξε O(n), μπερδεύοντας την τιμή του n με το πλήθος των bit του, που είναι περίπου $\log_2 n$.
- **K15.4** Πολυπλοκότητα πολλαπλασιασμού πινάκων (49% σωστές): Το 23% επέλεξε O(n²), μετρώντας μόνο τα N² στοιχεία του αποτελέσματος και όχι το εσωτερικό άθροισμα N γινομένων για το καθένα.

Ερωτήσεις κατανόησης

- **E15.1** Ποιες είναι οι δύο μετρικές της πολυπλοκότητας, και γιατί δεν μετράμε τον χρόνο σε δευτερόλεπτα;[120]
- **E15.2** Ισχύει ότι $5n^2 + 3n = O(n^2)$; Ισχύει ότι $5n^2 + 3n = O(n)$;[121]
- **E15.3** Τι χρονική και τι χωρική πολυπλοκότητα έχουν δύο εμφωλευμένοι βρόχοι από 0 ως n με σώμα $O(1)$;[122]
- **E15.4** Γιατί το αναδρομικό `factorial` έχει χωρική πολυπλοκότητα $O(n)$ ενώ ο επαναληπτικός υπολογισμός έχει $O(1)$;[123]
- **E15.5** Σε τι επεκτείνεται το `MAX(a, b + 1)` με τον ορισμό της διάλεξης;[124]
- **E15.6** Ποια η διαφορά ανάμεσα σε `#include <file.h>` και `#include "file.h"`;[125]
- **E15.7** Πώς θα μεταγλωττίσετε το πρόγραμμα με το `#ifdef DEBUG` (χωρίς τη γραμμή `#define DEBUG`) ώστε να τυπώσει `debugging is on`;[126]

Kahoot από το αμφιθέατρο (K15.1–K15.8)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K15.1** Γραμμική αναζήτηση σε αμφιθέατρο n × n: 75% σωστές απαντήσεις

[120] Ο χρόνος εκτέλεσης και ο χώρος μνήμης. Τα δευτερόλεπτα εξαρτώνται από το μηχάνημα· μας ενδιαφέρει πώς αυξάνονται τα βήματα με το n.

[121] Ναι, με $c = 6$ για $n > 3$. Όχι· το n^2 ξεπερνά κάθε $c \cdot n$.

[122] Χρόνος $O(n^2)$, χώρος $O(1)$.

[123] Κάθε αναδρομική κλήση κρατά πλαίσιο στη στοίβα, και οι $n + 1$ κλήσεις είναι ενεργές ταυτόχρονα· ο βρόχος χρησιμοποιεί σταθερό πλήθος μεταβλητών.

[124] `((a) > (b + 1) ? (a) : (b + 1))`.

[125] Η μορφή με `< >` ψάχνει στους φακέλους του συστήματος και του `-I`· η μορφή με `" "` ψάχνει πρώτα στον φάκελο του πηγαίου αρχείου.

[126] `gcc -DDEBUG prog.c -o prog`.

- **K15.2** Ορισμός μακροεντολής: 74% σωστές απαντήσεις
- **K15.3** Τι σημαίνει O(n): 62% σωστές απαντήσεις
- **K15.4** Πολυπλοκότητα πολλαπλασιασμού πινάκων: 49% σωστές απαντήσεις
- **K15.5** Πολυπλοκότητα μέτρησης bit: 43% σωστές απαντήσεις
- **K15.6** Πολυπλοκότητα μέτρησης ψηφίων: 38% σωστές απαντήσεις
- **K15.7** Η έξοδος του προεπεξεργαστή: 38% σωστές απαντήσεις
- **K15.8** Πολυπλοκότητα $O(n^2 + \log n)$: 36% σωστές απαντήσεις

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A15.1–A15.15)

- **A15.1** Πολυπλοκότητα της atoi: Διάλεξη 15, διαφάνεια 15 · ★☆☆ · short-answer · `slides-lec15-complexity-atoi`
- **A15.2** Πολυπλοκότητα εύρεσης μέγιστου σε πίνακα N x N: Διάλεξη 15, διαφάνεια 23 · ★☆☆ · short-answer · `slides-lec15-complexity-find-max-2d`
- **A15.3** Πολυπλοκότητα υπολογισμού βαθμολογίας: Διάλεξη 15, διαφάνεια 21 · ★☆☆ · short-answer · `slides-lec15-complexity-grade`
- **A15.4** Πολυπλοκότητα δυναμικού πίνακα με malloc: Διάλεξη 15, διαφάνεια 19 · ★☆☆ · short-answer · `slides-lec15-complexity-malloc`
- **A15.5** Πολυπλοκότητα: γινόμενο περιττών πολλαπλασίων του 7: Διάλεξη 15, διαφάνεια 13 · ★☆☆ · short-answer · `slides-lec15-complexity-odd-multiples-of-7`
- **A15.6** Πολυπλοκότητα της strlen: Διάλεξη 15, διαφάνεια 29 · ★☆☆ · short-answer · `slides-lec15-complexity-strlen`
- **A15.7** Τι κάνει το πρόγραμμα με την getchar: Διάλεξη 15, διαφάνεια 17 · ★☆☆ · trace · `slides-lec15-getchar-count`
- **A15.8** Τι επιστρέφει το πρόγραμμα με τη μακροεντολή PROD: Διάλεξη 15, διαφάνεια 47 · ★☆☆ · trace · `slides-lec15-macro-prod`
- **A15.9** Σε τι προεπεξεργάζεται το #if 0: Διάλεξη 15, διαφάνεια 50 · ★☆☆ · trace · `slides-lec15-preprocess-if-else`
- **A15.10** Πολυπλοκότητα του αναδρομικού παραγοντικού: Διάλεξη 15, διαφάνεια 25 · ★★☆ · short-answer · `slides-lec15-complexity-factorial`
- **A15.11** Πολυπλοκότητα του αναδρομικού Fibonacci: Διάλεξη 15, διαφάνεια 27 · ★★☆ · short-answer · `slides-lec15-complexity-fibonacci`
- **A15.12** Πολυπλοκότητα εύρεσης κατόπτρου ακεραίου: Διάλεξη 15, διαφάνεια 37 · ★★☆ · short-answer · `slides-lec15-complexity-mirror`
- **A15.13** Άθροισμα τέλειων τετραγώνων: δύο εκδοχές: Διάλεξη 15, διαφάνεια 33 · ★★☆ · short-answer · `slides-lec15-complexity-perfect-squares`
- **A15.14** Πολυπλοκότητα της strcmp: Διάλεξη 15, διαφάνεια 31 · ★★☆ · short-answer · `slides-lec15-complexity-strcmp`
- **A15.15** Πολυπλοκότητα ελέγχου αν ένας αριθμός είναι πρώτος: Διάλεξη 15, διαφάνεια 39 · ★★☆ · short-answer · `slides-lec15-prime-complexity`

Εργασίες (A15.16–A15.17)

- **A15.16** Κατοπτρικά Πρώτα Τετράγωνα: Εργασία 1 (2023-24), Άσκηση 2 · ★★★ · programming · `hw-2023-hw1-mirror`
- **A15.17** Παραγοντοποίηση ημιπρώτων (factor): Εργασία 1 (2024-25), Άσκηση 3 (Bonus) · ★★★ · programming · `hw-2024-hw1-factor`

Θέματα εξετάσεων (A15.18)

- **A15.18** Δίδυμοι Πρώτοι: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #7 (Star Wars Themed), Θέμα 3 · ★★★ · programming · `exam-2023-fall-ex7-q3`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A11.16** Η ακολουθία Fibonacci: Εργαστήριο 5, Άσκηση 2 · ★★☆ · programming · `lab-lab05-fib`
- **A14.21** Συνένωση Αλφαριθμητικών - join: Εξέταση Ιανουαρίου 2025, Θέμα 5 · ★★☆ · programming · `exam-2025-jan-q5`
- **A17.12** Εύρεση μηδενός σε πίνακα: Εξέταση Σεπτεμβρίου 2024, Θέμα 3 · ★★☆ · programming · `exam-2024-sep-q3`
- **A21.8** Αντιστροφή λίστας: Εξέταση Σεπτεμβρίου 2024, Θέμα 5 · ★★☆ · programming · `exam-2024-sep-q5`
- **A21.9** Μεσαίο Στοιχείο Λίστας: Εξέταση Σεπτεμβρίου 2025, Θέμα 4 · ★★☆ · programming · `exam-2025-sep-q4`
- **A21.10** N-οστό Στοιχείο Λίστας: Εξέταση Σεπτεμβρίου 2026, Θέμα 4 · ★★☆ · programming · `exam-2026-sep-q4`
- **A22.19** Reverse Inorder Traversal: Εξέταση Ιουλίου 2024, Θέμα 4 · ★★☆ · programming · `exam-2024-jul-q4`
- **A22.18** Αθροιστής Δέντρων - sumtree: Εξέταση Ιανουαρίου 2025, Θέμα 4 · ★☆☆ · programming · `exam-2025-jan-q4`
- **A22.20** Διερμηνέας Αριθμητικών Εκφράσεων - eval: Εξέταση Ιανουαρίου 2026, Θέμα 4 · ★★☆ · programming · `exam-2026-jan-q4`
- **A25.9** Το Νερό Νεράκι: Κατατακτήριες Δεκεμβρίου 2023, Θέμα 4 · ★★★ · programming · `exam-2023-dec-q4`
- **A25.13** Βέλτιστη Μοιρασιά Πίτσας: Εξέταση Ιουλίου 2024, Θέμα 3 · ★★★ · programming · `exam-2024-jul-q3`
- **A25.8** Επενδύσεις στο Χρηματιστήριο: Εξέταση Ιουνίου 2026, Θέμα 3 · ★★☆ · programming · `exam-2026-jun-q3`
- **A25.17** Η Μεγαλύτερη Χωρητικότητα - capacity: Εξέταση Σεπτεμβρίου 2026, Θέμα 5 · ★★★ · programming · `exam-2026-sep-q5`

Μέρος Δ: Αλγόριθμοι και δομές δεδομένων

Κεφάλαιο 16: Επίλυση Προβλημάτων #2

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να εκτιμάτε τη χρονική και τη χωρική πολυπλοκότητα μιας μικρής συνάρτησης· να ανταλλάσσετε χρόνο με μνήμη (ή το αντίστροφο) όταν ψάχνετε διπλότυπα σε πίνακα· να χειρίζεστε τα ψηφία ενός αριθμού με `/` και `%`· να ξεχωρίζετε τα `char *array[]`, `char **array` και `char array[10][10]`· να «επιστρέφετε» πολλές τιμές μέσω δεικτών· να βγαίνετε από εμφωλευμένους βρόχους· να φτιάχνετε δισδιάστατο πίνακα στον σωρό· να γράφετε αποδοτική αναδρομική Fibonacci· και να μετράτε μνήμη και να εξετάζετε δυαδικά αρχεία για την Εργασία #1.

Προαπαιτούμενα: Κεφάλαιο 10, Κεφάλαιο 11, Κεφάλαιο 12, Κεφάλαιο 13, Κεφάλαιο 15

Χρόνος μελέτης: ~2,5 ώρες

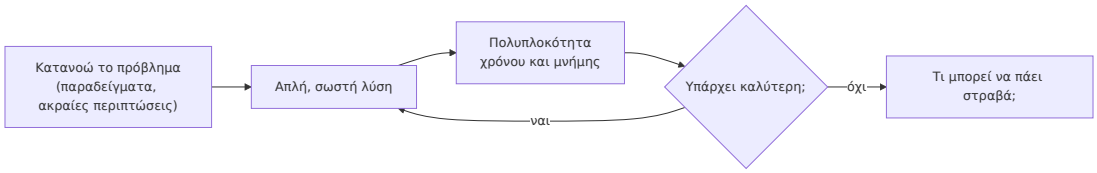
Σύνοψη

Η διάλεξη αυτή είναι μια διάλεξη εξάσκησης, με την ίδια λογική με το Κεφάλαιο 7: δεν εισάγει νέα στοιχεία της γλώσσας, αλλά θέτει μια σειρά από μικρά προβλήματα «Θέλω μια συνάρτηση που ... Πώς;» και τα λύνει ζωντανά, με εθελοντές από το ακροατήριο. Τα προβλήματα ανακυκλώνουν όλη την ύλη ως τώρα: πίνακες, ψηφία αριθμών, συμβολοσειρές, δείκτες, δισδιάστατους πίνακες, τον σωρό και την αναδρομή. Το νέο στοιχείο είναι η ερώτηση που συνοδεύει σχεδόν κάθε πρόβλημα: «Χρονική και χωρική πολυπλοκότητα;» Μετά το Κεφάλαιο 15 δεν αρκεί μια λύση που δουλεύει· θέλουμε να ξέρουμε πόσο κοστίζει και αν υπάρχει καλύτερη. Η διάλεξη κλείνει με πρακτικές απαντήσεις σε ερωτήσεις για την Εργασία #1.

Θεωρία

§16.1 Γιατί εξασκούμαστε στην επίλυση προβλημάτων

Η επίλυση προβλημάτων είναι δεξιότητα, όχι γνώση που αποστηθίζεται: το «δεν δουλεύει, γιατί;» και το εξίσου ύπουλο «δουλεύει, γιατί;» απαντώνται μόνο με εξάσκηση. Οι διαφάνειες δίνουν τέσσερις λόγους: **εξάσκηση**, **γνώση** (κάθε λυμένο πρόβλημα γίνεται μοτίβο για το επόμενο), **εφαρμογές** (τα ίδια μοτίβα εμφανίζονται σε εργασίες, εξετάσεις και πραγματικό κώδικα) και **δημιουργία / σύνθεση ιδεών**. Γι' αυτό πολλά προβλήματα της διάλεξης είναι γνωστά από προηγούμενα κεφάλαια· η αξία είναι να τα λύνετε γρήγορα και να τα αναλύετε.



Σχήμα: η σειρά ερωτήσεων που θέτει η διάλεξη για κάθε πρόβλημα.

§16.2 Χρονική και χωρική πολυπλοκότητα μιας συνάρτησης

Η **χρονική πολυπλοκότητα** (time complexity) μετρά πώς μεγαλώνει το πλήθος των βημάτων όσο μεγαλώνει η είσοδος n· η **χωρική πολυπλοκότητα** (space complexity) μετρά πόση *επιπλέον* μνήμη χρειάζεται, πέρα από την είσοδο. Και οι δύο γράφονται με τον συμβολισμό O του Κεφαλαίου 15. Πρακτικοί κανόνες:

- Ένα πέρασμα από τα n στοιχεία με σταθερή δουλειά ανά βήμα: $O(n)$ χρόνος. Δύο εμφωλευμένοι βρόχοι πάνω στα n: $O(n^2)$.
- Λίγες βοηθητικές μεταβλητές: $O(1)$ μνήμη· ένας βοηθητικός πίνακας n θέσεων: $O(n)$.
- Η αναδρομή κοστίζει μνήμη: κάθε ενεργή κλήση έχει το stack frame της (Κεφάλαιο 13), άρα βάθος n σημαίνει $O(n)$ μνήμη.
- Αναλύουμε τη **χειρότερη περίπτωση** (worst case). Η σειριακή αναζήτηση μπορεί να βρει το στοιχείο με την πρώτη, αλλά όταν λείπει ελέγχει και τα n: είναι $O(n)$.
- Με σταθερό μέγεθος (ακριβώς 100 στοιχεία) όλα είναι τυπικά $O(1)$· η ουσιαστική ερώτηση είναι πώς κλιμακώνεται η λύση όταν το 100 γίνει n.

§16.3 Χρόνος ή μνήμη: το αντιστάθμισμα

Συχνά μια λύση γίνεται ταχύτερη αν ξοδέψει μνήμη, ή το αντίστροφο: αυτό είναι το **αντιστάθμισμα χρόνου–μνήμης** (time–space tradeoff). Το «βρες το στοιχείο που εμφανίζεται δύο φορές» το δείχνει καθαρά:

Ιδέα	Χρόνος	Μνήμη
Σύγκριση κάθε ζεύγους με δύο εμφωλευμένους βρόχους	$O(n^2)$	$O(1)$
Ταξινόμηση και σύγκριση γειτονικών στοιχείων	$O(n \log n)$	ανάλογα με την ταξινόμηση
Πίνακας «το έχω δει» με μία θέση ανά δυνατή τιμή	$O(n)$	$O(k)$, k το εύρος των τιμών

Η τρίτη ιδέα δουλεύει μόνο όταν οι τιμές έχουν μικρό, γνωστό εύρος (π.χ. 0–999): για τυχαίους `int` θα χρειαζόταν 2^{32} θέσεις. Η ταξινόμηση έρχεται στο Κεφάλαιο 17.

Μερικές φορές μια ιδιότητα του προβλήματος δίνει και τα δύο. Όταν «όλα τα στοιχεία είναι διπλά εκτός από ένα», αρκεί το **αποκλειστικό Ή** (XOR, `^`) του Κεφαλαίου 4, χάρη στις ιδιότητες

$$x \oplus x = 0, \qquad x \oplus 0 = x, \qquad x \oplus y = y \oplus x$$

Το XOR όλων των στοιχείων μηδενίζει κάθε ζευγάρι, όποια σειρά κι αν έχουν, και αφήνει μόνο το μοναδικό: $O(n)$ χρόνος, $O(1)$ μνήμη.

§16.4 Τα ψηφία ενός αριθμού

Για μη αρνητικό ακέραιο `n`, το `n % 10` είναι το **τελευταίο ψηφίο** και το `n / 10` ο αριθμός **χωρίς** αυτό (123 ⟶ 3 και 12). Επαναλαμβάνοντας μέχρι `n == 0` παίρνουμε τα ψηφία από δεξιά προς τα αριστερά. Το αντίστροφο μοτίβο, `r = 10 * r + digit`, σπρώχνει ό,τι έχουμε μία θέση αριστερά και προσθέτει ψηφίο στις μονάδες. Τα δύο μαζί δίνουν την αντιστροφή ψηφίων, και από εκεί τον έλεγχο **παλινδρομικού** αριθμού (palindrome: διαβάζεται ίδια και από τις δύο μεριές, π.χ. 12321). Ο βρόχος κάνει τόσα βήματα όσα τα ψηφία, δηλαδή $O(\log n)$.

Δύο προσοχές: ο αντεστραμμένος αριθμός μπορεί να μη χωρά σε `int` (το 1000000009 χωρά, το 9000000001 όχι)· και για αρνητικό `n` το `n % 10` είναι αρνητικό στη C (το πρόσημο ακολουθεί τον διαιρετέο), οπότε το -45 γίνεται -54.

§16.5 Από χαρακτήρες σε αριθμό: η `atoi` και τα όριά της

Το ίδιο μοτίβο `10 * result + digit`, με `digit = c - '0'`, μετατρέπει μια συμβολοσειρά ψηφίων σε ακέραιο, σε $O(n)$ χρόνο και $O(1)$ μνήμη (Κεφάλαιο 11). Στο «Τι μπορεί να πάει στραβά;» η απάντηση είναι οι υποθέσεις που ο καλών μπορεί να παραβιάσει:

- **Μη ψηφία.** Το `"-5"` ή το `"12a"` δίνουν σκουπίδια χωρίς ένδειξη λάθους, και ο καλών δεν ξεχωρίζει το `"0"` από μια αποτυχία.
- **Υπερχείλιση.** Πάνω από `INT_MAX` (2147483647) ο αριθμός δεν χωρά· η υπερχείλιση προσημασμένου ακεραίου είναι απροσδιόριστη συμπεριφορά.
- **Χωρίς `'\0'`.** Ο βρόχος διαβάζει εκτός ορίων.
- **Όνομα.** Η `atoi` υπάρχει ήδη στη `stdlib.h` και οι δύο συγκρούονται.

§16.6 Λίγα μαθηματικά αντί για εξαντλητική αναζήτηση

Για το άθροισμα των τέλειων τετραγώνων στο $[a, b]$, η πρώτη ιδέα ελέγχει κάθε αριθμό του διαστήματος: $O(b-a)$ έλεγχοι. Όμως έως το b υπάρχουν μόνο $\lfloor \sqrt{b} \rfloor$ τετράγωνα· αν διατρέξουμε τα $k = 1, 2, \ldots$ και προσθέσουμε τα k^2 του διαστήματος, κάνουμε $O(\sqrt{b})$ βήματα: για $b = 10^{12}$, ένα εκατομμύριο αντί για ένα τρισεκατομμύριο. Ο τύπος

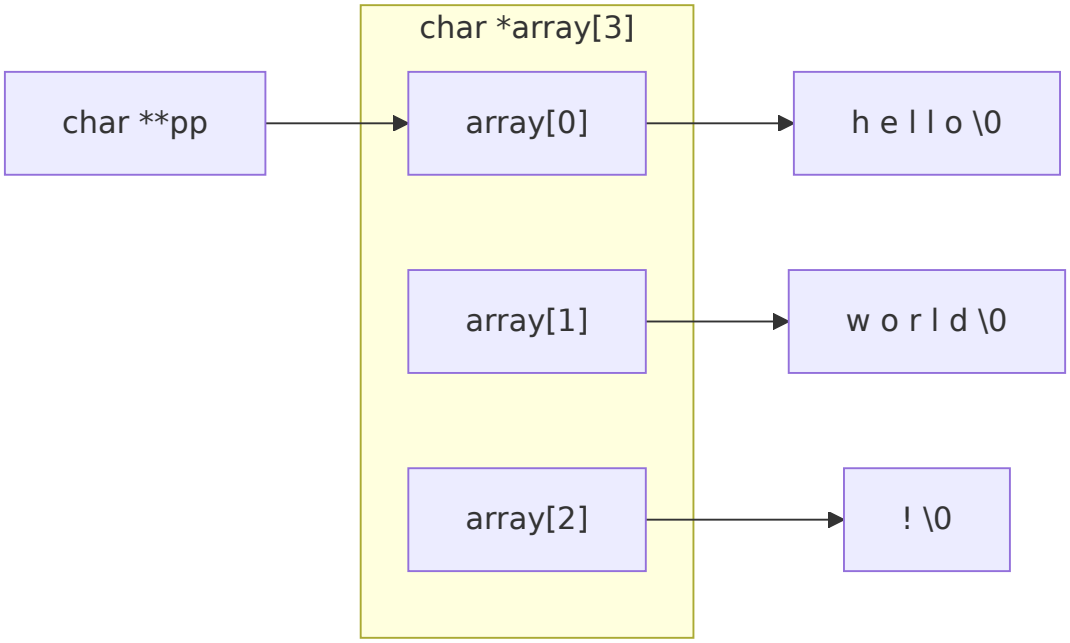
$$1^2 + 2^2 + \cdots + m^2 = \frac{m(m+1)(2m+1)}{6}$$

δίνει την απάντηση σε σταθερό χρόνο, ως διαφορά των αθροισμάτων για $m = \lfloor \sqrt{b} \rfloor$ και $m = \lfloor \sqrt{a-1} \rfloor$. Το δίδαγμα: πριν γράψετε βρόχο, σκεφτείτε αν η δομή του προβλήματος σας γλιτώνει από το να επισκεφθείτε όλη την είσοδο. Τα αθροίσματα μεγαλώνουν γρήγορα, οπότε χρειάζεται `long long`.

§16.7 `char *array[]`, `char **array` και `char array[10][10]`

Και με τις τρεις γράφουμε `array[i][j]`, αλλά περιγράφουν διαφορετικά πράγματα στη μνήμη (Κεφάλαιο 12):

Δήλωση	Τι είναι	`sizeof` (64 bit)
`char *array[3]`	πίνακας από 3 pointers, ο καθένας προς δική του συμβολοσειρά	24
`char **array`	**μία** μεταβλητή δείκτη, προς ένα `char *`	8
`char array[10][10]`	100 συνεχόμενοι χαρακτήρες σε 10 γραμμές· κανένας pointer	100



Σχήμα: πίνακας από pointers και ένας `char **` *προς το πρώτο στοιχείο του· ο* `char array[10][10]` *δεν έχει βέλη, μόνο 100 συνεχόμενα bytes.*

- Ο `char *array[]` σε έκφραση γίνεται δείκτης στο πρώτο του στοιχείο, δηλαδή `char **`. Γι' αυτό, **ως παράμετρος**, το `char *argv[]` και το `char **argv` είναι το ίδιο.
- Ο `char array[10][10]` γίνεται δείκτης στην πρώτη **γραμμή**, τύπου `char (*)[10]`, **όχι** `char **`. Η θέση του `array[i][j]` υπολογίζεται (`i * 10 + j`), ενώ στον `char *array[]`

διαβάζεται πρώτα ο pointer `array[i]`. Σε συνάρτηση περνά ως `char array[][10]`, με γνωστό το πλήθος των στηλών.

§16.8 Πολλές τιμές από μία συνάρτηση: δείκτες ως ορίσματα

Μια συνάρτηση επιστρέφει με `return` **μία** τιμή, και τα ορίσματα περνούν **κατά τιμή** (call by value): η συνάρτηση παίρνει αντίγραφα. Γι' αυτό μια `void swap(int x, int y)` ανταλλάσσει τα αντίγραφα και δεν αλλάζει τίποτα στον καλούντα. Η λύση είναι να περάσουμε τις **διευθύνσεις** (`swap(&a, &b)`) και η συνάρτηση να γράψει μέσω των δεικτών (`*a = ...`).

Με το ίδιο μοτίβο μια συνάρτηση «επιστρέφει» όσες τιμές θέλει: ο καλών δίνει μια διεύθυνση για κάθε αποτέλεσμα (**παράμετρος εξόδου**, output parameter), και το `return` μένει ελεύθερο για να πει αν η κλήση πέτυχε, όπως στη `scanf`. Έτσι γράφεται η `get_two_chars`. Εναλλακτικά ο καλών δίνει έναν πίνακα `char out[2]` να γεμίσει· οι δομές (`struct`), που ομαδοποιούν πολλές τιμές, έρχονται στο Κεφάλαιο 19.

§16.9 Έξοδος από εμφωλευμένους βρόχους

Η αναζήτηση σε δισδιάστατο πίνακα θέλει δύο εμφωλευμένους βρόχους, $O(\text{γραμμές} \cdot \text{στήλες})$. Η παγίδα είναι η έξοδος: το `break` βγάζει **μόνο από τον εσωτερικό** βρόχο (Κεφάλαιο 8), οπότε ο εξωτερικός συνεχίζει και το `"yes"` μπορεί να τυπωθεί πολλές φορές. Τρεις σωστοί τρόποι:

1. **Συνάρτηση με** `return`, που βγαίνει από όλους τους βρόχους μαζί. Ο πιο καθαρός.
2. **Σημαία** (flag) `found` στις συνθήκες και των δύο βρόχων.
3. `goto` σε ετικέτα μετά τους βρόχους: μία από τις λίγες αποδεκτές χρήσεις της.

§16.10 Δισδιάστατος πίνακας στον σωρό

Όταν οι διαστάσεις γίνονται γνωστές μόνο στην εκτέλεση, ο πίνακας φτιάχνεται στον σωρό, με δύο τρόπους:

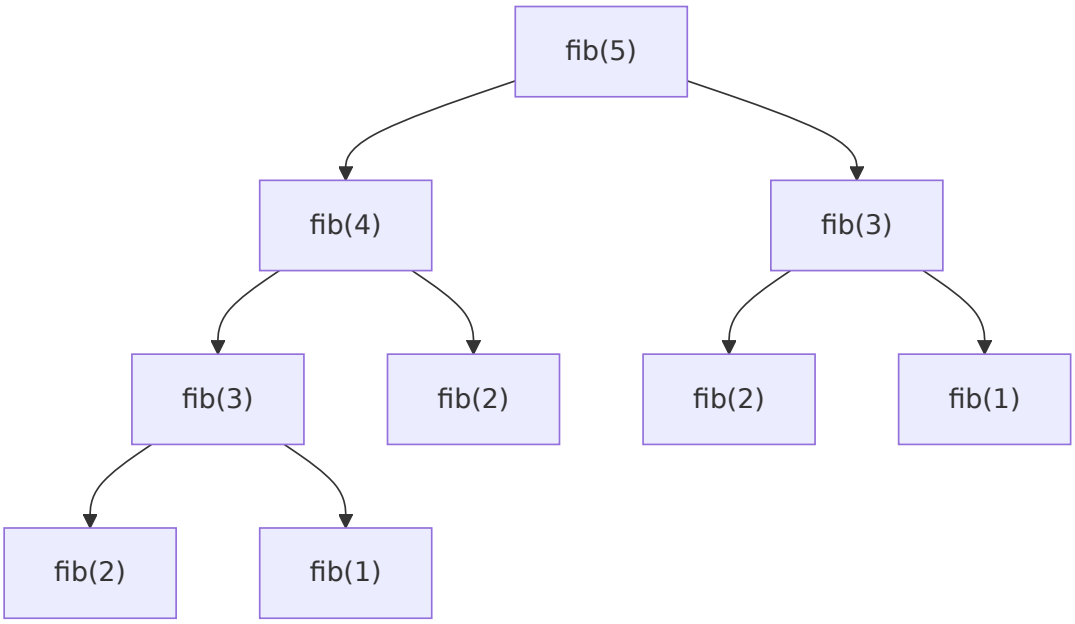
- `int **` **με μία** `malloc` **ανά γραμμή**, όπως στο Κεφάλαιο 13: γράφουμε `grid[i][j]`, αλλά χρειάζονται $M + 1$ κλήσεις `malloc` και $M + 1$ `free` (πρώτα οι γραμμές), και οι γραμμές δεν είναι συνεχόμενες.
- **Ένα μπλοκ** `M × N`, με υπολογισμό της θέσης όπως κάνει ο μεταγλωττιστής για τους στατικούς πίνακες: πρώτα προσπερνάμε `i` γραμμές και μετά `j` στοιχεία.

```
int *grid = malloc(M * N * sizeof(int));   // + έλεγχος για NULL
grid[i * N + j] = 42;                      // το "grid[i][j]"
free(grid);                                // μία free για όλα
```

§16.11 Αναδρομή χωρίς επανυπολογισμούς

Η αναδρομική Fibonacci του ορισμού, `fib(n) = fib(n-1) + fib(n-2)`, είναι σωστή αλλά εκθετική: η `fib(n-1)` ξαναϋπολογίζει την `fib(n-2)` που υπολογίζει και ο δεύτερος κλάδος, σε

κάθε επίπεδο. Το πλήθος των κλήσεων μεγαλώνει όπως οι ίδιοι οι αριθμοί Fibonacci (περίπου $1{,}6^n$): για `fib(40)` πάνω από 300 εκατομμύρια.



Σχήμα: μέρος του δέντρου κλήσεων της απλής `fib(5)`· *η* `fib(3)` *υπολογίζεται δύο φορές και η* `fib(2)` *τρεις.*

Στο «Γίνεται αναδρομικά και αποδοτικά;» η απάντηση είναι ναι:

- **Απομνημόνευση (memoization)**: κρατάμε κάθε αποτέλεσμα σε πίνακα `memo[]` την πρώτη φορά και το επιστρέφουμε αμέσως στις επόμενες. $O(n)$ χρόνος και μνήμη.
- **Οι δύο τελευταίες τιμές ως ορίσματα**: κάθε κλήση κουβαλά τα F_k, F_{k+1} και κάνει μία μόνο αναδρομική κλήση, άρα μια αλυσίδα n κλήσεων.

Το ίδιο πρόβλημα είναι το ερώτημα 2.6 (`fib_rec_fast`) του `fib.c` στο Εργαστήριο 5.

§16.12 Εργαλεία για την Εργασία #1

Η Εργασία #1 επεξεργάζεται αρχεία wav, έχει όριο μνήμης και διαβάζει δυαδικά δεδομένα. Η διάλεξη απαντά σε τρεις ερωτήσεις γι' αυτήν:

- **Πόση μνήμη χρησιμοποιεί το πρόγραμμά μου**; Η `/usr/bin/time -v ./prog` (με όλη τη διαδρομή, αλλιώς τρέχει η ενσωματωμένη `time` του shell) τυπώνει τη γραμμή «Maximum resident set size (kbytes)», τη μέγιστη μνήμη της διεργασίας. Η `memusage ./prog` δείχνει τη χρήση του σωρού (peak και κλήσεις `malloc`/`free`).
- **Πώς διαβάζω δυαδικά δεδομένα**; Όχι με `cat`. Οι `xxd` και `hexdump -C` δείχνουν τα bytes σε δεκαεξαδική μορφή δίπλα στους χαρακτήρες· η `hexedit` τα επεξεργάζεται. Για σύγκριση με την αναμενόμενη έξοδο, η `cmp` αναφέρει το πρώτο byte που διαφέρει (και

τίποτα αν τα αρχεία είναι ίδια) και η `vbindiff` δείχνει δύο αρχεία δίπλα-δίπλα.

- **Πώς φτιάχνω υποεντολές;** Μια υποεντολή (subcommand), όπως το `info` στο `./soundwave info`, είναι απλώς το `argv[1]`: ελέγχουμε πρώτα το `argc` και μετά συγκρίνουμε με `strcmp` (0 σημαίνει ίσες) και καλούμε μια συνάρτηση ανά υποεντολή.

Παραδείγματα

§16.13 Προθέρμανση: μέσος όρος και αναζήτηση

Οι λύσεις των διαφανειών για τα δύο πρώτα προβλήματα, με πίνακα 100 ακεραίων:

```
int average(int grades[100]) {
  int i, sum = 0;
  for(i = 0; i < 100; i++) {
    sum += grades[i];
  }
  return sum / 100;
}

int find(int haystack[100], int needle) {
  int i;
  for(i = 0; i < 100; i++) {
    if (haystack[i] == needle) {
      return i;
    }
  }
  return -1;
}
```

Και οι δύο είναι $O(n)$ χρόνος και $O(1)$ μνήμη («Χρονική και χωρική πολυπλοκότητα»). Στην `average` το `sum / 100` είναι ακέραια διαίρεση (5,5 γίνεται 5)· για ακρίβεια επιστρέψτε `double` με `sum / 100.0`. Στη `find` το `return i` σταματά στην πρώτη εμφάνιση και το `-1` σημαίνει «δεν βρέθηκε», αφού δεν είναι ποτέ έγκυρη θέση. Σε ταξινομημένο πίνακα η δυαδική αναζήτηση του Κεφαλαίου 17 θα έκανε $O(\log n)$.

§16.14 Διπλό και μοναδικό στοιχείο

Για το «στοιχείο που υπάρχει δύο φορές» (ο `{8, 1, 5, 42, 7, 3, 42}` δίνει 42), η απλή λύση της πρώτης γραμμής του πίνακα της Θεωρίας, και για το «όλα διπλά εκτός από ένα» η λύση με XOR:

```
int find_duplicate(int a[], int n) {     // O(n^2) χρόνος, O(1) μνήμη
  for (int i = 0; i < n; i++)
    for (int j = i + 1; j < n; j++)
```

```
      if (a[i] == a[j])
        return a[i];
  return -1;
}

int find_single(int a[], int n) {        // O(n) χρόνος, O(1) μνήμη
  int x = 0;
  for (int i = 0; i < n; i++)
    x ^= a[i];
  return x;
}
```

Στη `find_duplicate` το `j` ξεκινά από `i + 1`, ώστε κάθε ζεύγος να ελεγχθεί μία φορά και κανένα στοιχείο να μη συγκριθεί με τον εαυτό του. Για τον `{4, 9, 2, 4, 7, 2, 9}` η `find_single` δίνει $(4 \oplus 4) \oplus (9 \oplus 9) \oplus (2 \oplus 2) \oplus 7 = 7$.

§16.15 Αντιστροφή ψηφίων και παλινδρομικοί αριθμοί

Με το μοτίβο «Τα ψηφία ενός αριθμού»:

```
int reverse(int n) {
  int rev = 0;
  while (n != 0) {
    rev = rev * 10 + n % 10;   // κόλλα το τελευταίο ψηφίο του n στο rev
    n /= 10;                   // και πέτα το από το n
  }
  return rev;
}

int is_palindrome(int n) {
  return n >= 0 && n == reverse(n);
}
```

Τα `reverse(123)`, `reverse(1200)` και `reverse(-45)` δίνουν 321, 21 και -54· το `is_palindrome(12321)` δίνει 1 και το `is_palindrome(1231)` 0. Το 1200 γίνεται 21 γιατί ένας ακέραιος δεν έχει αρχικά μηδενικά, κάτι που δεν πειράζει τον έλεγχο παλινδρομικού. Εναλλακτικά, βάλτε τα ψηφία σε πίνακα και συγκρίνετε με δύο δείκτες θέσης που κινούνται από τα άκρα προς τη μέση.

§16.16 Η `atoi`

Η λύση των διαφανειών:

```
int atoi(char digits[]) {
  int result = 0;
  for(int i = 0; digits[i]; i++) {
```

```
    result = 10 * result + digits[i] - '0';
  }
  return result;
}
```

Για το `"123"` το `result` γίνεται 1, 12, 123. Τα προβλήματά της αναλύονται στη Θεωρία («Από χαρακτήρες σε αριθμό»). Μια ανθεκτικότερη εκδοχή ελέγχει κάθε χαρακτήρα με `'0' <= c && c <= '9'`, χειρίζεται το πρόσημο και επιστρέφει την επιτυχία χωριστά από την τιμή (με παράμετρο εξόδου).

§16.17 Άθροισμα τέλειων τετραγώνων σε διάστημα

```
long long sum_squares(long long a, long long b) {
  long long sum = 0;
  for (long long k = 0; k * k <= b; k++)
    if (k * k >= a)
      sum += k * k;
  return sum;
}
```

Η `sum_squares(10, 50)` δίνει $16 + 25 + 36 + 49 = 126$. Η `sum_squares(1, 1000000000000LL)` κάνει ένα εκατομμύριο βήματα, τελειώνει ακαριαία και δίνει 333333833333500000, όσο και ο τύπος της Θεωρίας με $m = 10^6$.

§16.18 Η `get_two_chars`

Δύο χαρακτήρες «επιστρέφονται» μέσω παραμέτρων εξόδου, και το `return` λέει αν πέτυχε η ανάγνωση:

```
int get_two_chars(char *first, char *second) {
  int c1 = getchar();
  int c2 = getchar();
  if (c1 == EOF || c2 == EOF)
    return 0;
  *first = c1;
  *second = c2;
  return 1;
}
```

Με `char a, b;` και `while (get_two_chars(&a, &b)) printf("[%c%c]\n", a, b);`, η είσοδος `abcde` τυπώνει `[ab]` και `[cd]`· το μονό `e` αγνοείται. Οι `c1`, `c2` είναι `int` γιατί η `getchar` επιστρέφει `int` ώστε να χωρά το `EOF` (Κεφάλαιο 9). Στην Εργασία #1, όπου οι δομές απαγορεύονται και η είσοδος διαβάζεται μόνο με `getchar`, έτσι γράφονται βοηθητικές που διαβάζουν πεδία πολλών bytes.

§16.19 Η swap

Οι διαφάνειες δίνουν τη `main` και ζητούν τη `swap(...)` που λείπει. Η λύση:

```c
#include <stdio.h>

void swap(int *a, int *b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}

int main() {
  int a = 100, b = 200;
  printf("%d %d\n", a, b);
  swap(&a, &b);
  printf("%d %d\n", a, b);
  return 0;
}
```

```
$ ./swap
100 200
200 100
```

Η `tmp` χρειάζεται γιατί μετά το `*a = *b` η αρχική τιμή του `*a` έχει χαθεί. Οι παράμετροι λέγονται κι αυτές `a` και `b`, αλλά είναι άλλες μεταβλητές (τύπου `int *`), τοπικές στη `swap`.

§16.20 «yes» αν ένα στοιχείο υπάρχει σε δισδιάστατο πίνακα

Με τον πρώτο τρόπο της «Έξοδος από εμφωλευμένους βρόχους»:

```c
int contains(int grid[ROWS][COLS], int needle) {
  for (int i = 0; i < ROWS; i++)
    for (int j = 0; j < COLS; j++)
      if (grid[i][j] == needle)
        return 1;          // βγαίνει και από τους δύο βρόχους
  return 0;
}
```

Η `main` κάνει `if (contains(grid, 7)) printf("yes\n");` και το `"yes"` τυπώνεται μία φορά. Με σημαία, οι βρόχοι θα γίνονταν `for (i = 0; i < ROWS && !found; i++)` και αντίστοιχα για τον εσωτερικό.

§16.21 Αποδοτική αναδρομική Fibonacci

Με απομνημόνευση («Αναδρομή χωρίς επανυπολογισμούς»):

```
#define MAXN 92

long long memo[MAXN + 1];   // global: αρχικοποιείται με μηδενικά

long long fib(int n) {
  if (n <= 1)
    return n;
  if (memo[n] != 0)         // το έχουμε ήδη υπολογίσει
    return memo[n];
  memo[n] = fib(n - 1) + fib(n - 2);
  return memo[n];
}
```

Το 0 σημαίνει «δεν έχει υπολογιστεί», αφού για $n \geq 2$ κανένας αριθμός Fibonacci δεν είναι 0. Η `fib(50)` δίνει 12586269025 και η `fib(92)`, ο μεγαλύτερος που χωρά σε `long long`, 7540113804746346429, ακαριαία. Η απλή εκδοχή θα έκανε περίπου $2{,}4 \cdot 10^{19}$ κλήσεις για τη `fib(92)`: αιώνες, ακόμη και με ένα δισεκατομμύριο κλήσεις το δευτερόλεπτο. Η δεύτερη τεχνική, με τις δύο τελευταίες τιμές ως ορίσματα:

```
long long fib_acc(int n, long long a, long long b) {
  return n == 0 ? a : fib_acc(n - 1, b, a + b);
}
```

Η `fib_acc(50, 0, 1)` δίνει πάλι 12586269025, με 51 κλήσεις.

§16.22 Υποεντολές με `argv[1]`

Ο σκελετός μιας `main` με υποεντολές, για την Εργασία #1:

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv) {
  if (argc < 2) {
    fprintf(stderr, "Usage: %s info|rate ...\n", argv[0]);
    return 1;
  }
  if (strcmp(argv[1], "info") == 0) {
    printf("running info\n");
  } else if (strcmp(argv[1], "rate") == 0 && argc == 3) {
    printf("running rate with %s\n", argv[2]);
  } else {
    fprintf(stderr, "Unknown subcommand: %s\n", argv[1]);
    return 1;
  }
```

```
  return 0;
}
```

Το `./sub rate 2.0` τυπώνει `running rate with 2.0`, ενώ το `./sub foo` τυπώνει μήνυμα λάθους στο `stderr` και τερματίζει με κωδικό 1. Στην πράξη κάθε κλάδος καλεί τη δική του συνάρτηση (`do_info()`, `do_rate(...)`).

Κύρια σημεία

1. Η επίλυση προβλημάτων μαθαίνεται με εξάσκηση: κάθε λυμένο πρόβλημα γίνεται μοτίβο για το επόμενο.
2. Για κάθε λύση ρωτάμε «χρονική και χωρική πολυπλοκότητα;», στη χειρότερη περίπτωση.
3. Ο μέσος όρος και η σειριακή αναζήτηση σε n στοιχεία είναι $O(n)$ χρόνος και $O(1)$ μνήμη.
4. Συχνά ανταλλάσσουμε μνήμη με χρόνο: το διπλό στοιχείο βρίσκεται σε $O(n^2)$ χωρίς μνήμη, σε $O(n \log n)$ με ταξινόμηση, ή σε $O(n)$ με βοηθητικό πίνακα αν οι τιμές έχουν μικρό εύρος.
5. Όταν όλα τα στοιχεία είναι διπλά εκτός από ένα, το XOR όλων τους δίνει το μοναδικό σε $O(n)$ χρόνο και $O(1)$ μνήμη.
6. Τα `n % 10`, `n / 10` και `10 * r + digit` δίνουν την αντιστροφή ψηφίων, τον έλεγχο παλινδρομικού και την `atoi`.
7. Μια συνάρτηση που μετατρέπει δεδομένα πρέπει να σκεφτεί την άκυρη είσοδο, την υπερχείλιση και το τερματικό `'\0'`.
8. Λίγα μαθηματικά αλλάζουν την πολυπλοκότητα: τα τέλεια τετράγωνα έως b είναι μόνο $\sqrt{b}$.
9. Ο `char *array[]` είναι πίνακας από pointers, ο `char **array` ένας pointer και ο `char array[10][10]` 100 συνεχόμενοι χαρακτήρες· ως παράμετροι, μόνο οι δύο πρώτοι είναι ισοδύναμοι.
10. Τα ορίσματα περνούν κατά τιμή· για να αλλάξει μεταβλητές του καλούντος (swap) ή να «επιστρέψει» πολλές τιμές, μια συνάρτηση παίρνει τις διευθύνσεις τους.
11. Το `break` βγάζει μόνο από τον εσωτερικό βρόχο· για εμφωλευμένους βρόχους χρησιμοποιούμε `return`, σημαία ή `goto`.
12. Ένας δισδιάστατος πίνακας στον σωρό είναι είτε `int **` με μία `malloc` ανά γραμμή είτε ένα μπλοκ `M × N` με δείκτη `i * N + j`.
13. Η απλή αναδρομική Fibonacci είναι εκθετική· με απομνημόνευση ή με τις δύο τελευταίες τιμές ως ορίσματα γίνεται $O(n)$.
14. Για την Εργασία #1: `/usr/bin/time -v` και `memusage` μετρούν μνήμη· `xxd`, `hexdump`, `hexedit`, `cmp` και `vbindiff` δείχνουν και συγκρίνουν δυαδικά αρχεία· οι υποεντολές είναι σύγκριση του `argv[1]` με `strcmp`.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
χρονική πολυπλοκότητα	time complexity	Πώς αυξάνονται τα βήματα με το μέγεθος της εισόδου.
χωρική πολυπλοκότητα	space complexity	Πόση επιπλέον μνήμη χρειάζεται σε σχέση με την είσοδο.
χειρότερη περίπτωση	worst case	Η είσοδος που κάνει τον αλγόριθμο να δουλέψει περισσότερο.
αντιστάθμισμα χρόνου–μνήμης	time–space tradeoff	Ταχύτερη λύση με περισσότερη μνήμη, ή το αντίστροφο.
αποκλειστικό Ή	XOR (^)	Bit 1 όταν τα δύο bits διαφέρουν· $x \oplus x = 0$.
παλινδρομικός	palindrome	Που διαβάζεται ίδια και από τις δύο μεριές.
κλήση κατά τιμή	call by value	Η συνάρτηση παίρνει αντίγραφα των ορισμάτων.
παράμετρος εξόδου	output parameter	Δείκτης μέσω του οποίου η συνάρτηση γράφει ένα αποτέλεσμα.
σημαία	flag	Μεταβλητή που καταγράφει ότι συνέβη κάτι (π.χ. `found`).
απομνημόνευση	memoization	Αποθήκευση αποτελεσμάτων ώστε να μην ξαναϋπολογίζονται.
υποεντολή	subcommand	Λέξη στο `argv[1]` που επιλέγει τι θα κάνει το πρόγραμμα.

Διάβασμα

- **Διαφάνειες**: Διάλεξη 16, σελ. 1–28· γιατί εξάσκηση σελ. 5–6· μέσος όρος σελ. 8–9· αναζήτηση σελ. 10 και 15· διπλό στοιχείο σελ. 11· αντιστροφή ψηφίων σελ. 13· μοναδικό στοιχείο σελ. 14· `atoi` σελ. 16–17· τέλεια τετράγωνα σελ. 18· `char *array[]` / `char **` / `char [10][10]` σελ. 19· `get_two_chars` σελ. 20· παλινδρομικός σελ. 21· `swap` σελ. 22–23· δισδιάστατη αναζήτηση σελ. 24· δισδιάστατος στον σωρό σελ. 25· Fibonacci σελ. 26· Εργασία #1 σελ. 27.
- **Σημειώσεις**:
  - Κεφάλαιο 11: Ταξινόμηση και αναζήτηση, ενότητες «Ταξινόμηση πινάκων» (K04, σελ. 160–166, για τον συμβολισμό O) και «Αναζήτηση σε πίνακες» (K04, σελ. 172).
  - Κεφάλαιο 5: Δείκτες και πίνακες, ενότητα «Δείκτες» (K04, σελ. 72–77, για τη `swap`).
  - Κεφάλαιο 6: Δυναμική μνήμη, συμβολοσειρές και πολυδιάστατοι πίνακες, ενότητες «Πίνακες δεικτών και δείκτες σε δείκτες» (K04, σελ. 97) και «Πολυδιάστατοι πίνακες» (K04, σελ. 100–102).
  - Κεφάλαιο 3: Η ροή του ελέγχου, ενότητες «Εντολές `break` και `continue`» και «Εντολή `goto` και ετικέτες» (K04, σελ. 56–57).

- **Εργαστήρια**: Εργαστήριο 5: άσκηση `fib.c` (ερωτήματα 2.3–2.6)· Εργαστήριο 6: άσκηση `myprog.c` (αποτελέσματα μέσω δεικτών)· Εργαστήριο 7: ασκήσεις `twodim.c` και `mines.c` (δισδιάστατοι πίνακες, στατικοί και στον σωρό).
- **Άλλα**: `man 1 time`, `man 1 memusage`, `man 1 xxd`, `man 1 hexdump`, `man 1 cmp`, `man 3 strcmp`.

Συχνά λάθη

- **Ακέραια διαίρεση στον μέσο όρο**. Το `sum / 100` δίνει 5 αντί για 5,5. Επιστρέψτε `double` με `sum / 100.0`.
- **Πολυπλοκότητα της καλύτερης περίπτωσης**. «Η `find` είναι $O(1)$ γιατί μπορεί να το βρει πρώτο» είναι λάθος: αναλύουμε τη χειρότερη, $O(n)$.
- **Σύγκριση στοιχείου με τον εαυτό του**. Με `for (j = 0; ...)` αντί για `j = i + 1`, το `a[i] == a[j]` ισχύει για `i == j` και «βρίσκεται» διπλό σε κάθε πίνακα.
- **Υπερχείλιση**. `reverse(1999999999)` ή `atoi("99999999999")` δίνουν λάθος αριθμό. Χρησιμοποιήστε ευρύτερο τύπο ή ελέγξτε πριν πολλαπλασιάσετε με 10.
- `swap` **κατά τιμή**. Η `void swap(int x, int y)` δεν αλλάζει τίποτα στην `main`. Δηλώστε `int *` παραμέτρους και καλέστε `swap(&a, &b)`.
- **Δισδιάστατος πίνακας σε παράμετρο** `char **`. Το `f(grid)` με `char grid[10][10]` δίνει incompatible pointer type (στο gcc 14 σφάλμα) και, αν τρέξει, κρασάρει. Δηλώστε `char grid[][10]`.
- `break` **σε εμφωλευμένους βρόχους**. Το `"yes"` τυπώνεται πολλές φορές. Βάλτε την αναζήτηση σε συνάρτηση με `return`.
- `char` **για το αποτέλεσμα της** `getchar`. Το `EOF` δεν ξεχωρίζει αξιόπιστα. Κρατήστε το σε `int` και ελέγξτε το πριν το αποθηκεύσετε.
- `time -v` **αντί για** `/usr/bin/time -v`. Η ενσωματωμένη `time` του shell δεν δέχεται `-v`.
- `argv[1]` **χωρίς έλεγχο του** `argc`. Το `./soundwave` χωρίς ορίσματα περνά `NULL` στην `strcmp` και κρασάρει. Ελέγξτε πρώτα `argc < 2`.

Ερωτήσεις κατανόησης

- **E16.1** Ποια η χρονική και ποια η χωρική πολυπλοκότητα της `average` για n στοιχεία;[127]
- **E16.2** Γιατί η σειριακή αναζήτηση είναι $O(n)$, αφού μερικές φορές τελειώνει σε ένα βήμα;[128]
- **E16.3** Γιατί το XOR όλων των στοιχείων δίνει το μοναδικό στοιχείο χωρίς ζευγάρι;[129]
- **E16.4** Τι δίνουν τα `4567 % 10` και `4567 / 10`;[130]
- **E16.5** Πόσα βήματα κάνει ο βρόχος των τέλειων τετραγώνων για $b = 10^{12}$;[131]

[127] $O(n)$ χρόνος (ένα πέρασμα) και $O(1)$ μνήμη (μόνο `i` και `sum`).
[128] Η πολυπλοκότητα αναφέρεται στη χειρότερη περίπτωση, όταν το στοιχείο λείπει και ελέγχονται και τα n.
[129] Επειδή $x \oplus x = 0$, $x \oplus 0 = x$ και η σειρά δεν παίζει ρόλο: τα ζευγάρια μηδενίζονται και μένει το μοναδικό.
[130] 7 (το τελευταίο ψηφίο) και 456.
[131] Περίπου $\sqrt{10^{12}} = 10^6$.

- **E16.6** Μπορείτε να περάσετε έναν `char grid[10][10]` σε παράμετρο `char **`;[132]
- **E16.7** Γιατί η `void swap(int x, int y)` δεν δουλεύει;[133]
- **E16.8** Στο ένα μπλοκ `M × N`, πού βρίσκεται το στοιχείο γραμμής `i`, στήλης `j`;[134]
- **E16.9** Γιατί η απλή αναδρομική `fib` είναι αργή, και τι κάνει η απομνημόνευση;[135]
- **E16.10** Πώς βλέπετε τη μέγιστη μνήμη που χρησιμοποίησε το πρόγραμμά σας;[136]

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A16.1–A16.15)

- **A16.1** Μέσος όρος πίνακα και πολυπλοκότητα: Διάλεξη 16, διαφάνειες 8–9 · ★☆☆ · programming · `slides-lec16-average-complexity`
- **A16.2** Αναζήτηση σε πίνακα και πολυπλοκότητα: Διάλεξη 16, διαφάνειες 10 και 15 · ★☆☆ · programming · `slides-lec16-find-complexity`
- **A16.3** Η συνάρτηση get_two_chars: Διάλεξη 16, διαφάνεια 20 · ★☆☆ · programming · `slides-lec16-get-two-chars`
- **A16.4** Παλινδρομικός αριθμός: Διάλεξη 16, διαφάνεια 21 · ★☆☆ · programming · `slides-lec16-palindrome-number`
- **A16.5** Αντιστροφή ψηφίων αριθμού: Διάλεξη 16, διαφάνεια 13 · ★☆☆ · programming · `slides-lec16-reverse-digits`
- **A16.6** yes αν ένα στοιχείο υπάρχει σε δισδιάστατο πίνακα: Διάλεξη 16, διαφάνεια 24 · ★☆☆ · programming · `slides-lec16-search-2d`
- **A16.7** Η συνάρτηση swap: Διάλεξη 16, διαφάνειες 22–23 · ★☆☆ · programming · `slides-lec16-swap`
- **A16.8** Γιατί εξάσκηση στην επίλυση προβλημάτων;: Διάλεξη 16, διαφάνειες 5–6 · ★☆☆ · short-answer · `slides-lec16-why-practice`
- **A16.9** Η atoi και τι μπορεί να πάει στραβά: Διάλεξη 16, διαφάνειες 16–17 · ★★☆ · programming · `slides-lec16-atoi`
- **A16.10** char *array[], char **array και char array[10][10]: Διάλεξη 16, διαφάνεια 19 · ★★☆ · short-answer · `slides-lec16-char-pointer-arrays`
- **A16.11** Αποδοτική αναδρομική Fibonacci: Διάλεξη 16, διαφάνεια 26 · ★★☆ · programming · `slides-lec16-fibonacci-efficient`
- **A16.12** Το στοιχείο που υπάρχει δύο φορές: Διάλεξη 16, διαφάνεια 11 · ★★☆ · programming · `slides-lec16-find-duplicate`
- **A16.13** Δισδιάστατος πίνακας στον σωρό: Διάλεξη 16, διαφάνεια 25 · ★★☆ · programming · `slides-lec16-heap-2d-array`

[132] Όχι. Ο `grid` γίνεται `char (*)[10]` και δεν περιέχει pointers· η παράμετρος πρέπει να είναι `char grid[][10]`.

[133] Τα ορίσματα περνούν κατά τιμή: ανταλλάσσει αντίγραφα, όχι τις μεταβλητές του καλούντος.

[134] Στη θέση `i * N + j`.

[135] Ξαναϋπολογίζει τις ίδιες τιμές, με εκθετικό πλήθος κλήσεων. Η απομνημόνευση κρατά κάθε `fib(k)` σε πίνακα ώστε να υπολογίζεται μία φορά: $O(n)$.

[136] Με `/usr/bin/time -v ./prog`, γραμμή «Maximum resident set size».

- **A16.14** Το στοιχείο χωρίς ζευγάρι: Διάλεξη 16, διαφάνεια 14 · ★★☆ · programming · `slides-lec16-single-unpaired`
- **A16.15** Άθροισμα τέλειων τετραγώνων σε διάστημα: Διάλεξη 16, διαφάνεια 18 · ★★☆ · programming · `slides-lec16-sum-perfect-squares`

Εργαστήριο (A16.16–A16.17)

- **A16.16** Σκαλί-σκαλί (Παλιό θέμα, Προαιρετικό): Εργαστήριο 5, Άσκηση 3 · ★★★ · programming · `lab-lab05-ladder`
- **A16.17** Χτίζοντας έναν χιονάνθρωπο (Παλιό θέμα): Εργαστήριο 7, Άσκηση 4 · ★★★ · programming · `lab-lab07-olaf`

Εργασίες (A16.18)

- **A16.18** Ο Γρίφος του Στέργιου: Bonus #0 (2025-26, προαιρετική) · ★★★ · programming · `hw-2025-bonus0-stergios`

Θέματα εξετάσεων (A16.19–A16.26)

- **A16.19** Εαυτοί Αριθμοί: Κατατακτήριες Δεκεμβρίου 2023, Θέμα 2 · ★★☆ · programming · `exam-2023-dec-q2`
- **A16.20** Αγαπήσιμοι Αριθμοί: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #0 (Valentine's Themed), Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex0-q4`
- **A16.21** Clyde Πρώτοι: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #11, Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex11-q4`
- **A16.22** Εαυτοί Αριθμοί: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #12, Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex12-q4`
- **A16.23** Τυχεροί Αριθμοί: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #13, Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex13-q4`
- **A16.24** Οι Καλύτεροι Αριθμοί, Παμψηφεί: Κατατακτήριες Δεκεμβρίου 2024, Θέμα 2 · ★★☆ · programming · `exam-2024-dec-q2`
- **A16.25** Χτίζοντας έναν Χιονάνθρωπο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #3 (Frozen Themed), Θέμα 4 · ★★★ · programming · `exam-2023-fall-ex3-q4`
- **A16.26** Μετρώντας τα Αστέρια - stars: Εξέταση Ιανουαρίου 2025, Θέμα 6 · ★★★ · programming · `exam-2025-jan-q6`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A11.20** Άψογα Τετράγωνα (Bonus): Εργασία 1 (2023-24), Άσκηση 3 (Bonus) · ★★★ · programming · `hw-2023-hw1-flawless`
- **A15.17** Παραγοντοποίηση ημιπρώτων (factor): Εργασία 1 (2024-25), Άσκηση 3 (Bonus) · ★★★ · programming · `hw-2024-hw1-factor`

- **A25.5** Σκαλί-Σκαλί: Κατατακτήριες Δεκεμβρίου 2023, Θέμα 3 · ★★☆ · programming · `exam-2023-dec-q3`
- **A25.10** Τρόποι να Φάμε Παϊδάκια: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #14, Θέμα 4 · ★★★ · programming · `exam-2023-fall-ex14-q4`
- **A25.7** Ανεβαίνοντας Επίπεδο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #2 (Pokémon Themed), Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex2-q4`

Κεφάλαιο 17: Δυαδική Αναζήτηση και Ταξινόμηση

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να υλοποιείτε γραμμική και δυαδική αναζήτηση και να εξηγείτε πότε και γιατί η δεύτερη είναι ταχύτερη· να αποφεύγετε την υπερχείλιση στον υπολογισμό του μέσου· να γράφετε και να ιχνηλατείτε τους αλγορίθμους ταξινόμησης επιλογής, εισαγωγής, φυσαλίδας, συγχώνευσης και την ταχυταξινόμηση· και να δίνετε την πολυπλοκότητα χρόνου και χώρου του καθενός.

Προαπαιτούμενα: Κεφάλαιο 11 (δείκτες, αναδρομή), Κεφάλαιο 15 (πολυπλοκότητα)

Χρόνος μελέτης: ~2,5 ώρες

Σύνοψη

Η διάλεξη εξετάζει δύο από τα πιο κλασικά προβλήματα της πληροφορικής: πώς βρίσκουμε ένα στοιχείο σε μια ακολουθία (αναζήτηση) και πώς βάζουμε μια ακολουθία σε σειρά (ταξινόμηση). Η γραμμική αναζήτηση ελέγχει τα στοιχεία ένα προς ένα σε χρόνο $O(n)$· αν όμως η ακολουθία είναι ταξινομημένη, η δυαδική αναζήτηση μοιράζει κάθε φορά το διάστημα στη μέση και τελειώνει σε $O(\log n)$ βήματα. Η ταξινόμηση είναι λοιπόν το κλειδί για γρήγορη αναζήτηση. Η διάλεξη παρουσιάζει τρεις απλούς αλγορίθμους ταξινόμησης σε $O(n^2)$ (επιλογής, εισαγωγής, φυσαλίδας) και δύο αναδρομικούς αλγορίθμους «διαίρει και βασίλευε» (συγχώνευσης και ταχυταξινόμηση) με $O(n \log n)$. Βλέπουμε επίσης ότι ακόμη και ένας «απλός» αλγόριθμος όπως η δυαδική αναζήτηση κρύβει ένα λεπτό σφάλμα υπερχείλισης.

Θεωρία

§17.1 Αναζήτηση και η ιδέα της διχοτόμησης

Αναζήτηση (search) είναι το πρόβλημα να βρούμε αν ένα στοιχείο υπάρχει σε μια συλλογή στοιχείων και, αν ναι, σε ποια θέση. Η διάλεξη ξεκινά με παιχνίδια. Αν μαντεύουμε έναν κρυμμένο αριθμό στο $[1, 100]$ ρωτώντας «είναι ο 1;», «είναι ο 2;», … χρειαζόμαστε στη χειρότερη περίπτωση 100 ερωτήσεις· αν ρωτάμε «είναι μεγαλύτερος από το μέσο του διαστήματος;», κάθε απάντηση πετά τους μισούς υποψηφίους. Σε ένα λεξικό ανοίγουμε κάπου στη μέση και, επειδή τα λήμματα είναι σε αλφαβητική σειρά, ξέρουμε αν θα πάμε μπροστά ή πίσω. Στο Guess Who οι καλές ερωτήσεις χωρίζουν τους υποψήφιους χαρακτήρες σε δύο περίπου ίσα μέρη.

Η κοινή ιδέα είναι η **διχοτόμηση**: αν κάθε ερώτηση μειώνει τους υποψηφίους στο μισό, τότε από n υποψηφίους μένει ένας μετά από περίπου $\log_2 n$ ερωτήσεις. Για $n = 100$ αυτό είναι 7 ερωτήσεις ($2^7 = 128 \geq 100$) και για $n = 10^9$ μόλις 30 ($2^{30} \approx 1{,}07 \cdot 10^9$). Το λεξικό μάς δείχνει και την προϋπόθεση: η διχοτόμηση δουλεύει μόνο όταν τα δεδομένα είναι σε **σειρά**.

§17.2 Γραμμική αναζήτηση

Η **γραμμική** ή **σειριακή αναζήτηση** (linear / serial search) είναι ο απλούστερος αλγόριθμος αναζήτησης ενός στοιχείου σε μια ακολουθία. Ξεκινά από το πρώτο στοιχείο και σε κάθε βήμα συγκρίνει το στοιχείο που ψάχνουμε με το τρέχον της ακολουθίας. Αν είναι ίσα σταματά, αλλιώς συνεχίζει στο επόμενο, μέχρι να φτάσει στο τελευταίο.

```
for (i = 0; i < 100; i++)
  if (haystack[i] == needle)
    return i;
return -1;
```

Συνηθίζεται η συνάρτηση να επιστρέφει τη θέση του στοιχείου ή `-1` όταν δεν το βρει, αφού το `-1` δεν είναι ποτέ έγκυρη θέση πίνακα. Στη χειρότερη περίπτωση (το στοιχείο είναι τελευταίο ή λείπει) κάνει n συγκρίσεις, άρα η **χρονική πολυπλοκότητα** είναι $O(n)$, ενώ ο **χώρος** που χρειάζεται είναι σταθερός, $O(1)$ (μόνο ο μετρητής `i`). Το πλεονέκτημά της είναι ότι δεν απαιτεί τίποτα από τα δεδομένα: δουλεύει και σε αταξινόμητο πίνακα.

Μπορούμε να βρούμε ένα στοιχείο πιο γρήγορα από $O(n)$; Σε αταξινόμητο πίνακα όχι: οποιοδήποτε στοιχείο που δεν κοιτάξαμε μπορεί να είναι αυτό που ψάχνουμε. Χρειαζόμαστε επιπλέον πληροφορία για τα δεδομένα, και αυτή είναι η ταξινόμηση.

§17.3 Ταξινομημένη ακολουθία

Μια ακολουθία στοιχείων a_i λέγεται **ταξινομημένη** (sorted) ως προς έναν τελεστή σύγκρισης $\leq_\alpha$ αν και μόνο αν

$$\forall i \leq j.\ a_i \leq_\alpha a_j$$

δηλαδή κάθε στοιχείο «προηγείται ή ισούται» με όλα όσα ακολουθούν. Η ταξινόμηση εξαρτάται πάντα από τη σχέση διάταξης που διαλέγουμε:

- Η $1, 7, 8, 10, 19$ είναι ταξινομημένη ως προς τη συνηθισμένη σύγκριση ακεραίων, $a_i \leq_\alpha a_j \Leftrightarrow a_i \leq a_j$: είναι σε **αύξουσα** σειρά.
- Η $19, 10, 8, 7, 1$ είναι ταξινομημένη ως προς τον τελεστή $a_i \leq_\varphi a_j \Leftrightarrow -a_i \leq -a_j \Leftrightarrow a_j \leq a_i$: είναι σε **φθίνουσα** σειρά.

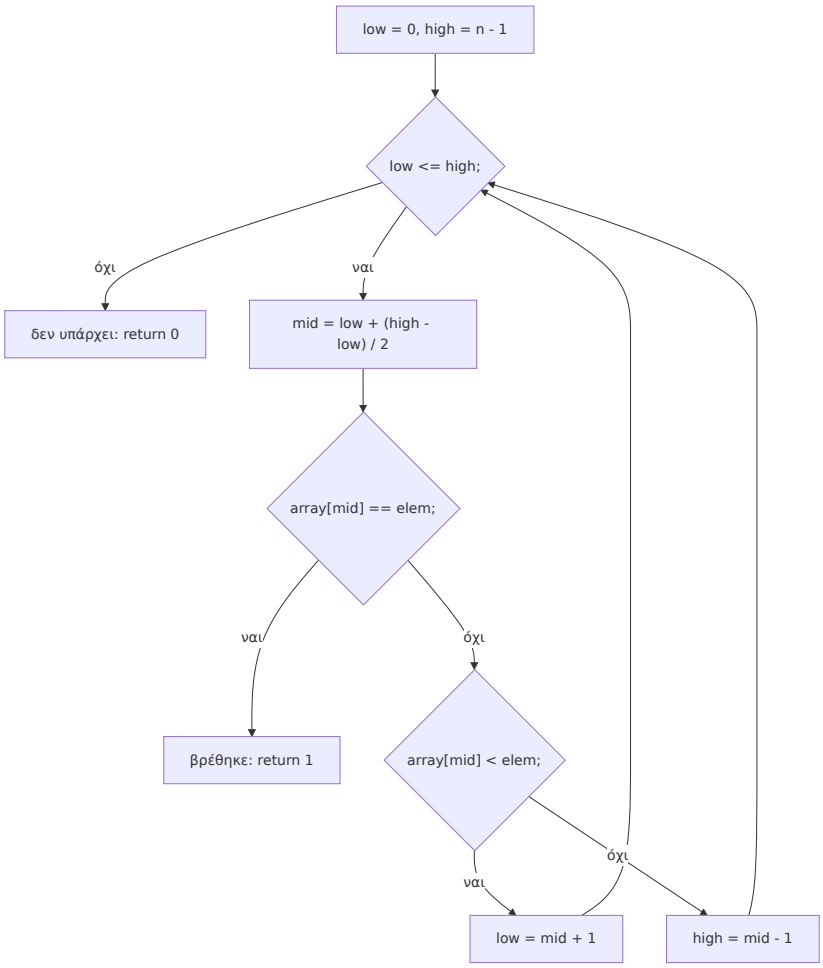
Με τον ίδιο τρόπο ταξινομούμε και άλλα δεδομένα, αρκεί να ορίσουμε τη σύγκριση: λέξεις αλφαβητικά, ή εγγραφές φοιτητών ως προς τον βαθμό τους. Στη συνέχεια, «ταξινομημένος πίνακας» σημαίνει ακέραιοι σε αύξουσα σειρά.

§17.4 Δυαδική αναζήτηση

Η **δυαδική αναζήτηση** (binary search) βρίσκει αν ένα στοιχείο υπάρχει σε μια **ταξινομημένη** ακολουθία. Σε κάθε βήμα χωρίζει την ακολουθία σε δύο τμήματα και συγκρίνει το στοιχείο με το **μεσαίο** στοιχείο:

- αν είναι ίσα, το βρήκαμε·
- αν το μεσαίο είναι μικρότερο, το στοιχείο (αν υπάρχει) βρίσκεται δεξιά του μέσου·
- αν το μεσαίο είναι μεγαλύτερο, βρίσκεται αριστερά του.

Ο αλγόριθμος συνεχίζει στο κατάλληλο τμήμα, μέχρι να βρει το στοιχείο ή να δείξει ότι δεν μπορεί να βρίσκεται πουθενά (το τμήμα άδειασε). Στον κώδικα, το τμήμα που μένει να ψάξουμε είναι οι θέσεις από `low` έως `high` (και οι δύο μέσα)· αρχικά είναι όλος ο πίνακας, `low = 0` και `high = n - 1`.



Σχήμα: η ροή της δυαδικής αναζήτησης· κάθε γύρος μικραίνει το `[low, high]` στο μισό.

Προσέξτε τρεις λεπτομέρειες που κάνουν τον αλγόριθμο σωστό:

1. Η συνθήκη είναι `low <= high` και όχι `low < high`: όταν `low == high` μένει ακόμη ένα στοιχείο να ελέγξουμε.
2. Τα νέα όρια είναι `mid + 1` και `mid - 1`, όχι `mid`: το `array[mid]` το ελέγξαμε ήδη. Με `low = mid` ο βρόχος μπορεί να μην τερματίσει ποτέ.
3. Ο πίνακας **πρέπει** να είναι ταξινομημένος. Σε αταξινόμητο πίνακα η δυαδική αναζήτηση δεν «σκάει», απλώς δίνει λάθος απάντηση.

§17.5 Υπερχείλιση στον υπολογισμό του μέσου

Η πρώτη υλοποίηση της διάλεξης υπολογίζει το μέσο με το προφανές `mid = (low + high) / 2;` και έχει σφάλμα. Τα `low` και `high` είναι έγκυρες θέσεις, άρα χωρούν σε `int`, αλλά το **άθροισμά** τους μπορεί να ξεπεράσει το `INT_MAX` ($2^{31} - 1 = 2147483647$). Αυτό συμβαίνει σε πίνακες με περισσότερα από περίπου 2^{30} (ένα δισεκατομμύριο) στοιχεία: για `low = 1500000000` και `high = 2000000000` το άθροισμα είναι $3{,}5 \cdot 10^9$. Η υπερχείλιση προσημασμένου ακεραίου είναι **απροσδιόριστη συμπεριφορά** (undefined behavior)· στην πράξη το άθροισμα «τυλίγεται» σε αρνητικό αριθμό (με τον `gcc` το `mid` βγαίνει `-397483648` αντί για `1750000000`) και το `array[mid]` διαβάζει εκτός ορίων.

Η διόρθωση είναι να υπολογίζουμε την **απόσταση** των ορίων και να προσθέτουμε το μισό της στο `low`:

```
mid = low + (high - low) / 2;
```

Η διαφορά `high - low` δεν ξεπερνά ποτέ το μέγεθος του πίνακα, οπότε δεν υπάρχει υπερχείλιση, και μαθηματικά το αποτέλεσμα είναι το ίδιο. Η διάλεξη τονίζει ότι ο αλγόριθμος θεωρείται διάσημα **δύσκολος** να υλοποιηθεί **σωστά**: λάθη στη συνθήκη, στα όρια και στον μέσο έχουν βρεθεί ακόμη και σε βιβλία και βιβλιοθήκες (δείτε τους συνδέσμους στο «Διάβασμα»). Το ίδιο μοτίβο `(lower + upper) / 2` εμφανίζεται και στην ταχυταξινόμηση παρακάτω.

§17.6 Πολυπλοκότητα της δυαδικής αναζήτησης

Κάθε γύρος του βρόχου κάνει σταθερή δουλειά και υποδιπλασιάζει το μέγεθος του διαστήματος: $n, n/2, n/4, \ldots, 1$. Μετά από k γύρους μένουν περίπου $n/2^k$ στοιχεία, άρα ο βρόχος τελειώνει μετά από το πολύ περίπου $\log_2 n + 1$ γύρους. Η χρονική πολυπλοκότητα είναι λοιπόν $O(\log n)$ και ο χώρος $O(1)$ (τρεις μεταβλητές, ανεξάρτητα από το n).

n	γραμμική (χειρότερη)	δυαδική (χειρότερη)
100	100	7
10^6	10^6	20
$2 \cdot 10^9$	$2 \cdot 10^9$	31

Η διαφορά είναι τεράστια, αλλά έχει ένα κόστος: ο πίνακας πρέπει πρώτα να ταξινομηθεί, κάτι που (όπως θα δούμε) κοστίζει τουλάχιστον $O(n \log n)$. Η ταξινόμηση αξίζει όταν ψάχνουμε

πολλές φορές στα ίδια δεδομένα. Η δυαδική αναζήτηση υπάρχει έτοιμη στη βιβλιοθήκη της C ως η συνάρτηση `bsearch` του `stdlib.h`.

§17.7 Αλγόριθμοι ταξινόμησης και η swap

Ταξινόμηση (sorting) είναι η αναδιάταξη των στοιχείων μιας ακολουθίας ώστε να γίνει ταξινομημένη. Η διάλεξη παρουσιάζει πέντε αλγορίθμους: bubblesort, selection sort, insertion sort, merge sort και quicksort. Όλοι ταξινομούν έναν πίνακα `int` **επί τόπου**, δηλαδή αλλάζουν τον ίδιο τον πίνακα που τους δίνουμε μέσω δείκτη (`int *x`).

Οι περισσότεροι χρησιμοποιούν ως βασικό βήμα την **αντιμετάθεση** (swap) δύο στοιχείων. Επειδή στη C τα ορίσματα περνούν με τιμή, μια `swap(int a, int b)` θα άλλαζε μόνο τα τοπικά της αντίγραφα. Η `swap` πρέπει να παίρνει **δείκτες** στις δύο μεταβλητές και να αλλάζει τις τιμές μέσω αποαναφοράς, με μια βοηθητική μεταβλητή (`void swap(int *a, int *b)`, κώδικας στα «Παραδείγματα»). Την καλούμε με διευθύνσεις: `swap(&a, &b)` για μεταβλητές, `swap(&x[i], &x[j])` για στοιχεία πίνακα (θυμηθείτε το Κεφάλαιο 11).

§17.8 Ταξινόμηση επιλογής (selection sort)

Η **ταξινόμηση επιλογής** βρίσκει το μικρότερο από όλα τα στοιχεία και το αντιμεταθέτει με το πρώτο· έπειτα βρίσκει το μικρότερο από τα υπόλοιπα και το αντιμεταθέτει με το δεύτερο, κ.ο.κ. Μετά από $n-1$ επιλογές ο πίνακας είναι ταξινομημένος (το τελευταίο στοιχείο μένει αναγκαστικά στη θέση του).

```
for (i = 1 ; i <= n - 1 ; i++) {
  min = i - 1;
  for (j = i ; j <= n - 1 ; j++)
    if (x[j] < x[min])
      min = j;
  swap(&x[i-1], &x[min]);
}
```

Στον γύρο `i` το `min` ξεκινά από τη θέση `i - 1` και ο εσωτερικός βρόχος ψάχνει στις θέσεις `i` έως `n - 1` για κάτι μικρότερο. **Αναλλοίωτη** (invariant): μετά τον γύρο `i`, οι θέσεις `0` έως `i - 1` περιέχουν τα `i` μικρότερα στοιχεία, ταξινομημένα. Ο εσωτερικός βρόχος κάνει $(n-1)+(n-2)+\ldots+1 = n(n-1)/2$ συγκρίσεις ό,τι κι αν περιέχει ο πίνακας, οπότε ο χρόνος είναι $O(n^2)$· ο χώρος είναι $O(1)$. Κάνει όμως το πολύ $n-1$ αντιμεταθέσεις.

§17.9 Ταξινόμηση εισαγωγής (insertion sort)

Η **ταξινόμηση εισαγωγής** δουλεύει όπως ταξινομούμε τα χαρτιά στο χέρι: κρατά ένα ταξινομημένο πρόθεμα και **εισάγει** το επόμενο στοιχείο στη σωστή θέση μέσα του. Τοποθετεί το δεύτερο στοιχείο πριν ή μετά το πρώτο, το τρίτο στη σωστή θέση ανάμεσα στα δύο πρώτα, κ.ο.κ.

```
for (i = 1 ; i <= n - 1 ; i++) {
  j = i - 1;
  while (j >= 0 && x[j] > x[j+1]) {
    swap(&x[j], &x[j+1]);
    j--;
  }
}
```

Το νέο στοιχείο ξεκινά στη θέση `i` και «βουλιάζει» προς τα αριστερά με διαδοχικές αντιμεταθέσεις όσο ο αριστερός του γείτονας είναι μεγαλύτερος. Η σειρά των ελέγχων `j >= 0 && x[j] > x[j+1]` έχει σημασία: λόγω βραχυκύκλωσης του `&&`, το `x[j]` δεν διαβάζεται ποτέ για `j == -1`. Στη χειρότερη περίπτωση (πίνακας σε φθίνουσα σειρά) κάθε στοιχείο ταξιδεύει ως την αρχή, άρα ο χρόνος είναι $O(n^2)$ και ο χώρος $O(1)$. Σε πίνακα σχεδόν ταξινομημένο όμως ο `while` σταματά αμέσως και ο αλγόριθμος είναι πολύ γρήγορος.

§17.10 Ταξινόμηση φυσαλίδας (bubblesort)

Η **ταξινόμηση φυσαλίδας** συγκρίνει ζευγάρια **διαδοχικών** στοιχείων, από το τέλος του πίνακα προς την αρχή, και αντιμεταθέτει όσα δεν είναι στη σωστή σειρά. Μετά το πρώτο πέρασμα το μικρότερο στοιχείο έχει «ανέβει σαν φυσαλίδα» στη θέση 0· το δεύτερο πέρασμα φέρνει το δεύτερο μικρότερο στη θέση 1, κ.ο.κ. Μετά από $n - 1$ περάσματα ο πίνακας είναι ταξινομημένος.

```
for (i = 1 ; i <= n - 1 ; i++)
  for (j = n - 1 ; j >= i ; j--)
    if (x[j-1] > x[j])
      swap(&x[j-1], &x[j]);
```

Ο εσωτερικός βρόχος σταματά στο `i`, γιατί οι θέσεις `0` έως `i - 2` έχουν ήδη τακτοποιηθεί από τα προηγούμενα περάσματα. Ο χρόνος είναι $O(n^2)$ και ο χώρος $O(1)$.

§17.11 Διαίρει και βασίλευε

Οι τρεις παραπάνω αλγόριθμοι τοποθετούν ένα στοιχείο ανά πέρασμα και γι' αυτό κάνουν $O(n^2)$ δουλειά. Για να πάμε ταχύτερα χρησιμοποιούμε τη στρατηγική **διαίρει και βασίλευε** (divide and conquer), που ήδη είδαμε στη δυαδική αναζήτηση:

1. **διαίρεσε** το πρόβλημα σε μικρότερα υποπροβλήματα του ίδιου είδους·
2. **λύσε** τα υποπροβλήματα, αναδρομικά·
3. **συνδύασε** τις λύσεις τους σε λύση του αρχικού.

Η αναδρομή σταματά σε υποπροβλήματα τόσο μικρά που η λύση τους είναι προφανής (ένας πίνακας με ένα ή κανένα στοιχείο είναι ήδη ταξινομημένος). Οι δύο αλγόριθμοι που ακολουθούν διαφέρουν στο πού βάζουν τη δουλειά: η merge sort διαιρεί εύκολα και δουλεύει στη **συνένωση**, η quicksort δουλεύει στη **διαίρεση** και δεν χρειάζεται συνένωση.

§17.12 Ταξινόμηση συγχώνευσης (merge sort)

Η **ταξινόμηση συγχώνευσης** είναι αλγόριθμος διαίρει και βασίλευε (αποδίδεται στον John von Neumann) με δύο βήματα:

1. χώρισε τον πίνακα σε δύο υποπίνακες (στη μέση) και κάλεσε ταξινόμηση συγχώνευσης σε καθέναν·
2. **συγχώνευσε** (merge) τα στοιχεία των δύο ταξινομημένων υποπινάκων.

Η `merge_sort(array, left, right)` της διάλεξης (κώδικας στα «Παραδείγματα») παίρνει τις θέσεις του πρώτου και του τελευταίου στοιχείου του τμήματος· για όλο τον πίνακα καλούμε `merge_sort(a, 0, n - 1)`. Αν `left < right`, υπολογίζει το `middle = left + (right - left) / 2`, ταξινομεί αναδρομικά τα `[left, middle]` και `[middle + 1, right]` και τα συγχωνεύει με `merge(array, left, middle, right)`. Η βάση της αναδρομής είναι το `left >= right`, δηλαδή τμήμα με ένα ή κανένα στοιχείο.

Όλη η δουλειά γίνεται στη `merge`, η οποία παίρνει δύο **ήδη ταξινομημένα** διαδοχικά τμήματα, `x[l..m]` και `x[m+1..r]`. Τα αντιγράφει σε δύο βοηθητικούς πίνακες `left` και `right` και ύστερα, με δύο δείκτες θέσης `i` και `j`, κοιτά κάθε φορά το μικρότερο μη χρησιμοποιημένο στοιχείο κάθε πίνακα και γράφει το μικρότερο από τα δύο πίσω στο `x`. Όταν ο ένας πίνακας εξαντληθεί, αντιγράφει ό,τι έμεινε στον άλλο. Επειδή κάθε σύγκριση βγάζει ένα στοιχείο, η συγχώνευση n στοιχείων κοστίζει $O(n)$.

Η αναδρομή διχοτομεί το μέγεθος, άρα έχει περίπου $\log_2 n$ επίπεδα· σε κάθε επίπεδο οι συγχωνεύσεις αγγίζουν συνολικά και τα n στοιχεία. Ο χρόνος είναι λοιπόν $O(n \log n)$, και μάλιστα ακόμη και στη χειρότερη περίπτωση: θεωρητικά η καλύτερη πολυπλοκότητα που μπορεί να πετύχει ταξινόμηση με συγκρίσεις. Το τίμημα είναι ο **χώρος** $O(n)$ για τους βοηθητικούς πίνακες της `merge`.

§17.13 Ταχυταξινόμηση (quicksort)

Η **ταχυταξινόμηση** (quicksort, του Tony Hoare) είναι επίσης διαίρει και βασίλευε και είναι ιδιαίτερα δημοφιλής. Έχει τρία βήματα:

1. διάλεξε (π.χ. τυχαία) ένα **στοιχείο διαμέρισης** (pivot element)·
2. **διαμέρισε** (partition) τον πίνακα σε δύο υποπίνακες: αριστερά τα στοιχεία που είναι μικρότερα από το pivot και δεξιά αυτά που είναι μεγαλύτερα·
3. τρέξε ταχυταξινόμηση στους δύο υποπίνακες.

Μετά τη διαμέριση κάθε στοιχείο του αριστερού τμήματος είναι μικρότερο ή ίσο από κάθε στοιχείο του δεξιού, οπότε δεν χρειάζεται συγχώνευση: αρκεί να ταξινομηθεί το καθένα.

Η υλοποίηση της διάλεξης, `quicksort(x, lower, upper)` (κώδικας στα «Παραδείγματα»), παίρνει ως pivot το μεσαίο στοιχείο, `x[(lower + upper) / 2]`. Η διαμέριση κινεί δύο δείκτες θέσης τον έναν προς τον άλλον: το `i` προχωρά από αριστερά όσο βρίσκει στοιχεία μικρότερα του pivot, το `j` από δεξιά όσο βρίσκει μεγαλύτερα. Όταν σταματήσουν και οι δύο, τα `x[i]` και `x[j]` είναι σε λάθος πλευρά και αντιμετατίθενται. Όταν τα `i` και `j` διασταυρωθούν, το

`x[lower..j]` έχει τα «μικρά» και το `x[i..upper]` τα «μεγάλα», και η συνάρτηση καλείται αναδρομικά σε αυτά.

Αν το pivot χωρίζει τον πίνακα σε δύο περίπου ίσα μέρη, έχουμε $\log_2 n$ επίπεδα με $O(n)$ δουλειά το καθένα, άρα $O(n \log n)$ **κατά μέση περίπτωση** (average case). Αν όμως το pivot είναι κάθε φορά το μικρότερο ή το μεγαλύτερο στοιχείο, το ένα τμήμα είναι άδειο και το άλλο μικραίνει μόνο κατά ένα: n επίπεδα και $O(n^2)$ **στη χειρότερη περίπτωση** (worst case). Ο χώρος είναι η στοίβα της αναδρομής: $O(n)$ στη χειρότερη περίπτωση για αυτή την υλοποίηση, αλλά γίνεται και $O(\log n)$ αν καλούμε αναδρομικά πάντα πρώτα το μικρότερο τμήμα και χειριζόμαστε το μεγαλύτερο με βρόχο (tail call optimization).

§17.14 Σύνοψη αλγορίθμων και η βιβλιοθήκη

Αλγόριθμος	Χρόνος	Χώρος	Ιδέα
Γραμμική αναζήτηση	$O(n)$	$O(1)$	κοίτα όλα τα στοιχεία με τη σειρά
Δυαδική αναζήτηση	$O(\log n)$	$O(1)$	σύγκρινε με το μέσο, κράτα το μισό
Selection sort	$O(n^2)$	$O(1)$	επίλεξε το ελάχιστο του υπολοίπου
Insertion sort	$O(n^2)$	$O(1)$	εισήγαγε στο ταξινομημένο πρόθεμα
Bubblesort	$O(n^2)$	$O(1)$	αντιμετάθεσε γειτονικά ζεύγη
Merge sort	$O(n \log n)$	$O(n)$	μοίρασε στη μέση, συγχώνευσε
Quicksort	$O(n \log n)$ μέση, $O(n^2)$ χειρότερη	$O(n)$ εδώ	διαμέρισε γύρω από pivot

Στην πράξη σπάνια γράφουμε δική μας ταξινόμηση ή δυαδική αναζήτηση: η βιβλιοθήκη `stdlib.h` έχει την `qsort` (ταξινόμηση) και την `bsearch` (δυαδική αναζήτηση), που δουλεύουν για πίνακες οποιουδήποτε τύπου, αφού τους δώσουμε μια συνάρτηση σύγκρισης (`man 3 qsort`, `man 3 bsearch`). Τους αλγορίθμους όμως πρέπει να τους ξέρετε: είναι κλασικό θέμα εξετάσεων και η βάση για να καταλάβετε την πολυπλοκότητα.

Παραδείγματα

§17.15 Ο γρίφος με τον κρυμμένο αριθμό

Ο γρίφος της διάλεξης (ερώτηση συνέντευξης της Microsoft από τον Steve Ballmer): ένας αριθμός στο $[1, 100]$, απαντήσεις ναι/όχι, και κέρδος που μειώνεται με κάθε προσπάθεια (5€

στην 1η, ..., 0€ στην 6η, ενώ στην 7η πληρώνετε 1€). Εφαρμόζει την ιδέα της διχοτόμησης: ρωτάμε «είναι μεγαλύτερος από 50;», μετά «από 75;» ή «από 25;», κ.ο.κ. Στη χειρότερη περίπτωση οι υποψήφιοι πέφτουν $100 \to 50 \to 25 \to 13 \to 7 \to 4 \to 2 \to 1$. Χρειάζονται λοιπόν έως 7 ερωτήσεις για το $[1, 100]$ και έως 30 για το $[1, 10^9]$. Αν αξίζει να παίξετε το παιχνίδι είναι η ερώτηση της διαφάνειας 5. Το ίδιο κάνουμε σε ένα λεξικό ή στο Guess Who (Θεωρία: «Αναζήτηση και η ιδέα της διχοτόμησης»).

§17.16 Γραμμική αναζήτηση σε πίνακα 100 ακεραίων

Η διάλεξη ζητά μια συνάρτηση που δέχεται έναν πίνακα 100 ακεραίων και έναν ακέραιο και επιστρέφει τη θέση του στοιχείου ή `-1` (Θεωρία: «Γραμμική αναζήτηση»):

```
int find(int haystack[100], int needle) {
  int i;
  for (i = 0; i < 100; i++) {
    if (haystack[i] == needle) {
      return i;
    }
  }
  return -1;
}
```

Αν ο `a` έχει `a[i] == i * i`, τότε το `find(a, 49)` επιστρέφει 7 και το `find(a, 50)` επιστρέφει `-1`.

§17.17 Δυαδική αναζήτηση βήμα προς βήμα

Η συνάρτηση της διάλεξης (διορθωμένη ως προς τον μέσο) επιστρέφει `1` αν το `elem` υπάρχει στον ταξινομημένο πίνακα `array` με `n` στοιχεία και `0` αλλιώς (Θεωρία: «Δυαδική αναζήτηση», «Υπερχείλιση στον υπολογισμό του μέσου»):

```
int binary_search(int elem, int *array, int n) {
  int mid, low = 0, high = n - 1;
  while (low <= high) {
    mid = low + (high - low) / 2;
    if (array[mid] == elem)
      return 1;
    else if (array[mid] < elem)
      low = mid + 1;
    else
      high = mid - 1;
  }
  return 0;
}
```

Για τον πίνακα `{1, 7, 8, 10, 19, 23, 42, 50, 61, 77}` (θέσεις 0–9):

αναζήτηση	low	high	mid	array[mid]	απόφαση
42	0	9	4	19	19 < 42 ⟶ low = 5
42	5	9	7	50	50 > 42 ⟶ high = 6
42	5	6	5	23	23 < 42 ⟶ low = 6
42	6	6	6	42	βρέθηκε ⟶ return 1
9	0	9	4	19	19 > 9 ⟶ high = 3
9	0	3	1	7	7 < 9 ⟶ low = 2
9	2	3	2	8	8 < 9 ⟶ low = 3
9	3	3	3	10	10 > 9 ⟶ high = 2
9	3	2			low > high ⟶ return 0

Και οι δύο αναζητήσεις τελειώνουν σε 4 γύρους, ενώ η γραμμική θα έκανε 7 και 10 συγκρίσεις αντίστοιχα. Η διάλεξη παραπέμπει σε μια οπτικοποίηση που δείχνει τη γραμμική και τη δυαδική αναζήτηση βήμα βήμα.

§17.18 Αναζήτηση στους χρήστες του Instagram

Το Instagram έχει 2 δισεκατομμύρια χρήστες σε έναν πίνακα ακεραίων (ένας ακέραιος ανά χρήστη). Πόσα βήματα χρειάζονται για να βρούμε αν ο χρήστης 424242 υπάρχει; Με γραμμική αναζήτηση έως $2 \cdot 10^9$ συγκρίσεις. Αν ο πίνακας είναι ταξινομημένος, η δυαδική αναζήτηση χρειάζεται έως 31 γύρους, αφού $2^{31} = 2147483648$, που είναι τουλάχιστον $2 \cdot 10^9$ (Θεωρία: «Πολυπλοκότητα της δυαδικής αναζήτησης»). Προσέξτε ότι ένας τέτοιος πίνακας είναι ακριβώς το μέγεθος όπου το `(low + high) / 2` υπερχειλίζει.

§17.19 Η συνάρτηση swap

Η άσκηση της διάλεξης: συμπληρώστε τη `swap( ... )` ώστε να ανταλλάξει τα `a` και `b` (Θεωρία: «Αλγόριθμοι ταξινόμησης και η swap»).

```c
#include <stdio.h>

void swap(int *a, int *b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}

int main() {
  int a = 100, b = 200;
```

```
  printf("%d %d\n", a, b);
  swap(&a, &b);
  printf("%d %d\n", a, b);
  return 0;
}
```

```
$ ./swap
100 200
200 100
```

§17.20 Τρεις απλές ταξινομήσεις στον ίδιο πίνακα

Τρέχοντας τις `selection_sort`, `insertion_sort` και `bubblesort` της διάλεξης στον πίνακα `{5, 2, 9, 1, 7, 3}` και τυπώνοντας τον πίνακα στο τέλος κάθε γύρου του εξωτερικού βρόχου παίρνουμε (Θεωρία: οι τρεις αντίστοιχες ενότητες):

μετά τον γύρο	selection	insertion	bubble
`i = 1`	1 2 9 5 7 3	2 5 9 1 7 3	1 5 2 9 3 7
`i = 2`	1 2 9 5 7 3	2 5 9 1 7 3	1 2 5 3 9 7
`i = 3`	1 2 3 5 7 9	1 2 5 9 7 3	1 2 3 5 7 9
`i = 4`	1 2 3 5 7 9	1 2 5 7 9 3	1 2 3 5 7 9
`i = 5`	1 2 3 5 7 9	1 2 3 5 7 9	1 2 3 5 7 9

Παρατηρήστε τις αναλλοίωτες: στη selection sort και στη bubblesort, μετά τον γύρο `i` τα `i` πρώτα στοιχεία είναι τα `i` μικρότερα, στις τελικές τους θέσεις. Στην insertion sort τα `i + 1` πρώτα στοιχεία είναι ταξινομημένα μεταξύ τους, αλλά όχι απαραίτητα στις τελικές τους θέσεις (το 3 μπαίνει στη θέση του μόλις στον τελευταίο γύρο). Και οι τρεις κάνουν όλους τους γύρους, ακόμη κι όταν ο πίνακας έχει ήδη ταξινομηθεί.

§17.21 Merge sort και quicksort σε ένα πρόγραμμα

Οι `merge`, `merge_sort` και `quicksort` της διάλεξης με ένα `main` που ταξινομεί δύο αντίγραφα του ίδιου πίνακα (Θεωρία: «Ταξινόμηση συγχώνευσης», «Ταχυταξινόμηση»). Οι `left[n1]` και `right[n2]` της `merge` είναι πίνακες μεταβλητού μήκους στη στοίβα, από όπου προέρχεται ο χώρος $O(n)$. Για να φανεί η διαμέριση, η `quicksort` εδώ τυπώνει επιπλέον το τμήμα `x[lower..upper]` μετά από κάθε διαμέριση.

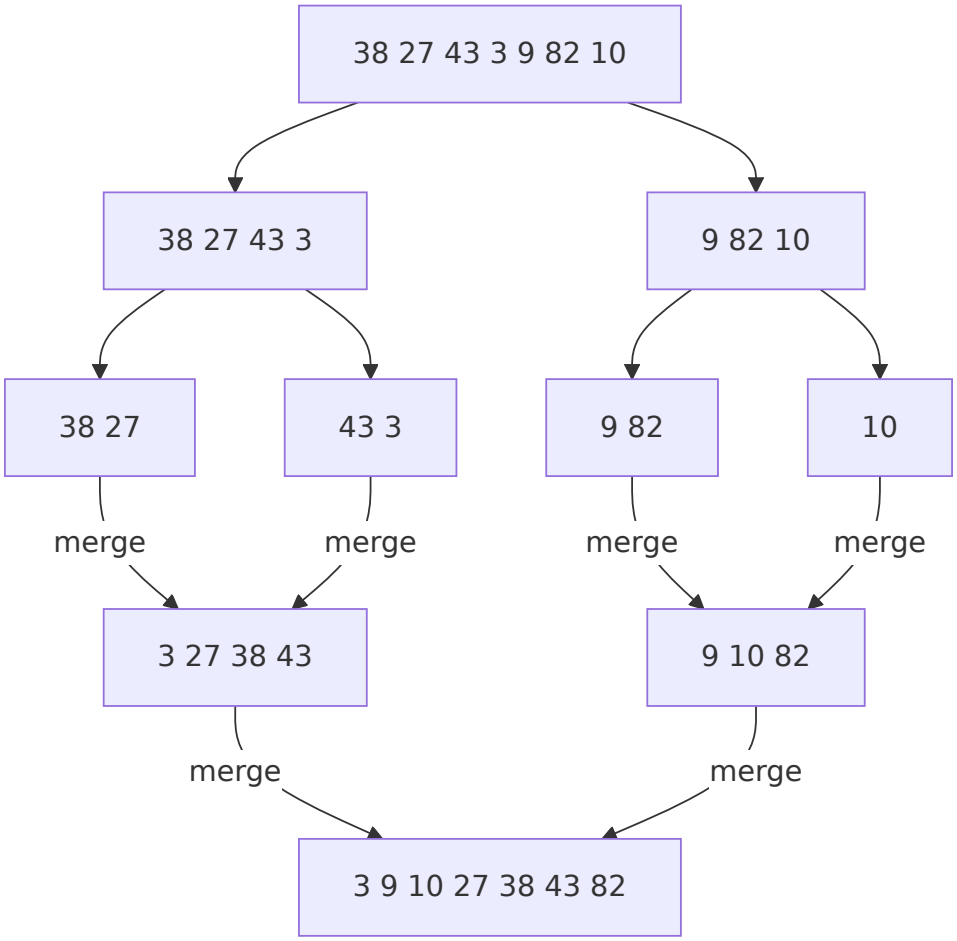
```
#include <stdio.h>

void swap(int *a, int *b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
```

```
}

void merge(int *x, int l, int m, int r) {
  int i, j, k, n1 = m - l + 1, n2 = r - m;
  int left[n1], right[n2];
  for (i = 0; i < n1; i++) left[i] = x[l + i];
  for (j = 0; j < n2; j++) right[j] = x[m + 1 + j];
  i = 0; j = 0; k = l;
  while (i < n1 && j < n2) {
    if (left[i] <= right[j]) x[k++] = left[i++];
    else x[k++] = right[j++];
  }
  while (i < n1) x[k++] = left[i++];
  while (j < n2) x[k++] = right[j++];
}

void merge_sort(int *array, int left, int right) {
  if (left < right) {
    int middle = left + (right - left) / 2;
    merge_sort(array, left, middle);
    merge_sort(array, middle + 1, right);
    merge(array, left, middle, right);
  }
}

void quicksort(int *x, int lower, int upper) {
  if (lower < upper) {
    int pivot = x[(lower + upper) / 2];
    int i, j;
    for (i = lower, j = upper; i <= j;) {
      while (x[i] < pivot) i++;
      while (x[j] > pivot) j--;
      if (i <= j) swap(&x[i++], &x[j--]);
    }
    printf("pivot %2d:", pivot);            /* trace, not in the slides */
    for (int k = lower; k <= upper; k++) printf(" %d", x[k]);
    printf("\n");
    quicksort(x, lower, j);
    quicksort(x, i, upper);
  }
}

void print(int n, int *x) {
```

```
  for (int i = 0; i < n; i++)
    printf("%d ", x[i]);
  printf("\n");
}

int main() {
  int a[] = {38, 27, 43, 3, 9, 82, 10};
  int b[] = {38, 27, 43, 3, 9, 82, 10};
  int n = sizeof(a) / sizeof(a[0]);
  merge_sort(a, 0, n - 1);
  print(n, a);
  quicksort(b, 0, n - 1);
  print(n, b);
  return 0;
}
```

```
$ ./sorts
3 9 10 27 38 43 82
pivot  3: 3 27 43 38 9 82 10
pivot 38: 27 10 9 38 82 43
pivot 10: 9 10 27
pivot 82: 38 43 82
pivot 38: 38 43
3 9 10 27 38 43 82
```

Η πρώτη διαμέριση της quicksort είναι η χειρότερη δυνατή: το pivot 3 είναι το ελάχιστο, οπότε το αριστερό τμήμα έχει ένα στοιχείο και το δεξί έξι. Αν αυτό συνέβαινε σε κάθε επίπεδο, θα είχαμε τη χειρότερη περίπτωση $O(n^2)$. Η merge sort αντίθετα χωρίζει πάντα στη μέση:



Σχήμα: η merge sort χωρίζει στη μέση ως τα ζεύγη (που ταξινομούνται με τον ίδιο τρόπο) και ύστερα συγχωνεύει προς τα πάνω.

Κύρια σημεία

1. Η γραμμική αναζήτηση ελέγχει τα στοιχεία ένα προς ένα· δουλεύει σε οποιονδήποτε πίνακα, με χρόνο $O(n)$ και χώρο $O(1)$.
2. Μια ακολουθία είναι ταξινομημένη ως προς έναν τελεστή σύγκρισης $\leq_\alpha$ όταν $a_i \leq_\alpha a_j$ για κάθε $i \leq j$· αύξουσα και φθίνουσα σειρά είναι απλώς διαφορετικοί τελεστές.
3. Η δυαδική αναζήτηση συγκρίνει με το μεσαίο στοιχείο και συνεχίζει στο μισό όπου μπορεί να βρίσκεται το στοιχείο· απαιτεί ταξινομημένο πίνακα και έχει χρόνο $O(\log n)$ και χώρο $O(1)$.
4. Ο μέσος πρέπει να υπολογίζεται ως `low + (high - low) / 2`: το `(low + high) / 2` υπερχειλίζει σε πολύ μεγάλους πίνακες· η δυαδική αναζήτηση είναι διάσημα δύσκολη να υλοποιηθεί σωστά.
5. Η `swap` παίρνει δείκτες (`swap(&x[i], &x[j])`), γιατί στη C τα ορίσματα περνούν με τιμή.

6. Η selection sort, η insertion sort και η bubblesort έχουν χρόνο $O(n^2)$ και χώρο $O(1)$.
7. Η merge sort (αναδρομικός αλγόριθμος διαίρει και βασίλευε) χωρίζει στη μέση και συγχωνεύει δύο ταξινομημένους υποπίνακες σε γραμμικό χρόνο· έχει χρόνο $O(n \log n)$ σε κάθε περίπτωση και χώρο $O(n)$.
8. Η quicksort (επίσης διαίρει και βασίλευε) διαμερίζει γύρω από ένα pivot· έχει χρόνο $O(n \log n)$ κατά μέση περίπτωση αλλά $O(n^2)$ στη χειρότερη, και χώρο $O(n)$ στην υλοποίηση της διάλεξης ($O(\log n)$ με κατάλληλη υλοποίηση).
9. Η `stdlib.h` προσφέρει έτοιμες την `bsearch` (δυαδική αναζήτηση) και την `qsort` (ταξινόμηση).

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
αναζήτηση	search	Εύρεση ενός στοιχείου (και της θέσης του) σε μια συλλογή.
γραμμική / σειριακή αναζήτηση	linear / serial search	Έλεγχος των στοιχείων ένα προς ένα· $O(n)$.
δυαδική αναζήτηση	binary search	Σύγκριση με το μέσο και συνέχεια στο μισό· $O(\log n)$.
ταξινομημένη ακολουθία	sorted sequence	$a_i \leq_\alpha a_j$ για κάθε $i \leq j$.
ταξινόμηση	sorting	Αναδιάταξη μιας ακολουθίας ώστε να γίνει ταξινομημένη.
αντιμετάθεση	swap	Ανταλλαγή των τιμών δύο θέσεων.
ταξινόμηση επιλογής	selection sort	Φέρνει κάθε φορά το ελάχιστο του υπολοίπου μπροστά.
ταξινόμηση εισαγωγής	insertion sort	Εισάγει κάθε στοιχείο σε ταξινομημένο πρόθεμα.
ταξινόμηση φυσαλίδας	bubblesort	Αντιμεταθέτει γειτονικά στοιχεία σε λάθος σειρά.
διαίρει και βασίλευε	divide and conquer	Διαίρεση σε υποπροβλήματα, αναδρομική λύση, συνδυασμός.
ταξινόμηση συγχώνευσης	merge sort	Ταξινομεί τα δύο μισά και τα συγχωνεύει.
συγχώνευση	merge	Ένωση δύο ταξινομημένων ακολουθιών σε μία ταξινομημένη.
ταχυταξινόμηση	quicksort	Διαμερίζει γύρω από ένα pivot και ταξινομεί τα δύο μέρη.
στοιχείο διαμέρισης	pivot element	Το στοιχείο γύρω από το οποίο γίνεται η διαμέριση.
χειρότερη / μέση περίπτωση	worst / average case	Κόστος για τη δυσκολότερη / κατά μέσο όρο είσοδο.

Διάβασμα

- **Διαφάνειες**: Διάλεξη 17, σελ. 1–37: γρίφοι αναζήτησης 5–8· γραμμική αναζήτηση 9–11· ταξινομημένη ακολουθία 12· δυαδική αναζήτηση 13–19· swap 20–22· selection sort 23–24· insertion sort 25–26· bubblesort 27–28· merge sort 29–32· quicksort 33–35.
- **Σημειώσεις**: η διάλεξη καλύπτει τις σελ. 160–177 των σημειώσεων του κ. Σταματόπουλου: Κεφάλαιο 11: Ταξινόμηση και αναζήτηση, ενότητες «Ταξινόμηση πινάκων» (K04, σελ. 160–166), «Μέθοδοι ταξινόμησης» (167–171), «Αναζήτηση σε πίνακες» (172) και «Μέθοδοι αναζήτησης» (173–177).
- **Εργαστήριο**: κανένα εργαστήριο δεν είναι αφιερωμένο στην ταξινόμηση ή στην αναζήτηση.
- **Άλλα**: Visualizing binary search· Binary search algorithm και οι δυσκολίες υλοποίησής της (Wikipedia)· άρθρο για λάθη υλοποίησης σε βιβλία (και στην ACM Digital Library)· Generalizing std::midpoint και βίντεο για τον μέσο· ο γρίφος του Ballmer· Divide-and-conquer algorithm· Sorting algorithm (εκτενής λίστα αλγορίθμων)· Quicksort analysis· quicksort σε χώρο $O(\log n)$· John von Neumann, Tony Hoare· `man 3 qsort`, `man 3 bsearch`.

Συχνά λάθη

- **Δυαδική αναζήτηση σε αταξινόμητο πίνακα.** Δεν βγάζει σφάλμα, απλώς απαντά «δεν υπάρχει» για στοιχεία που υπάρχουν. Ταξινομήστε πρώτα ή χρησιμοποιήστε γραμμική.
- `mid = (low + high) / 2`. Υπερχειλίζει όταν `low + high > INT_MAX`: αρνητικό `mid` και συνήθως `Segmentation fault`. Γράψτε `low + (high - low) / 2`.
- `while (low < high)` **αντί για** `<=`. Σε πίνακα ενός στοιχείου, ή όταν μένει ένα υποψήφιο, το στοιχείο δεν ελέγχεται ποτέ: η αναζήτηση του 5 στο `{5}` αποτυγχάνει.
- `low = mid` **ή** `high = mid`. Όταν `low == mid`, το διάστημα δεν μικραίνει και ο βρόχος δεν τερματίζει. Χρησιμοποιήστε `mid + 1` και `mid - 1`.
- `swap(int a, int b)` **ή** `swap(x[i], x[j])`. Η συνάρτηση αλλάζει αντίγραφα και ο πίνακας μένει ίδιος (ή ο `gcc` διαμαρτύρεται `makes pointer from integer without a cast`). Περάστε διευθύνσεις: `swap(&x[i], &x[j])`.
- **Λάθος όρια στους βρόχους ταξινόμησης.** Π.χ. στη bubblesort `j >= 0` αντί για `j >= i` διαβάζει το `x[-1]`· στην insertion sort `x[j] > x[j+1] && j >= 0` διαβάζει το `x[-1]` πριν ελέγξει το `j`. Ελέγχετε πρώτα το όριο.
- **Λάθος αρχική κλήση των αναδρομικών.** Οι `merge_sort` και `quicksort` παίρνουν θέσεις πρώτου και **τελευταίου** στοιχείου: `merge_sort(a, 0, n - 1)`, όχι `merge_sort(a, 0, n)`, που διαβάζει εκτός ορίων.
- **«Η quicksort είναι πάντα $O(n \log n)$».** Μόνο κατά μέση περίπτωση· με κακή επιλογή pivot γίνεται $O(n^2)$.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K17.8** Η πιο γρήγορη ταξινόμηση στη χειρότερη περίπτωση (28% σωστές): Το 35% επέλεξε την quicksort, που είναι O(n log n) στη μέση περίπτωση αλλά O(n²) στη χειρότερη (κακή επιλογή pivot). Ένα 27% επέλεξε την bubblesort, ίσως επειδή τελειώνει γρήγορα σε ήδη ταξινομημένο πίνακα, που όμως είναι η καλύτερη και όχι η χειρότερη περίπτωση.
- **K17.7** Αναζήτηση αριθμημένης θέσης σε αίθουσα (48% σωστές): Το 33% επέλεξε O(n²), σαν να έπρεπε να ελέγξει κάθε θέση μία-μία. Επειδή όμως οι θέσεις είναι ταξινομημένες, δεν χρειάζεται να τις κοιτάξετε όλες.

Ερωτήσεις κατανόησης

- **E17.1** Ποια προϋπόθεση πρέπει να ισχύει για να εφαρμόσουμε δυαδική αναζήτηση, και γιατί η γραμμική δεν την χρειάζεται;[137]
- **E17.2** Γιατί το `low + (high - low) / 2` είναι ασφαλέστερο από το `(low + high) / 2`;[138]
- **E17.3** Μετά τον γύρο `i` της selection sort, τι ισχύει για τις θέσεις `0` έως `i - 1`;[139]
- **E17.4** Ποια είναι η χειρότερη είσοδος για την insertion sort και ποια η καλύτερη;[140]
- **E17.5** Γιατί η merge sort χρειάζεται χώρο $O(n)$ ενώ η selection sort $O(1)$;[141]
- **E17.6** Πότε η quicksort γίνεται $O(n^2)$;[142]

Kahoot από το αμφιθέατρο (K17.1–K17.8)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K17.1** Δυαδική αναζήτηση σε μη ταξινομημένο πίνακα: 94% σωστές απαντήσεις
- **K17.2** Αναζήτηση σε μη ταξινομημένο πίνακα: 85% σωστές απαντήσεις
- **K17.3** Πολυπλοκότητα της bubblesort: 75% σωστές απαντήσεις
- **K17.4** Ταξινόμηση ενός εκατομμυρίου ακεραίων: 68% σωστές απαντήσεις
- **K17.5** Μέση πολυπλοκότητα της quicksort: 66% σωστές απαντήσεις
- **K17.6** Βήματα δυαδικής αναζήτησης σε 2^50 στοιχεία: 52% σωστές απαντήσεις
- **K17.7** Αναζήτηση αριθμημένης θέσης σε αίθουσα: 48% σωστές απαντήσεις

[137] Ο πίνακας πρέπει να είναι ταξινομημένος, για να ξέρουμε σε ποιο μισό μπορεί να βρίσκεται το στοιχείο. Η γραμμική ελέγχει όλα τα στοιχεία, άρα δεν χρειάζεται σειρά.

[138] Το `high - low` δεν ξεπερνά το μέγεθος του πίνακα, ενώ το `low + high` μπορεί να ξεπεράσει το `INT_MAX` και να υπερχειλίσει.

[139] Περιέχουν τα `i` μικρότερα στοιχεία του πίνακα, ταξινομημένα και στις τελικές τους θέσεις.

[140] Χειρότερη: πίνακας σε φθίνουσα σειρά ($O(n^2)$ αντιμεταθέσεις). Καλύτερη: ήδη ταξινομημένος, όπου ο `while` σταματά αμέσως σε κάθε γύρο.

[141] Η `merge` αντιγράφει τα δύο τμήματα σε βοηθητικούς πίνακες, ενώ η selection sort μόνο αντιμεταθέτει στοιχεία μέσα στον ίδιο πίνακα.

[142] Όταν το pivot είναι κάθε φορά το μικρότερο ή το μεγαλύτερο στοιχείο, οπότε κάθε διαμέριση αφαιρεί μόνο ένα στοιχείο.

- **K17.8** Η πιο γρήγορη ταξινόμηση στη χειρότερη περίπτωση: 28% σωστές απαντήσεις

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A17.1–A17.9)

- **A17.1** Πολυπλοκότητα της δυαδικής αναζήτησης: Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 18 · ★☆☆ · short-answer · `slides-lec17-binary-search-complexity`
- **A17.2** Γρηγορότερα από O(n);: Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 11 · ★☆☆ · short-answer · `slides-lec17-faster-than-linear`
- **A17.3** Κρυμμένος αριθμός έως 10^9: Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 6 · ★☆☆ · short-answer · `slides-lec17-guess-billion`
- **A17.4** Αναζήτηση χρήστη στο Instagram: Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 19 · ★☆☆ · short-answer · `slides-lec17-instagram`
- **A17.5** Αναζήτηση σε πίνακα 100 ακεραίων: Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 9 · ★☆☆ · programming · `slides-lec17-linear-search`
- **A17.6** Η συνάρτηση swap: Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 21 · ★☆☆ · programming · `slides-lec17-swap`
- **A17.7** Δυαδική αναζήτηση σε ταξινομημένο πίνακα: Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 14 · ★★☆ · programming · `slides-lec17-binary-search`
- **A17.8** Το σφάλμα της δυαδικής αναζήτησης: Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 16 · ★★☆ · debug · `slides-lec17-binary-search-bug`
- **A17.9** Ο γρίφος του κρυμμένου αριθμού: Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 5 · ★★☆ · short-answer · `slides-lec17-guess-100`

Θέματα εξετάσεων (A17.10–A17.13)

- **A17.10** Η συνάρτηση what: Εξέταση Σεπτεμβρίου 2024, Θέμα 2 · ★☆☆ · trace · `exam-2024-sep-q2`
- **A17.11** Πλησιάζοντας στον Στόχο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #10, Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex10-q4`
- **A17.12** Εύρεση μηδενός σε πίνακα: Εξέταση Σεπτεμβρίου 2024, Θέμα 3 · ★★☆ · programming · `exam-2024-sep-q3`
- **A17.13** Η συνάρτηση compute: Εξέταση Ιανουαρίου 2026, Θέμα 2 · ★★☆ · trace · `exam-2026-jan-q2`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A16.25** Χτίζοντας έναν Χιονάνθρωπο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #3 (Frozen Themed), Θέμα 4 · ★★★ · programming · `exam-2023-fall-ex3-q4`

- **A18.16** Μετρήσεις Θερμοκρασίας: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #14, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex14-q3`
- **A18.21** Ταξινομώντας τα Άλματα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #15, Θέμα 3 · ★★★ · programming · `exam-2023-fall-ex15-q3`
- **A19.11** Πρωτάθλημα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #13, Θέμα 3 · ★★★ · programming · `exam-2023-fall-ex13-q3`
- **A25.6** Μένοντας στις Σωστές Θερμίδες: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #14, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex14-q2`
- **A25.11** Η Τριπλέτα Στόχος: Κατατακτήριες Δεκεμβρίου 2024, Θέμα 3 · ★★★ · programming · `exam-2024-dec-q3`
- **A25.13** Βέλτιστη Μοιρασιά Πίτσας: Εξέταση Ιουλίου 2024, Θέμα 3 · ★★★ · programming · `exam-2024-jul-q3`

Κεφάλαιο 18: Ταξινόμηση και Δεδομένα Εισόδου #2

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να συγκρίνετε τους πέντε αλγορίθμους ταξινόμησης ως προς χρόνο και χώρο και να ιχνηλατείτε τη merge sort και την quicksort· να διαβάζετε `double` και συμβολοσειρές με `scanf` χωρίς υπερχείλιση· να ανοίγετε, να διαβάζετε, να γράφετε και να κλείνετε αρχεία με `fopen`, `fread`, `fwrite`, `fscanf`, `fprintf` και `fclose`· και να εξηγείτε τι είναι οι `stdin`, `stdout`, `stderr` και οι file descriptors.

Προαπαιτούμενα: Κεφάλαιο 9 (πρώτο μέρος για την είσοδο), Κεφάλαιο 12 (endianness), Κεφάλαιο 17 (αλγόριθμοι ταξινόμησης)

Χρόνος μελέτης: ~2,5 ώρες

Σύνοψη

Η διάλεξη κλείνει την ταξινόμηση και ανοίγει το δεύτερο μέρος των δεδομένων εισόδου. Πρώτα ξαναβλέπει τους πέντε αλγορίθμους ταξινόμησης (επιλογής, εισαγωγής, φυσαλίδας, συγχώνευσης, ταχυταξινόμηση) με την πολυπλοκότητα χρόνου και χώρου του καθενός· οι δύο τελευταίοι, με τη στρατηγική «διαίρει και βασίλευε», πέφτουν στο $O(n \log n)$. Έπειτα επιστρέφει στη `scanf`, για `double` και για συμβολοσειρές, όπου ένα `%s` χωρίς όριο μπορεί να γράψει έξω από τον πίνακα. Το κύριο θέμα είναι η τρίτη πηγή εισόδου, τα **αρχεία**: ο τύπος `FILE`, το άνοιγμα και το κλείσιμο, το διάβασμα και το γράψιμο bytes με `fread`/`fwrite` ή κειμένου με `fscanf`/`fprintf`. Με αυτά τα προγράμματά σας μπορούν να κρατούν δεδομένα και αφού τερματίσουν.

Θεωρία

§18.1 Δύο διευκρινίσεις: sizeof και log n

Η δήλωση `int array[5][10];` δεσμεύει $5 \cdot 10$· `sizeof(int)` bytes, δηλαδή 200 με `int` των 4 bytes. Το `array[3]` είναι μία γραμμή, πίνακας 10 ακεραίων, άρα `sizeof(array[3])` είναι `10 * sizeof(int)` = 40 (Κεφάλαιο 12).

Πώς αναγνωρίζετε πολυπλοκότητα $O(\log n)$ (Κεφάλαιο 15); Δύο τρόποι:

- Ελέγξτε αν το πρόβλημα «σπάει» σε μικρότερα υποπροβλήματα παρόμοιου μεγέθους και αρκεί να λύσετε **μόνο ένα** από αυτά (όπως η δυαδική αναζήτηση).
- Παρατηρήστε πώς αυξάνονται οι επαναλήψεις όσο μεγαλώνει το N. Αν αυξάνονται **γραμμικά** ενώ το N αυξάνεται **εκθετικά**, η πολυπλοκότητα είναι $O(\log n)$: π.χ. $N = 10$ δίνει 1 επανάληψη, $N = 100$ δίνει 2, $N = 1000000$ δίνει 6.

§18.2 Γιατί ταξινομούμε

Αν το Instagram κρατά 2 δισεκατομμύρια χρήστες σε έναν πίνακα ακεραίων, η γραμμική αναζήτηση για τον χρήστη 424242 κάνει έως $2 \cdot 10^9$ συγκρίσεις. Σε **ταξινομημένο** πίνακα η δυαδική αναζήτηση χρειάζεται περίπου $\log_2(2 \cdot 10^9) \approx 31$. Η ταξινόμηση πληρώνεται μία φορά και κάνει κάθε επόμενη αναζήτηση φθηνή. Η διάλεξη εξετάζει πέντε **αλγορίθμους ταξινόμησης** (sorting algorithms): bubblesort, selection sort, insertion sort, merge sort και quicksort. Όλοι παίρνουν πίνακα `int` μέσω δείκτη και τον ταξινομούν σε αύξουσα σειρά επί τόπου.

§18.3 Η swap

Το βασικό βήμα των περισσότερων είναι η **αντιμετάθεση** (swap) δύο στοιχείων. Επειδή στη C τα ορίσματα περνούν με τιμή, η `swap` πρέπει να παίρνει **δείκτες** και να αλλάζει τις τιμές μέσω `*`:

```
void swap(int *a, int *b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}
```

Την καλούμε με διευθύνσεις: `swap(&a, &b)` ή `swap(&x[i], &x[j])`.

§18.4 Οι τρεις αλγόριθμοι O(n²)

Και οι τρεις έχουν δύο φωλιασμένους βρόχους πάνω στον πίνακα, άρα **χρόνο** $O(n^2)$, και χρειάζονται μόνο λίγες τοπικές μεταβλητές, άρα **χώρο** $O(1)$. Ο αναλυτικός κώδικας και η ιχνηλάτησή τους βρίσκονται στο Κεφάλαιο 17.

- **Ταξινόμηση επιλογής** (selection sort): στον γύρο `i` βρίσκει τη θέση `min` του μικρότερου στοιχείου από τη θέση `i - 1` ως το τέλος και κάνει `swap(&x[i-1], &x[min])`. Μετά από $n - 1$ γύρους ο πίνακας είναι ταξινομημένος.
- **Ταξινόμηση εισαγωγής** (insertion sort): το πρόθεμα `x[0..i-1]` είναι ήδη ταξινομημένο· το `x[i]` «βουλιάζει» αριστερά με `swap(&x[j], &x[j+1])` όσο `j >= 0 && x[j] > x[j+1]`. Σε πίνακα ήδη ταξινομημένο ο `while` δεν εκτελείται ποτέ.
- **Ταξινόμηση φυσαλίδας** (bubblesort): για `j` από `n - 1` ως `i`, αν `x[j-1] > x[j]` τα αντιμεταθέτει. Κάθε πέρασμα ανεβάζει το μικρότερο από τα υπόλοιπα στη θέση `i - 1`,

σαν φυσαλίδα.

§18.5 Διαίρει και βασίλευε: merge sort

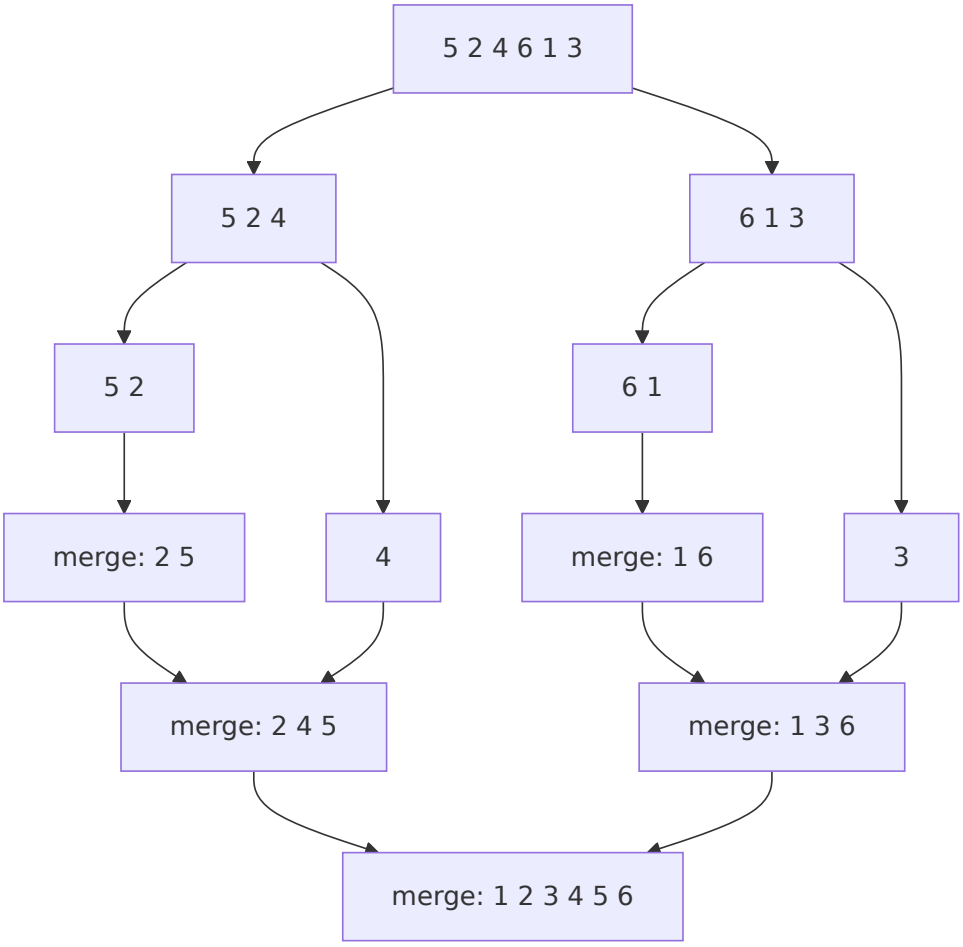
Ένας αλγόριθμος **διαίρει και βασίλευε** (divide and conquer) χωρίζει το πρόβλημα σε μικρότερα του ίδιου είδους, τα λύνει αναδρομικά και συνδυάζει τις λύσεις. Η **ταξινόμηση συγχώνευσης** (merge sort, του John von Neumann) έχει θεωρητικά την καλύτερη πολυπλοκότητα και δύο βήματα:

1. χώρισε τον πίνακα σε δύο υποπίνακες και κάλεσε merge sort σε καθέναν·
2. **συγχώνευσε** τα στοιχεία των δύο ταξινομημένων υποπινάκων.

```
void merge_sort(int *array, int left, int right) {
  if (left < right) {
    int middle = left + (right - left) / 2;
    merge_sort(array, left, middle);
    merge_sort(array, middle + 1, right);
    merge(array, left, middle, right);
  }
}
```

Το `left + (right - left) / 2` αποφεύγει την υπερχείλιση του `(left + right) / 2`. Η `merge` αντιγράφει τα ταξινομημένα `x[l..m]` και `x[m+1..r]` σε δύο βοηθητικούς πίνακες και γράφει πίσω στο `x` κάθε φορά το μικρότερο από τα δύο πρώτα αχρησιμοποίητα στοιχεία· όταν ο ένας εξαντληθεί, αντιγράφει τα υπόλοιπα του άλλου:

```
void merge(int *x, int l, int m, int r) {
  int i, j, k, n1 = m - l + 1, n2 = r - m;
  int left[n1], right[n2];
  for (i = 0; i < n1; i++) left[i] = x[l + i];
  for (j = 0; j < n2; j++) right[j] = x[m + 1 + j];
  i = 0; j = 0; k = l;
  while (i < n1 && j < n2) {
    if (left[i] <= right[j]) x[k++] = left[i++];
    else x[k++] = right[j++];
  }
  while (i < n1) x[k++] = left[i++];
  while (j < n2) x[k++] = right[j++];
}
```



Σχήμα: η merge sort χωρίζει ως τα μονά στοιχεία και συγχωνεύει προς τα πάνω.

Η διχοτόμηση δίνει περίπου $\log_2 n$ επίπεδα και σε κάθε επίπεδο οι συγχωνεύσεις αγγίζουν συνολικά n στοιχεία: **χρόνος** $O(n \log n)$, πάντα. Οι βοηθητικοί πίνακες κοστίζουν **χώρο** $O(n)$.

§18.6 Ταχυταξινόμηση (quicksort)

Η **ταχυταξινόμηση** (quicksort, του Tony Hoare) είναι επίσης διαίρει και βασίλευε και ιδιαίτερα δημοφιλής. Τρία βήματα:

1. διάλεξε (π.χ. τυχαία) ένα **στοιχείο διαμέρισης** (pivot element)·
2. **διαμέρισε** τον πίνακα: αριστερά τα στοιχεία μικρότερα του pivot, δεξιά τα μεγαλύτερα·
3. τρέξε ταχυταξινόμηση στους δύο υποπίνακες.

```
void quicksort (int *x, int lower, int upper) {
  if (lower < upper) {
    int pivot = x[(lower + upper) / 2];
    int i, j;
```

```
    for (i = lower, j = upper; i <= j;) {
      while (x[i] < pivot) i++;
      while (x[j] > pivot) j--;
      if (i <= j) swap(&x[i++], &x[j--]);
    }
    quicksort(x, lower, j);
    quicksort(x, i, upper);
  }
}
```

Το pivot εδώ είναι το μεσαίο στοιχείο. Το `i` προχωρά από αριστερά και το `j` από δεξιά ώσπου να βρουν στοιχεία στη λάθος πλευρά, τα οποία αντιμεταθέτουν. Όταν διασταυρωθούν, το `x[lower..j]` έχει τα μικρά και το `x[i..upper]` τα μεγάλα, οπότε δεν χρειάζεται συγχώνευση.

Αν το pivot μοιράζει περίπου στη μέση, έχουμε $\log_2 n$ επίπεδα: $O(n \log n)$ **κατά μέση περίπτωση** (average case). Αν είναι κάθε φορά το μικρότερο ή το μεγαλύτερο στοιχείο, το ένα κομμάτι μικραίνει μόνο κατά ένα: $O(n^2)$ **στη χειρότερη περίπτωση** (worst case). Ο χώρος είναι η στοίβα της αναδρομής, $O(n)$ σε αυτή την υλοποίηση· γίνεται $O(\log n)$ αν η αναδρομή γίνεται πάντα πρώτα στο μικρότερο κομμάτι και το μεγαλύτερο χειρίζεται βρόχος. Η quicksort είναι υλοποιημένη στη συνάρτηση `qsort` της `stdlib.h` (`man 3 qsort`).

§18.7 Σύγκριση των αλγορίθμων

Αλγόριθμος	Χρόνος	Χώρος
Selection sort	$O(n^2)$	$O(1)$
Insertion sort	$O(n^2)$	$O(1)$
Bubblesort	$O(n^2)$	$O(1)$
Merge sort	$O(n \log n)$	$O(n)$
Quicksort	$O(n \log n)$ μέση, $O(n^2)$ χειρότερη	$O(n)$ εδώ, $O(\log n)$ βελτιωμένη

Η merge sort εγγυάται $O(n \log n)$ αλλά θέλει βοηθητικούς πίνακες· η quicksort δεν θέλει, αλλά η χειρότερη περίπτωσή της είναι $O(n^2)$.

§18.8 Οι τέσσερις πηγές εισόδου

Τα **δεδομένα εισόδου** (input data) είναι μια σειρά από χαρακτήρες (bytes) που ο χρήστης δίνει στο πρόγραμμα, το οποίο παράγει δεδομένα εξόδου (output data). Υπάρχουν τέσσερις μέθοδοι (Κεφάλαιο 9): ορίσματα γραμμής εντολών, πρότυπη είσοδος, αρχεία και δίκτυο ή άλλες πηγές. Τα ορίσματα (`./grade 80 100 90`) και η πρότυπη είσοδος (`./aliquot` που ρωτά τον χρήστη) έχουν καλυφθεί. Σήμερα προστίθενται τα **αρχεία** (όπως η `cat hello.txt`), οπότε είμαστε στο «75%». Το δίκτυο (π.χ. `curl`) ανήκει σε επόμενα εξάμηνα.

§18.9 Η scanf με double και συμβολοσειρές

Για `double` η `scanf` θέλει `%lf` και τη διεύθυνση της μεταβλητής, π.χ. `scanf("%lf %lf", &d1, &d2);`. (Η `printf` τυπώνει `double` με `%f`· το `%.1f` κρατά ένα δεκαδικό.)

Για συμβολοσειρά, το `%s` διαβάζει μια **λέξη**: παραλείπει τα αρχικά κενά και σταματά στο επόμενο λευκό χαρακτήρα, προσθέτοντας `'\0'` στο τέλος. Το όρισμα είναι το όνομα του πίνακα **χωρίς** `&`: το `message` μετατρέπεται ήδη σε δείκτη στο πρώτο στοιχείο (Κεφάλαιο 12).

Το `%s` **δεν ελέγχει** το μέγεθος του πίνακα. Σε `char message[7]` χωρούν 6 χαρακτήρες και το `'\0'`· μια μεγαλύτερη λέξη γράφεται πέρα από το τέλος του, με απροσδιόριστη συμπεριφορά (συχνά `Segmentation fault`). Η λύση είναι ένα **πλάτος πεδίου**: το `%6s` διαβάζει το πολύ 6 χαρακτήρες. Ο κανόνας είναι πλάτος = μέγεθος πίνακα − 1. Οι χαρακτήρες που δεν διαβάστηκαν μένουν στην είσοδο για την επόμενη ανάγνωση.

§18.10 Η gets και το %ms

Η `gets(char *s)` διαβάζει μια ολόκληρη γραμμή από την `stdin` χωρίς κανένα όριο, όπως το `scanf("%s")`. Το εγχειρίδιο (`man gets`) τη χαρακτηρίζει DEPRECATED και γράφει «Never use this function»· αποφεύγεται για λόγους ασφαλείας (για γραμμές χρησιμοποιήστε `fgets`, παρακάτω).

Αν θέλετε οπωσδήποτε `scanf`, το `%ms` αφήνει τη `scanf` να δεσμεύσει με `malloc` όση μνήμη χρειάζεται. Δίνουμε τη διεύθυνση ενός `char *` (`&string`), ελέγχουμε ότι η `scanf` επέστρεψε 1 και στο τέλος καλούμε `free(string)`. Ο τροποποιητής `m` είναι επέκταση POSIX (υπάρχει στη glibc του Linux), όχι μέρος του προτύπου C.

§18.11 Αρχεία και σύστημα αρχείων

Ένα **αρχείο** (file) είναι ένας πόρος για να καταγράφουμε δεδομένα σε έναν υπολογιστή, συνήθως στη δευτερεύουσα μνήμη (π.χ. σκληρό δίσκο). Στο Linux σχεδόν **τα πάντα είναι αρχεία** (everything is a file). Το περιεχόμενο ενός αρχείου είναι απλώς ένας πίνακας από bytes, `char bytes[]`. Κάθε αρχείο:

- έχει ένα **όνομα** (filename/basename), π.χ. `students.txt`·
- βρίσκεται σε έναν **φάκελο** (directory/folder), π.χ. `/home/users/thanassis/documents`·
- έχει ένα πλήρες **μονοπάτι** (filepath) που λέει πού βρίσκεται: `/home/users/thanassis/documents/studen`

Το `.txt` λέγεται **επέκταση** (extension) και συνήθως περιγράφει τον τύπο του αρχείου (Κεφάλαιο 1).

§18.12 Ο τύπος FILE και η fopen

Ο τύπος `FILE` ορίζεται στην `stdio.h` και αναπαριστά ένα αρχείο που άνοιξε το πρόγραμμα. Είναι μια δομή με πολλά εσωτερικά πεδία: το `printf("%zu\n", sizeof(FILE));` τυπώνει 216 σε

ένα σύστημα Debian. Τι περιέχουν αυτά τα bytes είναι θέμα της υλοποίησης· εμείς δουλεύουμε πάντα με δείκτη `FILE *`, που λέγεται και **ρεύμα** (stream).

```
FILE *fopen(const char *restrict pathname, const char *restrict mode);
```

Η `fopen` ανοίγει το αρχείο `pathname` με τον τρόπο `mode` και επιστρέφει `FILE *`, ή `NULL` αν αποτύχει για οποιονδήποτε λόγο. Τα βασικά `mode` (`man 3 fopen`):

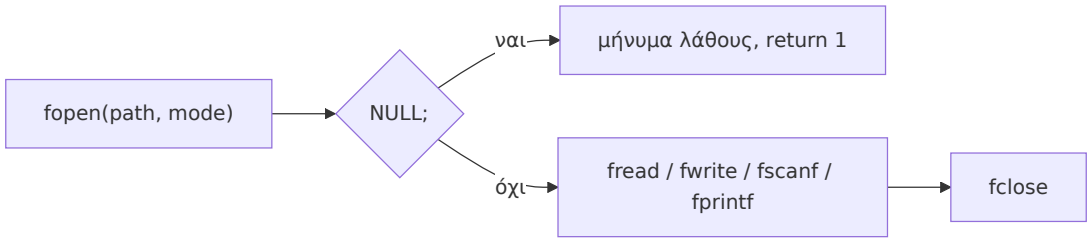
mode	Σημασία	Αν δεν υπάρχει	Αρχική θέση
`"r"`	διάβασμα	αποτυχία	αρχή
`"r+"`	διάβασμα και γράψιμο	αποτυχία	αρχή
`"w"`	γράψιμο, **σβήνει** το περιεχόμενο	δημιουργείται	αρχή
`"w+"`	διάβασμα και γράψιμο, σβήνει	δημιουργείται	αρχή
`"a"`	προσάρτηση (γράψιμο στο τέλος)	δημιουργείται	τέλος
`"a+"`	διάβασμα και προσάρτηση	δημιουργείται	τέλος για γράψιμο

Ελέγχουμε **πάντα** αν η `fopen` επέστρεψε `NULL`. Αποτυγχάνει αν το αρχείο ή ο φάκελος δεν υπάρχει, αν δεν υπάρχει ελεύθερος δίσκος για να γράψουμε, αν δεν έχουμε δικαιώματα να φτιάξουμε ή να διαβάσουμε το αρχείο, αν έχουμε ανοίξει τον μέγιστο επιτρεπτό αριθμό αρχείων, κ.ά.

§18.13 Η fclose

```
int fclose(FILE *stream);
```

Η `fclose` κλείνει ένα αρχείο και επιστρέφει 0 αν επιτύχει ή `EOF` αν αποτύχει. **Πάντα** κλείνουμε τα αρχεία μόλις τελειώσουμε: σε κάθε `fopen` αντιστοιχεί ένα `fclose`, όπως σε κάθε `malloc` ένα `free` (Κεφάλαιο 13).



Σχήμα: ο κύκλος ζωής ενός αρχείου μέσα στο πρόγραμμα.

§18.14 Διάβασμα και γράψιμο bytes: fread και fwrite

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *restrict stream);
```

```
size_t fwrite(const void *ptr, size_t size, size_t nmemb,
              FILE *restrict stream);
```

Η `fread` διαβάζει από το ανοιχτό αρχείο `stream` το πολύ `nmemb` δεδομένα των `size` bytes το καθένα και τα αποθηκεύει από τη διεύθυνση `ptr` και μετά. Η `fwrite` κάνει το αντίστροφο: γράφει `nmemb` δεδομένα των `size` bytes, παίρνοντάς τα από το `ptr`. Και οι δύο επιστρέφουν **πόσα δεδομένα** (όχι bytes) διαβάστηκαν ή γράφτηκαν· ένα μισό δεδομένο στο τέλος του αρχείου δεν μετράει.

Οι δύο συναρτήσεις αντιγράφουν bytes **αυτούσια**, χωρίς καμία μετατροπή. Ένα αρχείο κειμένου που περιέχει `hello` διαβασμένο ως `int` δεν δίνει αριθμό γραμμένο με ψηφία, αλλά τα bytes `68 65 6c 6c` ερμηνευμένα ως ακέραιο, με τη σειρά του endianness της μηχανής (Κεφάλαιο 12). Αντίστροφα, η `fwrite` ενός `int` γράφει τα 4 bytes της μνήμης του, όχι το κείμενο του αριθμού. Για τα `char` το `size` είναι 1 και τα δεδομένα είναι bytes· για να τυπώσουμε ό,τι διαβάσαμε ως συμβολοσειρά, αφήνουμε μία θέση και βάζουμε `'\0'` στο τέλος.

§18.15 File descriptors: stdin, stdout, stderr

Κάθε ανοιχτό αρχείο ενός προγράμματος έχει έναν μοναδικό ακέραιο, τον **file descriptor** (FD), που τον βρίσκουμε με τη `fileno(FILE *)`. Τρία ρεύματα ανοίγουν αυτόματα όταν ξεκινά το πρόγραμμα και κλείνουν όταν τερματίζει:

Ρεύμα	Αντιστοιχεί σε	FD
`stdin`	πρότυπη είσοδο (standard input)	0
`stdout`	πρότυπη έξοδο (standard output)	1
`stderr`	έξοδο σφάλματος (standard error)	2

Γι' αυτό τα πρώτα αρχεία που ανοίγουμε παίρνουν συνήθως τους αριθμούς 3, 4, ... Το shell χρησιμοποιεί τους ίδιους αριθμούς στις ανακατευθύνσεις: το `2> error.txt` στέλνει στο αρχείο μόνο την έξοδο σφάλματος. Μηνύματα λάθους γράφονται λοιπόν στο `stderr` (`fprintf(stderr, ...)`), ώστε να μην ανακατεύονται με τα αποτελέσματα.

§18.16 Κείμενο σε αρχεία: fscanf και fprintf

```
int fscanf(FILE *stream, const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
```

Η `fscanf` είναι η `scanf` με ένα επιπλέον πρώτο όρισμα, το αρχείο από το οποίο διαβάζει· η `fprintf` είναι η `printf` για οποιοδήποτε αρχείο. Η κλήση `scanf(x, y, z)` είναι ουσιαστικά ίδια με `fscanf(stdin, x, y, z)`, και η `printf(x, y, z)` με `fprintf(stdout, x, y, z)`. Σε αντίθεση με τη `fread`, αυτές **μετατρέπουν** κείμενο σε τιμές και αντίστροφα: το `"%d"` διαβάζει τους χαρακτήρες `42` ως τον ακέραιο 42.

§18.17 Άλλες χρήσιμες συναρτήσεις

Δήλωση	Τι κάνει (σημειώσεις, κεφ. 9)
`char *fgets(char *s, int size, FILE *stream);`	διαβάζει μια γραμμή, το πολύ `size - 1` χαρακτήρες μαζί με το `\n`· `NULL` στο τέλος
`int feof(FILE *stream);`	μη μηδενικό αν έχουμε φτάσει στο τέλος του αρχείου
`int fseek(FILE *stream, long offset, int whence);`	μετακινεί την τρέχουσα θέση (`SEEK_SET`, `SEEK_CUR`, `SEEK_END`)
`int fgetc(FILE *stream);`	επόμενος χαρακτήρας ή `EOF` (η `getchar` για αρχεία)
`int fputc(int c, FILE *stream);`	γράφει έναν χαρακτήρα (η `putchar` για αρχεία)

Παραδείγματα

§18.18 Merge sort και quicksort βήμα προς βήμα

Στον πίνακα `{5, 2, 4, 6, 1, 3}` η `merge_sort(x, 0, 5)` καλεί τις συγχωνεύσεις με τη σειρά `[0..1]`, `[0..2]`, `[3..4]`, `[3..5]`, `[0..5]`, όπως στο σχήμα της Θεωρίας. Η `quicksort(x, 0, 5)` στον ίδιο πίνακα (Θεωρία: «Ταχυταξινόμηση»):

Κλήση	pivot	Πίνακας μετά τη διαμέριση
`quicksort(x, 0, 5)`	4	3 2 1 6 4 5
`quicksort(x, 0, 2)`	2	1 2 3 6 4 5
`quicksort(x, 3, 5)`	4	1 2 3 4 6 5
`quicksort(x, 4, 5)`	6	1 2 3 4 5 6

Οι υπόλοιπες κλήσεις έχουν `lower >= upper` και επιστρέφουν αμέσως. Πλήρη προγράμματα με `main` για όλους τους αλγορίθμους θα βρείτε στο Κεφάλαιο 17.

§18.19 Υποτείνουσα με scanf

«Τι κάνει το παρακάτω πρόγραμμα;» (Θεωρία: «Η scanf με double και συμβολοσειρές»).

```
#include <stdio.h>
#include <math.h>

int main() {
  double d1, d2;
  printf("Gimme two doubles: ");
```

```
  scanf("%lf %lf", &d1, &d2);
  printf("Hypotenuse: %.1f\n", sqrt(d1 * d1 + d2 * d2));
  return 0;
}
```

Διαβάζει δύο double και τυπώνει, με ένα δεκαδικό, την υποτείνουσα ορθογωνίου τριγώνου με αυτές τις κάθετες πλευρές (αν ο linker δεν βρίσκει τη sqrt, προσθέστε -lm στον gcc).

```
$ ./hypotenuse
Gimme two doubles: 3.0 4.0
Hypotenuse: 5.0
```

§18.20 Μια λέξη σε char[7]

(Θεωρία: «Η scanf με double και συμβολοσειρές».)

```
#include <stdio.h>

int main() {
  char message[7];
  printf("Say something: ");
  scanf("%s", message);
  printf("%s\n", message);
  return 0;
}
```

Με τη λέξη hello! (6 χαρακτήρες και '\0') όλα πάνε καλά. Δεν βάλαμε & πριν το message, γιατί το όνομα του πίνακα είναι ήδη διεύθυνση. «Μπορεί να πάει κάτι στραβά εδώ;» Ναι:

```
$ ./message
Say something: hello!
hello!
$ ./message
Say something: Houston, we've had a problem here.
Houston,
Segmentation fault
```

Το %s διάβασε το Houston, (8 χαρακτήρες και '\0') σε πίνακα 7 θέσεων, χωρίς κανέναν έλεγχο. Με scanf("%6s", message); το πρόγραμμα τυπώνει Housto.

§18.21 Συμβολοσειρά οποιουδήποτε μήκους με %ms

(Θεωρία: «Η gets και το %ms».)

```
#include <stdio.h>
#include <stdlib.h>
```

```c
int main(int argc, char **argv) {
  char *string;
  int items_read;
  items_read = scanf("%ms", &string);
  if (items_read != 1) {
    fprintf(stderr, "No matching characters\n");
    return 1;
  }
  printf("read the following string: %s\n", string);
  // don't forget to free!
  free(string);
  return 0;
}
```

§18.22 Άνοιγμα και κλείσιμο αρχείων

(Θεωρία: «Ο τύπος FILE και η fopen», «Η fclose».) Το `input.txt` ανοίγει για διάβασμα και το `output.txt` για γράψιμο (δημιουργείται ή αδειάζει). Κάθε `fopen` ελέγχεται για `NULL` και κάθε αρχείο που άνοιξε κλείνει.

```c
#include <stdio.h>

int main() {
  FILE *fileToRead, *fileToWrite;
  fileToRead = fopen("input.txt", "r");
  if (!fileToRead) {
    return 1;
  }
  fileToWrite = fopen("output.txt", "w");
  if (!fileToWrite) {
    fclose(fileToRead);
    return 1;
  }
  // ... read and write ...
  fclose(fileToRead);
  fclose(fileToWrite);
  return 0;
}
```

Αν η δεύτερη `fopen` αποτύχει, το πρώτο αρχείο είναι ήδη ανοιχτό· γι' αυτό το κλείνουμε πριν το `return 1` (η διαφάνεια απλώς επιστρέφει).

§18.23 Διάβασμα κειμένου με fread

(Θεωρία: «Διάβασμα και γράψιμο bytes: fread και fwrite».)

```c
#include <stdio.h>

int main() {
  FILE *fileToRead;
  fileToRead = fopen("input.txt", "r");
  if (!fileToRead) return 1;
  char buffer[1024];
  size_t bytesRead = fread(buffer, sizeof(char), 1023, fileToRead);
  buffer[bytesRead] = '\0';
  printf("# of bytes read: %zu\n", bytesRead);
  printf("String read: %s\n", buffer);
  fclose(fileToRead);
  return 0;
}
```

Διαβάζουμε το πολύ 1023 bytes ώστε να μένει θέση για το `'\0'`. Η πρώτη εκτέλεση δεν τυπώνει τίποτα: το `input.txt` δεν υπάρχει, η `fopen` επιστρέφει `NULL` και το πρόγραμμα τερματίζει με 1. Η `echo` γράφει `hello` και αλλαγή γραμμής, άρα 6 bytes:

```
$ ./fread
$ echo hello > input.txt
$ ./fread
# of bytes read: 6
String read: hello
```

§18.24 «What?!»: ακέραιοι από αρχείο κειμένου

Ίδιο πρόγραμμα, αλλά με πίνακα ακεραίων (το `'\0'` φεύγει):

```c
int buffer[1024];
size_t integersRead = fread(buffer, sizeof(int), 1023, fileToRead);
printf("# of integers read: %zu\n", integersRead);
printf("Integer read: %d %08x\n", buffer[0], buffer[0]);
```

```
$ ./freadint
# of integers read: 1
Integer read: 1819043176 6c6c6568
$ hexdump -C input.txt
00000000  68 65 6c 6c 6f 0a  |hello.|
```

Τα 6 bytes του αρχείου χωρούν ένα ολόκληρο `int` (4 bytes)· τα 2 που περισσεύουν δεν σχηματίζουν δεδομένο, άρα η `fread` επιστρέφει 1. Τα bytes `68 65 6c 6c` (`h e l l`) διαβάζονται

σε μηχανή little endian ως `0x6c6c6568` = 1819043176 (Θεωρία: «Διάβασμα και γράψιμο bytes»). Η `fread` δεν ξέρει τίποτα από κείμενο· για να διαβάσετε αριθμούς γραμμένους με ψηφία χρησιμοποιήστε `fscanf`.

§18.25 Γράψιμο με fwrite

(Θεωρία: «Διάβασμα και γράψιμο bytes: fread και fwrite».)

```c
#include <stdio.h>

int main() {
  FILE *fileToWrite;
  fileToWrite = fopen("output.txt", "w");
  if (!fileToWrite) return 1;
  int numbers[4] = {0x42, 0x43, 0x44, 0x45};
  size_t numsWritten = fwrite(numbers, sizeof(int), 4, fileToWrite);
  printf("Wrote: %zu numbers\n", numsWritten);
  fclose(fileToWrite);
  return 0;
}
```

```
$ ./fwrite
Wrote: 4 numbers
$ hexdump -C output.txt
00000000  42 00 00 00 43 00 00 00  44 00 00 00 45 00 00 00  |B...C...D...E...|
```

Γράφτηκαν 16 bytes, 4 ανά ακέραιο, με το λιγότερο σημαντικό byte πρώτο (little endian). Το `hexdump` δείχνει δεξιά ως χαρακτήρες όσα bytes είναι εκτυπώσιμα: `0x42` είναι το `'B'`, και τα μηδενικά φαίνονται ως τελείες.

§18.26 Οι file descriptors ενός προγράμματος

(Θεωρία: «File descriptors: stdin, stdout, stderr».)

```c
fileToRead = fopen("input.txt", "r");
fileToWrite = fopen("output.txt", "w");
// ...
printf("fileToRead: %d\n", fileno(fileToRead));
printf("fileToWrite: %d\n", fileno(fileToWrite));
printf("stdin: %d\n", fileno(stdin));
printf("stdout: %d\n", fileno(stdout));
printf("stderr: %d\n", fileno(stderr));
```

```
$ ./fileno
fileToRead: 3
fileToWrite: 4
```

```
stdin: 0
stdout: 1
stderr: 2
```

Η διάλεξη προτείνει επίσης να συγκρίνετε το `find / -name foo` με το `find / -name foo 2> error.txt`: στο δεύτερο, τα μηνύματα λάθους (π.χ. για φακέλους χωρίς δικαιώματα) πηγαίνουν στο `error.txt` και στην οθόνη μένουν μόνο τα αποτελέσματα.

§18.27 fscanf και fprintf

(Θεωρία: «Κείμενο σε αρχεία: fscanf και fprintf».)

```c
#include <stdio.h>

int main() {
  FILE *fileToRead, *fileToWrite;
  fileToRead = fopen("input.txt", "r");
  fileToWrite = fopen("output.txt", "w");
  if (!fileToRead || !fileToWrite) return 1;
  int num;
  fscanf(fileToRead, "%d", &num);
  fprintf(fileToWrite, "Number: %d\n", num);
  fclose(fileToRead);
  fclose(fileToWrite);
  return 0;
}
```

```
$ echo "     42" > input.txt
$ ./stream
$ cat output.txt
Number: 42
```

Το `%d` παραλείπει τα αρχικά κενά, όπως και στη `scanf`, και μετατρέπει τους χαρακτήρες `42` στον ακέραιο 42. Το πρόγραμμα δεν τυπώνει τίποτα στην οθόνη: όλη η έξοδος πήγε στο αρχείο.

§18.28 Για το εργαστήριο

Το Εργαστήριο 10 εξασκεί ακριβώς αυτά: το `more.c` διαβάζει αρχείο κειμένου γραμμή-γραμμή (`fgets`), το `bgrades.c` γράφει και διαβάζει δυαδικό αρχείο με `fwrite`/`fread` (δείτε το με `hexdump -C`), το `filediff.c` συγκρίνει δύο αρχεία byte προς byte και το `count.c` μετρά χαρακτήρες και γραμμές όπως η `wc`. Σε όλα: το όνομα του αρχείου έρχεται από το `argv`, ελέγχετε το `argc` και το αποτέλεσμα της `fopen`, και κλείνετε ό,τι ανοίξατε.

Κύρια σημεία

1. Για `int array[5][10]`, το `sizeof(array[3])` είναι `10 * sizeof(int)`, μία γραμμή.
2. Πολυπλοκότητα $O(\log n)$ έχουμε όταν λύνουμε μόνο ένα από υποπροβλήματα παρόμοιου μεγέθους, ή όταν οι επαναλήψεις αυξάνονται γραμμικά ενώ το N αυξάνεται εκθετικά.
3. Η ταξινόμηση κάνει την αναζήτηση φθηνή: 2 δισεκατομμύρια στοιχεία θέλουν περίπου 31 βήματα δυαδικής αναζήτησης αντί για $2 \cdot 10^9$.
4. Selection, insertion και bubblesort έχουν χρόνο $O(n^2)$ και χώρο $O(1)$.
5. Η merge sort (διαίρει και βασίλευε) έχει χρόνο $O(n \log n)$ πάντα και χώρο $O(n)$ για τη συγχώνευση.
6. Η quicksort έχει χρόνο $O(n \log n)$ κατά μέση περίπτωση και $O(n^2)$ στη χειρότερη, χώρο $O(n)$ στην υλοποίηση της διάλεξης (βελτιώνεται σε $O(\log n)$), και είναι υλοποιημένη στην `qsort` της `stdlib.h`.
7. Τα δεδομένα εισόδου έρχονται από ορίσματα, πρότυπη είσοδο, αρχεία ή δίκτυο.
8. Η `scanf` διαβάζει `double` με `%lf`· με `%s` δεν ελέγχει το μέγεθος του πίνακα, άρα δίνετε πάντα πλάτος πεδίου (`%6s` για `char[7]`).
9. Η `gets` δεν χρησιμοποιείται ποτέ· το `%ms` δεσμεύει μνήμη που πρέπει να απελευθερώσετε με `free`.
10. Ένα αρχείο έχει όνομα, φάκελο και πλήρες μονοπάτι· το περιεχόμενό του είναι ένας πίνακας από bytes.
11. Η `fopen` επιστρέφει `FILE *` ή `NULL`· ελέγχουμε πάντα για `NULL`, και σε κάθε `fopen` αντιστοιχεί ένα `fclose`.
12. Οι `fread`/`fwrite` αντιγράφουν bytes αυτούσια και επιστρέφουν πόσα δεδομένα μεταφέρθηκαν· οι `fscanf`/`fprintf` μετατρέπουν κείμενο.
13. Οι `stdin`, `stdout`, `stderr` ανοίγουν αυτόματα με file descriptors 0, 1 και 2· η `scanf(...)` είναι η `fscanf(stdin, ...)` και η `printf(...)` η `fprintf(stdout, ...)`.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
ταξινόμηση	sorting	Αναδιάταξη στοιχείων σε αύξουσα (ή φθίνουσα) σειρά.
αντιμετάθεση	swap	Ανταλλαγή των τιμών δύο θέσεων μνήμης.
διαίρει και βασίλευε	divide and conquer	Σπάσε σε μικρότερα υποπροβλήματα, λύσε, συνδύασε.
συγχώνευση	merge	Ένωση δύο ταξινομημένων ακολουθιών σε μία ταξινομημένη.
στοιχείο διαμέρισης	pivot element	Το στοιχείο γύρω από το οποίο η quicksort χωρίζει τον πίνακα.

Ελληνικά	English	Σύντομος ορισμός
μέση / χειρότερη περίπτωση	average / worst case	Κόστος κατά μέσο όρο / για τη χειρότερη είσοδο.
πρότυπη είσοδος / έξοδος	standard input / output	Τα ρεύματα `stdin` / `stdout` του προγράμματος.
έξοδος σφάλματος	standard error	Το ρεύμα `stderr`, για μηνύματα λάθους.
πλάτος πεδίου	field width	Ο αριθμός στο `%6s`: μέγιστοι χαρακτήρες που θα διαβαστούν.
αρχείο	file	Πόρος καταγραφής δεδομένων, συνήθως στον δίσκο.
μονοπάτι	filepath	Η πλήρης θέση ενός αρχείου, π.χ. `/home/…/students.txt`.
επέκταση	extension	Το τέλος του ονόματος (`.txt`) που δείχνει τον τύπο.
ρεύμα	stream	Ένα ανοιχτό αρχείο, ως `FILE *`.
περιγραφέας αρχείου	file descriptor (FD)	Ο ακέραιος που ταυτίζει ένα ανοιχτό αρχείο (`fileno`).

Διάβασμα

- **Διαφάνειες**: Διάλεξη 18, σελ. 1–63: διευκρινίσεις 2· αλγόριθμοι ταξινόμησης 5–21· πηγές εισόδου 22–28· `scanf`, `gets`, `%ms` 29–37· αρχεία, `FILE`, `fopen`, `fclose` 38–46· `fread`, `fwrite` 47–54· file descriptors 55–56· `fscanf`, `fprintf` και άλλες συναρτήσεις 57–61· διάβασμα για την επόμενη φορά 62.
- **Σημειώσεις**: η διάλεξη αναφέρει ότι κάλυψε τις σελ. 136–151 και 160–177:
  - Κεφάλαιο 9: Είσοδος και έξοδος, ενότητες «Είσοδος και έξοδος» (K04, σελ. 136–149) και «Αντιγραφή αρχείων» (150–151).
  - Κεφάλαιο 11: Ταξινόμηση και αναζήτηση, ενότητες «Ταξινόμηση πινάκων» (K04, σελ. 160–166), «Μέθοδοι ταξινόμησης» (167–171), «Αναζήτηση σε πίνακες» (172) και «Μέθοδοι αναζήτησης» (173–177).
- **Εργαστήριο**: Εργαστήριο 10: ασκήσεις `more.c`, `bgrades.c`, `filediff.c`, `count.c`.
- **Άλλα**: οπτικοποιήσεις ταξινόμησης 1, 2, 3· Wikipedia: Sorting algorithm, Divide and conquer, John von Neumann, Tony Hoare, Scanf, Computer file, Everything is a file, File descriptor, Endianness· Quicksort analysis· Quicksort σε χώρο O(log n)· scanf specifiers· fread· `man 3 fopen`, `man 3 gets`.

Συχνά λάθη

- `swap(int a, int b)` **με τιμές**. Αλλάζει μόνο τα αντίγραφα· ο πίνακας μένει ίδιος. Περάστε δείκτες: `swap(&x[i], &x[j])`.
- `%s` **χωρίς πλάτος πεδίου**. Μια μακριά λέξη γράφει έξω από τον πίνακα και το πρόγραμμα καταρρέει (`Segmentation fault`) ή, χειρότερα, συνεχίζει με αλλοιωμένα δεδομένα. Γράψτε `%6s` για `char[7]`.
- `%f` **αντί για** `%lf` **στη** `scanf` **για** `double`. Ο `gcc -Wall` προειδοποιεί (`format '%f' expects argument of type 'float *'`) και η τιμή βγαίνει λάθος.
- `gets`. Ο `gcc` προειδοποιεί ότι είναι επικίνδυνη· χρησιμοποιήστε `fgets(buf, sizeof(buf), stdin)`.
- **Χωρίς έλεγχο της** `fopen`. Αν το αρχείο δεν υπάρχει, η `fopen` επιστρέφει `NULL` και η επόμενη `fread`/`fscanf` δίνει `Segmentation fault`. Ελέγξτε `if (!fp)` και τυπώστε μήνυμα στο `stderr`.
- `"w"` **σε αρχείο που θέλετε να κρατήσετε**. Το `"w"` σβήνει αμέσως το περιεχόμενο· για προσθήκη στο τέλος χρησιμοποιήστε `"a"`.
- **Ξεχασμένο** `fclose`. Χάνεται ένας file descriptor (το όριο ανοιχτών αρχείων είναι πεπερασμένο) και, αν το πρόγραμμα καταρρεύσει, δεδομένα που περιμένουν στον buffer μπορεί να μη γραφτούν ποτέ. Σε κάθε `fopen` ένα `fclose`.
- `buffer[bytesRead] = '\0'` **χωρίς χώρο**. Αν διαβάσετε `sizeof(buffer)` bytes, το `'\0'` πέφτει έξω από τον πίνακα· διαβάστε το πολύ `sizeof(buffer) - 1`.
- `fread` **αρχείου κειμένου σε** `int`. Παίρνετε `1819043176` αντί για αριθμό: η `fread` δεν μετατρέπει ψηφία. Για κείμενο χρησιμοποιήστε `fscanf`.
- `size_t` **με** `%d`. Η διαφάνεια τυπώνει το αποτέλεσμα της `fread` με `%d`· ο `gcc -Wall` προειδοποιεί (`expects argument of type 'int'`). Το σωστό είναι `%zu`.
- **Μηνύματα λάθους στο** `stdout`. Ανακατεύονται με τα αποτελέσματα όταν η έξοδος ανακατευθύνεται σε αρχείο. Γράψτε τα με `fprintf(stderr, ...)`.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K18.5** Συνάρτηση σύγκρισης για αύξουσα ταξινόμηση (12% σωστές): Το 43% επέλεξε `s2.grade - s1.grade`, αντιστρέφοντας τη σύμβαση: αρνητική τιμή σημαίνει ότι το `s1` μπαίνει πρώτο, οπότε η αφαίρεση `s2 - s1` δίνει φθίνουσα σειρά. Οι επιλογές με `==` και `^` (20% η καθεμία) δεν δίνουν καν πρόσημο που να λέει ποιο στοιχείο προηγείται.
- **K18.3** Ο file descriptor του stderr (51% σωστές): Το 29% επέλεξε `1`, που είναι ο αριθμός του `stdout`· τα τρία πρότυπα ρεύματα έχουν τους πρώτους αριθμούς με τη σειρά `stdin`, `stdout`, `stderr`.

Ερωτήσεις κατανόησης

- **E18.1** Ποιοι από τους πέντε αλγορίθμους ταξινόμησης χρειάζονται χώρο $O(1)$, και γιατί η merge sort χρειάζεται $O(n)$;[143]
- **E18.2** Πότε η quicksort της διάλεξης κάνει $O(n^2)$ βήματα;[144]
- **E18.3** Ποιο πλάτος πεδίου δίνετε στη `scanf` για να διαβάσετε λέξη σε `char name[20]`;[145]
- **E18.4** Τι επιστρέφει η `fopen` όταν αποτύχει, και τι γίνεται στο περιεχόμενο ενός υπάρχοντος αρχείου που ανοίγει με `"w"`;[146]
- **E18.5** Ένα αρχείο έχει 10 bytes. Τι επιστρέφει η `fread(buf, sizeof(int), 100, fp)`;[147]
- **E18.6** Ποιοι είναι οι file descriptors των `stdin`, `stdout`, `stderr`, και τι κάνει το `2> error.txt`;[148]

Kahoot από το αμφιθέατρο (K18.1–K18.5)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K18.1** stdin και αρχεία: 84% σωστές απαντήσεις
- **K18.2** Ανάγνωση int σε little endian: 62% σωστές απαντήσεις
- **K18.3** Ο file descriptor του stderr: 51% σωστές απαντήσεις
- **K18.4** Αποτυχία της fopen: 50% σωστές απαντήσεις
- **K18.5** Συνάρτηση σύγκρισης για αύξουσα ταξινόμηση: 12% σωστές απαντήσεις

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A18.1–A18.6)

- **A18.1** Μέγεθος δισδιάστατου πίνακα και μιας γραμμής του: Διάλεξη 18, διαφάνεια 2 · ★☆☆ · short-answer · `slides-lec18-2d-sizeof`
- **A18.2** Γιατί μπορεί να αποτύχει η fopen;: Διάλεξη 18, διαφάνειες 43–44 · ★☆☆ · short-answer · `slides-lec18-fopen-fail`
- **A18.3** Υποτείνουσα με scanf: Διάλεξη 18, διαφάνειες 29–30 · ★☆☆ · trace · `slides-lec18-hypotenuse`
- **A18.4** Ανακατεύθυνση της stderr: Διάλεξη 18, διαφάνεια 56 · ★☆☆ · tooling · `slides-lec18-stderr-redirect`
- **A18.5** Ακέραιοι από αρχείο κειμένου με fread: Διάλεξη 18, διαφάνειες 50–51 · ★★☆ · trace · `slides-lec18-fread-int`

[143] Οι selection, insertion και bubblesort· η `merge` αντιγράφει τα δύο μισά σε βοηθητικούς πίνακες.

[144] Όταν το pivot είναι κάθε φορά το μικρότερο ή το μεγαλύτερο στοιχείο του τμήματος, οπότε το ένα κομμάτι μικραίνει μόνο κατά ένα.

[145] `%19s`: 19 χαρακτήρες και το `'\0'`.

[146] `NULL`· το περιεχόμενο σβήνεται (το αρχείο γίνεται μήκους 0).

[147] 2: δύο ολόκληρα `int` των 4 bytes· τα 2 bytes που περισσεύουν δεν μετράνε.

[148] 0, 1 και 2· το `2> error.txt` ανακατευθύνει την έξοδο σφάλματος στο αρχείο.

- **A18.6** Μια λέξη σε char[7] με scanf: Διάλεξη 18, διαφάνειες 31–35 · ★★☆ · debug · `slides-lec18-scanf-string`

Εργαστήριο (A18.7–A18.10)

- **A18.7** Μέτρηση στατιστικών αρχείων: Εργαστήριο 10, Άσκηση 4 · ★☆☆ · programming · `lab-lab10-count`
- **A18.8** Σύγκριση αρχείων: Εργαστήριο 10, Άσκηση 3 · ★☆☆ · programming · `lab-lab10-filediff`
- **A18.9** Δυαδικά αρχεία: Εργαστήριο 10, Άσκηση 2 · ★★☆ · programming · `lab-lab10-bgrades`
- **A18.10** Αρχεία κειμένου: Εργαστήριο 10, Άσκηση 1 · ★★☆ · programming · `lab-lab10-more`

Εργασίες (A18.11)

- **A18.11** Προβλέποντας το Μέλλον (future): Εργασία 2 (2024-25), Άσκηση 1 · ★★☆ · programming · `hw-2024-hw2-future`

Θέματα εξετάσεων (A18.12–A18.23)

- **A18.12** Κρυφό Μήνυμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #7 (Star Wars Themed), Θέμα 2 · ★☆☆ · programming · `exam-2023-fall-ex7-q2`
- **A18.13** Κόψιμο Αρχείων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #7 (Star Wars Themed), Θέμα 4 · ★☆☆ · programming · `exam-2023-fall-ex7-q4`
- **A18.14** Έλεγχος Εκτελέσιμου: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #9, Θέμα 1 · ★☆☆ · programming · `exam-2023-fall-ex9-q1`
- **A18.15** Κρυμμένο Μήνυμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #10, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex10-q3`
- **A18.16** Μετρήσεις Θερμοκρασίας: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #14, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex14-q3`
- **A18.17** Αλλαγή Μεγέθους: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #4 (Rick Astley Themed), Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex4-q4`
- **A18.18** Επιλογή: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #5 (HP Themed), Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex5-q3`
- **A18.19** Ταξινόμηση Αρχείων Καταγραφής: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #6 (Crypto Themed), Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex6-q4`
- **A18.20** Καλύτερο Ταίριασμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #0 (Valentine's Themed), Θέμα 3 · ★★★ · programming · `exam-2023-fall-ex0-q3`
- **A18.21** Ταξινομώντας τα Άλματα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #15, Θέμα 3 · ★★★ · programming · `exam-2023-fall-ex15-q3`
- **A18.22** Μίνι Βάση Δεδομένων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #2 (Pokémon Themed), Θέμα 3 · ★★★ · programming · `exam-2023-fall-ex2-q3`

- **A18.23** Ταξινόμηση Πακέτων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #8, Θέμα 4 · ★★★ · programming · `exam-2023-fall-ex8-q4`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A12.16** Περιστροφή Πίνακα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #1 (Coreutils Themed), Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex1-q3`
- **A12.17** Κινήσεις σε Πλέγμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #11, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex11-q3`
- **A12.18** Πολλαπλασιασμός Πινάκων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #8, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex8-q3`
- **A12.19** Πολύτιμοι Πίνακες: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #9, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex9-q3`
- **A12.12** FauxtoShop: περιστροφή εικόνας BMP: Εργασία 2 (2023-24), Άσκηση 1 · ★★★ · programming · `hw-2023-hw2-fauxtoshop`
- **A14.16** Κρεμάλα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #12, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex12-q3`
- **A14.17** Δυνατότητες: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #15, Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex15-q4`
- **A14.18** Εύρεση Λέξεων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #3 (Frozen Themed), Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex3-q3`
- **A14.10** Το Δικό σου Chatbot (jason): Εργασία 2 (2024-25), Άσκηση 3 · ★★★ · programming · `hw-2024-hw2-jason`
- **A16.25** Χτίζοντας έναν Χιονάνθρωπο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #3 (Frozen Themed), Θέμα 4 · ★★★ · programming · `exam-2023-fall-ex3-q4`
- **A16.26** Μετρώντας τα Αστέρια - stars: Εξέταση Ιανουαρίου 2025, Θέμα 6 · ★★★ · programming · `exam-2025-jan-q6`
- **A17.4** Αναζήτηση χρήστη στο Instagram: Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 19 · ★☆☆ · short-answer · `slides-lec17-instagram`
- **A17.6** Η συνάρτηση swap: Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 21 · ★☆☆ · programming · `slides-lec17-swap`
- **A19.11** Πρωτάθλημα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #13, Θέμα 3 · ★★★ · programming · `exam-2023-fall-ex13-q3`
- **A19.12** World Cup 2026: Εξέταση Ιουνίου 2026, Θέμα 4 · ★★★ · programming · `exam-2026-jun-q4`
- **A23.6** Η Newton-Raphson Ξαναχτυπά! (Bonus): Εργασία 3 (2023-24), Άσκηση 2 (Bonus) και 2.1 (Bonus) · ★★★ · programming · `hw-2023-hw3-fractal`
- **A25.15** Το Καλό το Μονοπάτι - path: Εξέταση Σεπτεμβρίου 2025, Θέμα 5 · ★★★ · programming · `exam-2025-sep-q5`
- **A25.2** DNA Matching: Εργασία 2 (2023-24), Άσκηση 2 · ★★★ · programming · `hw-2023-hw2-dna`
- **A25.3** Το Καλύτερο GPS (jabbamaps): Εργασία 2 (2024-25), Άσκηση 2 · ★★★ · programming · `hw-2024-hw2-jabbamaps`

Κεφάλαιο 19: Δομές

> **Στόχοι**: μετά από αυτό το κεφάλαιο θα μπορείτε να ορίζετε δικούς σας τύπους με `struct`, να δηλώνετε, να αρχικοποιείτε και να αντιγράφετε μεταβλητές τύπου δομής, να προσπελαύνετε τα πεδία τους με `.` και, μέσω δείκτη, με `->`· να εξηγείτε πώς μια δομή κάθεται στη μνήμη και γιατί το `sizeof` της μπορεί να είναι μεγαλύτερο από το άθροισμα των πεδίων της (padding)· και να δίνετε συνώνυμα τύπων με `typedef`.
>
> **Προαπαιτούμενα**: Κεφάλαιο 12, Κεφάλαιο 13
>
> **Χρόνος μελέτης**: ~2 ώρες

Σύνοψη

Μέχρι τώρα τα δεδομένα μας ήταν `int`, `double`, `char`, δείκτες και πίνακες. Τα πραγματικά δεδομένα όμως είναι πιο δομημένα: ένας φοιτητής έχει όνομα, επώνυμο, έτος και βαθμό, διαφορετικών τύπων, που ανήκουν μαζί. Η **δομή (struct)** της C ομαδοποιεί τέτοια πεδία κάτω από ένα όνομα και έτσι ορίζουμε δικούς μας τύπους. Η διάλεξη δείχνει πώς δηλώνουμε, αρχικοποιούμε, αντιγράφουμε και συγκρίνουμε δομές, πώς απλώνονται στη μνήμη (με το padding που προσθέτει ο μεταγλωττιστής), πώς το `typedef` δίνει σύντομα ονόματα, και πώς δουλεύουμε με ένθετες δομές και δείκτες σε δομές. Οι δομές είναι το υλικό από το οποίο θα φτιάξουμε λίστες και δέντρα στα επόμενα κεφάλαια.

Θεωρία

§19.1 Γιατί χρειαζόμαστε δομές

Έστω ένα πρόγραμμα που διαχειρίζεται μια λίστα φοιτητών. Για κάθε φοιτητή χρειάζεστε `char first_name[128]`, `char last_name[128]`, `unsigned int year` και `double grade`. Για 100 φοιτητές, χωρίς δομές, θα έπρεπε να κρατάτε τέσσερις παράλληλους πίνακες (`first_names[100][128]`, `last_names[100][128]`, `years[100]`, `grades[100]`) και να φροντίζετε ότι η θέση `i` σε όλους αφορά τον ίδιο άνθρωπο. Αυτό είναι εύθραυστο: μια ταξινόμηση που ξεχνά να μετακινήσει έναν από τους πίνακες χαλάει τα δεδομένα.

Μια **δομή (struct)**, αλλιώς **εγγραφή (record)**, είναι μια συλλογή **πεδίων (fields)** που ομαδοποιούν την πληροφορία μιας λογικής οντότητας. Με δομή, ο φοιτητής γίνεται μία μεταβλητή και οι 100 φοιτητές ένας πίνακας.

§19.2 Δήλωση τύπου δομής

Η δήλωση δομής δημιουργεί έναν **τύπο ορισμένο από τον χρήστη (user-defined type)**:

```
struct student {
  char first_name[128];
  char last_name[128];
  unsigned int year;
  double grade;
};
```

Η λέξη-κλειδί `struct` λέει ότι ορίζουμε δομή. Το `student` είναι το **όνομα** ή **ετικέτα (tag)** της δομής, και ο τύπος λέγεται `struct student` (και οι δύο λέξεις). Στα άγκιστρα δηλώνονται τα πεδία όπως μεταβλητές· πεδίο μπορεί να είναι βασικός τύπος, πίνακας, δείκτης ή άλλη δομή. Η δήλωση κλείνει με `};`.

Η δήλωση του τύπου **δεν δεσμεύει μνήμη**· περιγράφει μόνο το «καλούπι». Μνήμη δεσμεύεται όταν δηλώσουμε μεταβλητές αυτού του τύπου.

§19.3 Μεταβλητές τύπου δομής

Μεταβλητή δηλώνεται με `struct όνομα_δομής όνομα_μεταβλητής;`, και ο τύπος συνδυάζεται με δείκτες και πίνακες όπως κάθε άλλος:

```
struct student st1, st2;          // δύο μεταβλητές
struct student *student_ptr;      // δείκτης σε struct student
struct student student_array[100];// πίνακας με 100 δομές
```

Μπορείτε επίσης να δηλώσετε μεταβλητές μαζί με τον τύπο, γράφοντάς τες πριν από το τελικό `;`: `struct student { ... } st1;`.

§19.4 Προσπέλαση πεδίων με τον τελεστή `.`

Στο πεδίο μιας μεταβλητής δομής αναφερόμαστε με `μεταβλητή.πεδίο`. Το `st1.year` είναι το ακέραιο πεδίο `year` της δομής `st1`. Ένα πεδίο συμπεριφέρεται ακριβώς όπως μια μεταβλητή του τύπου του: ανατίθεται (`st1.year = 1;`), διαβάζεται, περνά σε συναρτήσεις, και αν είναι πίνακας χαρακτήρων δουλεύει με τις συναρτήσεις συμβολοσειρών (`strncpy(st1.first_name, ...)`). Ένας πίνακας όμως δεν ανατίθεται με `=`, οπότε το `st1.first_name = "Thanos";` δεν μεταγλωττίζεται.

§19.5 Αρχικοποίηση δομής

Όπως οι πίνακες, μια δομή αρχικοποιείται με λίστα τιμών σε άγκιστρα:

```
struct student st1 = { "Thanos", "Barbounis", 1, 7.54 };
```

Οι τιμές αντιστοιχούν στα πεδία **με τη σειρά της δήλωσης της δομής**: πρώτη τιμή στο `first_name`, δεύτερη στο `last_name` κ.ο.κ. Αν παραλείψετε τιμές στο τέλος, τα πεδία που μένουν **αρχικοποιούνται στο 0**: με `{ "Thanos", "Barbounis", 1 }` το `grade` είναι `0.0`. Αντίθετα, μια τοπική δομή χωρίς καθόλου αρχικοποίηση περιέχει ό,τι υπήρχε στη μνήμη (σκουπίδια), όπως κάθε τοπική μεταβλητή.

Υπάρχει και αρχικοποίηση με ονόματα πεδίων, ανεξάρτητη από τη σειρά: `{ .year = 2, .grade = 8.5 }` (σύνδεσμος «struct initialization» στο «Διάβασμα»).

§19.6 Η δομή στη μνήμη

Όπως κάθε μεταβλητή, μια μεταβλητή δομής πιάνει bytes στη μνήμη. Τα πεδία αποθηκεύονται **με τη σειρά της δήλωσης**, το ένα μετά το άλλο. Η διαφάνεια δείχνει το `st1` με αρχή στη διεύθυνση 31000:

Πεδίο	Bytes	Μέγεθος	Τιμή
`first_name`	31000–31127	128	`"Thanos"`
`last_name`	31128–31255	128	`"Barbounis"`
`year`	31256–31259	4	`1`
`grade`	31260–31267	8	`7.54`

Σύνολο $128 + 128 + 4 + 8 = 268$ bytes, και πράγματι με `gcc -m32` (μεταγλώττιση για 32 bit) το `sizeof(struct student)` είναι 268. Σε ένα σύγχρονο σύστημα 64 bit όμως τυπώνει **272**: ο μεταγλωττιστής αφήνει 4 κενά bytes μετά το `year` ώστε το `grade` να ξεκινά σε διεύθυνση πολλαπλάσιο του 8. Αυτό είναι το padding της επόμενης ενότητας.

§19.7 Μέγεθος δομής και padding

Ο επεξεργαστής διαβάζει γρηγορότερα ένα `int` ή ένα `double` όταν η διεύθυνσή του είναι πολλαπλάσιο του μεγέθους του. Γι' αυτό ο μεταγλωττιστής μπορεί να προσθέσει **padding** («κενά» bytes που δεν χρησιμοποιούνται) ανάμεσα στα πεδία ή στο τέλος της δομής, ώστε οι διευθύνσεις των πεδίων να είναι πολλαπλάσια του 4 (ή του 8, `sizeof(void *)`). Αυτό λέγεται **ευθυγράμμιση μνήμης (memory alignment)** και γίνεται για λόγους απόδοσης.

Στη δομή `{ char red; char green; char blue; int alpha; }` τα πεδία είναι $3 + 4 = 7$ bytes, αλλά το `sizeof` δίνει 8: ένα byte padding μπαίνει μετά το `blue` ώστε το `alpha` να ξεκινά σε πολλαπλάσιο του 4.

Offset	0	1	2	3	4–7
Πεδίο	`red`	`green`	`blue`	padding	`alpha`

Η **σειρά των πεδίων** μετράει. Αν εναλλάξετε `char` και `int` (`char red; int alpha; char green; int beta; char blue; int gamma;`), κάθε `char` ακολουθείται από 3 bytes padding και το μέγεθος γίνεται 24, ενώ χρησιμοποιούνται μόνο $3 \cdot 1 + 3 \cdot 4 = 15$:

Offset	0	1–3	4–7	8	9–11	12–15	16	17–19	20–23
Πεδίο	`red`	pad	`alpha`	`green`	pad	`beta`	`blue`	pad	`gamma`

Το συμπέρασμα της διάλεξης: **το μέγεθος μιας δομής είναι ό,τι επιστρέφει το** `sizeof`. Μην το υπολογίζετε με το χέρι αθροίζοντας πεδία· γράφετε πάντα `sizeof(struct student)` (π.χ. στη `malloc`). Τυπώνεται με `%zu`.

§19.8 Ανάθεση δομών

Ο τελεστής `=` αντιγράφει **όλα** τα περιεχόμενα μιας δομής σε μια άλλη, πεδίο προς πεδίο, ακόμα και πίνακες που είναι πεδία της (που αλλιώς δεν ανατίθενται). Με `struct point pt1 = { 3, 4 }, pt2;`, μετά το `pt2 = pt1;` ισχύει `pt2.x == 3` και `pt2.y == 4`. Οι δύο μεταβλητές πρέπει να έχουν **ακριβώς τον ίδιο τύπο**. Δύο δομές με διαφορετικό όνομα είναι διαφορετικοί τύποι, ακόμα κι αν έχουν ίδια πεδία: η ανάθεση ενός `struct point1` σε `struct point2` δίνει `error: incompatible types when assigning to type 'struct point2' from type 'struct point1'`.

§19.9 Σύγκριση δομών

Δύο δομές **δεν** συγκρίνονται με `==` ή `!=`. Το `if (pt1 == pt2)` δίνει `error: invalid operands to binary ==`. Πρέπει να συγκρίνετε τα πεδία ένα-ένα: `pt1.x == pt2.x && pt1.y == pt2.y` (και για πεδία-συμβολοσειρές με `strcmp`).

§19.10 Το `typedef`

Το προσδιοριστικό `typedef` (type definition) ορίζει ένα **συνώνυμο** για έναν υπάρχοντα τύπο: `typedef υπάρχων_τύπος νέος_τύπος;`. Μετά από `typedef unsigned int myuint;` οι δηλώσεις `unsigned int x;` και `myuint x;` είναι ισοδύναμες. Δεν δημιουργείται νέος τύπος, μόνο νέο όνομα.

Η σύνταξη μοιάζει με δήλωση μεταβλητής, με το νέο όνομα στη θέση της μεταβλητής. Έτσι ονομάζουμε και τύπους πίνακα: μετά από `typedef int page[1024];` η δήλωση `page mypage;` φτιάχνει πίνακα 1024 ακεραίων, και το `sizeof(mypage)` είναι 4096.

Με δομές το `typedef` γλιτώνει τη λέξη `struct` σε κάθε δήλωση: μετά από `typedef struct student { ... } Student;` γράφετε `Student st1 = {...};`, αφού το `Student` και το `struct student` είναι ο ίδιος τύπος. Το συνώνυμο μπορεί να έχει και το ίδιο όνομα με την ετικέτα (`} student;`), γιατί οι ετικέτες των δομών ζουν σε ξεχωριστό χώρο ονομάτων. Μπορείτε και να

παραλείψετε την ετικέτα (`typedef struct { ... } student;`): τότε η δομή είναι **ανώνυμη** και αναφέρεστε σε αυτήν μόνο με το συνώνυμο.

§19.11 Ένθετες δομές

Μια δομή μπορεί να έχει ως πεδία μία ή περισσότερες άλλες δομές, που λέγονται **εμφωλευμένες ή ένθετες δομές (nested structs)**. Για παράδειγμα ένα προϊόν έχει δύο ημερομηνίες:

```
struct date { int day; int month; int year; };
struct product {
  char *name;
  double price;
  struct date created;
  struct date updated;
};
```

Η εσωτερική δομή πρέπει να έχει δηλωθεί πριν χρησιμοποιηθεί. Στην αρχικοποίηση κάθε ένθετη δομή παίρνει τα δικά της άγκιστρα: `{"eclass", 3.14, {1, 1, 2021}, {11, 12, 2022}}`. Για να φτάσετε σε ένα βαθύ πεδίο χρησιμοποιείτε το `.` όσες φορές χρειάζεται: `prod.updated.year = 2023;`. Ο τελεστής `.` είναι αριστερά προσεταιριστικός, άρα αυτό σημαίνει `(prod.updated).year`.

§19.12 Δείκτες σε δομές και ο τελεστής `->`

Οι δείκτες συνδυάζονται με δομές όπως με κάθε τύπο. Ένας δείκτης σε δομή παίρνει τη διεύθυνση μιας υπάρχουσας δομής (`Date *d2 = &d1;`) ή ένα μπλοκ από τη `malloc` (`Date *d3 = malloc(sizeof(Date));`). Με `*d2` φτάνετε στη δομή, άρα στο πεδίο της με `(*d2).day`. Οι παρενθέσεις είναι **απαραίτητες**: το `.` έχει μεγαλύτερη προτεραιότητα από το `*`, οπότε το `*d2.day` σημαίνει `*(d2.day)` και δεν μεταγλωττίζεται.

Επειδή αυτό γράφεται συνέχεια, η C δίνει τον **τελεστή βέλους (arrow operator)** `->`: το `ptr->field` είναι **ισοδύναμο** με το `(*ptr).field`. Γράφετε λιγότερο, και ο αναγνώστης βλέπει αμέσως ότι η μεταβλητή είναι δείκτης. Κανόνας: `.` όταν έχετε τη δομή, `->` όταν έχετε δείκτη σε αυτήν.

Η ανάθεση `*d3 = *d2;` αντιγράφει ολόκληρη τη δομή όπου δείχνει το `d2` στη μνήμη όπου δείχνει το `d3`, ενώ το `d3 = d2;` θα αντέγραφε μόνο τη διεύθυνση.

Παραδείγματα

§19.13 Ανάθεση και χρήση πεδίων

Το πρόγραμμα της διάλεξης (Θεωρία: «Προσπέλαση πεδίων», «Δήλωση τύπου δομής») γεμίζει ένα `struct student` πεδίο-πεδίο. Τα ονόματα αντιγράφονται με `strncpy`, που δεν βάζει πάντα

`'\0'`, γι' αυτό η επόμενη γραμμή γράφει τον τερματικό χαρακτήρα στο τελευταίο byte του πίνακα. Το `sizeof(st1.first_name)` είναι 128, το μέγεθος του πεδίου.

```
#include <stdio.h>
#include <string.h>

struct student {
  char first_name[128]; char last_name[128];
  unsigned int year; double grade;
};

int main() {
  struct student st1;
  st1.year = 1;
  st1.grade = 7.54;
  strncpy(st1.first_name, "Thanos", sizeof(st1.first_name) - 1);
  st1.first_name[sizeof(st1.first_name) - 1] = '\0';
  strncpy(st1.last_name, "Barbounis", sizeof(st1.last_name) - 1);
  st1.last_name[sizeof(st1.last_name) - 1] = '\0';
  printf("%s %s: %f [%u year]\n", st1.first_name, st1.last_name,
         st1.grade, st1.year);
  return 0;
}
```

```
$ ./struct
Thanos Barbounis: 7.540000 [1 year]
```

Με αρχικοποίηση `struct student st1 = { "Thanos", "Barbounis", 1, 7.54 };` η έξοδος είναι ίδια. Αν παραλείψετε το `7.54`, το `grade` γίνεται 0:

```
$ ./struct
Thanos Barbounis: 0.000000 [1 year]
```

§19.14 Μέγεθος δομής και padding

Τρία προγράμματα της διάλεξης μεταγλωττισμένα με `-m32` (Θεωρία: «Η δομή στη μνήμη», «Μέγεθος δομής και padding»):

```
#include <stdio.h>

int main() {
  struct pixel_tag {
    char red; char green; char blue; int alpha;
  } pixel = {0xFF, 0xFF, 0xFF, 42};
  printf("%zu\n", sizeof(pixel));
  return 0;
```

```
}
```

Το `struct3.c` είναι το ίδιο με τα πεδία ανακατεμένα (`char red; int alpha; char green; int beta; char blue; int gamma;`), και το `struct.c` τυπώνει `sizeof(struct student)`:

```
$ gcc -m32 -o struct2 struct2.c
$ ./struct2
8
$ gcc -m32 -o struct3 struct3.c
$ ./struct3
24
$ gcc -m32 -o struct struct.c
$ ./struct
268
```

Χωρίς `-m32`, σε 64 bit, τα δύο πρώτα δίνουν πάλι 8 και 24, αλλά το `struct student` δίνει 272 λόγω του padding πριν από το `double`.

§19.15 Αντιγραφή δομής με ανάθεση

Θεωρία: «Ανάθεση δομών». Το `pt2` τυπώνεται πρώτα χωρίς αρχικοποίηση:

```c
#include <stdio.h>

struct point { int x; int y; };

int main() {
  struct point pt1 = { 3, 4 };
  struct point pt2;
  printf("%d %d\n", pt2.x, pt2.y);
  pt2 = pt1;
  printf("%d %d\n", pt2.x, pt2.y);
  return 0;
}
```

```
$ ./copy
-29387249 0
3 4
```

Η πρώτη γραμμή είναι ό,τι υπήρχε στη μνήμη και αλλάζει από εκτέλεση σε εκτέλεση (ο `gcc -Wall` προειδοποιεί ότι το `pt2` χρησιμοποιείται χωρίς αρχικοποίηση). Η δεύτερη δείχνει ότι η ανάθεση αντέγραψε όλα τα πεδία.

§19.16 Ένθετες δομές με και χωρίς `typedef`

Θεωρία: «Ένθετες δομές», «Το `typedef`». Με τους τύπους `struct date` και `struct product` της Θεωρίας, το `sizeof(prod)` είναι 40 (8 για τον δείκτη, 8 για το `double`, 2 × 12 για τις ημερομηνίες). Η ίδια λογική γραμμένη με `typedef` και ανώνυμες δομές:

```c
#include <stdio.h>

typedef struct { int day; int month; int year; } Date;

typedef struct {
  char *name;
  double price;
  Date created, updated;
} Product;

int main() {
  Product prod = {"eclass", 3.14, {1, 1, 2021}, {11, 12, 2022}};
  prod.updated.year = 2023;
  printf("%s [eu: %.2f] [created: %d/%d/%d] [updated: %d/%d/%d]\n",
         prod.name, prod.price,
         prod.created.day, prod.created.month, prod.created.year,
         prod.updated.day, prod.updated.month, prod.updated.year);
  return 0;
}
```

```
$ ./nested3
eclass [eu: 3.14] [created: 1/1/2021] [updated: 11/12/2023]
```

Η εκδοχή με `struct date` / `struct product` (χωρίς `typedef`) τυπώνει ακριβώς το ίδιο.

§19.17 Δείκτες σε δομές: η διαφορά ημερομηνιών

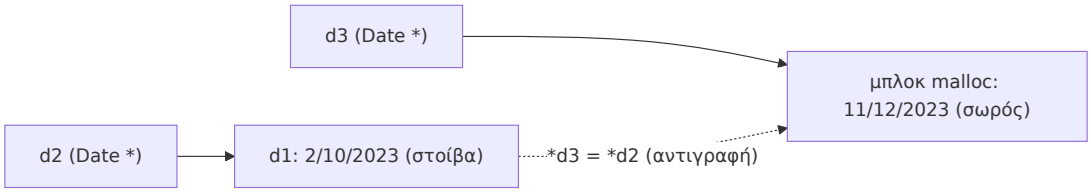
Θεωρία: «Δείκτες σε δομές και ο τελεστής `->`». Γραμμένο με `->`· η εκδοχή της διάλεξης με `(*d2).day`, `(*d3).month` κ.λπ. δίνει την ίδια έξοδο.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct { int day; int month; int year; } Date;

int main() {
  Date d1 = {1, 10, 2023};
  Date *d2 = &d1;
  d2->day = 2;
  Date *d3 = malloc(sizeof(Date));
```

```
  *d3 = *d2;
  d3->month = 12; d3->day = 11;
  printf("Diff: %d/%d/%d\n", d3->day - d1.day, d3->month - d1.month,
         d3->year - d1.year);
  return 0;
}
```

```
$ ./dateptr
Diff: 9/2/0
```

Το `d2` δείχνει στο ίδιο το `d1`, άρα το `d2->day = 2` αλλάζει το `d1` σε 2/10/2023. Το `*d3 = *d2` αντιγράφει αυτή την ημερομηνία στο μπλοκ της `malloc`, που μετά γίνεται 11/12/2023. Οι διαφορές είναι $11-2=9$, $12-10=2$, $2023-2023=0$. (Σε κανονικό πρόγραμμα θα ελέγχατε ότι η `malloc` δεν επέστρεψε `NULL` και θα καλούσατε `free(d3)`· δείτε Κεφάλαιο 13.)



Σχήμα: το `d2` δείχνει στο `d1`, το `d3` σε νέα μνήμη· η ανάθεση `*d3 = *d2` αντιγράφει τη δομή, όχι τη διεύθυνση.

§19.18 Για το εργαστήριο

Το Εργαστήριο 9 εφαρμόζει αυτό το κεφάλαιο στις δύο πρώτες ασκήσεις: το `point.c` ορίζει `struct point` με συντεταγμένες `double` και μια συνάρτηση που επιστρέφει δομή (το μέσο δύο σημείων), και το `person.c` δεσμεύει δομή με `malloc` και την προσπελαύνει με `->`. Το πέρασμα και η επιστροφή δομών από συναρτήσεις αναλύονται στο Κεφάλαιο 20.

Κύρια σημεία

1. Μια δομή (`struct`) ομαδοποιεί πεδία, πιθανώς διαφορετικών τύπων, που περιγράφουν μια λογική οντότητα, και ορίζει έναν νέο τύπο, π.χ. `struct student`.
2. Η δήλωση του τύπου δεν δεσμεύει μνήμη· μνήμη δεσμεύουν οι μεταβλητές, οι πίνακες και οι δομές από `malloc` αυτού του τύπου.
3. Στα πεδία φτάνουμε με `μεταβλητή.πεδίο`, και κάθε πεδίο χρησιμοποιείται όπως μια μεταβλητή του τύπου του.
4. Η αρχικοποίηση `{ ... }` αντιστοιχεί τιμές στα πεδία με τη σειρά της δήλωσης· όσα πεδία παραλείπονται γίνονται 0.
5. Τα πεδία αποθηκεύονται στη μνήμη με τη σειρά της δήλωσης, αλλά ο μεταγλωττιστής μπορεί να προσθέσει padding για ευθυγράμμιση, οπότε η σειρά των πεδίων επηρεάζει το

μέγεθος.

6. Το μέγεθος μιας δομής είναι ό,τι επιστρέφει το `sizeof(struct όνομα)`, όχι το άθροισμα των πεδίων της.
7. Η ανάθεση `=` αντιγράφει όλα τα πεδία, μόνο ανάμεσα σε δομές του ίδιου τύπου· σύγκριση με `==` δεν υπάρχει, συγκρίνουμε τα πεδία ένα-ένα.
8. Το `typedef` δίνει συνώνυμο σε έναν τύπο· με δομές γλιτώνει το `struct`, και η ετικέτα μπορεί να παραλειφθεί (ανώνυμη δομή).
9. Μια δομή μπορεί να περιέχει άλλες δομές, και φτάνουμε σε βαθιά πεδία με διαδοχικά `.` (`prod.updated.year`).
10. Για δείκτη σε δομή, το `ptr->field` είναι ισοδύναμο με το `(*ptr).field`.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
δομή / εγγραφή	struct / record	Συλλογή πεδίων που περιγράφουν μια οντότητα· νέος τύπος.
πεδίο / μέλος	field / member	Μία από τις μεταβλητές μέσα σε μια δομή.
ετικέτα δομής	struct tag	Το όνομα μετά το `struct`, π.χ. `student`.
τύπος ορισμένος από τον χρήστη	user-defined type	Τύπος που ορίζει το πρόγραμμα, όπως μια δομή.
αρχικοποίηση δομής	struct initialization	Τιμές σε `{ }` με τη σειρά των πεδίων.
γέμισμα	padding	Αχρησιμοποίητα bytes που προσθέτει ο μεταγλωττιστής.
ευθυγράμμιση μνήμης	memory alignment	Διευθύνσεις πεδίων πολλαπλάσιες του 4 ή του 8.
συνώνυμο τύπου	`typedef`	Νέο όνομα για υπάρχοντα τύπο.
ανώνυμη δομή	anonymous struct	Δομή χωρίς ετικέτα, συνήθως με `typedef`.
ένθετη / εμφωλευμένη δομή	nested struct	Δομή που είναι πεδίο άλλης δομής.
τελεστής βέλους	arrow operator (`->`)	`ptr->f` ισοδυναμεί με `(*ptr).f`.

Διάβασμα

- **Διαφάνειες**: Διάλεξη 19, σελ. 1–47: δήλωση και προσπέλαση 6–13· αρχικοποίηση 14–16· μνήμη, `sizeof` και padding 17–28· ανάθεση και σύγκριση 29–32· `typedef` 33–37· ένθετες δομές 38–41· δείκτες σε δομές και `->` 42–45.
- **Σημειώσεις**: η διάλεξη καλύπτει τις σελ. 109–119 και 126 των σημειώσεων: Κεφάλαιο 7: Απαριθμήσεις, δομές και ενώσεις, ενότητες «Δομές» (K04, σελ. 109–116), «Εύρεση

τριγώνων μεγίστου και ελαχίστου εμβαδού» (117–119) και «Δημιουργία νέων ονομάτων τύπων» (126). Η ενότητα «Απαριθμήσεις» (σελ. 108) δεν καλύπτεται σε αυτή τη διάλεξη.
- **Εργαστήριο**: Εργαστήριο 9: ασκήσεις `point.c`, `person.c`.
- **Βιβλίο**: K&R, §6.1–6.7 (παραπομπή των σημειώσεων).
- **Άλλα**: Struct (C programming language), Structures in C, struct initialization, Typedef, Structure member alignment, padding and data packing, Data structure alignment, Arrow operator in C· η δομή `FILE` της glibc (περίπου 216 bytes): δείτε τι περιέχει.

Συχνά λάθη

- **Ξεχασμένο `;` μετά το `}` της δομής.** Ο `gcc` αναφέρει σφάλμα στην *επόμενη* γραμμή: `expected ';', identifier or '(' before 'int'`. Κλείνετε με `};`.
- `student st1;` **χωρίς** `typedef`. `unknown type name 'student'; use 'struct' keyword to refer to the type`: ο τύπος είναι `struct student`.
- **Ανάθεση συμβολοσειράς σε πεδίο-πίνακα.** Το `st1.first_name = "Thanos";` δίνει `assignment to expression with array type`. Χρησιμοποιήστε `strncpy` (και βάλτε `'\0'`) ή αρχικοποίηση.
- **Σύγκριση δομών με `==`.** `invalid operands to binary ==`: συγκρίνετε πεδίο προς πεδίο.
- **Ανάθεση ανάμεσα σε δομές με ίδια πεδία αλλά διαφορετικό όνομα.** `incompatible types when assigning to type 'struct point2' from type 'struct point1'`: είναι διαφορετικοί τύποι.
- **Υπολογισμός μεγέθους με το χέρι.** Το `malloc(268)` για `struct student` είναι λίγο σε 64 bit (272). Γράψτε `malloc(sizeof(struct student))`.
- **`.` σε δείκτη (και `*ptr.field`).** Με `Date *d2`, τα `d2.day` και `*d2.day` δίνουν `'d2' is a pointer; did you mean to use '->'?`. Γράψτε `d2->day`. Το αντίστροφο, `d1->day` με `Date d1`, δίνει `invalid type argument of '->'`.
- **Ανάγνωση πεδίων χωρίς αρχικοποίηση.** Μια τοπική δομή χωρίς `= { ... }` έχει σκουπίδια (`-29387249 0`)· αρκεί `= {0}` για να μηδενιστεί.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K19.9** Μερική αρχικοποίηση δομής (42% σωστές): Το 26% επέλεξε `42`, πιστεύοντας ότι η μία τιμή αντιγράφεται σε όλα τα πεδία, και άλλο 26% «ό,τι έτυχε να έχει η μνήμη», ξεχνώντας ότι με αρχικοποιητή τα πεδία που λείπουν μηδενίζονται.
- **K19.7** Ανάθεση δομών (49% σωστές): Το 28% επέλεξε `2`, σαν η ανάθεση `bar = cafe` να μην άλλαζε τα πεδία της `bar`· στην πραγματικότητα η ανάθεση δομών αντιγράφει όλα τα πεδία.
- **K19.6** Μέγεθος δομής και padding (52% σωστές): Το 24% επέλεξε `4 + 4 + 8 = 16`, την τιμή που δίνει συνήθως ένας 64-bit υπολογιστής· όμως το padding ανάμεσα στα πεδία και τα

μεγέθη των τύπων δεν ορίζονται από το πρότυπο.

- **Κ19.5** Σύγκριση δομών με == (57% σωστές): Το 24% επέλεξε `0`, υποθέτοντας ότι το `==` συγκρίνει τις δομές πεδίο προς πεδίο, όπως το `=` τις αντιγράφει.
- **Κ19.4** sizeof ενός typedef πίνακα (66% σωστές): Το 24% επέλεξε «Εξαρτάται», ενώ το `typedef` απλώς δίνει όνομα στον τύπο «πίνακας 1024 `double`», που έχει σταθερό μέγεθος.

Ερωτήσεις κατανόησης

- **Ε19.1** Πόση μνήμη δεσμεύει η δήλωση `struct student { ... };` χωρίς μεταβλητές;[149]
- **Ε19.2** Με `struct student st1 = { "Ada", "Lovelace" };`, ποια είναι η τιμή του `st1.grade`;[150]
- **Ε19.3** Γιατί το `sizeof` μιας δομής μπορεί να είναι μεγαλύτερο από το άθροισμα των μεγεθών των πεδίων της;[151]
- **Ε19.4** Πώς θα αναδιατάσσατε τα πεδία της `{ char red; int alpha; char green; int beta; char blue; int gamma; }` για να μικρύνει;[152]
- **Ε19.5** Τι κάνει το `pt2 = pt1;` για δύο `struct point` και γιατί δεν γίνεται `pt1 == pt2`;[153]
- **Ε19.6** Γράψτε με `->` την παράσταση `(*p).created.day`, όπου `p` είναι `struct product *`.[154]
- **Ε19.7** Ποια η διαφορά ανάμεσα σε `d3 = d2;` και `*d3 = *d2;` για δύο δείκτες `Date *`;[155]

Kahoot από το αμφιθέατρο (Κ19.1–Κ19.9)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **Κ19.1** Δικοί μας τύποι: 81% σωστές απαντήσεις
- **Κ19.2** Σε τι χρησιμεύει το typedef: 79% σωστές απαντήσεις
- **Κ19.3** Δομή χωρίς αρχικοποίηση: 74% σωστές απαντήσεις
- **Κ19.4** sizeof ενός typedef πίνακα: 66% σωστές απαντήσεις
- **Κ19.5** Σύγκριση δομών με ==: 57% σωστές απαντήσεις
- **Κ19.6** Μέγεθος δομής και padding: 52% σωστές απαντήσεις
- **Κ19.7** Ανάθεση δομών: 49% σωστές απαντήσεις
- **Κ19.8** Ο τελεστής ->: 48% σωστές απαντήσεις
- **Κ19.9** Μερική αρχικοποίηση δομής: 42% σωστές απαντήσεις

[149] Καμία. Η δήλωση ορίζει μόνο τον τύπο· μνήμη δεσμεύεται όταν δηλωθεί μεταβλητή (ή κληθεί `malloc`).

[150] `0.0`: τα πεδία που λείπουν από τη λίστα αρχικοποίησης γίνονται 0 (και το `year` επίσης 0).

[151] Λόγω padding: ο μεταγλωττιστής αφήνει κενά bytes ώστε κάθε πεδίο να ξεκινά σε διεύθυνση κατάλληλη για τον τύπο του (memory alignment), για λόγους απόδοσης.

[152] Βάζοντας μαζί τα `int` και μαζί τα `char`, π.χ. `int alpha, beta, gamma; char red, green, blue;`: 12 + 3 = 15 bytes, που με padding στο τέλος γίνονται 16 αντί για 24.

[153] Αντιγράφει όλα τα πεδία του `pt1` στο `pt2`. Η C δεν ορίζει `==` για δομές (`invalid operands to binary ==`)· συγκρίνουμε `pt1.x == pt2.x && pt1.y == pt2.y`.

[154] `p->created.day`: το `->` για τον δείκτη, το `.` για την ένθετη δομή.

[155] Το `d3 = d2;` αντιγράφει τη διεύθυνση (και οι δύο δείχνουν στην ίδια δομή, ενώ χάνεται το μπλοκ του `d3`)· το `*d3 = *d2;` αντιγράφει τα περιεχόμενα της δομής.

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A19.1–A19.8)

- **A19.1** Ανάθεση και χρήση πεδίων δομής: Διάλεξη 19, διαφάνειες 12–13 · ★☆☆ · trace · `slides-lec19-field-assignment`
- **A19.2** Πώς αναπαριστούμε 100 φοιτητές;: Διάλεξη 19, διαφάνειες 6–7 · ★☆☆ · short-answer · `slides-lec19-many-students`
- **A19.3** Ανάθεση με δομές: Διάλεξη 19, διαφάνειες 29–30 · ★☆☆ · trace · `slides-lec19-struct-assign`
- **A19.4** Σύγκριση με δομές: Διάλεξη 19, διαφάνεια 32 · ★☆☆ · short-answer · `slides-lec19-struct-compare`
- **A19.5** Δείκτες σε δομές: διαφορά ημερομηνιών: Διάλεξη 19, διαφάνειες 42–43 · ★★☆ · trace · `slides-lec19-date-pointers`
- **A19.6** Padding: η σειρά των πεδίων μετράει: Διάλεξη 19, διαφάνειες 25–27 · ★★☆ · trace · `slides-lec19-padding-interleaved`
- **A19.7** Padding: το μέγεθος ενός pixel: Διάλεξη 19, διαφάνειες 22–24 · ★★☆ · trace · `slides-lec19-padding-pixel`
- **A19.8** Το μέγεθος του struct student: Διάλεξη 19, διαφάνειες 20–21, 28 · ★★☆ · trace · `slides-lec19-sizeof-student`

Εργαστήριο (A19.9–A19.10)

- **A19.9** Δομές και συναρτήσεις: Εργαστήριο 9, Άσκηση 1 · ★☆☆ · programming · `lab-lab09-point`
- **A19.10** Δομές και δείκτες: Εργαστήριο 9, Άσκηση 2 · ★★☆ · programming · `lab-lab09-person`

Θέματα εξετάσεων (A19.11–A19.12)

- **A19.11** Πρωτάθλημα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #13, Θέμα 3 · ★★★ · programming · `exam-2023-fall-ex13-q3`
- **A19.12** World Cup 2026: Εξέταση Ιουνίου 2026, Θέμα 4 · ★★★ · programming · `exam-2026-jun-q4`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A17.11** Πλησιάζοντας στον Στόχο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #10, Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex10-q4`
- **A18.20** Καλύτερο Ταίριασμα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #0 (Valentine's Themed), Θέμα 3 · ★★★ · programming · `exam-2023-fall-ex0-q3`
- **A18.16** Μετρήσεις Θερμοκρασίας: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #14, Θέμα 3 · ★★☆ · programming · `exam-2023-fall-ex14-q3`

- **A18.21** Ταξινομώντας τα Άλματα: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #15, Θέμα 3 · ★★★ · programming · `exam-2023-fall-ex15-q3`
- **A18.22** Μίνι Βάση Δεδομένων: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #2 (Pokémon Themed), Θέμα 3 · ★★★ · programming · `exam-2023-fall-ex2-q3`
- **A21.7** Συνδεδεμένες λίστες: Εργαστήριο 9, Άσκηση 3 · ★★☆ · programming · `lab-lab09-grades`
- **A23.6** Η Newton-Raphson Ξαναχτυπά! (Bonus): Εργασία 3 (2023-24), Άσκηση 2 (Bonus) και 2.1 (Bonus) · ★★★ · programming · `hw-2023-hw3-fractal`

Κεφάλαιο 20: Προχωρημένες Δομές

> **Στόχοι**: μετά από αυτό το κεφάλαιο θα μπορείτε να δηλώνετε πεδία bit σε δομές και να προβλέπετε το μέγεθος και τις τιμές τους· να εξηγείτε πώς τα μέλη μιας ένωσης (union) μοιράζονται την ίδια μνήμη και πόσο μεγάλη είναι μια ένωση· να ορίζετε απαριθμήσεις (enum) και να ξέρετε τι τιμές παίρνουν οι σταθερές τους· και να γράφετε αυτοαναφορικές δομές, τη βάση για συνδεδεμένες λίστες και δυαδικά δέντρα.
>
> **Προαπαιτούμενα**: Κεφάλαιο 19, Κεφάλαιο 12, Κεφάλαιο 2
>
> **Χρόνος μελέτης**: ~2 ώρες

Σύνοψη

Η διάλεξη συνεχίζει τις δομές (Κεφάλαιο 19) με τέσσερα εργαλεία που χτίζουν πάνω τους. Τα **πεδία bit** δίνουν σε ένα μέλος δομής συγκεκριμένο πλήθος bits, για να χωρέσουν πολλές μικρές τιμές σε λίγα bytes. Οι **ενώσεις** μοιάζουν με δομές, αλλά όλα τα μέλη τους μοιράζονται την ίδια μνήμη. Οι **απαριθμήσεις** δίνουν ονόματα σε ακέραιες σταθερές. Τέλος, οι **αυτοαναφορικές δομές** περιέχουν δείκτες σε δομές του ίδιου τύπου· με αυτές φτιάχνουμε αλυσίδες και ιεραρχίες δεδομένων, όπως οι συνδεδεμένες λίστες και τα δυαδικά δέντρα που θα δουλέψουμε στα επόμενα κεφάλαια.

Θεωρία

§20.1 Πεδία bit

Ένα μέλος δομής μπορεί να δηλωθεί ώστε να πιάνει στη μνήμη **συγκεκριμένο αριθμό από bits** αντί για ολόκληρα bytes. Αυτό είναι ένα **πεδίο bit (bit field)**, και γράφεται με άνω-κάτω τελεία και το πλήθος των bits μετά το όνομα του μέλους:

```
struct όνομα {
  τύπος1 πεδίο1 : αριθμός_bits1;
  τύπος2 πεδίο2 : αριθμός_bits2;
  ...
};
```

Ο λόγος ύπαρξης είναι η **εξοικονόμηση μνήμης**. Οι μεταγλωττιστές συνήθως δέχονται για πεδία bit τους τύπους `int`, `long` και `char` (και τις `unsigned` εκδοχές τους). Για να διαλέξετε

το πλήθος των bits, σκεφτείτε πόσες διαφορετικές τιμές πρέπει να χωρέσει το πεδίο: με n bits αναπαριστάτε 2^n τιμές (Κεφάλαιο 2). Στο παράδειγμα της διάλεξης:

```
struct student_status {
  int registered : 1;  /* εγγεγραμμένος ή όχι: 2 τιμές, 1 bit */
  int year : 3;        /* έτος σπουδών */
  int grade : 4;       /* βαθμός 0-10: 11 τιμές, 4 bits αρκούν */
};
```

§20.2 Αναπαράσταση πεδίων bit στη μνήμη

Χωρίς πεδία bit, μια δομή με τρία μέλη `char` πιάνει 3 bytes, ένα για κάθε μέλος. Με `char registered : 1; char year : 3; char grade : 4;` τα τρία μέλη χρειάζονται μαζί $1+3+4=8$ bits, οπότε ο μεταγλωττιστής τα **πακετάρει σε ένα byte** και το `sizeof` της δομής γίνεται 1. Με τιμές `registered = 1`, `year = 1`, `grade = 10` το byte περιέχει τα κομμάτια 1, 001 και 1010:

Πεδίο	`registered`	`year`	`grade`
Bits	1	3	4
Τιμή (δυαδικά)	1	001	1010

Με τύπο `int` αντί για `char` τα πεδία πακετάρονται σε μία «μονάδα» μεγέθους `int`, οπότε το `sizeof` βγαίνει 4 αντί για 12 που θα έπιαναν τρία κανονικά `int`. Το ποια ακριβώς bits του byte παίρνει κάθε πεδίο (από τα αριστερά ή από τα δεξιά) το αποφασίζει ο μεταγλωττιστής· δεν πρέπει να βασίζεστε σε αυτό.

§20.3 Εύρος τιμών και ανάθεση σε πεδία bit

Στα πεδία bit αναθέτουμε και διαβάζουμε με την τελεία, όπως σε κάθε μέλος δομής (`st.year = 2;`). Ένα `unsigned` πεδίο n bits κρατά μόνο τιμές από 0 έως 2^n-1. Αν του αναθέσετε κάτι μεγαλύτερο, κρατιούνται μόνο τα n χαμηλά bits, δηλαδή η τιμή **modulo** 2^n, χωρίς κανένα μήνυμα λάθους. Για το `year` των 3 bits το μέγιστο είναι $2^3-1=7$, και το `st.year = 2; st.year += 7;` δίνει $(2+7) \bmod 8 = 1$. Είναι η ίδια υπερχείλιση που είδαμε στους ακεραίους (Κεφάλαιο 2), μόνο που με λίγα bits συμβαίνει πολύ πιο εύκολα: όταν χρησιμοποιείτε πεδία bit, βεβαιωθείτε ότι κάθε τιμή που μπορεί να πάρει το πεδίο χωράει.

§20.4 Περιορισμοί των πεδίων bit

Ένα πεδίο bit δεν πιάνει ολόκληρα bytes, άρα δεν έχει δικό του μέγεθος ούτε δική του διεύθυνση. Γι' αυτό:

1. **Δεν παίρνουμε το `sizeof` ενός πεδίου bit.** Ο gcc απαντά `error: 'sizeof' applied to a bit-field`.

2. **Δεν παίρνουμε τη διεύθυνση ενός πεδίου bit** με `&`. Ο gcc απαντά `error: cannot take address of bit-field 'year'`. Κατά συνέπεια δεν μπορείτε να δώσετε `&st.year` στη `scanf`.
3. Μπορούν να οδηγήσουν σε **προβλήματα απόδοσης** (ο επεξεργαστής χρειάζεται επιπλέον πράξεις για να απομονώσει τα bits), **δυσκολία ανάγνωσης** και **προβλήματα συμβατότητας** (η διάταξη αλλάζει από μεταγλωττιστή σε μεταγλωττιστή). Η σύσταση της διάλεξης: τα αποφεύγουμε ή τα χρησιμοποιούμε με φειδώ.

§20.5 Ενώσεις

Η **ένωση (union)** μοιάζει με δομή, με μία διαφορά: τα μέλη της **μοιράζονται την ίδια μνήμη**. Σε μια δομή κάθε μέλος έχει τον δικό του χώρο, το ένα μετά το άλλο· σε μια ένωση όλα τα μέλη ξεκινούν από την ίδια διεύθυνση. Η κοινή μνήμη επιτρέπει **εξοικονόμηση χώρου**, με το τίμημα ότι μια γραφή σε ένα μέλος αλλάζει και τα υπόλοιπα. Γι' αυτό σε μια μεταβλητή τύπου ένωσης συνήθως έχει νόημα να χρησιμοποιούμε **μόνο ένα** από τα μέλη της κάθε φορά. Η δήλωση είναι ίδια με της δομής, με τη λέξη-κλειδί `union`:

```
union anything {
  char c;
  int i;
  float f;
  double d;
};
```

Στα μέλη αναφερόμαστε με `.` (ή με `->` μέσω δείκτη), όπως στις δομές.

§20.6 Μέγεθος και κοινή μνήμη της ένωσης

Το **μέγεθος μιας ένωσης** είναι το μέγεθος του **μεγαλύτερου** μέλους της. Για την `union anything`, με `sizeof(double) == 8`, το `sizeof` είναι 8, ενώ μια δομή με τα ίδια μέλη θα έπιανε τουλάχιστον $1 + 4 + 4 + 8 = 17$ bytes. Όλα τα μέλη ξεκινούν από το πρώτο byte:

Byte	0	1	2	3	4	5	6	7
`c`	▯							
`i`, `f`	▯	▯	▯	▯				
`d`	▯	▯	▯	▯	▯	▯	▯	▯

Διαλέγοντας διαφορετικό μέλος, διαλέγουμε **άλλον τρόπο ανάγνωσης των ίδιων bytes**. Αν γράψετε `a1.i = 0x42` και διαβάσετε `a1.c`, παίρνετε το byte 0 του ακεραίου· σε μηχάνημα little-endian (Κεφάλαιο 12) αυτό είναι το λιγότερο σημαντικό byte, `0x42`, δηλαδή ο χαρακτήρας `'B'`.

§20.7 Απαριθμήσεις

Ο **τύπος απαρίθμησης (enumeration type)** `enum` ορίζει ένα σύνολο ακεραίων με συγκεκριμένα ονόματα και σταθερές τιμές:

```
enum όνομα { επιλογή1, επιλογή2, ... };
```

Το όνομα είναι το όνομα της απαρίθμησης, και οι επιλογές είναι οι **σταθερές απαρίθμησης (enumeration constants)** που την αποτελούν. Οι κανόνες για τις τιμές τους είναι δύο:

- Η πρώτη σταθερά **αρχικοποιείται στο 0**, εκτός αν της δοθεί συγκεκριμένη τιμή.
- Κάθε σταθερά χωρίς ρητή τιμή παίρνει την τιμή της **προηγούμενης σταθεράς αυξημένη κατά 1**.

Έτσι στο `enum weekday {Mon, Tue, Wed, Thu, Fri, Sat, Sun};` είναι `Mon == 0` έως `Sun == 6`, ενώ στο `enum weekday {Mon = 1, Tue, ...};` είναι `Mon == 1` έως `Sun == 7`. Μια μεταβλητή δηλώνεται `enum weekday day1 = Mon;` και συμπεριφέρεται σαν ακέραιος: τυπώνεται με `%d`, συγκρίνεται και αυξάνεται με `++`. Οι σημειώσεις παρατηρούν ότι οι απαριθμήσεις κάνουν ό,τι και ένα `#define` για κάθε σταθερά (Κεφάλαιο 15), μόνο που τις διαχειρίζεται ο μεταγλωττιστής και όχι ο προεπεξεργαστής.

§20.8 Αυτοαναφορά

Αυτοαναφορά (self-reference) είναι όταν κάτι αναφέρεται στον εαυτό του. Η διάλεξη τη συστήνει με το παράδοξο του Επιμενίδη (6ος αιώνας π.Χ.): «Αυτή η πρόταση είναι ψευδής». Στον προγραμματισμό την έχουμε ήδη συναντήσει στην αναδρομή (Κεφάλαιο 11), όπου μια συνάρτηση καλεί τον εαυτό της. Εδώ τη συναντάμε στα δεδομένα: μια δομή που περιγράφεται με τη βοήθεια του εαυτού της.

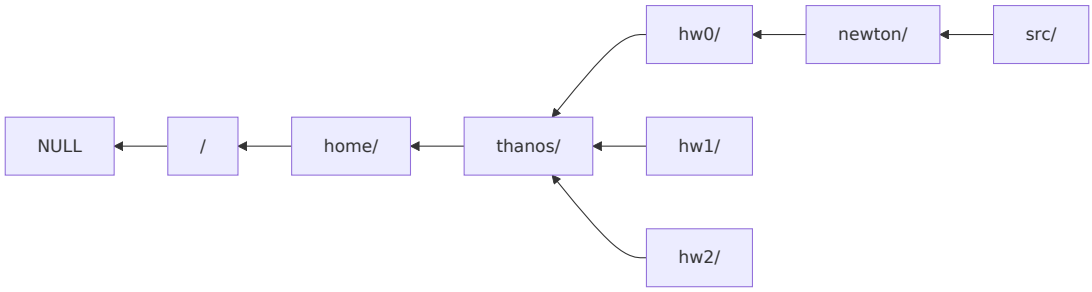
§20.9 Αυτοαναφορικές δομές

Τα μέλη μιας δομής μπορούν να είναι οποιουδήποτε τύπου, **ακόμα και δείκτες σε δομές του ίδιου τύπου**. Μια τέτοια δομή λέγεται **αυτοαναφορική δομή (self-referential struct)**. Το κίνητρο της διάλεξης: θέλουμε να αναπαραστήσουμε έναν φάκελο σε ένα σύστημα αρχείων. Κάθε φάκελος έχει ένα όνομα και βρίσκεται μέσα σε έναν άλλο, **γονικό (parent)** φάκελο· μόνο ο αρχικός φάκελος, η ρίζα `/`, δεν έχει γονέα.

```
struct folder {
  char name[128];          /* κάθε φάκελος έχει ένα όνομα */
  struct folder *parent;   /* και δείκτη στον γονικό φάκελο */
};
```

Το μέλος πρέπει να είναι **δείκτης**. Μια δομή δεν μπορεί να περιέχει ολόκληρη δομή του εαυτού της (`struct folder parent;`), γιατί τότε θα περιείχε ένα αντίγραφο του εαυτού της, που θα περιείχε άλλο αντίγραφο, επ' άπειρον. Ένας δείκτης έχει σταθερό μέγεθος, όποιον τύπο κι αν

δείχνει. Η ρίζα, που δεν έχει γονέα, έχει `parent = NULL`. Από κάθε φάκελο, ακολουθώντας τους δείκτες `parent`, φτάνουμε πάντα στη ρίζα:



Σχήμα: κάθε φάκελος δείχνει με το `parent` στον γονικό του· η ρίζα `/` δείχνει στο `NULL`.

Μια δομή μπορεί να έχει και **πολλούς** δείκτες στον ίδιο τύπο. Για ένα γενεαλογικό δέντρο κάθε άτομο δείχνει στους δύο γονείς του:

```
struct person {
  char name[128];
  struct person *parent1;
  struct person *parent2;
};
```

Με ένα `typedef` (Κεφάλαιο 19) γράφουμε `typedef struct folder { ... } Folder;`. Μέσα στο σώμα της δομής το νέο όνομα `Folder` δεν υπάρχει ακόμα, γι' αυτό ο δείκτης γράφεται `struct folder *parent`.

§20.10 Απλά συνδεδεμένη λίστα

Αν ζωγραφίσουμε τους φακέλους από το `newton/` προς τη ρίζα, η πραγματική διάταξη στη μνήμη μπορεί να είναι περίπλοκη (οι δομές κάθονται όπου τις έβαλε ο μεταγλωττιστής), αλλά σε **αφηρημένη (abstract)** μορφή είναι μια αλυσίδα: κάθε στοιχείο δείχνει στο επόμενο και το τελευταίο στο `NULL`. Αυτή η οργάνωση λέγεται **απλά συνδεδεμένη λίστα (singly linked list)**: ένας τύπος δεδομένων όπου κάθε στοιχείο **δείχνει (links)** στο επόμενο και το τελευταίο δείχνει στο `NULL`. Στη γενική μορφή κάθε **κόμβος (node)** κρατά μια τιμή και τον δείκτη `next`:

```
struct listnode {
  int value;
  struct listnode *next;
};
```

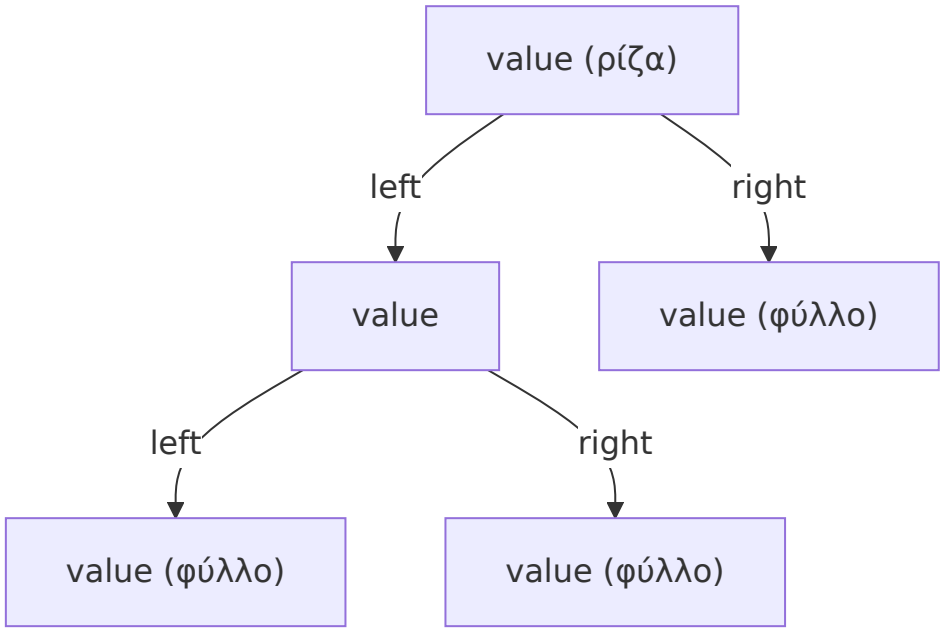
Σχήμα: απλά συνδεδεμένη λίστα μήκους 5.

Το **μήκος λίστας (list length)** είναι ο αριθμός των στοιχείων που περιέχει (5 παραπάνω). Η λίστα μοιάζει με πίνακα, αλλά έχει βασικές διαφορές, που θα δούμε στο Κεφάλαιο 21: συνήθως αποθηκεύεται **εξ ολοκλήρου δυναμικά, στον σωρό** (Κεφάλαιο 13), με μία `malloc` ανά κόμβο, άρα οι κόμβοι δεν είναι συνεχόμενοι στη μνήμη.

§20.11 Δυαδικό δέντρο

Το γενεαλογικό δέντρο της `struct person`, με δύο δείκτες ανά κόμβο, οδηγεί σε μια άλλη οργάνωση. Το **δυαδικό δέντρο (binary tree)** είναι ένας τύπος δεδομένων που οργανώνει τα δεδομένα σε δενδρική διάταξη, όπου κάθε **κόμβος (node)** έχει από 0 έως 2 **κόμβους-παιδιά (children)**, το αριστερό και το δεξί:

```
struct treenode {
  int value;
  struct treenode *left;
  struct treenode *right;
};
```



Σχήμα: δυαδικό δέντρο με βάθος 2· οι δείκτες των φύλλων είναι `NULL`.

Οι βασικοί όροι:

- **Ρίζα (root)**: ο πρώτος κόμβος του δέντρου, που δεν είναι παιδί κανενός.
- **Φύλλα (leaves)**: οι κόμβοι χωρίς παιδιά (και τα δύο `left`, `right` είναι `NULL`).
- **Βάθος (depth)** του δέντρου: ο μέγιστος αριθμός συνδέσμων από τη ρίζα μέχρι τα φύλλα (2 στο σχήμα).

Τα δυαδικά δέντρα έχουν εφαρμογές από βάσεις δεδομένων και αναζήτηση μέχρι μεταγλωττιστές, και από συμπίεση δεδομένων μέχρι κρυπτογραφία: όπου χρειάζεται αναπαράσταση γνώσης. Θα τα δουλέψουμε στα Κεφάλαια 21 και 22.

§20.12 Γράφοι και συστήματα αρχείων

Η διάλεξη κλείνει με δύο ανοιχτά ερωτήματα σχεδίασης, χωρίς έτοιμη απάντηση στις διαφάνειες. Πρώτο, πώς να αναπαραστήσουμε με αυτοαναφορικές δομές έναν χάρτη σε μορφή **γράφου (graph)**: πόλεις A, B, C, D, όπου κάθε ζεύγος συνδέεται με δρόμο που έχει μια απόσταση (A–B 20, A–C 42, A–D 35, B–C 30, B–D 34, C–D 12). Δεύτερο, πώς να αναπαραστήσουμε ένα σύστημα αρχείων ώστε από κάθε φάκελο να βρίσκουμε **άμεσα** και τους υποφακέλους του, όχι μόνο τον γονικό. Και τα δύο είναι ασκήσεις (δείτε «Ασκήσεις»): σκεφτείτε ποιους δείκτες πρέπει να κρατά κάθε κόμβος, και τι κάνετε όταν το πλήθος τους δεν είναι σταθερό.

Παραδείγματα

§20.13 Το μέγεθος μιας δομής με πεδία bit

Εφαρμόζει τις ενότητες «Πεδία bit» και «Αναπαράσταση πεδίων bit στη μνήμη». Η διάλεξη τυπώνει το `sizeof(struct student_status)` για τρεις εκδοχές της δομής:

```c
#include <stdio.h>

struct status_int  { int  registered : 1; int  year : 3; int  grade : 4; };
struct status_bits { char registered : 1; char year : 3; char grade : 4; };
struct status_char { char registered;     char year;     char grade;     };

int main() {
  printf("%zu\n", sizeof(struct status_int));
  printf("%zu\n", sizeof(struct status_bits));
  printf("%zu\n", sizeof(struct status_char));
  return 0;
}
```

Στις διαφάνειες κάθε εκδοχή λέγεται `struct student_status` και είναι ξεχωριστό πρόγραμμα (`./bitfields`, `./bitfields2`)· εδώ τις βάλαμε μαζί με διαφορετικά ονόματα. Τα αποτελέσματα:

```
$ ./bitfields
4
$ ./bitfields2
1
```

και 3 για τη δομή χωρίς πεδία bit. Τα 8 bits των πεδίων χωράνε σε μία μονάδα του τύπου τους: ένα `int` (4 bytes) στην πρώτη περίπτωση, ένα `char` (1 byte) στη δεύτερη. Χωρίς πεδία bit κάθε

char πιάνει το δικό του byte.

§20.14 Ανάθεση σε πεδίο bit: 2 + 7 = 1

Εφαρμόζει την ενότητα «Εύρος τιμών και ανάθεση σε πεδία bit». Η διάλεξη ρωτά τι τυπώνει το πρόγραμμα:

```c
#include <stdio.h>

typedef struct {
  unsigned char registered : 1;
  unsigned char year : 3;
  unsigned char grade : 4;
} status;

int main() {
  status st = {1, 1, 10};
  printf("Status: %u %u %u\n", st.registered, st.year, st.grade);
  st.year = 2;
  printf("Status: %u %u %u\n", st.registered, st.year, st.grade);
  st.year += 7;
  printf("Status: %u %u %u\n", st.registered, st.year, st.grade);
  return 0;
}
```

```
$ ./bitfields3
Status: 1 1 10
Status: 1 2 10
Status: 1 1 10
```

Η αρχικοποίηση `{1, 1, 10}` γεμίζει τα πεδία με τη σειρά δήλωσης, όπως σε κάθε δομή. Το ενδιαφέρον είναι η τρίτη γραμμή: το `year` έχει μόνο 3 bits, άρα κρατά τιμές 0 έως 7. Το $2 + 7 = 9$ είναι `1001` στο δυαδικό· κρατιούνται τα 3 χαμηλά bits, `001`, δηλαδή $9 \bmod 8 = 1$. Η C δεν δίνει κανένα λάθος: «θα υποστούμε τις συνέπειες».

§20.15 Μια ένωση με τέσσερα πρόσωπα

Εφαρμόζει τις ενότητες «Ενώσεις» και «Μέγεθος και κοινή μνήμη της ένωσης»:

```c
#include <stdio.h>

union anything {
  char c;  int i;  float f;  double d;
};
```

```c
int main() {
  union anything a1;
  a1.i = 0x42;
  printf("%d %c\n", a1.i, a1.c);
  a1.c = 'C';
  printf("%d %c\n", a1.i, a1.c);
  printf("%zu\n", sizeof(a1));
  return 0;
}
```

```
$ ./union
66 B
67 C
8
```

- Το `a1.i = 0x42` γράφει τον ακέραιο 66 στα bytes 0–3. Το `a1.c` διαβάζει το byte 0, που (σε little-endian μηχάνημα) είναι το `0x42`, ο χαρακτήρας `'B'`.
- Το `a1.c = 'C'` γράφει 67 (`0x43`) μόνο στο byte 0. Τα bytes 1–3 του `i` ήταν ήδη 0, οπότε τώρα και το `a1.i` διαβάζεται 67.
- Το `sizeof(a1)` είναι 8, όσο το μεγαλύτερο μέλος, το `double`.

§20.16 Απαρίθμηση ημερών

Εφαρμόζει την ενότητα «Απαριθμήσεις». Τι τυπώνει το πρόγραμμα;

```c
#include <stdio.h>

enum weekday {Mon, Tue, Wed, Thu, Fri, Sat, Sun};

int main() {
  enum weekday day1 = Mon, day2;
  day2 = Fri;
  printf("%d %d\n", day1, day2);
  return 0;
}
```

```
$ ./enum1
0 4
```

Η πρώτη σταθερά, `Mon`, είναι 0 και κάθε επόμενη ένα παραπάνω, άρα `Fri` είναι 4.

§20.17 Επανάληψη πάνω σε απαρίθμηση

Με ρητή τιμή στην πρώτη σταθερά, οι ημέρες αριθμούνται από το 1. Μια μεταβλητή `enum` μπορεί να είναι μετρητής βρόχου:

```c
#include <stdio.h>

enum weekday {Mon = 1, Tue, Wed, Thu, Fri, Sat, Sun};

int main() {
  for (enum weekday day = Mon; day <= Sun; day++) {
    if (day == Mon || day == Fri)
      printf("We have class on day # %d of the week\n", day);
  }
  return 0;
}
```

```
$ ./enum2
We have class on day # 1 of the week
We have class on day # 5 of the week
```

Ο βρόχος πάει από το 1 (Mon) έως το 7 (Sun)· τυπώνει μόνο για Δευτέρα (1) και Παρασκευή (5). Τα ονόματα κάνουν τον κώδικα πιο ευανάγνωστο από τα σκέτα 1 και 5.

§20.18 Ακολουθώντας τον γονικό φάκελο

Εφαρμόζει την ενότητα «Αυτοαναφορικές δομές». Οι φάκελοι είναι τοπικές μεταβλητές και ο καθένας δείχνει στη διεύθυνση του γονικού του:

```c
#include <stdio.h>

typedef struct folder {
  char name[128];
  struct folder *parent;
} Folder;

int main() {
  Folder root = {"/", NULL};
  Folder home = {"home/", &root};
  Folder user = {"thanos/", &home};
  Folder hw0 = {"hw0/", &user}, hw1 = {"hw1/", &user};
  Folder newton = {"newton/", &hw0};
  printf("%s -> %s -> %s\n", newton.name, newton.parent->name,
         newton.parent->parent->name);
  return 0;
}
```

```
$ ./self
newton/ -> hw0/ -> thanos/
```

Το `newton.parent` είναι δείκτης, άρα στο μέλος του γονέα φτάνουμε με `->` (`newton.parent-`

`->name`). Αλυσιδωτά, `newton.parent->parent` είναι ο δείκτης στον `user`, και το `->name` του δίνει `thanos/`. Ο `hw1` δηλώνεται αλλά δεν χρησιμοποιείται (ο gcc με `-Wall` θα προειδοποιήσει για αχρησιμοποίητη μεταβλητή).

§20.19 Διάσχιση μέχρι τη ρίζα

Στην ίδια `main`, με τις ίδιες δηλώσεις φακέλων, η διάλεξη αντικαθιστά το `printf` με έναν βρόχο που ακολουθεί τους δείκτες μέχρι το `NULL`:

```
for (Folder *iterator = &newton; iterator; iterator = iterator->parent)
  printf("folder: %s\n", iterator->name);
```

```
$ ./self2
folder: newton/
folder: hw0/
folder: thanos/
folder: home/
folder: /
```

Ο δείκτης `iterator` ξεκινά από τον `newton` και σε κάθε βήμα γίνεται ο γονέας του. Η συνθήκη `iterator` είναι ψευδής μόνο όταν ο δείκτης γίνει `NULL`, δηλαδή μετά τη ρίζα. Στη μνήμη οι πέντε δομές κάθονται στη στοίβα η μία δίπλα στην άλλη με οποιαδήποτε σειρά διαλέξει ο μεταγλωττιστής (στη διαφάνεια 37 π.χ. ο `hw0` είναι πάνω από τον `home`)· οι δείκτες τις συνδέουν ανεξάρτητα από τη θέση τους. Αυτό ακριβώς το μοτίβο, «ξεκίνα από την αρχή, ακολούθα το `next` μέχρι το `NULL`», είναι η διάσχιση μιας συνδεδεμένης λίστας (Κεφάλαιο 21).

Κύρια σημεία

1. Ένα πεδίο bit (`τύπος όνομα : bits;`) δίνει σε ένα μέλος δομής συγκεκριμένο αριθμό bits για να εξοικονομήσουμε μνήμη· τα πεδία πακετάρονται σε μονάδες του τύπου τους (τρία πεδία `char` με 8 bits συνολικά πιάνουν 1 byte).
2. Ένα `unsigned` πεδίο n bits κρατά τιμές 0 έως $2^n - 1$· μεγαλύτερες τιμές αποθηκεύονται modulo 2^n χωρίς προειδοποίηση.
3. Δεν μπορούμε να πάρουμε το `sizeof` ή τη διεύθυνση (`&`) ενός πεδίου bit, και τα πεδία bit έχουν κόστος σε απόδοση, αναγνωσιμότητα και συμβατότητα· τα χρησιμοποιούμε με φειδώ.
4. Σε μια ένωση (`union`) όλα τα μέλη μοιράζονται την ίδια μνήμη και κάθε μέλος είναι άλλος τρόπος ανάγνωσης των ίδιων bytes· συνήθως χρησιμοποιούμε ένα μέλος κάθε φορά, και το μέγεθος της ένωσης είναι του μεγαλύτερου μέλους.
5. Μια απαρίθμηση (`enum`) δίνει ονόματα σε ακέραιες σταθερές: η πρώτη είναι 0 εκτός αν δοθεί τιμή, και κάθε επόμενη χωρίς τιμή είναι η προηγούμενη συν 1.
6. Μια αυτοαναφορική δομή έχει μέλη που είναι δείκτες σε δομές του ίδιου τύπου (δείκτες, όχι ολόκληρες δομές)· ακολουθώντας τους με `->` μέχρι το `NULL` διασχίζουμε μια αλυσίδα δεδομένων.

7. Η απλά συνδεδεμένη λίστα είναι αλυσίδα κόμβων όπου ο καθένας δείχνει στον επόμενο και ο τελευταίος στο `NULL`· συνήθως αποθηκεύεται δυναμικά στον σωρό.
8. Το δυαδικό δέντρο έχει κόμβους με 0 έως 2 παιδιά (`left`, `right`)· ξεκινά από τη ρίζα, καταλήγει στα φύλλα, και το βάθος του είναι ο μέγιστος αριθμός συνδέσμων από τη ρίζα ως ένα φύλλο.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
πεδίο bit	bit field	Μέλος δομής με δηλωμένο πλήθος bits (`int year : 3;`).
ένωση	union	Τύπος σαν τη δομή, όπου όλα τα μέλη μοιράζονται την ίδια μνήμη.
απαρίθμηση	enumeration (`enum`)	Τύπος με ονομασμένες ακέραιες σταθερές.
σταθερά απαρίθμησης	enumeration constant	Ένα από τα ονόματα μιας απαρίθμησης, π.χ. `Mon`.
αυτοαναφορά	self-reference	Όταν κάτι αναφέρεται στον εαυτό του.
αυτοαναφορική δομή	self-referential struct	Δομή με μέλος-δείκτη σε δομή του ίδιου τύπου.
γονικός φάκελος	parent folder	Ο φάκελος που περιέχει έναν άλλο φάκελο.
απλά συνδεδεμένη λίστα	singly linked list	Αλυσίδα κόμβων, ο καθένας δείχνει στον επόμενο, ο τελευταίος στο `NULL`.
κόμβος	node	Ένα στοιχείο λίστας ή δέντρου (μια δομή).
μήκος λίστας	list length	Ο αριθμός των στοιχείων της λίστας.
δυαδικό δέντρο	binary tree	Δενδρική διάταξη κόμβων με 0 έως 2 παιδιά ο καθένας.
ρίζα	root	Ο πρώτος κόμβος ενός δέντρου.
φύλλο	leaf	Κόμβος χωρίς παιδιά.
βάθος	depth	Ο μέγιστος αριθμός συνδέσμων από τη ρίζα ως ένα φύλλο.
γράφος	graph	Κόμβοι συνδεδεμένοι με ακμές, π.χ. πόλεις και δρόμοι.

Διάβασμα

- **Διαφάνειες**: Διάλεξη 20, σελ. 1–49: πεδία bit 5–15· ενώσεις 16–22· απαριθμήσεις 23–28· αυτοαναφορά και αυτοαναφορικές δομές 29–37, 42–43· συνδεδεμένη λίστα 38–41·

δυαδικό δέντρο 44–45· ανοιχτά ερωτήματα (γράφος, σύστημα αρχείων) 46–47.

- **Σημειώσεις**: η διάλεξη προτείνει τις σελ. 108 και 120–135 των διαφανειών του κ. Σταματόπουλου:
  - Κεφάλαιο 7: Απαριθμήσεις, δομές και ενώσεις: «Απαριθμήσεις» (K04, σελ. 108), «Αυτο-αναφορικές δομές» (120–125), «Δημιουργία νέων ονομάτων τύπων» (126), «Ενώσεις και πεδία bit» (127).
  - Κεφάλαιο 8: Λίστες και δυαδικά δέντρα: «Διαχείριση συνδεδεμένων λιστών» (K04, σελ. 128–131), «Διαχείριση δυαδικών δέντρων» (132–135), για την επόμενη διάλεξη.
- **Εργαστήριο**: Εργαστήριο 9: ασκήσεις `grades.c` (αυτοαναφορική δομή λίστας) και `tree.c` (αυτοαναφορική δομή δυαδικού δέντρου).
- **Άλλα**: Bit fields (GeeksforGeeks) και reference (cppreference)· Unions (cppreference)· Enums (GeeksforGeeks)· Linked list και Binary tree (Wikipedia)· Self-reference και Epimenides paradox (Wikipedia).

Συχνά λάθη

- **Πεδίο bit πολύ μικρό για τις τιμές του**. Με `unsigned char year : 3;` το `st.year = 9;` αποθηκεύει 1. Υπολογίστε τις τιμές που χρειάζεστε και δώστε $\lceil \log_2(\text{πλήθος τιμών}) \rceil$ bits.
- **`sizeof` ή `&` σε πεδίο bit**. `sizeof(st.year)` και `&st.year` δεν μεταγλωττίζονται (`'sizeof' applied to a bit-field`, `cannot take address of bit-field`). Διαβάστε σε μια κανονική μεταβλητή με `scanf` και μετά αναθέστε τη στο πεδίο.
- **Πεδίο bit 1 bit τύπου `int`**. Αν το `int` πεδίο θεωρηθεί προσημασμένο (όπως κάνει ο gcc), το `int registered : 1` κρατά 0 και -1, όχι 0 και 1. Για σημαίες χρησιμοποιήστε `unsigned` τύπο, όπως το πρόγραμμα `bitfields3`.
- **Χρήση πολλών μελών μιας ένωσης σαν να ήταν δομή**. Μετά το `a1.i = 1000; a1.c = 'x';` το `a1.i` δεν είναι πια 1000, γιατί το `c` έγραψε πάνω στο πρώτο byte του. Αν χρειάζεστε όλα τα μέλη ταυτόχρονα, θέλετε `struct`.
- **Ολόκληρη δομή αντί για δείκτη μέσα στον εαυτό της**. Το `struct folder { char name[128]; struct folder parent; };` δεν μεταγλωττίζεται (`field 'parent' has incomplete type`). Το μέλος πρέπει να είναι `struct folder *parent;`.
- **Χρήση του ονόματος του `typedef` μέσα στη δομή**. Στο `typedef struct folder { Folder *parent; } Folder;` το `Folder` δεν έχει ακόμα οριστεί (`unknown type name 'Folder'`). Μέσα στο σώμα γράψτε `struct folder *`.
- **`.` αντί για `->` σε δείκτη**. Το `newton.parent.name` δεν μεταγλωττίζεται, γιατί το `parent` είναι δείκτης· γράψτε `newton.parent->name`.
- **Ξεχασμένο `NULL` στο τέλος της αλυσίδας**. Αν η ρίζα δεν έχει `parent = NULL`, ο βρόχος `for (p = &newton; p; p = p->parent)` ακολουθεί σκουπίδια και καταλήγει σε `Segmentation fault`.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K20.8** Μέγεθος ένωσης (36% σωστές): Το 22% επέλεξε `sizeof(double)`, θεωρώντας τον `double` το μεγαλύτερο πεδίο, ενώ ο πίνακας `name` πιάνει 20 bytes· και το 18% πρόσθεσε τα μεγέθη, όπως σε μια δομή.
- **K20.7** Περιορισμοί πεδίων bit (44% σωστές): Το 26% επέλεξε μόνο το `sizeof`, χωρίς να δει ότι ένα πεδίο bit δεν έχει ούτε δική του διεύθυνση (δεν ξεκινάει απαραίτητα σε byte) ούτε χωράει τιμές πέρα από τα n bits του.
- **K20.4** Αυτοαναφορική δομή (56% σωστές): Το 22% επέλεξε «περιέχει τον εαυτό της», κάτι αδύνατο: μια δομή που περιέχει αντίγραφο του εαυτού της θα είχε άπειρο μέγεθος.

Ερωτήσεις κατανόησης

- **E20.1** Πόσα bits χρειάζεται ένα πεδίο bit που κρατά τον μήνα (1–12);[156]
- **E20.2** Ένα `unsigned char x : 4;` έχει τιμή 12. Τι τιμή έχει μετά το `x += 5;`;[157]
- **E20.3** Γιατί δεν μπορείτε να γράψετε `scanf("%u", &st.year);` όταν το `year` είναι πεδίο bit;[158]
- **E20.4** Πόσο είναι το `sizeof` μιας `union` με μέλη `char`, `int` και `double`, και γιατί;[159]
- **E20.5** Στο `enum color {RED, GREEN = 5, BLUE};` τι τιμή έχει το `BLUE`;[160]
- **E20.6** Γιατί η `struct folder` περιέχει `struct folder *parent` και όχι `struct folder parent`;[161]
- **E20.7** Πώς καταλαβαίνει ένας βρόχος που ακολουθεί τους δείκτες `parent` ότι έφτασε στη ρίζα;[162]
- **E20.8** Πόσους δείκτες σε `struct person` χρειάζεται κάθε κόμβος γενεαλογικού δέντρου, και σε ποια δομή δεδομένων της διάλεξης μοιάζει αυτό;[163]

Kahoot από το αμφιθέατρο (K20.1–K20.8)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K20.1** Κόμβος με left και right: 67% σωστές απαντήσεις

[156] 4 bits: οι 12 τιμές δεν χωράνε σε 3 bits ($2^3 = 8$), αλλά χωράνε σε 4 ($2^4 = 16$).

[157] $(12 + 5) \bmod 16 = 1$: το πεδίο κρατά μόνο τα 4 χαμηλά bits του 17.

[158] Δεν μπορούμε να πάρουμε τη διεύθυνση ενός πεδίου bit (`cannot take address of bit-field`). Διαβάστε σε κανονική μεταβλητή και αναθέστε.

[159] 8, όσο το μεγαλύτερο μέλος (`double`), γιατί όλα τα μέλη μοιράζονται την ίδια μνήμη.

[160] 6: το `BLUE` δεν έχει ρητή τιμή, οπότε παίρνει την προηγούμενη (`GREEN = 5`) συν 1.

[161] Μια δομή δεν μπορεί να περιέχει αντίγραφο του εαυτού της (θα είχε άπειρο μέγεθος)· ένας δείκτης έχει σταθερό μέγεθος.

[162] Η ρίζα έχει `parent = NULL`· ο βρόχος σταματά όταν ο δείκτης γίνει `NULL`.

[163] Δύο (`parent1`, `parent2`)· η δομή μοιάζει με δυαδικό δέντρο, όπου κάθε κόμβος έχει έως δύο δείκτες προς άλλους κόμβους.

- **K20.2** Σε τι χρησιμεύει η ένωση: 65% σωστές απαντήσεις
- **K20.3** Δομή ή ένωση;: 63% σωστές απαντήσεις
- **K20.4** Αυτοαναφορική δομή: 56% σωστές απαντήσεις
- **K20.5** Κόμβος με next: 52% σωστές απαντήσεις
- **K20.6** Τιμές απαρίθμησης: 48% σωστές απαντήσεις
- **K20.7** Περιορισμοί πεδίων bit: 44% σωστές απαντήσεις
- **K20.8** Μέγεθος ένωσης: 36% σωστές απαντήσεις

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A20.1–A20.12)

- **A20.1** Μέγεθος δομής με πεδία bit: Διάλεξη 20, διαφάνεια 7 · ★☆☆ · trace · `slides-lec20-bitfield-sizeof`
- **A20.2** Βρόχος με απαρίθμηση: Διάλεξη 20, διαφάνεια 27 · ★☆☆ · trace · `slides-lec20-enum-loop`
- **A20.3** Τιμές απαρίθμησης: Διάλεξη 20, διαφάνεια 25 · ★☆☆ · trace · `slides-lec20-enum-values`
- **A20.4** Γενεαλογικό δέντρο: Διάλεξη 20, διαφάνεια 43 · ★☆☆ · short-answer · `slides-lec20-family-tree`
- **A20.5** Γονικοί φάκελοι: Διάλεξη 20, διαφάνεια 33 · ★☆☆ · trace · `slides-lec20-folder-parent`
- **A20.6** Δομή για φάκελο: Διάλεξη 20, διαφάνεια 31 · ★☆☆ · short-answer · `slides-lec20-folder-struct`
- **A20.7** Η διάταξη των φακέλων: Διάλεξη 20, διαφάνεια 38 · ★☆☆ · short-answer · `slides-lec20-list-layout`
- **A20.8** Ανάθεση σε πεδία bit: Διάλεξη 20, διαφάνεια 12 · ★★☆ · trace · `slides-lec20-bitfield-assign`
- **A20.9** Σύστημα αρχείων με υποφακέλους: Διάλεξη 20, διαφάνεια 47 · ★★☆ · short-answer · `slides-lec20-filesystem-children`
- **A20.10** Διάσχιση φακέλων: Διάλεξη 20, διαφάνεια 35 · ★★☆ · trace · `slides-lec20-folder-iterate`
- **A20.11** Χρήση ένωσης και μέγεθος: Διάλεξη 20, διαφάνεια 19 · ★★☆ · trace · `slides-lec20-union-size`
- **A20.12** Χάρτης ως γράφος: Διάλεξη 20, διαφάνεια 46 · ★★★ · short-answer · `slides-lec20-graph-map`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A22.20** Διερμηνέας Αριθμητικών Εκφράσεων - eval: Εξέταση Ιανουαρίου 2026, Θέμα 4 · ★★☆ · programming · `exam-2026-jan-q4`

Κεφάλαιο 21: Λίστες και Δέντρα

> **Στόχοι**: μετά από αυτό το κεφάλαιο θα μπορείτε να ορίζετε απλά συνδεδεμένες λίστες και δυαδικά δέντρα με αυτοαναφορικές δομές· να γράφετε τις βασικές λειτουργίες τους (`is_empty`, `insert`, `print`, `length`, `find`, `delete`, `depth`)· να διασχίζετε ένα δέντρο pre-order, in-order, post-order και κατά πλάτος· να αναζητάτε σε δυαδικό δέντρο αναζήτησης· και να εκτιμάτε την πολυπλοκότητα χρόνου και χώρου κάθε λειτουργίας.
>
> **Προαπαιτούμενα**: Κεφάλαιο 11, Κεφάλαιο 13, Κεφάλαιο 20
>
> **Χρόνος μελέτης**: ~3 ώρες

Σύνοψη

Οι αυτοαναφορικές δομές της προηγούμενης διάλεξης γίνονται εδώ πραγματικές δομές δεδομένων. Η απλά συνδεδεμένη λίστα είναι μια αλυσίδα από κόμβους στον σωρό, όπου κάθε κόμβος δείχνει στον επόμενο· σε αντίθεση με τον πίνακα, μεγαλώνει και αναδιατάσσεται εύκολα, αλλά η πρόσβαση σε ένα στοιχείο κοστίζει γραμμικό χρόνο. Το δυαδικό δέντρο δίνει σε κάθε κόμβο δύο δείκτες, αριστερό και δεξί, και οργανώνει τα δεδομένα ιεραρχικά. Η διάλεξη υλοποιεί τις βασικές λειτουργίες και των δύο, κυρίως αναδρομικά για τα δέντρα, παρουσιάζει τις διασχίσεις κατά βάθος (DFS) και κατά πλάτος (BFS), και δείχνει πώς ένα δυαδικό δέντρο αναζήτησης κάνει την αναζήτηση λογαριθμική. Είναι η βάση για τη διάλεξη 22 και για την Εργασία #2.

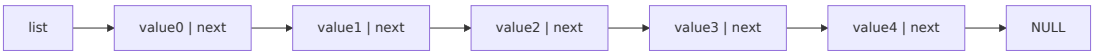
Θεωρία

§21.1 Απλά συνδεδεμένη λίστα

Η **απλά συνδεδεμένη λίστα (single linked list)** είναι ένας τύπος δεδομένων όπου κάθε στοιχείο δείχνει (links) στο επόμενο, και το τελευταίο δείχνει στο `NULL`. Κάθε στοιχείο είναι ένας **κόμβος (node)**: μια αυτοαναφορική δομή (Κεφάλαιο 20) με την τιμή και έναν δείκτη σε δομή του ίδιου τύπου:

```
struct listnode {
  int value;
  struct listnode * next;
};
```

Το πρώτο στοιχείο λέγεται **κεφαλή (head)** της λίστας και το τελευταίο (συνήθως) **ουρά (tail)**. Το **μήκος λίστας (list length)** είναι ο αριθμός των στοιχείων που περιέχει.



Σχήμα: λίστα μήκους 5· η κεφαλή είναι το `value0`*, η ουρά το* `value4`*.*

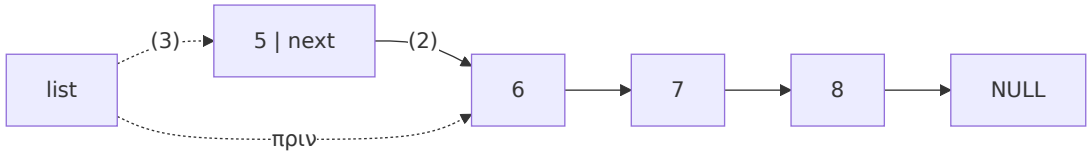
Η λίστα ως σύνολο αναπαριστάται από έναν δείκτη στην κεφαλή της, γι' αυτό η διάλεξη ορίζει τον τύπο `List` ως **δείκτη** σε κόμβο: `typedef struct listnode {int value; struct listnode * next;} * List;`. Η **κενή λίστα** είναι απλώς ο δείκτης `NULL`. Οι κόμβοι δεν βρίσκονται σε συνεχόμενες θέσεις: ο καθένας δεσμεύεται χωριστά με `malloc` και μπορεί να είναι οπουδήποτε στη μνήμη (στη διαφάνεια 4, στις διευθύνσεις 4800, 4900, 5000 και 3000). Η σειρά τους ορίζεται μόνο από τους δείκτες `next`.

§21.2 Βασικές λειτουργίες με λίστες

Η διάλεξη υλοποιεί έξι λειτουργίες: `is_empty` (η λίστα είναι `NULL`), `insert` (προσθήκη), `print` (τύπωμα), `length` (μήκος), `find` (εύρεση) και `delete` (αφαίρεση στοιχείου). Οι `print`, `length` και `find` είναι **διασχίσεις (traversal)**: ένας δείκτης ξεκινά από την κεφαλή και προχωρά με `list = list->next` μέχρι το `NULL`. Επειδή η συνάρτηση παίρνει αντίγραφο του δείκτη, η μετακίνησή του δεν αλλάζει τη λίστα του καλούντος.

§21.3 Εισαγωγή στην αρχή της λίστας

Για να προσθέσουμε το 5 στη λίστα $6 \longrightarrow 7 \longrightarrow 8$, ο φθηνότερος τρόπος είναι να γίνει ο νέος κόμβος η κεφαλή: (1) δεσμεύουμε νέο κόμβο στον **σωρό (heap)** με `malloc`, (2) τον αρχικοποιούμε με την τιμή και με `next` την παλιά κεφαλή, (3) κάνουμε τον δείκτη της λίστας να δείχνει στον νέο κόμβο. Κανένας άλλος κόμβος δεν μετακινείται, άρα το κόστος είναι σταθερό, $O(1)$.



Σχήμα: εισαγωγή του 5 στην αρχή· ο δείκτης `list` *αλλάζει από το 6 στο 5.*

Το βήμα (3) αλλάζει τη μεταβλητή `list` του καλούντος. Επειδή η C περνά τα ορίσματα με τιμή, η `insert` πρέπει να πάρει τη **διεύθυνσή** της, δηλαδή έναν `List *` (δείκτη σε δείκτη, Κεφάλαιο 12), και να γράψει `*list = new_head`. Αν έπαιρνε σκέτο `List`, θα άλλαζε μόνο το τοπικό της αντίγραφο και η νέα κεφαλή θα χανόταν. Αφού κάθε νέο στοιχείο μπαίνει μπροστά, η λίστα καταλήγει με τα στοιχεία σε **αντίστροφη** σειρά εισαγωγής.

§21.4 Μήκος λίστας: επανάληψη και αναδρομή

Το μήκος υπολογίζεται με έναν μετρητή μέσα στη διάσχιση. Υπάρχει όμως και μια φυσική αναδρομική διατύπωση, αφού η λίστα είναι αναδρομική δομή: η κενή λίστα έχει μήκος 0, και μια μη κενή λίστα έχει μήκος 1 συν το μήκος της υπόλοιπης (`list->next`).

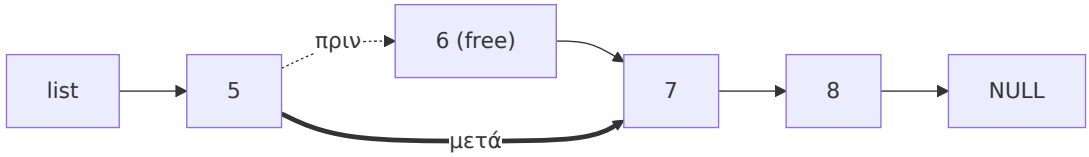
```
int length(List list) {
  if (!list) return 0;
  return 1 + length(list->next);
}
```

Και οι δύο εκδοχές επισκέπτονται κάθε κόμβο μία φορά, άρα χρόνος $O(n)$. Η επαναληπτική χρειάζεται σταθερό χώρο, $O(1)$· η αναδρομική κρατά στη στοίβα n ενεργές κλήσεις ταυτόχρονα, άρα χώρο $O(n)$.

§21.5 Αναζήτηση και αφαίρεση στοιχείου

Η `find` σταματά τη διάσχιση στον πρώτο κόμβο με την τιμή που ψάχνουμε και επιστρέφει δείκτη σε αυτόν, ή `NULL` αν φτάσει στο τέλος. Η συνθήκη `list && list->value != value` βασίζεται στη βραχυκυκλωμένη αποτίμηση του `&&`: αν ο `list` είναι `NULL`, το `list->value` δεν αποτιμάται. Χρόνος $O(n)$, χώρος $O(1)$.

Για να αφαιρέσουμε το 6 από τη λίστα 5 ⟶ 6 ⟶ 7 ⟶ 8, κάνουμε τον δείκτη που έδειχνε στο 6 (το `next` του 5) να δείχνει στον επόμενο του 6 (το 7) και αποδεσμεύουμε τον κόμβο με `free`:



Σχήμα: αφαίρεση του 6· το `next` *του 5 παρακάμπτει τον κόμβο, που αποδεσμεύεται.*

Το δύσκολο είναι ότι ο «δείκτης που δείχνει στον κόμβο» μπορεί να είναι είτε το `next` του προηγούμενου κόμβου είτε, αν αφαιρούμε την κεφαλή, η ίδια η μεταβλητή `list` του καλούντος. Η `delete` της διάλεξης λύνει και τις δύο περιπτώσεις μαζί: παίρνει `List *` και κρατά πάντα τη **διεύθυνση του δείκτη** που δείχνει στον τρέχοντα κόμβο. Με `list = &((*list)->next)` προχωρά στον επόμενο σύνδεσμο, και όταν βρει την τιμή, το `*list = temp->next` αλλάζει ακριβώς τον σωστό σύνδεσμο. Αν η τιμή δεν υπάρχει, η λίστα μένει ως έχει.

§21.6 Πίνακες ή λίστες;

	Πίνακες	Λίστες
Θέση στη μνήμη	συνεχόμενες θέσεις	οποιαδήποτε θέση

	Πίνακες	Λίστες
Χώρος	όσος χρειάζεται για τα στοιχεία	επιπλέον ένα `sizeof(pointer)` ανά στοιχείο
Πρόσβαση στο i-οστό	σταθερός χρόνος, $O(1)$, με `array[i]`	γραμμικός χρόνος, $O(n)$
Αναδιάταξη	συνήθως $O(n)$ (π.χ. εισαγωγή στο `array[0]`)	εύκολη και γρήγορη, με αλλαγή δεικτών
Δήλωση	πρέπει να ξέρουμε πόσα στοιχεία θα μπουν (πιο στατική δομή)	δεν χρειάζεται (πιο δυναμική δομή)

Η επιλογή εξαρτάται από το τι κάνει συχνότερα το πρόγραμμα: πολλές προσβάσεις με θέση ευνοούν τον πίνακα, πολλές εισαγωγές και αφαιρέσεις με άγνωστο πλήθος τη λίστα.

§21.7 Δυαδικό δέντρο

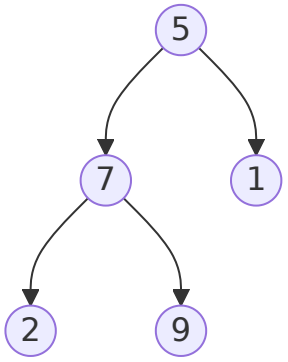
Το **δυαδικό δέντρο (binary tree)** είναι ένας τύπος δεδομένων που οργανώνει τα δεδομένα σε δενδρική διάταξη: κάθε κόμβος έχει από 0 έως 2 **κόμβους-παιδιά (children)**, έναν αριστερό και έναν δεξί.

```
struct treenode {
  int value;
  struct treenode * left;
  struct treenode * right;
};
```

Όπως στη λίστα, η διάλεξη ορίζει `typedef struct treenode {...} * Tree;` και το άδειο δέντρο είναι το `NULL`. Τα δέντρα χρησιμοποιούνται από βάσεις δεδομένων και αναζήτηση μέχρι μεταγλωττιστές, συμπίεση δεδομένων και κρυπτογραφία, όπου χρειάζεται αναπαράσταση γνώσης.

Οι βασικοί όροι:

- **Ρίζα (root)**: ο πρώτος κόμβος του δέντρου.
- **Φύλλα (leaves)**: οι κόμβοι χωρίς παιδιά.
- **Βάθος (depth)** του δέντρου: ο μέγιστος αριθμός συνδέσμων από τη ρίζα μέχρι τα φύλλα.
- **Ύψος (height)** του δέντρου: ο μέγιστος αριθμός συνδέσμων από τα φύλλα μέχρι τη ρίζα. Για ολόκληρο το δέντρο είναι ο ίδιος αριθμός με το βάθος, μετρημένος από την άλλη άκρη.
- **Επίπεδο κόμβου (node level)**: ο αριθμός των κόμβων που μεσολαβούν μέχρι τη ρίζα, μετρώντας και τους δύο· η ρίζα είναι στο επίπεδο 1.



Σχήμα: το δέντρο της διάλεξης· ρίζα το 5, φύλλα τα 2, 9, 1, βάθος 2, τρία επίπεδα (5 | 7, 1 | 2, 9).

§21.8 Τύποι δυαδικών δέντρων

- **Τέλειο (perfect binary tree)**: όλοι οι εσωτερικοί κόμβοι έχουν δύο παιδιά και όλα τα φύλλα βρίσκονται στο ίδιο επίπεδο. Παράδειγμα: 5 με παιδιά 7, 4 και εγγόνια 2, 9, 8, 3. Κάθε επίπεδο έχει διπλάσιους κόμβους από το προηγούμενο.
- **Γεμάτο (full binary tree)**: όλοι οι κόμβοι έχουν 0 ή 2 παιδιά. Παράδειγμα: 5 με παιδιά 7 (φύλλο) και 4, και το 4 με παιδιά 8, 3.
- **Πλήρες (complete binary tree)**: κάθε επίπεδο, εκτός ίσως από το τελευταίο, είναι γεμάτο, και οι κόμβοι του τελευταίου επιπέδου είναι όσο πιο αριστερά γίνεται. Παράδειγμα: το τέλειο δέντρο χωρίς το 3.
- **Ισορροπημένο (balanced binary tree)**: σε κάθε κόμβο, τα ύψη του αριστερού και του δεξιού υποδέντρου διαφέρουν το πολύ κατά 1. Παράδειγμα: 5 με παιδιά 7 (με παιδιά 2, 9) και 4 (φύλλο).
- **Εκφυλισμένο (degenerate binary tree)**: κάθε κόμβος έχει το πολύ ένα παιδί. Παράδειγμα: $5 \longrightarrow 7 \longrightarrow 2$. Στην πράξη συμπεριφέρεται σαν λίστα.

Η διάκριση μετρά για την πολυπλοκότητα: σε ένα τέλειο ή ισορροπημένο δέντρο με n κόμβους το βάθος είναι περίπου $\log_2 n$, ενώ σε ένα εκφυλισμένο είναι $n - 1$.

§21.9 N-αδικά δέντρα

Δεν είναι όλα τα δέντρα δυαδικά. Είναι συνηθισμένο να αναπαριστούμε **καταστάσεις** με δέντρα, κάποιες φορές με περισσότερα από 2 παιδιά ανά κόμβο. Η διάλεξη δείχνει ένα δέντρο παιχνιδιού τρίλιζας: κάθε κόμβος είναι μια κατάσταση του ταμπλό, κάθε παιδί μια πιθανή επόμενη κίνηση, και τα φύλλα είναι τελικές καταστάσεις με βαθμολογία (+10, 0, −10). Περισσότερα στη διάλεξη 22.

§21.10 Βασικές λειτουργίες με δυαδικά δέντρα

Οι λειτουργίες είναι `is_empty` (το δέντρο είναι `NULL`), `depth`, `print`, `find`, και `insert` και `delete`, που η διάλεξη αφήνει ως άσκηση. Σχεδόν όλες είναι **αναδρομικές**, γιατί και το δέντρο είναι αναδρομική δομή: ένα δέντρο είναι είτε άδειο είτε ένας κόμβος με δύο υποδέντρα. Η βάση της αναδρομής είναι το άδειο δέντρο (`t == NULL`), και το αναδρομικό βήμα καλεί τη συνάρτηση για τα `t->left` και `t->right` και συνδυάζει τα αποτελέσματα.

Για το **βάθος**: το άδειο δέντρο έχει βάθος −1, ώστε ένα φύλλο να βγαίνει $1 + \max(-1, -1) = 0$· κάθε άλλος κόμβος έχει βάθος 1 συν το μεγαλύτερο βάθος των δύο υποδέντρων του. Κάθε κόμβος επισκέπτεται μία φορά, άρα χρόνος $O(n)$. Ο χώρος είναι όσες κλήσεις είναι ταυτόχρονα στη στοίβα, δηλαδή όσο το βάθος: για ένα τέλειο δέντρο $O(\log n)$.

§21.11 Διάσχιση κατά βάθος (DFS)

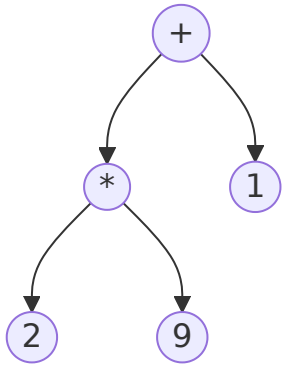
Η **αναζήτηση κατά βάθος (depth-first search, DFS)** είναι ένας αλγόριθμος διάσχισης και αναζήτησης σε δέντρα και γράφους. Ξεκινά από τον αρχικό κόμβο και εξερευνά όσο πιο βαθιά μπορεί σε έναν κλάδο πριν **οπισθοδρομήσει (backtracking)** για να δοκιμάσει τον επόμενο. Η αναδρομή το κάνει φυσικά: η στοίβα των κλήσεων θυμάται πού πρέπει να επιστρέψουμε.

Σε ένα δυαδικό δέντρο η DFS έχει τρεις παραλλαγές, ανάλογα με το πότε **επεξεργαζόμαστε** (π.χ. τυπώνουμε) τον τρέχοντα κόμβο σε σχέση με τα παιδιά του:

Διάσχιση	Σειρά	Δέντρο 5 (7 (2, 9), 1)
Pre-order	πρώτα ο τρέχων κόμβος, μετά τα παιδιά	`5 7 2 9 1`
In-order	πρώτα ο αριστερός, μετά ο τρέχων, τέλος ο δεξιός	`2 7 9 5 1`
Post-order	πρώτα τα παιδιά, στο τέλος ο τρέχων κόμβος	`2 9 7 1 5`

Οι τρεις κώδικες διαφέρουν μόνο στη θέση του `printf` ως προς τις δύο αναδρομικές κλήσεις. Όλες έχουν χρόνο $O(n)$ και χώρο ίσο με το βάθος, $O(\log n)$ για ισορροπημένο δέντρο.

Η σειρά έχει σημασία. Σε ένα **δέντρο έκφρασης**, όπως το `2 * 9 + 1`, οι τελεστές είναι εσωτερικοί κόμβοι και οι αριθμοί φύλλα:



Σχήμα: το δέντρο της έκφρασης `2 * 9 + 1`.

Ένας αποτιμητής εκφράσεων (calculator / evaluator / interpreter) χρειάζεται τις τιμές **και των δύο** υποδέντρων πριν εφαρμόσει τον τελεστή του κόμβου, άρα κάνει post-order διάσχιση. Η in-order διάσχιση αυτού του δέντρου δίνει την έκφραση στη συνηθισμένη της μορφή, `2 * 9 + 1`.

§21.12 Αναζήτηση σε δυαδικό δέντρο

Σε ένα τυχαίο δυαδικό δέντρο η τιμή που ψάχνουμε μπορεί να βρίσκεται οπουδήποτε. Η `find` της διάλεξης είναι μια pre-order DFS: ελέγχει τον τρέχοντα κόμβο, μετά ψάχνει στο αριστερό υποδέντρο, και μόνο αν δεν τη βρει εκεί ψάχνει στο δεξί. Στη χειρότερη περίπτωση επισκέπτεται όλους τους κόμβους: χρόνος $O(n)$, χώρος $O(\log n)$ για ισορροπημένο δέντρο.

§21.13 Διάσχιση κατά πλάτος (BFS)

Η **αναζήτηση κατά πλάτος ή κατά επίπεδα (breadth-first search, BFS)** είναι επίσης αλγόριθμος διάσχισης και αναζήτησης σε δέντρα και γράφους. Ξεκινά από τον αρχικό κόμβο και σε κάθε βήμα εξερευνά **όλους** τους κόμβους του τρέχοντος επιπέδου πριν περάσει στο επόμενο. Σε ένα τέλειο δέντρο με αριθμημένους κόμβους 1–7 η σειρά επίσκεψης είναι 1 | 2, 3 | 4, 5, 6, 7.

Η BFS δεν γράφεται φυσικά με αναδρομή. Χρειάζεται μια βοηθητική λίστα, το **μέτωπο (frontier)** ή λίστα εργασιών (worklist), με τους κόμβους που έχουμε δει αλλά δεν έχουμε επεξεργαστεί ακόμα:

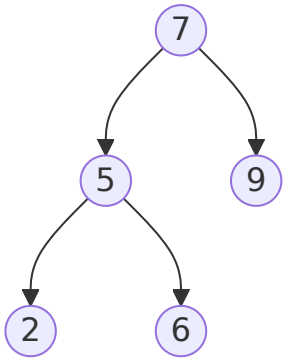
1. Βάζουμε τη ρίζα στο μέτωπο.
2. Όσο το μέτωπο δεν είναι άδειο, βγάζουμε τον κόμβο που μπήκε **νωρίτερα**, τον επεξεργαζόμαστε και βάζουμε στο μέτωπο τα παιδιά του.

Επειδή η `insert` βάζει στοιχεία στην κεφαλή, η διάλεξη βγάζει από την ουρά (`pop_last`): ο πρώτος που μπήκε βγαίνει πρώτος, οπότε όλο το επίπεδο k επεξεργάζεται πριν το επίπεδο $k+1$. Προσέξτε ότι η λίστα αυτή δεν κρατά πια ακεραίους αλλά δείκτες σε κόμβους του δέντρου

(`Tree`). Χρόνος $O(n)$· χώρος $O(n)$, γιατί το μέτωπο μπορεί να κρατά ένα ολόκληρο επίπεδο, και το τελευταίο επίπεδο ενός τέλειου δέντρου έχει περίπου τους μισούς κόμβους.

§21.14 Δυαδικό δέντρο αναζήτησης (BST)

Ένα **δυαδικό δέντρο αναζήτησης (binary search tree, BST ή ordered tree)** είναι ένα ταξινομημένο δέντρο: για κάθε κόμβο, όλοι οι κόμβοι στο αριστερό του υποδέντρο έχουν μικρότερη τιμή και όλοι οι κόμβοι στο δεξί του υποδέντρο μεγαλύτερη.



Σχήμα: δυαδικό δέντρο αναζήτησης· αριστερά του 7 τα 5, 2, 6, δεξιά το 9.

Η ιδιότητα αυτή επιτρέπει να ελέγξουμε αν υπάρχει μια τιμή (`exists`) χωρίς να επισκεφτούμε όλο το δέντρο: αν η τιμή είναι μικρότερη από του τρέχοντος κόμβου, μπορεί να βρίσκεται μόνο αριστερά, αλλιώς μόνο δεξιά. Κάθε βήμα κατεβαίνει ένα επίπεδο και απορρίπτει ένα ολόκληρο υποδέντρο, όπως η δυαδική αναζήτηση σε ταξινομημένο πίνακα. Σε ισορροπημένο δέντρο αυτό δίνει χρόνο $O(\log n)$ και χώρο $O(\log n)$ (για τις αναδρομικές κλήσεις). Σε εκφυλισμένο BST, π.χ. αν εισάγουμε τους αριθμούς ήδη ταξινομημένους, το βάθος είναι $n-1$ και η αναζήτηση γίνεται $O(n)$, όπως σε λίστα. Η in-order διάσχιση ενός BST τυπώνει τις τιμές σε αύξουσα σειρά.

§21.15 DFS ή BFS;

Κανένας από τους δύο δεν είναι γενικά καλύτερος· εξαρτάται από το πρόβλημα:

- Η **BFS** επισκέπτεται τους κόμβους κατά σειρά απόστασης από τη ρίζα, άρα η πρώτη λύση που βρίσκει είναι η πιο κοντινή. Είναι η φυσική επιλογή για το **συντομότερο μονοπάτι**, π.χ. την έξοδο από έναν λαβύρινθο. Το κόστος είναι ότι κρατά ολόκληρα επίπεδα στη μνήμη.
- Η **DFS** χρειάζεται μνήμη ανάλογη μόνο με το βάθος, και γράφεται απλά με αναδρομή. Όταν πρέπει να επισκεφτούμε **όλους** τους κόμβους, π.χ. για το μέγιστο στοιχείο ενός (μη ταξινομημένου) δέντρου, και οι δύο κάνουν $O(n)$ βήματα, άρα μετρά κυρίως η μνήμη.

Παραδείγματα

§21.16 Ένα πρώτο πρόγραμμα: `is_empty`

Εφαρμόζει τον ορισμό της λίστας και της κενής λίστας (Θεωρία: «Απλά συνδεδεμένη λίστα»). Ο κόμβος εδώ είναι τοπική μεταβλητή, δεν χρειάζεται ακόμα `malloc`:

```
#include <stdio.h>
typedef struct listnode {int value; struct listnode * next;} * List;
int is_empty(List list) {
  return list == NULL;
}
int main() {
  struct listnode node = {42, NULL};
  List list1 = &node;
  List list2 = NULL;
  printf("Is empty: %d\n", is_empty(list1));
  printf("Is empty: %d\n", is_empty(list2));
  return 0;
}
```

```
$ ./list
Is empty: 0
Is empty: 1
```

§21.17 Εισαγωγή και τύπωμα

Εφαρμόζει την «Εισαγωγή στην αρχή της λίστας» και τη διάσχιση. Η διάλεξη ρωτά τι θα τυπώσει το πρόγραμμα:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct listnode {int value; struct listnode * next;} * List;
void insert(List * list, int value) {
  List current_head = *list;
  List new_head = malloc(sizeof(struct listnode)); // νέος κόμβος στον σωρό
  new_head->value = value;                         // αρχικοποίηση κόμβου
  new_head->next = current_head;
  *list = new_head;                  // ο νέος κόμβος γίνεται η νέα κεφαλή
}
void print(List list) {
  printf("list: ");
  while(list) {
    printf(" -> %d", list->value);
    list = list->next;
```

```
  }
  printf(" -> NULL\n");
}
int main() {
  List list = NULL;
  insert(&list, 42); insert(&list, 43); insert(&list, 44);
  print(list);
  return 0;
}
```

```
$ ./insert
list:  -> 44 -> 43 -> 42 -> NULL
```

Το 44 μπήκε τελευταίο, άρα είναι η κεφαλή. Τα δύο κενά μετά το `list:` προέρχονται από το `"list: "` και το `" -> %d"`.

§21.18 Το μήκος, επαναληπτικά

Η επαναληπτική εκδοχή της `length` (η αναδρομική είναι στη Θεωρία):

```
int length(List list) {
  int counter = 0;
  while(list) {
    counter++;
    list = list->next;
  }
  return counter;
}
```

§21.19 Αναζήτηση: `find`

Εφαρμόζει την «Αναζήτηση και αφαίρεση στοιχείου». Υποθέτει την `insert` του προηγούμενου παραδείγματος:

```
// #include, typedef και insert όπως παραπάνω
....
List find(List list, int value) {
  while(list && list->value != value) {
    list = list->next;
  }
  return list;
}
int main() {
  List list = NULL;
  insert(&list, 42); insert(&list, 43); insert(&list, 44);
```

```
  printf("Found 43: %x\n", find(list, 43));
  printf("Found 34: %x\n", find(list, 34));
  return 0;
}
```

```
$ ./find
Found 43: 161862c0
Found 34: 0
```

Για το 43 τυπώνεται η διεύθυνση του κόμβου (διαφορετική σε κάθε εκτέλεση), για το 34 το `NULL`, δηλαδή 0. Το `%x` περιμένει `unsigned int`, οπότε ο `gcc -Wall` προειδοποιεί και σε 64-bit μηχάνημα τυπώνονται μόνο τα χαμηλά 32 bits της διεύθυνσης. Το σωστό είναι `printf("%p\n", (void *) find(list, 43));`.

§21.20 Αφαίρεση: `delete`

Η `delete` της διάλεξης, με τον δείκτη σε δείκτη που περιγράφει η Θεωρία:

```
void delete(List * list, int value) {
  List temp;
  while(*list && (*list)->value != value) {
    list = &((*list)->next);
  }
  if (*list) {
    temp = *list;
    *list = temp->next;
    free(temp);
  }
}
```

Στη λίστα 44 ⟶ 43 ⟶ 42, το `delete(&list, 43)` προχωρά μία φορά, ώστε ο `list` να δείχνει στο πεδίο `next` του 44, και μετά το αλλάζει ώστε να δείχνει στο 42. Το `delete(&list, 44)` δεν μπαίνει καν στον βρόχο και αλλάζει την ίδια τη μεταβλητή της `main`. Αν αφαιρέσετε το `free(temp)`, η λίστα είναι σωστή αλλά ο κόμβος μένει δεσμευμένος (διαρροή μνήμης, που το `valgrind` αναφέρει ως `definitely lost`).

§21.21 Βάθος δέντρου

Εφαρμόζει τις «Βασικές λειτουργίες με δυαδικά δέντρα». Το δέντρο 5 (7 (2, 9), 1) χτίζεται με τοπικές μεταβλητές και αρχικοποιητές δομών:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct treenode {
  int value; struct treenode * left; struct treenode * right;
} * Tree;
```

```c
int depth(Tree t) {
  if (t == NULL) return -1;
  int left_depth = depth(t->left);
  int right_depth = depth(t->right);
  return 1 + ((left_depth > right_depth) ? left_depth : right_depth);
}
int main() {
  struct treenode t2 = {2, NULL, NULL}, t9 = {9, NULL, NULL};
  struct treenode t1 = {1, NULL, NULL};
  struct treenode t7 = {7, &t2, &t9}, t5 = {5, &t7, &t1};
  Tree t = &t5;
  printf("Depth: %d\n", depth(t));
  return 0;
}
```

```
$ ./depth
Depth: 2
```

Η `is_empty` για δέντρο είναι ίδια με της λίστας (`return t == NULL;`) και με `Tree t = NULL;` τυπώνει `Empty: 1`.

§21.22 Οι τρεις διασχίσεις DFS

Εφαρμόζει τη «Διάσχιση κατά βάθος». Στο ίδιο δέντρο 5 (7 (2, 9), 1):

```c
void print(Tree t) {          // pre-order
  if (t == NULL) return;
  printf("%d ", t->value);
  print(t->left);
  print(t->right);
}
```

Η **in-order** εκδοχή μετακινεί το `printf` ανάμεσα στις δύο κλήσεις, και η **post-order** μετά από αυτές:

```
$ ./preorder
5 7 2 9 1
$ ./inorder
2 7 9 5 1
$ ./postorder
2 9 7 1 5
```

§21.23 Αναζήτηση σε δέντρο και σε BST

Η `find` για τυχαίο δυαδικό δέντρο ψάχνει και στα δύο υποδέντρα:

```
Tree find(Tree t, int value) {
  if (t == NULL) return NULL;
  if (t->value == value) return t;
  Tree left = find(t->left, value);
  if (left != NULL) return left;
  return find(t->right, value);
}
```

Η `exists` για BST ακολουθεί **ένα** μόνο μονοπάτι από τη ρίζα:

```
int exists(Tree t, int value) {
  if (t == NULL) return 0;
  if (t->value == value) return 1;
  if (value < t->value) return exists(t->left, value);
  return exists(t->right, value);
}
```

Στο BST 7 (5 (2, 6), 9), το `exists(t, 6)` επισκέπτεται 7 ⟶ 5 ⟶ 6 και επιστρέφει 1· το `exists(t, 8)` επισκέπτεται 7 ⟶ 9 ⟶ `NULL` και επιστρέφει 0.

§21.24 BFS με λίστα από κόμβους δέντρου

Εφαρμόζει τη «Διάσχιση κατά πλάτος». Ο κώδικας της διάλεξης:

```
void bfs(Tree t) {
  List frontier = NULL;
  Tree tmp;
  insert(&frontier, t);
  while(frontier) {
    tmp = pop_last(&frontier);
    printf("%d ", tmp->value);
    if (tmp->left) insert(&frontier, tmp->left);
    if (tmp->right) insert(&frontier, tmp->right);
  }
}
```

Η διάλεξη δεν δίνει την `pop_last` και ρωτά τι περιέχει πια ο τύπος `List`: όχι ακεραίους αλλά κόμβους δέντρου, άρα το πεδίο τιμής του κόμβου λίστας γίνεται `Tree value;` και η `insert` παίρνει `Tree`. Η `pop_last` πηγαίνει στον τελευταίο σύνδεσμο με την τεχνική της `delete`, τον αποσυνδέει και επιστρέφει την τιμή του:

```
Tree pop_last(List * list) {  // προϋπόθεση: *list != NULL
  while ((*list)->next)
    list = &((*list)->next);
  List last = *list;
  Tree value = last->value;
```

```
  *list = NULL;
  free(last);
  return value;
}
```

Με αυτές τις αλλαγές (και ένα `printf("\n")` στο τέλος), η `bfs` στο δέντρο 5 (7 (2, 9), 1) τυπώνει `5 7 1 2 9`.

Η `pop_last` διασχίζει όλη τη λίστα, οπότε αυτή η απλή εκδοχή κοστίζει $O(n)$ ανά αφαίρεση· ένας δείκτης και στην ουρά θα το έκανε $O(1)$. Για λίστες που δουλεύουν με **κάθε** τύπο δεδομένων, το πεδίο τιμής μπορεί να γίνει `void *`, όπως στους αφηρημένους τύπους δεδομένων.

§21.25 Για το εργαστήριο

Το Εργαστήριο 9 ζητά ακριβώς αυτά: στο `grades.c` μια λίστα βαθμών με `insert_at_start` και `average` (διάσχιση), και στο `tree.c` ένα ταξινομημένο δέντρο (BST) με την αναδρομική `addtree` και εκτύπωση in-order με την `treeprint`. Η `addtree` είναι η `insert` για δέντρα που η διάλεξη αφήνει «μόνοι σας»: ακολουθεί το ίδιο μονοπάτι με την `exists` και δημιουργεί νέο κόμβο εκεί όπου συναντά `NULL`. Το παράρτημα του εργαστηρίου δείχνει πώς το `valgrind` βρίσκει τη διαρροή μιας λίστας χωρίς `free`, και μια `free_list` που την αποδεσμεύει κόμβο-κόμβο.

Κύρια σημεία

1. Μια απλά συνδεδεμένη λίστα είναι αλυσίδα από κόμβους `{value, next}`· το τελευταίο `next` είναι `NULL`, η λίστα αναπαριστάται από δείκτη στην κεφαλή της, και η κενή λίστα (όπως και το κενό δέντρο) είναι απλώς `NULL`.
2. Η εισαγωγή στην κεφαλή κοστίζει $O(1)$ και αφήνει τα στοιχεία σε αντίστροφη σειρά εισαγωγής.
3. Όποια συνάρτηση αλλάζει την κεφαλή (`insert`, `delete`) παίρνει `List *` και γράφει στο `*list`· με σκέτο `List` αλλάζει μόνο ένα τοπικό αντίγραφο. Η `delete` παρακάμπτει τον κόμβο αλλάζοντας τον δείκτη που έδειχνε σε αυτόν, και τον αποδεσμεύει με `free`.
4. Οι `print`, `length` και `find` είναι διασχίσεις με `list = list->next`, χρόνου $O(n)$ και χώρου $O(1)$· η αναδρομική `length` χρειάζεται χώρο $O(n)$ στη στοίβα.
5. Οι πίνακες δίνουν πρόσβαση $O(1)$ και συνεχόμενη μνήμη· οι λίστες εύκολη αναδιάταξη και μέγεθος που δεν χρειάζεται να είναι γνωστό από πριν, με κόστος πρόσβασης $O(n)$ και έναν επιπλέον δείκτη ανά στοιχείο.
6. Στο δυαδικό δέντρο κάθε κόμβος έχει 0–2 παιδιά· ρίζα, φύλλα, βάθος/ύψος και επίπεδο περιγράφουν το σχήμα του.
7. Τα δέντρα διακρίνονται σε τέλεια, γεμάτα, πλήρη, ισορροπημένα και εκφυλισμένα· το βάθος ενός ισορροπημένου δέντρου είναι $O(\log n)$, ενός εκφυλισμένου $O(n)$.
8. Οι λειτουργίες σε δέντρα γράφονται φυσικά αναδρομικά, με βάση το `t == NULL`· ο χώρος τους είναι ανάλογος του βάθους.

9. Η DFS εξερευνά όσο πιο βαθιά γίνεται πριν οπισθοδρομήσει· οι παραλλαγές pre-order, in-order και post-order διαφέρουν μόνο στο πότε επεξεργάζονται τον τρέχοντα κόμβο.
10. Ένας αποτιμητής εκφράσεων χρειάζεται post-order διάσχιση, γιατί ένας τελεστής θέλει πρώτα τις τιμές των τελεστέων του.
11. Η BFS επισκέπτεται τους κόμβους επίπεδο-επίπεδο με τη βοήθεια μιας λίστας «πρώτος μέσα, πρώτος έξω»· κοστίζει $O(n)$ χρόνο και $O(n)$ χώρο.
12. Σε ένα BST τα μικρότερα είναι αριστερά και τα μεγαλύτερα δεξιά, οπότε η αναζήτηση ακολουθεί ένα μονοπάτι: $O(\log n)$ σε ισορροπημένο δέντρο.
13. Η BFS βρίσκει το συντομότερο μονοπάτι· η DFS χρειάζεται λιγότερη μνήμη. Η επιλογή εξαρτάται από το πρόβλημα.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
απλά συνδεδεμένη λίστα	single linked list	Κόμβοι όπου ο καθένας δείχνει στον επόμενο και ο τελευταίος στο `NULL`.
κεφαλή / ουρά	head / tail	Το πρώτο / (συνήθως) το τελευταίο στοιχείο της λίστας.
διάσχιση	traversal	Επίσκεψη όλων των κόμβων με μια συγκεκριμένη σειρά.
δυαδικό δέντρο	binary tree	Δέντρο όπου κάθε κόμβος έχει 0 έως 2 παιδιά.
ρίζα / φύλλο	root / leaf	Ο πρώτος κόμβος / κόμβος χωρίς παιδιά.
βάθος / ύψος	depth / height	Μέγιστος αριθμός συνδέσμων από τη ρίζα ως τα φύλλα / από τα φύλλα ως τη ρίζα.
επίπεδο κόμβου	node level	Πόσοι κόμβοι μεσολαβούν ως τη ρίζα· η ρίζα είναι στο 1.
τέλειο / γεμάτο / πλήρες δέντρο	perfect / full / complete binary tree	Βλ. «Τύποι δυαδικών δέντρων».
ισορροπημένο / εκφυλισμένο δέντρο	balanced / degenerate binary tree	Ύψη υποδέντρων που διαφέρουν ≤ 1 / κάθε κόμβος με ≤ 1 παιδί.
αναζήτηση κατά βάθος	depth-first search (DFS)	Εξερεύνηση όσο πιο βαθιά γίνεται, μετά οπισθοδρόμηση.
αναζήτηση κατά πλάτος	breadth-first search (BFS)	Εξερεύνηση επίπεδο-επίπεδο.
μέτωπο	frontier / worklist	Οι κόμβοι που περιμένουν επεξεργασία στη BFS.
δυαδικό δέντρο αναζήτησης	binary search tree (BST)	Δέντρο με μικρότερα αριστερά και μεγαλύτερα δεξιά σε κάθε κόμβο.

Ελληνικά	English	Σύντομος ορισμός
αφηρημένος τύπος δεδομένων	abstract data type (ADT)	Τύπος που ορίζεται από τις λειτουργίες του, όχι από την υλοποίηση.

Διάβασμα

- **Διαφάνειες**: Διάλεξη 21, σελ. 1–65: λίστες 5–25· πίνακες και λίστες 26· δυαδικά δέντρα και τύποι 27–37· λειτουργίες σε δέντρα 38–42· DFS και διασχίσεις 43–53· BFS 54–56· BST 57–59· DFS ή BFS 60–63.
- **Σημειώσεις**: η διάλεξη προτείνει τις σελ. 120–135 των διαφανειών του κ. Σταματόπουλου (K04):
  - Κεφάλαιο 7: «Αυτο-αναφορικές δομές» (K04, σελ. 120–125), «Δημιουργία νέων ονομάτων τύπων» (126), «Ενώσεις και πεδία bit» (127).
  - Κεφάλαιο 8: «Διαχείριση συνδεδεμένων λιστών» (K04, σελ. 128–131), «Διαχείριση δυαδικών δέντρων» (132–135).
- **Εργαστήριο**: Εργαστήριο 9: ασκήσεις `grades.c` (λίστα), `tree.c` (BST, in-order) και το παράρτημα για το `valgrind` (διαρροή σε λίστα, `free_list`).
- **Άλλα**: Linked list, Binary Tree Problems (Stanford CS Education Library), Tree traversal, Depth-first search, Breadth-first search, Binary search tree και η οπτικοποίησή του, Αφηρημένοι Τύποι Δεδομένων.

Συχνά λάθη

- `List` **αντί για** `List *` **στην** `insert`. Η συνάρτηση γράφει `list = new_head` στο τοπικό της αντίγραφο και η λίστα της `main` μένει `NULL`. Περάστε `&list` και γράψτε `*list = new_head`. Ούτε το `list = &new_head` βοηθά: αλλάζει πάλι μόνο την τοπική μεταβλητή.
- **Πρόσβαση μέσω** `NULL`. Το `list->value` πριν ελέγξετε ότι `list != NULL` (ή `t->left` σε κενό δέντρο) δίνει `Segmentation fault`. Ελέγχετε πρώτα, όπως το `while(list && list->value != value)`, όπου η σειρά των όρων μετράει. Κάθε αναδρομική συνάρτηση δέντρου ξεκινά με βάση `if (t == NULL) ...`.
- `free` **πριν διαβάσετε το** `next`. Το `free(list); list = list->next;` διαβάζει αποδεσμευμένη μνήμη (`Invalid read` στο `valgrind`). Κρατήστε πρώτα το `next` σε προσωρινή μεταβλητή, όπως κάνει η `delete` με το `temp`.
- **Ξεχασμένο** `free`. Κόμβοι που αφαιρούνται από τη λίστα ή λίστες που δεν χρειάζονται πια πρέπει να αποδεσμεύονται· αλλιώς το `valgrind` αναφέρει `definitely lost`.
- `%x` **ή** `%d` **για δείκτες**. `format '%x' expects argument of type 'unsigned int'`: τυπώνετε δείκτες με `%p` και cast σε `void *`.
- **Μπέρδεμα των διασχίσεων**. Η in-order τυπώνει ταξινομημένα μόνο σε BST· σε τυχαίο δέντρο (5 (7 (2, 9), 1)) δίνει `2 7 9 5 1`.

- **«Το BST είναι πάντα $O(\log n)$».** Μόνο αν είναι ισορροπημένο· με ταξινομημένη είσοδο γίνεται εκφυλισμένο και η αναζήτηση $O(n)$.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K21.3** Εισαγωγή στη μέση λίστας (39% σωστές): Το 26% επέλεξε O(1), μετρώντας μόνο την αλλαγή των δύο δεικτών και ξεχνώντας ότι πρώτα πρέπει να φτάσουμε στη μέση διατρέχοντας τη λίστα από την αρχή (άλλο ένα 32% διάλεξε το αστείο O(f(x))).

Ερωτήσεις κατανόησης

- **E21.1** Γιατί η `insert` παίρνει `List *` ενώ η `print` αρκείται σε `List`;[164]
- **E21.2** Ποια είναι η πολυπλοκότητα χρόνου της πρόσβασης στο i-οστό στοιχείο σε πίνακα και σε λίστα, και γιατί;[165]
- **E21.3** Τι επιστρέφει η `depth` για ένα δέντρο με έναν μόνο κόμβο, και γιατί η βάση της επιστρέφει −1;[166]
- **E21.4** Πόσους κόμβους έχει ένα τέλειο δυαδικό δέντρο με 4 επίπεδα;[167]
- **E21.5** Τι τυπώνουν οι pre-order, in-order και post-order διασχίσεις του BST 7 (5 (2, 6), 9);[168]
- **E21.6** Γιατί η BFS χρειάζεται βοηθητική λίστα ενώ η DFS όχι;[169]
- **E21.7** Πότε η αναζήτηση σε BST παύει να είναι $O(\log n)$;[170]

Kahoot από το αμφιθέατρο (K21.1–K21.3)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K21.1** Ανάγνωση διαγραμμένου στοιχείου: 92% σωστές απαντήσεις
- **K21.2** Εργαλείο για memory leaks: 86% σωστές απαντήσεις
- **K21.3** Εισαγωγή στη μέση λίστας: 39% σωστές απαντήσεις

[164] Η `insert` αλλάζει τη μεταβλητή-κεφαλή του καλούντος, άρα χρειάζεται τη διεύθυνσή της· η `print` μόνο διαβάζει, και της αρκεί αντίγραφο του δείκτη.

[165] $O(1)$ στον πίνακα, γιατί η διεύθυνση υπολογίζεται από τη θέση· $O(n)$ στη λίστα, γιατί πρέπει να ακολουθήσουμε i δείκτες `next` από την κεφαλή.

[166] 0: $1 + \max(-1, -1)$. Με −1 για το κενό δέντρο, το βάθος μετρά συνδέσμους, όπως ο ορισμός.

[167] $1 + 2 + 4 + 8 = 15 = 2^4 - 1$.

[168] Pre-order `7 5 2 6 9`, in-order `2 5 6 7 9` (ταξινομημένα), post-order `2 6 5 9 7`.

[169] Η DFS επιστρέφει στους κόμβους που άφησε μέσω της στοίβας των αναδρομικών κλήσεων· η BFS πρέπει να θυμάται όλους τους κόμβους του επόμενου επιπέδου, και τους κρατά σε λίστα «πρώτος μέσα, πρώτος έξω».

[170] Όταν το δέντρο δεν είναι ισορροπημένο· στη χειρότερη περίπτωση (εκφυλισμένο, π.χ. από ταξινομημένη είσοδο) γίνεται $O(n)$.

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A21.1–A21.6)

- **A21.1** Προσθήκη στοιχείου σε λίστα: Διάλεξη 21, διαφάνειες 11–12 · ★☆☆ · short-answer · `slides-lec21-insert-how`
- **A21.2** Τι τυπώνει η εισαγωγή σε λίστα;: Διάλεξη 21, διαφάνειες 13–16 · ★☆☆ · trace · `slides-lec21-insert-output`
- **A21.3** Αναδρομικό μήκος λίστας: Διάλεξη 21, διαφάνεια 18 · ★☆☆ · programming · `slides-lec21-recursive-length`
- **A21.4** Αφαίρεση στοιχείου από λίστα: Διάλεξη 21, διαφάνειες 23–25 · ★★☆ · short-answer · `slides-lec21-delete-how`
- **A21.5** Τι τυπώνει η find σε λίστα;: Διάλεξη 21, διαφάνειες 21–22 · ★★☆ · trace · `slides-lec21-find-output`
- **A21.6** Πολυπλοκότητα του μήκους λίστας: Διάλεξη 21, διαφάνειες 19–20 · ★★☆ · short-answer · `slides-lec21-length-complexity`

Εργαστήριο (A21.7)

- **A21.7** Συνδεδεμένες λίστες: Εργαστήριο 9, Άσκηση 3 · ★★☆ · programming · `lab-lab09-grades`

Θέματα εξετάσεων (A21.8–A21.10)

- **A21.8** Αντιστροφή λίστας: Εξέταση Σεπτεμβρίου 2024, Θέμα 5 · ★★☆ · programming · `exam-2024-sep-q5`
- **A21.9** Μεσαίο Στοιχείο Λίστας: Εξέταση Σεπτεμβρίου 2025, Θέμα 4 · ★★☆ · programming · `exam-2025-sep-q4`
- **A21.10** N-οστό Στοιχείο Λίστας: Εξέταση Σεπτεμβρίου 2026, Θέμα 4 · ★★☆ · programming · `exam-2026-sep-q4`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A13.10** Debugging: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #1 (Coreutils Themed), Θέμα 4 · ★★☆ · debug · `exam-2023-fall-ex1-q4`
- **A20.7** Η διάταξη των φακέλων: Διάλεξη 20, διαφάνεια 38 · ★☆☆ · short-answer · `slides-lec20-list-layout`
- **A22.18** Αθροιστής Δέντρων - sumtree: Εξέταση Ιανουαρίου 2025, Θέμα 4 · ★☆☆ · programming · `exam-2025-jan-q4`
- **A22.14** Δυαδικά δένδρα: Εργαστήριο 9, Άσκηση 4 · ★★☆ · programming · `lab-lab09-tree`
- **A22.1** Ποιος αλγόριθμος αναζήτησης είναι καλύτερος;: Διαλέξεις 21–22: Δέντρα, διαφάνεια 38 (διάλεξη 21: διαφάνεια 60) · ★☆☆ · short-answer · `slides-lec22-best-search`

- **A22.2** Έλεγχος ύπαρξης σε δυαδικό δέντρο αναζήτησης: Διαλέξεις 21–22: Δέντρα, διαφάνειες 35–37 (διάλεξη 21: διαφάνειες 57–59) · ★☆☆ · programming · `slides-lec22-bst-exists`
- **A22.6** Πολυπλοκότητα της depth: Διαλέξεις 21–22: Δέντρα, διαφάνεια 20 (διάλεξη 21: διαφάνειες 41–42) · ★★☆ · short-answer · `slides-lec22-depth-complexity`
- **A22.7** Διάσχιση για αποτιμητή εκφράσεων: Διαλέξεις 21–22: Δέντρα, διαφάνεια 29 (διάλεξη 21: διαφάνεια 51) · ★★☆ · short-answer · `slides-lec22-expression-evaluator`
- **A22.8** Λίστα για το BFS και λίστες κάθε τύπου: Διαλέξεις 21–22: Δέντρα, διαφάνεια 33 (διάλεξη 21: διαφάνεια 55) · ★★☆ · short-answer · `slides-lec22-generic-list`
- **A22.9** Συντομότερο μονοπάτι σε δέντρα-λαβύρινθους: BFS ή DFS;: Διαλέξεις 21–22: Δέντρα, διαφάνεια 40 (διάλεξη 21: διαφάνεια 62) · ★★☆ · short-answer · `slides-lec22-maze-shortest-path`
- **A22.3** Κόμβοι τέλειου δυαδικού δέντρου: Διαλέξεις 21–22: Δέντρα, διαφάνεια 10 (διάλεξη 21: διαφάνεια 32) · ★☆☆ · short-answer · `slides-lec22-perfect-tree-nodes`
- **A22.10** Αποθήκευση και αναζήτηση σε 1 PetaByte: Διαλέξεις 21–22: Δέντρα, διαφάνεια 39 (διάλεξη 21: διαφάνεια 61) · ★★☆ · short-answer · `slides-lec22-petabyte`
- **A22.4** Διασχίσεις pre-order, in-order, post-order: Διαλέξεις 21–22: Δέντρα, διαφάνειες 22–28 (διάλεξη 21: διαφάνειες 44–50) · ★☆☆ · trace · `slides-lec22-traversals`
- **A22.13** Αφαίρεση στοιχείου από δυαδικό δέντρο: Διαλέξεις 21–22: Δέντρα, διαφάνεια 16 (διάλεξη 21: διαφάνεια 38) · ★★★ · programming · `slides-lec22-tree-delete`
- **A22.12** Προσθήκη στοιχείου σε δυαδικό δέντρο: Διαλέξεις 21–22: Δέντρα, διαφάνεια 16 (διάλεξη 21: διαφάνεια 38) · ★★☆ · programming · `slides-lec22-tree-insert`
- **A22.5** Μέγιστο στοιχείο δέντρου: BFS ή DFS;: Διαλέξεις 21–22: Δέντρα, διαφάνεια 41 (διάλεξη 21: διαφάνεια 63) · ★☆☆ · short-answer · `slides-lec22-tree-max`
- **A25.1** Καθαρή διαχείριση μνήμης: Εργαστήριο 9, Άσκηση 5 · ★★☆ · programming · `lab-lab09-grades-tree`

Κεφάλαιο 22: Δέντρα

> **Στόχοι**: μετά από αυτό το κεφάλαιο θα μπορείτε να ορίζετε σε C έναν κόμβο δυαδικού δέντρου· να εξηγείτε ρίζα, φύλλα, βάθος, ύψος και επίπεδο· να αναγνωρίζετε τέλεια, γεμάτα, πλήρη, ισορροπημένα και εκφυλισμένα δέντρα· να γράφετε αναδρομικά τις `is_empty`, `depth`, `print` (pre-order, in-order, post-order) και `find`· να τυπώνετε ένα δέντρο κατά πλάτος (BFS) με μια λίστα· να αναζητάτε σε δυαδικό δέντρο αναζήτησης· και να διαλέγετε ανάμεσα σε DFS και BFS.
>
> **Προαπαιτούμενα**: Κεφάλαιο 11 (αναδρομή), Κεφάλαιο 19, Κεφάλαιο 21 (λίστες)
>
> **Χρόνος μελέτης**: ~2,5 ώρες

Σύνοψη

Η διάλεξη συνεχίζει τις αυτοαναφορικές δομές της προηγούμενης φοράς: από τη λίστα, όπου κάθε κόμβος έχει έναν επόμενο, περνάμε στο δυαδικό δέντρο, όπου κάθε κόμβος έχει έως δύο παιδιά. Ορίζει την ορολογία (ρίζα, φύλλα, βάθος, ύψος, επίπεδο) και τα είδη των δυαδικών δέντρων, και υλοποιεί τις βασικές λειτουργίες αναδρομικά, με την πολυπλοκότητα χρόνου και χώρου της καθεμιάς. Γνωρίζουμε τους δύο τρόπους να εξερευνήσουμε ένα δέντρο, κατά βάθος (DFS) και κατά πλάτος (BFS), και το δυαδικό δέντρο αναζήτησης, όπου η διάταξη των τιμών κάνει την αναζήτηση λογαριθμική. Τα δέντρα είναι παντού: βάσεις δεδομένων, μεταγλωττιστές, συμπίεση, κρυπτογραφία, παιχνίδια.

Θεωρία

§22.1 Το δυαδικό δέντρο

Το **δυαδικό δέντρο (binary tree)** είναι ένας τύπος δεδομένων που οργανώνει τα δεδομένα σε δενδρική διάταξη: κάθε **κόμβος (node)** έχει από 0 έως 2 **κόμβους-παιδιά (children)**, το αριστερό και το δεξί. Όπως ο κόμβος λίστας (Κεφάλαιο 21), είναι μια **αυτοαναφορική δομή (self-referential structure)**, μόνο που έχει δύο δείκτες αντί για έναν. Τα προγράμματα της διάλεξης ορίζουν μαζί με τη δομή και έναν συνώνυμο τύπο `Tree` για τον δείκτη σε κόμβο, όπως το `List` της προηγούμενης διάλεξης:

```
typedef struct treenode {
  int value;
  struct treenode *left;
  struct treenode *right;
```

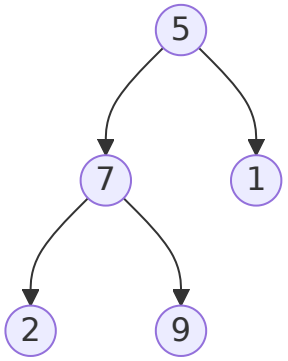
```
} *Tree;
```

Ένα `Tree` είναι δείκτης στη ρίζα· ένα παιδί που λείπει, όπως και το **άδειο δέντρο**, είναι `NULL`. Κάθε παιδί είναι κι αυτό ένα `Tree`, δηλαδή η ρίζα ενός **υποδέντρου (subtree)**. Αυτός ο αναδρομικός ορισμός (ένα δέντρο είναι είτε άδειο είτε ένας κόμβος με δύο υποδέντρα) είναι ο λόγος που σχεδόν όλοι οι αλγόριθμοι σε δέντρα γράφονται φυσικά με αναδρομή (Κεφάλαιο 11).

§22.2 Ρίζα, φύλλα, βάθος, ύψος, επίπεδο

- **Ρίζα (root)** είναι ο πρώτος κόμβος, αυτός από τον οποίο ξεκινάμε.
- **Φύλλα (leaves)** είναι οι κόμβοι χωρίς παιδιά (και οι δύο δείκτες `NULL`).
- **Βάθος (depth)** του δέντρου είναι ο μέγιστος αριθμός συνδέσμων από τη ρίζα μέχρι τα φύλλα.
- **Ύψος (height)** του δέντρου είναι ο μέγιστος αριθμός συνδέσμων από τα φύλλα μέχρι τη ρίζα. Για ολόκληρο το δέντρο βάθος και ύψος συμπίπτουν.
- **Επίπεδο (level)** ενός κόμβου είναι η «γραμμή» όπου βρίσκεται: η ρίζα είναι στο επίπεδο 1, τα παιδιά της στο 2, τα εγγόνια στο 3 κ.ο.κ. Με άλλα λόγια, το επίπεδο είναι το πλήθος των κόμβων στη διαδρομή από τη ρίζα μέχρι τον κόμβο (μαζί με τα δύο άκρα).

Το δέντρο που χρησιμοποιεί η διάλεξη σε όλα τα παραδείγματα:



Σχήμα: ρίζα το 5 (επίπεδο 1), 7 και 1 στο επίπεδο 2, 2 και 9 στο επίπεδο 3· φύλλα τα 1, 2, 9· βάθος 2 (δύο σύνδεσμοι από το 5 στο 2).

Προσέξτε ότι το βάθος μετρά **συνδέσμους** ενώ το επίπεδο μετρά **κόμβους**: ένα δέντρο με 3 επίπεδα έχει βάθος 2.

§22.3 Είδη δυαδικών δέντρων

- **Τέλειο (perfect binary tree)**: όλοι οι εσωτερικοί κόμβοι έχουν δύο παιδιά και όλα τα φύλλα βρίσκονται στο ίδιο επίπεδο. Κάθε επίπεδο είναι γεμάτο: το επίπεδο k έχει 2^{k-1} κόμβους.

- **Γεμάτο (full binary tree)**: κάθε κόμβος έχει 0 ή 2 παιδιά (ποτέ ένα). Τα φύλλα δεν χρειάζεται να είναι στο ίδιο επίπεδο.
- **Πλήρες (complete binary tree)**: κάθε επίπεδο, εκτός ίσως από το τελευταίο, είναι γεμάτο, και οι κόμβοι του τελευταίου επιπέδου είναι όσο πιο αριστερά γίνεται.
- **Ισορροπημένο (balanced binary tree)**: σε κάθε κόμβο, τα ύψη του αριστερού και του δεξιού υποδέντρου διαφέρουν το πολύ κατά 1.
- **Εκφυλισμένο (degenerate binary tree)**: κάθε κόμβος έχει το πολύ ένα παιδί. Στην ουσία είναι μια λίστα.

Τα παραδείγματα των διαφανειών: αν από το τέλειο δέντρο αφαιρέσουμε το 3, μένει **πλήρες** (τα 2, 9, 8 είναι αριστερά στο τελευταίο επίπεδο). Αν κρατήσουμε μόνο 5, 7, 4, 8, 3 (το 7 φύλλο, το 4 με δύο παιδιά), είναι **γεμάτο** αλλά όχι πλήρες. Το δέντρο 5, 7, 4, 2, 9 (τα 2 και 9 παιδιά του 7) είναι **ισορροπημένο**: στη ρίζα τα ύψη είναι 1 και 0. Τα είδη αυτά έχουν σημασία επειδή η πολυπλοκότητα των λειτουργιών εξαρτάται από το ύψος: ένα ισορροπημένο δέντρο με n κόμβους έχει ύψος περίπου $\log_2 n$, ενώ ένα εκφυλισμένο έχει ύψος $n - 1$.

§22.4 N-αδικά δέντρα και δέντρα καταστάσεων

Δεν είναι όλα τα δέντρα δυαδικά. Συχνά αναπαριστούμε **καταστάσεις (states)** ενός προβλήματος με ένα δέντρο όπου κάθε κόμβος έχει **περισσότερα από 2 παιδιά** (**N-αδικό δέντρο**, **N-ary tree**). Το παράδειγμα της διάλεξης είναι το δέντρο καταστάσεων της τρίλιζας (tic-tac-toe): η ρίζα είναι μια θέση του παιχνιδιού, κάθε παιδί είναι η θέση μετά από μία δυνατή κίνηση, και τα φύλλα, όπου το παιχνίδι τελειώνει, παίρνουν μια βαθμολογία (+10 νίκη, -10 ήττα, 0 ισοπαλία). Τέτοια δέντρα είναι η βάση των αλγορίθμων για παιχνίδια· η διάλεξη ρωτά αν το δέντρο της διαφάνειας είναι σωστό και πώς θα βρίσκαμε αυτόματα τα λάθη του.

§22.5 Βασικές λειτουργίες και πολυπλοκότητα

Οι βασικές λειτουργίες ενός δυαδικού δέντρου είναι:

1. `is_empty`: έλεγχος αν το δέντρο είναι άδειο.
2. `depth`: εύρεση του βάθους.
3. `print`: τύπωμα των στοιχείων (διάσχιση).
4. `find`: εύρεση στοιχείου.
5. `insert`: προσθήκη στοιχείου (για εξάσκηση, μόνοι σας).
6. `delete`: αφαίρεση στοιχείου (για εξάσκηση, μόνοι σας).

Το `is_empty` είναι απλώς `return t == NULL;`. Οι υπόλοιπες ακολουθούν το ίδιο αναδρομικό σχήμα: **βασική περίπτωση** το άδειο δέντρο (`t == NULL`), και **αναδρομικό βήμα** η επεξεργασία του κόμβου και των δύο υποδέντρων.

Για κάθε λειτουργία η διάλεξη δίνει **πολυπλοκότητα χρόνου** και **χώρου** (Κεφάλαιο 15) για ένα τέλειο δέντρο με n κόμβους. Όταν μια λειτουργία επισκέπτεται κάθε κόμβο μία φορά, ο χρόνος είναι $O(n)$. Ο χώρος μιας αναδρομικής λειτουργίας είναι το μέγιστο πλήθος κλήσεων που είναι

ταυτόχρονα στη στοίβα, δηλαδή το ύψος του δέντρου συν ένα: σε τέλειο δέντρο αυτό είναι $O(\log n)$. Σε εκφυλισμένο δέντρο το ύψος είναι $n - 1$, άρα ο χώρος γίνεται $O(n)$.

§22.6 Βάθος με αναδρομή

Το βάθος ενός δέντρου είναι 1 συν το μεγαλύτερο από τα βάθη των δύο υποδέντρων. Η διάλεξη ορίζει το βάθος του άδειου δέντρου ως -1, ώστε ένας μόνος κόμβος (φύλλο) να έχει βάθος $1 + (-1) = 0$, σύμφωνα με τον ορισμό «σύνδεσμοι από τη ρίζα στα φύλλα». Η υλοποίηση βρίσκεται στο παράδειγμα «Βάθος του δέντρου 5, 7, 1, 2, 9».

Η `treedepth` των σημειώσεων επιστρέφει 0 για το άδειο δέντρο, άρα μετρά **επίπεδα** (κόμβους) αντί για συνδέσμους και δίνει πάντα ένα παραπάνω. Και οι δύο είναι σωστές για τον δικό τους ορισμό· προσέξτε ποιον ζητά η εκφώνηση. Χρόνος $O(n)$, χώρος $O(\log n)$ για τέλειο δέντρο.

§22.7 Διάσχιση κατά βάθος (DFS)

Η **αναζήτηση κατά βάθος (depth-first search, DFS)** είναι ένας αλγόριθμος διάσχισης/αναζήτησης σε δέντρα και γράφους: ξεκινά από τον αρχικό κόμβο και προχωρά όσο πιο βαθιά μπορεί σε έναν κλάδο, πριν **οπισθοδρομήσει (backtracking)** για να δοκιμάσει τον επόμενο. Η αναδρομή υλοποιεί το DFS «δωρεάν»: η κλήση για το αριστερό υποδέντρο τελειώνει ολόκληρη πριν αρχίσει η κλήση για το δεξί, και η επιστροφή από την κλήση είναι η οπισθοδρόμηση.

Διάσχιση (traversal) είναι η επίσκεψη όλων των κόμβων με μια συγκεκριμένη σειρά. Σε δυαδικό δέντρο, το DFS έχει τρεις παραλλαγές, ανάλογα με το πότε επεξεργαζόμαστε (π.χ. τυπώνουμε) τον τρέχοντα κόμβο σε σχέση με τα παιδιά του:

Διάσχιση	Σειρά	Για το δέντρο 5, 7, 1, 2, 9
pre-order (προδιατεταγμένη)	κόμβος, αριστερό, δεξί	`5 7 2 9 1`
in-order (ενδοδιατεταγμένη)	αριστερό, κόμβος, δεξί	`2 7 9 5 1`
post-order (μεταδιατεταγμένη)	αριστερό, δεξί, κόμβος	`2 9 7 1 5`

Ο κώδικας είναι ο ίδιος και στις τρεις· αλλάζει μόνο η θέση του `printf` σε σχέση με τις δύο αναδρομικές κλήσεις. Χρόνος $O(n)$, χώρος $O(\log n)$ για τέλειο δέντρο.

Κάθε διάσχιση ταιριάζει σε άλλες δουλειές. Η post-order επεξεργάζεται τα παιδιά πριν από τον γονιό: ταιριάζει όταν ο κόμβος χρειάζεται τα αποτελέσματα των υποδέντρων του (αποτίμηση παράστασης, αποδέσμευση δέντρου με `free`). Η in-order σε δυαδικό δέντρο αναζήτησης δίνει τις τιμές ταξινομημένες. Η pre-order βλέπει τη ρίζα πρώτη (π.χ. για αντιγραφή ενός δέντρου).

§22.8 Αναζήτηση στοιχείου σε δυαδικό δέντρο

Σε ένα τυχαίο δυαδικό δέντρο οι τιμές δεν έχουν καμία διάταξη, άρα η `find` πρέπει να ψάξει παντού με DFS: ελέγχει τον τρέχοντα κόμβο, μετά ψάχνει στο αριστερό υποδέντρο, και μόνο αν δεν το βρει εκεί ψάχνει στο δεξί:

```c
Tree find(Tree t, int value) {
  if (t == NULL) return NULL;
  if (t->value == value) return t;
  Tree left = find(t->left, value);
  if (left != NULL) return left;
  return find(t->right, value);
}
```

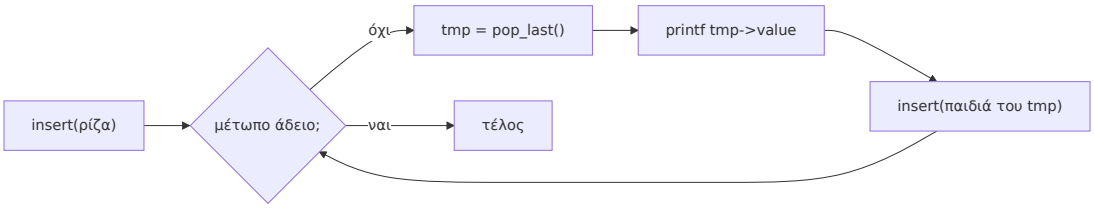
Επιστρέφει δείκτη στον κόμβο (ή `NULL`), ώστε ο καλών να μπορεί και να τον αλλάξει. Στη χειρότερη περίπτωση (η τιμή λείπει) επισκέπτεται όλους τους κόμβους: χρόνος $O(n)$, χώρος $O(\log n)$ για τέλειο δέντρο.

§22.9 Διάσχιση κατά πλάτος (BFS)

Η **αναζήτηση κατά πλάτος ή κατά επίπεδα (breadth-first search, BFS)** ξεκινά από τον αρχικό κόμβο και σε κάθε βήμα εξερευνά **όλους** τους κόμβους του τρέχοντος επιπέδου πριν περάσει στο επόμενο. Σε ένα τέλειο δέντρο με αριθμημένους κόμβους 1–7 επισκέπτεται 1· 2, 3· 4, 5, 6, 7.

Η αναδρομή δεν βοηθά εδώ, γιατί δεν θέλουμε να κατεβούμε σε έναν κλάδο μέχρι το τέλος. Χρειαζόμαστε μια δομή που θυμάται ποιους κόμβους έχουμε δει αλλά δεν έχουμε επεξεργαστεί ακόμα: το **μέτωπο (frontier)**, ή **λίστα εργασιών (worklist)**. Η διάλεξη χρησιμοποιεί μια λίστα από το Κεφάλαιο 21: προσθέτουμε κόμβους στην αρχή με `insert` και βγάζουμε από το τέλος με `pop_last`. Έτσι ο κόμβος που μπήκε πρώτος βγαίνει πρώτος: η λίστα λειτουργεί ως **ουρά (queue, FIFO)**. Ο αλγόριθμος:

1. Βάλε τη ρίζα στο μέτωπο.
2. Όσο το μέτωπο δεν είναι άδειο: βγάλε τον παλαιότερο κόμβο, επεξεργάσου τον (π.χ. τύπωσέ τον) και βάλε στο μέτωπο τα παιδιά του που υπάρχουν.



Σχήμα: ο βρόχος του BFS με λίστα εργασιών.

Στο δέντρο 5, 7, 1, 2, 9 το BFS τυπώνει `5 7 1 2 9`. Κάθε κόμβος μπαίνει και βγαίνει μία φορά, άρα ο χρόνος είναι $O(n)$. Ο χώρος όμως είναι $O(n)$: σε ένα τέλειο δέντρο το τελευταίο επίπεδο έχει περίπου τους μισούς κόμβους, και κάποια στιγμή βρίσκονται όλοι μαζί στο μέτωπο.

Εδώ η λίστα δεν κρατά πια ακεραίους αλλά **δείκτες σε κόμβους δέντρου** (`Tree`). Αυτό σημαίνει ότι ο κόμβος λίστας του προηγούμενου κεφαλαίου πρέπει να αλλάξει (πεδίο `Tree value` αντί για `int value`). Η διάλεξη ρωτά πώς θα γράφαμε λίστες που δουλεύουν με **κάθε** τύπο δεδομένων·

εκεί οδηγεί η ιδέα του **αφηρημένου τύπου δεδομένων (abstract data type, ΑΤΔ)**, που η διάλεξη δίνει για διάβασμα.

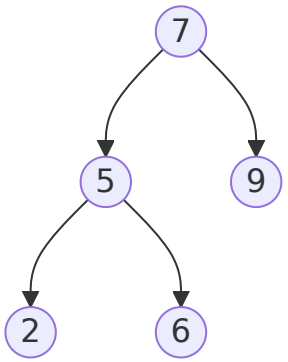
§22.10 DFS ή BFS;

Κανένας από τους δύο δεν είναι «καλύτερος» γενικά· η επιλογή εξαρτάται από το πρόβλημα:

- Το **BFS** επισκέπτεται τους κόμβους με σειρά απόστασης από την αρχή. Ο πρώτος κόμβος-στόχος που θα βρει είναι ο πιο κοντινός, άρα είναι η φυσική επιλογή για **συντομότερο μονοπάτι**. Θέλει όμως μνήμη ανάλογη με το πλάτος του δέντρου.
- Το **DFS** θέλει μνήμη ανάλογη με το ύψος (τη στοίβα της αναδρομής), που σε ένα ισορροπημένο δέντρο είναι πολύ μικρότερη. Όταν πρέπει να δούμε **όλους** τους κόμβους ούτως ή άλλως (π.χ. μέγιστο στοιχείο), και οι δύο κάνουν $O(n)$ βήματα, και το DFS είναι πιο απλό και πιο φειδωλό σε μνήμη.

§22.11 Δυαδικό δέντρο αναζήτησης (BST)

Ένα **δυαδικό δέντρο αναζήτησης (binary search tree, BST)** ή **ταξινομημένο δέντρο (ordered tree)** είναι ένα δυαδικό δέντρο όπου, για **κάθε** κόμβο, όλοι οι κόμβοι στο αριστερό του υποδέντρο έχουν μικρότερη τιμή και όλοι οι κόμβοι στο δεξί του υποδέντρο μεγαλύτερη.



Σχήμα: BST· αριστερά του 7 μόνο τα 2, 5, 6, δεξιά μόνο το 9.

Η διάταξη επιτρέπει να ψάχνουμε σαν τη δυαδική αναζήτηση σε ταξινομημένο πίνακα (Κεφάλαιο 17): σε κάθε κόμβο συγκρίνουμε και συνεχίζουμε **μόνο** σε ένα από τα δύο υποδέντρα, πετώντας το άλλο. Η `exists` της διάλεξης:

```
int exists(Tree t, int value) {
  if (t == NULL) return 0;
  if (t->value == value) return 1;
  if (value < t->value) return exists(t->left, value);
  return exists(t->right, value);
}
```

Κάνει μία κλήση ανά επίπεδο, άρα σε ισορροπημένο (π.χ. τέλειο) BST χρόνος και χώρος είναι $O(\log n)$, αντί για $O(n)$ χρόνο της `find`. Αν όμως το BST είναι εκφυλισμένο (π.χ. αν εισαγάγουμε τις τιμές ήδη ταξινομημένες, οπότε κάθε νέα τιμή πάει δεξιά), το ύψος είναι $n-1$ και η αναζήτηση γίνεται $O(n)$, όπως σε λίστα. Επίσης, η in-order διάσχιση ενός BST τυπώνει τις τιμές σε αύξουσα σειρά.

Η εισαγωγή σε BST (η `addtree` των σημειώσεων και του Εργαστηρίου 9) ακολουθεί την ίδια διαδρομή με την αναζήτηση και, όταν φτάσει σε `NULL`, φτιάχνει εκεί νέο κόμβο με `malloc`.

Παραδείγματα

§22.12 Άδειο δέντρο: `is_empty`

Το πρώτο πρόγραμμα της διάλεξης (σελ. 17) ορίζει τον τύπο `Tree` όπως στο «Το δυαδικό δέντρο», μαζί με τη `is_empty`, και τη δοκιμάζει σε άδειο δέντρο:

```
Tree t = NULL;
printf("Empty: %d\n", is_empty(t));
```

```
$ ./tree
Empty: 1
```

§22.13 Βάθος του δέντρου 5, 7, 1, 2, 9

Εφαρμόζει «Βάθος με αναδρομή». Η διάλεξη χτίζει το δέντρο χωρίς `malloc`, με τοπικές μεταβλητές `struct treenode` που αρχικοποιούνται με `{τιμή, αριστερό, δεξί}` και δείχνουν η μία στην άλλη με `&`:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct treenode {
  int value;
  struct treenode *left;
  struct treenode *right;
} *Tree;

int depth(Tree t) {
  if (t == NULL) return -1;
  int left_depth = depth(t->left);
  int right_depth = depth(t->right);
  return 1 + ((left_depth > right_depth) ? left_depth : right_depth);
}
```

```c
int main() {
  struct treenode t2 = {2, NULL, NULL}, t9 = {9, NULL, NULL};
  struct treenode t1 = {1, NULL, NULL};
  struct treenode t7 = {7, &t2, &t9}, t5 = {5, &t7, &t1};
  Tree t = &t5;
  printf("Depth: %d\n", depth(t));
  return 0;
}
```

```
$ ./depth
Depth: 2
```

Τα φύλλα 2, 9, 1 επιστρέφουν $1 + \max(-1, -1) = 0$· το 7 επιστρέφει $1 + \max(0, 0) = 1$· η ρίζα 5 επιστρέφει $1 + \max(1, 0) = 2$. Η διάλεξη ρωτά την πολυπλοκότητα για τέλειο δέντρο: χρόνος $O(n)$ (μία κλήση ανά κόμβο), χώρος $O(\log n)$ (το πολύ τόσες κλήσεις στη στοίβα όσο το ύψος).

§22.14 Οι τρεις διασχίσεις DFS

Εφαρμόζει «Διάσχιση κατά βάθος (DFS)». Η διάλεξη δίνει τρεις εκδοχές της `print` για το ίδιο δέντρο· εδώ μαζί, με διαφορετικά ονόματα:

```c
void preorder(Tree t) {      // node, left, right
  if (t == NULL) return;
  printf("%d ", t->value);
  preorder(t->left);
  preorder(t->right);
}

void inorder(Tree t) {       // left, node, right
  if (t == NULL) return;
  inorder(t->left);
  printf("%d ", t->value);
  inorder(t->right);
}

void postorder(Tree t) {     // left, right, node
  if (t == NULL) return;
  postorder(t->left);
  postorder(t->right);
  printf("%d ", t->value);
}
```

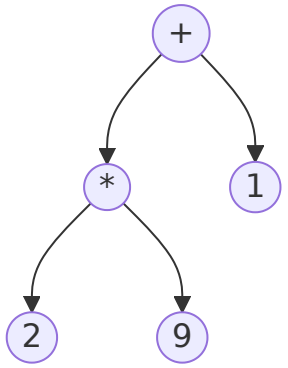
```
$ ./preorder
5 7 2 9 1
$ ./inorder
```

```
2 7 9 5 1
$ ./postorder
2 9 7 1 5
```

Για να βρείτε τη σειρά με το χέρι, ακολουθήστε τον ορισμό αναδρομικά: π.χ. in-order του 5 = in-order του 7 (δηλαδή `2 7 9`), μετά 5, μετά in-order του 1 (1).

§22.15 Αποτιμητής εκφράσεων

Η διάλεξη ρωτά: θέλουμε να γράψουμε έναν **αποτιμητή εκφράσεων (calculator / evaluator / interpreter)** για το δέντρο της παράστασης `2 * 9 + 1`· ποια διάσχιση θα χρησιμοποιήσουμε;



Σχήμα: δέντρο παράστασης· οι τελεστές είναι εσωτερικοί κόμβοι, οι αριθμοί φύλλα.

Ένας τελεστής μπορεί να υπολογιστεί μόνο όταν ξέρουμε τις τιμές των δύο υποδέντρων του: πρώτα `2 * 9 = 18`, μετά `18 + 1 = 19`. Αυτή είναι η σειρά της **post-order** (`2 9 * 1 +`). Η in-order δίνει τη συνηθισμένη γραφή `2 * 9 + 1` και η pre-order τη γραφή `+ * 2 9 1`, αλλά για τον υπολογισμό χρειαζόμαστε τα παιδιά πριν από τον γονιό.

§22.16 BFS με λίστα εργασιών

Εφαρμόζει «Διάσχιση κατά πλάτος (BFS)». Η διάλεξη δίνει μόνο τη `bfs` (πρώτα με μεταβλητή `worklist`, μετά με το όνομα `frontier`)· οι `insert` και `pop_last` δεν φαίνονται στις διαφάνειες. Το πλήρες πρόγραμμα παρακάτω τις συμπληρώνει: η λίστα κρατά `Tree` αντί για `int`, η `insert` βάζει στην αρχή (όπως στο Κεφάλαιο 21) και η `pop_last` βγάζει από το τέλος.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct treenode {
  int value;
  struct treenode *left;
  struct treenode *right;
```

```
} *Tree;

typedef struct listnode {
  Tree value;                  // the list now holds trees, not ints
  struct listnode *next;
} *List;

void insert(List *list, Tree value) {    // insert at the head
  List new_head = malloc(sizeof(struct listnode));
  new_head->value = value;
  new_head->next = *list;
  *list = new_head;
}

Tree pop_last(List *list) {              // remove from the tail
  while ((*list)->next != NULL)
    list = &((*list)->next);
  List last = *list;
  Tree value = last->value;
  *list = NULL;
  free(last);
  return value;
}

void bfs(Tree t) {
  List frontier = NULL;
  Tree tmp;
  insert(&frontier, t);
  while (frontier) {
    tmp = pop_last(&frontier);
    printf("%d ", tmp->value);
    if (tmp->left) insert(&frontier, tmp->left);
    if (tmp->right) insert(&frontier, tmp->right);
  }
}

int main() {
  struct treenode t2 = {2, NULL, NULL}, t9 = {9, NULL, NULL};
  struct treenode t1 = {1, NULL, NULL};
  struct treenode t7 = {7, &t2, &t9}, t5 = {5, &t7, &t1};
  bfs(&t5);
  printf("\n");
  return 0;
```

```
}
```

```
$ ./bfs
5 7 1 2 9
```

Η εξέλιξη του μετώπου (κεφαλή αριστερά, τέλος δεξιά):

Βήμα	Βγαίνει	Τυπώνει	Μέτωπο μετά
αρχή			5
1	5	5	1, 7
2	7	7	9, 2, 1
3	1	1	9, 2
4	2	2	9
5	9	9	(άδειο)

Η `pop_last` διατρέχει όλη τη λίστα κάθε φορά· με έναν επιπλέον δείκτη στο τέλος της λίστας θα ήταν $O(1)$, όπως υποθέτει η εκτίμηση $O(n)$ της διάλεξης.

§22.17 Συμβουλές για το εργαστήριο

Η Άσκηση 4 του Εργαστηρίου 9 (`tree.c`) ζητά ένα ταξινομημένο δυαδικό δέντρο ακεραίων: μια αναδρομική `addtree` που επιστρέφει το νέο δέντρο (`p->left = addtree(p->left, x);`) και μια `treeprint` με in-order διάσχιση, που άρα τυπώνει τους αριθμούς ταξινομημένους. Η Άσκηση 5 ζητά μια αναδρομική `free_tree`: αποδεσμεύστε **πρώτα** τα υποδέντρα και **μετά** τον κόμβο (post-order), και ελέγξτε με `valgrind` ότι δεν μένουν διαρροές.

Κύρια σημεία

1. Ένα δυαδικό δέντρο είναι αυτοαναφορική δομή όπου κάθε κόμβος έχει μια τιμή και δείκτες `left` και `right` σε έως δύο παιδιά· το άδειο δέντρο είναι `NULL`.
2. Ρίζα είναι ο πρώτος κόμβος, φύλλα οι κόμβοι χωρίς παιδιά, βάθος/ύψος ο μέγιστος αριθμός συνδέσμων ρίζας–φύλλων, και η ρίζα βρίσκεται στο επίπεδο 1.
3. Ένα δέντρο είναι τέλειο (όλα τα επίπεδα γεμάτα), γεμάτο (0 ή 2 παιδιά), πλήρες (γεμάτο εκτός από το τελευταίο επίπεδο, που γεμίζει από αριστερά), ισορροπημένο (ύψη υποδέντρων διαφέρουν έως 1) ή εκφυλισμένο (έως ένα παιδί, σαν λίστα).
4. Τα δέντρα καταστάσεων, όπως αυτό της τρίλιζας, έχουν συχνά περισσότερα από 2 παιδιά ανά κόμβο.
5. Οι λειτουργίες σε δέντρα γράφονται φυσικά με αναδρομή: βασική περίπτωση το `NULL`, αναδρομή στα δύο υποδέντρα.

6. Το DFS προχωρά όσο πιο βαθιά γίνεται και μετά οπισθοδρομεί· σε δυαδικό δέντρο δίνει τις διασχίσεις pre-order, in-order και post-order, που διαφέρουν μόνο στο πότε επεξεργαζόμαστε τον κόμβο.
7. Τα `depth`, `print` και `find` σε τέλειο δέντρο κοστίζουν χρόνο $O(n)$ και χώρο $O(\log n)$ για τη στοίβα της αναδρομής.
8. Το BFS επισκέπτεται τους κόμβους επίπεδο-επίπεδο με μια λίστα εργασιών που λειτουργεί ως ουρά· κοστίζει χρόνο $O(n)$ και χώρο $O(n)$.
9. Για συντομότερο μονοπάτι προτιμάμε BFS· όταν πρέπει να δούμε όλους τους κόμβους, το DFS χρειάζεται λιγότερη μνήμη.
10. Σε ένα BST κάθε κόμβος έχει μικρότερες τιμές αριστερά και μεγαλύτερες δεξιά, οπότε η αναζήτηση ακολουθεί ένα μόνο μονοπάτι: $O(\log n)$ σε ισορροπημένο δέντρο.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
δυαδικό δέντρο	binary tree	Δέντρο όπου κάθε κόμβος έχει 0 έως 2 παιδιά.
κόμβος / παιδί	node / child	Στοιχείο του δέντρου / κόμβος ακριβώς κάτω από έναν άλλο.
υποδέντρο	subtree	Ένα παιδί μαζί με όλους τους απογόνους του.
ρίζα	root	Ο πρώτος κόμβος του δέντρου.
φύλλο	leaf	Κόμβος χωρίς παιδιά.
βάθος / ύψος	depth / height	Μέγιστος αριθμός συνδέσμων από τη ρίζα στα φύλλα.
επίπεδο κόμβου	node level	Η γραμμή του κόμβου· η ρίζα είναι στο επίπεδο 1.
τέλειο δυαδικό δέντρο	perfect binary tree	Όλοι οι εσωτερικοί κόμβοι με 2 παιδιά, όλα τα φύλλα στο ίδιο επίπεδο.
γεμάτο δυαδικό δέντρο	full binary tree	Κάθε κόμβος έχει 0 ή 2 παιδιά.
πλήρες δυαδικό δέντρο	complete binary tree	Γεμάτα επίπεδα εκτός ίσως του τελευταίου, που γεμίζει από αριστερά.
ισορροπημένο δυαδικό δέντρο	balanced binary tree	Σε κάθε κόμβο τα ύψη των υποδέντρων διαφέρουν έως 1.
εκφυλισμένο δυαδικό δέντρο	degenerate binary tree	Κάθε κόμβος έχει έως ένα παιδί.
N-αδικό δέντρο	N-ary tree	Δέντρο με περισσότερα από 2 παιδιά ανά κόμβο.
διάσχιση	traversal	Επίσκεψη όλων των κόμβων με συγκεκριμένη σειρά.
αναζήτηση κατά βάθος	depth-first search (DFS)	Προχωρά σε βάθος και οπισθοδρομεί.

Ελληνικά	English	Σύντομος ορισμός
οπισθοδρόμηση	backtracking	Επιστροφή σε προηγούμενο κόμβο για να δοκιμαστεί άλλος κλάδος.
αναζήτηση κατά πλάτος	breadth-first search (BFS)	Εξερευνά ένα επίπεδο ολόκληρο πριν από το επόμενο.
μέτωπο / λίστα εργασιών	frontier / worklist	Οι κόμβοι που περιμένουν επεξεργασία στο BFS.
δυαδικό δέντρο αναζήτησης	binary search tree (BST)	Μικρότερες τιμές αριστερά, μεγαλύτερες δεξιά, σε κάθε κόμβο.
αφηρημένος τύπος δεδομένων	abstract data type (ΑΤΔ)	Τύπος που ορίζεται από τις λειτουργίες του, ανεξάρτητα από την υλοποίηση.

Διάβασμα

- **Διαφάνειες**: Διάλεξη 22, σελ. 1–43: δυαδικό δέντρο και ορολογία 5–9· είδη δέντρων 10–14· N-αδικά δέντρα 15· λειτουργίες 16–17· βάθος 18–20· DFS και διασχίσεις 21–29· `find` 30–31· BFS 32–34· BST 35–37· ερωτήσεις DFS/BFS 38–41.
- **Σημειώσεις**: η διάλεξη προτείνει τις σελ. 120–135 των σημειώσεων του κ. Σταματόπουλου (K04):
  - Κεφάλαιο 7: «Αυτο-αναφορικές δομές» (K04, σελ. 120–125), «Δημιουργία νέων ονομάτων τύπων» (126).
  - Κεφάλαιο 8: «Διαχείριση συνδεδεμένων λιστών» (K04, σελ. 128–131), «Διαχείριση δυαδικών δέντρων» (132–135), με τις `addtree`, `treeprint`, `nodesprint`, `treedepth`, `treesearch`.
- **Εργαστήριο**: Εργαστήριο 9: Άσκηση 4 `tree.c`, Άσκηση 5 (`free_tree` στο `tree.c`).
- **Άλλα**: Binary Tree Problems (Stanford CS Education Library)· Tree traversal, Depth-first search, Breadth-first search, Binary search tree (Wikipedia) και οπτικοποίηση BST· Αφηρημένος τύπος δεδομένων (Βικιπαίδεια).

Συχνά λάθη

- **Πρόσβαση σε `t->left` χωρίς έλεγχο για `NULL`.** Μια αναδρομική συνάρτηση χωρίς `if (t == NULL) return ...;` στην αρχή σκάει στα φύλλα με `Segmentation fault`.
- **Λάθος βασική περίπτωση στο βάθος.** Με `return 0` για το άδειο δέντρο, το δέντρο 5, 7, 1, 2, 9 δίνει 3 (επίπεδα) αντί για 2 (συνδέσμους). Διαλέξτε -1 ή 0 ανάλογα με τον ορισμό που ζητείται.
- **Σύγχυση των διασχίσεων.** Το «pre», «in», «post» αναφέρεται στη θέση του **κόμβου** σε σχέση με τα παιδιά του· τα παιδιά πάντα αριστερό πριν από δεξί.
- **Αναζήτηση BST σε δέντρο που δεν είναι BST.** Η `exists` πάει μόνο αριστερά ή μόνο δεξιά· σε τυχαίο δέντρο (π.χ. το 5, 7, 1, 2, 9) η `exists(t, 9)` επιστρέφει 0 ενώ το 9 υπάρχει.

Εκεί χρειάζεται η `find`.

- **Έλεγχος BST μόνο με τα άμεσα παιδιά.** Η ιδιότητα αφορά **όλο** το υποδέντρο: ένα 8 κάτω δεξιά από το 5 στο αριστερό υποδέντρο του 7 χαλάει το BST, αν και `5 < 8` ισχύει.
- **Εισαγωγή σε BST χωρίς να κρατηθεί το αποτέλεσμα.** Γράφοντας `addtree(p->left, x);` αντί για `p->left = addtree(p->left, x);`, ο νέος κόμβος χάνεται και το δέντρο μένει άδειο.
- **`free` πριν από τα παιδιά.** Αν αποδεσμεύσετε τον κόμβο και μετά διαβάσετε `p->left`, το `valgrind` δείχνει `Invalid read`· αποδεσμεύστε με post-order.
- **BFS με στοίβα αντί για ουρά.** Αν η λίστα εργασιών βγάζει από την ίδια άκρη που βάζει, οι κόμβοι δεν βγαίνουν ανά επίπεδο.

Τι δυσκόλεψε την τάξη

Από τα Kahoot των διαλέξεων: οι ερωτήσεις όπου μια λάθος απάντηση μάζεψε πολλές ψήφους, με το ποσοστό σωστών απαντήσεων.

- **K22.9** Χειρότερη περίπτωση αναζήτησης σε BST (29% σωστές): Το 30% επέλεξε το ισορροπημένο δέντρο, ενώ σε ισορροπημένο BST η σύγκριση με κάθε κόμβο δείχνει πάντα προς τα πού να ψάξουμε, άρα αρκούν O(log n) βήματα.
- **K22.8** Αποτίμηση δέντρου εκφράσεων (35% σωστές): Το 28% επέλεξε preorder, όμως για να εφαρμόσουμε τον τελεστή της ρίζας χρειαζόμαστε πρώτα τις τιμές και των δύο υποδέντρων.
- **K22.7** BFS σε τέλειο δυαδικό δέντρο (37% σωστές): Το 61% απάντησε True, μπερδεύοντας το ύψος του δέντρου (log n) με το κόστος της αναζήτησης· η BFS σε δέντρο που δεν είναι BST μπορεί να χρειαστεί να επισκεφθεί όλους τους κόμβους.
- **K22.4** Κόμβοι δέντρου με n επίπεδα (53% σωστές): Το 37% επέλεξε `2^n`, που είναι μόνο το μέγιστο για ένα τέλειο δυαδικό δέντρο· ένα τυχαίο δέντρο μπορεί να έχει από έναν κόμβο ανά επίπεδο μέχρι πολύ περισσότερους.

Ερωτήσεις κατανόησης

- **E22.1** Πόσους κόμβους έχει ένα τέλειο δυαδικό δέντρο με n επίπεδα;[171]
- **E22.2** Είναι κάθε τέλειο δέντρο πλήρες; Είναι κάθε πλήρες δέντρο γεμάτο;[172]
- **E22.3** Τι επιστρέφει η `depth` της διάλεξης για ένα δέντρο με έναν μόνο κόμβο, και γιατί η βασική περίπτωση επιστρέφει -1;[173]
- **E22.4** Ποια διάσχιση τυπώνει ταξινομημένα τα στοιχεία ενός BST;[174]

[171] $1 + 2 + 4 + \cdots + 2^{n-1} = 2^n - 1$, αφού το επίπεδο k έχει 2^{k-1} κόμβους.

[172] Ναι, το τέλειο είναι και πλήρες και γεμάτο. Όχι, ένα πλήρες δέντρο μπορεί να έχει κόμβο με ένα παιδί (π.χ. το 4 με μόνο το 8 στο παράδειγμα της διάλεξης), άρα δεν είναι γεμάτο.

[173] 0, αφού $1 + \max(-1, -1) = 0$. Το -1 κάνει το αποτέλεσμα να μετρά συνδέσμους, όπως ο ορισμός του βάθους.

[174] Η in-order (αριστερό, κόμβος, δεξί), αφού όλα τα αριστερά είναι μικρότερα και όλα τα δεξιά μεγαλύτερα.

- **E22.5** Γιατί το BFS χρειάζεται χώρο $O(n)$ ενώ το DFS σε τέλειο δέντρο $O(\log n)$;[175]
- **E22.6** Γιατί η `exists` είναι $O(\log n)$ ενώ η `find` είναι $O(n)$; Πότε η `exists` γίνεται κι αυτή $O(n)$;[176]
- **E22.7** Σε ποια σειρά πρέπει να αποδεσμεύσουμε τους κόμβους ενός δέντρου, και ποια διάσχιση είναι αυτή;[177]

Kahoot από το αμφιθέατρο (K22.1–K22.9)

Ερωτήσεις που παίχτηκαν στις διαλέξεις, με το ποσοστό των φοιτητών που απάντησαν σωστά.

- **K22.1** Κόμβοι τέλειου δυαδικού δέντρου: 81% σωστές απαντήσεις
- **K22.2** Ο καλύτερος αλγόριθμος αναζήτησης: 75% σωστές απαντήσεις
- **K22.3** Ταξινομημένη εκτύπωση BST: 66% σωστές απαντήσεις
- **K22.4** Κόμβοι δέντρου με n επίπεδα: 53% σωστές απαντήσεις
- **K22.5** Ελάχιστο βάθος δυαδικού δέντρου: 47% σωστές απαντήσεις
- **K22.6** Φύλλα τέλειου δυαδικού δέντρου: 45% σωστές απαντήσεις
- **K22.7** BFS σε τέλειο δυαδικό δέντρο: 37% σωστές απαντήσεις
- **K22.8** Αποτίμηση δέντρου εκφράσεων: 35% σωστές απαντήσεις
- **K22.9** Χειρότερη περίπτωση αναζήτησης σε BST: 29% σωστές απαντήσεις

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A22.1–A22.13)

- **A22.1** Ποιος αλγόριθμος αναζήτησης είναι καλύτερος;: Διαλέξεις 21–22: Δέντρα, διαφάνεια 38 (διάλεξη 21: διαφάνεια 60) · ★☆☆ · short-answer · `slides-lec22-best-search`
- **A22.2** Έλεγχος ύπαρξης σε δυαδικό δέντρο αναζήτησης: Διαλέξεις 21–22: Δέντρα, διαφάνειες 35–37 (διάλεξη 21: διαφάνειες 57–59) · ★☆☆ · programming · `slides-lec22-bst-exists`
- **A22.3** Κόμβοι τέλειου δυαδικού δέντρου: Διαλέξεις 21–22: Δέντρα, διαφάνεια 10 (διάλεξη 21: διαφάνεια 32) · ★☆☆ · short-answer · `slides-lec22-perfect-tree-nodes`
- **A22.4** Διασχίσεις pre-order, in-order, post-order: Διαλέξεις 21–22: Δέντρα, διαφάνειες 22–28 (διάλεξη 21: διαφάνειες 44–50) · ★☆☆ · trace · `slides-lec22-traversals`
- **A22.5** Μέγιστο στοιχείο δέντρου: BFS ή DFS;: Διαλέξεις 21–22: Δέντρα, διαφάνεια 41 (διάλεξη 21: διαφάνεια 63) · ★☆☆ · short-answer · `slides-lec22-tree-max`

[175] Το μέτωπο του BFS κρατά κάποια στιγμή ολόκληρο το τελευταίο επίπεδο, περίπου $n/2$ κόμβους· η στοίβα του DFS κρατά μόνο ένα μονοπάτι από τη ρίζα, μήκους $\log_2 n$.

[176] Η `exists` χρησιμοποιεί τη διάταξη του BST και κατεβαίνει σε ένα μόνο υποδέντρο, μία κλήση ανά επίπεδο· η `find` πρέπει να ψάξει και τα δύο. Σε εκφυλισμένο BST το ύψος είναι $n-1$, οπότε και η `exists` κάνει $O(n)$ βήματα.

[177] Πρώτα τα δύο υποδέντρα, μετά τον κόμβο: post-order. Αλλιώς θα διαβάζαμε `p->left` από μνήμη που έχει ήδη αποδεσμευτεί.

- **A22.6** Πολυπλοκότητα της depth: Διαλέξεις 21–22: Δέντρα, διαφάνεια 20 (διάλεξη 21: διαφάνειες 41–42) · ★★☆ · short-answer · `slides-lec22-depth-complexity`
- **A22.7** Διάσχιση για αποτιμητή εκφράσεων: Διαλέξεις 21–22: Δέντρα, διαφάνεια 29 (διάλεξη 21: διαφάνεια 51) · ★★☆ · short-answer · `slides-lec22-expression-evaluator`
- **A22.8** Λίστα για το BFS και λίστες κάθε τύπου: Διαλέξεις 21–22: Δέντρα, διαφάνεια 33 (διάλεξη 21: διαφάνεια 55) · ★★☆ · short-answer · `slides-lec22-generic-list`
- **A22.9** Συντομότερο μονοπάτι σε δέντρα-λαβύρινθους: BFS ή DFS;: Διαλέξεις 21–22: Δέντρα, διαφάνεια 40 (διάλεξη 21: διαφάνεια 62) · ★★☆ · short-answer · `slides-lec22-maze-shortest-path`
- **A22.10** Αποθήκευση και αναζήτηση σε 1 PetaByte: Διαλέξεις 21–22: Δέντρα, διαφάνεια 39 (διάλεξη 21: διαφάνεια 61) · ★★☆ · short-answer · `slides-lec22-petabyte`
- **A22.11** Δέντρο καταστάσεων τρίλιζας: Διάλεξη 22: Δέντρα, διαφάνεια 15 · ★★☆ · short-answer · `slides-lec22-tictactoe-tree`
- **A22.12** Προσθήκη στοιχείου σε δυαδικό δέντρο: Διαλέξεις 21–22: Δέντρα, διαφάνεια 16 (διάλεξη 21: διαφάνεια 38) · ★★☆ · programming · `slides-lec22-tree-insert`
- **A22.13** Αφαίρεση στοιχείου από δυαδικό δέντρο: Διαλέξεις 21–22: Δέντρα, διαφάνεια 16 (διάλεξη 21: διαφάνεια 38) · ★★★ · programming · `slides-lec22-tree-delete`

Εργαστήριο (A22.14)

- **A22.14** Δυαδικά δένδρα: Εργαστήριο 9, Άσκηση 4 · ★★☆ · programming · `lab-lab09-tree`

Εργασίες (A22.15–A22.17)

- **A22.15** Zoomba: συντομότερη διαδρομή σε δωμάτιο: Εργασία 3 (2023-24), Άσκηση 1 · ★★★ · programming · `hw-2023-hw3-zoomba`
- **A22.16** Νέα Μηχανή Σκακιού (chess engine): Εργασία 3 (2024-25), Άσκηση 1 · ★★★ · programming · `hw-2024-hw3-chess`
- **A22.17** Νέα Μηχανή Go (goteam): Εργασία 3 (2025-26), Άσκηση 1 · ★★★ · programming · `hw-2025-hw3-goteam`

Θέματα εξετάσεων (A22.18–A22.20)

- **A22.18** Αθροιστής Δέντρων - sumtree: Εξέταση Ιανουαρίου 2025, Θέμα 4 · ★☆☆ · programming · `exam-2025-jan-q4`
- **A22.19** Reverse Inorder Traversal: Εξέταση Ιουλίου 2024, Θέμα 4 · ★★☆ · programming · `exam-2024-jul-q4`
- **A22.20** Διερμηνέας Αριθμητικών Εκφράσεων - eval: Εξέταση Ιανουαρίου 2026, Θέμα 4 · ★★☆ · programming · `exam-2026-jan-q4`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A20.4** Γενεαλογικό δέντρο: Διάλεξη 20, διαφάνεια 43 · ★☆☆ · short-answer · `slides-lec20-family-tree`
- **A25.16** Περικύκλωση - encirclement: Εξέταση Ιανουαρίου 2026, Θέμα 5 · ★★★ · programming · `exam-2026-jan-q5`
- **A25.1** Καθαρή διαχείριση μνήμης: Εργαστήριο 9, Άσκηση 5 · ★★☆ · programming · `lab-lab09-grades-tree`

Μέρος Ε: Οργάνωση κώδικα και προχωρημένα θέματα

Κεφάλαιο 23: Οργάνωση Κώδικα

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να εξηγείτε γιατί ένα μεγάλο πρόγραμμα σπάει σε πολλά αρχεία· να χωρίζετε ένα πρόγραμμα σε αρχεία υλοποίησης (`.c`) και αρχεία κεφαλίδας (`.h`)· να μεταγλωττίζετε κάθε αρχείο χωριστά με `gcc -c` και να συνδέετε τα αντικειμενικά αρχεία· να γράφετε πρωτότυπα συναρτήσεων· και να χρησιμοποιείτε το `const` σε μεταβλητές, δείκτες και παραμέτρους.

Προαπαιτούμενα: Κεφάλαιο 3, Κεφάλαιο 14, Κεφάλαιο 15

Χρόνος μελέτης: ~1,5 ώρα

Σύνοψη

Τα προγράμματα που γράφαμε ως τώρα χωρούσαν σε ένα αρχείο· τα πραγματικά προγράμματα έχουν χιλιάδες ή εκατομμύρια γραμμές κώδικα. Η διάλεξη ξεκινά από το μέγεθος των προγραμμάτων, μετρημένο σε γραμμές κώδικα, και ρωτά πώς οργανώνουμε ένα σύστημα 40.000 γραμμών. Η απάντηση είναι η αφαίρεση: χωρίζουμε τον κώδικα σε αρχεία υλοποίησης και σε αρχεία κεφαλίδας που περιγράφουν τη διεπαφή τους, μεταγλωττίζουμε κάθε αρχείο χωριστά σε αντικειμενικό αρχείο και τα συνδέουμε στο τέλος. Επειδή ο μεταγλωττιστής της C διαβάζει τον κώδικα γραμμικά, κάθε συνάρτηση πρέπει να έχει δηλωθεί με πρωτότυπο πριν κληθεί. Τέλος, ο προσδιοριστής `const` δηλώνει ότι κάτι δεν αλλάζει και κάνει τις δηλώσεις των συναρτήσεων συμβόλαια με τους χρήστες τους.

Θεωρία

§23.1 Το μέγεθος ενός προγράμματος: γραμμές κώδικα

Μία από τις βασικές μετρικές για το μέγεθος ενός προγράμματος είναι οι **γραμμές κώδικα** (lines of code, LOC, ή source lines of code, SLOC): οι εντολές που γράφουμε μέχρι την αλλαγή γραμμής. Για μεγάλα προγράμματα χρησιμοποιούμε πολλαπλάσια:

1 KLOC $= 10^3$ LOC και 1 MLOC $= 10^6$ LOC.

Με λίγες γραμμές κώδικα μπορούμε να πάρουμε ιδιαίτερα σύνθετα συστήματα: το Game of Life του Conway (1970) θέλει περίπου 100–200 LOC. Άλλα είναι τεράστια:

Πρόγραμμα	Έτος	Μέγεθος
Λογισμικό του Apollo 11	1969	145 KLOC
Windows Vista	2006	50 MLOC
Πυρήνας του Linux	σήμερα	πάνω από 40 MLOC

Ακόμα και οι εργασίες του μαθήματος, όλες μαζί, ξεπερνούν τις 67.000 γραμμές (δείτε το παράδειγμα «Πόσες γραμμές γράψαμε στις εργασίες»).

§23.2 Λύση #1: όλος ο κώδικας σε ένα αρχείο

Ας πούμε ότι υλοποιούμε ένα καινούριο σύστημα και περιμένουμε να έχει ~40 χιλιάδες γραμμές κώδικα. Η πρώτη ιδέα είναι να τα βάλουμε όλα σε ένα αρχείο C.

Θετικά	Αρνητικά
Απλή οργάνωση, εύκολη μεταφορά: όλος ο κώδικας σε ένα μέρος. Οι ορισμοί όλων των συναρτήσεων είναι προσβάσιμοι στο ίδιο αρχείο.	Το να ψάχνετε κάτι σε ένα αρχείο με δεκάδες χιλιάδες γραμμές είναι οδυνηρό. Αλλάζετε μία γραμμή και πρέπει να ξαναμεταγλωττίσετε τα πάντα. Η συντήρηση, η αναβάθμιση και η κατανόηση όλου του προγράμματος γίνονται δύσκολες.

Για τις 100 γραμμές μιας άσκησης αυτό είναι μια χαρά· για 40.000 όχι.

§23.3 Αφαίρεση και διάσπαση σε υποπροβλήματα

Η ιδέα που λύνει το πρόβλημα είναι η **αφαίρεση** (abstraction) και η **διάσπαση σε υποπροβλήματα**. Χωρίζουμε το σύστημα σε κομμάτια, και κάθε κομμάτι δίνει στα υπόλοιπα μια απλή περιγραφή του *τι* κάνει, κρύβοντας το *πώς* το κάνει. Όποιος χρησιμοποιεί ένα κομμάτι χρειάζεται να ξέρει μόνο την περιγραφή του. Την ίδια ιδέα την εφαρμόζουμε ήδη με τις συναρτήσεις (Κεφάλαιο 3)· εδώ την εφαρμόζουμε σε επίπεδο αρχείων.

§23.4 Λύση #2: οργάνωση σε πολλά αρχεία

Στη δεύτερη λύση κάθε αρχείο περιέχει μεταβλητές και συναρτήσεις που σχετίζονται θεματικά, λειτουργικά ή με άλλο κριτήριο. Ένα πρόγραμμα C έχει δύο είδη αρχείων:

- **Αρχεία υλοποίησης** (`.c`, implementation files): ο κώδικας των συναρτήσεων.
- **Αρχεία κεφαλίδας** (`.h`, header files): οι δηλώσεις, δηλαδή τι προσφέρει ένα κομμάτι του προγράμματος στα υπόλοιπα.

Για παράδειγμα, η βιβλιοθήκη κρυπτογραφίας openssl είναι οργανωμένη σε καταλόγους και αρχεία:

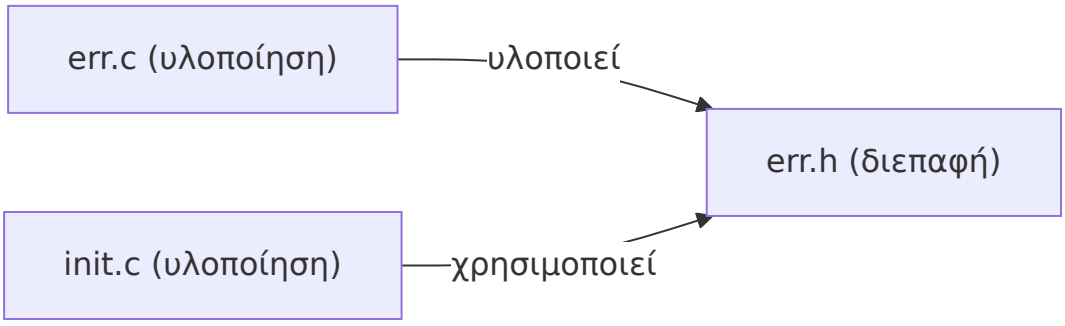
```
□□□ README.md
□□□ ssl
□   □□□ ssl_init.c
□   □□□ event_queue.c
□   □□□ ssl_err.c
□   □□□ sslerr.h
...
□□□ test
□   □□□ aborttest.c
□   □□□ acvp_test.c
...
```

Ο κατάλογος `ssl` κρατά τον κώδικα του πρωτοκόλλου και ο `test` τα τεστ. Ποιος είναι ο καλύτερος τρόπος να οργανώσετε τον κώδικά σας δεν έχει μία απάντηση· ένας καλός κανόνας είναι ότι ό,τι αλλάζει μαζί μένει μαζί.

§23.5 Εξαρτήσεις: διεπαφή και υλοποίηση

Πάρτε ένα πρόγραμμα με τρία αρχεία. Το `err.c` ορίζει μια συνάρτηση `print_err`, το `err.h` δηλώνει το πρωτότυπό της, και το `init.c` κάνει `#include "err.h"` και την καλεί (ο κώδικας είναι στο παράδειγμα «Χωριστή μεταγλώττιση των err.c και init.c»).

Το `err.h` είναι η **διεπαφή** (interface): λέει ότι υπάρχει μια συνάρτηση `print_err` που παίρνει `char *` και επιστρέφει `int`. Το `err.c` **υλοποιεί** τη διεπαφή, και το `init.c` τη **χρησιμοποιεί**. Κανένα από τα δύο `.c` δεν χρειάζεται να ξέρει το άλλο· και τα δύο εξαρτώνται μόνο από το `.h`.



Σχήμα: το `err.c` *υλοποιεί και το* `init.c` *χρησιμοποιεί τη διεπαφή* `err.h`.

Οι **εξαρτήσεις** (dependencies) απαντούν στο ερώτημα «αν αλλάξω ένα αρχείο, τι επηρεάζεται;». Αν αλλάξει το σώμα της `print_err` στο `err.c`, το `init.c` δεν επηρεάζεται, αφού βλέπει μόνο τη διεπαφή. Αν αλλάξει το `err.h`, επηρεάζονται και τα δύο. Τα αρχεία και οι εξαρτήσεις τους σχηματίζουν έναν **γράφο** (κόμβοι τα αρχεία, ακμές οι εξαρτήσεις), και αυτόν τον γράφο διαβάζει το `make` για να αποφασίσει τι πρέπει να ξαναχτιστεί.

§23.6 Τα στάδια της μεταγλώττισης

Ο μεταγλωττιστής (`gcc`) παίρνει τον **πηγαίο κώδικα** (source code, π.χ. `hello.c`) και βγάζει το **δυαδικό πρόγραμμα** (binary program, π.χ. `hello`). Στο εσωτερικό του η δουλειά γίνεται σε τρία στάδια, και καθένα έχει το δικό του ενδιάμεσο αποτέλεσμα:

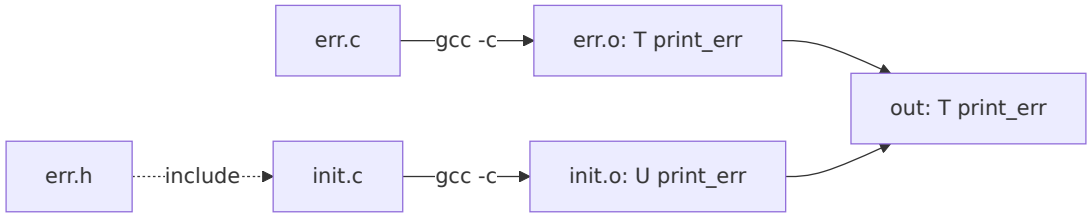
Σχήμα: από τον πηγαίο κώδικα στο εκτελέσιμο, μέσα από τα τρία στάδια του gcc.

1. Ο **προεπεξεργαστής** (preprocessor) εκτελεί τις οδηγίες `#include` και `#define` (Κεφάλαιο 15): το `#include "err.h"` αντικαθίσταται κυριολεκτικά από το περιεχόμενο του `err.h`.
2. Η **μεταγλώττιση** μετατρέπει τον κώδικα σε εντολές μηχανής και γράφει ένα **αντικειμενικό αρχείο** (object file, `.o`).
3. Η **σύνδεση** (linking) ενώνει ένα ή περισσότερα αντικειμενικά αρχεία σε ένα εκτελέσιμο.

Όταν γράφουμε `gcc -o hello hello.c` γίνονται και τα τρία μαζί. Με επιλογές τα χωρίζουμε: το `-c` σταματά μετά τη μεταγλώττιση και αφήνει το `.o`.

§23.7 Χωριστή μεταγλώττιση και σύνδεση

Με πολλά αρχεία, κάθε `.c` μεταγλωττίζεται **ανεξάρτητα** σε δικό του `.o`, και μετά τα `.o` συνδέονται: `gcc -c -o err.o err.c`, `gcc -c -o init.o init.c` και τέλος `gcc -o out init.o err.o`.



Σχήμα: χωριστή μεταγλώττιση των δύο `.c` και σύνδεση των `.o` στο `out`.

Όταν μεταγλωττίζεται το `init.c`, ο μεταγλωττιστής ξέρει από το `err.h` πώς καλείται η `print_err`, αλλά όχι πού βρίσκεται ο κώδικάς της. Γι' αυτό σημειώνει στο `init.o` το σύμβολο `print_err` ως **απροσδιόριστο** (`U`, undefined). Στο `err.o` το ίδιο σύμβολο είναι **ορισμένο** στον κώδικα (`T`, text). Αυτά φαίνονται με το `nm` (Κεφάλαιο 14). Η σύνδεση ταιριάζει κάθε `U` με ένα `T`· στο τελικό `out` η `print_err` είναι πια `T`. Αν κάποιο `U` δεν βρει ορισμό, η σύνδεση αποτυγχάνει με `undefined reference`.

§23.8 Γιατί χωριστά: η μεταγλώττιση είναι χρονοβόρα

Σε μεγάλα projects, όπως ο πυρήνας του Linux, η μεταγλώττιση μπορεί να πάρει **ώρες** (ή και μέρες). Σπάζοντας το πρόγραμμα σε αρχεία κερδίζουμε χρόνο: μετά από μια αλλαγή

ξαναμεταγλωττίζουμε μόνο ό,τι χρειάζεται. Αν αλλάξει μόνο το `err.c`, φτιάχνουμε ξανά το `err.o` και ξανασυνδέουμε· το `init.o` μένει ως έχει. Μετά την πρώτη μεταγλώττιση οι επόμενες επαναλήψεις είναι συνήθως πολύ γρηγορότερες.

Το να θυμόμαστε τι πρέπει να ξαναφτιαχτεί είναι δουλειά για το εργαλείο `make` και τα Makefiles: δηλώνουμε τον γράφο των εξαρτήσεων και το `make` ξαναχτίζει μόνο όσα άλλαξαν. Το `make` το είδαμε στην προσκεκλημένη διάλεξη (Παράρτημα).

§23.9 Η μεταγλώττιση είναι γραμμική: πρωτότυπα συναρτήσεων

Ο μεταγλωττιστής της C διαβάζει το αρχείο **μία φορά, από πάνω προς τα κάτω**. Όταν φτάσει σε μια κλήση, πρέπει να ξέρει ήδη τον τύπο επιστροφής και τα ορίσματα της συνάρτησης. Αν η συνάρτηση ορίζεται πιο κάτω, ο `gcc` δεν την έχει δει ακόμα και διαμαρτύρεται με `implicit declaration of function` (δείτε το παράδειγμα «Κλήση πριν από τον ορισμό»).

Η λύση είναι η **δήλωση πρωτοτύπου συνάρτησης** (function prototype): δηλώνει το **όνομα** της συνάρτησης, τον **τύπο επιστροφής** της και τα **ορίσματά** της, χωρίς σώμα, και τελειώνει με `;`.

```
τύπος όνομα(λίστα_ορισμάτων);   /* γενική μορφή */
int print_err(char * message);
int print_err(char *);          /* τα ονόματα των ορισμάτων παραλείπονται */
```

Τα ονόματα των ορισμάτων δεν χρειάζονται, αφού ο μεταγλωττιστής θέλει μόνο τους τύπους· συχνά όμως τα κρατάμε γιατί τεκμηριώνουν τι σημαίνει κάθε όρισμα.

Εδώ δένουν όλα: ένα αρχείο κεφαλίδας είναι ουσιαστικά μια λίστα από πρωτότυπα. Όταν ένα `.c` κάνει `#include "err.h"`, ο προεπεξεργαστής βάζει τα πρωτότυπα στην αρχή του αρχείου, και έτσι κάθε κλήση που ακολουθεί είναι γνωστή στον μεταγλωττιστή, αν και ο κώδικας βρίσκεται σε άλλο αρχείο.

§23.10 Δηλώσεις και ορισμοί σε πολλά αρχεία

Η διαφάνεια «Σήμερα» αναφέρει δηλώσεις μεταβλητών και συναρτήσεων· οι σημειώσεις και το Εργαστήριο 10 συμπληρώνουν τους κανόνες που χρειάζεστε για να σπάσετε σωστά ένα πρόγραμμα:

- Ένας **ορισμός** (definition) δεσμεύει μνήμη ή δίνει κώδικα και υπάρχει **ακριβώς μία φορά** σε όλο το πρόγραμμα. Μια **δήλωση** (declaration) απλώς ενημερώνει τον μεταγλωττιστή για τον τύπο και μπορεί να επαναλαμβάνεται. Γι' αυτό στα `.h` βάζουμε δηλώσεις, όχι κώδικα, και ποτέ δεν κάνουμε `#include` ένα `.c`.
- Μια καθολική μεταβλητή που τη χρειάζονται πολλά αρχεία ορίζεται σε **ένα** `.c` (`int sp;`) και δηλώνεται στα υπόλοιπα με `extern int sp;`.
- Το `static` μπροστά σε καθολική μεταβλητή ή συνάρτηση την κάνει ορατή **μόνο στο δικό της αρχείο**· έτσι ένα αρχείο κρύβει τις βοηθητικές του λεπτομέρειες.
- Γράφουμε `#include "err.h"` με εισαγωγικά για τα δικά μας αρχεία (αναζήτηση πρώτα στον κατάλογο του πηγαίου αρχείου) και `#include <stdio.h>` για τα αρχεία του

συστήματος.
- Κάθε `.h` προστατεύεται με **include guard**, ώστε αν συμπεριληφθεί δύο φορές στο ίδιο `.c` να διαβαστεί μόνο την πρώτη:

```
#ifndef ERR_H
#define ERR_H
int print_err(char *);
#endif
```

§23.11 Ο προσδιοριστής const

Ο **προσδιοριστής** `const` (const qualifier) δηλώνει ότι το περιεχόμενο κάποιων θέσεων μνήμης είναι σταθερό. Η γενική σύνταξη είναι `const δήλωση_μεταβλητής;`:

```
const int x = 42;
const char message[] = "Hello";
const char const * msg_ptr = message;
```

Μια `const` μεταβλητή πρέπει να πάρει την τιμή της στη δήλωση, γιατί μετά δεν επιτρέπεται να αλλάξει. Αν κάτι δηλωθεί ως `const`, **δεν** πρέπει να το αλλάξουμε· και ο μεταγλωττιστής το ελέγχει: το `x++` δίνει `error: increment of read-only variable 'x'`.

Στους δείκτες έχει σημασία **πού** γράφεται το `const`. Διαβάστε τη δήλωση από τα δεξιά προς τα αριστερά:

Δήλωση	Τι είναι σταθερό	Επιτρέπεται `*p = 'a';`	Επιτρέπεται `p++;`
`const char *p` (ή `char const *p`)	οι χαρακτήρες όπου δείχνει	όχι	ναι
`char * const p`	ο ίδιος ο δείκτης	ναι	όχι
`const char * const p`	και τα δύο	όχι	όχι

Στο τρίτο παράδειγμα της διαφάνειας, `const char const * msg_ptr`, και τα δύο `const` βρίσκονται **πριν** από το `*`, άρα αφορούν και τα δύο τους χαρακτήρες: ο `msg_ptr` είναι δείκτης σε σταθερούς χαρακτήρες, και ο `gcc -Wall` προειδοποιεί `duplicate 'const' declaration specifier`. Για σταθερό δείκτη σε σταθερούς χαρακτήρες γράφουμε `const char * const msg_ptr`.

Το `const` δεν είναι μόνο υπόσχεση προς τον μεταγλωττιστή. Ένας καθολικός πίνακας `const`, όπως το `message`, μπαίνει συνήθως σε μνήμη **μόνο για ανάγνωση**. Αν «ξεγελάσετε» τον μεταγλωττιστή με ένα cast σε `char *` και γράψετε εκεί, το πρόγραμμα μεταγλωττίζεται αλλά σκάει με `Segmentation fault` (δείτε το παράδειγμα «Εγγραφή σε const μνήμη»). Δεν αλλάζουμε `const` θέσεις μνήμης.

§23.12 const σε ορισμούς συναρτήσεων: συμβόλαια

Με το `const` στις δηλώσεις συναρτήσεων (ή στα πρωτότυπα) δημιουργούμε **συμβόλαια** (contracts) με τους χρήστες της συνάρτησης:

```
int print_err(const char * message);
```

Αυτό το πρωτότυπο εγγυάται ότι η `print_err` **δεν θα αλλάξει** τους χαρακτήρες της `message`. Όποιος την καλεί μπορεί να της δώσει και σταθερή συμβολοσειρά, χωρίς να φοβάται ότι θα του την πειράξει· και αν ο κώδικας της `print_err` προσπαθήσει να γράψει στο `message[0]`, η μεταγλώττιση αποτυγχάνει. Τέτοιες εγγυήσεις κάνουν τους προγραμματιστές πιο αποδοτικούς και επιτρέπουν στους μεταγλωττιστές να γράφουν πιο γρήγορο κώδικα. Τις έχετε ήδη συναντήσει στη βιβλιοθήκη: `strlen(const char *s)`, `strcmp(const char *s1, const char *s2)`.

Παραδείγματα

§23.13 Πόσες γραμμές γράψαμε στις εργασίες

Το εργαλείο `sloccount` μετρά τις γραμμές κώδικα ενός καταλόγου και, με το μοντέλο COCOMO, εκτιμά πόσο κόστος και χρόνο θα ήθελε η ανάπτυξή τους. Στις υποβολές των εργασιών του μαθήματος (θέμα «Το μέγεθος ενός προγράμματος»):

```
$ sloccount hw-submissions/
Total Physical Source Lines of Code (SLOC)                = 67,826
Development Effort Estimate, Person-Years (Person-Months) = 16.75 (200.99)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 1.56 (18.76)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 10.72
Total Estimated Cost to Develop                           = $ 2,262,599
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```

Σχεδόν 68 KLOC: όλες οι εργασίες μαζί είναι μισό Apollo 11.

§23.14 Χωριστή μεταγλώττιση των err.c και init.c

Το πρόγραμμα της ενότητας «Εξαρτήσεις: διεπαφή και υλοποίηση», ολοκληρωμένο. Εφαρμόζει τη «Χωριστή μεταγλώττιση και σύνδεση».

```
/* err.h */
int print_err(char *);
```

```
/* err.c */
```

```c
#include <stdio.h>
#include "err.h"

int print_err(char *msg) {
  return fprintf(stderr, "%s\n", msg);
}
```

```c
/* init.c (does-not-compile χωρίς το err.h δίπλα του) */
#include "err.h"

int main(void) {
  print_err("hi");
  return 0;
}
```

Το `err.c` κάνει κι αυτό `#include "err.h"`: έτσι, αν το πρωτότυπο και ο ορισμός πάψουν κάποτε να συμφωνούν, ο μεταγλωττιστής θα το δει. Μεταγλωττίζουμε, κοιτάμε τα σύμβολα και συνδέουμε:

```
$ gcc -c -o err.o err.c
$ gcc -c -o init.o init.c
$ nm err.o | grep print_err
0000000000000000 T print_err
$ nm init.o | grep print_err
                 U print_err
$ gcc -o out init.o err.o
$ ./out
hi
```

Στο `init.o` η `print_err` είναι `U`· η σύνδεση τη βρίσκει ως `T` στο `err.o`. Αν ξεχάσετε το `err.o` στη σύνδεση (`gcc -o out init.o`), ο linker δεν βρίσκει ορισμό και σταματά με `undefined reference to 'print_err'`.

§23.15 Κλήση πριν από τον ορισμό

Το `prototype.c` καλεί την `print_err` πριν την ορίσει (θέμα «Η μεταγλώττιση είναι γραμμική: πρωτότυπα συναρτήσεων»):

```c
// does-not-compile (σε νεότερους gcc)
#include <stdio.h>

int main() {
  print_err("hello");
  return 0;
}
int print_err(char * msg) {
```

```
  return fprintf(stderr, "%s\n", msg);
}
```

```
$ gcc -o prototype prototype.c
prototype.c: In function ‘main’:
prototype.c:4:3: warning: implicit declaration of function ‘print_err’
[-Wimplicit-function-declaration]
    4 |   print_err("hello");
      |   ^~~~~~~~~
```

Στη γραμμή 4 ο gcc δεν έχει δει ακόμα την print_err. Ο gcc των διαφανειών δίνει προειδοποίηση· ο gcc 14 και νεότεροι το θεωρούν **σφάλμα** (error: αντί για warning:). Και στις δύο περιπτώσεις η διόρθωση είναι η ίδια: προσθέτουμε το πρωτότυπο πριν από τη main.

```
#include <stdio.h>
int print_err(char *msg);

int main() {
  print_err("hello");
  return 0;
}
int print_err(char * msg) {
  return fprintf(stderr, "%s\n", msg);
}
```

```
$ gcc -o prototype prototype.c
$ ./prototype
hello
```

§23.16 Αλλαγή μιας const μεταβλητής

Το const1.c προσπαθεί να αυξήσει μια const μεταβλητή (θέμα «Ο προσδιοριστής const»):

```
// does-not-compile
const char message[] = "Hello";
int main() {
  const int x = 42;
  const char const * msg_ptr = message;
  x++;
  return 0;
}
```

```
$ gcc -o const1 const1.c
const1.c: In function ‘main’:
const1.c:5:4: error: increment of read-only variable ‘x’
```

```
    5 |   x++;
      |    ^~
```

Το λάθος πιάνεται **κατά τη μεταγλώττιση**, πριν τρέξει οτιδήποτε. Αυτό είναι το μεγάλο πλεονέκτημα του `const`.

§23.17 Εγγραφή σε const μνήμη

Το `const2.c` παρακάμπτει τον έλεγχο με ένα cast (θέμα «Ο προσδιοριστής const»):

```
const char message[] = "Hello";
int main() {
  const int x = 42;
  char * bad = (char*)message;
  bad[1] = 'o';
  return 0;
}
```

```
$ gcc -o const2 const2.c
$ ./const2
Segmentation fault
```

Το `(char*)` λέει στον μεταγλωττιστή «εμπιστέψου με», οπότε η μεταγλώττιση περνά. Ο πίνακας `message` όμως βρίσκεται σε μνήμη μόνο για ανάγνωση, και η εγγραφή `bad[1] = 'o'` σκοτώνει το πρόγραμμα. Ένα cast που αφαιρεί το `const` είναι σχεδόν πάντα λάθος.

Κύρια σημεία

1. Οι γραμμές κώδικα (LOC, KLOC, MLOC) είναι μια βασική μετρική για το μέγεθος ενός προγράμματος· τα πραγματικά συστήματα έχουν από χιλιάδες ως δεκάδες εκατομμύρια γραμμές.
2. Όλος ο κώδικας σε ένα αρχείο είναι απλός, αλλά σε μεγάλα προγράμματα κάνει την αναζήτηση, τη συντήρηση και τη μεταγλώττιση αργές και δύσκολες.
3. Η αφαίρεση και η διάσπαση σε υποπροβλήματα οδηγούν σε πολλά αρχεία: τα `.c` περιέχουν την υλοποίηση και τα `.h` τη διεπαφή.
4. Ένα `.c` που υλοποιεί μια διεπαφή και ένα `.c` που τη χρησιμοποιεί εξαρτώνται μόνο από το `.h`· οι εξαρτήσεις σχηματίζουν γράφο που δείχνει τι επηρεάζει μια αλλαγή.
5. Ο `gcc` περνά από προεπεξεργασία, μεταγλώττιση σε αντικειμενικό αρχείο και σύνδεση.
6. Με `gcc -c` κάθε `.c` γίνεται χωριστά `.o`· η σύνδεση ταιριάζει τα απροσδιόριστα σύμβολα (`U`) με τους ορισμούς τους (`T`).
7. Η χωριστή μεταγλώττιση γλιτώνει χρόνο, γιατί μετά από μια αλλαγή ξαναφτιάχνουμε μόνο ό,τι χρειάζεται· αυτό αυτοματοποιεί το `make`.
8. Η μεταγλώττιση στη C είναι γραμμική: μια συνάρτηση πρέπει να έχει δηλωθεί με πρωτότυπο (όνομα, τύπος επιστροφής, ορίσματα) πριν κληθεί· τα ονόματα των

ορισμάτων παραλείπονται.

9. Το `const` δηλώνει ότι μια θέση μνήμης δεν αλλάζει· ο μεταγλωττιστής απορρίπτει την αλλαγή, και η παράκαμψή του με cast μπορεί να δώσει `Segmentation fault`.
10. Το `const` στις παραμέτρους μιας συνάρτησης είναι συμβόλαιο με τους χρήστες της και βοηθά και τον προγραμματιστή και τον μεταγλωττιστή.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
γραμμή κώδικα	line of code (LOC / SLOC)	Εντολές μέχρι την αλλαγή γραμμής· μετρική μεγέθους.
αφαίρεση	abstraction	Περιγραφή του τι κάνει ένα κομμάτι, κρύβοντας το πώς.
αρχείο υλοποίησης	implementation file (`.c`)	Περιέχει τον κώδικα των συναρτήσεων.
αρχείο κεφαλίδας	header file (`.h`)	Περιέχει δηλώσεις: τη διεπαφή ενός κομματιού.
διεπαφή	interface	Τι προσφέρει ένα κομμάτι του προγράμματος στα άλλα.
εξάρτηση	dependency	Ένα αρχείο χρειάζεται ένα άλλο για να χτιστεί.
αντικειμενικό αρχείο	object file (`.o`)	Αποτέλεσμα μεταγλώττισης ενός `.c` με `gcc -c`.
σύνδεση	linking	Ένωση αντικειμενικών αρχείων σε εκτελέσιμο.
πρωτότυπο συνάρτησης	function prototype	Όνομα, τύπος επιστροφής και ορίσματα, χωρίς σώμα.
δήλωση / ορισμός	declaration / definition	Ενημερώνει για τον τύπο / δεσμεύει μνήμη ή δίνει κώδικα.
include guard	include guard	`#ifndef`/`#define`/`#endif` γύρω από ένα `.h`.
προσδιοριστής const	const qualifier	Δηλώνει ότι μια θέση μνήμης δεν αλλάζει.
συμβόλαιο	contract	Εγγύηση που δίνει η δήλωση μιας συνάρτησης στους χρήστες της.

Διάβασμα

- **Διαφάνειες**: Διάλεξη 23, σελ. 1–27: γραμμές κώδικα 5–10· ένα ή πολλά αρχεία 11–14· εξαρτήσεις και χωριστή μεταγλώττιση 15–18· πρωτότυπα 19–21· `const` 22–25· αναφορές 26.

- **Σημειώσεις**: η διάλεξη σημειώνει ότι με αυτήν έχει καλυφθεί όλη η ύλη των διαφανειών του κ. Σταματόπουλου. Για τα θέματα της διάλεξης:
  - Κεφάλαιο 4: «Δομή ενός προγράμματος C – Συναρτήσεις» (K04, σελ. 58–60) για τα πρωτότυπα· «Εμβέλεια και χρόνος ζωής μεταβλητών» (63–68) για `extern` και `static` σε πολλά αρχεία.
  - Κεφάλαιο 2: «Μεταβλητές, σταθερές, τύποι και δηλώσεις στην C» (K04, σελ. 30–33) για το `const`.
  - Κεφάλαιο 6: «Συμβολοσειρές» (K04, σελ. 93–96) για τις παραμέτρους `const char *` της βιβλιοθήκης.
  - Κεφάλαιο 10: «Ο προεπεξεργαστής της C» (K04, σελ. 154–159) για το `#include` και το `#ifndef`.
  - Κεφάλαιο 12: «Ένα πρόγραμμα C πρέπει να είναι ...» (K04, σελ. 178).
- **Εργαστήριο**: Εργαστήριο 10: «Παράρτημα: Οργάνωση προγράμματος σε πολλαπλά αρχεία» και Άσκηση 5 (`more.c` σε `pager.c`/`pager.h`)· Εργαστήριο 1: ασκήσεις 9–11 (`gcc -c` και σύνδεση με το `myfunct.o`).
- **Άλλα**: Make (software) (Wikipedia)· openssl ως παράδειγμα οργάνωσης· Lines of Code Written· Linux Kernel surpasses 40MLOC· `man nm`.

Συχνά λάθη

- **Κλήση πριν από τη δήλωση**. `implicit declaration of function 'print_err'` (προειδοποίηση ή, στον gcc 14+, σφάλμα). Βάλτε πρωτότυπο πριν από την κλήση ή κάντε `#include` το `.h` που το περιέχει.
- **Ξεχασμένο `.o` στη σύνδεση**. `gcc -o out init.o` δίνει `undefined reference to 'print_err'`. Δώστε στη σύνδεση όλα τα `.o`.
- **Κώδικας ή ορισμός μεταβλητής μέσα σε `.h`**. Αν δύο `.c` κάνουν `#include` ένα `.h` με `int count = 0;`, η σύνδεση λέει `multiple definition of 'count'`. Στο `.h` βάλτε `extern int count;` και τον ορισμό σε ένα `.c`.
- **`#include` ενός `.c`**. Ο κώδικας αντιγράφεται σε δύο αρχεία και ορίζεται δύο φορές. Κάνετε `#include` μόνο `.h`.
- **`#include <err.h>` για δικό σας αρχείο**. `fatal error: err.h: No such file or directory`: τα δικά σας αρχεία θέλουν εισαγωγικά, `#include "err.h"`.
- **Αλλαγή μιας `const` μεταβλητής**. `error: increment of read-only variable 'x'`. Αν η τιμή πρέπει να αλλάζει, μην τη δηλώνετε `const`.
- **Cast για να αφαιρεθεί το `const`**. Το `(char*)message` μεταγλωττίζεται, αλλά η εγγραφή δίνει `Segmentation fault`.
- **`const` στη λάθος θέση**. Το `const char const *p` δεν κάνει σταθερό τον δείκτη (`duplicate 'const'`)· για σταθερό δείκτη γράψτε `char * const p`.

Ερωτήσεις κατανόησης

- **E23.1** Αναφέρετε ένα θετικό και δύο αρνητικά του να είναι όλος ο κώδικας σε ένα αρχείο.[178]
- **E23.2** Τι περιέχει ένα αρχείο `.h` και τι ένα `.c`;[179]
- **E23.3** Τι σημαίνουν τα `T` και `U` στην έξοδο του `nm` και τι κάνει μαζί τους η σύνδεση;[180]
- **E23.4** Αλλάζετε μόνο το σώμα μιας συνάρτησης στο `err.c`. Ποια αρχεία πρέπει να ξαναμεταγλωττιστούν;[181]
- **E23.5** Γιατί χρειάζεται πρωτότυπο μια συνάρτηση που ορίζεται κάτω από τη `main`;[182]
- **E23.6** Ποια είναι η διαφορά ανάμεσα σε `const char *p` και `char * const p`;[183]

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A23.1–A23.4)

- **A23.1** Κλήση πριν από τον ορισμό: Διάλεξη 23, διαφάνειες 19-21 · ★☆☆ · debug · `slides-lec23-implicit-declaration`
- **A23.2** Ένα σύστημα 40.000 γραμμών: Διαλέξεις 23–24, διαφάνειες 11 και 14 (διάλεξη 24: διαφάνειες 12 και 15) · ★☆☆ · short-answer · `slides-lec23-organize-40kloc`
- **A23.3** Αλλάζοντας κάτι const: Διάλεξη 23, διαφάνειες 23-24 · ★★☆ · trace · `slides-lec23-const-violations`
- **A23.4** Εξαρτήσεις ανάμεσα σε αρχεία: Διαλέξεις 23–24, διαφάνεια 15 (διάλεξη 24: διαφάνεια 16) · ★★☆ · short-answer · `slides-lec23-dependencies`

Εργαστήριο (A23.5)

- **A23.5** Σπάστε το πρόγραμμά σας σε αρθρώματα: Εργαστήριο 10, Άσκηση 5 · ★★☆ · tooling · `lab-lab10-more-modules`

Εργασίες (A23.6)

- **A23.6** Η Newton-Raphson Ξαναχτυπά! (Bonus): Εργασία 3 (2023-24), Άσκηση 2 (Bonus) και 2.1 (Bonus) · ★★★ · programming · `hw-2023-hw3-fractal`

[178] Θετικό: απλή οργάνωση και μεταφορά. Αρνητικά: δύσκολη αναζήτηση και συντήρηση· κάθε αλλαγή ξαναμεταγλωττίζει τα πάντα.

[179] Το `.h` δηλώσεις (πρωτότυπα, τη διεπαφή)· το `.c` τον κώδικα των συναρτήσεων.

[180] `T`: σύμβολο ορισμένο στον κώδικα του αρχείου· `U`: σύμβολο που χρησιμοποιείται αλλά ορίζεται αλλού. Η σύνδεση ταιριάζει κάθε `U` με ένα `T`.

[181] Μόνο το `err.c` σε `err.o`, και μετά ξανά η σύνδεση.

[182] Ο μεταγλωττιστής διαβάζει γραμμικά και στην κλήση πρέπει να ξέρει ήδη τον τύπο επιστροφής και τα ορίσματα.

[183] Στο πρώτο δεν αλλάζουν οι χαρακτήρες (ο δείκτης αλλάζει)· στο δεύτερο δεν αλλάζει ο δείκτης (οι χαρακτήρες αλλάζουν).

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A22.16** Νέα Μηχανή Σκακιού (chess engine): Εργασία 3 (2024-25), Άσκηση 1 · ★★★ · programming · `hw-2024-hw3-chess`
- **A22.17** Νέα Μηχανή Go (goteam): Εργασία 3 (2025-26), Άσκηση 1 · ★★★ · programming · `hw-2025-hw3-goteam`
- **A25.4** Ανελκυστήρες για Ανυπόμονους και Ανυπόμονες (elevate): Εργασία 2 (2025-26), Άσκηση 1 · ★★★ · programming · `hw-2025-hw2-elevate`

Κεφάλαιο 24: Προχωρημένα Θέματα

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να εξηγείτε γιατί τα μεγάλα προγράμματα σπάνε σε πολλά αρχεία `.c` και `.h` και πώς αυτά μεταγλωττίζονται και συνδέονται· να γράφετε πρωτότυπα συναρτήσεων και δηλώσεις `extern`· να χρησιμοποιείτε το `const` ως συμβόλαιο· να δηλώνετε και να καλείτε δείκτες σε συναρτήσεις και variadic συναρτήσεις· και να αναγνωρίζετε τα `volatile`, `register`, `restrict`, `auto`, τις δυαδικές σταθερές, το IEEE 754, τις `exec*` / `system` και τις βασικές εντολές του `gdb`.

Προαπαιτούμενα: Κεφάλαιο 3, Κεφάλαιο 14, Κεφάλαιο 23

Χρόνος μελέτης: ~2,5 ώρες

Σύνοψη

Η τελευταία θεωρητική διάλεξη πριν τα Χριστούγεννα ξεκινά από το μέγεθος των προγραμμάτων: από το Game of Life με μερικές εκατοντάδες γραμμές ως τα Windows Vista με 50 εκατομμύρια. Ένα σύστημα δεκάδων χιλιάδων γραμμών δεν χωρά λογικά σε ένα αρχείο, οπότε το σπάμε σε αρχεία υλοποίησης (`.c`) και αρχεία κεφαλίδας (`.h`), τα μεταγλωττίζουμε χωριστά και τα συνδέουμε. Αυτό απαιτεί δηλώσεις: πρωτότυπα συναρτήσεων, `extern` για μεταβλητές, `const` για να υποσχεθούμε τι δεν αλλάζει. Το δεύτερο μισό είναι μια περιήγηση σε «προχωρημένα» εργαλεία της C: δείκτες σε συναρτήσεις, variadic συναρτήσεις όπως η `printf`, δυαδικές σταθερές, το πρότυπο IEEE 754, εκτέλεση άλλων προγραμμάτων, οι προσδιοριστές `volatile`, `register`, `restrict` και `auto`, και ο debugger `gdb`.

Θεωρία

§24.1 Γραμμές κώδικα (LOC)

Μια βασική μετρική για το μέγεθος ενός προγράμματος είναι οι **γραμμές κώδικα (Lines of Code, LOC ή Source Lines of Code, SLOC)**: οι εντολές που γράφουμε μέχρι την αλλαγή γραμμής. Για μεγάλα μεγέθη χρησιμοποιούμε πολλαπλάσια: 1 KLOC = 10^3 LOC και 1 MLOC = 10^6 LOC.

Σύστημα	Έτος	Μέγεθος
Conway's Game of Life	1970	~100–200 LOC
Λογισμικό του Apollo 11	1969	145 KLOC
Windows Vista	2006	50 MLOC

Το Game of Life δείχνει ότι με λίγες γραμμές κώδικα παίρνουμε ιδιαίτερα σύνθετη συμπεριφορά. Τα άλλα δύο δείχνουν ότι τα πραγματικά συστήματα είναι τάξεις μεγέθους μεγαλύτερα από τις ασκήσεις μας, και αυτό αλλάζει τον τρόπο που τα οργανώνουμε.

§24.2 Όλος ο κώδικας σε ένα αρχείο;

Ας πούμε ότι φτιάχνουμε ένα νέο σύστημα και περιμένουμε περίπου 40.000 γραμμές κώδικα. Η πρώτη λύση είναι να τα βάλουμε όλα σε ένα αρχείο `.c`.

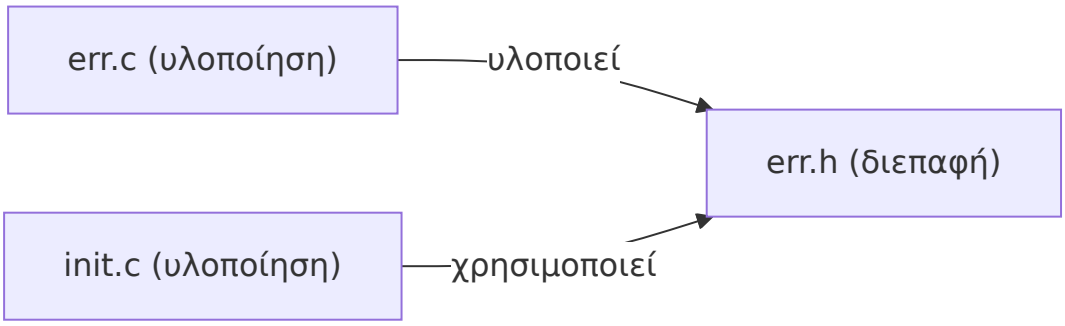
Θετικά	Αρνητικά
Απλή οργάνωση, εύκολη μεταφορά: όλος ο κώδικας σε ένα μέρος. Οι ορισμοί όλων των συναρτήσεων είναι προσβάσιμοι στο ίδιο αρχείο.	Το ψάξιμο σε αρχείο με δεκάδες χιλιάδες γραμμές είναι οδυνηρό. Αλλάζετε μία γραμμή και πρέπει να μεταγλωττίσετε τα πάντα. Η συντήρηση, η αναβάθμιση και η κατανόηση όλου του προγράμματος γίνονται δύσκολες.

§24.3 Αφαίρεση και οργάνωση σε πολλά αρχεία

Η ιδέα που λύνει το πρόβλημα είναι η **αφαίρεση (abstraction)** και η **διάσπαση σε υποπροβλήματα**: χωρίζουμε το σύστημα σε κομμάτια, και καθένα κρύβει τις λεπτομέρειές του πίσω από ένα απλό σύνολο συναρτήσεων. Στην C αυτό σημαίνει **οργάνωση του κώδικα σε πολλά αρχεία**. Κάθε αρχείο περιέχει μεταβλητές και συναρτήσεις που σχετίζονται θεματικά, λειτουργικά ή με κάποιο άλλο κριτήριο. Για παράδειγμα, το αποθετήριο του openssl έχει έναν κατάλογο `ssl/` με αρχεία όπως `ssl_init.c`, `event_queue.c`, `ssl_err.c` και `sslerr.h`, κι έναν κατάλογο `test/` με αρχεία όπως `aborttest.c` και `acvp_test.c`. Τα **αρχεία υλοποίησης (`.c`)** έχουν τους ορισμούς συναρτήσεων και μεταβλητών· τα **αρχεία κεφαλίδας (header files, `.h`)** έχουν τις δηλώσεις που χρειάζονται τα άλλα αρχεία για να τις χρησιμοποιήσουν.

§24.4 Εξαρτήσεις: διεπαφή και υλοποίηση

Πάρτε τρία αρχεία. Το `err.c` ορίζει τη συνάρτηση `int print_err(char *msg)`, το `err.h` περιέχει μόνο τη δήλωσή της `int print_err(char *);`, και το `init.c` κάνει `#include "err.h"` και καλεί `print_err("hi")`. Το `err.h` είναι η **διεπαφή (interface)**: λέει *τι* προσφέρεται. Το `err.c` την **υλοποιεί** και το `init.c` τη **χρησιμοποιεί**.

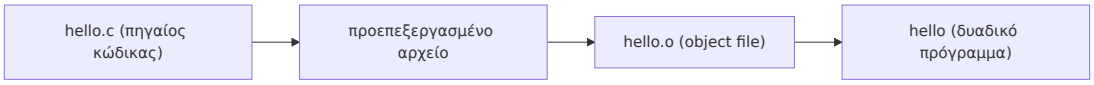


Σχήμα: οι εξαρτήσεις (dependencies) ανάμεσα σε δύο αρχεία υλοποίησης και μια διεπαφή.

Οι **εξαρτήσεις (dependencies)** απαντούν στο «αν αλλάξω ένα αρχείο, τι επηρεάζεται;». Αν αλλάξει το σώμα της `print_err` στο `err.c`, το `init.c` δεν χρειάζεται καμία αλλαγή, αφού ξέρει μόνο τη διεπαφή. Αν όμως αλλάξει το `err.h`, επηρεάζονται όλα τα αρχεία που το κάνουν `#include`. Τα αρχεία και οι εξαρτήσεις τους σχηματίζουν έναν γράφο.

§24.5 Μεταγλώττιση με πολλά αρχεία

Ο `gcc` περνά τον πηγαίο κώδικα από τρία στάδια: τον προεπεξεργαστή, που βγάζει το **προεπεξεργασμένο αρχείο (preprocessed file)**· τη μεταγλώττιση, που βγάζει το **αντικειμενικό αρχείο (object file)**· και τη **σύνδεση (linking)**, που βγάζει το τελικό **δυαδικό πρόγραμμα (binary)**.



Σχήμα: τα στάδια της μεταγλώττισης μέσα στον `gcc`.

Με πολλά αρχεία κάνουμε τα στάδια χωριστά. Η σημαία `-c` σταματά πριν τη σύνδεση και γράφει ένα `.o` για κάθε `.c` (`gcc -c -o err.o err.c`)· μια τελευταία κλήση, `gcc -o out init.o err.o`, **συνδέει** τα αντικειμενικά αρχεία. Κάθε αντικειμενικό αρχείο έχει έναν πίνακα συμβόλων. Στο `err.o` το `print_err` σημειώνεται `T` (ορίζεται εδώ, στον κώδικα), ενώ στο `init.o` σημειώνεται `U` (undefined: χρησιμοποιείται αλλά ορίζεται αλλού). Ο linker ταιριάζει κάθε `U` με ένα `T`, και στο `out` το `print_err` είναι πια `T`.

Η μεταγλώττιση είναι **χρονοβόρα διαδικασία**: σε μεγάλα έργα όπως ο πυρήνας του Linux μπορεί να πάρει ώρες (ή και μέρες!). Όταν το πρόγραμμα είναι σπασμένο σε αρχεία, μετά από κάθε αλλαγή ξαναμεταγλωττίζουμε μόνο ό,τι χρειάζεται (αυτό αυτοματοποιεί το make, βλ. Παράρτημα). Έτσι, μετά την πρώτη μεταγλώττιση, οι επόμενες είναι συνήθως πολύ γρηγορότερες.

§24.6 Πρωτότυπα συναρτήσεων

Η μεταγλώττιση στην C είναι **γραμμική διαδικασία**: ο μεταγλωττιστής διαβάζει το αρχείο από πάνω προς τα κάτω και, όταν συναντά μια κλήση, πρέπει να ξέρει ήδη πώς μοιάζει η συνάρτηση. Αν τον ορισμό τον βρει πιο κάτω (ή σε άλλο αρχείο), δίνει προειδοποίηση για **implicit declaration**.

Η λύση είναι η **δήλωση πρωτοτύπου (function prototype)**. Καθορίζει το **όνομα** της συνάρτησης, τον **τύπο επιστροφής** της και τα **ορίσματά** της, χωρίς σώμα:

```
τύπος όνομα(λίστα_ορισμάτων);
```

```
int print_err(char * message);
int print_err(char *);          // τα ονόματα των ορισμάτων παραλείπονται
```

Τα ονόματα των ορισμάτων μπορούν να παραλειφθούν: για να ελέγξει μια κλήση, ο μεταγλωττιστής χρειάζεται μόνο τους τύπους. Αυτές ακριβώς οι δηλώσεις γεμίζουν τα αρχεία κεφαλίδας.

§24.7 Ο προσδιοριστής `const`

Ο **προσδιοριστής** `const` **(const qualifier)** δηλώνει ότι το περιεχόμενο κάποιων θέσεων μνήμης είναι σταθερό. Η σύνταξη είναι `const δήλωση_μεταβλητής;`:

```
const int x = 42;
const char message[] = "Hello";
const char * const msg_ptr = message;
```

Η τρίτη δήλωση διαβάζεται από δεξιά προς τα αριστερά: το `msg_ptr` είναι σταθερός δείκτης (`* const`) προς σταθερούς χαρακτήρες (`const char`). Δεν αλλάζει ούτε ο δείκτης ούτε οι χαρακτήρες όπου δείχνει. Αν γράφαμε μόνο `const char *p`, θα μπορούσαμε να κάνουμε το `p` να δείχνει αλλού, αλλά όχι να αλλάξουμε τους χαρακτήρες μέσω του `p`.

Ό,τι δηλωθεί `const` **δεν** πρέπει να το αλλάξουμε. Ο μεταγλωττιστής το ελέγχει: το `x++` σε `const int` είναι σφάλμα μεταγλώττισης. Μπορούμε να «ξεγελάσουμε» τον μεταγλωττιστή με μετατροπή τύπου (cast) σε `char *`, αλλά αυτό δεν κάνει τη μνήμη εγγράψιμη: ένα καθολικό `const` συνήθως μπαίνει σε περιοχή μόνο για ανάγνωση, και η εγγραφή εκεί δίνει `Segmentation fault`.

Η μεγάλη χρησιμότητα του `const` είναι στις δηλώσεις συναρτήσεων: εκεί δημιουργεί **συμβόλαιο (contract)** με όσους καλούν τη συνάρτηση. Το

```
int print_err(const char * message);
```

εγγυάται ότι η `print_err` δεν θα αλλάξει τους χαρακτήρες της `message`. Τέτοιες εγγυήσεις κάνουν τους προγραμματιστές πιο αποδοτικούς (ξέρουν τι δεν θα πειραχτεί) και επιτρέπουν στους μεταγλωττιστές να γράφουν πιο γρήγορο κώδικα. Γι' αυτό η βιβλιοθήκη γράφει `strlen(const char *s)` και `strcpy(char *s1, const char *s2)` (Κεφάλαιο 14).

§24.8 Εξωτερικές μεταβλητές: `extern`

Ο **προσδιοριστής** `extern` μάς επιτρέπει να δηλώσουμε και να χρησιμοποιήσουμε μια μεταβλητή που **ορίζεται σε άλλο αρχείο**. Ο ορισμός (που δεσμεύει τη μνήμη) γίνεται σε ακριβώς ένα αρχείο· οι υπόλοιποι βλέπουν μια δήλωση `extern`, συνήθως μέσω της κεφαλίδας:

Αρχείο	Περιεχόμενο	Ρόλος
`err.c`	`int error_counter;`	ορισμός: εδώ δεσμεύεται η μνήμη
`err.h`	`extern int error_counter;`	δήλωση: «υπάρχει κάπου αλλού»
`init.c`	`#include "err.h"` … `error_counter++;`	χρήση

Το `extern` κάνει για τις μεταβλητές ό,τι κάνει το πρωτότυπο για τις συναρτήσεις. Οι καθολικές μεταβλητές, όμως, θέλουν φειδώ: όσο περισσότερα αρχεία τις αλλάζουν, τόσο δυσκολότερα καταλαβαίνει κανείς το πρόγραμμα (Κεφάλαιο 14 για εμβέλεια και `static`).

§24.9 Δείκτες σε συναρτήσεις

Όπως έχουμε δείκτες σε δεδομένα, μπορούμε να έχουμε και **δείκτες σε συναρτήσεις (function pointers)**. Ο κώδικας είναι κι αυτός δεδομένα στη μνήμη (δείτε τον με `objdump -d`), άρα έχει διεύθυνση. Η γενική μορφή είναι:

```
τύπος (*όνομα_δείκτη)(λίστα_ορισμάτων);
```

```
int (*myprint)(char *message);
```

Διαβάζεται: η `myprint` είναι δείκτης σε συνάρτηση που επιστρέφει `int` και παίρνει ένα όρισμα τύπου `char *`. Οι παρενθέσεις γύρω από το `*myprint` είναι απαραίτητες: το `int *myprint(char *);` θα δήλωνε *συνάρτηση* που επιστρέφει `int *`.

Στον δείκτη αναθέτουμε το όνομα μιας συνάρτησης με συμβατό τύπο (`myprint = print_err;`) και τον καλούμε σαν συνάρτηση, `myprint("...")`, ή ισοδύναμα `(*myprint)("...")`. Έτσι μια συνάρτηση επιλέγεται κατά την εκτέλεση: το `sorting.c` των σημειώσεων διαλέγει με ένα `switch` ποια συνάρτηση ταξινόμησης θα καλέσει μέσω του `void (*fun)(int, double *)`, και η `qsort` παίρνει τη συνάρτηση σύγκρισης ως όρισμα (Κεφάλαιο 17).

§24.10 Variadic συναρτήσεις

Συναρτήσεις με **μεταβαλλόμενο αριθμό ορισμάτων** λέγονται **variadic**. Τον μεταβλητό αριθμό ορισμάτων τον δηλώνουμε με την **έλλειψη (ellipsis)** `...`, μετά από τουλάχιστον ένα κανονικό όρισμα:

```
int printf(const char * format, ...);
int scanf(const char * format, ...);
```

Για να διατρέξουμε τα ορίσματα χρησιμοποιούμε τα «μαγικά» εργαλεία της `stdarg.h`:

- `va_list args;` είναι ο «δρομέας» πάνω στα ορίσματα·
- `va_start(args, last)` τον ξεκινά μετά το τελευταίο κανονικό όρισμα `last`·
- `va_arg(args, int)` επιστρέφει το επόμενο όρισμα, ερμηνευμένο ως `int`·
- `va_end(args)` τελειώνει τη διάσχιση.

Η συνάρτηση δεν μαθαίνει μόνη της πόσα ορίσματα πήρε ή τι τύπου είναι: πρέπει να της το πει κάποιο κανονικό όρισμα. Η `printf` το βρίσκει από το format string (ένα `%d` σημαίνει «επόμενο όρισμα `int`»)· το παράδειγμα `sum` παρακάτω παίρνει πρώτα το πλήθος. Όταν δεν ξέρετε τι κάνουν, `man stdarg`.

§24.11 Δυαδικές σταθερές

Εκτός από δεκαδικές (`42`) και δεκαεξαδικές (`0x2a`) σταθερές, η C2X (C23) δέχεται και **δυαδικές σταθερές (binary literals)** με πρόθεμα `0b`: το `0b101010` είναι $32+8+2=42$. Ο `gcc` τις δεχόταν ήδη ως επέκταση.

§24.12 Αριθμοί κινητής υποδιαστολής: IEEE 754

Με πεπερασμένο πλήθος bit (2^n, π.χ. 32 ή 64) μπορούμε να αναπαραστήσουμε μόνο ένα **υποσύνολο** των πραγματικών αριθμών. Συνήθως χρησιμοποιούμε το πρότυπο IEEE 754. Ένας `double` (64 bit) χωρίζεται έτσι:

Πεδίο	Bit	Ρόλος
Πρόσημο (sign)	1	0 για θετικό, 1 για αρνητικό
Εκθέτης (exponent)	11	η δύναμη του 2
Mantissa (significand)	52	τα σημαντικά ψηφία, ο αριθμός που πολλαπλασιάζεται με $2^{\text{εκθέτη}}$

Επειδή η mantissa έχει σταθερό πλήθος bit, η ακρίβεια είναι πεπερασμένη και πολλοί αριθμοί (π.χ. το 0.1) αποθηκεύονται κατά προσέγγιση (Κεφάλαιο 5). Με τον διαδραστικό μετατροπέα βλέπετε τα bit οποιουδήποτε `float`.

§24.13 Εκτέλεση άλλων προγραμμάτων: `exec*` και `system`

Οποιοδήποτε πρόγραμμα μπορούμε να τρέξουμε από τον φλοιό (shell) του Linux, μπορούμε να το τρέξουμε και από ένα πρόγραμμα C:

- Η οικογένεια `exec*` της `unistd.h` (π.χ. `execl`) **αντικαθιστά** το τρέχον πρόγραμμα με ένα άλλο. Στην `execl` δίνουμε τη διαδρομή του εκτελέσιμου και μετά τα ορίσματα που θα δει ως `argv` (πρώτο το `argv[0]`), τερματισμένα με `NULL`. Αν πετύχει, δεν επιστρέφει.

- Η `system` της `stdlib.h` δίνει ένα ολόκληρο string εντολής στον φλοιό και περιμένει να τελειώσει, όπως αν την πληκτρολογούσαμε.

§24.14 Οι προσδιοριστές `volatile`, `register`, `restrict` και `auto`

Οι διαφάνειες τους παρουσιάζουν ως type qualifiers· αυστηρά, τα `volatile` και `restrict` είναι qualifiers όπως το `const`, ενώ τα `register` και `auto` είναι storage-class specifiers. Και οι τέσσερις είναι λέξεις-κλειδιά της C.

- `volatile`: κάθε ανάγνωση και εγγραφή της μεταβλητής είναι **παρατηρήσιμη**. Ο compiler δεν επιτρέπεται να κρατήσει την τιμή σε register, να αφαιρέσει προσπελάσεις ή να αναδιατάξει reads/writes. Χρειάζεται όταν η τιμή αλλάζει εκτός ελέγχου του προγράμματος, όπως στο **memory-mapped I/O**, όπου μια «μεταβλητή» είναι στην πραγματικότητα καταχωρητής μιας συσκευής.
- `register`: υπόδειξη στον compiler να κρατήσει τη μεταβλητή σε **καταχωρητή (register)** της CPU. Απαγορεύει τη λήψη διεύθυνσης (`&var`). Οι σύγχρονοι compilers το αγνοούν, γιατί κάνουν καλύτερο register allocation μόνοι τους· οι διαφάνειες το σημειώνουν ως deprecated στη C23.
- `restrict` **(από τη C99)**: υπόδειξη ότι ένας δείκτης είναι ο **μόνος τρόπος πρόσβασης** στο αντικείμενο και καμία άλλη αναφορά δεν δείχνει στην ίδια μνήμη. Έτσι δηλώνεται η `strcpy`: `char * strcpy(char * restrict s1, const char * restrict s2);`. Αν περάσουμε τον ίδιο πίνακα και στα δύο ορίσματα, παραβιάζουμε την υπόσχεση· ο `gcc` με `-Wrestrict` το εντοπίζει.
- `auto`: δηλώνει **automatic storage duration** για μεταβλητές μέσα σε blocks. Είναι ήδη το default, οπότε το `auto int x = 42;` ισοδυναμεί με `int x = 42;` και σπάνια το βλέπουμε.

§24.15 Debugging με `gdb`

Ο **debugger** `gdb` τρέχει το πρόγραμμα ελεγχόμενα, ώστε να το σταματάμε και να κοιτάμε μέσα του. Τα βήματα:

1. Μεταγλώττιση με πληροφορίες αποσφαλμάτωσης: `gcc -g -ggdb -o prog prog.c`.
2. Εκκίνηση, με τα ορίσματα του προγράμματος: `gdb --args ./program arg1 arg2`.
3. Έλεγχος της εκτέλεσης: `run` (ξεκίνα), `break` (σημείο διακοπής σε συνάρτηση ή γραμμή), `step` (μία γραμμή, μπαίνοντας σε κλήσεις), `continue` (συνέχισε ως το επόμενο breakpoint), `finish` (τελείωσε την τρέχουσα συνάρτηση).
4. `backtrace`: η στοίβα κλήσεων, δηλαδή ποια συνάρτηση κάλεσε ποια ως εδώ.
5. `print`: η τιμή μιας μεταβλητής ή παράστασης.

Περισσότερες εντολές στο GDB Cheat Sheet.

Παραδείγματα

§24.16 Πόσες γραμμές γράψαμε στις εργασίες μας;

Το εργαλείο `sloccount` μετρά τις γραμμές κώδικα ενός καταλόγου και εκτιμά, με το μοντέλο COCOMO, το κόστος ανάπτυξης. Στις υποβολές των εργασιών του μαθήματος («Γραμμές κώδικα»):

```
$ sloccount hw-submissions/
Total Physical Source Lines of Code (SLOC)                = 119,059
Development Effort Estimate, Person-Years (Person-Months) = 30.24 (362.88)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 1.96 (23.48)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 15.46
Total Estimated Cost to Develop                           = $ 4,084,999
 (average salary = $56,286/year, overhead = 2.40).
```

Σχεδόν 120 KLOC, κοντά στο μέγεθος του λογισμικού του Apollo 11. Not bad.

§24.17 Η `print_err` σε τρία αρχεία

Το πλήρες παράδειγμα των «Εξαρτήσεις» και «Μεταγλώττιση με πολλά αρχεία». Το `err.c` υλοποιεί, το `err.h` δηλώνει, το `init.c` χρησιμοποιεί:

```c
// err.c
#include <stdio.h>
int print_err(char *msg) {
  return fprintf(stderr, "%s\n", msg);
}
```

```c
// err.h
int print_err(char *);
```

```c
// init.c (μεταγλωττίζεται μαζί με το err.h: does-not-compile μόνο του)
#include "err.h"
int main() {
  print_err("hi");
  return 0;
}
```

```
gcc -c -o err.o err.c
gcc -c -o init.o init.c
gcc -o out init.o err.o
```

Το `err.o` έχει το `print_err` ως `T` και το `init.o` ως `U` (αυτά τυπώνει το εργαλείο `nm`). Αν αλλάξετε μόνο το σώμα της `print_err`, ξαναφτιάχνετε μόνο το `err.o` και ξανασυνδέετε.

§24.18 Κλήση πριν τον ορισμό: το `prototype.c`

Εφαρμόζει τα «Πρωτότυπα συναρτήσεων». Η `main` καλεί την `print_err` πριν ο μεταγλωττιστής δει τον ορισμό της:

```c
#include <stdio.h>

int main() {
  print_err("hello");   // does-not-compile (σε gcc ≥ 14 είναι σφάλμα)
  return 0;
}
int print_err(char * msg) {
  return fprintf(stderr, "%s\n", msg);
}
```

```
$ gcc -o prototype prototype.c
prototype.c: In function ‘main’:
prototype.c:4:3: warning: implicit declaration of function ‘print_err’
[-Wimplicit-function-declaration]
    4 |   print_err("hello");
      |   ^~~~~~~~~
```

Η διόρθωση είναι μία γραμμή: το πρωτότυπο `int print_err(char *msg);` αμέσως μετά το `#include`. Τώρα η μεταγλώττιση είναι καθαρή:

```
$ gcc -o prototype prototype.c
$ ./prototype
hello
```

§24.19 Αλλάζοντας κάτι `const`

Εφαρμόζει τον «Ο προσδιοριστής `const`». Πρώτα ο μεταγλωττιστής αρνείται να αυξήσει μια `const` μεταβλητή (`const1.c`):

```c
const char message[] = "Hello";
int main() {
  const int x = 42;
  const char * const msg_ptr = message;
  x++;                  // does-not-compile
  return 0;
}
```

```
$ gcc -o const1 const1.c
const1.c: In function ‘main’:
const1.c:5:4: error: increment of read-only variable ‘x’
    5 |   x++;
      |    ^~
```

Με cast ο κώδικας μεταγλωττίζεται, αλλά η εγγραφή πέφτει σε μνήμη μόνο για ανάγνωση (`const2.c`):

```
const char message[] = "Hello";
int main() {
  const int x = 42;
  char * bad = (char*)message;
  bad[1] = 'o';
  return 0;
}
```

```
$ gcc -o const2 const2.c
$ ./const2
Segmentation fault
```

§24.20 Κλήση μέσω δείκτη σε συνάρτηση

Εφαρμόζει τους «Δείκτες σε συναρτήσεις» (`funcptr.c`):

```
#include <stdio.h>
int print_err(char * message) {
  return fprintf(stderr, "%s\n", message);
}
int main() {
  int (*myprint)(char *message);
  myprint = print_err;
  myprint("hello function pointer");  // ή: (*myprint)("...");
  return 0;
}
```

```
$ gcc -o funcptr funcptr.c
$ ./funcptr
hello function pointer
```

§24.21 Άθροισμα με variadic συνάρτηση

Εφαρμόζει τις «Variadic συναρτήσεις». Το πρώτο όρισμα λέει πόσοι αριθμοί ακολουθούν:

```
#include <stdarg.h>
int sum(int count, ...) {
  int result = 0;
  va_list args;
  va_start(args, count);
  for (int i = 0; i < count; i++)
    result += va_arg(args, int);
  va_end(args);
```

```
  return result;
}
int main() {
  return sum(6, 1, 2, 3, 4, 5, 6);
}
```

Το πρόγραμμα δεν τυπώνει τίποτα: επιστρέφει το άθροισμα ως κωδικό εξόδου, που τον βλέπουμε με `echo $?`:

```
$ ./variadic
$ echo $?
21
```

§24.22 Ο τυχερός αριθμός σε δυαδικό

Εφαρμόζει τις «Δυαδικές σταθερές». Τι θα τυπώσει;

```
#include <stdio.h>

int main() {
  int i = 0b101010;
  printf("My lucky number is: %d\n", i);
  return 0;
}
```

```
$ ./bin
My lucky number is: 42
```

§24.23 `execl` και `system`

Εφαρμόζει την «Εκτέλεση άλλων προγραμμάτων». Το πρώτο πρόγραμμα γίνεται το `ls`· το δεύτερο ζητά από τον φλοιό να τρέξει την `sort`, που διαβάζει γραμμές CSV από την είσοδο και τις ταξινομεί αριθμητικά (`-n`) με βάση το δεύτερο πεδίο (`-k2`), με διαχωριστικό το κόμμα (`-t,`):

```
#include <unistd.h>
int main() {
  execl("/bin/ls", "/bin/ls", NULL);
}
```

```
#include <stdlib.h>
int main() {
  system("sort -k2 -t, -n");
}
```

§24.24 `volatile`, `register`, `restrict`, `auto` στην πράξη

Εφαρμόζει την ενότητα των προσδιοριστών. Με `volatile` ο compiler πρέπει να ξαναδιαβάζει τον καταχωρητή κατάστασης μιας σειριακής θύρας (UART) σε κάθε επανάληψη· αλλιώς θα μπορούσε να τον διαβάσει μία φορά και ο βρόχος να μην τελειώσει ποτέ. Τα `register` και `auto` δεν αλλάζουν τίποτα σε σύγχρονο compiler:

```
#define UART_STATUS (*(volatile unsigned int *)0x40001000)

void wait_ready(void) {
    while ((UART_STATUS & 0x1) == 0) {
        /* compiler *must* re-read */
    }
}

void sum(const int *a, int n) {
    register int i;
    int s = 0;
    for (i = 0; i < n; i++)
        s += a[i];
}

void foo() {
    auto int x = 42;  /* equivalent to: int x = 42; */
    printf("%d\n", x);
}
```

Με `restrict`, ο gcc εντοπίζει ότι τα δύο ορίσματα της `strcpy` είναι ο ίδιος πίνακας:

```
#include <string.h>
int main() {
    char s[50] = "hello world";
    strcpy(s, s);
    return 0;
}
```

```
$ gcc -Wrestrict -o main main.c
main.c: In function ‘main’:
main.c:4:12: warning: passing argument 1 to ‘restrict’-qualified parameter
aliases with argument 2 [-Wrestrict]
    4 |     strcpy(s, s);
      |            ^  ~
```

Κύρια σημεία

1. Τα πραγματικά συστήματα μετριούνται σε KLOC και MLOC· σε ένα μόνο αρχείο θα ήταν αδύνατο να τα ψάξετε, να τα συντηρήσετε και να τα μεταγλωττίζετε.
2. Με αφαίρεση σπάμε το πρόγραμμα σε αρχεία: τα `.c` έχουν υλοποιήσεις και τα `.h` τη διεπαφή (δηλώσεις) που χρησιμοποιούν τα υπόλοιπα.
3. Με `gcc -c` κάθε `.c` γίνεται χωριστά `.o`, και ο linker ταιριάζει κάθε σύμβολο που χρησιμοποιείται (`U`) με τον ορισμό του (`T`)· μετά από μια αλλαγή ξαναμεταγλωττίζεται μόνο ό,τι επηρεάζεται.
4. Η μεταγλώττιση στην C είναι γραμμική, οπότε μια συνάρτηση δηλώνεται με πρωτότυπο (όνομα, τύπος επιστροφής, ορίσματα) πριν κληθεί.
5. Ό,τι δηλώνεται `const` δεν αλλάζει· σε παραμέτρους είναι συμβόλαιο με όσους καλούν τη συνάρτηση, και η παράκαμψή του με cast μπορεί να δώσει segmentation fault.
6. Το `extern` δηλώνει μια καθολική μεταβλητή που ορίζεται (μία φορά) σε άλλο αρχείο.
7. Ένας δείκτης σε συνάρτηση, τύπος `(*όνομα)(ορίσματα)`, κρατά τη διεύθυνση κώδικα και καλείται όπως η ίδια η συνάρτηση.
8. Οι variadic συναρτήσεις δηλώνονται με `...` και διατρέχουν τα ορίσματα με `va_list`, `va_start`, `va_arg`, `va_end`· πλήθος και τύπους τα μαθαίνουν από κάποιο κανονικό όρισμα.
9. Οι σταθερές `0b...` γράφουν ακεραίους σε δυαδικό· ένας `double` κατά IEEE 754 έχει 1 bit πρόσημο, 11 bit εκθέτη και 52 bit mantissa.
10. Με `exec*` και `system` ένα πρόγραμμα C τρέχει άλλα προγράμματα.
11. Το `volatile` απαγορεύει στον compiler να «βελτιστοποιήσει» προσπελάσεις, το `restrict` υπόσχεται ότι δεν υπάρχει aliasing, ενώ τα `register` και `auto` είναι σήμερα ουσιαστικά περιττά.
12. Ο `gdb` (με μεταγλώττιση `-g`) σταματά το πρόγραμμα (`break`), το προχωρά (`step`, `continue`, `finish`) και δείχνει τιμές (`print`) και τη στοίβα κλήσεων (`backtrace`).

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
γραμμές κώδικα	lines of code (LOC)	μετρική μεγέθους ενός προγράμματος
αφαίρεση	abstraction	απόκρυψη λεπτομερειών πίσω από απλή διεπαφή
αρχείο κεφαλίδας	header file (`.h`)	αρχείο με δηλώσεις που μοιράζονται πολλά αρχεία
διεπαφή	interface	το «τι» προσφέρει ένα κομμάτι κώδικα, χωρίς το «πώς»
εξάρτηση	dependency	«αν αλλάξει το A, επηρεάζεται το B»
αντικειμενικό αρχείο	object file (`.o`)	μεταγλωττισμένο `.c`, πριν τη σύνδεση
σύνδεση	linking	ένωση αντικειμενικών αρχείων σε εκτελέσιμο

Ελληνικά	English	Σύντομος ορισμός
πρωτότυπο συνάρτησης	function prototype	δήλωση ονόματος, τύπου επιστροφής και ορισμάτων
συμβόλαιο	contract	εγγύηση μιας συνάρτησης προς όσους την καλούν
δείκτης σε συνάρτηση	function pointer	μεταβλητή με τη διεύθυνση μιας συνάρτησης
variadic συνάρτηση	variadic function	συνάρτηση με μεταβλητό αριθμό ορισμάτων (`...`)
καταχωρητής	register	ταχύτατη θέση αποθήκευσης μέσα στη CPU
αποσφαλμάτωση	debugging	εντοπισμός και διόρθωση λαθών

Διάβασμα

- **Διαφάνειες**: Διάλεξη 24, σελ. 1–41: γραμμές κώδικα 6–12· οργάνωση σε αρχεία 13–16· μεταγλώττιση 17–19· πρωτότυπα 20–22· `const` 23–26· `extern` 27· δείκτες σε συναρτήσεις 28–29· variadic 30–31· δυαδικές σταθερές 32· IEEE 754 33· `exec`/`system` 34· `volatile`, `register`, `restrict`, `auto` 35–38· `gdb` 39.
- **Σημειώσεις**: η διάλεξη ζητά να διαβάσετε τις σελ. 64–68, 104 και 167–168 των διαφανειών του κ. Σταματόπουλου:
  - Κεφάλαιο 4: «Εμβέλεια και χρόνος ζωής μεταβλητών» (K04, σελ. 63–68), για `extern` και `static`.
  - Κεφάλαιο 6: «Δείκτες σε συναρτήσεις» (K04, σελ. 104).
  - Κεφάλαιο 11: «Μέθοδοι ταξινόμησης» (K04, σελ. 167–168), το `sorting.c` με δείκτη σε συνάρτηση.
  - Κεφάλαιο 12: «Ένα πρόγραμμα C πρέπει να είναι …» (K04, σελ. 178) και «Συχνά προγραμματιστικά λάθη στην C» (179–182).
- **Εργαστήριο**: Εργαστήριο 10, «Παράρτημα: Οργάνωση προγράμματος σε πολλαπλά αρχεία», άσκηση `more.c` (Άσκηση 5)· Εργαστήριο 7, «Ο debugger gdb», άσκηση `my_prog.c`· Εργαστήριο 8: `string.c` (πρωτότυπα με `const`) και `wages.c` (`gdb`).
- **Άλλα**: How much computer code has been written?· Variadic function (Wikipedia), Variadic functions και `va_start` (cppreference)· `const` και `extern` (GeeksforGeeks)· Function pointer και exec (Wikipedia)· IEEE 754 και διαδραστικός μετατροπέας· Type qualifiers: register, volatile, restrict· GDB Cheat Sheet· Make (Wikipedia)· `man stdarg`, `man 3 exec`, `man 3 system`.

Συχνά λάθη

- **Κλήση συνάρτησης πριν από τη δήλωσή της**. `warning: implicit declaration of function` (σε νεότερους gcc, σφάλμα). Βάλτε το πρωτότυπο πάνω από την κλήση ή κάντε `#include` την κεφαλίδα που το περιέχει.
- **Ορισμός μεταβλητής μέσα σε κεφαλίδα**. Το `int error_counter;` στο `err.h`, που το κάνουν `#include` δύο αρχεία, δίνει στη σύνδεση `multiple definition`. Στην κεφαλίδα μπαίνει το `extern int error_counter;` και ο ορισμός σε ένα `.c`.
- `extern` **χωρίς ορισμό πουθενά**. Ο linker αναφέρει `undefined reference to 'error_counter'`. Ορίστε τη μεταβλητή σε ακριβώς ένα `.c` και συνδέστε και το `.o` του.
- **Ξεχασμένο** `.o` **στη σύνδεση**. `gcc -o out init.o` χωρίς το `err.o` δίνει `undefined reference to 'print_err'`.
- **Παράκαμψη του** `const` **με cast**. Το `((char *)message)[1] = 'o';` μεταγλωττίζεται, αλλά δίνει `Segmentation fault`. Αν χρειάζεστε αλλαγές, μην δηλώνετε `const`.
- `int *f(char *)` **αντί για** `int (*f)(char *)`. Χωρίς παρενθέσεις δηλώνετε συνάρτηση που επιστρέφει δείκτη, όχι δείκτη σε συνάρτηση.
- **Λάθος πλήθος ή τύπος σε variadic κλήση**. Το `sum(6, 1, 2, 3)` ή ένα `%d` με όρισμα `double` στην `printf` διαβάζει σκουπίδια: η συνάρτηση δεν μπορεί να ελέγξει τι της δόθηκε.
- `gdb` **χωρίς** `-g`. Χωρίς πληροφορίες αποσφαλμάτωσης ο `gdb` δεν δείχνει γραμμές και ονόματα μεταβλητών. Ξαναμεταγλωττίστε με `-g`.

Ερωτήσεις κατανόησης

- **E24.1** Τι σημαίνουν τα `T` και `U` για το `print_err` στα `err.o` και `init.o`;[184]
- **E24.2** Τι δηλώνει το `const char * const p = message;`;[185]
- **E24.3** Ποια η διαφορά ανάμεσα σε `int (*f)(char *);` και `int *f(char *);`;[186]
- **E24.4** Πώς ξέρει η `printf` πόσα ορίσματα της δόθηκαν;[187]
- **E24.5** Γιατί ο καταχωρητής κατάστασης μιας συσκευής διαβάζεται μέσω `volatile`;[188]

[184] `T`: ορίζεται σε αυτό το αρχείο· `U`: χρησιμοποιείται αλλά ορίζεται αλλού, και τον ορισμό τον βρίσκει ο linker.

[185] Σταθερό δείκτη προς σταθερούς χαρακτήρες: δεν αλλάζει ούτε ο δείκτης ούτε ό,τι δείχνει.

[186] Το πρώτο είναι δείκτης σε συνάρτηση που επιστρέφει `int`· το δεύτερο δηλώνει συνάρτηση που επιστρέφει `int *`.

[187] Από το format string: κάθε προσδιοριστής όπως το `%d` αντιστοιχεί σε ένα επόμενο όρισμα.

[188] Η τιμή αλλάζει εκτός ελέγχου του προγράμματος· χωρίς `volatile` ο compiler μπορεί να τη διαβάσει μία φορά και να μη δει ποτέ την αλλαγή.

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A24.1)

- **A24.1** Ο τυχερός αριθμός σε δυαδικό: Διάλεξη 24: Προχωρημένα Θέματα, διαφάνεια 32 · ★☆☆ · trace · `slides-lec24-binary-literal`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A10.17** Εντοπισμός σφάλματος μνήμης με τον gdb: Εργαστήριο 7, Άσκηση 6 · ★☆☆ · debug · `lab-lab07-my_prog`
- **A11.18** Υπολογισμός μισθών: Εργαστήριο 8, Άσκηση 4 · ★★☆ · debug · `lab-lab08-wages`
- **A23.4** Εξαρτήσεις ανάμεσα σε αρχεία: Διαλέξεις 23–24, διαφάνεια 15 (διάλεξη 24: διαφάνεια 16) · ★★☆ · short-answer · `slides-lec23-dependencies`
- **A23.2** Ένα σύστημα 40.000 γραμμών: Διαλέξεις 23–24, διαφάνειες 11 και 14 (διάλεξη 24: διαφάνειες 12 και 15) · ★☆☆ · short-answer · `slides-lec23-organize-40kloc`

Κεφάλαιο 25: Επίλυση Προβλημάτων #3

Στόχοι: μετά από αυτό το κεφάλαιο θα μπορείτε να

- ξέρετε ποια ύλη καλύπτει το τελικό διαγώνισμα και ποιοι τύποι προβλημάτων εμφανίζονται στα θέματά του·
- αναγνωρίζετε σε ένα νέο πρόβλημα αν λύνεται με ανάγνωση κώδικα, γραμμικό πέρασμα, άπληστο αλγόριθμο, απομνημόνευση / δυναμικό προγραμματισμό ή αναδρομή·
- χρησιμοποιείτε την ταξινόμηση και τη δυαδική αναζήτηση ως δομικά στοιχεία μιας λύσης·
- τρέχετε ένα πρόγραμμα κάτω από το `valgrind` και να διαβάζετε τις αναφορές του για λάθη μνήμης και διαρροές.

Προαπαιτούμενα: Κεφάλαιο 15, Κεφάλαιο 16, Κεφάλαιο 17, Κεφάλαιο 22

Χρόνος μελέτης: ~2 ώρες (χωρίς τα παλιά θέματα)

Σύνοψη

Η τελευταία διάλεξη του εξαμήνου είναι διάλεξη επανάληψης πριν από το τελικό διαγώνισμα. Απαντά στο ερώτημα «τι ελέγχει το διαγώνισμα;»: όλη την ύλη, από τύπους και μεταβλητές μέχρι λίστες και δέντρα, μέσα από λίγους επαναλαμβανόμενους τύπους προβλημάτων (ανάγνωση κώδικα, γραμμικό πέρασμα, άπληστοι αλγόριθμοι, απομνημόνευση και δυναμικός προγραμματισμός, αναδρομή), με την ταξινόμηση και τη δυαδική αναζήτηση ως προαπαιτούμενα. Το κύριο μέρος ήταν ζωντανή επίλυση περσινών θεμάτων, και ακολούθησε προσκεκλημένη παρουσίαση για το `valgrind` από τον βοηθό του μαθήματος Γιώργο Σπύρου. Το κεφάλαιο συγκεντρώνει τις τεχνικές που χρειάζεστε για κάθε τύπο προβλήματος και σας παραπέμπει στα παλιά θέματα για εξάσκηση.

Θεωρία

§25.1 Ανακοινώσεις και το τέλος του εξαμήνου

Η διάλεξη άνοιξε με τα πρακτικά του τέλους του εξαμήνου:

- **Διαγώνισμα**: τα θέματα θα είναι παρεμφερή με αυτά των δύο προηγούμενων ετών. Τα παλιά θέματα είναι λοιπόν το καλύτερο υλικό επανάληψης.

- **Εργαστηριακή εξέταση**: 4 Φεβρουαρίου 2026, ή σε συνεννόηση με τον/την υπεύθυνο/η του εργαστηρίου σας· η ανακοίνωση θα σταλεί και στο piazza.
- **Επαναληπτικές διαλέξεις** για απορίες την επόμενη εβδομάδα (Δευτέρα στο Αμφιθέατρο, Παρασκευή στην Α2), για άρτιους *και* περιττούς.
- **Παράταση** για την Εργασία #2 και το Bonus #0 μέχρι τις 17 Ιανουαρίου, 23:59. Η προφορική εξέταση θα ακολουθήσει όπως και στην πρώτη υποβολή.
- **Διαγωνισμός Δημιουργικότητας**: η πρόσκληση για ψηφοφορία θα σταλεί από το piazza.

Η προηγούμενη διάλεξη (Κεφάλαιο 24) κάλυψε μεγάλα προγράμματα, δηλώσεις, το `const` και άλλα προχωρημένα θέματα· εδώ δεν προστίθεται νέα ύλη της C.

§25.2 Τι ελέγχει το διαγώνισμα

Το διαγώνισμα **καλύπτει όλη την ύλη**. Οι διαφάνειες την απαριθμούν σε δώδεκα ενότητες, που αντιστοιχούν στα κεφάλαια του οδηγού:

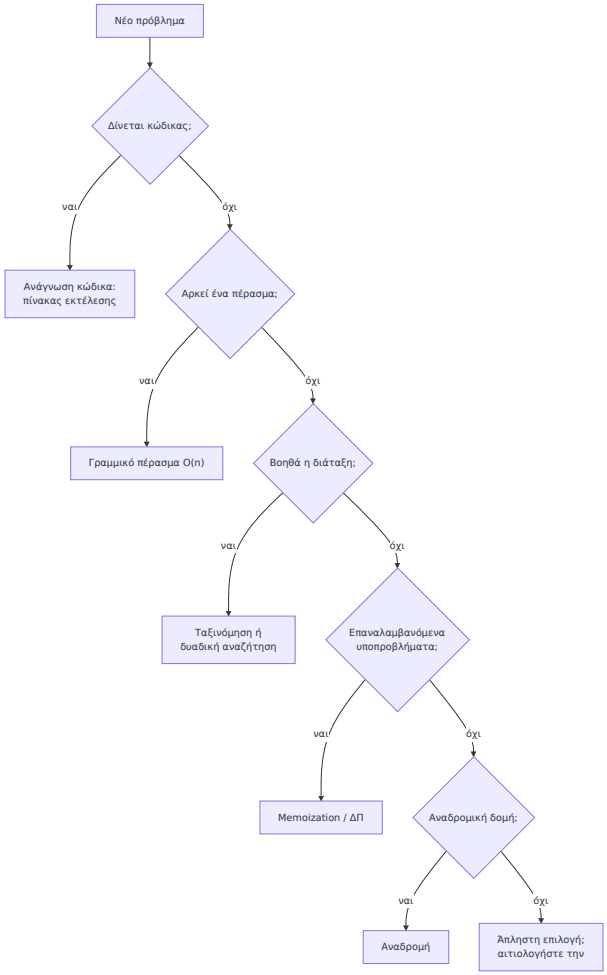
Ενότητα	Κεφάλαια
Τύποι / Μεταβλητές	2
Συναρτήσεις	3
Τελεστές, Εντολές, Ροή Ελέγχου	4, 5, 6, 8
Δεδομένα Εισόδου	9, 18
Πίνακες	10
Δείκτες και Διαχείριση Μνήμης	11, 12, 13, 14
Αναδρομή	11
Πολυπλοκότητα	15
Δυαδική Αναζήτηση	17
Ταξινόμηση	17, 18
Δομές	19, 20
Λίστες και Δέντρα	21, 22

Στα παλιά θέματα κάθε πρόβλημα προγραμματισμού ζητά σχεδόν πάντα και τη **χρονική και χωρική πολυπλοκότητα** της λύσης σας, με ένα σημαντικό μέρος της βαθμολογίας (π.χ. 6/20 ή 8/25 μονάδες). Η πολυπλοκότητα δεν είναι λοιπόν ξεχωριστό κεφάλαιο αλλά ερώτημα που συνοδεύει κάθε λύση. Επιπλέον, τα προγράμματα πρέπει να είναι δομημένα, ευανάγνωστα και τεκμηριωμένα.

§25.3 Τύποι προβλημάτων

Οι διαφάνειες ομαδοποιούν τα θέματα σε **τύπους προβλημάτων**. Το να αναγνωρίσετε τον τύπο είναι το πρώτο βήμα της λύσης, γιατί ο τύπος υποδεικνύει την τεχνική και, συχνά, και την πολυπλοκότητα που μπορείτε να πετύχετε.

Οι τύποι των διαφανειών είναι: ανάγνωση κώδικα, γραμμικό πέρασμα, άπληστοι αλγόριθμοι, memoization / δυναμικός προγραμματισμός, αναδρομικά, και γενική «επίλυση προβλημάτων» που τους συνδυάζει. Η ταξινόμηση και η δυαδική αναζήτηση σημειώνονται ως *προαπαιτούμενα*: σπάνια είναι το ζητούμενο, συνήθως είναι ένα βήμα μέσα στη λύση. Οι επόμενες ενότητες εξηγούν κάθε τύπο.



Σχήμα: ένας πρόχειρος οδηγός για το ποια τεχνική να δοκιμάσετε πρώτη· στην πράξη οι τεχνικές συνδυάζονται.

§25.4 Ανάγνωση κώδικα

Στα θέματα ανάγνωσης κώδικα (τα «Mystery» και οι «Η συνάρτηση …» των παλιών εξετάσεων) δεν γράφετε κώδικα, αλλά τον **εκτελείτε με το χέρι**. Η αξιόπιστη μέθοδος είναι ο **πίνακας εκτέλεσης** (trace table): μία στήλη για κάθε μεταβλητή και μία γραμμή για κάθε επανάληψη ή κλήση, όπου σημειώνετε τις τιμές *μετά* από κάθε εντολή και ό,τι τυπώνεται.

Τα θέματα αυτά ελέγχουν συνήθως λεπτομέρειες της γλώσσας που έχετε δει στα προηγούμενα

κεφάλαια:

- τελεστές bit, όπως το `x << 2` που ισούται με $4x$ για μη αρνητικό `x` (Κεφάλαιο 4)·
- ακέραια διαίρεση, υπόλοιπο και υπερχείλιση (Κεφάλαιο 2)·
- χαρακτήρες ως αριθμούς ASCII: ο πίνακας `{71, 111, 111, 100, 0}` είναι η συμβολοσειρά `"Good"`· το `0` στο τέλος είναι ο τερματικός χαρακτήρας (Κεφάλαιο 14)·
- αριθμητική δεικτών, `*p++`, και συναρτήσεις που επιστρέφουν μνήμη από `malloc` (Κεφάλαιο 12, Κεφάλαιο 13)·
- **μη αρχικοποιημένες μεταβλητές**: αν ο κώδικας τυπώνει μια τοπική μεταβλητή πριν της δοθεί τιμή, η σωστή απάντηση είναι «απροσδιόριστη τιμή», όχι «0».

Όταν το θέμα ρωτά και «τι κάνει η συνάρτηση», απαντήστε σε μία πρόταση με το *σκοπό* της (π.χ. «ενώνει δύο συμβολοσειρές σε νέα μνήμη»), όχι με περιγραφή κάθε γραμμής.

§25.5 Γραμμικό πέρασμα

Πολλά προβλήματα λύνονται διατρέχοντας τα δεδομένα **μία φορά** από την αρχή ως το τέλος, κρατώντας σε λίγες μεταβλητές ό,τι χρειάζεται: ένα άθροισμα, ένα μέγιστο, το προηγούμενο στοιχείο, έναν μετρητή. Η λύση έχει χρόνο $O(n)$ και συνήθως μνήμη $O(1)$ πέρα από την είσοδο. Κάθε λύση πρέπει να διαβάσει τουλάχιστον μία φορά την είσοδο, άρα για τέτοια προβλήματα το $O(n)$ είναι και το καλύτερο δυνατό.

Το ερώτημα που σχεδιάζει ένα γραμμικό πέρασμα είναι: *τι πρέπει να θυμάμαι από όσα έχω ήδη δει για να απαντήσω στο τέλος;* Μερικές παραλλαγές:

- **Δύο περάσματα**, όταν το δεύτερο χρειάζεται αποτέλεσμα του πρώτου (η τυπική απόκλιση θέλει πρώτα τον μέσο όρο). Ο χρόνος παραμένει $O(n)$.
- **Παράθυρο**: ο κινούμενος μέσος όρος με παράθυρο k χρησιμοποιεί μόνο τα τελευταία k στοιχεία.
- **Δύο δείκτες**: δύο θέσεις που κινούνται στα δεδομένα με διαφορετικούς κανόνες, π.χ. ένας αργός και ένας γρήγορος δείκτης για το μεσαίο στοιχείο μιας λίστας (Κεφάλαιο 21).

Τα δεδομένα έρχονται συχνά από τη γραμμή εντολών (`argv`, μετατροπή με `atoi`) ή από την πρότυπη είσοδο μέχρι το EOF (Κεφάλαιο 9).

§25.6 Άπληστοι αλγόριθμοι

Ένας **άπληστος αλγόριθμος** (greedy algorithm) χτίζει τη λύση βήμα-βήμα, κάνοντας σε κάθε βήμα την επιλογή που φαίνεται καλύτερη *εκείνη τη στιγμή*, χωρίς να την αναθεωρεί ποτέ. Είναι απλός και γρήγορος, αλλά **δεν δίνει πάντα τη βέλτιστη λύση**. Όταν τον χρησιμοποιείτε, το θέμα θέλει και αιτιολόγηση γιατί η άπληστη επιλογή δεν χάνει τίποτα.

Το κλασικό παράδειγμα είναι τα ρέστα με τα λιγότερα κέρματα: δίνουμε κάθε φορά το μεγαλύτερο κέρμα που χωρά. Με τα κέρματα του ευρώ αυτό είναι βέλτιστο. Με κέρματα $\{1, 3, 4\}$ και ποσό 6 όμως, ο άπληστος δίνει $4 + 1 + 1$ (τρία κέρματα), ενώ η βέλτιστη λύση είναι $3 + 3$ (δύο κέρματα).

Πολύ συχνά η άπληστη στρατηγική ξεκινά με **ταξινόμηση**: αν θέλουμε να ικανοποιήσουμε όσο το δυνατόν περισσότερα άτομα με περιορισμένη ποσότητα, εξυπηρετούμε πρώτα όσους ζητούν το λιγότερο. Η πολυπλοκότητα καθορίζεται τότε από την ταξινόμηση, $O(n \log n)$ με quicksort / mergesort.

§25.7 Memoization και δυναμικός προγραμματισμός

Όταν η αναδρομική λύση ενός προβλήματος καλεί **τα ίδια υποπροβλήματα ξανά και ξανά**, όπως η απλή αναδρομική Fibonacci (Κεφάλαιο 16), ο χρόνος γίνεται εκθετικός. Δύο συγγενικές τεχνικές το διορθώνουν:

- **Απομνημόνευση (memoization)**: κρατάμε την αναδρομική λύση, αλλά αποθηκεύουμε κάθε αποτέλεσμα σε έναν πίνακα την πρώτη φορά που υπολογίζεται. Στις επόμενες κλήσεις για το ίδιο όρισμα επιστρέφουμε την αποθηκευμένη τιμή.
- **Δυναμικός προγραμματισμός (dynamic programming, ΔΠ)**: γεμίζουμε τον πίνακα των υποπροβλημάτων **από κάτω προς τα πάνω**, από τα μικρότερα στα μεγαλύτερα, με έναν βρόχο αντί για αναδρομή.

Για να εφαρμόσετε ΔΠ χρειάζεστε μια **αναδρομική σχέση**: πώς η λύση για μέγεθος n προκύπτει από λύσεις μικρότερων μεγεθών. Για τα λιγότερα κέρματα:

$$best(a) = 1 + \min_{c \le a} best(a - c), \quad best(0) = 0$$

Για τους τρόπους να ανεβείτε μια σκάλα με βήματα 1, 2 ή 3 σκαλιών, $W(n) = W(n-1) + W(n-2) + W(n-3)$. Για τη διαδρομή ελαχίστου κόστους σε πλέγμα, όπου επιτρέπονται μόνο κινήσεις κάτω και δεξιά, το κόστος ενός κελιού είναι η τιμή του συν το μικρότερο από το κόστος του κελιού από πάνω και του κελιού από αριστερά.

Η πολυπλοκότητα της ΔΠ είναι (πλήθος υποπροβλημάτων) × (κόστος ανά υποπρόβλημα): για τα κέρματα $O(A \cdot k)$ χρόνος και $O(A)$ μνήμη, για ποσό A και k είδη κερμάτων.

Μια απλή μορφή της ίδιας ιδέας είναι ο **προϋπολογισμός** (precomputation): όταν έρχονται πολλές ερωτήσεις πάνω στα ίδια δεδομένα, πληρώνουμε μία φορά για έναν βοηθητικό πίνακα ώστε κάθε ερώτηση να απαντιέται γρήγορα. Με **προθεματικά αθροίσματα** (prefix sums), `pre[i] = a[0] + … + a[i-1]`, το άθροισμα οποιουδήποτε διαστήματος `a[l..r]` είναι `pre[r + 1] - pre[l]`: $O(n)$ μία φορά και $O(1)$ ανά ερώτηση, αντί για $O(n)$ ανά ερώτηση. Η ιδέα γενικεύεται σε δύο διαστάσεις, για αθροίσματα σε ορθογώνια περιοχή ενός πλέγματος.

§25.8 Ταξινόμηση και δυαδική αναζήτηση ως προαπαιτούμενα

Οι διαφάνειες χαρακτηρίζουν την ταξινόμηση και τη δυαδική αναζήτηση «προαπαιτούμενο»: πρέπει να τις ξέρετε τόσο καλά ώστε να τις χρησιμοποιείτε μέσα σε μια λύση χωρίς δεύτερη σκέψη (Κεφάλαιο 17).

- **Ταξινόμηση** σε $O(n \log n)$, με δική σας mergesort / quicksort ή με την `qsort` της `stdlib.h` και μια συνάρτηση σύγκρισης. Μετά την ταξινόμηση τα ίσα στοιχεία είναι γειτονικά (διπλότυπα σε ένα πέρασμα), και δύο δείκτες από τις δύο άκρες βρίσκουν ζεύγη με δοσμένο άθροισμα σε $O(n)$.
- **Δυαδική αναζήτηση** σε $O(\log n)$ σε **ταξινομημένο** πίνακα: συγκρίνουμε με το μεσαίο στοιχείο και κρατάμε το μισό όπου μπορεί να βρίσκεται η απάντηση. Αν ένα θέμα λέει ότι ο πίνακας είναι ταξινομημένος, αυτό είναι σχεδόν πάντα υπόδειξη για δυαδική αναζήτηση· μια σειριακή αναζήτηση $O(n)$ θα είναι σωστή αλλά όχι βέλτιστη.

Ένα πρόβλημα με n στοιχεία και τρεις εμφωλευμένους βρόχους ($O(n^3)$) γίνεται συχνά $O(n^2)$ ή $O(n \log n)$ αν πρώτα ταξινομήσετε.

§25.9 Αναδρομικά προβλήματα

Ένα πρόβλημα είναι **αναδρομικό** όταν η λύση του προκύπτει από λύσεις μικρότερων στιγμιοτύπων του ίδιου προβλήματος. Τα δέντρα είναι το πιο φυσικό παράδειγμα: ένα δέντρο είναι είτε κενό είτε ένας κόμβος με δύο υποδέντρα. Κάθε αναδρομική συνάρτηση χρειάζεται:

1. μια **βασική περίπτωση** που απαντά χωρίς αναδρομή (π.χ. `t == NULL`)·
2. **αναδρομικές κλήσεις** σε μικρότερα προβλήματα (τα υποδέντρα)·
3. έναν **συνδυασμό** των αποτελεσμάτων (άθροισμα, μέγιστο, `+ 1`).

Μια συνάρτηση που επισκέπτεται κάθε κόμβο ενός δέντρου n κόμβων μία φορά έχει χρόνο $O(n)$. Η χωρική της πολυπλοκότητα είναι το βάθος της στοίβας κλήσεων, δηλαδή το **ύψος** h του δέντρου: $O(\log n)$ για ισορροπημένο δέντρο, $O(n)$ στη χειρότερη περίπτωση (δέντρο-«αλυσίδα»). Στα θέματα αναφέρετε και τα δύο.

Άλλα αναδρομικά θέματα είναι η εξερεύνηση πλέγματος: γέμισμα περιοχής με χρώμα, εύρεση διαδρομής σε λαβύρινθο, έλεγχος περικύκλωσης. Εκεί η αναδρομή επισκέπτεται τα γειτονικά κελιά και χρειάζεται σήμανση των κελιών που έχει ήδη δει, αλλιώς δεν τερματίζει.

§25.10 Valgrind: έλεγχος της μνήμης

Το δεύτερο μέρος της διάλεξης ήταν προσκεκλημένη παρουσίαση για το valgrind από τον Γιώργο Σπύρου. Οι διαφάνειες της διάλεξης δεν περιέχουν την παρουσίαση· η ενότητα βασίζεται στο παράρτημα του Εργαστηρίου 9, που γράφτηκε από την ίδια παρουσίαση.

Το **valgrind** είναι συλλογή εργαλείων αποσφαλμάτωσης (debugging) και ανάλυσης επιδόσεων. Το πιο γνωστό του εργαλείο, το **Memcheck**, παρακολουθεί κάθε εντολή του προγράμματος που αφορά μνήμη, ενώ αυτό εκτελείται (dynamic program instrumentation). Το πρόγραμμα τρέχει 10 με 30 φορές πιο αργά, κάτι αδιάφορο για τα προγράμματα του μαθήματος. Μεταγλωττίζετε με `-g3`, ώστε οι αναφορές να δείχνουν αρχεία και γραμμές:

```
gcc -g3 -o prog prog.c
valgrind --leak-check=full ./prog
```

Το Memcheck εντοπίζει:

1. **παράνομες προσπελάσεις μνήμης**: ανάγνωση ή εγγραφή εκτός ορίων ενός μπλοκ ή σε μνήμη που έχει ήδη αποδεσμευτεί (`Invalid read` / `Invalid write`)·
2. **χρήση μη αρχικοποιημένης μνήμης** (`Conditional jump or move depends on uninitialised value(s)`)· η επιλογή `--track-origins=yes` δείχνει από πού προήλθε η τιμή·
3. **λάθη αποδέσμευσης**: διπλό `free`, ή `free` σε δείκτη που δεν ήρθε από `malloc`·
4. **διαρροές μνήμης** (memory leaks): μπλοκ του σωρού που δεν αποδεσμεύτηκαν ποτέ.

Η στοίβα καθαρίζει μόνη της όταν επιστρέφει μια συνάρτηση, ενώ ό,τι δεσμεύτηκε με `malloc` μένει δεσμευμένο μέχρι το `free`. Γι' αυτό οι διαρροές αφορούν τον σωρό, και γίνονται σοβαρότερες στις λίστες και στα δέντρα, όπου κάθε κόμβος είναι ξεχωριστό `malloc`. Το valgrind κατατάσσει τις διαρροές σε κατηγορίες:

Κατηγορία	Τι σημαίνει
`definitely lost`	Κανένας δείκτης δεν δείχνει πια στο μπλοκ: σίγουρη διαρροή.
`indirectly lost`	Το μπλοκ χάθηκε επειδή χάθηκε η δομή που έδειχνε σε αυτό.
`possibly lost`	Υπάρχει δείκτης, αλλά στη μέση του μπλοκ: σχεδόν πάντα λάθος.
`still reachable`	Δεν έγινε `free`, αλλά ο δείκτης υπήρχε ακόμη στο τέλος.

Διορθώνετε πάντα πρώτα τα `definitely lost`: όταν αποδεσμευτεί σωστά η κεφαλή μιας λίστας ή η ρίζα ενός δέντρου μαζί με ό,τι δείχνει, εξαφανίζονται και τα `indirectly lost`. Η μόνη καθαρή έξοδος είναι `All heap blocks were freed -- no leaks are possible` και `ERROR SUMMARY: 0 errors`.

Παραδείγματα

Τα παρακάτω παραδείγματα δείχνουν μία τεχνική το καθένα. Οι διαφάνειες δεν καταγράφουν ποια περσινά θέματα λύθηκαν στη ζωντανή συνεδρία· για εξάσκηση πάνω σε πραγματικά θέματα δείτε τον πίνακα στο τέλος της ενότητας.

§25.11 Πίνακας εκτέλεσης μιας συνάρτησης

Εφαρμογή της «Ανάγνωσης κώδικα». Τι τυπώνει το παρακάτω πρόγραμμα;

```c
#include <stdio.h>

int puzzle(int n) {
    int s = 1;
```

```
    for (int i = 0; i < 3; i++) {
        printf("%d %d\n", i, n - s);
        s = (s << 1) + 1;
    }
    return n + s;
}

int main(void) {
    printf("%d\n", puzzle(100));
    return 0;
}
```

Το `(s << 1) + 1` διπλασιάζει το s και προσθέτει 1. Ο πίνακας εκτέλεσης:

i	s πριν	τυπώνεται	s μετά
0	1	0 99	3
1	3	1 97	7
2	7	2 93	15

Μετά τον βρόχο η `puzzle` επιστρέφει $100 + 15 = 115$:

```
$ ./puzzle
0 99
1 97
2 93
115
```

§25.12 Μεγαλύτερη σειρά ίσων διαδοχικών τιμών

Εφαρμογή του «Γραμμικού περάσματος» σε ορίσματα γραμμής εντολών. Αρκεί να θυμόμαστε την προηγούμενη τιμή, το μήκος της τρέχουσας σειράς και το καλύτερο μήκος ως τώρα.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {
    if (argc < 2) {
        fprintf(stderr, "Usage: %s n1 n2 ...\n", argv[0]);
        return 1;
    }
    int best = 1, cur = 1;
    int prev = atoi(argv[1]);
    for (int i = 2; i < argc; i++) {
```

```
        int x = atoi(argv[i]);
        cur = (x == prev) ? cur + 1 : 1;
        if (cur > best)
            best = cur;
        prev = x;
    }
    printf("Longest run: %d\n", best);
    return 0;
}
```

```
$ ./run 3 3 5 5 5 2 2 7
Longest run: 4
```

Χρόνος $O(n)$ για n ορίσματα, μνήμη $O(1)$. Προσέξτε τον έλεγχο του `argc`: χωρίς αυτόν, το `argv[1]` χωρίς ορίσματα είναι `NULL` και η `atoi(NULL)` καταρρέει.

§25.13 Ρέστα: άπληστη λύση και δυναμικός προγραμματισμός

Εφαρμογή των «Άπληστων αλγορίθμων» και του «Δυναμικού προγραμματισμού» στο ίδιο πρόβλημα: τα λιγότερα κέρματα για ένα ποσό.

```
#include <stdio.h>

#define MAXA 1000
#define INF 1000000

int greedy(const int *c, int k, int amount) {
    int count = 0;
    for (int i = 0; i < k; i++) {   // c[] in descending order
        count += amount / c[i];
        amount %= c[i];
    }
    return count;
}

int dp(const int *c, int k, int amount) {
    int best[MAXA + 1];
    best[0] = 0;
    for (int a = 1; a <= amount; a++) {
        best[a] = INF;
        for (int i = 0; i < k; i++)
            if (c[i] <= a && best[a - c[i]] + 1 < best[a])
                best[a] = best[a - c[i]] + 1;
    }
    return best[amount];
```

```
}

int main(void) {
    int euro[] = {200, 100, 50, 20, 10, 5, 2, 1};
    int odd[] = {4, 3, 1};
    printf("euro 289: greedy %d, dp %d\n", greedy(euro, 8, 289),
           dp(euro, 8, 289));
    printf("{4,3,1} 6: greedy %d, dp %d\n", greedy(odd, 3, 6),
           dp(odd, 3, 6));
    return 0;
}
```

```
$ ./coins
euro 289: greedy 7, dp 7
{4,3,1} 6: greedy 3, dp 2
```

Για 289 λεπτά σε κέρματα ευρώ ο άπληστος δίνει $200 + 50 + 20 + 10 + 5 + 2 + 2$, που είναι και βέλτιστο. Για κέρματα $\{4, 3, 1\}$ ο άπληστος χάνει. Η `dp` γεμίζει τον πίνακα `best[]` από το 1 ως το ποσό· για τα κέρματα $\{4, 3, 1\}$:

a	0	1	2	3	4	5	6
best[a]	0	1	2	1	1	2	2

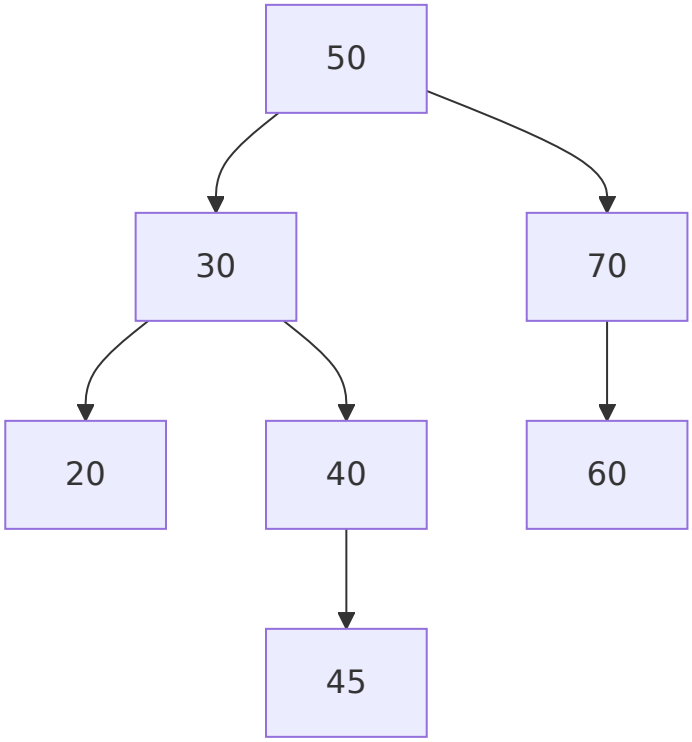
Η `greedy` κάνει $O(k)$ βήματα· η `dp` κάνει $O(A \cdot k)$ βήματα με $O(A)$ μνήμη. Ο πίνακας `best` είναι τοπικός, άρα η `dp` προϋποθέτει `amount <= MAXA`.

§25.14 Ύψος δέντρου και αποδέσμευση με αναδρομή

Εφαρμογή των «Αναδρομικών προβλημάτων» και της «Valgrind». Και οι δύο συναρτήσεις ακολουθούν το ίδιο σχήμα: βασική περίπτωση για το κενό δέντρο, κλήσεις στα δύο υποδέντρα, συνδυασμός.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
    int value;
    struct node *left, *right;
} *Tree;

int height(Tree t) {
    if (t == NULL)
        return 0;
```

```
    int hl = height(t->left), hr = height(t->right);
    return 1 + (hl > hr ? hl : hr);
}

void free_tree(Tree t) {
    if (t == NULL)
        return;
    free_tree(t->left);     // children first,
    free_tree(t->right);
    free(t);                // then the node itself
}

Tree node(int v, Tree l, Tree r) {
    Tree t = malloc(sizeof(*t));
    if (t == NULL)
        exit(1);
    t->value = v;
    t->left = l;
    t->right = r;
    return t;
}

int main(void) {
    Tree t = node(50,
                  node(30, node(20, NULL, NULL),
                           node(40, NULL, node(45, NULL, NULL))),
                  node(70, node(60, NULL, NULL), NULL));
    printf("height = %d\n", height(t));
    free_tree(t);
    return 0;
}
```



Σχήμα: το δέντρο που χτίζει η `main`*· η μακρύτερη διαδρομή 50 → 30 → 40 → 45 έχει 4 κόμβους.*

```
$ ./height
height = 4
```

Η `height` και η `free_tree` επισκέπτονται κάθε κόμβο μία φορά: χρόνος $O(n)$, μνήμη στοίβας $O(h)$. Η `free_tree` αποδεσμεύει τα παιδιά **πριν** από τον κόμβο (μεταδιατεταγμένα): αν κάνατε πρώτα `free(t)`, τα `t->left` και `t->right` θα διαβάζονταν από αποδεσμευμένη μνήμη, και το valgrind θα ανέφερε `Invalid read`. Αν παραλείψετε την κλήση `free_tree(t)` στη `main`, το valgrind αναφέρει τη ρίζα ως `definitely lost` και τους υπόλοιπους έξι κόμβους ως `indirectly lost`.

§25.15 Παλιά θέματα ανά τύπο προβλήματος

Αφού το διαγώνισμα έχει θέματα παρεμφερή με των δύο προηγούμενων ετών, λύστε τα παλιά θέματα σε συνθήκες εξέτασης (135 λεπτά για το θέμα Ιανουαρίου 2025) και μετά αναγνωρίστε σε ποιον τύπο ανήκει το καθένα:

Τύπος	Παλιά θέματα
Ανάγνωση κώδικα	Ιαν. 2025: «Mystery», «Η συνάρτηση dog»· Σεπ. 2025: «Mystery», «Η συνάρτηση transform»· Σεπ. 2024: «Η συνάρτηση about», «Η συνάρτηση what»
Γραμμικό πέρασμα	Ιαν. 2025: «Κινούμενος Μέσος Όρος - sma»· Ιούλ. 2024: «Στατιστικές»· Σεπ. 2025: «Μέση Τιμή Τυχαίων Μεταβλητών», «Μεσαίο Στοιχείο Λίστας»· Σεπ. 2024: «Μετρητής λέξεων»
Άπληστοι + ταξινόμηση	Ιούλ. 2024: «Βέλτιστη Μοιρασιά Πίτσας»· Δεκ. 2024: «Η Τριπλέτα Στόχος»
Memoization / ΔΠ	Σεπ. 2025: «Το Καλό το Μονοπάτι - path»· Ιαν. 2025: «Μετρώντας τα Αστέρια - stars» (αποδοτικές ερωτήσεις περιοχής)
Δυαδική αναζήτηση	Σεπ. 2024: «Εύρεση μηδενός σε πίνακα»
Αναδρομικά	Ιαν. 2025: «Αθροιστής Δέντρων - sumtree»· Ιούλ. 2024: «Reverse Inorder Traversal»· Σεπ. 2024: «Γεμίζοντας με χρώμα»· Δεκ. 2024: «Λύσε τον Λαβύρινθο»
Συμβολοσειρές, λίστες και μνήμη	Ιαν. 2025: «Συνένωση Αλφαριθμητικών - join»· Σεπ. 2024: «Αντιστροφή λίστας»

Η κατάταξη είναι του οδηγού, όχι των διαφανειών· πολλά θέματα συνδυάζουν τύπους. Οι εκφωνήσεις και υποδείξεις είναι στην ενότητα «Ασκήσεις» των αντίστοιχων κεφαλαίων.

Κύρια σημεία

1. Το διαγώνισμα καλύπτει όλη την ύλη, από τύπους και μεταβλητές μέχρι λίστες και δέντρα, και τα θέματά του είναι παρεμφερή με αυτά των δύο προηγούμενων ετών.
2. Σχεδόν κάθε πρόβλημα προγραμματισμού ζητά και τη χρονική και χωρική πολυπλοκότητα της λύσης, με σημαντικό μέρος της βαθμολογίας.
3. Τα θέματα ανήκουν σε λίγους τύπους: ανάγνωση κώδικα, γραμμικό πέρασμα, άπληστοι αλγόριθμοι, memoization / δυναμικός προγραμματισμός, αναδρομικά και γενική επίλυση προβλημάτων· η αναγνώριση του τύπου είναι το πρώτο βήμα της λύσης.
4. Στην ανάγνωση κώδικα χρησιμοποιήστε πίνακα εκτέλεσης και προσέξτε τελεστές bit, κωδικούς ASCII, αριθμητική δεικτών και μη αρχικοποιημένες μεταβλητές.
5. Ένα γραμμικό πέρασμα κρατά σε λίγες μεταβλητές ό,τι χρειάζεται από όσα έχει δει και δίνει $O(n)$ χρόνο, το καλύτερο δυνατό όταν πρέπει να διαβαστεί όλη η είσοδος.
6. Ένας άπληστος αλγόριθμος δεν αναθεωρεί τις επιλογές του και δεν είναι πάντα βέλτιστος· χρειάζεται αιτιολόγηση, και συχνά ξεκινά με ταξινόμηση.
7. Memoization και δυναμικός προγραμματισμός αποθηκεύουν τις λύσεις υποπροβλημάτων ώστε καθένα να υπολογίζεται μία φορά, μετατρέποντας εκθετικές λύσεις σε πολυωνυμικές.
8. Η ταξινόμηση ($O(n \log n)$) και η δυαδική αναζήτηση ($O(\log n)$) είναι προαπαιτούμενα: εργαλεία μέσα σε μεγαλύτερες λύσεις.

9. Μια αναδρομική συνάρτηση σε δέντρο χρειάζεται βασική περίπτωση για το `NULL` και έχει χρόνο $O(n)$ και μνήμη στοίβας $O(h)$.
10. Το `valgrind --leak-check=full` (με μεταγλώττιση `-g3`) εντοπίζει παράνομες προσπελάσεις, μη αρχικοποιημένες τιμές, λάθη αποδέσμευσης και διαρροές· πρώτα διορθώνετε τα `definitely lost`.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
πίνακας εκτέλεσης	trace table	Πίνακας με τις τιμές των μεταβλητών σε κάθε βήμα μιας εκτέλεσης με το χέρι.
γραμμικό πέρασμα	linear pass	Διάσχιση των δεδομένων μία φορά, σε χρόνο $O(n)$.
άπληστος αλγόριθμος	greedy algorithm	Αλγόριθμος που κάνει σε κάθε βήμα την τοπικά καλύτερη επιλογή χωρίς να την αναθεωρεί.
απομνημόνευση	memoization	Αποθήκευση των αποτελεσμάτων κλήσεων ώστε να μην ξαναϋπολογίζονται.
δυναμικός προγραμματισμός	dynamic programming	Επίλυση υποπροβλημάτων από τα μικρότερα στα μεγαλύτερα, με αποθήκευση των λύσεων σε πίνακα.
αναδρομική σχέση	recurrence	Τύπος που εκφράζει τη λύση ενός προβλήματος μέσω λύσεων μικρότερων στιγμιοτύπων.
προθεματικά αθροίσματα	prefix sums	Πίνακας με τα αθροίσματα των πρώτων i στοιχείων, για αθροίσματα διαστημάτων σε $O(1)$.
δύο δείκτες	two pointers	Δύο θέσεις που κινούνται στα δεδομένα για να αποφύγουν εμφωλευμένους βρόχους.
διαρροή μνήμης	memory leak	Μνήμη του σωρού που δεσμεύτηκε και δεν αποδεσμεύτηκε ποτέ.

Διάβασμα

- **Διαφάνειες**: Διάλεξη 25, σελ. 1–10: ανακοινώσεις σελ. 2· περσινή και σημερινή διάλεξη σελ. 3–4· τι ελέγχει το διαγώνισμα σελ. 5–6· τύποι προβλημάτων σελ. 7· ζωντανή συνεδρία σελ. 8.
- **Σημειώσεις**:
  - Κεφάλαιο 11: Ταξινόμηση και αναζήτηση, ενότητες «Ταξινόμηση πινάκων», «Μέθοδοι ταξινόμησης» (K04, σελ. 160–171), «Αναζήτηση σε πίνακες» και

«Μέθοδοι αναζήτησης» (K04, σελ. 172–177).
  - Κεφάλαιο 8: Συνδεδεμένες λίστες και δυαδικά δέντρα, ενότητες «Διαχείριση συνδεδεμένων λιστών» και «Διαχείριση δυαδικών δέντρων» (K04, σελ. 128–135).
  - Κεφάλαιο 12: Καλές πρακτικές, ενότητες «Ένα πρόγραμμα C πρέπει να είναι ...» και «Συχνά προγραμματιστικά λάθη στην C» (K04, σελ. 178–182).
- **Εργαστήρια**:
  - Εργαστήριο 9: «Παράρτημα: Αποσφαλμάτωση προγραμμάτων (Πράξη 5η)» για το valgrind και η Άσκηση 5 (`grades.c`, `tree.c`, καθαρή διαχείριση μνήμης).
  - Εργαστήριο 5: `fib.c` (αναδρομή και memoization) και `ladder.c` (δυναμικός προγραμματισμός).
- **Παλιά θέματα**: Ιανουάριος 2025, Σεπτέμβριος 2025, Ιούλιος 2024, Σεπτέμβριος 2024, Δεκέμβριος 2024.
- **Άλλα**: valgrind.org, `man 1 valgrind`, `man 3 qsort`.

Συχνά λάθη

- **Δίνετε μόνο κώδικα, χωρίς πολυπλοκότητα**. Το ερώτημα «ποια η χρονική και χωρική πολυπλοκότητα» έχει δικές του μονάδες. Γράψτε και τα δύο, με το σύμβολο O και τις μεταβλητές του προβλήματος (π.χ. $O(N^2 + Q)$), και μια πρόταση αιτιολόγησης.
- **«Η μη αρχικοποιημένη μεταβλητή είναι 0»**. Μια τοπική `int i;` που τυπώνεται πριν πάρει τιμή δίνει απροσδιόριστη τιμή. Το `valgrind` το αναφέρει ως `uninitialised value`.
- **Άπληστη λύση χωρίς αιτιολόγηση**. Η επιλογή «το μεγαλύτερο κέρμα πρώτα» αποτυγχάνει με κέρματα $\{1, 3, 4\}$ για το 6. Ελέγξτε με ένα μικρό αντιπαράδειγμα, και αν βρείτε, στραφείτε σε ΔΠ.
- **Σειριακή αναζήτηση σε ταξινομημένο πίνακα**. Σωστή αλλά $O(n)$, ενώ ζητείται η βέλτιστη λύση: χρησιμοποιήστε δυαδική αναζήτηση, $O(\log n)$.
- **Αναδρομή σε δέντρο χωρίς έλεγχο** `NULL`. Η `t->left` σε κενό δέντρο δίνει `Segmentation fault`. Η βασική περίπτωση `if (t == NULL)` μπαίνει πρώτη.
- `free(t)` **πριν από τα παιδιά**. Η `free_tree` που αποδεσμεύει πρώτα τον κόμβο και μετά διαβάζει το `t->left` προσπελαύνει αποδεσμευμένη μνήμη (`Invalid read` στο valgrind). Αποδεσμεύστε μεταδιατεταγμένα.
- `valgrind time ./prog`. Έτσι το valgrind ελέγχει την εντολή `time`, όχι το πρόγραμμά σας. Γράψτε `valgrind ./prog`.

Ερωτήσεις κατανόησης

- E25.1 Ποιες ενότητες της ύλης καλύπτει το τελικό διαγώνισμα;[189]

[189] Όλη την ύλη: τύπους και μεταβλητές, συναρτήσεις, τελεστές, εντολές και ροή ελέγχου, δεδομένα εισόδου, πίνακες, δείκτες και διαχείριση μνήμης, αναδρομή, πολυπλοκότητα, δυαδική αναζήτηση, ταξινόμηση, δομές, λίστες και δέντρα.

- **E25.2** Γιατί ένα πρόβλημα που πρέπει να διαβάσει όλα τα n στοιχεία της εισόδου δεν μπορεί να λυθεί γρηγορότερα από $O(n)$;[190]
- **E25.3** Δώστε ένα σύνολο κερμάτων και ένα ποσό για τα οποία η άπληστη λύση των ρέστων δεν είναι βέλτιστη.[191]
- **E25.4** Ποια η διαφορά ανάμεσα στο memoization και στον δυναμικό προγραμματισμό «από κάτω προς τα πάνω»;[192]
- **E25.5** Με προθεματικά αθροίσματα `pre[]`, πώς υπολογίζετε το άθροισμα του `a[l..r]` και σε τι χρόνο;[193]
- **E25.6** Ποια η χωρική πολυπλοκότητα μιας αναδρομικής διάσχισης δέντρου n κόμβων με ύψος h, και γιατί;[194]
- **E25.7** Τι σημαίνουν τα `definitely lost` και `indirectly lost` στην αναφορά του valgrind, και ποιο διορθώνετε πρώτα;[195]
- **E25.8** Γιατί η `free_tree` αποδεσμεύει τα υποδέντρα πριν από τον ίδιο τον κόμβο;[196]

Ασκήσεις

Εργαστήριο (A25.1)

- **A25.1** Καθαρή διαχείριση μνήμης: Εργαστήριο 9, Άσκηση 5 · ★★☆ · programming · `lab-lab09-grades-tree`

Εργασίες (A25.2–A25.4)

- **A25.2** DNA Matching: Εργασία 2 (2023-24), Άσκηση 2 · ★★★ · programming · `hw-2023-hw2-dna`
- **A25.3** Το Καλύτερο GPS (jabbamaps): Εργασία 2 (2024-25), Άσκηση 2 · ★★★ · programming · `hw-2024-hw2-jabbamaps`
- **A25.4** Ανελκυστήρες για Ανυπόμονους και Ανυπόμονες (elevate): Εργασία 2 (2025-26), Άσκηση 1 · ★★★ · programming · `hw-2025-hw2-elevate`

[190] Γιατί κάθε στοιχείο πρέπει να διαβαστεί τουλάχιστον μία φορά, και αυτό μόνο κοστίζει n βήματα· ένα γραμμικό πέρασμα είναι επομένως βέλτιστο.

[191] Κέρματα $\{1, 3, 4\}$ και ποσό 6: ο άπληστος δίνει $4 + 1 + 1$ (3 κέρματα), ενώ το $3 + 3$ θέλει 2.

[192] Το memoization κρατά την αναδρομή και αποθηκεύει κάθε αποτέλεσμα την πρώτη φορά που υπολογίζεται· ο δυναμικός προγραμματισμός γεμίζει τον πίνακα με βρόχο, από τα μικρότερα υποπροβλήματα προς τα μεγαλύτερα, χωρίς αναδρομή.

[193] `pre[r + 1] - pre[l]`, σε $O(1)$ χρόνο, αφού ο πίνακας `pre` έχει υπολογιστεί μία φορά σε $O(n)$.

[194] $O(h)$: κάθε ενεργή κλήση κρατά ένα πλαίσιο στη στοίβα, και οι ενεργές κλήσεις είναι όσες οι κόμβοι μιας διαδρομής από τη ρίζα. Για ισορροπημένο δέντρο $O(\log n)$, στη χειρότερη περίπτωση $O(n)$.

[195] `definitely lost`: κανένας δείκτης δεν δείχνει πια στο μπλοκ. `indirectly lost`: το μπλοκ χάθηκε επειδή χάθηκε η δομή που έδειχνε σε αυτό. Διορθώνετε πρώτα τα `definitely lost`, και τα `indirectly lost` εξαφανίζονται μαζί τους.

[196] Γιατί μετά το `free(t)` τα `t->left` και `t->right` βρίσκονται σε αποδεσμευμένη μνήμη και δεν επιτρέπεται να διαβαστούν.

Θέματα εξετάσεων (A25.5–A25.17)

- **A25.5** Σκαλί-Σκαλί: Κατατακτήριες Δεκεμβρίου 2023, Θέμα 3 · ★★☆ · programming · `exam-2023-dec-q3`
- **A25.6** Μένοντας στις Σωστές Θερμίδες: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #14, Θέμα 2 · ★★☆ · programming · `exam-2023-fall-ex14-q2`
- **A25.7** Ανεβαίνοντας Επίπεδο: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #2 (Pokémon Themed), Θέμα 4 · ★★☆ · programming · `exam-2023-fall-ex2-q4`
- **A25.8** Επενδύσεις στο Χρηματιστήριο: Εξέταση Ιουνίου 2026, Θέμα 3 · ★★☆ · programming · `exam-2026-jun-q3`
- **A25.9** Το Νερό Νεράκι: Κατατακτήριες Δεκεμβρίου 2023, Θέμα 4 · ★★★ · programming · `exam-2023-dec-q4`
- **A25.10** Τρόποι να Φάμε Παϊδάκια: Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #14, Θέμα 4 · ★★★ · programming · `exam-2023-fall-ex14-q4`
- **A25.11** Η Τριπλέτα Στόχος: Κατατακτήριες Δεκεμβρίου 2024, Θέμα 3 · ★★★ · programming · `exam-2024-dec-q3`
- **A25.12** Λύσε τον Λαβύρινθο: Κατατακτήριες Δεκεμβρίου 2024, Θέμα 4 · ★★★ · programming · `exam-2024-dec-q4`
- **A25.13** Βέλτιστη Μοιρασιά Πίτσας: Εξέταση Ιουλίου 2024, Θέμα 3 · ★★★ · programming · `exam-2024-jul-q3`
- **A25.14** Γεμίζοντας με χρώμα: Εξέταση Σεπτεμβρίου 2024, Θέμα 6 · ★★★ · programming · `exam-2024-sep-q6`
- **A25.15** Το Καλό το Μονοπάτι - path: Εξέταση Σεπτεμβρίου 2025, Θέμα 5 · ★★★ · programming · `exam-2025-sep-q5`
- **A25.16** Περικύκλωση - encirclement: Εξέταση Ιανουαρίου 2026, Θέμα 5 · ★★★ · programming · `exam-2026-jan-q5`
- **A25.17** Η Μεγαλύτερη Χωρητικότητα - capacity: Εξέταση Σεπτεμβρίου 2026, Θέμα 5 · ★★★ · programming · `exam-2026-sep-q5`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A10.19** Η συνάρτηση cons: Εξέταση Σεπτεμβρίου 2026, Θέμα 2 · ★☆☆ · trace · `exam-2026-sep-q2`

Παραρτήματα

Παράρτημα A: How to Make?

> **Στόχοι:** μετά από αυτό το παράρτημα θα μπορείτε να ξεχωρίζετε ένα σφάλμα μεταγλώττισης από ένα σφάλμα σύνδεσης· να χτίζετε ένα project πολλών αρχείων σε βήματα (`gcc -c` και σύνδεση)· να εξηγείτε γιατί ένα bash script δεν αρκεί· και να γράφετε ένα `Makefile` με rules, macros, automatic variables, wildcard rules και built-in rules.
>
> **Προαπαιτούμενα:** Κεφάλαιο 0 (preprocessor, compiler, linker), Κεφάλαιο 3 (`-lm`), Κεφάλαιο 23 (οργάνωση κώδικα σε αρχεία)
>
> **Χρόνος μελέτης:** ~1,5 ώρα

Σύνοψη

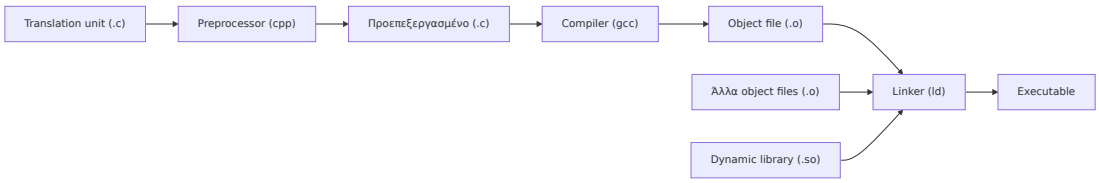
Η προσκεκλημένη διάλεξη, από τον Πέτρο, δευτεροετή φοιτητή και βοηθό του μαθήματος, παρουσιάζει το **Make**, το εργαλείο που αυτοματοποιεί την κατασκευή (build) ενός προγράμματος. Ξεκινά από τη διαδικασία μετάφρασης της C (preprocessor, compiler, linker) και από το πώς ξεχωρίζουμε τα λάθη κάθε σταδίου. Μετά δείχνει γιατί, μόλις ένα project έχει πολλά αρχεία, οι εντολές `gcc` γίνονται μακριές, πολλές και αργές, και γιατί ένα bash script λύνει μόνο τα δύο πρώτα προβλήματα. Το Make λύνει και το τρίτο: ξέρει τι εξαρτάται από τι και ξαναχτίζει μόνο ό,τι άλλαξε. Η διάλεξη κλείνει με τα εργαλεία που κάνουν ένα `Makefile` σύντομο: macros, automatic variables, wildcard rules και built-in rules.

Θεωρία

§26.1 Η διαδικασία μετάφρασης της C

Ένα εκτελέσιμο δεν βγαίνει από το `.c` σε ένα βήμα. Κάθε αρχείο `.c`, που λέγεται **translation unit** (μονάδα μετάφρασης), περνά από τρία στάδια (Κεφάλαιο 0):

1. ο **προεπεξεργαστής (preprocessor**, `cpp`**)** εκτελεί τις οδηγίες `#include`, `#define` κ.λπ. και βγάζει ένα προεπεξεργασμένο translation unit·
2. ο **μεταγλωττιστής (compiler**, `gcc`**)** το μεταφράζει σε ένα **object file** (αντικειμενικό αρχείο, `.o`) σε γλώσσα μηχανής·
3. ο **συνδέτης (linker**, `ld`**)** ενώνει όλα τα object files και τις **δυναμικές βιβλιοθήκες (dynamic libraries**, `.so`**)** σε ένα **εκτελέσιμο (executable)**.



Σχήμα: από το translation unit στο εκτελέσιμο (διαφάνεια 5).

Η εντολή `gcc main.c -o main` κάνει και τα τρία στάδια μαζί· το `gcc` στην ουσία καλεί με τη σειρά τα `cpp`, τον compiler και το `ld`.

§26.2 Σφάλματα μεταγλώττισης και σφάλματα σύνδεσης

Κάθε στάδιο έχει τα δικά του λάθη, και το να καταλαβαίνετε ποιο στάδιο παραπονιέται είναι το μισό της διόρθωσης.

- Ένα **σφάλμα μεταγλώττισης (compiler error)** σημαίνει ότι ο compiler δεν βρίσκει κάτι μέσα στο *translation unit*: για παράδειγμα τη **δήλωση** μιας συνάρτησης (`implicit declaration of function 'sqrt'`). Ο compiler βλέπει τον κώδικά σας, οπότε δίνει αριθμό γραμμής και συχνά την ακριβή διόρθωση («include <math.h>»).
- Ένα **σφάλμα σύνδεσης (linking error)** σημαίνει ότι ο linker δεν βρίσκει μέσα στο **linking scope**, δηλαδή στα object files και τις βιβλιοθήκες που του δώσατε, την **υλοποίηση** ενός συμβόλου (`undefined reference to 'sqrt'`). Το μήνυμα είναι λιγότερο βοηθητικό: ο linker δουλεύει πάνω σε object files, όχι σε πηγαίο κώδικα, οπότε δεν ξέρει σε ποια γραμμή σας γράψατε την κλήση ούτε ποια βιβλιοθήκη ξεχάσατε.

Ένα αρχείο επικεφαλίδας όπως το `math.h` δίνει μόνο **δηλώσεις**. Η υλοποίηση της `sqrt` βρίσκεται στη μαθηματική βιβλιοθήκη του συστήματος, `libm.so`, και πρέπει να πείτε στον linker να τη χρησιμοποιήσει με το flag `-l:libm.so`, ή σύντομα `-lm`.

§26.3 Projects με πολλά αρχεία

Τα πραγματικά projects αποτελούνται από πολλά αρχεία `.c`, που πρέπει να μεταγλωττιστούν με όμοιο τρόπο και να συνδεθούν. Το project της διάλεξης έχει ένα `main.c`, που καλεί τη `n_primes`, και ένα `primes.c`, που την υλοποιεί. Το `main.c` χρειάζεται μόνο τη δήλωση `ull *n_primes(ull n);`· την υλοποίηση τη φέρνει ο linker από το `primes.o`.

Το πιο απλό είναι να δώσετε όλα τα αρχεία σε ένα `gcc main.c primes.c -o main`. Πίσω από αυτή τη γραμμή τρέχουν ξεχωριστά ο `cpp` και ο compiler για κάθε αρχείο και στο τέλος ένα `ld` με όλα τα `.o` (Παράδειγμα «Ένα project με δύο αρχεία»).

§26.4 Τρία προβλήματα του χειροκίνητου compile

Με περισσότερα από ένα αρχεία, η στρατηγική «γράφω την εντολή `gcc` με το χέρι» έχει τρία προβλήματα:

1. **Μακριές εντολές**. Κάθε νέα απαίτηση προσθέτει flags: `-Wall`, `-Wextra`, `-Werror`, `-std=c99`, `-pedantic`, ... και η εντολή γίνεται δύσκολη στο πληκτρολόγηση και στη μνήμη.
2. **Πολλές εντολές**. Όταν ένα αρχείο θέλει διαφορετικά flags από τα άλλα, το compile πρέπει να σπάσει σε βήματα: ένα `gcc -c` ανά αρχείο και ένα για τη σύνδεση. Παράδειγμα: το `primes.c` θέλει την `reallocarray`, που δεν είναι στο πρότυπο της C, άρα δεν μπορεί να μεταγλωττιστεί με `-std=c99` όπως το `main.c`.
3. **Επαναμεταγλώττιση ολόκληρου του project** σε κάθε αλλαγή (βλ. παρακάτω).

Ήδη με πέντε αρχεία (η Εργασία 2, elevate) χρειάζονται έξι εντολές. Κανείς δεν τις θυμάται απ' έξω, και σε μεγάλο project ο χρόνος να τις τρέχετε μία μία ξεπερνά τον χρόνο της ίδιας της αλλαγής στον κώδικα.

§26.5 Build scripts σε bash

Η πρώτη λύση υπάρχει από τη δεκαετία του '70: βάζετε όλες τις εντολές σε ένα **bash script** (`build.sh`) και τρέχετε αυτό. Το `set -xe` κάνει το script να τυπώνει κάθε εντολή πριν την τρέξει (`-x`) και να σταματά στο πρώτο λάθος (`-e`). Έτσι λύνονται τα προβλήματα 1 και 2: οι εντολές γράφονται μία φορά.

Το bash όμως είναι γλώσσα γενικού σκοπού, όχι φτιαγμένη για compile. Δεν ξέρει ποια αρχεία άλλαξαν, οπότε **κάθε εκτέλεση ξαναμεταγλωττίζει όλο το project από την αρχή**.

§26.6 Γιατί η πλήρης επαναμεταγλώττιση είναι πρόβλημα

Η GNU C Library (glibc), η πιο διαδεδομένη υλοποίηση της standard library της C (και των προτύπων POSIX, GNU C, System V κ.ά.), έχει 1.524.568 γραμμές κώδικα σε πάνω από 14.000 αρχεία. Αν προσθέσετε ένα σχόλιο σε ένα αρχείο, ένα script θα ξαναμεταγλωττίσει και τα 14.000.

Η «λύση» να ξαναμεταγλωττίζετε με το χέρι μόνο τα αρχεία που αλλάξατε και να συνδέετε με ένα script έχει μια παγίδα: **τι γίνεται αν ξεχάσετε ένα**; Τότε τρέχετε ένα παλιό `.o` και ψάχνετε ένα bug που έχετε ήδη διορθώσει. Ακριβώς αυτό οδήγησε στο Make: ο Stuart Feldman γράφει ότι ο Steve Johnson (δημιουργός του yacc) έχασε ένα πρωί αποσφαλματώνοντας ένα σωστό πρόγραμμα, επειδή η διόρθωση δεν είχε μεταγλωττιστεί. Το εργαλείο γράφτηκε εκείνο το Σαββατοκύριακο, και τα Makefiles έγιναν απλά αρχεία κειμένου, «printable, debuggable, understandable», όπως θέλει η φιλοσοφία του Unix.

§26.7 Το Make και οι βασικές έννοιες

Το **Make** είναι πρόγραμμα που εκτελεί εντολές με βάση προκαθορισμένες **σχέσεις εξάρτησης (dependencies)**. Υπάρχουν πολλές παραλλαγές του· εδώ χρησιμοποιούμε το **GNU Make**, αλλά τα περισσότερα ισχύουν και για τις άλλες. Τρεις έννοιες:

- **target** (στόχος): ένα αποτέλεσμα που μπορεί να παραχθεί, συνήθως ένα αρχείο (`main`, `main.o`) ή και κάτι άλλο·
- **recipe** (συνταγή): οι εντολές που κατασκευάζουν ένα target·
- **rule** (κανόνας): ορίζει ένα target, τα **prerequisites** (προαπαιτούμενα) από τα οποία εξαρτάται, και το recipe του.

§26.8 Το Makefile και η σύνταξη ενός rule

Ένα **Makefile** είναι ένα σύνολο από rules με σκοπό την παραγωγή targets:

```
# rule
target_name1 target_name2: prerequisite1 prerequisite2
    # recipe
    command1
    command2
```

Το rule λέει ότι τα `target_name1` και `target_name2` φτιάχνονται τρέχοντας τα `command1` και `command2` σε shell, και ότι για να φτιαχτούν πρέπει πρώτα να έχουν φτιαχτεί τα `prerequisite1` και `prerequisite2`.

Προσοχή στο **indentation**: ό,τι είναι indented, το Make το θεωρεί μέρος του recipe. Στο GNU Make κάθε γραμμή του recipe ξεκινά με **tab**, όχι με κενά (έτσι το γράφει και το Εργαστήριο 10).

§26.9 Η εντολή make

Αν στον τρέχοντα φάκελο υπάρχει αρχείο με όνομα `Makefile` (ή ένα από λίγα άλλα αναγνωρισμένα ονόματα, όπως `makefile`), τρέχετε:

```
make target_name
```

Το `make` προσπαθεί να κατασκευάσει το `target_name` ακολουθώντας το **τελευταίο** rule του Makefile που ορίζει recipe για αυτό το όνομα. Χωρίς όρισμα, `make` σκέτο, χτίζει το target του **πρώτου** rule του αρχείου. Γι' αυτό το πρώτο rule είναι συνήθως το τελικό εκτελέσιμο.

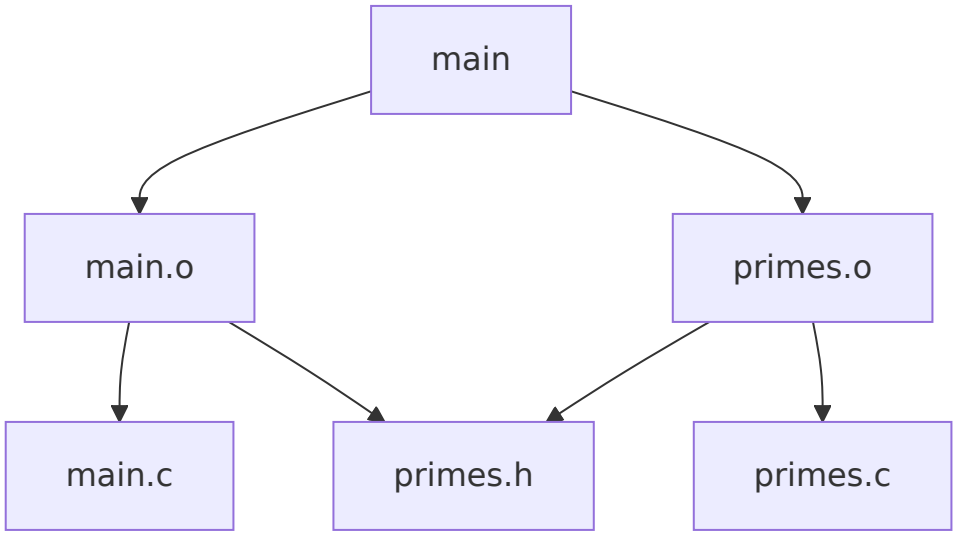
§26.10 Ο αλγόριθμος κατασκευής

Αφού επιλεγεί το build target, το Make ακολουθεί περίπου τον εξής αναδρομικό αλγόριθμο (διαφάνεια 28):

```
proc Make-Target(target):
    foreach prerequisite of target:
        if prerequisite is out of date:
            mark target out of date
            Make-Target(prerequisite)

    if target is out of date:
```

```
        run recipe of target
endproc
```

Ένα target είναι **out of date** (παρωχημένο) όταν δεν υπάρχει, ή όταν κάποιο prerequisite του είναι νεότερο από αυτό: το Make συγκρίνει τους χρόνους τελευταίας τροποποίησης των αρχείων. Οι εξαρτήσεις σχηματίζουν ένα δέντρο (γενικότερα, γράφο):



Σχήμα: οι εξαρτήσεις του project `main` *(διαφάνεια 29).*

Αν αλλάξετε το `main.c`, μόνο το `main.o` και το `main` είναι out of date· το `primes.o` δεν ξαναφτιάχνεται. Αν αλλάξετε το `primes.h`, ξαναφτιάχνονται και τα δύο `.o`, γιατί και τα δύο το κάνουν `#include`. Το Make δεν μπορεί να «ξεχάσει» ένα αρχείο, όπως εσείς: αρκεί οι εξαρτήσεις να είναι σωστά δηλωμένες.

§26.11 Macros

Αν κάποιος θέλει να χτίσει το project με `clang` αντί για `gcc` (π.χ. σε Mac), πρέπει να αλλάξει κάθε εμφάνιση του `gcc` στο Makefile. Η λύση είναι η ίδια με της C: μια μεταβλητή. Στο Make οι μεταβλητές λέγονται **macros** και περιέχουν **strings**:

```
MACRO_NAME <assignment-operator> string value
```

Τα ονόματα γράφονται συνήθως με κεφαλαία, και η τιμή είναι string **χωρίς εισαγωγικά**. Χρησιμοποιείτε ένα macro οπουδήποτε στο Makefile γράφοντας `$(MACRO_NAME)`. Η τελική τιμή ενός macro είναι η τελευταία που του ανατέθηκε μέσα στο Makefile (ή μέσα σε ένα rule).

Υπάρχουν δύο είδη ανάθεσης:

- **lazily evaluated** ή **recursive** macros, που αποτιμώνται μόνο όταν χρησιμοποιηθούν, αφού διαβαστεί ολόκληρο το Makefile·

- **eagerly evaluated** ή **simple** macros, που αποτιμώνται τη στιγμή της ανάθεσης, κατά την ανάγνωση του Makefile.

Τελεστής	Τι κάνει
`=`	lazy ανάθεση
`:=`	eager ανάθεση
`?=`	lazy ανάθεση, μόνο αν το macro δεν έχει ήδη τιμή
`+=`	προσθέτει τη δεξιά μεριά στο τέλος της αριστερής (lazily)

Η διαφορά `=` και `:=` φαίνεται όταν το δεξί μέλος χρησιμοποιεί ένα macro που ορίζεται **αργότερα**:

```
A = $(B)
C := $(B)
B = hello
```

Εδώ το `$(A)` δίνει `hello` (αποτιμάται στη χρήση, όταν το `B` έχει ήδη τιμή), ενώ το `$(C)` είναι κενό (αποτιμήθηκε όταν το `B` δεν υπήρχε ακόμη).

§26.12 Automatic variables

Μέσα σε ένα recipe μπορείτε να χρησιμοποιήσετε και **automatic variables**, μεταβλητές που αλλάζουν αυτόματα τιμή από rule σε rule:

Μεταβλητή	Τιμή
`$@`	το όνομα του target που φτιάχνεται τώρα
`$^`	η λίστα όλων των prerequisites
`$<`	το πρώτο prerequisite

Έτσι ένα recipe δεν επαναλαμβάνει ονόματα αρχείων: `$(CC) -o $@ $^` σημαίνει «σύνδεσε όλα τα prerequisites στο target».

§26.13 Wildcard rules

Με macros και automatic variables, τα rules για `main.o` και `primes.o` γίνονται σχεδόν πανομοιότυπα. Όπως στον προγραμματισμό, όταν κάτι επαναλαμβάνεται, το αφαιρούμε (to abstract it). Ένα **wildcard rule** (στο εγχειρίδιο του GNU Make: *pattern rule*) ορίζει με τον ειδικό χαρακτήρα `%` μια ολόκληρη οικογένεια από rules:

```
%.o: %.c
    recipe
```

Αυτό λέει ότι **κάθε** `X.o` έχει prerequisite το αντίστοιχο `X.c` και φτιάχνεται με το ίδιο recipe. Μέσα στο recipe, το `$<` είναι το `X.c` και το `$@` το `X.o`.

§26.14 Built-in rules

Το Make φτιάχτηκε για να χτίζει projects σε C, οπότε έχει ήδη ορισμένα **built-in rules** για τα πιο συνηθισμένα:

- compile object files από translation units (`X.o` από `X.c`)·
- link object files σε εκτελέσιμο (`X` από `X.o` και όσα άλλα `.o` δηλωθούν)·
- και πολλά άλλα.

Τα built-in rules χρησιμοποιούν γνωστά macros, όπως το `CC` (ο compiler) και το `CFLAGS` (τα flags του compiler). Ορίζοντας μόνο αυτά και τις εξαρτήσεις, το Makefile του project γίνεται 2–3 γραμμές (Παράδειγμα «Makefile με built-in rules»).

§26.15 Το Make στον πραγματικό κόσμο

Η απλότητα και η δύναμη του Make το πάνε πολύ μακριά. Ο **Linux kernel** (~35 εκατομμύρια γραμμές) χτίζεται με Make. Τα πιο προχωρημένα εργαλεία (`cmake`, `premake`, `automake`, `autoconf`) είναι χτισμένα πάνω στο Make και το χρησιμοποιούν, οπότε όλα τα μεγάλα projects χρησιμοποιούν Make, άμεσα ή έμμεσα. Και το Make δεν είναι δεμένο με τη C: δουλεύει και για άλλες γλώσσες.

Παραδείγματα

§26.16 Το math.h και η libm.so

Εφαρμόζει: «Σφάλματα μεταγλώττισης και σφάλματα σύνδεσης».

Ξεκινάμε χωρίς `#include <math.h>`:

```c
#include <stdio.h>
// does-not-compile
int main(void) {
    printf("%.2f\n", sqrt(3));
    return 0;
}
```

```
$ gcc -o main main.c
main.c: In function 'main':
main.c:4:20: error: implicit declaration of function 'sqrt'
[-Wimplicit-function-declaration]
    4 |   printf("%.2f\n", sqrt(3));
      |                     ^~~~
main.c:2:1: note: include '<math.h>' or provide a declaration of 'sqrt'
    1 | #include <stdio.h>
  +++ |+#include <math.h>
```

```
    2 |
```

Είναι **compiler error**: ο compiler δεν βρίσκει τη δήλωση της `sqrt` στο αρχείο, και μας λέει ακριβώς πώς να το διορθώσουμε. Προσθέτουμε `#include <math.h>`:

```
$ gcc -o main main.c
/usr/bin/ld: /tmp/ccaw9D8e.o: in function `main':
main.c:(.text+0x1f): undefined reference to `sqrt'
collect2: error: ld returned 1 exit status
```

Τώρα είναι **linking error** (το λέει το `ld returned 1`): η δήλωση υπάρχει, η υλοποίηση όχι. Δίνουμε στον linker τη βιβλιοθήκη:

```
$ gcc -o main main.c -l:libm.so # or -lm for short
$ ./main
1.73
```

§26.17 Ένα project με δύο αρχεία

Εφαρμόζει: «Projects με πολλά αρχεία», «Τρία προβλήματα του χειροκίνητου compile».

Τα δύο αρχεία της διαφάνειας 10 (το σώμα των συναρτήσεων παραλείπεται στις διαφάνειες):

```c
/* file: main.c */
typedef unsigned long long ull;
ull *n_primes(ull n);

int main(int argc, const char **argv) {
....
    ull *primes = n_primes(n);
....
}
```

```c
/* file: primes.c */
typedef unsigned long long ull;
ull *n_primes(ull n) {
....
}
```

Η εύκολη και η «πλήρης» εκδοχή του ίδιου build:

```
$ gcc main.c primes.c -o main
$ ./main 21
Found primes: 2 3 5 7 11 13 17 19
```

```
$ cpp main.c -o main.C
$ cpp primes.c -o primes.C
$ gcc main.C -c
```

```
$ gcc primes.C -c
$ ld -o main --dynamic-linker /lib64/ld-linux-x86-64.so.2 \
    /lib/crt1.o /lib/crti.o main.o primes.o -lc /lib/crtn.o
$ ./main 21
Found primes: 2 3 5 7 11 13 17 19
```

Η δεύτερη δείχνει τι κάνει το `gcc` για εμάς: προεπεξεργασία, compile κάθε αρχείου, και σύνδεση με τη standard library (`-lc`) και τα αρχεία εκκίνησης (`crt*.o`).

Όταν τα αρχεία θέλουν διαφορετικά flags (το `primes.c` χωρίς `-std=c99`, για την `reallocarray`), το build σπάει σε βήματα:

```
$ gcc -Wall -Wextra -Werror -std=c99 -pedantic -c main.c
$ gcc -Wall -Wextra -Werror -pedantic -c primes.c
$ gcc -o main primes.o main.o
```

§26.18 Το build της Εργασίας 2 (elevate)

Εφαρμόζει: «Τρία προβλήματα του χειροκίνητου compile».

Η Εργασία 2 του 2025-26 (elevate) χτίζεται με έξι εντολές, όπως τις δίνει η εκφώνηση:

```
$ gcc -Os -c -Wall -Wextra -Werror -pedantic recurse.c
$ gcc -Os -c -Wall -Wextra -Werror -pedantic brute.c
$ gcc -Os -c -Wall -Wextra -Werror -pedantic memoize.c
$ gcc -Os -c -Wall -Wextra -Werror -pedantic dp.c
$ gcc -Os -c -Wall -Wextra -Werror -pedantic elevate.c
$ gcc -Os -o elevate recurse.o brute.o memoize.o dp.o elevate.o
```

Είναι το ιδανικό πρώτο `Makefile` σας (βλ. «Makefile με built-in rules»).

§26.19 Το build.sh

Εφαρμόζει: «Build scripts σε bash».

```
#!/usr/bin/env bash
# file: build.sh
set -xe
gcc -Wall -Werror -Wextra -pedantic -std=c99 -c main.c
gcc -Wall -Werror -Wextra -pedantic -c primes.c
gcc -o main main.o primes.o
```

```
$ chmod +x ./build.sh
$ ./build.sh 2> /dev/null
$ ./main 42
Found primes: 2 3 5 7 11 13 17 19 23 29 31 37 41
```

Το `2> /dev/null` κρύβει την έξοδο του `set -x`, που γράφεται στο stderr. Το script ξαναμεταγλωττίζει και τα δύο αρχεία σε κάθε εκτέλεση, όποιο κι αν άλλαξε.

§26.20 Το πρώτο Makefile

Εφαρμόζει: «Το Makefile και η σύνταξη ενός rule», «Ο αλγόριθμος κατασκευής».

Το Makefile της διαφάνειας 31 γράφει το δέντρο εξαρτήσεων ως τρία rules, με το αρχείο επικεφαλίδας `primes.h` (όπου πηγαίνουν πλέον το `typedef` και η δήλωση της `n_primes`) ως prerequisite και των δύο `.o`:

```
main: main.o primes.o
    gcc -o main main.o primes.o

main.o: main.c primes.h
    gcc -Wall -Wextra -Werror -pedantic -std=c99 -c main.c

primes.o: primes.c primes.h
    gcc -Wall -Wextra -Werror -pedantic -c primes.c
```

Μια συνεδρία (τα μηνύματα είναι του GNU Make 4):

```
$ make
gcc -Wall -Wextra -Werror -pedantic -std=c99 -c main.c
gcc -Wall -Wextra -Werror -pedantic -c primes.c
gcc -o main main.o primes.o
$ make
make: 'main' is up to date.
$ touch main.c
$ make
gcc -Wall -Wextra -Werror -pedantic -std=c99 -c main.c
gcc -o main main.o primes.o
```

Μετά το `touch main.c` το `primes.o` δεν ξαναφτιάχτηκε. Λειτουργεί· είναι καλό; Όχι ακόμη: το `gcc` και τα flags επαναλαμβάνονται παντού.

§26.21 Makefile με macros, automatic variables και wildcard rule

Εφαρμόζει: «Macros», «Automatic variables», «Wildcard rules».

Οι διαφάνειες 32–38 περιγράφουν τα βήματα χωρίς να δείχνουν το τελικό αρχείο. Μια εκδοχή που τα εφαρμόζει όλα:

```
CC = gcc
CFLAGS = -Wall -Wextra -Werror -pedantic

main: main.o primes.o
```

```
    $(CC) -o $@ $^

main.o: CFLAGS += -std=c99

%.o: %.c primes.h
    $(CC) $(CFLAGS) -c $<
```

- Το `CC` αλλάζει σε ένα σημείο· ή από τη γραμμή εντολών, `make CC=clang`, χωρίς αλλαγή στο αρχείο.
- Το `$@ $^` στο link γίνεται `main main.o primes.o`.
- Το wildcard rule αντικαθιστά τα δύο rules των `.o`· το `$<` είναι το `.c`.
- Το `main.o: CFLAGS += -std=c99` είναι ανάθεση **μέσα σε rule**: ισχύει μόνο όταν φτιάχνεται το `main.o`, οπότε μόνο το `main.c` παίρνει `-std=c99`.

```
$ make CC=clang
clang -Wall -Wextra -Werror -pedantic -std=c99 -c main.c
clang -Wall -Wextra -Werror -pedantic -c primes.c
clang -o main main.o primes.o
```

§26.22 Makefile με built-in rules

Εφαρμόζει: «Built-in rules».

Αφού το Make ξέρει ήδη πώς φτιάχνεται ένα `.o` από ένα `.c` και ένα εκτελέσιμο από `.o`, αρκεί να δηλώσουμε τα flags και τις εξαρτήσεις:

```
CFLAGS = -Wall -Wextra -Werror -pedantic
main: main.o primes.o
main.o primes.o: primes.h
```

```
$ make
cc -Wall -Wextra -Werror -pedantic   -c -o main.o main.c
cc -Wall -Wextra -Werror -pedantic   -c -o primes.o primes.c
cc   main.o primes.o   -o main
```

Το built-in rule χρησιμοποιεί `$(CC)`, που από προεπιλογή είναι `cc` (στο Linux συνήθως το `gcc`). Ένα rule χωρίς recipe, όπως το `main.o primes.o: primes.h`, απλώς προσθέτει prerequisites.

§26.23 Συμβουλή για το Εργαστήριο 10

Το Εργαστήριο 10, ενότητα «Αυτοματοποίηση με make», χτίζει το `collatz` με ένα Makefile με `CC`, `CFLAGS` και έναν στόχο `clean` (`rm -f collatz main.o collatz.o`). Στην Άσκηση 5 γράφετε το δικό σας Makefile για το `more.c`: επιβεβαιώστε, αλλάζοντας μόνο το `main.c`, ότι το `make` ξαναμεταγλωττίζει μόνο αυτό.

Κύρια σημεία

1. Κάθε `.c` (translation unit) περνά από τον preprocessor και τον compiler και γίνεται object file· ο linker ενώνει τα object files και τις βιβλιοθήκες σε εκτελέσιμο.
2. Ένα compiler error αφορά το translation unit (π.χ. λείπει μια δήλωση)· ένα linking error σημαίνει ότι λείπει μια υλοποίηση από το linking scope.
3. Το `#include <math.h>` δίνει μόνο δηλώσεις· την υλοποίηση της `sqrt` τη φέρνει το `-lm` (`libm.so`).
4. Σε projects πολλών αρχείων το χειροκίνητο compile έχει τρία προβλήματα: μακριές εντολές, πολλές εντολές και επαναμεταγλώττιση όλου του project.
5. Ένα bash script λύνει τα δύο πρώτα, αλλά ξαναμεταγλωττίζει τα πάντα σε κάθε εκτέλεση· αν πάλι ξαναμεταγλωττίζετε με το χέρι, κάποια στιγμή θα ξεχάσετε ένα αρχείο.
6. Το Make εκτελεί recipes με βάση σχέσεις εξάρτησης και ξαναχτίζει μόνο τα targets που είναι out of date.
7. Ένα rule γράφεται `target: prerequisites` και από κάτω το recipe, με κάθε γραμμή indented με tab.
8. `make target` χτίζει το συγκεκριμένο target· `make` σκέτο χτίζει το target του πρώτου rule.
9. Τα macros (`CC`, `CFLAGS`) κρατούν strings· το `=` αποτιμάται στη χρήση, το `:=` στην ανάθεση, το `?=` αναθέτει μόνο αν δεν υπάρχει τιμή και το `+=` προσθέτει.
10. Οι automatic variables `$@`, `$^`, `$<` και τα wildcard rules (`%.o: %.c`) αφαιρούν την επανάληψη από ένα Makefile.
11. Τα built-in rules του Make ήδη ξέρουν να κάνουν compile και link, άρα ένα Makefile για C μπορεί να είναι 2–3 γραμμές.
12. Το Make χτίζει τον Linux kernel και βρίσκεται κάτω από τα `cmake`, `automake` κ.λπ.· δεν περιορίζεται στη C.

Ορολογία

Ελληνικά	English	Σύντομος ορισμός
μονάδα μετάφρασης	translation unit	Ένα αρχείο `.c` μαζί με ό,τι φέρνουν τα `#include` του
αντικειμενικό αρχείο	object file	Το `.o` που βγάζει ο compiler από ένα translation unit
συνδέτης	linker (`ld`)	Ενώνει object files και βιβλιοθήκες σε εκτελέσιμο
δυναμική βιβλιοθήκη	dynamic library (`.so`)	Βιβλιοθήκη που φορτώνεται στην εκτέλεση, π.χ. `libm.so`
σφάλμα σύνδεσης	linking error	Ο linker δεν βρίσκει υλοποίηση συμβόλου (`undefined reference`)
σύστημα κατασκευής	build system	Εργαλείο που αυτοματοποιεί το χτίσιμο ενός project

Ελληνικά	English	Σύντομος ορισμός
στόχος	target	Ό,τι μπορεί να παραχθεί, συνήθως ένα αρχείο
προαπαιτούμενο	prerequisite	Target από το οποίο εξαρτάται ένα άλλο target
συνταγή	recipe	Οι εντολές shell που φτιάχνουν ένα target
κανόνας	rule	`target: prerequisites` μαζί με το recipe
μακροεντολή	macro	Μεταβλητή του Make με τιμή string, `$(NAME)`
αυτόματη μεταβλητή	automatic variable	`$@`, `$^`, `$<`: αλλάζουν τιμή ανά rule
κανόνας μοτίβου	wildcard / pattern rule	Rule με `%` που ορίζει οικογένεια rules
ενσωματωμένος κανόνας	built-in rule	Rule που το Make ξέρει ήδη (π.χ. `.o` από `.c`)
παρωχημένο	out of date	Target που λείπει ή είναι παλαιότερο από prerequisite του

Διάβασμα

- **Διαφάνειες**: How to Make?, σελ. 1–42. Διαδικασία μετάφρασης και `-lm`: σελ. 4–8· προβλήματα του χειροκίνητου build και bash: σελ. 9–21· βασικά του Make: σελ. 22–29· macros, automatic variables, wildcard και built-in rules: σελ. 30–40.
- **Σημειώσεις**: Κεφάλαιο 0: Εισαγωγή, ενότητες «Πώς κατασκευάζουμε εκτελέσιμο πρόγραμμα;», «Η διαδικασία της μεταγλώττισης και σύνδεσης», «Παραδείγματα χρήσης του gcc» (K04, σελ. 14–17)· Κεφάλαιο 1: Πρώτα προγράμματα για το `-lm`. Οι σημειώσεις δεν καλύπτουν το Make.
- **Εργαστήριο**: Εργαστήριο 10, «Παράρτημα: Οργάνωση προγράμματος σε πολλαπλά αρχεία», ενότητες «Χωριστή μεταγλώττιση και σύνδεση», «Αυτοματοποίηση με make» και Άσκηση 5 (`more.c` με `Makefile` και στόχο `clean`).
- **Άλλα**: GNU Make manual: Automatic Variables, Flavors (τα δύο είδη macros), Catalogue of Built-In Rules· Wikipedia: The Art of Unix Programming, Eric S. Raymond, Stephen C. Johnson, Unix philosophy.

Συχνά λάθη

- **Κενά αντί για tab στο recipe**. Το GNU Make απαντά `Makefile:2: *** missing separator. Stop.` Ρυθμίστε τον editor να βάζει tab στα Makefiles.
- `#include <math.h>` **χωρίς** `-lm`. `undefined reference to 'sqrt'`: είναι linking error, όχι compiler error· προσθέστε `-lm` στη σύνδεση (ή `LDLIBS = -lm` όταν χρησιμοποιείτε built-in

rules).

- **Μπέρδεμα compiler error και linking error.** Αν το μήνυμα λέει `ld returned 1 exit status`, ψάξτε για βιβλιοθήκη ή `.o` που λείπει από την εντολή σύνδεσης, όχι για λάθος σύνταξης.
- **Ξεχασμένο αρχείο επικεφαλίδας στα prerequisites.** Αν το `main.o` δεν εξαρτάται από το `primes.h`, μια αλλαγή στο `primes.h` δεν ξαναφτιάχνει το `main.o` και το πρόγραμμα τρέχει με παλιό κώδικα. Βάλτε κάθε `.h` που κάνει `#include` ένα `.c` στα prerequisites του `.o` του.
- **Λάθος πρώτο rule.** Αν πρώτο στο αρχείο είναι το `primes.o: …`, το σκέτο `make` φτιάχνει μόνο αυτό. Βάλτε πρώτο το rule του εκτελέσιμου.
- **Εισαγωγικά στην τιμή ενός macro.** `CC = "gcc"` κάνει τα εισαγωγικά μέρος του string· γράψτε `CC = gcc`.
- **`:=` όπου χρειάζεται `=`.** Με `C := $(B)` πριν οριστεί το `B`, το `C` μένει κενό.
- **Build με script σε μεγάλο project.** Κάθε αλλαγή ξαναμεταγλωττίζει τα πάντα· στο άλλο άκρο, το χειροκίνητο compile μόνο όσων αλλάξατε κάποτε ξεχνά ένα αρχείο και τρέχετε παλιό `.o`.

Ερωτήσεις κατανόησης

- **E26.1** Ποια τρία στάδια μεσολαβούν από ένα `.c` σε ένα εκτελέσιμο, και τι βγάζει το καθένα;[197]
- **E26.2** Γιατί το `#include <math.h>` δεν αρκεί για να χρησιμοποιήσετε τη `sqrt`;[198]
- **E26.3** Γιατί ένα linking error είναι λιγότερο κατατοπιστικό από ένα compiler error;[199]
- **E26.4** Ποιο πρόβλημα του χειροκίνητου compile δεν λύνει ένα bash script;[200]
- **E26.5** Σε ένα Makefile, τι είναι target, prerequisite, recipe και rule;[201]
- **E26.6** Ποιο target χτίζει το `make` όταν δεν του δώσετε όρισμα;[202]
- **E26.7** Με το Makefile του σχήματος εξαρτήσεων, ποια αρχεία ξαναφτιάχνονται αν αλλάξει το `primes.c`; Και αν αλλάξει το `primes.h`;[203]
- **E26.8** Ποια η διαφορά ανάμεσα σε `CFLAGS = …` και `CFLAGS := …`;[204]
- **E26.9** Στο rule `main: main.o primes.o`, τι τιμή έχουν τα `$@`, `$^` και `$<`;[205]
- **E26.10** Τι σημαίνει `%.o: %.c`, και γιατί ένα Makefile για C μπορεί να είναι 2–3 γραμμές;[206]

[197] Preprocessor (προεπεξεργασμένο `.c`), compiler (object file `.o`), linker (εκτελέσιμο, από τα `.o` και τις βιβλιοθήκες).

[198] Το `math.h` έχει μόνο τη δήλωση· η υλοποίηση είναι στη `libm.so`, που πρέπει να δοθεί στον linker με `-lm`.

[199] Ο linker δουλεύει με object files, όχι με τον πηγαίο κώδικα, άρα δεν ξέρει γραμμή ούτε ποια βιβλιοθήκη λείπει.

[200] Την επαναμεταγλώττιση ολόκληρου του project: το script ξαναχτίζει τα πάντα σε κάθε εκτέλεση.

[201] Target: ό,τι παράγεται· prerequisite: ό,τι χρειάζεται πρώτα· recipe: οι εντολές που το φτιάχνουν· rule: το σύνολο `target: prerequisites` και recipe.

[202] Το target του πρώτου rule του Makefile.

[203] `primes.c`: το `primes.o` και το `main`. `primes.h`: και τα δύο `.o` και το `main`.

[204] Το `=` αποτιμά το δεξί μέλος όταν χρησιμοποιηθεί το macro (lazy)· το `:=` τη στιγμή της ανάθεσης (eager).

[205] `$@` = `main`, `$^` = `main.o primes.o`, `$<` = `main.o`.

[206] Wildcard rule: κάθε `X.o` φτιάχνεται από το `X.c` με το ίδιο recipe. Το Make έχει ήδη built-in rules για compile

Ασκήσεις

Ζέσταμα: από τις διαφάνειες (A26.1–A26.5)

- **A26.1** Τα στάδια του C build process: How to Make? (προσκεκλημένη διάλεξη), διαφάνεια 5 · ★☆☆ · short-answer · `slides-lecmake-build-pipeline`
- **A26.2** Script ή recompile με το χέρι;: How to Make? (προσκεκλημένη διάλεξη), διαφάνειες 17-21 · ★☆☆ · short-answer · `slides-lecmake-forgot-recompile`
- **A26.3** Διαφορετικά flags ανά αρχείο: How to Make? (προσκεκλημένη διάλεξη), διαφάνειες 10-14 · ★☆☆ · tooling · `slides-lecmake-per-file-flags`
- **A26.4** Compiler error ή linking error; (math.h και libm.so): How to Make? (προσκεκλημένη διάλεξη), διαφάνειες 6-8 · ★☆☆ · debug · `slides-lecmake-sqrt-errors`
- **A26.5** Ένα Makefile 2-3 γραμμών: How to Make? (προσκεκλημένη διάλεξη), διαφάνειες 31-39 · ★★☆ · tooling · `slides-lecmake-short-makefile`

Σχετικές ασκήσεις από άλλα κεφάλαια

- **A22.16** Νέα Μηχανή Σκακιού (chess engine): Εργασία 3 (2024-25), Άσκηση 1 · ★★★ · programming · `hw-2024-hw3-chess`
- **A22.17** Νέα Μηχανή Go (goteam): Εργασία 3 (2025-26), Άσκηση 1 · ★★★ · programming · `hw-2025-hw3-goteam`
- **A23.5** Σπάστε το πρόγραμμά σας σε αρθρώματα: Εργαστήριο 10, Άσκηση 5 · ★★☆ · tooling · `lab-lab10-more-modules`

και link, οπότε αρκεί να δηλώσετε flags και εξαρτήσεις.

Γλωσσάριο

Όλοι οι όροι του οδηγού, από τους πίνακες «Ορολογία» των κεφαλαίων, με τον αγγλικό όρο, σύντομο ορισμό και τα κεφάλαια όπου εμφανίζονται.

Ελληνικά	English	Ορισμός	Κεφάλαια
bitwise τελεστής	bitwise operator	`&`, `|`, `^`, `~`, `<<`, `>>`, bit προς bit	5
byte, οκτάδα	byte, octet	Ομάδα (συνήθως 8) bits σε ένα κελί μνήμης	2
camelCase / snake_case	camelCase / snake_case	Στυλ ονομάτων: `piApprox` / `pi_approx`.	9
include guard	include guard	`#ifndef`/`#define`/`#endif` γύρω από ένα `.h`.	23
lvalue / rvalue	lvalue / rvalue	Αριστερός (μεταβλητή) / δεξιός (τιμή) τελεστέος της ανάθεσης	4
null byte	null byte / null terminator	Ο χαρακτήρας `'\0'` (τιμή 0) που σημειώνει το τέλος ενός string.	10
null byte	null byte	Ο χαρακτήρας `'\0'` που σημαδεύει το τέλος.	14
variadic συνάρτηση	variadic function	συνάρτηση με μεταβλητό αριθμό ορισμάτων (`...`)	24
ακέραια διαίρεση	integer division	`/` μεταξύ ακεραίων, που κρατάει μόνο το πηλίκο	5
ακολουθία διαφυγής	escape sequence	`\` και ένας χαρακτήρας, π.χ. `\n`	2, 3
αλγόριθμος	algorithm	Σαφής διαδικασία από εκτελέσιμα βήματα που τερματίζει	0
αλλαγή γραμμής	newline	Ο χαρακτήρας `\n`	0
αλφαριθμητικό μορφοποίησης	format string	Η πρώτη παράμετρος της `printf`	2, 3, 9, 10
αμφισημία	ambiguity	Όταν μια λέξη ή πρόταση έχει περισσότερες από μία σημασίες	0
αναδρομή	recursion	Μια συνάρτηση καλεί τον εαυτό της.	11
αναδρομική περίπτωση	recursive case	Το σημείο όπου η συνάρτηση καλεί τον εαυτό της.	11

Ελληνικά	English	Ορισμός	Κεφάλαια
αναδρομική σχέση	recurrence	Τύπος που εκφράζει τη λύση ενός προβλήματος μέσω λύσεων μικρότερων στιγμιοτύπων.	25
αναζήτηση	search	Εύρεση ενός στοιχείου (και της θέσης του) σε μια συλλογή.	17
αναζήτηση κατά βάθος	depth-first search (DFS)	Εξερεύνηση όσο πιο βαθιά γίνεται, μετά οπισθοδρόμηση.	21, 22
αναζήτηση κατά πλάτος	breadth-first search (BFS)	Εξερεύνηση επίπεδο-επίπεδο.	21, 22
ανάθεση	assignment	Αποθήκευση τιμής σε μεταβλητή (`x = 42;`)	2, 3
ανοχή σε σφάλματα	fault tolerance	Ικανότητα ενός συστήματος να λειτουργεί σωστά παρά την αποτυχία κάποιου μέρους του.	7
αντικειμενικό αρχείο	object file (`.o`)	Αποτέλεσμα μεταγλώττισης ενός `.c` με `gcc -c`.	23, 24, 26
αντιμετάθεση	swap	Ανταλλαγή των τιμών δύο θέσεων.	17, 18
αντιστάθμισμα χρόνου–μνήμης	time–space tradeoff	Ταχύτερη λύση με περισσότερη μνήμη, ή το αντίστροφο.	16
άνω όριο	upper bound, Big-O	$g = O(f)$: η g δεν ξεπερνά τη $c \cdot f$ για μεγάλα n.	15
ανώνυμη δομή	anonymous struct	Δομή χωρίς ετικέτα, συνήθως με `typedef`.	19
απαρίθμηση	enumeration (`enum`)	Τύπος με ονομασμένες ακέραιες σταθερές.	20
απλά συνδεδεμένη λίστα	singly linked list	Αλυσίδα κόμβων, ο καθένας δείχνει στον επόμενο, ο τελευταίος στο `NULL`.	20
απλά συνδεδεμένη λίστα	single linked list	Κόμβοι όπου ο καθένας δείχνει στον επόμενο και ο τελευταίος στο `NULL`.	21
άπληστος αλγόριθμος	greedy algorithm	Αλγόριθμος που κάνει σε κάθε βήμα την τοπικά καλύτερη επιλογή χωρίς να την αναθεωρεί.	25
αποαναφορά	dereference	Πρόσβαση με `*` στη μεταβλητή όπου δείχνει ο δείκτης.	11, 12
αποθετήριο	repository	Φάκελος αρχείων μαζί με το ιστορικό τους	4, 7

Ελληνικά	English	Ορισμός	Κεφάλαια
αποθήκευση κατά γραμμές	row-major order	Οι γραμμές αποθηκεύονται η μία μετά την άλλη.	12
αποκλειστικό Ή	XOR (^)	Bit 1 όταν τα δύο bits διαφέρουν· $x \oplus x = 0$.	16
απομνημόνευση	memoization	Αποθήκευση αποτελεσμάτων ώστε να μην ξαναϋπολογίζονται.	16, 25
αποσφαλμάτωση	debugging	Εντοπισμός και διόρθωση λαθών σε πρόγραμμα	0, 24
απροσδιόριστη συμπεριφορά	undefined behavior	Περίπτωση όπου η C δεν ορίζει τι θα συμβεί	8, 10
αριθμητική δεικτών	pointer arithmetic	Το `p + n` δείχνει `n` στοιχεία (όχι bytes) μετά.	11
αρχείο	file	Πόρος για αποθήκευση δεδομένων, συνήθως στον δίσκο	1, 18
αρχείο επικεφαλίδας	header file	Αρχείο με δηλώσεις, π.χ. `stdio.h`	0, 15, 23, 24
αρχείο υλοποίησης	implementation file (`.c`)	Περιέχει τον κώδικα των συναρτήσεων.	23
αρχικοποίηση	initialization	Ανάθεση κατά τον ορισμό (`int x = 42;`)	2
αρχικοποίηση / βήμα	initialization / step	Το πρώτο και το τρίτο μέρος της `for`	6
αρχικοποίηση δομής	struct initialization	Τιμές σε `{ }` με τη σειρά των πεδίων.	19
ατέρμονας βρόχος	infinite loop	Βρόχος που δεν τερματίζει ποτέ	6
αυτοαναφορά	self-reference	Όταν κάτι αναφέρεται στον εαυτό του.	20
αυτοαναφορική δομή	self-referential struct	Δομή με μέλος-δείκτη σε δομή του ίδιου τύπου.	20
αυτόματη μεταβλητή	automatic variable	`$@`, `$^`, `$<`: αλλάζουν τιμή ανά rule	26
αφαίρεση	abstraction	Περιγραφή του τι κάνει ένα κομμάτι, κρύβοντας το πώς.	23, 24
αφηρημένος τύπος δεδομένων	abstract data type (ADT)	Τύπος που ορίζεται από τις λειτουργίες του, όχι από την υλοποίηση.	21, 22
βάθος	depth	Ο μέγιστος αριθμός συνδέσμων από τη ρίζα ως ένα φύλλο.	20

Ελληνικά	English	Ορισμός	Κεφάλαια
βάθος / ύψος	depth / height	Μέγιστος αριθμός συνδέσμων από τη ρίζα ως τα φύλλα / από τα φύλλα ως τη ρίζα.	21, 22
βασική αιτία	root cause	Το αρχικό σφάλμα από το οποίο ξεκίνησε μια αποτυχία.	7
βασική περίπτωση	base case	Η συνθήκη όπου η αναδρομή σταματά.	11
βήμα	step	Η έκφραση που αλλάζει τη μεταβλητή του βρόχου στο τέλος κάθε επανάληψης της `for`.	7
βρόχος	loop	Εντολή που επαναλαμβάνει ένα σώμα	6, 7
γεμάτο δυαδικό δέντρο	full binary tree	Κάθε κόμβος έχει 0 ή 2 παιδιά.	22
γέμισμα	padding	Αχρησιμοποίητα bytes που προσθέτει ο μεταγλωττιστής.	19
γλώσσα προγραμματισμού	programming language	Γλώσσα χωρίς αμφισημίες για εντολές σε υπολογιστή	0
γονικός φάκελος	parent folder	Ο φάκελος που περιέχει έναν άλλο φάκελο.	20
γραμμές κώδικα	lines of code (LOC)	μετρική μεγέθους ενός προγράμματος	24
γραμμή κώδικα	line of code (LOC / SLOC)	Εντολές μέχρι την αλλαγή γραμμής· μετρική μεγέθους.	23
γραμμική / σειριακή αναζήτηση	linear / serial search	Έλεγχος των στοιχείων ένα προς ένα· $O(n)$.	17
γραμμικό πέρασμα	linear pass	Διάσχιση των δεδομένων μία φορά, σε χρόνο $O(n)$.	25
γραφική διεπαφή χρήστη	GUI	Αλληλεπίδραση με παράθυρα και ποντίκι	1
γράφος	graph	Κόμβοι συνδεδεμένοι με ακμές, π.χ. πόλεις και δρόμοι.	20
δεδομένα εισόδου / εξόδου	input / output data	Ό,τι δίνεται στο πρόγραμμα / ό,τι παράγει.	9
δεδομένο εξόδου	output	Ό,τι παράγει ένα πρόγραμμα όταν τελειώσει	1
δείκτης	pointer	Μεταβλητή που κρατά τη διεύθυνση ενός δεδομένου.	11
δείκτης σε δείκτη	pointer to pointer	Δείκτης που κρατά διεύθυνση δείκτη, π.χ. `int **`.	12

Ελληνικά	English	Ορισμός	Κεφάλαια
δείκτης σε κενό	`void *`	Pointer σε «κάτι»: σκέτη διεύθυνση.	13
δείκτης σε συνάρτηση	function pointer	μεταβλητή με τη διεύθυνση μιας συνάρτησης	24
δείχνει σε	points to	Ο δείκτης κρατά τη διεύθυνση της μεταβλητής.	11
δεκαεξαδικό σύστημα	hexadecimal	Αρίθμηση με βάση το 16 (`0–9`, `A–F`)	2
δεσμευμένη λέξη	reserved keyword	Λέξη της C που δεν γίνεται όνομα (`int`, `if`, …)	2, 3
δευτερεύουσα μνήμη	secondary memory	Μόνιμη αποθήκευση: δίσκοι, flash, DVD	2
δήλωση	declaration	Εισαγωγή μεταβλητής με τύπο και όνομα	2
δήλωση / ορισμός	declaration / definition	Ενημερώνει για τον τύπο / δεσμεύει μνήμη ή δίνει κώδικα.	23
διάγραμμα ενεργοποίησης	activation record / stack frame	Ο χώρος στη στοίβα για μία κλήση συνάρτησης.	13
διάγραμμα ροής	flowchart	Σχήμα με ρόμβους (συνθήκες) και ορθογώνια (εντολές)	6
διαίρει και βασίλευε	divide and conquer	Διαίρεση σε υποπροβλήματα, αναδρομική λύση, συνδυασμός.	17, 18
διαρροή μνήμης	memory leak	Μνήμη που δεσμεύτηκε και δεν αποδεσμεύτηκε ποτέ.	13, 25
διάσχιση	traversal	Επίσκεψη όλων των κόμβων με μια συγκεκριμένη σειρά.	21, 22
διαχειριστής	root	Χρήστης με πλήρη δικαιώματα στο σύστημα	1, 20, 22
διεπαφή	interface	Τι προσφέρει ένα κομμάτι του προγράμματος στα άλλα.	23, 24
διεπαφή γραμμής εντολών	CLI, terminal, console	Αλληλεπίδραση με εντολές κειμένου	1
διεύθυνση	address	Η θέση ενός κελιού (byte) στη μνήμη	2, 9, 10, 11
διπλή αποδέσμευση	double free	Δεύτερη `free` στον ίδιο pointer.	13
δισδιάστατος πίνακας	two-dimensional array	Πίνακας από πίνακες, `a[γραμμές][στήλες]`.	12
δομή / εγγραφή	struct / record	Συλλογή πεδίων που περιγράφουν μια οντότητα· νέος τύπος.	19

Ελληνικά	English	Ορισμός	Κεφάλαια
δομή δεδομένων	data structure	Τρόπος οργάνωσης δεδομένων στη μνήμη για αποδοτική χρήση.	10
δομημένος προγραμματισμός	structured programming	Προγράμματα μόνο από ακολουθία, επιλογή, επανάληψη	8
δυαδική αναζήτηση	binary search	Σύγκριση με το μέσο και συνέχεια στο μισό· $O(\log n)$.	17
δυαδικό δέντρο	binary tree	Δενδρική διάταξη κόμβων με 0 έως 2 παιδιά ο καθένας.	20, 21, 22
δυαδικό δέντρο αναζήτησης	binary search tree (BST)	Δέντρο με μικρότερα αριστερά και μεγαλύτερα δεξιά σε κάθε κόμβο.	21, 22
δυαδικό σύστημα	binary	Αρίθμηση με βάση το 2	2
δυαδικό ψηφίο	bit (binary digit)	Η μικρότερη μονάδα πληροφορίας: 0 ή 1	2
δυναμική βιβλιοθήκη	dynamic library (`.so`)	Βιβλιοθήκη που φορτώνεται στην εκτέλεση, π.χ. `libm.so`	26
δυναμικός πίνακας	dynamic array	Πίνακας με μέγεθος που αποφασίζεται κατά την εκτέλεση.	12, 13
δυναμικός προγραμματισμός	dynamic programming	Επίλυση υποπροβλημάτων από τα μικρότερα στα μεγαλύτερα, με αποθήκευση των λύσεων σε πίνακα.	25
δύο δείκτες	two pointers	Δύο θέσεις που κινούνται στα δεδομένα για να αποφύγουν εμφωλευμένους βρόχους.	25
εκτελέσιμο	executable	Αρχείο που μπορεί να τρέξει ο επεξεργαστής	0
εκτελέσιμο, δυαδικό	executable, binary	Το πρόγραμμα σε κώδικα μηχανής, π.χ. `a.out`	1
έκφραση	expression	Συνδυασμός τελεστών και τελεστέων με τιμή	5
εκφυλισμένο δυαδικό δέντρο	degenerate binary tree	Κάθε κόμβος έχει έως ένα παιδί.	22
έλεγχος εκδόσεων	version control	Σύστημα που κρατάει το ιστορικό όλων των αλλαγών	4
εμβέλεια	scope	Το μέρος του προγράμματος όπου ένα όνομα είναι ορατό.	14
εμφωλευμένες if	nested if	`if` μέσα στο σώμα άλλης `if` ή `else`	6

Ελληνικά	English	Ορισμός	Κεφάλαια
ενδιάμεση μνήμη	buffer	Όπου περιμένουν οι χαρακτήρες πριν φτάσουν στο πρόγραμμα.	9
ένθετη / εμφωλευμένη δομή	nested struct	Δομή που είναι πεδίο άλλης δομής.	19
ενσωματωμένος κανόνας	built-in rule	Rule που το Make ξέρει ήδη (π.χ. `.o` από `.c`)	26
εντολή	statement	Συντακτική δομή που εκτελείται	5, 6
εντολή break	break statement	Τερματίζει αμέσως τον βρόχο ή τη `switch`	8
εντολή continue	continue statement	Προχωρά στην επόμενη επανάληψη του βρόχου	8
εντολή goto	goto statement	Άλμα σε εντολή με ετικέτα στην ίδια συνάρτηση	8
εντολή switch	switch statement	Επιλογή περίπτωσης με βάση μια ακέραια τιμή	8
εντολή έκφρασης	expression statement	Μια έκφραση που τελειώνει με `;`	6
ένωση	union	Τύπος σαν τη δομή, όπου όλα τα μέλη μοιράζονται την ίδια μνήμη.	20
εξάρτηση	dependency	Ένα αρχείο χρειάζεται ένα άλλο για να χτιστεί.	23, 24
έξοδος σφάλματος	standard error	Το ρεύμα `stderr`, για μηνύματα λάθους.	18
επανάληψη	iteration	Μία εκτέλεση του σώματος του βρόχου	6
επέκταση	extension	Το τέλος του ονόματος (`.txt`, `.c`) που δείχνει τον τύπο	1, 18
επεξεργαστής κειμένου	editor	Πρόγραμμα για τη σύνταξη του κώδικα (π.χ. `vim`, VS Code).	7
επιθεματικός / προθεματικός	postfix / prefix	`a++` (παλιά τιμή) / `++a` (νέα τιμή)	4
επίπεδο κόμβου	node level	Πόσοι κόμβοι μεσολαβούν ως τη ρίζα· η ρίζα είναι στο 1.	21, 22
επισκίαση	shadowing	Εσωτερική δήλωση με ίδιο όνομα κρύβει την εξωτερική.	14
ετικέτα	label	Όνομα με `:` μπροστά από εντολή, στόχος της `goto`	8
ετικέτα δομής	struct tag	Το όνομα μετά το `struct`, π.χ. `student`.	19
ευθυγράμμιση μνήμης	memory alignment	Διευθύνσεις πεδίων πολλαπλάσιες του 4 ή του 8.	19

Ελληνικά	English	Ορισμός	Κεφάλαια
θέση	index	Ο αριθμός (από 0) που επιλέγει ένα στοιχείο του πίνακα.	10
ισορροπημένο / εκφυλισμένο δέντρο	balanced / degenerate binary tree	Ύψη υποδέντρων που διαφέρουν ≤ 1 / κάθε κόμβος με ≤ 1 παιδί.	21
ισορροπημένο δυαδικό δέντρο	balanced binary tree	Σε κάθε κόμβο τα ύψη των υποδέντρων διαφέρουν έως 1.	22
κανόνας	rule	`target: prerequisites` μαζί με το recipe	26
κανόνας μοτίβου	wildcard / pattern rule	Rule με `%` που ορίζει οικογένεια rules	26
κανονική λειτουργία	canonical mode	Το τερματικό στέλνει την είσοδο ανά γραμμή.	9
κατάλογος, φάκελος	directory, folder	Περιέχει αρχεία και άλλους καταλόγους	1
κατάσταση ροής	flow	Κατάσταση πλήρους απορρόφησης και συγκέντρωσης σε μια δραστηριότητα.	7
καταχώρηση	commit	Αποθήκευση των αλλαγών στο τοπικό repository	4
καταχωρητής	register	ταχύτατη θέση αποθήκευσης μέσα στη CPU	24
κάτω όριο	lower bound, Ω	$g = \Omega(f)$: η g είναι τουλάχιστον $c \cdot f$ για μεγάλα n.	15
κέλυφος	shell	Πρόγραμμα που τρέχει τις εντολές που πληκτρολογούμε	1
κενή εντολή	empty / null statement	Το σκέτο `;`, που δεν κάνει τίποτα (no-op)	6
κενοί χαρακτήρες	whitespace	Κενά, tabs και αλλαγές γραμμής.	10
κενός δείκτης	null pointer (`NULL`)	Δείκτης με τιμή 0 που δεν δείχνει πουθενά.	11, 12, 13
κενός τύπος	`void`	Τύπος χωρίς τιμές· π.χ. συνάρτηση που δεν επιστρέφει τίποτα.	13
κεφαλή / ουρά	head / tail	Το πρώτο / (συνήθως) το τελευταίο στοιχείο της λίστας.	21
κινητή υποδιαστολή	floating point	Αναπαράσταση πραγματικών (`float`, `double`), κατά προσέγγιση.	9

Ελληνικά	English	Ορισμός	Κεφάλαια
κλήση κατά τιμή	call by value	Η συνάρτηση παίρνει αντίγραφα των ορισμάτων.	16
κλήση συνάρτησης	function call	Εκτέλεση της συνάρτησης με συγκεκριμένα ορίσματα	3
κόμβος	node	Ένα στοιχείο λίστας ή δέντρου (μια δομή).	20
κόμβος / παιδί	node / child	Στοιχείο του δέντρου / κόμβος ακριβώς κάτω από έναν άλλο.	22
κρεμασμένο else	dangling else	Αμφισημία για το σε ποια `if` ανήκει ένα `else`	6
κύρια μνήμη	primary memory (RAM)	Προσωρινή μνήμη με άμεση πρόσβαση από τον επεξεργαστή	2
κώδικας-μακαρονάδα	spaghetti code	Κώδικας με μπερδεμένη ροή, δύσκολος στην ανάγνωση	8
κωδικός εξόδου	exit code	Η τιμή που επιστρέφει η `main`· `0` = επιτυχία	1, 3
λάθος κατά ένα	off-by-one error	Βρόχος που κάνει μία επανάληψη παραπάνω ή λιγότερο λόγω λάθους στο όριο.	7
λειτουργικό σύστημα	operating system (OS)	Λογισμικό που διαχειρίζεται υλικό και πόρους και εξυπηρετεί τα προγράμματα	1
λογικός τελεστής	logical operator	`&&`, `||`, `!`	5
λογισμικό / υλικό	software / hardware	Τα προγράμματα / η ίδια η συσκευή	0
μακροεντολή	macro	Όνομα (με ή χωρίς παραμέτρους) που αντικαθίσταται με κείμενο.	15, 26
μέγεθος	size	Πόσα στοιχεία έχει ο πίνακας· στατικό μετά τη δήλωση.	10
μέση / χειρότερη περίπτωση	average / worst case	Κόστος κατά μέσο όρο / για τη χειρότερη είσοδο.	18
μεταβλητή	variable	Τμήμα της μνήμης με όνομα και τύπο	2
μεταβλητή-σημαία	flag	Μεταβλητή 0/1 που καταγράφει αν συνέβη κάτι	8, 16
μεταγλώττιση υπό συνθήκη	conditional compilation	Κράτημα ή αφαίρεση κώδικα με `#if` / `#ifdef`.	15
μεταγλωττιστής	compiler	Μετατρέπει πηγαίο κώδικα σε γλώσσα μηχανής (π.χ. `gcc`)	0, 1, 3, 15
μέτωπο	frontier / worklist	Οι κόμβοι που περιμένουν επεξεργασία στη BFS.	21, 22

Ελληνικά	English	Ορισμός	Κεφάλαια
μη έγκυρος δείκτης	invalid pointer	Δείκτης που δεν κρατά έγκυρη διεύθυνση.	11
μη προσημασμένος	unsigned	Ακέραιος τύπος χωρίς αρνητικές τιμές	2
μήκος λίστας	list length	Ο αριθμός των στοιχείων της λίστας.	20
μονάδα μετάφρασης	translation unit	Ένα αρχείο `.c` μαζί με ό,τι φέρνουν τα `#include` του	26
μοναδιαίος / δυαδικός / τριαδικός	unary / binary / ternary	Τελεστής με έναν / δύο / τρεις τελεστέους	5
μονοπάτι	path	Η θέση ενός αρχείου στην ιεραρχία, π.χ. `/home/users`	1
μονοπάτι	filepath	Η πλήρης θέση ενός αρχείου, π.χ. `/home/…/students.txt`.	18
μορφοποιητής κώδικα	code formatter	Εργαλείο (π.χ. `clang-format`) που ξαναγράφει τον κώδικα σύμφωνα με ένα στυλ.	7
N-αδικό δέντρο	N-ary tree	Δέντρο με περισσότερα από 2 παιδιά ανά κόμβο.	22
οδηγία	directive	Γραμμή που αρχίζει με #, π.χ. `#include`	1, 3
οδηγία προεπεξεργαστή	preprocessor directive	Γραμμή που αρχίζει με #, π.χ. `#include`.	15
οκταδικό σύστημα	octal	Αρίθμηση με βάση το 8	2
ολοκληρωμένο περιβάλλον ανάπτυξης	IDE	Editor με ενσωματωμένη μεταγλώττιση, τερματικό και debugger.	7
οπισθοδρόμηση	backtracking	Επιστροφή σε προηγούμενο κόμβο για να δοκιμαστεί άλλος κλάδος.	22
όρισμα	argument	Τιμή που δίνουμε σε ένα πρόγραμμα όταν το τρέχουμε	1, 3
ορίσματα γραμμής εντολών	command-line arguments	Οι λέξεις της κλήσης, στα `argc`/`argv`.	12
ορισμός συνάρτησης	function definition	Τύπος, όνομα, ορίσματα και σώμα	3
παγκόσμια / στατική μνήμη	global / static memory	Μνήμη για μεταβλητές που ζουν όσο το πρόγραμμα.	13, 14

Ελληνικά	English	Ορισμός	Κεφάλαια
παγκόσμια μεταβλητή	global variable	Δηλώνεται έξω από συναρτήσεις· ορατή ως το τέλος του αρχείου.	14
παλινδρομικός	palindrome	Που διαβάζεται ίδια και από τις δύο μεριές.	16
παραγοντικό	factorial	$n! = 1 \cdot 2 \cdots n$, με $0! = 1$.	11
παράμετρος εξόδου	output parameter	Δείκτης μέσω του οποίου η συνάρτηση γράφει ένα αποτέλεσμα.	16
παρωχημένο	out of date	Target που λείπει ή είναι παλαιότερο από prerequisite του	26
πεδίο / μέλος	field / member	Μία από τις μεταβλητές μέσα σε μια δομή.	19
πεδίο bit	bit field	Μέλος δομής με δηλωμένο πλήθος bits (`int year : 3;`).	20
περιγραφέας αρχείου	file descriptor (FD)	Ο ακέραιος που ταυτίζει ένα ανοιχτό αρχείο (`fileno`).	18
περίπτωση	case	Σημείο εισόδου της `switch` για μια σταθερά	8
πηγαίο αρχείο	source file	Αρχείο κειμένου με τον κώδικα, π.χ. `helloworld.c`	0
πηγαίος κώδικας	source code	Το πρόγραμμα όπως το γράφουμε, π.χ. `hello.c`	1
πίνακας	array	Σύνολο στοιχείων ίδιου τύπου σε συνεχόμενες θέσεις μνήμης, με ένα όνομα.	10
πίνακας / θέση	array / index	Στοιχεία ίδιου τύπου σε συνεχόμενη μνήμη / ο `i` στο `a[i]`.	12
πίνακας ASCII	ASCII table	Αντιστοίχιση κωδικών 0–127 σε χαρακτήρες	2
πίνακας από δείκτες	array of pointers	Πίνακας με στοιχεία τύπου δείκτη, π.χ. `char *s[5]`.	12
πίνακας εκτέλεσης	trace table	Πίνακας με τις τιμές των μεταβλητών σε κάθε βήμα μιας εκτέλεσης με το χέρι.	25
πλάτος πεδίου	field width	Ο αριθμός στο `%6s`: μέγιστοι χαρακτήρες που θα διαβαστούν.	18
πλήθος τελεστέων	arity	Μοναδιαίος (1), δυαδικός (2), τριαδικός (3)	4

Ελληνικά	English	Ορισμός	Κεφάλαια
πλήρες δυαδικό δέντρο	complete binary tree	Γεμάτα επίπεδα εκτός ίσως του τελευταίου, που γεμίζει από αριστερά.	22
πολυπλοκότητα	complexity	Μέτρο απόδοσης ενός αλγορίθμου ως συνάρτηση του μεγέθους του προβλήματος.	15
προαπαιτούμενο	prerequisite	Target από το οποίο εξαρτάται ένα άλλο target	26
πρόγραμμα	program	Η καταγραφή της επίλυσης ενός προβλήματος, ως σύνολο εντολών	0
προγραμματισμός	programming	Σαφής καθορισμός διαδικασίας που λύνει πρόβλημα υπολογισμού	0
προγραμματισμός σε ζεύγη	pair programming	Driver γράφει, navigator ελέγχει, αλλάζουν ρόλους	3, 4
προεπεξεργαστής	preprocessor	Πρώτο στάδιο του μεταγλωττιστή· μετασχηματίζει το κείμενο του κώδικα.	15
προεπιλογή	default	Η περίπτωση όταν δεν ταιριάζει κανένα `case`	8
προθεματικά αθροίσματα	prefix sums	Πίνακας με τα αθροίσματα των πρώτων i στοιχείων, για αθροίσματα διαστημάτων σε $O(1)$.	25
προθεματικός / επιθεματικός	prefix / postfix	Πριν / μετά τη μεταβλητή (`++a` / `a++`)	5
προσδιοριστής const	const qualifier	Δηλώνει ότι μια θέση μνήμης δεν αλλάζει.	23
προσδιοριστικό μορφοποίησης	format specifier	`%` και χαρακτήρες, π.χ. `%d`	2, 3
προσεταιριστικότητα	associativity	Σειρά υπολογισμού για ίση προτεραιότητα	4, 5
προσωρινή λίστα	staging area	Οι αλλαγές που θα μπουν στο επόμενο commit	4
προτεραιότητα	precedence	Ποιος τελεστής εφαρμόζεται πρώτος	4, 5
προτροπή	prompt	Το `user@host:dir$` πριν από κάθε εντολή	1
πρότυπη είσοδος	standard input (`stdin`)	Η προεπιλεγμένη είσοδος, συνήθως το πληκτρολόγιο.	9

Ελληνικά	English	Ορισμός	Κεφάλαια
πρότυπη είσοδος / έξοδος	standard input / output	Τα ρεύματα `stdin` / `stdout` του προγράμματος.	18
πρωτότυπο συνάρτησης	function prototype	Όνομα, τύπος επιστροφής και ορίσματα, χωρίς σώμα.	23, 24
πτώση	fall-through	Συνέχιση στο επόμενο `case` όταν λείπει το `break`	8
πυρήνας	kernel	Το κεντρικό μέρος του OS, που μιλάει με το υλικό	1
ρεύμα	stream	Ένα ανοιχτό αρχείο, ως `FILE *`.	18
ρίζα	root directory	Ο κατάλογος `/`, κορυφή της ιεραρχίας	1
ρίζα / φύλλο	root / leaf	Ο πρώτος κόμβος / κόμβος χωρίς παιδιά.	21
ροή ελέγχου	control flow	Η σειρά εκτέλεσης των εντολών	5, 6
σειρά bytes	endianness	Η σειρά αποθήκευσης των bytes ενός ακεραίου.	12, 13
σειριακή αναζήτηση	linear search	Έλεγχος των στοιχείων ένα-ένα μέχρι να βρεθεί το ζητούμενο.	10
σιωπηρή μετατροπή	implicit type conversion	Αυτόματη μετατροπή στον «μεγαλύτερο» τύπο	4, 5
σταθερά απαρίθμησης	enumeration constant	Ένα από τα ονόματα μιας απαρίθμησης, π.χ. `Mon`.	20
σταθερή / λογαριθμική / γραμμική	constant / logarithmic / linear	$O(1)$ / $O(\log n)$ / $O(n)$.	15
σταθερή συμβολοσειρά	string literal	Κείμενο σε εισαγωγικά μέσα στον κώδικα.	14
στατική μεταβλητή	static variable	Κρατά την τιμή της ανάμεσα σε κλήσεις, ως το τέλος του προγράμματος.	14
στοίβα	stack	Συνεχόμενη μνήμη LIFO για τοπικές μεταβλητές και ορίσματα.	13
στοιχείο διαμέρισης	pivot element	Το στοιχείο γύρω από το οποίο γίνεται η διαμέριση.	17, 18
στοίχιση	indentation	Τα κενά στην αρχή κάθε γραμμής που δείχνουν το επίπεδο εμφώλευσης.	7
στόχος	target	Ό,τι μπορεί να παραχθεί, συνήθως ένα αρχείο	26

Ελληνικά	English	Ορισμός	Κεφάλαια
συγκριτικός τελεστής	comparison / relational operator	`==`, `!=`, `<`, `>`, `<=`, `>=`	5
συγχώνευση	merge	Ένωση δύο ταξινομημένων ακολουθιών σε μία ταξινομημένη.	17, 18
συμβόλαιο	contract	Εγγύηση που δίνει η δήλωση μιας συνάρτησης στους χρήστες της.	23, 24
σύμβολο	symbol	Όνομα συνάρτησης ή μεταβλητής σε αρχείο αντικειμένου.	14
συμβολοσειρά	string	Κείμενο ανάμεσα σε διπλά εισαγωγικά, π.χ. `"Hello world\n"`	0, 10, 14
συμπλήρωμα ως προς 2	two's complement	Αναπαράσταση αρνητικών: flip όλα τα bits και +1	2
συνάρτηση	function	Υπολογισμός από εισόδους σε έξοδο, με όνομα	3
σύνδεση	linking	Ένωση αντικειμενικών αρχείων σε εκτελέσιμο.	23, 24
συνδέτης	linker	Ενώνει αντικειμενικά αρχεία και βιβλιοθήκες σε εκτελέσιμο	0, 3, 26
συνδυαστικός τελεστής ανάθεσης	compound assignment	`+=`, `-=`, …	5
συνένωση	concatenation	Προσάρτηση ενός string στο τέλος άλλου.	14
σύνθετη εντολή	compound statement / block	Εντολές μέσα σε `{ }`, που μετράνε ως μία	6
σύνολο ορισμού / τιμών	domain / co-domain	Τα σύνολα εισόδων και εξόδων	3
συνταγή	recipe	Οι εντολές shell που φτιάχνουν ένα target	26
συνώνυμο τύπου	`typedef`	Νέο όνομα για υπάρχοντα τύπο.	19
συσσωρευτής	accumulator	Μεταβλητή που μαζεύει ένα αποτέλεσμα (άθροισμα, γινόμενο) κατά τη διάρκεια ενός βρόχου.	7
σύστημα κατασκευής	build system	Εργαλείο που αυτοματοποιεί το χτίσιμο ενός project	26
σφάλμα κατάτμησης	segmentation fault	Τερματισμός από πρόσβαση σε μη επιτρεπτή μνήμη.	11

Ελληνικά	English	Ορισμός	Κεφάλαια
σφάλμα σύνδεσης	linking error	Ο linker δεν βρίσκει υλοποίηση συμβόλου (`undefined reference`)	26
σχόλιο	comment	Κείμενο μέσα σε `/* */` που αγνοεί ο μεταγλωττιστής	0, 1, 7
σωρός	heap	Η περιοχή μνήμης από την οποία δεσμεύει η `malloc`.	12, 13
τάξη μεγέθους	order of growth, Θ	Ισχύουν ταυτόχρονα O και Ω.	15
ταξινομημένη ακολουθία	sorted sequence	$a_i \leq_\alpha a_j$ για κάθε $i \leq j$.	17
ταξινόμηση	sorting	Αναδιάταξη μιας ακολουθίας ώστε να γίνει ταξινομημένη.	17, 18
ταξινόμηση εισαγωγής	insertion sort	Εισάγει κάθε στοιχείο σε ταξινομημένο πρόθεμα.	17
ταξινόμηση επιλογής	selection sort	Φέρνει κάθε φορά το ελάχιστο του υπολοίπου μπροστά.	17
ταξινόμηση συγχώνευσης	merge sort	Ταξινομεί τα δύο μισά και τα συγχωνεύει.	17
ταξινόμηση φυσαλίδας	bubblesort	Αντιμεταθέτει γειτονικά στοιχεία σε λάθος σειρά.	17
ταχυταξινόμηση	quicksort	Διαμερίζει γύρω από ένα pivot και ταξινομεί τα δύο μέρη.	17
τεκμηρίωση	documentation	Ονόματα, σχόλια και `README` που κάνουν ένα πρόγραμμα κατανοητό.	7
τέλειο / γεμάτο / πλήρες δέντρο	perfect / full / complete binary tree	Βλ. «Τύποι δυαδικών δέντρων».	21
τέλειο δυαδικό δέντρο	perfect binary tree	Όλοι οι εσωτερικοί κόμβοι με 2 παιδιά, όλα τα φύλλα στο ίδιο επίπεδο.	22
τελεστέος	operand	Μεταβλητή ή σταθερά πάνω στην οποία δουλεύει ο τελεστής	4, 5
τελεστής	operator	Σύμβολο που κάνει υπολογισμό και επιστρέφει τιμή	4, 5
τελεστής ανάθεσης	assignment operator	`=`· αναθέτει και επιστρέφει την τιμή	4, 5
τελεστής αύξησης / μείωσης	increment / decrement operator	`++` / `--`	5
τελεστής βέλους	arrow operator (`->`)	`ptr->f` ισοδυναμεί με `(*ptr).f`.	19
τελεστής μετατροπής	cast operator	`(τύπος)τελεστέος`, ρητή μετατροπή τύπου	4, 5

Ελληνικά	English	Ορισμός	Κεφάλαια
τελεστής παράθεσης	comma operator	`a, b`: υπολογίζει το `a`, επιστρέφει το `b`	4, 5
τελεστής συνθήκης	conditional operator	`σ ? α : β`, ο μόνος τριαδικός τελεστής	5
τέλος αρχείου	End-Of-File (`EOF`)	Τιμή (`-1`) που δηλώνει ότι δεν υπάρχει άλλη είσοδος.	9, 10
τετραγωνική / εκθετική	quadratic / exponential	$O(n^2)$ / $O(2^n)$.	15
τιμή επιστροφής	return value	Η τιμή που δίνει πίσω μια συνάρτηση.	9
τοπική μεταβλητή	local variable	Δηλώνεται σε συνάρτηση· ορατή ως το τέλος του block.	14
τρέχων / γονικός κατάλογος	current / parent directory	`.` και `..`	1
τυπική έξοδος	standard output (stdout)	Το αρχείο όπου γράφει η `printf`	2, 9
τύπος	type	Πόση μνήμη πιάνει μια τιμή και πώς ερμηνεύεται	2, 3
τύπος ορισμένος από τον χρήστη	user-defined type	Τύπος που ορίζει το πρόγραμμα, όπως μια δομή.	19
υλικό	hardware	Οι φυσικές συσκευές του υπολογιστή	1
υπερχείλιση	overflow	Αποτέλεσμα που δεν χωράει στον τύπο της μεταβλητής	8, 11
υπερχείλιση / υποχείλιση	overflow / underflow	Πρόσβαση μετά το τέλος / πριν την αρχή ενός πίνακα.	10
υπερχείλιση buffer	buffer overflow	Εγγραφή πέρα από το τέλος ενός πίνακα.	14
υπερχείλιση ακεραίων	integer overflow	Αποτέλεσμα που δεν χωράει στον τύπο του	2, 3
υπερχείλιση στοίβας	stack overflow	Η στοίβα ξεπερνά το όριό της, π.χ. από ατέρμονη αναδρομή.	13
υποδέντρο	subtree	Ένα παιδί μαζί με όλους τους απογόνους του.	22
υποεντολή	subcommand	Λέξη στο `argv[1]` που επιλέγει τι θα κάνει το πρόγραμμα.	16
υπολογιστής	computer	Κατασκευή που επεξεργάζεται δεδομένα και παράγει αποτελέσματα	0
φύλλο	leaf	Κόμβος χωρίς παιδιά.	20, 22

Ελληνικά	English	Ορισμός	Κεφάλαια
χειρότερη / μέση περίπτωση	worst / average case	Κόστος για τη δυσκολότερη / κατά μέσο όρο είσοδο.	17
χειρότερη περίπτωση	worst case	Η είσοδος που κάνει τον αλγόριθμο να δουλέψει περισσότερο.	16
χρήση μετά την αποδέσμευση	use after free	Πρόσβαση σε μνήμη μετά την `free` της.	13
χρονική / χωρική πολυπλοκότητα	time / space complexity	Πώς αυξάνεται ο χρόνος / η μνήμη με το n.	15
χρονική πολυπλοκότητα	time complexity	Πώς αυξάνονται τα βήματα με το μέγεθος της εισόδου.	16
χρόνος ζωής	lifetime	Το διάστημα της εκτέλεσης όπου υπάρχει η μεταβλητή.	14
χωρική πολυπλοκότητα	space complexity	Πόση επιπλέον μνήμη χρειάζεται σε σχέση με την είσοδο.	16

Τράπεζα ασκήσεων

Στα θέματα της online εξέτασης Δεκεμβρίου 2023 (`exam-2023-fall-*`) ισχύει για όλες τις ασκήσεις: τα προγράμματα πρέπει να είναι ευανάγνωστα, αποδοτικά σε χώρο και χρόνο και να έχουν έξοδο ίδια με τα παραδείγματα εκτέλεσης. Για είσοδο εκτός προδιαγραφών το πρόγραμμα τερματίζει με exit code 1 και μήνυμα σφάλματος.

Κεφάλαιο 0: Καλημέρα Κόσμε!

Kahoot από το αμφιθέατρο (K0.1–K0.6)

K0.1 · Τι κάνει η printf

K0.1 · *Kahoot «Καλημέρα Κόσμε - lec00!», «Καλημέρα Κόσμε!» (διάλεξη 0)* · ★☆☆ · multiple-choice · 94% σωστές · `kahoot-printf-prints`

Τι κάνει η συνάρτηση `printf`;

- Τυπώνει
- Διαβάζει
- Δεν θυμάμαι :)

Υπόδειξη Θυμηθείτε το πρώτο πρόγραμμα του μαθήματος, το Hello World.

K0.2 · Εκτύπωση με νέα γραμμή

K0.2 · *Kahoot «Καλημέρα Κόσμε - lec00!», «Καλημέρα Κόσμε!» (διάλεξη 0)* · ★☆☆ · multiple-choice · 86% σωστές · `kahoot-printf-newline`

Παραλλαγή θέματος εξέτασης: ποια από τις παρακάτω γραμμές τυπώνει `Michael Jordan` ακολουθούμενο από νέα γραμμή;

- `print "Michael Jordan";`
- `printf("Michael Jordan\n");`
- `printf("Michael Jordan");`
- `print(Lebron James);`

Υπόδειξη Ελέγξτε το όνομα της συνάρτησης, τις παρενθέσεις και τα εισαγωγικά, και ποια ακολουθία διαφυγής αλλάζει γραμμή.

K0.3 · Η φόρμα του μαθήματος

K0.3 · *Kahoot «Καλημέρα Κόσμε - lec00!», «Καλημέρα Κόσμε!» (διάλεξη 0)* · ★☆☆ · multiple-choice · 85% σωστές · `kahoot-course-form`

Δεν συμπλήρωσα τη φόρμα του μαθήματος, υπάρχει περίπτωση να βαθμολογηθούν οι ασκήσεις μου;

- True
- False

Υπόδειξη Θυμηθείτε τη λίστα με τα βήματα εγγραφής στα εργαλεία του μαθήματος, και σκεφτείτε πώς αντιστοιχίζονται οι υποβολές σας με εσάς.

K0.4 · Τα εργαλεία του μαθήματος

K0.4 · *Kahoot «Καλημέρα Κόσμε - lec00!» (διάλεξη 0)* · ★☆☆ · multiple-choice · 82% σωστές · `kahoot-course-tools`

Ποιο από τα παρακάτω είναι εργαλείο του μαθήματος;

- Piazza
- GitHub
- E-Class
- Όλα τα παραπάνω

Υπόδειξη Θυμηθείτε πού γίνονται οι συζητήσεις, πού υποβάλλονται οι ασκήσεις και πού βρίσκονται οι ανακοινώσεις.

K0.5 · Το χαρακτηριστικό της C

K0.5 · *Kahoot «Καλημέρα Κόσμε - lec00!» (διάλεξη 0)* · ★★☆ · multiple-choice · 61% σωστές · `kahoot-c-main-feature`

Ποιο είναι το κύριο χαρακτηριστικό της γλώσσας C;

- Δημοφιλής
- Γρήγορη/Αποδοτική
- Χρησιμοποιείται παντού
- Όλα τα παραπάνω

Συχνή παρανόηση Το 27% διάλεξε «Γρήγορη/Αποδοτική»: η ταχύτητα είναι πράγματι ένα από τα πλεονεκτήματα της C, αλλά όχι το μόνο που τονίστηκε.

Υπόδειξη Σκεφτείτε τους λόγους που δόθηκαν στη διάλεξη για την επιλογή της C: ισχύει μόνο ένας;

K0.6 · Έχω μια ερώτηση για το μάθημα

K0.6 · *Kahoot «Καλημέρα Κόσμε - lec00!», «Καλημέρα Κόσμε!» (διάλεξη 0)* · ★★☆ · multiple-choice · 57% σωστές · `kahoot-ask-question`

Έχω μια ερώτηση για το μάθημα, τι κάνω;

- Την κρατάω μέσα μου και περιμένω να μου έρθει η επιφοίτηση
- Στέλνω email στον Αυγερινό
- Κατευθείαν ποστάρω στο piazza με τον τίτλο ”Ερώτηση”
- Εάν *δεν* έχει απαντηθεί στο piazza, ανοίγω νέο θέμα με περιγραφικό τίτλο

Συχνή παρανόηση Το 23% θα πόσταρε κατευθείαν στο Piazza με τίτλο «Ερώτηση»: το Piazza είναι το σωστό κανάλι, αλλά πρώτα ψάχνουμε αν έχει ήδη απαντηθεί, και ο τίτλος πρέπει να λέει τι ρωτάμε.

Υπόδειξη Σκεφτείτε πόσοι συμφοιτητές σας μπορεί να έχουν την ίδια απορία, και πώς θα βρουν εκείνοι (και εσείς) μια απάντηση που ήδη υπάρχει.

Ζέσταμα: από τις διαφάνειες (A0.1–A0.7)

A0.1 · Έχω μια ερώτηση, τι κάνω;

A0.1 · *Διάλεξη 0: Καλημέρα Κόσμε!, διαφάνεια 16* · ★☆☆ · multiple-choice · `slides-lec00-ask-question`

Έχω μια ερώτηση για το μάθημα, τι κάνω;

- A. Στέλνω email στον Αυγερινό/Τάκη
- B. Ελέγχω αν έχει απαντηθεί στο piazza και εφόσον *δεν* έχει απαντηθεί, ποστάρω καινούρια ερώτηση

Υπόδειξη Σκεφτείτε πόσοι φοιτητές έχουν πιθανότατα την ίδια απορία με εσάς, και ποιος τρόπος επικοινωνίας κάνει την απάντηση διαθέσιμη σε όλους.

A0.2 · Ψωμί και αυγά

A0.2 · *Διάλεξη 0: Καλημέρα Κόσμε!, διαφάνεια 26* · ★☆☆ · short-answer · `slides-lec00-bread-and-eggs`

Γιατί επινοήσαμε τις γλώσσες προγραμματισμού;

> Joe says to his programmer friend Charlie:
>
> "Go to the store and buy a loaf of bread. If they have eggs, buy a dozen."
>
> Charlie returns with 12 loaves of bread.

Εξηγήστε πώς προκύπτουν οι δύο ερμηνείες της οδηγίας. Τι είδους αμφισημία είναι αυτή (λεξιλογική ή συντακτική); Δώστε και ένα παράδειγμα του άλλου είδους.

Υπόδειξη Ρωτήστε «δώδεκα από τι;». Η λεξιλογική αμφισημία αφορά μία λέξη με πολλές σημασίες, ενώ η συντακτική αφορά τον τρόπο που συνδέονται οι λέξεις μιας πρότασης.

A0.3 · Hello World σε online compiler

A0.3 · *Διάλεξη 0: Καλημέρα Κόσμε!, διαφάνεια 32* · ★☆☆ · tooling · `slides-lec00-hello-world`

Ας τρέξουμε το παρακάτω πρόγραμμα με έναν online compiler:

```c
/* File: helloworld.c */
#include <stdio.h>

int main() {
  printf("Hello world\n");
}
```

Τι τυπώνει; Στη συνέχεια αλλάξτε το ώστε να τυπώνει δύο γραμμές, και δοκιμάστε τι συμβαίνει αν αφαιρέσετε το `\n`.

Υπόδειξη Κάθε κλήση της `printf` τυπώνει ακριβώς ό,τι βρίσκεται ανάμεσα στα διπλά εισαγωγικά· η αλλαγή γραμμής γίνεται μόνο όπου υπάρχει `\n`.

A0.4 · Το πιο ψηλό βουνό του κόσμου

A0.4 · *Διάλεξη 0: Καλημέρα Κόσμε!, διαφάνεια 25* · ★☆☆ · short-answer · `slides-lec00-highest-mountain`

Quiz: Ποιο είναι το πιο ψηλό βουνό του κόσμου;

Τι μας διδάσκει η ερώτηση αυτή για το αν μπορούμε να δίνουμε οδηγίες σε έναν υπολογιστή σε ανθρώπινη γλώσσα;

Υπόδειξη Πριν απαντήσετε, ορίστε τι σημαίνει «ψηλό»: ύψος από τη στάθμη της θάλασσας, απόσταση από το κέντρο της Γης ή ύψος από τη βάση ως την κορυφή; Κάθε ορισμός δίνει άλλο βουνό.

Α0.5 · Περνάω χωρίς εργασίες και εργαστήριο;

A0.5 · *Διάλεξη 0: Καλημέρα Κόσμε!, διαφάνεια 11* · ★☆☆ · short-answer · `slides-lec00-pass-without-labs`

Με βάση τον τρόπο βαθμολόγησης του μαθήματος:

- πρωτοετείς: 50% Τελική Εξέταση + 30% Ασκήσεις + 20% Εργαστήριο·
- αλλιώς: 70% Τελική Εξέταση + 30% Ασκήσεις·

όπου οι Ασκήσεις είναι προαιρετικές: άρα μπορώ να περάσω χωρίς να κάνω εργασίες/εργαστήριο;

Υπόδειξη Χωρίστε την απάντηση σε δύο μέρη: τι επιτρέπουν οι αριθμοί (θεωρητικά) και πώς προετοιμάζεται κανείς για μια γραπτή εξέταση προγραμματισμού (στην πράξη). «In theory, theory and practice are the same. In practice, they are not.»

Α0.6 · Παραδείγματα προγραμμάτων

A0.6 · *Διάλεξη 0: Καλημέρα Κόσμε!, διαφάνεια 23* · ★☆☆ · short-answer · `slides-lec00-programs-you-know`

Poll: Έχει κάποιο παιδί προγραμματίσει; Κάποια παραδείγματα προγραμμάτων που ξέρετε;

Υπόδειξη Πρόγραμμα είναι η καταγραφή της επίλυσης ενός προβλήματος. Σκεφτείτε ποιο πρόβλημα λύνει κάθε εφαρμογή που χρησιμοποιείτε καθημερινά (στο κινητό, στον υπολογιστή, ακόμα και σε ένα πλυντήριο), ποια δεδομένα δέχεται και ποια αποτελέσματα παράγει.

Α0.7 · Γιατί προγραμματισμός το 2025;

A0.7 · *Διάλεξη 0: Καλημέρα Κόσμε!, διαφάνεια 5* · ★☆☆ · short-answer · `slides-lec00-why-programming`

Γιατί να ασχοληθείς με τον προγραμματισμό το 2025; Δώστε τουλάχιστον τρεις λόγους.

Υπόδειξη Σκεφτείτε σε πόσα διαφορετικά επιστημονικά πεδία και σε πόσες χώρες χρειάζεται κάποιος που γράφει κώδικα, πού βρίσκονται οι μεγάλες τεχνολογικές εξελίξεις των τελευταίων δεκαετιών, και τι είδους δουλειά είναι η επίλυση προβλημάτων.

Κεφάλαιο 1: Η Γραμμή Εντολών

Kahoot από το αμφιθέατρο (K1.1–K1.10)

K1.1 · Η καταγωγή του Linux

K1.1 · *Kahoot «Command Line» (διάλεξη 1)* · ★☆☆ · multiple-choice · 87% σωστές · `kahoot-linux-unix`

Το Linux είναι βασισμένο σε ένα λειτουργικό σύστημα που λέγεται

- Multics
- Simics
- Unix
- Κανένα από τα παραπάνω

Υπόδειξη Το Linux είναι ένα λειτουργικό σύστημα «τύπου …»· σκεφτείτε από ποιο σύστημα των Bell Labs πήρε τη φιλοσοφία του.

K1.2 · Το 2ο όρισμα του echo

K1.2 · *Kahoot «Command Line» (διάλεξη 1)* · ★☆☆ · multiple-choice · 73% σωστές · `kahoot-echo-second-arg`

Στην εντολή `/bin/echo hello good world` ποιο είναι το 2ο όρισμα του προγράμματος;

- `echo`
- `hello`
- `good`
- `world`

Υπόδειξη Το shell χωρίζει τη γραμμή σε λέξεις με βάση τα κενά· η πρώτη λέξη είναι το πρόγραμμα που εκτελείται, και τα ορίσματα μετράνε από τη λέξη μετά από αυτό.

K1.3 · Η εντολή man

K1.3 · *Kahoot «Τύπωμα και Συναρτήσεις» (διάλεξη 3)* · ★☆☆ · multiple-choice · 72% σωστές · `kahoot-man-command`

Ποια εντολή τρέχω για να βρω τι κάνει ένα πρόγραμμα στο σύστημά μου;

- cat
- man
- woman
- tac

Υπόδειξη Ψάχνουμε το εγχειρίδιο (manual) του προγράμματος· η εντολή είναι συντομογραφία αυτής της λέξης.

K1.4 · Στο Linux όλα είναι...

K1.4 · *Kahoot «Command Line» (διάλεξη 1)* · ★☆☆ · multiple-choice · 71% σωστές · `kahoot-everything-is-a-file`

Στο Linux (σχεδόν) όλα είναι ένα:

- Ψέμα
- Αρχείο
- IP Address
- Κλικ

Υπόδειξη Θυμηθείτε τη φιλοσοφία του Unix για το σύστημα αρχείων: ακόμα και οι συσκευές εμφανίζονται κάτω από το `/dev`.

K1.5 · Δικαιώματα υπερχρήστη

K1.5 · *Kahoot «Command Line» (διάλεξη 1)* · ★★☆ · multiple-choice · 68% σωστές · `kahoot-sudo`

Ποια εντολή μας δίνει δικαιοδοσία υπερ-χρήστη (root) σε ένα σύστημα Linux;

- `doit`
- `root`
- `sudo`
- Run as Administrator

Υπόδειξη Το όνομα της εντολής σημαίνει «κάνε το ως (super)user»: μπαίνει μπροστά από την εντολή που θέλουμε να τρέξουμε με αυξημένα δικαιώματα.

K1.6 · Το ΛΣ των web servers

K1.6 · *Kahoot «Command Line» (διάλεξη 1)* · ★★☆ · multiple-choice · 65% σωστές · `kahoot-linux-servers`

Το λειτουργικό σύστημα που τρέχει το 90% των web servers σήμερα είναι το

- Linux
- Limux
- MacOS
- Windows

Υπόδειξη Είναι το ίδιο ελεύθερο λειτουργικό σύστημα που χρησιμοποιούμε στα εργαστήρια του μαθήματος (προσοχή στην ορθογραφία των επιλογών).

K1.7 · Το basename ενός αρχείου

K1.7 · *Kahoot «Command Line» (διάλεξη 1)* · ★★☆ · multiple-choice · 53% σωστές · `kahoot-basename`

Ποιο είναι το basename του αρχείου `/home/users/ethan/example/hello.txt`;

- `/home/users/ethan/example/`
- `example`
- `hello.txt`
- Δεν υπάρχει

Υπόδειξη Το basename είναι το όνομα του ίδιου του αρχείου, χωρίς τους καταλόγους που οδηγούν σε αυτό.

K1.8 · Φορτίο απομακρυσμένου server

K1.8 · *Kahoot «Command Line» (διάλεξη 1)* · ★★☆ · multiple-choice · 53% σωστές · `kahoot-uptime`

Netflix Interview Question: Με ποια εντολή θα έλεγχες το workload ενός απομακρυσμένου server;

- `ls`
- `uptime`
- `cat`
- `users`

Υπόδειξη Σκεφτείτε ποια από τις βασικές εντολές δείχνει, εκτός από την ώρα, και πόσο «απασχολημένο» είναι το σύστημα.

K1.9 · Εκτύπωση περιεχομένων αρχείου

K1.9 · *Kahoot «Command Line» (διάλεξη 1)* · ★★★ · multiple-choice · 29% σωστές · `kahoot-cat-prints-file`

Ποια από τις παρακάτω εντολές τυπώνει τα περιεχόμενα ενός αρχείου;

- `printf`
- `cat`
- `dog`
- `echo`

Συχνή παρανόηση Το 47% διάλεξε `printf`, μπερδεύοντας τη συνάρτηση της C (και την ομώνυμη εντολή του shell), που τυπώνει το κείμενο που της δίνουμε, με μια εντολή που διαβάζει αρχεία.

Υπόδειξη Ποια εντολή παίρνει ως όρισμα ένα όνομα αρχείου (και όχι ένα κείμενο) και στέλνει ό,τι περιέχει στην έξοδο;

K1.10 · Το μονοπάτι ενός αρχείου

K1.10 · *Kahoot «Command Line» (διάλεξη 1)* · ★★★ · multiple-choice · 20% σωστές · `kahoot-filepath`

Ποιο είναι το filepath του αρχείου `/home/users/ethan/example/hello.txt`;

- `/home/users/ethan/example/`
- `example`
- `hello.txt`
- `/home/users/ethan/example/hello.txt`

Συχνή παρανόηση Το 66% διάλεξε `/home/users/ethan/example/`, δηλαδή τον κατάλογο (dirname) που περιέχει το αρχείο· το filepath όμως περιλαμβάνει και το όνομα του αρχείου.

Υπόδειξη Το μονοπάτι πρέπει να προσδιορίζει μοναδικά *ποιο* αρχείο εννοούμε, όχι μόνο τον κατάλογο στον οποίο βρίσκεται.

Ζέσταμα: από τις διαφάνειες (A1.1–A1.5)

A1.1 · Παραδείγματα για κάθε επίπεδο ενός υπολογιστικού συστήματος

A1.1 · *Διάλεξη 1: Η Γραμμή Εντολών, διαφάνεια 7* · ★☆☆ · short-answer · `slides-lec01-architecture-examples`

Η αρχιτεκτονική ενός υπολογιστικού συστήματος περιγράφεται από την αλυσίδα:

```
Users / Χρήστες --(Χρησιμοποιούν)--> Applications or Programs / Εφαρμογές ή
↪ Προγράμματα
 --(Χρειάζονται)--> Operating System / Λειτουργικό Σύστημα --(Χρειάζεται)-->
↪ Hardware / Υλικό
```

Παραδείγματα;

Υπόδειξη Για κάθε κουτί της αλυσίδας σκεφτείτε κάτι που χρησιμοποιείτε καθημερινά: ποιες εφαρμογές ανοίγετε, σε ποιο λειτουργικό σύστημα τρέχουν και σε ποιες συσκευές.

A1.2 · Πρόγραμμα και ορίσματα στο /bin/echo

A1.2 · *Διάλεξη 1: Η Γραμμή Εντολών, διαφάνεια 15* · ★☆☆ · tooling · `slides-lec01-echo-arguments`

Για να τρέξουμε ένα πρόγραμμα με N ορίσματα πρέπει να γράψουμε:

`Πρόγραμμα(Όρισμα0) Όρισμα1 Όρισμα2 … ΌρισμαN`

Παράδειγμα:

```
/bin/echo hello good world
```

Ποιο είναι το πρόγραμμα; Πόσα ορίσματα έχει;

Υπόδειξη Τα ορίσματα χωρίζονται με κενά. Προσέξτε τι ρόλο έχει η πρώτη λέξη της γραμμής σύμφωνα με τη γενική μορφή, και μετά τρέξτε την εντολή για να δείτε τι τυπώνει.

A1.3 · Τρέξτε μόνοι σας τις εντολές της διάλεξης

A1.3 · *Διάλεξη 1: Η Γραμμή Εντολών, διαφάνεια 42* · ★☆☆ · tooling · `slides-lec01-try-commands`

Εγκαταστήστε WSL και προσπαθήστε να τρέξετε τις εντολές που είπαμε μόνοι/ες σας!

Συγκεκριμένα, στη διάλεξη είδαμε:

- τις εντολές `who`, `last`, `whoami`, `id`, `users`, `factor 2394872891`, `hostname`, `sleep 3`, `uptime`, `uname -a`, `top`, `man`, `ps` (διαφάνεια 25)·

- τον χειρισμό αρχείων με `pwd`, `ls`, `ls -l`, `cat`, `mkdir`, `cd`, `cp`, `rm`, `rmdir` (διαφάνειες 29–30)·
- τη μεταγλώττιση και εκτέλεση του Hello World με `gcc hello.c`, `./a.out`, `file a.out`, `gcc -o hello hello.c`, `./hello` (διαφάνειες 33–34), και τον κωδικό εξόδου με `echo $?` (διαφάνεια 39).

Υπόδειξη Αν δεν έχετε Windows, αρκεί ένα τερματικό σε Linux ή macOS, ή σύνδεση με `ssh` στα μηχανήματα του εργαστηρίου. Συγκρίνετε την έξοδο κάθε εντολής με αυτή των διαφανειών και εξηγήστε τις διαφορές (άλλος χρήστης, άλλο μηχάνημα, άλλη ώρα). Για τις διαδραστικές εντολές, βγαίνετε με `q`.

A1.4 · Γιατί χρειαζόμαστε λειτουργικό σύστημα

A1.4 · *Διάλεξη 1: Η Γραμμή Εντολών, διαφάνεια 9* · ★☆☆ · short-answer · `slides-lec01-why-os`

Operating System: Το λογισμικό του υπολογιστή που είναι υπεύθυνο για:

- Την διαχείριση του υλικού του υπολογιστή (συσκευές)
- Την διαχείριση πόρων του συστήματος (πόση μνήμη μπορεί να χρησιμοποιηθεί από ένα πρόγραμμα, πότε θα πάρει κύκλους στον επεξεργαστή, κ.ο.κ.)
- Προσφορά υπηρεσιών σε προγράμματα χρηστών (αποθήκευση πληροφοριών στον δίσκο, μεταφορά πληροφοριών μέσω δικτύου, κ.ο.κ.)

Γιατί να μην είναι το πρόγραμμά μας υπεύθυνο για αυτά;

Υπόδειξη Σκεφτείτε πόσα προγράμματα τρέχουν ταυτόχρονα στον υπολογιστή σας και ότι μοιράζονται την ίδια μνήμη, τον ίδιο επεξεργαστή και τον ίδιο δίσκο. Σκεφτείτε επίσης πόσα διαφορετικά μοντέλα συσκευών θα έπρεπε να «ξέρει» κάθε πρόγραμμα.

A1.5 · Εξερευνήστε κι άλλα βασικά προγράμματα

A1.5 · *Διάλεξη 1: Η Γραμμή Εντολών, διαφάνεια 31* · ★★☆ · tooling · `slides-lec01-explore-coreutils`

Υπάρχουν και άλλα βασικά προγράμματα που δεν κοιτάξαμε (στα coreutils ή εκτός) τα οποία προτείνουμε να εξερευνήσετε:

`find, grep, sort, df -h, du -sh, wc -l, file, which, basename, dirname, ping, curl`

Για καθένα, βρείτε τι κάνει και τρέξτε το τουλάχιστον μία φορά σε κάποιο αρχείο ή κατάλογο του λογαριασμού σας.

Υπόδειξη Ξεκινήστε από το `man` κάθε εντολής (π.χ. `man wc`) και διαβάστε τις πρώτες γραμμές της περιγραφής και τις επιλογές που αναφέρει η διαφάνεια (`-h`, `-sh`, `-l`). Δοκιμάστε τις σε αρχεία που ήδη έχετε, όπως το `hello.c`.

Εργαστήριο (A1.6–A1.10)

A1.6 · Πλοήγηση στο σύστημα αρχείων

A1.6 · *Εργαστήριο 1, Βήμα 1* · ★☆☆ · tooling · `lab-lab01-step1-navigation`

Σε ένα από τα μηχανήματα Linux του εργαστηρίου, εκτελέστε με τη σειρά:

1. Εμφανίστε τα περιεχόμενα του καταλόγου /usr/include.

2. Εμφανίστε το πλήρες όνομα του τρέχοντος καταλόγου.

3. Δημιουργήστε κάτω από τον τρέχοντα κατάλογο ένα νέο κατάλογο με όνομα myWork.

4. Μεταβείτε σε αυτόν τον νέο κατάλογο.

Υπόδειξη Χρειάζεστε τέσσερις εντολές: μία για λίστα περιεχομένων, μία για τον τρέχοντα κατάλογο, μία για δημιουργία καταλόγου και μία για αλλαγή καταλόγου. Προσέξτε τη διαφορά απόλυτου μονοπατιού (ξεκινά από `/`) και σχετικού μονοπατιού, και ότι τα ονόματα είναι case-sensitive.

A1.7 · Μεταγλώττιση, σύνδεση και εκτέλεση

A1.7 · *Εργαστήριο 1, Βήμα 2* · ★☆☆ · tooling · `lab-lab01-step2-compile-link`

Συνεχίζοντας μέσα στον κατάλογο myWork:

5. Αντιγράψτε στον κατάλογο αυτό το πηγαίο αρχείο `~iphw/samples/mysin.c`

6. Δημιουργήστε το εκτελέσιμο αρχείο mysin από το πηγαίο αρχείο που μόλις αντιγράψατε.

7. Τρέξτε το εκτελέσιμο αυτό.

8. Αντιγράψτε στον τρέχοντα κατάλογο το πηγαίο αρχείο `~iphw/samples/mymain.c` (διατηρήστε το όνομα `mymain.c`).

9. Αντιγράψτε στον τρέχοντα κατάλογο το αντικειμενικό αρχείο `~iphw/samples/myfunct.o`

10. Μεταγλωττίστε το πηγαίο αρχείο `mymain.c` στο αντίστοιχο αντικειμενικό αρχείο.

11. Συνδέστε το αντικειμενικό αρχείο που κατασκευάσατε και το αντικειμενικό αρχείο myfunct.o δημιουργώντας το εκτελέσιμο αρχείο myprog.

12. Εκτελέστε το myprog και ακολουθήστε τις οδηγίες που θα εκτυπωθούν.

13. Διαγράψτε όλα τα αρχεία που δημιουργήσατε καθώς και τον κατάλογο myWork.

Υπόδειξη Το `mysin.c` χρησιμοποιεί τη μαθηματική βιβλιοθήκη, οπότε η σύνδεση θέλει την κατάλληλη επιλογή του gcc. Διακρίνετε τα δύο στάδια: με `-c` παίρνετε αντικειμενικό αρχείο `.o`, και με `-o` πάνω σε αντικειμενικά αρχεία τα συνδέετε σε εκτελέσιμο. Για να διαγράψετε τον κατάλογο πρέπει πρώτα να βγείτε από αυτόν.

A1.8 · Αντιγραφή, μετονομασία και προβολή αρχείων

A1.8 · *Εργαστήριο 1, Βήμα 3* · ★☆☆ · tooling · `lab-lab01-step3-copy-view`

14. Αντιγράψτε στον τρέχοντα κατάλογο το αρχείο `stdio.h` από τον κατάλογο `/usr/include`

15. Μετονομάστε το αρχείο που αντιγράψατε σε `Mystdio.h`

16. Εμφανίστε στην οθόνη τα περιεχόμενα του αρχείου αυτού.

Υπόδειξη Ο τρέχων κατάλογος γράφεται σύντομα ως `..` Η μετονομασία στο Unix είναι «μετακίνηση» με νέο όνομα. Για την προβολή, δοκιμάστε και την επιλογή που αριθμεί τις γραμμές.

A1.9 · Δικαιώματα προστασίας

A1.9 · *Εργαστήριο 1, Βήμα 4* · ★☆☆ · tooling · `lab-lab01-step4-permissions`

17. Δημιουργήστε στον τρέχοντα κατάλογο ένα κενό αρχείο με όνομα .my_file (προσέξτε την τελεία στην αρχή του ονόματος).

18. Εμφανίστε τα δικαιώματα προστασίας του αρχείου αυτού και όποιες άλλες χρήσιμες πληροφορίες μας δίνει το λειτουργικό σύστημα για αυτό.

19. Αλλάξτε τα δικαιώματα προστασίας έτσι ώστε εσείς (ο ιδιοκτήτης του) να έχετε όλα τα δικαιώματα προστασίας (ανάγνωσης, εγγραφής και εκτέλεσης), τα μέλη της ομάδας στην οποία ανήκει το αρχείο να έχουν δικαιώματα ανάγνωσης και εκτέλεσης και οι υπόλοιποι να έχουν μόνο δικαίωμα ανάγνωσης.

20. Εμφανίστε πάλι τα δικαιώματα προστασίας του αρχείου .my_file (και τις άλλες πληροφορίες).

21. Εμφανίστε όλα τα αρχεία του τρέχοντος καταλόγου, μαζί με τα δικαιώματα προστασίας (και τις άλλες πληροφορίες), μη συμπεριλαμβανομένων των αρχείων που το όνομά τους αρχίζει από τελεία.

22. Το ίδιο με το προηγούμενο, αλλά να συμπεριληφθούν και τα αρχεία που το όνομά τους αρχίζει από τελεία.

Υπόδειξη Τα δικαιώματα της `chmod` είναι τρία οκταδικά ψηφία (ιδιοκτήτης, ομάδα, υπόλοιποι), το καθένα άθροισμα των 4 (ανάγνωση), 2 (εγγραφή) και 1 (εκτέλεση). Το `.my_file` είναι κρυφό αρχείο: σκεφτείτε ποια επιλογή της `ls` εμφανίζει και τα κρυφά αρχεία.

A1.10 · Αναζήτηση, ανακατεύθυνση και σωληνώσεις

A1.10 · *Εργαστήριο 1, Βήμα 5* · ★★☆ · tooling · `lab-lab01-step5-grep-pipes`

23. Μέσω της εντολής `man` ενημερωθείτε σχετικά με τις δυνατότητες της εντολής grep.

24. Χρησιμοποιήστε την εντολή `grep` για να βρείτε μέσα στο αρχείο `/usr/include/stdlib.h` τις γραμμές που περιλαμβάνουν το κείμενο size.

25. Επαναλάβετε την προηγούμενη εντολή αλλά να κάνετε ανακατεύθυνση της εξόδου της στο αρχείο grepout.txt

26. Συνδυάστε με κάποιον τρόπο τις εντολές ls και grep για να βρείτε στον τρέχοντα κατάλογο αρχεία στα οποία ο ιδιοκτήτης έχει δικαιώματα ανάγνωσης και εγγραφής, αλλά όχι εκτέλεσης και όλοι οι υπόλοιποι δεν έχουν κανένα δικαίωμα.

27. Αντιγράψτε στον τρέχοντα κατάλογο το πηγαίο αρχείο `~iphw/samples/capitalize.c` δημιουργήστε το αντίστοιχο εκτελέσιμο και τρέξτε το με ανακατεύθυνση της εισόδου από το ίδιο το πηγαίο αρχείο `capitalize.c` στέλνοντας την έξοδο (πάλι με ανακατεύθυνση) στο αρχείο `CAPITALIZE.c`

28. Δοκιμάστε να μεταγλωττίσετε το αρχείο `CAPITALIZE.c`. Μην ανησυχείτε όμως αν δεν μπορεί να μεταγλωττιστεί (γιατί άραγε;)

Υπόδειξη Για το 26, η έξοδος της `ls -l` είναι κείμενο: γράψτε το ζητούμενο μοτίβο δικαιωμάτων (π.χ. πώς φαίνεται το «rw- --- ---») και φιλτράρετέ το με `grep` μέσω σωλήνωσης. Για το 28, σκεφτείτε τι έπαθαν οι λέξεις-κλειδιά της C και τα ονόματα των συναρτήσεων όταν έγιναν κεφαλαία.

Εργασίες (A1.11–A1.12)

A1.11 · Ξεκινώντας με την γραμμή εντολών (bandit)

A1.11 · *Εργασία 0 (2023-24), Άσκηση 2* · ★★☆ · tooling · `hw-2023-hw0-bandit`

Έχεις χρησιμοποιήσει την γραμμή εντολών; Linux; Γνωρίζεις πως να κάνεις SSH σε απομακρυσμένους servers και να λύνεις προβλήματα με εντολές κονσόλας; Αν όχι, καιρός να μάθουμε!

Προκειμένου να ξεκινήσεις, πήγαινε στην σελίδα: overthewire. Για να ολοκληρώσεις την άσκηση, θα χρειαστεί να περάσεις όλα τα επίπεδα - μέχρι και το 10ο. Σε κάθε επίπεδο, απαιτείται να συνδεθείς μέσω SSH σε έναν διαφορετικό server με συγκεκριμένο port number και στην συνέχεια να λύσεις ένα μικρό πρόβλημα προκειμένου να αποκαλυφθεί ο κωδικός (ένα string μορφής `Iequeiw2geecaeHee0losaa3aek`) το οποίο είναι ο κωδικός SSH για το επόμενο επίπεδο.

Τεχνικές Προδιαγραφές

- Repository Name: `progintro/hw0-<YourUsername>`
- Αρχείο Λύσης Filepath: `command/solution.txt`
- README Filepath: `command/README.md`
- **Κάθε γραμμή που θα προστεθεί στο αρχείο `solution.txt` πρέπει να γίνει με διαφορετικό git commit. Κοινώς, θέλουμε να δούμε *12* διαφορετικά git commits. Η κάθε γραμμή πρέπει να προστεθεί όταν λύσετε το αντίστοιχο πρόβλημα, όχι όλες μαζί στο τέλος.**

Το περιεχόμενο του `solution.txt` πρέπει να έχει την ακόλουθη μορφή:

```
$ cat solution.txt
bandit0: bandit0
bandit1: ...
bandit2: ...
...
bandit10: ...
bandit11: ...
```

Όπου οι τελείες (...) περιγράφουν το μέρος της λύσης που έχει παραληφθεί. Μια σωστή λύση, θα έχει το ακόλουθο SHA hash:

```
$ sha256sum solution.txt
a95444caa1805c4efa936a720d2761ee0f2f24af8347fbec7e6bd2834607a95c  solution.txt
```

Η λύση πρέπει να περιέχει όλους τους κωδικούς μέχρι και τον bandit11. Στο αρχείο `README.md` πρέπει να προσθέσετε λεπτομέρειες για τον τρόπο που λύσατε τα επίπεδα. Αν επιθυμείτε να λύσετε και τα επόμενα επίπεδα (φτάνουν μέχρι το 33! αλλά απαιτούν γνώσεις από μελλοντικά μαθήματα) προφανώς κάτι τέτοιο είναι ευπρόσδεκτο και λεπτομέρειες μπορούν να προστεθούν στο αρχείο `README.md`.

Υπόδειξη Κάθε επίπεδο του bandit γράφει στη σελίδα του ποιες εντολές θα σας χρειαστούν (`ls -a`, `cat`, `file`, `find`, `grep`, `sort`, `uniq`, `strings`, `base64`, `tr` κ.ά.)· διαβάστε τα `man` τους. Προσέξτε ονόματα αρχείων που ξεκινούν με `-` ή έχουν κενά. Κρατήστε τον κωδικό κάθε επιπέδου μόλις τον βρείτε και κάντε αμέσως commit, γιατί ζητούνται 12 ξεχωριστά commits· στο τέλος

ελέγξτε το αρχείο με `sha256sum` (προσοχή στην ακριβή μορφή κάθε γραμμής και στην αλλαγή γραμμής στο τέλος).

Δείτε επίσης

- Υποδειγματική υποβολή φοιτητή (περιέχει λύση: δείτε την αφού λύσετε την άσκηση)

A1.12 · Παιχνίδια με Κονσόλα (cmdline)

A1.12 · *Εργασία 0 (2025-26), Άσκηση 2* · ★★☆ · tooling · `hw-2025-hw0-cmdline`

Έχεις χρησιμοποιήσει την γραμμή εντολών; Linux; Γνωρίζεις πως να κάνεις SSH σε απομακρυσμένους servers και να λύνεις προβλήματα με εντολές κονσόλας; Αν όχι, καιρός να μάθουμε! Αν ναι, τότε αυτή η άσκηση θα σου είναι παιχνιδάκι.

Σε αυτήν την άσκηση θα μπούμε με ssh σε έναν απομακρυσμένο server που βρίσκεται στην IP διεύθυνση `35.169.96.19` μέσω τερματικού και θα ανακτήσουμε συγκεκριμένες πληροφορίες. Υπάρχουν διαφορετικοί χρήστες (`byte0`, `byte1`, ...) που αντιστοιχούν σε διαφορετικά προβλήματα/υποερωτήματα. Για όλους τους χρήστες ο κωδικός για να μπείτε είναι `bits&bytes`. Σε κάθε ένα υποερώτημα θα αναζητούμε ένα string που μπορεί να είναι μια μόνο λέξη ή πολλές λέξεις διαχωρισμένες με τον χαρακτήρα underscore _, για παράδειγμα: `something_like_that`.

byte0: Αλλάζοντας Φάκελο (cd) Ο χρήστης byte0 είναι ελαφρώς μανιακός με την οργανωτικότητα. Του αρέσει να οργανώνει όλα του τα αρχεία σε φακέλους, υποφακέλους κ.ο.κ.. Με αυτά και με αυτά όμως, δεν μπορεί να βρει το αρχείο του, μπορείτε να τον βοηθήσετε; Για να μπείτε στον server τρέξτε:

```
ssh byte0@35.169.96.19
```

και βάλτε τον κωδικό που γράψαμε παραπάνω. Η εντολή `cd` μπορεί να σας φανεί χρήσιμη όπως και η εντολή `cat` (ίσως και το tab / autocomplete :)). Όταν διαβάσετε τα περιεχόμενα του αρχείου, σημειώστε το string που βρήκατε μέσα στο αρχείο ώστε να το προσθέσετε στο `solution.txt` (δες παρακάτω στις τεχνικές προδιαγραφές).

byte1: Μια Εντολή Άγνωστη Ο χρήστης byte1 έφτιαξε ένα καινούριο πρόγραμμα με άγνωστη σε μας λειτουργία ονόματι `supercalifragilisticexpialidocious`. Πως μπορείτε να βρείτε τι κάνει;

byte2: Τα Άπαντα του Shakespeare (grep) Ο χρήστης byte2 έχει δύο βασικά χαρακτηριστικά: (1) είναι μεγάλος θαυμαστής του Shakespeare και (2) είναι κάπως μυστικοπαθής. Αποφάσισε να συνδυάσει τα δύο και καταχώνιασε ένα μυστικό string μέσα στην συλλογή του με τα άπαντα του Shakespeare. Μπορείτε να τον βοηθήσετε να το βρεί;

Το μόνο που γνωρίζουμε είναι πως το κρυφό κείμενο περιέχει το string "will find". Η εντολή `grep` μπορεί να σας φανεί χρήσιμη.

byte3: Τα Άπαντα του Shakespeare Άλλαξαν (diff) Ο χρήστης byte3 αποφάσισε να αλλάξει κάτι στα άπαντα του Shakespeare ελπίζοντας πως η αλλαγή θα περάσει απαρατήρητη. Ελέγξαμε τον αριθμό των λέξεων και χαρακτήρων του κάθε αρχείου:

```
byte3@ip-172-31-37-131:~$ wc -w *.txt
  901325 shakespeare.modified.txt
  901325 shakespeare.txt
 1802650 total
byte3@ip-172-31-37-131:~$ wc -c *.txt
 5458203 shakespeare.modified.txt
 5458199 shakespeare.txt
```

και παρατηρήσαμε πως ενώ ο αριθμός λέξεων παρέμεινε ίδιος (901.325), ο αριθμός των χαρακτήρων άλλαξε. Επομένως, υποψιαζόμαστε πως άλλαξε μια λέξη του αυθεντικού με μια καινούρια. Βρείτε την καινούρια λέξη—η εντολή `diff` ίσως σας φανεί χρήσιμη.

byte4: Λαβύρινθος (find) Ο χρήστης byte4 είναι συγγενής του byte2 παραπάνω και ελαφρώς πιο μυστικοπαθής. Προκειμένου να "ανεβάσει" το παιχνίδι του αποφάσισε να κρύψει τα δεδομένα του σε ένα δαιδαλώδες κατασκεύασμα από φακέλους. Μπορείτε και πάλι να τον βοηθήσετε να βρει το μυστικό που έχει κρύψει μέσα σε ένα από όλα τα αρχεία των υποφακέλων; Το μόνο που γνωρίζουμε είναι πως έχει κρύψει το μυστικό του σε ένα αρχείο που λέγεται `cup.txt`. Η εντολή `find` μπορεί να σας φανεί χρήσιμη.

byte5: Compile and Run (gcc, mkdir, cp) Ο byte5 έγραψε ένα προγραμματάκι σε C ονόματι `byte5.c`. Μεταγλωττίστε το και τρέξτε το προκειμένου να πάρετε τον κωδικό σας! Προσοχή: ο default φάκελος του byte5 δεν σας επιτρέπει να χρησιμοποιήσετε αρχεία, ίσως χρειαστεί να φτιάξετε έναν καινούριο υποφάκελο μέσα στον φάκελο `/tmp` προκειμένου να τοποθετήσετε το εκτελέσιμό σας. Οι εντολές `mkdir`, `cp`, `gcc` μπορεί να σας φανούν χρήσιμες.

byte6: Αποσυμπίεση αρχείου 1 (unzip) Κάποιες φορές προκειμένου να μικρύνουμε τον χώρο που πιάνει ένα αρχείο το συμπιέζουμε. Ο χρήστης byte6 έχει ένα αρχείο `byte6.zip` το οποίο χρειάζεται αποσυμπίεση. Ανοίξτε το για να αποκτήσετε πρόσβαση στα περιεχόμενά του. Το πρόγραμμα `unzip` μπορεί να σας βοηθήσει.

byte7: Αποσυμπίεση αρχείου 2 (tar) Υπάρχουν διάφοροι τρόποι και εργαλεία για να συμπιέσεις αρχεία και τα περιεχόμενα τους. Ο χρήστης byte7 χρησιμοποίησε ένα πρόγραμμα που λέγεται `tar` - μπορείτε να χρησιμοποιήσετε το ίδιο προκειμένου να βρείτε τα περιεχόμενα του `byte7.tar.gz`;

byte8: Carriage Return (xxd, pico, vim) Ο χρήστης byte8 μας είπε πως ο κωδικός που ψάχνουμε είναι μέσα στο αρχείο `carriage_return.txt`. Όμως δοκιμάσαμε να το τυπώσουμε:

```
byte8@ip-172-31-37-131:~$ ls
carriage_return.txt
byte8@ip-172-31-37-131:~$ cat carriage_return.txt
There is absolutely nothing to see here. Move along.
```

και δεν είδαμε κάτι. Μπορείς να μας βοηθήσεις; Πιστεύουμε πως οι εντολές `xxd`, `pico` ή `vim` μπορεί να φανούν χρήσιμες.

byte9: Ένα Περίεργο Όνομα (cat) Ο χρήστης byte9 ανέφερε πως ο κωδικός του είναι μέσα στον φάκελο `/home/byte9` - μπορείς να μας βοηθήσεις να τον ανακτήσουμε; Ίσως σε βοηθήσει να θυμηθείς τι είναι το filepath (μονοπάτι) ενός αρχείου.

byte10: Το 42ο Όνομα (sort, head) Ο χρήστης byte10 έχει ένα αρχείο γεμάτο με ονόματα και θέλει να τα ταξινομήσει αλφαβητικά και να βρει το 42ο όνομα στην σειρά. Γνωρίζει ότι το 1ο όνομα είναι το Aarika - μπορείτε να τον βοηθήσετε να βρει το 42ο; Οι εντολές `sort` και `head` μπορούν να συνδυαστούν και να βοηθήσουν.

byte11: Επιλογές Ονόματος (sort, uniq) Ο χρήστης byte11 μόλις απέκτησε ένα παιδάκι και προκειμένου να μην το κοροϊδεύουν στο σχολείο αποφάσισε να του δώσει το πιο συχνό όνομα εκείνης της χρονιάς. Στο αρχείο `births.txt` περιέχονται τα ονόματα όλων των φετινών γεννήσεων. Μπορείτε να τον βοηθήσετε να βρει το πιο συχνό όνομα; Οι εντολές `sort` και `uniq` ίσως σας φανούν χρήσιμες.

byte12: Βρες το Μονοπάτι (which, PATH) Ο byte12 είναι κάπως αφηρημένος. Μόλις έφτιαξε το πρόγραμμα `tuxsay` αλλά δεν θυμάται που το τοποθέτησε και δεν πρόλαβε να φτιάξει ένα manual page. Παρόλα αυτά, μπορεί να το τρέξει κανονικά. Μπορείτε να βρείτε το basename του φακέλου μέσα στον οποίο βρίσκεται το πρόγραμμα `tuxsay`;

byte13: Commit Messages (git log) Ο byte13 μόλις πήρε πρόσβαση σε ένα git repository (αποθετήριο) και θέλει να δει τα commit messages (μηνύματα για προηγούμενες αλλαγές) που έκαναν οι χρήστες που συνεισέφεραν στο repository. Μπορείτε να διαβάσετε αυτά τα μηνύματα;

byte14: Ιστορικό (git log -p) Ο byte14 ξεκίνησε να γράφει μια νουβέλα με την βοήθεια του AI. Μετά από αρκετές αλλαγές διαπίστωσε ότι το AI είχε προσθέσει (κατά λάθος;) έναν από τους κωδικούς του μέσα στο κείμενο. Ευτυχώς το είδε, το αφαίρεσε, έκανε commit τις αλλαγές και πλέον ξέρει δεν είναι ορατό σε άλλους χρήστες. Έχει δίκιο ο byte14;

byte15: Open the Vault (bonus / προαιρετικό) Ο byte15 είναι χρήστης του vault, ενός υπερασφαλούς προγράμματος που εμφανίζει τα περιεχόμενα του λογαριασμού σου, μόνο

εφόσον γνωρίζεις το μυστικό PIN. Δυστυχώς, ο byte15 ξέχασε το PIN του! Μπορείτε να τον βοηθήσετε να ανακτήσει τα περιεχόμενα του λογαριασμού του;

Τεχνικές Προδιαγραφές Η υποβολή σου πρέπει να έχει τα ακόλουθα χαρακτηριστικά:

- Repository Name: `progintro/hw0-<YourUsername>`
- README Filepath: `cmdline/README.md`
- Αρχείο Λύσης Filepath: `cmdline/solution.txt`
- **Κάθε γραμμή που θα προστεθεί στο αρχείο `solution.txt` πρέπει να γίνει με διαφορετικό git commit. Κοινώς, θέλουμε να δούμε τουλάχιστον *15* διαφορετικά git commits. Η κάθε γραμμή πρέπει να προστεθεί όταν λύσετε το αντίστοιχο πρόβλημα, όχι όλες μαζί στο τέλος.**

Το περιεχόμενο του `solution.txt` πρέπει να έχει την ακόλουθη μορφή:

```
$ cat solution.txt
byte0: ...
byte1: ...
byte2: ...
...
```

Όπου οι τελείες (...) περιγράφουν το μέρος της λύσης που έχει παραληφθεί. Στο αρχείο `README.md` πρέπει να γράψετε μια σύντομη περιγραφή για τον τρόπο που λύσατε το κάθε πρόβλημα.

Αν σου άρεσε αυτή η άσκηση ή έχεις κολλήσει με ένα από τα παραπάνω και δεν ξέρεις πως να συνεχίσεις, υπάρχουν και άλλα site online στα οποία μπορείς να εξασκηθείς όπως για παράδειγμα το overthewire τα οποία μπορούν να σου δώσουν ιδέες. Σημείωση: μέχρι το επίπεδο 11 τα προβλήματα σε αυτό το site είναι εντός θέματος για το μάθημα, από εκεί και πέρα γίνονται πιο δύσκολα / απαιτούν γνώσεις που θα αποκτήσουμε σε μεγαλύτερα έτη.

Εμφανίσεις

- Εργασία 0 (2024-25), Άσκηση 2: 12 επίπεδα (`byte0`–`byte11`) στον server `52.86.144.139`, 12 commits· το `byte10` ζητά το 10ο όνομα αντί για το 42ο.
- Εργασία 0 (2025-26), Άσκηση 2: η εκφώνηση που δίνεται εδώ, με τα νέα επίπεδα `byte12`–`byte15` (`which`/`PATH`, `git log`, `git log -p`, bonus vault) και τουλάχιστον 15 commits.

Υπόδειξη Κάθε επίπεδο δοκιμάζει μία ή δύο εντολές, και το `man <εντολή>` (ή το `--help`) είναι ο πρώτος σου σύμμαχος, ακόμα και για άγνωστα προγράμματα. Για τα επίπεδα με πολλά δεδομένα συνδύασε εντολές με σωληνώσεις (`|`), π.χ. ταξινόμηση πριν από μέτρηση διπλοτύπων. Τα ονόματα αρχείων που αρχίζουν με παύλα ή έχουν κενά, τα κρυφά αρχεία (τελεία στην αρχή), οι χαρακτήρες ελέγχου μέσα σε ένα αρχείο και το ιστορικό ενός git repository είναι οι συνηθισμένες «παγίδες» που αξίζει να έχεις στο μυαλό σου.

Κεφάλαιο 2: Μνήμη και Μεταβλητές

Kahoot από το αμφιθέατρο (K2.1–K2.15)

K2.1 · Bit ή byte;

K2.1 · *Kahoot «Μεταβλητές και Μνήμη», «Μεταβλητές και Συναρτήσεις» (διάλεξη 2)* · ★☆☆ · multiple-choice · 93% σωστές · `kahoot-bit-vs-byte`

Ποια μονάδα πληροφορίας είναι μεγαλύτερη;

- Bit
- Byte

Υπόδειξη Θυμηθείτε από πόσα bits αποτελείται ένα byte.

K2.2 · Άθροισμα θετικών int

K2.2 · *Kahoot «Μεταβλητές και Μνήμη», «Μεταβλητές και Συναρτήσεις» (διάλεξη 2)* · ★☆☆ · multiple-choice · 87% σωστές · `kahoot-int-overflow-sign`

Αν προσθέσω δύο θετικούς αριθμούς τύπου `int`, το αποτέλεσμα είναι πάντα θετικό;

- Ναι
- Όχι

Υπόδειξη Ο `int` έχει πεπερασμένο εύρος τιμών. Τι συμβαίνει όταν το άθροισμα ξεπεράσει τη μέγιστη τιμή του;

K2.3 · Κύρια και δευτερεύουσα μνήμη

K2.3 · *Kahoot «Μεταβλητές και Μνήμη», «Μεταβλητές και Συναρτήσεις» (διάλεξη 2)* · ★☆☆ · multiple-choice · 84% σωστές · `kahoot-volatile-memory`

Μόλις έπεσε το ρεύμα, τα δεδομένα σου διατηρήθηκαν μόνο αν τα έσωσες στην

- Κύρια μνήμη
- Δευτερεύουσα μνήμη

Υπόδειξη Ποια από τις δύο χρειάζεται ρεύμα για να κρατήσει τα περιεχόμενά της;

Κ2.4 · Η μορφή μιας IP διεύθυνσης

Κ2.4 · *Kahoot «Μεταβλητές και Μνήμη», «Μεταβλητές και Συναρτήσεις» (διάλεξη 2)* · ★☆☆ · multiple-choice · 82% σωστές · `kahoot-ip-address-format`

Ποια από τα παρακάτω αντιστοιχεί σε ένα IP address;

- 1.2.3
- 1.1.1.1
- A.B.C.D.E
- hello world

Υπόδειξη Μια διεύθυνση IPv4 γράφεται ως τέσσερις δεκαδικοί αριθμοί από 0 έως 255, χωρισμένοι με τελείες.

Κ2.5 · Τιμές ενός byte

Κ2.5 · *Kahoot «Μεταβλητές και Μνήμη», «Μεταβλητές και Συναρτήσεις» (διάλεξη 2)* · ★☆☆ · multiple-choice · 78% σωστές · `kahoot-byte-values`

Πόσες διαφορετικές τιμές μπορεί να πάρει ένα 8-bit byte;

- 2^8
- 2^{16}
- 8
- 16

Υπόδειξη Κάθε bit έχει δύο δυνατές τιμές, ανεξάρτητα από τα υπόλοιπα.

Κ2.6 · Η ταχύτητα του Ariane 5

Κ2.6 · *Kahoot «Μεταβλητές και Μνήμη», «Μεταβλητές και Συναρτήσεις» (διάλεξη 2)* · ★☆☆ · multiple-choice · 70% σωστές · `kahoot-ariane-uint16`

Το Ariane 5 αναπαριστούσε την ταχύτητα με έναν `uint16_t`: ποια είναι η μεγαλύτερη τιμή της ταχύτητας που επιτρέπεται;

- 100
- 2^{16}
- 16000
- Όλες

Υπόδειξη Ένας `uint16_t` έχει 16 bits χωρίς πρόσημο· σκεφτείτε πόσες τιμές χωράνε (αυστηρά, η μεγαλύτερη είναι κατά ένα μικρότερη από το πλήθος τους, αφού ξεκινάμε από το 0).

K2.7 · Άθροισμα θετικών unsigned int

K2.7 · *Kahoot «Μεταβλητές και Μνήμη», «Μεταβλητές και Συναρτήσεις» (διάλεξη 2)* · ★★☆ · multiple-choice · 63% σωστές · `kahoot-unsigned-sum-sign`

Αν προσθέσω δύο θετικούς αριθμούς τύπου `unsigned int`, το αποτέλεσμα είναι πάντα θετικό;

- Ναι
- Όχι

Συχνή παρανόηση Το 30% απάντησε «Όχι», μεταφέροντας τη συμπεριφορά του `int` (όπου η υπερχείλιση δίνει αρνητικό) στους unsigned· ένας `unsigned int` αναδιπλώνεται modulo 2^{32} και δεν γίνεται ποτέ αρνητικός (αυστηρά, μπορεί να βγει 0 ή μικρότερος από τους προσθετέους).

Υπόδειξη Ποιες τιμές μπορεί να αναπαραστήσει καθόλου ένας `unsigned int`; Υπάρχει στο εύρος του κάποιος αρνητικός;

K2.8 · Πλήθος διευθύνσεων IPv4

K2.8 · *Kahoot «Μεταβλητές και Μνήμη», «Μεταβλητές και Συναρτήσεις» (διάλεξη 2)* · ★★☆ · multiple-choice · 57% σωστές · `kahoot-ip-address-count`

Με δεδομένο ότι μια IP address περιέχει 4 byte, πόσες IP διευθύνσεις μπορούν να υπάρξουν;

- 1.101.101.10
- 4 δις
- 2^{16}
- 256.000

Συχνή παρανόηση Το 25% απάντησε 2^{16}, μετρώντας 16 bits αντί για 32 (4 bytes × 8 bits).

Υπόδειξη Τα 4 bytes είναι συνολικά 32 bits. Πόσους διαφορετικούς συνδυασμούς δίνουν 32 bits, και πόσο είναι αυτό περίπου;

K2.9 · Πόσους αριθμούς χωράει ένας uint64_t

K2.9 · *Kahoot «Μεταβλητές και Μνήμη», «Μεταβλητές και Συναρτήσεις» (διάλεξη 2)* · ★★☆ · multiple-choice · 57% σωστές · `kahoot-uint64-range`

Πόσους διαφορετικούς αριθμούς μπορώ να αναπαραστήσω με έναν `uint64_t`;

- 2^{64}
- 4 δις
- Όλους τους ακεραίους
- Εξαρτάται

Υπόδειξη Το όνομα του τύπου από το `stdint.h` λέει ακριβώς πόσα bits έχει, σε αντίθεση με τον `int`.

K2.10 · Πόσος χώρος για ένα κομμάτι IP

K2.10 · *Kahoot «Μεταβλητές και Μνήμη», «Μεταβλητές και Συναρτήσεις» (διάλεξη 2)* · ★★☆ · multiple-choice · 45% σωστές · `kahoot-ip-octet-size`

Κάθε ένας από τους 4 αριθμούς μιας IP address είναι από το 0 μέχρι το 255. Για να αναπαρασταθεί κάθε ένας επαρκεί:

- Ένα bit
- Ένα byte
- Ένας 32-bit αριθμός
- Ένας 64-bit αριθμός

Υπόδειξη Πόσες διαφορετικές τιμές χρειάζονται, και πόσες χωράνε σε n bits; Βρείτε τη μικρότερη επιλογή που αρκεί.

K2.11 · Ο μετρητής του Gangnam Style

K2.11 · *Kahoot «Μεταβλητές και Μνήμη», «Μεταβλητές και Συναρτήσεις» (διάλεξη 2)* · ★★☆ · multiple-choice · 41% σωστές · `kahoot-gangnam-views`

Όταν το Gangnam Style στο YouTube κόντευε τα 2 δις views, η Google άλλαξε τον τύπο των views. Τι τύπου ήταν αρχικά;

- `char`
- `double`
- `int64_t`
- `int`

Συχνή παρανόηση Το 26% διάλεξε `int64_t`, που είναι ο τύπος στον οποίο μετέβη η Google *μετά*· ένας 64-bit ακέραιος δεν θα κόντευε ποτέ να υπερχειλίσει στα 2 δις.

Υπόδειξη Ποιος από τους τύπους έχει μέγιστη τιμή κοντά στα 2 δισεκατομμύρια; Υπολογίστε το 2^{31}.

K2.12 · Πόσο είναι το 2^64

K2.12 · *Kahoot «Μεταβλητές και Μνήμη», «Μεταβλητές και Συναρτήσεις» (διάλεξη 2)* · ★★★ · multiple-choice · 38% σωστές · `kahoot-two-pow-64`

Πόσο μεγάλος αριθμός είναι περίπου το 2^{64};

- 10^{10}
- 10^{20}
- 10^{30}
- 42

Συχνή παρανόηση Το 48% απάντησε 10^{30}, υπερεκτιμώντας κατά 10 τάξεις μεγέθους· με $2^{10} \approx 10^3$, το 2^{64} είναι περίπου $1.8 \cdot 10^{19}$.

Υπόδειξη Χρησιμοποιήστε την προσέγγιση $2^{10} = 1024 \approx 10^3$ και γράψτε το 2^{64} ως $2^4 \cdot (2^{10})^6$.

K2.13 · Το μέγεθος του int

K2.13 · *Kahoot «Μεταβλητές και Μνήμη», «Μεταβλητές και Συναρτήσεις» (διάλεξη 2)* · ★★★ · multiple-choice · 37% σωστές · `kahoot-int-size`

Με πόσα bytes αναπαρίσταται ο τύπος `int` στη μνήμη;

- 2
- 4
- 8
- Εξαρτάται

Συχνή παρανόηση Το 46% απάντησε 4, που είναι το συνηθισμένο μέγεθος στα σημερινά συστήματα, αλλά το πρότυπο της C δεν το ορίζει· σε άλλες αρχιτεκτονικές ο `int` μπορεί να είναι 2 bytes.

Υπόδειξη Τι εγγυάται το πρότυπο της C για το μέγεθος του `int`; Γι' αυτό υπάρχουν το `sizeof` και οι τύποι του `stdint.h`.

K2.14 · Ο κωδικός ASCII του 'J'

K2.14 · *Kahoot «Μεταβλητές και Μνήμη», «Μεταβλητές και Συναρτήσεις» (διάλεξη 2)* · ★★★ · multiple-choice · 33% σωστές · `kahoot-ascii-next-letter`

Εάν ο χαρακτήρας `'I'` έχει κωδικό ASCII `0x49`, ποιος είναι ο κωδικός του `'J'`;

- `0x50`
- `0x48`
- `0x4A`
- `0x39`

Συχνή παρανόηση Το 44% απάντησε `0x50`, κάνοντας την πρόσθεση σαν να ήταν δεκαδικό (49 + 1 = 50)· στο δεκαεξαδικό μετά το 9 έρχεται το A.

Υπόδειξη Τα γράμματα έχουν διαδοχικούς κωδικούς, άρα προσθέτετε 1. Προσοχή όμως: το `0x49` είναι δεκαεξαδικό, όχι δεκαδικό.

K2.15 · Το 0xFF σε signed char

K2.15 · *Kahoot «Μεταβλητές και Μνήμη», «Μεταβλητές και Συναρτήσεις» (διάλεξη 2)* · ★★★ · multiple-choice · 33% σωστές · `kahoot-signed-char-ff`

Για ένα `signed char` που έχει τιμή `0xFF` (συμπλήρωμα του δύο), ποια είναι η δεκαδική τιμή του;

- 1
- -1
- 256
- 0

Συχνή παρανόηση Το 41% απάντησε 256, που δεν χωράει καν σε 8 bits (το `0xFF` ως unsigned είναι 255)· σε `signed char` με συμπλήρωμα ως προς 2 όλοι οι άσοι σημαίνουν αρνητικό αριθμό.

Υπόδειξη Στο συμπλήρωμα ως προς 2 το πιο σημαντικό bit δείχνει το πρόσημο. Για να βρείτε το μέτρο ενός αρνητικού, κάντε flip όλα τα bits και προσθέστε 1.

Ζέσταμα: από τις διαφάνειες (A2.1–A2.8)

A2.1 · Πόσοι χαρακτήρες ASCII υπάρχουν

A2.1 · *Διάλεξη 2: Μνήμη και Μεταβλητές, διαφάνεια 17* · ★☆☆ · short-answer · `slides-lec02-ascii-count`

Τα 8 bits ενός `char` μπορεί να ερμηνευθούν ως χαρακτήρας κειμένου με την αντιστοίχιση του πίνακα ASCII (`man ascii`). Π.χ., ο αριθμός $67_{(10)} = 43_{(16)}$ αντιστοιχεί στο γράμμα `'C'`.

Πόσους διαφορετικούς χαρακτήρες έχουμε στο σύστημα ASCII; Γιατί;

Υπόδειξη Δείτε ποιος είναι ο μεγαλύτερος κωδικός στον πίνακα της `man ascii` (και στις τρεις βάσεις). Πόσα bits χρειάζονται για να γραφτεί;

A2.2 · Πόσα διαφορετικά bytes υπάρχουν

A2.2 · *Διάλεξη 2: Μνήμη και Μεταβλητές, διαφάνεια 5* · ★☆☆ · short-answer · `slides-lec02-byte-values`

Ένα byte είναι μια ομάδα από 8 bits (octet), όπως το `00101010`.

Πόσα διαφορετικά 8-bit bytes μπορούμε να έχουμε;

Υπόδειξη Πόσες επιλογές έχει το πρώτο bit; Και για καθεμία από αυτές, πόσες το δεύτερο; Μετρήστε πρώτα για 1, 2 και 3 bits και γενικεύστε.

A2.3 · Δεκαεξαδικά ψηφία ενός byte

A2.3 · *Διάλεξη 2: Μνήμη και Μεταβλητές, διαφάνεια 10* · ★☆☆ · short-answer · `slides-lec02-hex-digits`

Ένας δεκαεξαδικός αριθμός εκφράζεται με μια ακολουθία από 16 σύμβολα: `0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F`. Για παράδειγμα, $2A_{(16)} = 2 \cdot 16^1 + A \cdot 16^0 = 42_{(10)}$.

Πόσα ψηφία χρειαζόμαστε στο δεκαεξαδικό σύστημα για να εκφράσουμε ένα byte; Είναι το ίδιο με το δεκαδικό;

Υπόδειξη Ποια είναι η μεγαλύτερη τιμή ενός byte, και πώς γράφεται σε κάθε βάση; Σκεφτείτε επίσης σε πόσα bits αντιστοιχεί ένα δεκαεξαδικό ψηφίο, αφού $16 = 2^4$.

A2.4 · Πόσο μεγάλος είναι ένας int

A2.4 · *Διάλεξη 2: Μνήμη και Μεταβλητές, διαφάνεια 27* · ★☆☆ · short-answer · `slides-lec02-int-size`

Είπαμε ότι το μέγεθος ενός `int` είναι *συνήθως* 4 bytes. Υπάρχει τρόπος να είμαστε βέβαιοι;

Υπόδειξη Υπάρχουν δύο τρόποι: ένας τελεστής της C που δίνει το μέγεθος ενός τύπου σε bytes, και μια βιβλιοθήκη που ορίζει ακέραιους τύπους με το πλήθος των bits στο όνομά τους.

A2.5 · Ακέραιοι με 32 bit

A2.5 · *Διάλεξη 2: Μνήμη και Μεταβλητές, διαφάνεια 18* · ★☆☆ · short-answer · `slides-lec02-int32-count`

Ο τύπος `int` που αναπαριστά ακεραίους απαιτεί *συνήθως* 4 bytes από τον μεταγλωττιστή. Για παράδειγμα, το 42 αποθηκεύεται ως

`00000000000000000000000000101010`

1. Πόσους διαφορετικούς ακεραίους μπορούμε να αναπαραστήσουμε με 32 bit;
2. Υπάρχει κάποιο πιθανό πρόβλημα με αυτήν την αναπαράσταση;

Υπόδειξη Για το (1) εφαρμόστε τον ίδιο κανόνα με το πλήθος των διαφορετικών bytes. Για το (2) σκεφτείτε δύο πράγματα: είναι αυτό το πλήθος αρκετό για κάθε πρόγραμμα, και πού «χωράει» το πρόσημο ενός αρνητικού αριθμού;

A2.6 · Η εντολή για να μάθουμε για ένα πρόγραμμα

A2.6 · *Διάλεξη 2: Μνήμη και Μεταβλητές, διαφάνεια 16* · ★☆☆ · tooling · `slides-lec02-man`

Ποια εντολή του Linux μας επιτρέπει να μάθουμε για ένα πρόγραμμα στο σύστημά μας;

Χρησιμοποιήστε την για να βρείτε τον κωδικό ASCII του γράμματος `C` σε δεκαδικό και σε δεκαεξαδικό.

Υπόδειξη Η εντολή εμφανίζει το εγχειρίδιο (manual) του συστήματος, και έχει σελίδες όχι μόνο για προγράμματα αλλά και για θέματα όπως ο πίνακας ASCII.

A2.7 · Αρνητικοί αριθμοί σε συμπλήρωμα ως προς 2

A2.7 · *Διάλεξη 2: Μνήμη και Μεταβλητές, διαφάνεια 21* · ★☆☆ · short-answer · `slides-lec02-twos-complement`

Υπάρχουν θετικοί και αρνητικοί ακέραιοι. Πώς αναπαριστούμε το γεγονός ότι ένας αριθμός είναι αρνητικός στη δυαδική αναπαράσταση; Τι κάνουμε με το πρόσημο;

Στο συμπλήρωμα ως προς 2 (two's complement) το πρόσημο φαίνεται από το σημαντικότερο bit του αριθμού ($0 \to +$, $1 \to -$). Για να πάρουμε την αναπαράσταση του $-N$ από έναν θετικό δυαδικό αριθμό N:

1. Αντιστρέφουμε (flip) όλα τα bit του N.
2. Προσθέτουμε 1.

Έστω $N = 11_{(10)} = 00001011_{(2)}$. Ποια είναι η αναπαράσταση του $-N$ σε 8 bits; Επαληθεύστε ότι $N + (-N) = 0$.

Υπόδειξη Κάντε τα δύο βήματα ένα-ένα, προσέχοντας τα κρατούμενα στην πρόσθεση. Στην επαλήθευση, το άθροισμα βγαίνει 9 bits· τι γίνεται με το bit που δεν χωράει στα 8;

A2.8 · Υπερχείλιση ακεραίων

A2.8 · *Διάλεξη 2: Μνήμη και Μεταβλητές, διαφάνεια 26* · ★★☆ · debug · `slides-lec02-overflow`

Τι θα τυπώσει το παρακάτω πρόγραμμα;

```c
#include <stdio.h>

int main() {
  printf("%d\n", 2000000000 + 2000000000);
  return 0;
}
```

```
$ ./overflow
-294967296
```

Τι συνέβη; Πώς το διορθώνουμε;

Υπόδειξη Ποιος είναι ο τύπος των δύο σταθερών, και ποια η μεγαλύτερη τιμή που χωράει σε αυτόν; Συγκρίνετε το $4.000.000.000$ με το 2^{32}. Για τη διόρθωση, χρειάζεστε έναν τύπο με περισσότερα bits, τόσο για τις σταθερές όσο και στο προσδιοριστικό της `printf`.

Εργαστήριο (A2.9–A2.11)

A2.9 · Στοιχεία φοιτητή (Παλιό θέμα)

A2.9 · *Εργαστήριο 0, Άσκηση 1* · ★☆☆ · programming · `lab-lab00-about`

Γράψτε ένα πρόγραμμα `about` το οποίο τυπώνει 3 γραμμές στην πρότυπη έξοδο (stdout):

1) το όνομά σας με λατινικούς χαρακτήρες στην 1η, 2) το sdi σας στην 2η και 3) `I'm 100% "ready"!` στην 3η. Παράδειγμα εξόδου από την εκτέλεση του προγράμματος `about`:

```
$ ./about
Michael Jordan
sdi2300999
I'm 100% "ready"!
$
```

Υπόδειξη Το δύσκολο σημείο είναι η τρίτη γραμμή: τα διπλά εισαγωγικά κλείνουν τη συμβολοσειρά της `printf` και το `%` ξεκινά προδιαγραφή μορφοποίησης. Σκεφτείτε πώς «αποδρά» κανείς έναν χαρακτήρα μέσα σε συμβολοσειρά και πώς τυπώνεται ένα σκέτο `%`. Μην ξεχάσετε την αλλαγή γραμμής στο τέλος κάθε γραμμής.

A2.10 · Υπερχείλιση ακεραίων

A2.10 · *Εργαστήριο 0, Άσκηση 2* · ★☆☆ · short-answer · `lab-lab00-overflow`

Μεταγλωττίστε και τρέξτε το ακόλουθο πρόγραμμα:

```c
#include <stdio.h>

int main() {
  printf("%d\n", 2000000000 + 2000000000);
  return 0;
}
```

Συμπεριφέρεται όπως θα περιμένατε; Γιατί / γιατί όχι; Διορθώστε το αν μπορείτε.

2.1 Ειδικές ακολουθίες χαρακτήρων (Προχωρημένο, Προαιρετικό)

Παραπάνω χρησιμοποιήσαμε την ειδική ακολουθία `\n` για να τυπώσουμε μια νέα γραμμή μετά το μήνυμά μας. Αντίστοιχα μπορούμε να χρησιμοποιήσουμε άλλες ακολουθίες προκειμένου να τυπώσουμε ειδικούς χαρακτήρες ή να δώσουμε συγκεκριμένη μορφοποίηση στο κείμενό μας.

Παράδειγμα 1. Δοκιμάστε να τυπώσετε την ειδική ακολουθία `\U0001F525`. Τι τυπώνεται στην κονσόλα; Μπορείτε να βρείτε μια εκτενή λίστα με άλλες ειδικές ακολουθίες εδώ ενώ για να μάθετε περισσότερα μπορείτε να διαβάσετε για Unicode χαρακτήρες στην wikipedia.

Παράδειγμα 2. Χρησιμοποιήστε την `printf` για να τυπώσετε την ακολουθία `"Hello \033[1m world \033[0m !"`. Τι παρατηρείτε; Μπορείτε να φανταστείτε τι σημαίνουν οι ακολουθίες `\033[1m` και `\033[0m`; Τι συμβαίνει αν αντικαταστήσετε την ακολουθία `\033[1m` με την ακολουθία `\033[5m`; Μπορείτε να μάθετε περισσότερα διαβάζοντας για ANSI Escape codes.

Υπόδειξη Ποιος είναι ο τύπος των δύο σταθερών και ποια είναι η μεγαλύτερη τιμή που χωράει σε έναν `int` των 4 bytes; Διαβάστε και τις προειδοποιήσεις του `gcc`. Για τη διόρθωση, σκεφτείτε έναν ευρύτερο ακέραιο τύπο και ποια προδιαγραφή της `printf` του αντιστοιχεί.

A2.11 · Εκτύπωση χαρακτήρων

A2.11 · *Εργαστήριο 4, Άσκηση 1* · ★☆☆ · programming · `lab-lab04-printchar`

Γράψτε ένα πρόγραμμα `printchar.c` που να εκτυπώνει τους χαρακτήρες με ASCII κωδικό που είναι πολλαπλάσιο του 3, και μεταξύ 33 και 105, μαζί με τους κωδικούς τους σε δεκαδική και δεκαεξαδική μορφή. Το πρόγραμμά σας θέλουμε να τυπώνει σε κάθε γραμμή έναν χαρακτήρα, ακολουθούμενο από τον δεκαδικό και τον δεκαεξαδικό κωδικό του. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ ./printchar
! - (dec: 33, hex: 21)
$ - (dec: 36, hex: 24)
' - (dec: 39, hex: 27)
* - (dec: 42, hex: 2a)
- - (dec: 45, hex: 2d)
0 - (dec: 48, hex: 30)
3 - (dec: 51, hex: 33)
...
```

Παρατηρήστε την έξοδο του προγράμματος σε σχέση με τον πίνακα των ASCII κωδικών (δείτε τον πλήρη πίνακα στο εργαστήριο ή με την εντολή `man ascii`).

Υπόδειξη Ένας χαρακτήρας στη C είναι απλώς ένας μικρός ακέραιος: η ίδια τιμή τυπώνεται ως χαρακτήρας, ως δεκαδικός ή ως δεκαεξαδικός ανάλογα με την προδιαγραφή της `printf` (`%c`, `%d`, `%x`). Ο βρόχος μπορεί να ξεκινά από το πρώτο πολλαπλάσιο του 3 στο διάστημα και να προχωρά με βήμα 3.

Θέματα εξετάσεων (A2.12)

A2.12 · Rickroll Τριπλέτες

A2.12 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #4 (Rick Astley Themed), Θέμα 3* · ★★★ ·

programming · `exam-2023-fall-ex4-q3`

Πρόγραμμα: `roll.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που παίρνει 3 αριθμούς ως ορίσματα (σε αναπαράσταση μη προσημασμένου 32-bit αριθμού) και ελέγχει αν είναι είναι μια rickroll τριπλέτα. Τρεις αριθμοί a, b και c λέγονται rickroll τριπλέτα εάν υπάρχουν ακέραιοι I, J, K (διαφορετικοί ο ένας από τον άλλον I <> J <> K) έτσι ώστε 2^I * a + 2^J * b = 2^K * c - όπου όλες οι πράξεις είναι μη προσημασμένες mod 2^32. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o roll roll.c
$ ./roll 17 42 101
Found rick-roll triplet: 2^30 * 17 + 2^25 * 42 = 2^26 * 101 = 2483027968
Found rick-roll triplet: 2^30 * 17 + 2^29 * 42 = 2^31 * 101 = 2147483648
Found rick-roll triplet: 2^31 * 17 + 2^26 * 42 = 2^27 * 101 = 671088640
$ ./roll 17 42 100
Found rick-roll triplet: 2^27 * 17 + 2^29 * 42 = 2^25 * 100 = 3355443200
Found rick-roll triplet: 2^28 * 17 + 2^25 * 42 = 2^24 * 100 = 1677721600
Found rick-roll triplet: 2^28 * 17 + 2^30 * 42 = 2^26 * 100 = 2415919104
Found rick-roll triplet: 2^29 * 17 + 2^26 * 42 = 2^25 * 100 = 3355443200
Found rick-roll triplet: 2^29 * 17 + 2^31 * 42 = 2^27 * 100 = 536870912
Found rick-roll triplet: 2^30 * 17 + 2^27 * 42 = 2^26 * 100 = 2415919104
Found rick-roll triplet: 2^30 * 17 + 2^31 * 42 = 2^28 * 100 = 1073741824
Found rick-roll triplet: 2^31 * 17 + 2^28 * 42 = 2^27 * 100 = 536870912
$ ./roll 3 5 7
No rick-roll triplets found.
$ ./roll 0 0 1
No rick-roll triplets found.
```

Υπόδειξη Πολλαπλασιασμός με 2^I είναι αριστερή ολίσθηση κατά I, και σε μη προσημασμένο 32-bit τύπο (`uint32_t`) η αναδίπλωση mod 2^32 γίνεται αυτόματα, οπότε μια εξαντλητική αναζήτηση στα I, J, K είναι αρκετά μικρή. Σκεφτείτε ποιο είναι το λογικό εύρος των εκθετών (τι παθαίνει μια ολίσθηση κατά 32 ή περισσότερα bits;) και ποια σειρά βρόχων δίνει τη σειρά εξόδου του παραδείγματος. Διαβάστε τα ορίσματα με `strtoul` και απορρίψτε ό,τι δεν χωράει σε 32 bits.

Κεφάλαιο 3: Συναρτήσεις

Kahoot από το αμφιθέατρο (K3.1–K3.8)

K3.1 · Προσδιοριστικά %d με σειρά

K3.1 · *Kahoot «Τύπωμα και Συναρτήσεις» (διάλεξη 3)* · ★☆☆ · multiple-choice · 87% σωστές ·

`kahoot-printf-format-order`

Τι θα τυπώσει η `printf("England-Greece %d-%d", 1, 2);`

- England-Greece 1-2
- England-Greece -1
- England-Greece %d-%d
- England-Greece 2-1

Υπόδειξη Κάθε `%d` αντικαθίσταται με το επόμενο όρισμα, με τη σειρά που δίνονται.

Κ3.2 · Τιμή επιστροφής double

Κ3.2 · *Kahoot «Τύπωμα και Συναρτήσεις» (διάλεξη 3)* · ★★☆ · multiple-choice · 58% σωστές · `kahoot-square-double-return`

Ποια η τιμή που θα επιστρέψει η `square(16.0)`, εφόσον:

```c
double square(double x) { return x * x; }
```

- 0
- 256
- 256.0
- 42

Υπόδειξη Ο τύπος της τιμής που επιστρέφει μια συνάρτηση ορίζεται στην αρχή του ορισμού της.

Κ3.3 · Από τι αποτελείται ένα πρόγραμμα C

Κ3.3 · *Kahoot «Τύπωμα και Συναρτήσεις» (διάλεξη 3)* · ★★☆ · multiple-choice · 55% σωστές · `kahoot-c-program-functions`

Ένα πρόγραμμα C είναι συνήθως:

- Μια σειρά από συναρτήσεις
- Η συνάρτηση main
- Μια σειρά από τελεστές
- Μια σειρά από εντολές

Συχνή παρανόηση Το 30% επέλεξε «μια σειρά από εντολές»· οι εντολές όμως υπάρχουν μόνο μέσα στο σώμα συναρτήσεων, και το πρόγραμμα οργανώνεται σε συναρτήσεις, με πρώτη τη `main`.

Υπόδειξη Οι εντολές δεν μπορούν να βρίσκονται «ελεύθερες» σε ένα αρχείο C· σκεφτείτε μέσα σε τι γράφονται πάντα.

K3.4 · Η ακολουθία `\r`

K3.4 · *Kahoot «Τύπωμα και Συναρτήσεις» (διάλεξη 3)* · ★★☆ · multiple-choice · 48% σωστές · `kahoot-printf-carriage-return`

Τι θα δω στην οθόνη αν τρέξω `printf("hello\rworld");`

- hello
- world
- hello world
- hellorworld

Υπόδειξη Το `\r` (carriage return) δεν αλλάζει γραμμή· σκεφτείτε πού μετακινεί τον κέρσορα και τι γίνεται με όσα γράφονται μετά.

K3.5 · printf με %c και 42

K3.5 · *Kahoot «Τύπωμα και Συναρτήσεις» (διάλεξη 3)* · ★★☆ · multiple-choice · 48% σωστές · `kahoot-printf-char-42`

Η εκτέλεση της `printf("%c", 42);` θα τυπώσει:

- %c
- *
- 42
- 0x42

Συχνή παρανόηση Το 24% επέλεξε `42`, σαν το `%c` να τυπώνει τον αριθμό· το `%c` τυπώνει τον χαρακτήρα με αυτόν τον κωδικό ASCII.

Υπόδειξη Το `%c` ερμηνεύει τον ακέραιο ως κωδικό χαρακτήρα· ψάξτε ποιος χαρακτήρας έχει κωδικό ASCII 42 (`man ascii`).

K3.6 · Ακολουθίες διαφυγής για `\\` και `"`

K3.6 · *Kahoot «Τύπωμα και Συναρτήσεις» (διάλεξη 3)* · ★★☆ · multiple-choice · 43% σωστές · `kahoot-printf-escapes-backslash-quote`

Πώς τυπώνουμε την ακολουθία: `I'm \\ "back"`

- `printf("I'm \\\\ \"back\"");`
- `printf("I\'m \\ \"back\"");`
- `printf("I'm \\ "back"");`
- `printf("I am \n %dback%d");`

Υπόδειξη Μέσα σε ένα αλφαριθμητικό, κάθε `\` που θέλουμε να τυπωθεί και κάθε `"` πρέπει να γραφτεί με ακολουθία διαφυγής. Μετρήστε πόσα `\` θέλουμε στην έξοδο.

Κ3.7 · Τα μέρη του ορισμού συνάρτησης

Κ3.7 · *Kahoot «Τύπωμα και Συναρτήσεις» (διάλεξη 3)* · ★★★ · multiple-choice · 31% σωστές · `kahoot-function-definition-parts`

Ποιο από τα παρακάτω είναι μέρος του ορισμού μιας συνάρτησης σε C; (Επιλέξτε όλες τις σωστές απαντήσεις.)

- Ο τύπος επιστροφής
- Το σώμα της συνάρτησης
- Τα ορίσματα
- Η συνολοθεωρία Cantor

Υπόδειξη Γράψτε νοερά τον ορισμό μιας απλής συνάρτησης, π.χ. της `square`, και ονομάστε κάθε κομμάτι του από αριστερά προς τα δεξιά.

Κ3.8 · Ο τύπος επιστροφής μιας συνάρτησης

Κ3.8 · *Kahoot «Τύπωμα και Συναρτήσεις» (διάλεξη 3)* · ★★★ · multiple-choice · 28% σωστές · `kahoot-function-return-type`

Ποιος ο τύπος επιστροφής της συνάρτησης:

```c
double square(int x) { return x * x; }
```

- square
- double
- int
- x

Συχνή παρανόηση Το 25% επέλεξε `int`, μπερδεύοντας τον τύπο της παραμέτρου `x` με τον τύπο επιστροφής· η τιμή του `x * x` μετατρέπεται σε `double` κατά την επιστροφή.

Υπόδειξη Ο τύπος επιστροφής γράφεται πριν από το όνομα της συνάρτησης· ό,τι βρίσκεται μέσα στις παρενθέσεις αφορά τις παραμέτρους.

Ζέσταμα: από τις διαφάνειες (A3.1–A3.6)

A3.1 · Τιμή μετά από κλήση συνάρτησης

A3.1 · *Διάλεξη 3: Συναρτήσεις, διαφάνεια 37* · ★☆☆ · trace · `slides-lec03-function-call`

Ποια η τιμή του `x` μετά την εκτέλεση της ανάθεσης στη `main`;

```
int g(int x, int y, int z) {
  return x * x + y * y + z * z + 42;
}

int main(int argc, char **argv) {
  ....
  int x = g(1, 2, 3);
  ....
}
```

Υπόδειξη Κατά την κλήση, οι παράμετροι `x`, `y`, `z` της `g` παίρνουν με τη σειρά τις τιμές των ορισμάτων. Αντικαταστήστε τις στην παράσταση του `return`.

A3.2 · Πόσα bytes είναι ένας `int`;

A3.2 · *Διάλεξη 3: Συναρτήσεις, διαφάνεια 9* · ★☆☆ · short-answer · `slides-lec03-int-size`

Είπαμε ότι το μέγεθος ενός `int` είναι συνήθως 4 bytes. Υπάρχει τρόπος να είμαστε βέβαιοι;

Υπόδειξη Υπάρχουν δύο δρόμοι: ένας τελεστής της C που μετράει το μέγεθος ενός τύπου σε bytes, και μια βιβλιοθήκη που ορίζει ακέραιους τύπους με το πλήθος των bits στο όνομά τους.

A3.3 · Διακοπή ρεύματος και μνήμη

A3.3 · *Διάλεξη 3: Συναρτήσεις, διαφάνεια 3* · ★☆☆ · multiple-choice · `slides-lec03-power-outage`

(Ερώτηση επανάληψης Kahoot από την προηγούμενη διάλεξη.)

Μόλις έπεσε το ρεύμα, τα δεδομένα σου διατηρήθηκαν μόνο αν τα έσωσες στην:

- Α. Κύρια μνήμη

- Β. Δευτερεύουσα μνήμη

Υπόδειξη Σκεφτείτε ποια από τις δύο μνήμες είναι πτητική, δηλαδή χάνει το περιεχόμενό της όταν σταματήσει η τροφοδοσία, και σε ποια ανήκει ένας σκληρός δίσκος.

Α3.4 · Πυθαγόρειο θεώρημα (pyth.c)

A3.4 · *Διάλεξη 3: Συναρτήσεις, διαφάνεια 40* · ★☆☆ · programming · `slides-lec03-pyth`

Challenge #1: Χρειαζόμαστε ένα πρόγραμμα `pyth.c` το οποίο **με δεδομένα τα μήκη των καθέτων πλευρών ενός ορθογωνίου τριγώνου** να τυπώνει τα ακόλουθα:

1. Το εμβαδόν του τριγώνου.
2. Την περίμετρο του τριγώνου.

Υπόδειξη Γράψτε μία συνάρτηση για κάθε υπολογισμό, με δύο ορίσματα `double`. Για την περίμετρο χρειάζεστε πρώτα την υποτείνουσα από το Πυθαγόρειο θεώρημα, με την `sqrt` του `math.h`· θυμηθείτε τι χρειάζεται στη γραμμή του `gcc` για να συνδεθεί η μαθηματική βιβλιοθήκη.

Α3.5 · Προσέγγιση του π με τη σειρά Leibniz

A3.5 · *Διάλεξη 3: Συναρτήσεις, διαφάνεια 43* · ★★☆ · programming · `slides-lec03-leibniz-pi`

Challenge #2: Ο Leibniz έδειξε ότι μπορούμε να προσεγγίσουμε την τιμή του π με την ακόλουθη φόρμουλα:

$$\frac{\pi}{4} = \sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots$$

Θέλουμε να γράψουμε ένα πρόγραμμα που να προσεγγίζει το π. Μπορούμε;

Υπόδειξη Αθροίστε ένα μεγάλο αλλά πεπερασμένο πλήθος όρων (π.χ. 100000) σε μεταβλητή `double`. Δεν χρειάζεται να υψώνετε το -1 σε δύναμη: κρατήστε τον αριθμητή και τον παρονομαστή του τρέχοντος όρου σε μεταβλητές και ενημερώνετε τις σε κάθε επανάληψη. Μην ξεχάσετε τον πολλαπλασιασμό με 4 στο τέλος.

A3.6 · Υπερχείλιση ακεραίων

A3.6 · *Διάλεξη 3: Συναρτήσεις, διαφάνεια 8* · ★★☆ · trace · `slides-lec03-overflow`

Τι θα τυπώσει το παρακάτω πρόγραμμα;

```c
#include <stdio.h>

int main() {
  printf("%d\n", 2000000000 + 2000000000);
  return 0;
}
```

Όταν το τρέχουμε, τυπώνει:

```
$ ./overflow
-294967296
```

Τι συνέβη; Πώς το διορθώνουμε;

Υπόδειξη Ποιος είναι ο τύπος των δύο σταθερών, άρα και του αθροίσματος, και ποια είναι η μέγιστη τιμή του; Συγκρίνετε το σωστό άθροισμα με το 2^{32}. Για τη διόρθωση, σκεφτείτε έναν τύπο με μεγαλύτερο εύρος, το suffix των σταθερών και το προσδιοριστικό της `printf` που του αντιστοιχεί.

Εργαστήριο (A3.7)

A3.7 · Πυθαγόρειο θεώρημα

A3.7 · *Εργαστήριο 2, Άσκηση 2* · ★☆☆ · programming · `lab-lab02-pyth`

Γράψτε ένα πρόγραμμα `pyth.c` που δέχεται 2 inputs - τα μήκη των καθέτων πλευρών ενός ορθογωνίου τριγώνου και τυπώνει τα ακόλουθα:

1. Το εμβαδόν του τριγώνου.
2. Την περίμετρο του τριγώνου.

Υπόδειξη Η υποτείνουσα προκύπτει από το πυθαγόρειο θεώρημα, οπότε θα χρειαστείτε τετραγωνική ρίζα από τη `math.h` (και `-lm` στη μεταγλώττιση). Διαβάστε τα μήκη ως αριθμούς κινητής υποδιαστολής και προσέξτε την ακέραια διαίρεση στο μισό του γινομένου των καθέτων.

Θέματα εξετάσεων (A3.8)

A3.8 · Η συνάρτηση about

A3.8 · *Εξέταση Σεπτεμβρίου 2024, Θέμα 1* · ★☆☆ · programming · `exam-2024-sep-q1`

About [5 Μονάδες]

Γράψτε μια συνάρτηση `about` η οποία τυπώνει 3 γραμμές στην πρότυπη έξοδο (stdout):

1) το όνομά σας με λατινικούς χαρακτήρες στην 1η, 2) το sdi σας στην 2η και 3) `I'm 100% "ready"!` στην 3η. Παράδειγμα εξόδου από την εκτέλεση της συνάρτησης `about`:

```
Michael Jordan
sdi2300999
I'm 100% "ready"!
```

Υπόδειξη Η παγίδα είναι η τρίτη γραμμή: σκεφτείτε ποιοι χαρακτήρες έχουν ειδική σημασία μέσα σε ένα string literal της C και ποιοι μέσα στο format string της `printf`. Τα διπλά εισαγωγικά και το `%` χρειάζονται ειδικό χειρισμό, το μονό εισαγωγικό όχι.

Κεφάλαιο 4: Git και Τελεστές

Kahoot από το αμφιθέατρο (K4.1–K4.9)

K4.1 · Ξέχασα κάτι στο git;

K4.1 · *Kahoot «Ροή Ελέγχου #2» (διάλεξη 8)* · ★☆☆ · multiple-choice · 95% σωστές · `kahoot-git-forgot-commit`

Μόλις έγραψα το `trolley.c` και έκανα `git add` και μετά `git push`. Ξέχασα κάτι;

- git branch
- git out
- git commit
- Όχι, όλα καλά (τι είναι το git?)

Υπόδειξη Το `git push` στέλνει στο GitHub ό,τι έχει καταγραφεί στο τοπικό ιστορικό· ποιο βήμα δημιουργεί αυτή την καταγραφή;

K4.2 · Git και GitHub

K4.2 · *Kahoot «Git και Τελεστές» (διάλεξη 4)* · ★☆☆ · multiple-choice · 82% σωστές · `kahoot-git-vs-github`

Το git και το GitHub είναι το ίδιο πράγμα.

- True
- False

Υπόδειξη Το ένα είναι πρόγραμμα που τρέχει στον υπολογιστή σας, το άλλο μια υπηρεσία στο διαδίκτυο. Μπορεί να υπάρχει το ένα χωρίς το άλλο;

K4.3 · 0xFF | 0x42

K4.3 · *Kahoot «Git και Τελεστές» (διάλεξη 4)* · ★★☆ · multiple-choice · 57% σωστές · `kahoot-or-mask-ff`

Ποια η τιμή της παράστασης `0xFF | 0x42`;

- `0xFF`
- `42`
- `0x42`
- `1`

Υπόδειξη Γράψτε και τους δύο αριθμούς στο δυαδικό. Αν ο ένας τελεστέος έχει ήδη 1 σε κάθε θέση, τι μπορεί να αλλάξει το bitwise OR;

K4.4 · 3 « 2

K4.4 · *Kahoot «Git και Τελεστές» (διάλεξη 4)* · ★★☆ · multiple-choice · 55% σωστές · `kahoot-shift-left`

Ποια η τιμή της παράστασης `3 << 2`;

- `3`
- `6`
- `12`
- `42`

Υπόδειξη Γράψτε το 3 στο δυαδικό και μετακινήστε όλα τα bits δύο θέσεις αριστερά· κάθε μετατόπιση κατά μία θέση αντιστοιχεί σε έναν πολλαπλασιασμό.

Κ4.5 · Ποιες είναι εντολές git;

Κ4.5 · *Kahoot «Git και Τελεστές» (διάλεξη 4)* · ★★★ · multiple-choice · 34% σωστές · `kahoot-git-commands`

Ποιες από τις ακόλουθες είναι εντολές git; (Μπορεί να είναι σωστές περισσότερες από μία.)

- `git push`
- `git status`
- `git gud`
- `git commit`

Συχνή παρανόηση Το 35% διάλεξε μόνο `git push` και `git commit`, ξεχνώντας το `git status`, που δεν αλλάζει τίποτα αλλά δείχνει ποια αρχεία έχουν αλλάξει ή είναι staged.

Υπόδειξη Σκεφτείτε όχι μόνο τις εντολές που αλλάζουν το αποθετήριο, αλλά και εκείνη που τρέχουμε συνεχώς για να δούμε σε ποια κατάσταση βρίσκονται τα αρχεία μας.

Κ4.6 · Η σειρά των εντολών git

Κ4.6 · *Kahoot «Git και Τελεστές» (διάλεξη 4)* · ★★★ · short-answer · 34% σωστές · `kahoot-git-order`

Βάλτε τις ακόλουθες εντολές σε χρονολογική σειρά:

- `git push`
- `git commit`
- `git clone`
- `git add`

Υπόδειξη Ακολουθήστε τον κύκλο του git: πρώτα αποκτάτε αντίγραφο του αποθετηρίου, μετά ετοιμάζετε (stage) τις αλλαγές, τις καταγράφετε τοπικά και στο τέλος τις στέλνετε στο GitHub.

Κ4.7 · 0xbeef | 0xcafe0000

Κ4.7 · *Kahoot «Git και Τελεστές» (διάλεξη 4)* · ★★★ · multiple-choice · 29% σωστές · `kahoot-or-combine`

Ποια η τιμή της παράστασης `0xbeef | 0xcafe0000`;

- `0xffffffff`
- `0xcafebeef`

- `0xbeefcafe`
- `0x00000000`

Συχνή παρανόηση Το 29% απάντησε `0xbeefcafe`, διαβάζοντας το OR σαν «παράθεση με τη σειρά που γράφονται»· όμως το OR δεν μετακινεί bits, το καθένα μένει στη θέση του, και το `0xcafe` βρίσκεται στα πάνω 16 bits.

Υπόδειξη Κάθε δεκαεξαδικό ψηφίο είναι 4 bits. Τα δύο νούμερα έχουν μη μηδενικά ψηφία σε διαφορετικές θέσεις· τι κάνει το OR όταν σε κάθε θέση ο ένας από τους δύο είναι 0;

K4.8 · 0x42 & 0xFF

K4.8 · *Kahoot «Git και Τελεστές» (διάλεξη 4)* · ★★★ · multiple-choice · 18% σωστές · `kahoot-and-mask-ff`

Ποια η τιμή της παράστασης `0x42 & 0xFF`;

- `0xFF`
- `42`
- `0x42`
- `0`

Συχνή παρανόηση Το 30% απάντησε `0`, πιθανότατα επειδή μπέρδεψε το bitwise `&` με το λογικό `&&` ή υπέθεσε ότι το AND «σβήνει» τα bits· στην πραγματικότητα το AND με `0xFF` αφήνει το byte ανέπαφο.

Υπόδειξη Το `0xFF` είναι οκτώ άσοι στο δυαδικό. Τι αφήνει το bitwise AND με 1 σε κάθε bit του άλλου τελεστέου;

K4.9 · Απομόνωση του πιο σημαντικού bit

K4.9 · *Kahoot «Git και Τελεστές» (διάλεξη 4)* · ★★★ · multiple-choice · 14% σωστές · `kahoot-msb-mask`

Google backend engineer interview question: πώς απομονώνουμε το πιο σημαντικό bit (το 8ο) σε ένα byte;

- `byte & 0xFF`
- `byte & 0x80`
- `byte | 0xFF`
- `byte | 0x80`

Συχνή παρανόηση Το 29% διάλεξε `byte & 0xFF`: η μάσκα `0xFF` έχει και τα 8 bits στο 1, άρα κρατάει ολόκληρο το byte αντί να απομονώνει μόνο το πιο σημαντικό.

Υπόδειξη Γράψτε κάθε μάσκα στο δυαδικό. Για να κρατήσουμε ένα bit και να μηδενίσουμε τα υπόλοιπα, χρειαζόμαστε τον τελεστή που δίνει 1 μόνο όπου *και τα δύο* bits είναι 1, και μια μάσκα με ένα μόνο 1.

Ζέσταμα: από τις διαφάνειες (A4.1–A4.10)

A4.1 · Η τιμή του 0xbeef | 0xcafe0000

A4.1 · *Διάλεξη 4: Git και Τελεστές, διαφάνεια 30* · ★☆☆ · multiple-choice · `slides-lec04-cafebeef`

Ποια η τιμή της παράστασης `0xbeef | 0xcafe0000`;

- A. `0xffffffff`
- B. `0xcafebeef`
- C. `0xbeefcafe`
- D. `0xfaceb00c`

Υπόδειξη Γράψτε και τους δύο αριθμούς με 8 δεκαεξαδικά ψηφία (συμπληρώνοντας μηδενικά αριστερά) τον έναν κάτω από τον άλλο. Κάθε δεκαεξαδικό ψηφίο είναι 4 bit, και `0 | x` είναι `x`.

A4.2 · Ο κύκλος clone, add, commit, push, pull

A4.2 · *Διάλεξη 4: Git και Τελεστές, διαφάνειες 15-18* · ★☆☆ · tooling · `slides-lec04-git-cycle`

Στο repository μιας εργασίας σας (π.χ. `git@github.com:progintro/hw0-barbouni-2005.git`):

1. Κάντε `git clone` το repository. Ελέγξτε τον φάκελο: τι αρχείο έχει; Τρέξτε `git status` μέσα στον φάκελο: υπάρχουν αλλαγές;
2. Δημιουργήστε ένα καινούργιο αρχείο, π.χ. `touch command/solution.txt`. Τρέξτε `git status`: πώς καταγράφεται το νέο αρχείο;
3. Προσθέστε το με `git add command/solution.txt` και τρέξτε ξανά `git status`.
4. Τρέξτε `git commit` και δώστε ένα μήνυμα που περιγράφει την αλλαγή σας. Τρέξτε `git status` και πάλι.
5. Τρέξτε `git push` και ελέγξτε ότι οι αλλαγές σας εμφανίζονται στο GitHub!
6. Κάντε κάποιες αλλαγές στο repository από το GitHub UI και μετά τρέξτε `git pull`.

Τα βήματα προϋποθέτουν ότι έχετε φτιάξει ένα κλειδί για τον GitHub λογαριασμό σας και έχετε τελειώσει το configuration όπως περιγράφεται στο Φυλλάδιο 1 του εργαστηρίου.

Υπόδειξη Μετά από κάθε βήμα, διαβάστε προσεκτικά τι λέει το `git status`: untracked, έτοιμο για commit, ή «ahead» του server. Αν το `clone` ή το `push` αποτυγχάνει με `Permission denied (publickey)`, ελέγξτε το κλειδί SSH (Εργαστήριο 1, Βήμα 6).

A4.3 · Κανόνας αναπροσαρμογής βαθμού ασκήσεων

A4.3 · *Διάλεξη 4: Git και Τελεστές, διαφάνεια 8* · ★☆☆ · programming · `slides-lec04-grade-adjustment`

Στο πρόγραμμα υπολογισμού βαθμολογίας (50% Τελική Εξέταση + 30% Ασκήσεις + 20% Εργαστήριο) προστίθεται ένας κανόνας:

Κανόνας Αναπροσαρμογής: Αν ο βαθμός από τις Ασκήσεις είναι πάνω από 3 μονάδες μεγαλύτερος του βαθμού από την Τελική Εξέταση, τότε ο βαθμός για τις Ασκήσεις αναπροσαρμόζεται σε Τελική Εξέταση + 3. Πως θα το εκφράσουμε;

Υπόδειξη Γράψτε πρώτα τη συνθήκη ως σύγκριση ανάμεσα σε `homework` και `final_exam + 3`. Ένας τελεστής της διάλεξης επιλέγει ανάμεσα σε δύο τιμές ανάλογα με μια συνθήκη, και το αποτέλεσμά του μπορεί να ανατεθεί ξανά στο `homework` πριν από τον υπολογισμό.

A4.4 · Υπολογισμός βαθμολογίας πρωτοετών

A4.4 · *Διάλεξη 4: Git και Τελεστές, διαφάνεια 8* · ★☆☆ · programming · `slides-lec04-grade-program`

Ο βαθμός των πρωτοετών υπολογίζεται ως:

50% * Τελική Εξέταση + 30% * Ασκήσεις + 20% * Εργαστήριο

Έστω ότι χρησιμοποιούμε 3 ακεραίους για να αναπαραστήσουμε τα αποτελέσματα [0 - 100]: `int final_exam`, `int homework`, `int lab`.

Στα μαθηματικά:

grade(final_exam, homework, lab) = final_exam x 50% + homework x 30% + lab x 20%

Έλεγχος ορθότητας: `grade(70, 80, 100) == 79`?

Σε C; Γράψτε μια συνάρτηση `grade` και ένα πρόγραμμα που παίρνει τους τρεις βαθμούς ως ορίσματα της γραμμής εντολών και τυπώνει τον τελικό βαθμό:

```
$ ./grade 70 80 100
My grade is: 79
```

Υπόδειξη Τα ορίσματα φτάνουν στη `main` ως κείμενο μέσω των `argc`/`argv`· ελέγξτε πρώτα ότι είναι τρία και μετατρέψτε τα σε ακεραίους με μια συνάρτηση του `stdlib.h`. Με ακεραίους, προσέξτε με ποια σειρά γίνονται ο πολλαπλασιασμός και η διαίρεση με το 100.

A4.5 · Συνάρτηση max με τον τελεστή συνθήκης

A4.5 · *Διάλεξη 4: Git και Τελεστές, διαφάνεια 31* · ★☆☆ · programming · `slides-lec04-max-conditional`

Ο τελεστής συνθήκης ελέγχει την συνθήκη και αν είναι αληθής επιστρέφει την πρώτη τιμή, αλλιώς επιστρέφει την δεύτερη. Γενική μορφή:

`συνθήκη ? τιμή1 : τιμή2`

Για παράδειγμα, `(42 > 0) ? 8 : 16` επιστρέφει `8` και `(42 == 0) ? 8 : 16` επιστρέφει `16`.

Τι τύπου είναι ο τελεστής συνθήκης; Πως θα φτιάχναμε μια συνάρτηση `max`;

Υπόδειξη Μετρήστε τους τελεστέους του. Επειδή ο τελεστής επιστρέφει τιμή, η συνάρτηση `max` δύο ακεραίων μπορεί να αποτελείται από μία μόνο εντολή `return`.

A4.6 · Απομόνωση του πιο σημαντικού bit

A4.6 · *Διάλεξη 4: Git και Τελεστές, διαφάνεια 29* · ★☆☆ · multiple-choice · `slides-lec04-msb`

Πως απομονώνουμε το πιο σημαντικό bit σε ένα byte;

- A. `byte & 0xFF`
- B. `byte & 0x80`
- C. `byte | 0xFF`
- D. `byte | 0x80`

Υπόδειξη Γράψτε τις σταθερές `0xFF` και `0x80` σε δυαδικό. «Απομονώνω» σημαίνει ότι όλα τα άλλα bit γίνονται 0 και μένει μόνο το bit που θέλουμε: ποιος τελεστής, `&` ή `|`, μπορεί να μηδενίσει bit;

A4.7 · Τι τύπου τελεστές είναι;

A4.7 · *Διάλεξη 4: Git και Τελεστές, διαφάνειες 25-28 και 31* · ★☆☆ · short-answer · `slides-lec04-operator-arity`

Με βάση το arity (πόσους τελεστέους έχουν) οι τελεστές είναι μοναδιαίοι (unary), δυαδικοί (binary) ή τριαδικοί (ternary). Σε ποια κατηγορία εντάσσονται:

1. οι αριθμητικοί τελεστές `+`, `-`, `*`, `/`, `%`;
2. οι συγκριτικοί τελεστές `==`, `!=`, `>`, `<`, `>=`, `<=`;
3. οι λογικοί τελεστές `&&`, `||`, `!`;
4. οι bitwise τελεστές `&`, `|`, `^`, `~`, `<<`, `>>`;
5. ο τελεστής συνθήκης `συνθήκη ? τιμή1 : τιμή2`;

Υπόδειξη Μετρήστε πόσους τελεστέους χρειάζεται κάθε τελεστής σε ένα παράδειγμα χρήσης του. Προσέξτε ότι σε δύο από τις ομάδες υπάρχει ένας τελεστής που διαφέρει από τους υπόλοιπους της ομάδας.

Α4.8 · Τελεστές ως συναρτήσεις

A4.8 · *Διάλεξη 4: Git και Τελεστές, διαφάνειες 21 και 25* · ★☆☆ · short-answer · `slides-lec04-operators-as-functions`

Στην παράσταση `x + 42` η μεταβλητή `x` είναι ο 1ος τελεστέος (operand), η σταθερά `42` ο 2ος, και το `+` ο τελεστής (operator) για πρόσθεση.

1. Μας θυμίζει κάτι ο τελεστής και οι τελεστέοι;
2. Τι τύπο θα είχαν οι αριθμητικοί τελεστές (`+`, `-`, `*`, `/`, `%`) αν ήταν συναρτήσεις;

Υπόδειξη Σκεφτείτε τι παίρνει και τι επιστρέφει μια συνάρτηση. Για το δεύτερο, γράψτε το πρωτότυπο μιας συνάρτησης `plus` για ακεραίους και μετά για πραγματικούς· τι σας λέει αυτό για το `85 / 2` και το `85.0 / 2.0`;

Α4.9 · Προτεραιότητα και προσεταιριστικότητα

A4.9 · *Διάλεξη 4: Git και Τελεστές, διαφάνεια 39* · ★☆☆ · trace · `slides-lec04-precedence`

Η προτεραιότητα ενός τελεστή καθορίζει την σειρά με την οποία θα εκτελεστούν οι υπολογισμοί. Για παράδειγμα, ο πολλαπλασιασμός έχει υψηλότερη προτεραιότητα από την πρόσθεση:

`5 * 6 + 3 * 4 επιστρέφει ??`

Αν δύο τελεστές έχουν την ίδια προτεραιότητα, η προσεταιριστικότητα τους καθορίζει την σειρά των υπολογισμών (αριστερά προς τα δεξιά ή το αντίστροφο). Για παράδειγμα, ο πολλαπλασιασμός και η διαίρεση έχουν την ίδια προτεραιότητα, ενώ η προσεταιριστικότητα τους είναι από αριστερά προς τα δεξιά:

`90 * 50 / 100 επιστρέφει ??`

Υπόδειξη Βάλτε παρενθέσεις σύμφωνα με τους δύο κανόνες και υπολογίστε από μέσα προς τα έξω. Για το δεύτερο, σκεφτείτε αν θα άλλαζε το αποτέλεσμα (με ακεραίους) αν η σειρά ήταν από δεξιά προς τα αριστερά.

A4.10 · Τι επιστρέφει το (double)3/4;

A4.10 · *Διάλεξη 4: Git και Τελεστές, διαφάνεια 32* · ★★☆ · trace · `slides-lec04-cast-division`

Ο τελεστής μετατροπής αλλάζει τον τύπο ενός τελεστέου. Γενική μορφή: `(τύπος)τελεστέος`.

Παραδείγματα:

- `(int)42.67` επιστρέφει `42`
- `(char)67.8` επιστρέφει `'C'`
- `(double)3` επιστρέφει `3.0`
- `(double)3/4` επιστρέφει ??

Υπόδειξη Βρείτε στον πίνακα προτεραιότητας ποιος εφαρμόζεται πρώτα, το cast ή η διαίρεση. Μετά σκεφτείτε τι τύπου είναι οι δύο τελεστέοι της διαίρεσης και τι κάνει η σιωπηρή μετατροπή τύπων. Συγκρίνετε με το `(double)(3/4)`.

Εργαστήριο (A4.11)

A4.11 · Το πρώτο σας repository

A4.11 · *Εργαστήριο 1, Άσκηση 1* · ★☆☆ · tooling · `lab-lab01-info`

Πατήστε τον ακόλουθο σύνδεσμο για να δημιουργήσετε το δικό σας repository `progintro/lab01-YourGitHub` και πραγματοποιήστε τα ακόλουθα βήματα:

1. Κατεβάστε το repository τοπικά τρέχοντας την εντολή:

```
$ git clone git@github.com:progintro/lab01-YourGitHub
Cloning into 'lab01-ethan42'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
Receiving objects: 100% (3/3), 5.10 KiB | 871.00 KiB/s, done.
remote: Total 3 (delta 0), reused 2 (delta 0), pack-reused 0 (from 0)
```

2. Μέσα στον φάκελο `lab01-YourGitHub` που δημιούργησε η παραπάνω εντολή, προσθέστε ένα αρχείο `info.txt` με τα ακόλουθα στοιχεία:

```
Όνομα, Επώνυμο, sdiXXYYYYY (εφόσον έχετε)
```

3. Προσθέστε το αρχείο σας στο repository χρησιμοποιώντας την εντολή `git add`:

```
git add info.txt
```

4. Τρέξτε την εντολή `git commit` για να μονιμοποιήσετε τις αλλαγές σας:

```
git commit
```

5. Τρέξτε την εντολή `git push` για να ανεβάσετε τις αλλαγές σας στο GitHub.

```
$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 319 bytes | 63.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:progintro/lab01-ethan42
   4ef457c..aa9f709  main -> main
```

Επιβεβαιώστε ότι βλέπετε τις αλλαγές πηγαίνοντας στο αντίστοιχο link του repository σας: https://github.com/progintro/lab01-YourGitHub.

Υπόδειξη Το `git clone` με διεύθυνση `git@github.com:` χρησιμοποιεί ssh, οπότε πρέπει πρώτα να έχετε προσθέσει το δημόσιο κλειδί σας στο GitHub και να έχετε ρυθμίσει `user.name` και `user.email`. Θυμηθείτε τα τρία στάδια: `add` (προετοιμασία), `commit` (τοπική καταγραφή), `push` (αποστολή στο GitHub). Με `git status` βλέπετε σε ποιο στάδιο είστε.

Εργασίες (A4.12)

A4.12 · Νέο URL στο GitHub (pages)

A4.12 · *Εργασία 0 (2025-26), Άσκηση 1* · ★☆☆ · tooling · `hw-2025-hw0-pages`

Έχεις λογαριασμό στο GitHub; Έχεις ξαναφτιάξει repository; Έχεις φτιάξει URL στο διαδίκτυο; Μετά από αυτήν την άσκηση θα μπορείς να απαντήσεις ”ναι” σε όλα.

Η ιστοσελίδα του μαθήματος βασίζεται σε ένα public GitHub repository. Σκοπός αυτής της άσκησης είναι να δημιουργήσεις ένα καινούριο repository στον προσωπικό σου λογαριασμό GitHub και να το συνδέσεις με GitHub Pages προκειμένου να είναι διαθέσιμο στο διαδίκτυο. Αυτό το tutorial μπορεί να σε βοηθήσει.

Τεχνικές Προδιαγραφές Η υποβολή σου πρέπει να έχει τα ακόλουθα χαρακτηριστικά:

- Repository Name: `progintro/hw0-<YourUsername>`
- Αρχείο Λύσης Filepath: `pages/solution.txt`

- README Filepath (optional): `pages/README.md`

Έστω ότι το GitHub username σου είναι `YourUsername`. Το αρχείο `solution.txt` πρέπει να περιέχει το URL με το domain `YourUsername.github.io` που δημιουργήσατε. Αν χρησιμοποιήσετε κάποιο `*.github.io` URL που δεν δημιουργήσατε η άσκηση μηδενίζεται. Ενδεικτική συμπεριφορά για μια επιτυχημένη υποβολή (**προσοχή**: **η γραμμή με την εντολή curl εκτείνεται σε δύο γραμμές, πρέπει να αντιγράψετε και τις δύο προκειμένου να τρέξετε αυτήν την εντολή**, το backslash `\` δείχνει ότι η εντολή είναι μία αλλά εκτείνεται σε δύο γραμμές):

```
##### Check out solution URL
$ cat solution.txt
YourUsername.github.io
##### Ensure the URL exists
$ curl --output /dev/null --silent --head --fail YourUsername.github.io && \
  echo "URL exists" || echo "URL does not exist"
URL exists
```

Το αρχείο `README.md` είναι προαιρετικό αν θέλετε να προσθέσετε κάτι στην υποβολή σας.

Εμφανίσεις

- Εργασία 0 (2023-24), Άσκηση 1: ίδια εκφώνηση, χωρίς τον σύνδεσμο προς το tutorial του GitHub Pages.
- Εργασία 0 (2024-25), Άσκηση 1: ίδια εκφώνηση (20 μονάδες).
- Εργασία 0 (2025-26), Άσκηση 1: η εκφώνηση που δίνεται εδώ.

Υπόδειξη Το GitHub Pages σερβίρει αυτόματα ένα repository με όνομα ακριβώς `<username>.github.io` στον λογαριασμό σου, αρκεί να περιέχει ένα αρχείο όπως το `index.html`. Δοκίμασε το URL με την εντολή `curl` της εκφώνησης πριν το γράψεις στο `solution.txt`, και θυμήσου ότι το `solution.txt` ζει σε άλλο repository (το `hw0-<YourUsername>`), οπότε χρειάζεσαι `git add`, `commit` και `push` και εκεί.

Θέματα εξετάσεων (A4.13)

A4.13 · Άρτια Bits

A4.13 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #8, Θέμα 1* · ★☆☆ · programming · `exam-2023-fall-ex8-q1`

Πρόγραμμα: `parity.c`

Γράψτε ένα πρόγραμμα που παίρνει δεκαδικούς ακεραίους ως ορίσματα από την γραμμή εντολών και τυπώνει για κάθε έναν από αυτούς εάν έχει έναν άρτιο αριθμό από bits που είναι

1 (even parity). Για παράδειγμα ο αριθμός 5 είναι ο 101 στο δυαδικό και επομένως έχει άρτιο αριθμό από bits που είναι 1. Αντίθετα ο αριθμός 7 (111 στο δυαδικό) έχει περιττό αριθμό από bits που έχουν τεθεί στο 1. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ ./parity 5 7 64 65 987234 97823462
Number 5 has even parity
Number 7 does not have even parity
Number 64 does not have even parity
Number 65 has even parity
Number 987234 has even parity
Number 97823462 does not have even parity
$ ./parity 92873847981279843
Number 92873847981279843 has even parity
$ ./parity 92873847981279844
Number 92873847981279844 does not have even parity
```

Υπόδειξη Μετρήστε τα bits που είναι 1 με μάσκα και ολίσθηση (`&`, `>>`) μέχρι ο αριθμός να γίνει 0. Προσέξτε ότι τα παραδείγματα έχουν αριθμούς πάνω από 2^31, άρα χρειάζεστε 64-bit τύπο (`long long` / `strtoll`) και έλεγχο ότι κάθε όρισμα είναι έγκυρος ακέραιος.

Κεφάλαιο 5: Τελεστές και Εντολές

Kahoot από το αμφιθέατρο (K5.1–K5.3)

K5.1 · Μετατροπή double σε int

K5.1 · *Kahoot «Τελεστές, Εντολές και Ροή Ελέγχου» (διάλεξη 6)* · ★☆☆ · multiple-choice · 86% σωστές · `kahoot-cast-truncation`

Τι επιστρέφει η παράσταση `(int)42.842`;

- 42
- 43
- 42.8
- 42.842

Υπόδειξη Ο τελεστής μετατροπής σε `int` κρατά μόνο το ακέραιο μέρος· στρογγυλοποιεί ή αποκόπτει;

K5.2 · Ποιος αριθμός ικανοποιεί τη συνθήκη

K5.2 · *Kahoot «Επίλυση Προβλημάτων» (διάλεξη 7)* · ★★☆ · multiple-choice · 58% σωστές · `kahoot-num-divisible-3-31`

Ποιος αριθμός `num` κάνει την ακόλουθη έκφραση αληθή;

```
(num > 10) && (num < 100) && (num % 3 == 0) && (num % 31 == 0)
```

- 3
- 31
- 42
- 93

Υπόδειξη Με το `&&` πρέπει να ισχύουν και οι τέσσερις συνθήκες ταυτόχρονα· ελέγξτε κάθε επιλογή μία προς μία.

K5.3 · x++ και ++y

K5.3 · *Kahoot «Τελεστές, Εντολές και Ροή Ελέγχου» (διάλεξη 6)* · ★★☆ · multiple-choice · 48% σωστές · `kahoot-post-pre-increment`

Τι θα τυπώσει το πρόγραμμα;

```
int x = 42, y = 42;
printf("%d %d", x++, ++y);
```

- 42 42
- 43 42
- 42 43
- 43 43

Υπόδειξη Και οι δύο τελεστές αυξάνουν τη μεταβλητή κατά 1· η διαφορά τους είναι ποια τιμή έχει η ίδια η παράσταση, πριν ή μετά την αύξηση.

Ζέσταμα: από τις διαφάνειες (A5.1–A5.7)

A5.1 · Η τιμή της 0xbeef | 0xcafe0000

A5.1 · *Διάλεξη 5: Τελεστές και Εντολές, διαφάνεια 12* · ★☆☆ · multiple-choice · `slides-lec05-bitwise-or-hex`

Ποια η τιμή της παράστασης `0xbeef | 0xcafe0000`;

- A. `0xffffffff`

- B. `0xcafebeef`
- C. `0xbeefcafe`
- D. `0xfaceb00c`

Υπόδειξη Γράψτε και τους δύο αριθμούς με 8 δεκαεξαδικά ψηφία, τον έναν κάτω από τον άλλον. Τι δίνει το `x | 0` σε κάθε θέση;

A5.2 · Cast και διαίρεση: (double)3/4

A5.2 · *Διάλεξη 5: Τελεστές και Εντολές, διαφάνεια 14* · ★☆☆ · trace · `slides-lec05-cast-division`

Ο τελεστής μετατροπής αλλάζει τον τύπο ενός τελεστέου, με γενική μορφή `(τύπος)τελεστέος`. Για παράδειγμα:

```
(int)42.67   επιστρέφει 42
(char)67.8   επιστρέφει 'C'
(double)3    επιστρέφει 3.0
(double)3/4  επιστρέφει ??
```

Τι επιστρέφει το `(double)3/4`;

Υπόδειξη Αποφασίστε πρώτα τι εφαρμόζεται πρώτο, το cast ή η διαίρεση (δείτε τον πίνακα προτεραιότητας). Μετά σκεφτείτε τι γίνεται σε μια διαίρεση όπου ο ένας τελεστέος είναι `double` και ο άλλος `int`. Συγκρίνετε με το `(double)(3/4)`.

A5.3 · Απομόνωση του πιο σημαντικού bit

A5.3 · *Διάλεξη 5: Τελεστές και Εντολές, διαφάνεια 11* · ★☆☆ · multiple-choice · `slides-lec05-msb`

Πώς απομονώνουμε το πιο σημαντικό bit σε ένα byte;

- A. `byte & 0xFF`
- B. `byte & 0x80`
- C. `byte | 0xFF`
- D. `byte | 0x80`

Υπόδειξη Γράψτε τα `0xFF` και `0x80` στο δυαδικό. Ποιος τελεστής μηδενίζει τα bit που δεν σας ενδιαφέρουν και ποιος τα κάνει 1;

A5.4 · Τι τύπου τελεστές είναι;

A5.4 · *Διάλεξη 5: Τελεστές και Εντολές, διαφάνειες 7-10, 13* · ★☆☆ · short-answer · `slides-lec05-operator-arity`

Οι τελεστές χωρίζονται σε μοναδιαίους (unary), δυαδικούς (binary) και τριαδικούς (ternary). Σε ποια κατηγορία ανήκουν:

1. οι αριθμητικοί τελεστές `+`, `-`, `*`, `/`, `%`;
2. οι συγκριτικοί τελεστές `==`, `!=`, `>`, `<`, `>=`, `<=`;
3. οι λογικοί τελεστές `&&`, `||`, `!`;
4. οι bitwise τελεστές `&`, `|`, `^`, `~`, `<<`, `>>`;
5. ο τελεστής συνθήκης `συνθήκη ? τιμή1 : τιμή2`;

Υπόδειξη Μετρήστε πόσους τελεστέους παίρνει ο καθένας στα παραδείγματα των πινάκων (π.χ. `42 > 0 && 2 > 3`, `!(42 < 0)`, `~0xF0`). Προσέξτε ότι μέσα στην ίδια οικογένεια μπορεί να υπάρχουν τελεστές διαφορετικής κατηγορίας.

A5.5 · Οι αριθμητικοί τελεστές ως συναρτήσεις

A5.5 · *Διάλεξη 5: Τελεστές και Εντολές, διαφάνεια 7* · ★☆☆ · short-answer · `slides-lec05-operator-as-function`

Δίνεται ο πίνακας των αριθμητικών τελεστών:

Τελεστής	Ερμηνεία	Παραδείγματα
`+`	πρόσθεση	`40 + 2` επιστρέφει 42, `40.0 + 2.0` επιστρέφει `42.0`
`-`	αφαίρεση	`44 - 2` επιστρέφει 42, `44.0 - 2.0` επιστρέφει `42.0`
`*`	πολλαπλασιασμός	`42 * 2` επιστρέφει 84, `42.0 * 2.0` επιστρέφει `84.0`
`/`	διαίρεση	`85.0 / 2.0` επιστρέφει `42.5`, `85 / 2` επιστρέφει 42
`%`	υπόλοιπο	`7 % 2` επιστρέφει 1

Τι τύπο θα είχαν οι τελεστές αν ήταν συναρτήσεις;

Υπόδειξη Γράψτε την επικεφαλίδα (πρωτότυπο) μιας συνάρτησης `add` που κάνει ό,τι το `40 + 2`: πόσα ορίσματα παίρνει, τι τύπου, και τι επιστρέφει; Μετά σκεφτείτε τι αλλάζει για το `40.0 + 2.0` και γιατί το `85 / 2` δίνει 42.

A5.6 · Προτεραιότητα και προσεταιριστικότητα

A5.6 · *Διάλεξη 5: Τελεστές και Εντολές, διαφάνεια 21* · ★☆☆ · trace · `slides-lec05-precedence`

Η προτεραιότητα ενός τελεστή καθορίζει την σειρά με την οποία θα εκτελεστούν οι υπολογισμοί. Για παράδειγμα, ο πολλαπλασιασμός έχει υψηλότερη προτεραιότητα από την πρόσθεση:

```
5 * 6 + 3 * 4    // επιστρέφει ??
```

Αν δύο τελεστές έχουν την ίδια προτεραιότητα, η προσεταιριστικότητα τους καθορίζει την σειρά των υπολογισμών (αριστερά προς τα δεξιά ή το αντίστροφο). Για παράδειγμα, ο πολλαπλασιασμός και η διαίρεση έχουν την ίδια προτεραιότητα, ενώ η προσεταιριστικότητα τους είναι από αριστερά προς τα δεξιά:

```
90 * 50 / 100    // επιστρέφει ??
```

Τι επιστρέφει η κάθε έκφραση (οι αριθμοί είναι `int`);

Υπόδειξη Βάλτε παρενθέσεις εκεί όπου τις βάζει ο μεταγλωττιστής και υπολογίστε από μέσα προς τα έξω. Στη δεύτερη, σκεφτείτε τι θα βγαίνει αν η διαίρεση γινόταν πρώτη, με ακέραια διαίρεση.

A5.7 · Συνάρτηση max με τον τελεστή συνθήκης

A5.7 · *Διάλεξη 5: Τελεστές και Εντολές, διαφάνεια 13* · ★☆☆ · programming · `slides-lec05-ternary-max`

Ο τελεστής συνθήκης ελέγχει την συνθήκη και αν είναι αληθής επιστρέφει την πρώτη τιμή, αλλιώς επιστρέφει την δεύτερη. Γενική μορφή:

`συνθήκη ? τιμή1 : τιμή2`

Για παράδειγμα, `(42 > 0) ? 8 : 16` επιστρέφει `8` και `(42 == 0) ? 8 : 16` επιστρέφει `16`.

Τι τύπου είναι ο τελεστής συνθήκης; Πώς θα φτιάχναμε μια συνάρτηση `max`;

Υπόδειξη Μετρήστε τους τελεστέους του. Για τη `max`: η συνάρτηση παίρνει δύο ακεραίους και ολόκληρο το σώμα της μπορεί να είναι ένα `return` με μια έκφραση συνθήκης.

Εργαστήριο (A5.8)

A5.8 · Υπολογισμός ημέρας μίας δεδομένης ημερομηνίας

A5.8 · *Εργαστήριο 3, Άσκηση 3* · ★☆☆ · programming · `lab-lab03-birthdate`

Ο ακόλουθος αλγόριθμος υπολογίζει την ημέρα που αντιστοιχεί σε μία δεδομένη ημερομηνία:

Έστω η ημερομηνία $DD/MM/YYYY$ (δύο ψηφία για την μέρα DD - Day, δύο ψηφία για τον μήνα MM - Month, 4 ψηφία για τον χρόνο YYYY - Year).

Αν MM ≤ 2, **τότε**:

$$\begin{aligned} \text{NYYYY} &= \text{YYYY} - 1 \\ \text{NMM} &= 0 \end{aligned}$$

Αν MM > 2, **τότε**:

$$\begin{aligned} \text{NYYYY} &= \text{YYYY} \\ \text{NMM} &= \left\lfloor \frac{4 \cdot \text{MM} + 23}{10} \right\rfloor \end{aligned}$$

$$IDAY = 365 \cdot YYYY + DD + 31 \cdot (MM - 1) - NMM + \left\lfloor \frac{NYYYY}{4} \right\rfloor - \left\lfloor \frac{3}{4} \cdot \left(\left\lfloor \frac{NYYYY}{100} \right\rfloor + 1 \right) \right\rfloor$$

Για τον υπολογισμό της ημέρας, αν:

- $IDAY \mod 7 = 0$, τότε DAY = **Saturday**
- $IDAY \mod 7 = 1$, τότε DAY = **Sunday**
- ...
- $IDAY \mod 7 = 6$, τότε DAY = **Friday**

Όπου, με $\lfloor x \rfloor$ συμβολίζουμε τον μέγιστο ακέραιο αριθμό που δεν είναι μεγαλύτερος του αριθμού x - γνωστή και ως συνάρτηση `floor` (αντίστοιχα υπάρχει και η συνάρτηση ceil). Δεν είμαστε σίγουροι πώς λειτουργεί - πώς θα το βρούμε;

Εργασία: Γράψτε ένα πρόγραμμα `birthdate.c` το οποίο να χρησιμοποιεί τον αλγόριθμο παραπάνω και να υπολογίζει τι ημέρα γεννηθήκατε (ενσωματώστε την ημερομηνία γέννησής σας μέσα στο πρόγραμμα).

Ο παραπάνω υπολογισμός φαίνεται περίπλοκος; Δεν έχετε δει τίποτε ακόμη! Πραγματικές συναρτήσεις βιβλιοθήκης ημερομηνιών πρέπει να χειρίζονται ημερομηνίες και στο παρελθόν, με ιδιαίτερα περίπλοκες αλλαγές - τι γίνεται για παράδειγμα αν ο χρήστης θέλει να βρει την ημερομηνία 16 Φεβρουαρίου 1923 στην Ελλάδα [8];

Υπόδειξη Για θετικούς αριθμούς, η ακέραια διαίρεση της C κάνει ήδη το $\lfloor \cdot \rfloor$, οπότε δεν χρειάζεστε `floor`. Προσέξτε τον όρο $\frac{3}{4} \cdot (\dots)$: αν τον γράψετε αφελώς σε ακέραια αριθμητική, το `3/4` βγαίνει 0, άρα αλλάξτε τη σειρά των πράξεων (πρώτα πολλαπλασιασμός, μετά διαίρεση). Για την ονομασία της ημέρας αρκεί μια αλυσίδα `if/else if` ή ένα `switch` πάνω στο υπόλοιπο.

Θέματα εξετάσεων (A5.9–A5.11)

A5.9 · Mystery

A5.9 · *Εξέταση Ιανουαρίου 2025, Θέμα 1* · ★☆☆ · trace · `exam-2025-jan-q1`

Mystery [10 Μονάδες]

Η συνάρτηση `mystery` δέχεται ως όρισμα έναν ακέραιο αριθμό που αποτελείται από τα 3 τελευταία ψηφία του sdi σας. Για παράδειγμα, αν ο sdi σας είναι ο sdi2400789 τότε καλούμε την συνάρτηση ως `mystery(789)`. Τι θα τυπώσει η συνάρτηση για τα ψηφία του δικού σας sdi; Αν δεν έχετε sdi, επιλέξτε έναν τυχαίο τριψήφιο. Αιτιολογήστε όπου νομίζετε πως χρειάζεται.

```
int mystery(int number) {
  int i;
  int step = 2;
  printf("%s %d\n", "Starting loop", i);
  for(i = 0; i < 3; i++) {
    printf("step %d %d\n", i, number + step);
    step = (step << 2) + 2;
  }
  printf("Ending loop %d\n", i);
  return number + step;
}
```

Υπόδειξη Κοιτάξτε προσεκτικά την πρώτη `printf`: τι τιμή έχει το `i` εκείνη τη στιγμή; Για το `step`, θυμηθείτε ότι το `<< 2` πολλαπλασιάζει επί 4, και φτιάξτε έναν πίνακα με τις τιμές του `i` και του `step` σε κάθε επανάληψη. Προσέξτε και την τιμή του `i` μετά το τέλος του βρόχου.

A5.10 · Mystery

A5.10 · *Εξέταση Ιανουαρίου 2026, Θέμα 1* · ★☆☆ · trace · `exam-2026-jan-q1`

Mystery [10 Μονάδες]

Η συνάρτηση `mystery` δέχεται ως όρισμα έναν ακέραιο αριθμό που αποτελείται από τα 3 τελευταία ψηφία του sdi σας. Για παράδειγμα, αν ο sdi σας είναι ο sdi2500789 τότε καλούμε την συνάρτηση ως `mystery(789)`. Τι θα τυπώσει η συνάρτηση για τα ψηφία του δικού σας sdi; Αν δεν έχετε sdi, επιλέξτε έναν τυχαίο τριψήφιο. Αιτιολογήστε όπου νομίζετε πως χρειάζεται.

```
int mystery(int number) {
  int seed = number % 10;
  int count = (seed == 0 || seed > 4) ? 2 : seed;
  printf("Seed %d, count: %d\n", seed, count);
  for(int i = 0; i < count; ++i) {
```

```
    printf("Loop %d: %d\n", i, ++seed);
    seed &= 0xF;
  }
  printf("Result: %d\n", seed);
  return seed;
}
```

Υπόδειξη Μόνο το τελευταίο ψηφίο μετράει· βρείτε πρώτα τι κάνει ο τελεστής συνθήκης `?:` για το `count`. Προσέξτε ότι το `++seed` αυξάνει την τιμή *πριν* τυπωθεί, και ότι το `& 0xF` κρατά μόνο τα 4 χαμηλότερα bit: τι γίνεται όταν το `seed` φτάσει το 16;

A5.11 · Η συνάρτηση mystery

A5.11 · *Εξέταση Ιουνίου 2026, Θέμα 1* · ★☆☆ · trace · `exam-2026-jun-q1`

Η Συνάρτηση mystery (15 Μονάδες)

Η συνάρτηση `mystery` δέχεται ως όρισμα έναν ακέραιο αριθμό που αποτελείται από τα 3 τελευταία ψηφία του sdi σας. Για παράδειγμα, αν ο sdi σας είναι ο sdi2500789 τότε καλούμε την συνάρτηση ως `mystery(789)`. Τι θα τυπώσει η συνάρτηση για τα ψηφία του δικού σας sdi; Αν δεν έχετε sdi, επιλέξτε έναν τυχαίο τριψήφιο. Αιτιολογήστε όπου νομίζετε πως χρειάζεται.

```
int mystery(int number) {
  int i = 0;
  while(number > 4) {
    printf("Loop %d: %d\n", i++, number);
    number >>= 2;
  }
  printf("Result: %c\n", '0' + number);
  return number;
}
```

Υπόδειξη Το `number >>= 2` για θετικό ακέραιο ισοδυναμεί με ακέραια διαίρεση με το 4. Κρατήστε έναν πίνακα με τις τιμές των `i` και `number` σε κάθε επανάληψη, προσέχοντας ότι το `i++` τυπώνει την τιμή *πριν* την αύξηση. Στο τέλος, σκεφτείτε ποιος χαρακτήρας αντιστοιχεί στον κωδικό ASCII `'0' + number` όταν το `number` είναι μεταξύ 0 και 4.

Κεφάλαιο 6: Εντολές και Ροή Ελέγχου

Kahoot από το αμφιθέατρο (K6.1–K6.9)

K6.1 · Απλό if-else

K6.1 · *Kahoot «Τελεστές, Εντολές και Ροή Ελέγχου» (διάλεξη 6)* · ★☆☆ · multiple-choice · 91% σωστές · `kahoot-if-else-trace`

Τι θα τυπώσει το ακόλουθο πρόγραμμα;

```
int x = 42;
if (x > 42)
    printf("Hello");
else
    printf("World");
```

- Hello
- World
- HelloWorld
- Hello World

Υπόδειξη Υπολογίστε τη συνθήκη `x > 42` για `x = 42` και θυμηθείτε ότι εκτελείται ακριβώς ένας από τους δύο κλάδους.

K6.2 · while και do-while

K6.2 · *Kahoot «Τελεστές, Εντολές και Ροή Ελέγχου» (διάλεξη 6)* · ★☆☆ · multiple-choice · 85% σωστές · `kahoot-while-vs-do-while`

Τα `while` και `do-while` loops είναι το ίδιο πράγμα;

- True
- False

Υπόδειξη Σκεφτείτε πότε ελέγχεται η συνθήκη σε κάθε βρόχο και τι γίνεται αν είναι ψευδής από την αρχή.

K6.3 · Η σύνταξη του for

K6.3 · *Kahoot «Πίνακες» (διάλεξη 10)* · ★★☆ · multiple-choice · 59% σωστές · `kahoot-for-empty-init`

50% το κάνουν λάθος: αφού αρχικοποιήσω `i = 0;`, θέλω να τυπώσω hello 100 φορές. Κάνω:

- `for(i < 100; i++;) printf("hello\n");`
- `for(; i < 100; i++) printf("hello\n");`

Συχνή παρανόηση Το 27% επέλεξε `for(i < 100; i++;)`, βάζοντας τη συνθήκη στη θέση της αρχικοποίησης. Τα τρία μέρη του `for` είναι πάντα αρχικοποίηση; συνθήκη; βήμα, και ένα κενό μέρος αφήνει μόνο το ερωτηματικό του.

Υπόδειξη Η παρένθεση του `for` έχει τρία μέρη με σταθερή σειρά· αν ένα λείπει, μένει κενό αλλά το ερωτηματικό του μένει.

Κ6.4 · Επαναλήψεις με βήμα 2

Κ6.4 · *Kahoot «Τελεστές, Εντολές και Ροή Ελέγχου» (διάλεξη 6)* · ★★☆ · multiple-choice · 57% σωστές · `kahoot-for-step-two`

Το loop `int i; for (i = 0; i < 100; i += 2);` θα εκτελεστεί:

- 100 φορές
- 99 φορές
- 49 φορές
- 50 φορές

Συχνή παρανόηση Το 27% επέλεξε 49 φορές, ένα κλασικό λάθος «κατά ένα» (off-by-one): ξεχνά ότι μετράει και η επανάληψη με `i = 0`.

Υπόδειξη Γράψτε τις τιμές του `i` για τις οποίες εκτελείται το σώμα: από πού ξεκινά, πού σταματά και πόσες είναι.

Κ6.5 · Ένα for χωρίς βήμα

Κ6.5 · *Kahoot «Επίλυση Προβλημάτων» (διάλεξη 7)* · ★★☆ · multiple-choice · 56% σωστές · `kahoot-for-missing-increment`

Πότε τελειώνει αυτό το loop;

```
int i;
for (i = 0; i < 42; );
```

- Μετά από 2^32 επαναλήψεις
- Όταν το i γίνει αρνητικό
- Μετά από 42 επαναλήψεις

- Όταν λιώσει η CPU

Υπόδειξη Κοιτάξτε ποιο από τα τρία μέρη της `for` λείπει και αν κάτι άλλο μέσα στον βρόχο αλλάζει την τιμή του `i`.

K6.6 · if χωρίς παρενθέσεις

K6.6 · *Kahoot «Τελεστές, Εντολές και Ροή Ελέγχου» (διάλεξη 6)* · ★★☆ · multiple-choice · 42% σωστές · `kahoot-if-without-parens`

Η εντολή `if x == 42 printf("Hello world\n");` είναι δεκτή στην C.

- True
- False

Συχνή παρανόηση Το 53% απάντησε True, ίσως από γλώσσες όπως η Python· στην C η συνθήκη της `if` πρέπει να είναι σε παρενθέσεις.

Υπόδειξη Θυμηθείτε ακριβώς τη σύνταξη της `if` όπως δόθηκε στη θεωρία: τι περιβάλλει τη συνθήκη;

K6.7 · while γραμμένο ως for

K6.7 · *Kahoot «Τελεστές, Εντολές και Ροή Ελέγχου» (διάλεξη 6)* · ★★★ · multiple-choice · 24% σωστές · `kahoot-while-as-for`

Το loop `while (condition) statement;` είναι ισοδύναμο με:

- `for(condition ; ; statement);`
- `for(; condition ; statement);`
- `for(; condition;) statement;`
- `for(;;condition) statement;`

Συχνή παρανόηση Το 34% επέλεξε `for(; condition ; statement);`· αυτό δουλεύει μόνο αν το `statement` είναι απλή έκφραση, αφού το τρίτο μέρος της `for` δέχεται έκφραση και όχι οποιαδήποτε εντολή (π.χ. ένα μπλοκ `{…}` ή ένα `if`).

Υπόδειξη Η `for` έχει τρία μέρη (αρχικοποίηση, συνθήκη, βήμα) και σώμα· αντιστοιχίστε σε καθένα τι έχει η `while`.

K6.8 · Πόσες φορές εκτελείται ένα for μέχρι N

K6.8 · *Kahoot «Επίλυση Προβλημάτων» (διάλεξη 7), δύο εκδοχές* · ★★★ · multiple-choice · 23% σωστές · `kahoot-for-n-times-trap`

Πόσες φορές θα τυπώσει `hello` αυτό το loop;

```c
for (int i = 0; i < N; i++)
    printf("hello\n");
```

- N
- N-1
- N+1
- Εξαρτάται

Συχνή παρανόηση Το 34% απάντησε N, υποθέτοντας σιωπηρά ότι το `N` είναι θετικό· αν το `N` είναι 0 ή αρνητικό, το σώμα δεν εκτελείται καθόλου.

Υπόδειξη Δεν ξέρουμε τίποτα για το `N`: δοκιμάστε νοερά τιμές όπως 5, 0 ή -3.

K6.9 · while(!42)

K6.9 · *Kahoot «Επίλυση Προβλημάτων» (διάλεξη 7)* · ★★★ · multiple-choice · 17% σωστές · `kahoot-while-not-42`

Η εντολή `while (!42) printf("yes\n");` θα τυπώσει `yes`:

- 0 φορές
- 42 φορές
- άπειρες φορές
- Δεν είναι έγκυρη C

Συχνή παρανόηση Το 53% απάντησε ότι δεν είναι έγκυρη C, πιστεύοντας ότι η συνθήκη πρέπει να είναι σύγκριση· στην C κάθε ακέραια έκφραση είναι συνθήκη, και το `!42` κάνει 0.

Υπόδειξη Η συνθήκη της `while` μπορεί να είναι οποιαδήποτε έκφραση· υπολογίστε την τιμή της `!42` θυμούμενοι ποιες τιμές η C θεωρεί αληθείς.

Ζέσταμα: από τις διαφάνειες (A6.1–A6.9)

A6.1 · Τι τυπώνει η do-while

A6.1 · *Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 26* · ★☆☆ · trace · `slides-lec06-do-while`

Τι κάνει το παρακάτω πρόγραμμα;

```
i = 0;
do
   i++;
while (i < 42);
printf("%d\n", i);
```

Υπόδειξη Η `do-while` εκτελεί πρώτα το σώμα και μετά ελέγχει τη συνθήκη. Ποια είναι η πρώτη τιμή του `i` για την οποία ο έλεγχος αποτυγχάνει;

Α6.2 · Τιμές μετά από εντολές έκφρασης

A6.2 · *Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 7* · ★☆☆ · trace · `slides-lec06-expression-statements`

Ποια η τιμή των `x`, `y`, `z` μετά την εκτέλεση αυτών των εντολών έκφρασης;

```
x = 4;
y = 7;
z = ++y;
y = z - (x++);
z = x - (--y);
```

Υπόδειξη Φτιάξτε έναν πίνακα με μία στήλη για κάθε μεταβλητή και μία γραμμή για κάθε εντολή. Θυμηθείτε ότι το `++y` αλλάζει πρώτα τη μεταβλητή και δίνει τη νέα τιμή, ενώ το `x++` δίνει την παλιά τιμή και αλλάζει τη μεταβλητή μετά.

Α6.3 · Πόσες φορές εκτελείται μια for

A6.3 · *Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 24* · ★☆☆ · short-answer · `slides-lec06-for-count`

Γενικό παράδειγμα:

```
int i;
for ( i = 0 ; i < N ; i++ ) {
    printf("%d\n", i);
}
```

Πόσες φορές θα εκτελεστεί το block εντολών μέσα στην `for`; Θα τυπωθεί το `N`; Τι θα συμβεί αν αλλάξω την αρχικοποίηση ή το βήμα;

Υπόδειξη Γράψτε τις τιμές του `i` για τις οποίες η συνθήκη είναι αληθής, για ένα μικρό `N` (π.χ. 3). Τι τιμή έχει το `i` τη στιγμή που ο βρόχος σταματάει; Δοκιμάστε μετά με `i = 1` ή με βήμα `i += 2`.

A6.4 · Ο βρόχος for (;;)

A6.4 · *Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 23* · ★☆☆ · short-answer · `slides-lec06-for-empty`

Τι κάνει το παρακάτω πρόγραμμα;

```
for (;;)
```

Υπόδειξη Ποια από τα τρία μέρη της `for` είναι προαιρετικά; Πώς θεωρείται μια συνθήκη που λείπει;

A6.5 · Μέγιστο με if-else και εναλλακτικές

A6.5 · *Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 13* · ★☆☆ · short-answer · `slides-lec06-if-else-max`

Δίνεται ο κώδικας:

```
if (x > y)
    max = x;
else
    max = y;
```

- Θα μπορούσαμε να ”ζήσουμε” χωρίς `else`;
- Υπάρχει εναλλακτικός τρόπος να γράψουμε το παραπάνω;
- Τι κάνουμε όταν έχουμε πάνω από δύο συνθήκες;

Υπόδειξη Για το πρώτο ερώτημα, σκεφτείτε τι θα γινόταν αν δίνατε μια αρχική τιμή στο `max` πριν από την `if`. Για το δεύτερο, θυμηθείτε τους τελεστές της προηγούμενης διάλεξης: υπάρχει ένας που επιλέγει ανάμεσα σε δύο τιμές μέσα σε μία έκφραση. Για το τρίτο, σκεφτείτε τι επιτρέπεται να είναι η εντολή μετά το `else`.

A6.6 · Ο βρόχος while (1)

A6.6 · *Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 20* · ★☆☆ · short-answer · `slides-lec06-while-forever`

Τι κάνει το παρακάτω πρόγραμμα;

```
while (1);
```

Υπάρχει κάποια χρησιμότητα στο να έχουμε μια λογική συνθήκη που είναι πάντα αληθής;

Υπόδειξη Ποια είναι η τιμή αλήθειας του 1 στην C και ποιο είναι το σώμα αυτού του βρόχου; Για το δεύτερο ερώτημα, σκεφτείτε προγράμματα που δεν πρέπει να σταματούν ποτέ μόνα τους.

A6.7 · Άθροισμα με while

A6.7 · *Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 19* · ★☆☆ · trace · `slides-lec06-while-sum`

Τι κάνει το παρακάτω πρόγραμμα;

```
int i = 0;
int sum = 0;
while (i < 42) {
  sum += i;
  i++;
}
printf("%d %d\n", i, sum);
```

Υπόδειξη Ποια είναι η πρώτη τιμή του `i` για την οποία η συνθήκη είναι ψευδής; Ποιες τιμές έχουν προστεθεί στο `sum` μέχρι τότε; Ο τύπος για το άθροισμα $1 + 2 + \cdots + n$ θα σας γλιτώσει από τις πράξεις.

A6.8 · Το πρόβλημα του dangling else

A6.8 · *Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 15* · ★★☆ · trace · `slides-lec06-dangling-else`

Πολλές εμφωλευμένες εντολές `if` μπορεί να οδηγήσουν σε προβλήματα αμφισημίας (κάτι που δεν μας αρέσει καθόλου στο computer science). Είναι τα δύο προγράμματα ισοδύναμα;

```
lab = -1;
if (year < 1)
    { /* empty block */ }
else if (year == 1)
    lab = 20;
else
    lab = 0;
```

```
lab = -1;
if (year <= 1)
    if (year == 1)
        lab = 20;
else
    lab = 0;
```

Υπόδειξη Ο μεταγλωττιστής αγνοεί τη στοίχιση. Βρείτε σε ποια `if` ανήκει το `else` του δεύτερου προγράμματος σύμφωνα με τον κανόνα της C, ξαναγράψτε το με αγκύλες και συγκρίνετε τα δύο προγράμματα για `year` ίσο με 0, 1 και 2.

A6.9 · Το ερωτηματικό μετά το while

A6.9 · *Διάλεξη 6: Εντολές και Ροή Ελέγχου, διαφάνεια 21* · ★★☆ · trace · `slides-lec06-while-semicolon`

Τι κάνει το παρακάτω πρόγραμμα;

```
i = 0;
while (i < 42);
  printf("%d\n", i++);
```

Υπόδειξη Μην εμπιστεύεστε τη στοίχιση. Ποια ακριβώς εντολή είναι το σώμα του βρόχου; Αλλάζει ποτέ το `i` μέσα σε αυτό;

Εργαστήριο (A6.10–A6.13)

A6.10 · Αποσφαλμάτωση αθροίσματος με όριο

A6.10 · *Εργαστήριο 3, Άσκηση 4* · ★☆☆ · debug · `lab-lab03-limit`

Το παρακάτω πρόγραμμα `limit.c` ευελπιστεί να υπολογίζει την παράσταση 1 - 1/2 + 1/4 - 1/8 + 1/16 - ... για όσους όρους είναι μεγαλύτεροι από το `LIMIT`. Για να το πετύχει αυτό, ο προγραμματιστής του έκανε την εξής σκέψη:

> ”Σε μια μεταβλητή `c` θα κρατάω τον τρέχοντα όρο που θα προστεθεί, ο οποίος παράγεται από τον προηγούμενο αν πολλαπλασιάσω με ½. Θέλω ανά πάσα στιγμή ο όρος `c` να μην ξεπερνά το `LIMIT`. Το `c` έχει διαφορετικό πρόσημο σε κάθε όρο, οπότε θα χρησιμοποιήσω μια μεταβλητή `sign` που θα την πολλαπλασιάζω με το -1 ώστε ανά δύο όρους να έχει το ίδιο πρόσημο. Τελικά θα εκτυπώσω το άθροισμα `S`”.

Το πρόγραμμα αυτό όμως έχει λογικά λάθη. Μπορείτε να το αποσφαλματώσετε και να γράψετε ένα σωστό `limit.c`;

```c
#include <stdio.h>

#define LIMIT 0.008

int main() {
    float S = 0, c = 1;
    int sign = 1;

    while (c < LIMIT) {
        S = S + c;
        sign = (-1) * sign;
        c = sign * (1.0 / 2) * c;
    }

    printf("%f\n", S);
}
```

Υπόδειξη Βάλτε μια `printf` μέσα στον βρόχο που τυπώνει `S`, `c` και `sign` σε κάθε επανάληψη και δείτε πόσες φορές εκτελείται. Ελέγξτε τη συνθήκη σε σχέση με το τι σημαίνει «όροι μεγαλύτεροι από το `LIMIT`», και αναρωτηθείτε τι γίνεται στη σύγκριση όταν ο όρος γίνει αρνητικός. Επίσης, αφού το `c` κουβαλά ήδη το πρόσημο του προηγούμενου όρου, τι κάνει ο πολλαπλασιασμός με το `sign` που αλλάζει κάθε φορά;

A6.11 · Κατασκευή πυραμίδας

A6.11 · *Εργαστήριο 4, Άσκηση 2* · ★☆☆ · programming · `lab-lab04-pyramid`

2.1 Γράψτε ένα πρόγραμμα `pyramid.c` που να εκτυπώνει στην οθόνη σχήμα με την ακόλουθη μορφή χρησιμοποιώντας αποκλειστικά την συνάρτηση `putchar`:

```
*
**
***
****
*****
******
```

Το πλήθος των γραμμών που θα εκτυπωθούν να το διαβάζετε με χρήση `scanf`.

> `int putchar(int c)`
>
> Η συνάρτηση `putchar(int c)` εκτυπώνει τον χαρακτήρα που αντιστοιχεί στον ASCII κωδικό που δέχεται σαν όρισμα.

Ισοδύναμες χρήσεις: `putchar('*')` ή `putchar(42)`, όπου 42 είναι ο ASCII κωδικός του χαρακτήρα *.

2.2 Τροποποιήστε το `pyramid.c` ώστε να εκτυπώνει στην οθόνη σχήμα με την ακόλουθη μορφή:

```
*
***
*****
*******
*********
***********
```

Extra: τροποποιήστε το `pyramid.c` ώστε το βήμα με το οποίο αυξάνεται ο αριθμός των * ανά γραμμή να καθορίζεται επίσης από τον χρήστη με χρήση `scanf`.

Υπόδειξη Δύο εμφωλευμένοι βρόχοι: ο εξωτερικός μετράει γραμμές και ο εσωτερικός τυπώνει τα αστεράκια της τρέχουσας γραμμής, και μετά τον εσωτερικό τυπώνετε την αλλαγή γραμμής. Βρείτε έναν τύπο για το πλήθος των * στη γραμμή i (για το 2.2 και για το γενικό βήμα).

Α6.12 · Υπολογισμός ριζών τριωνύμου

A6.12 · *Εργαστήριο 3, Άσκηση 2* · ★★☆ · programming · `lab-lab03-root`

2.1 Παραγωγή τυχαίων αριθμών

Γράψτε ένα πρόγραμμα `root.c` που να παράγει τρεις τυχαίους πραγματικούς αριθμούς στο διάστημα `[0,1]` και να τους αποθηκεύει στις μεταβλητές τύπου `double` `a`, `b` και `c`, τις οποίες στη συνέχεια να εκτυπώνει.

`int rand()`

Απαιτείται ενσωμάτωση του αρχείου επικεφαλίδας `stdlib.h`.

Επιστρέφει έναν ψευδο-τυχαίο ακέραιο από 0 έως `RAND_MAX` - συμβολική σταθερά που ορίζεται στο αρχείο επικεφαλίδας `stdlib.h`.

`void srand(unsigned int seed)`

Απαιτείται ενσωμάτωση του αρχείου επικεφαλίδας `stdlib.h`.

Αρχικοποιεί την γεννήτρια ψευδο-τυχαίων αριθμών με βάση τον αριθμό/φύτρα `seed`. Συνήθως χρησιμοποιούμε σα φύτρα την τρέχουσα ώρα, όπως επιστρέφεται από την κλήση της συνάρτησης `time(NULL)` (απαιτείται το αρχείο επικεφαλίδας `time.h`).

2.2 Εύρεση ριζών

Τροποποιήστε το `root.c` ώστε να υπολογίζει τις πραγματικές ρίζες ενός τριωνύμου. Τα δεδομένα εισόδου (συντελεστές του τριωνύμου) θα είναι τυχαίες πραγματικές μεταβλητές `a`, `b`,

c στο διάστημα [0,1]. Αν το τριώνυμο δεν έχει πραγματικές ρίζες, να εκτυπώνεται ένα σχετικό μήνυμα ενημέρωσης. Οι ρίζες να εκτυπώνονται με ακρίβεια 3 δεκαδικών ψηφίων.

Θυμηθείτε τη δομή ελέγχου `if...else`:

```
if (C)
    E1;
else
    E2;
```

Αν η συνθήκη `C` είναι αληθής εκτελείται η εντολή (ή το μπλοκ εντολών) `E1`, αλλιώς η εντολή (ή το μπλοκ εντολών) `E2`. Το σκέλος `else` είναι προαιρετικό: αν λείπει και η `C` είναι ψευδής, δεν εκτελείται τίποτα.

2.3 Μιγαδικές ρίζες

Ενσωματώστε στο πρόγραμμά σας και την περίπτωση το τριώνυμο να έχει μιγαδικές ρίζες, οπότε θα πρέπει να τις υπολογίζει.

Υπόδειξη Για τυχαίο πραγματικό στο [0,1] διαιρέστε το αποτέλεσμα της `rand()` με το `RAND_MAX`, προσέχοντας να γίνει η διαίρεση σε `double`. Οι περιπτώσεις καθορίζονται από το πρόσημο της διακρίνουσας. Για τις μιγαδικές ρίζες τυπώστε χωριστά πραγματικό και φανταστικό μέρος, χρησιμοποιώντας την απόλυτη τιμή της διακρίνουσας.

Α6.13 · Υπολογισμός αθροίσματος σειράς

A6.13 · *Εργαστήριο 3, Άσκηση 1* · ★★☆ · programming · `lab-lab03-seq`

Σε αυτήν την άσκηση, θα γράψουμε ένα πρόγραμμα `seq.c` το οποίο θα υπολογίζει διάφορα αθροίσματα σειράς. Για κάθε ένα από αυτά, θα φτιάχνουμε μια διαφορετική συνάρτηση υπολογισμού.

1.1 Γράψτε μια συνάρτηση `sum` που να υπολογίζει το άθροισμα της σειράς:

$$\sum_{i=1}^{100} i$$

και να το εκτυπώνει στην οθόνη. Να χρησιμοποιηθεί η δομή επανάληψης `while`.

1.2 Γράψτε μια συνάρτηση `basel` που να υπολογίζει το άθροισμα της σειράς:

$$\sum_{i=1}^{100} \frac{1}{i^2}$$

και να το εκτυπώνει στην οθόνη. Να χρησιμοποιηθεί η δομή επανάληψης `for`.

Τι συμβαίνει αν αυξήσετε τους όρους; Ρίξτε μια ματιά στο Basel Theorem και ελέγξτε αν τα αποτελέσματά σας συμφωνούν με τα μαθηματικά.

> Υπόδειξη: Χρησιμοποιήστε μία ενδιάμεση μεταβλητή για να αποθηκεύετε προσωρινά τον τρέχοντα όρο της σειράς.

1.3 Γράψτε μια συνάρτηση `pi_approx` που να προσεγγίζει το π χρησιμοποιώντας την έκφραση:

$$\pi = \sqrt{6 \cdot \sum_{i=1}^{\infty} \frac{1}{i^2}}$$

χρησιμοποιώντας τους 100 πρώτους όρους της σειράς και να τυπώνει το αποτέλεσμα. Να χρησιμοποιηθεί η δομή επανάληψης `do...while`. Για την υλοποίηση θα χρειαστείτε την συνάρτηση `sqrt` (square root).

> `double sqrt(double)`
>
> Απαιτείται η ενσωμάτωση του αρχείου επικεφαλίδας `math.h`.
>
> Επιστρέφει την τετραγωνική ρίζα του αριθμού που δέχεται σαν όρισμα.
>
> Στην μεταγλώττιση με τον `gcc` απαιτείται και η επιλογή `-lm` για να συμπεριληφθεί η μαθηματική βιβλιοθήκη.

1.3.1 Τροποποίηση της `pi_approx` μέχρι όριο ακρίβειας 10^{-15}.

Παρατηρήστε ότι οι όροι που προστίθενται στην σειρά γίνονται ολοένα και πιο μικροί, με αποτέλεσμα από ένα σημείο και μετά να είναι αρκούντως μικροί για να μην επηρεάζουν το αποτέλεσμα, αφού χρησιμοποιούμε αριθμητική πεπερασμένης ακρίβειας. Αλλάξτε το κριτήριο τερματισμού του υπολογισμού, ώστε ο υπολογισμός να σταματάει όταν ο τρέχων όρος που προστίθεται στην σειρά είναι μικρότερος από 10^{-15}.

1.3.2 Τροποποίηση της `pi_approx` ώστε να τυπώνει το αποτέλεσμα με ακρίβεια 8 δεκαδικών ψηφίων.

Η συνάρτηση `printf()` μπορεί να εκτυπώσει την τιμή πραγματικών μεταβλητών που δέχεται σαν όρισμα με επιθυμητή μορφοποίηση. Σύνταξη:

```
printf("%a.bf", var);
```

όπου:

- `a`: Ορίζουμε το πλήθος των θέσεων που θα γίνει η εκτύπωση.
- `b`: Ορίζουμε το πλήθος των ψηφίων μετά την υποδιαστολή που θα εκτυπωθούν.

Μέχρι πόσα ψηφία του π μπορείτε να υπολογίσετε / τυπώσετε; Αυτό το πρόβλημα έχει τυραννήσει γενιές και γενιές στα μαθηματικά και αργότερα - όταν οι αριθμοί έγιναν μεγάλοι - στην πληροφορική [1] [2].

1.4 Υλοποιήστε μια συνάρτηση `eta_two` η οποία να υπολογίζει το άθροισμα της σειράς:

$$S_1 = \frac{1}{1^2} - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \frac{1}{5^2} - \frac{1}{6^2} + \dots$$

ή αλλιώς γραμμένη ως:

$$S_1 = \sum_{i=1}^{\infty} \frac{(-1)^{i-1}}{i^2}$$

Η συνάρτησή σας θέλουμε να ζητάει από τον χρήστη το πλήθος των όρων που θα χρησιμοποιήσει και να τυπώνει το αποτέλεσμα με ακρίβεια 6 δεκαδικών ψηφίων. Μπορείτε να χρησιμοποιήσετε όποια δομή επανάληψης επιθυμείτε.

Φαίνεται το αποτέλεσμά σας να συγκλίνει σε κάποιον αριθμό; Μπορείτε να το εξηγήσετε με την βοήθεια της συνάρτησης Dirichlet eta;

1.5 Εξάσκηση σε σειρές

Μπορείτε να κάνετε περισσότερη εξάσκηση, υλοποιώντας τις ακόλουθες συναρτήσεις στο πρόγραμμα `seq.c`:

- Συνάρτηση `harmonic` [3] που υπολογίζει το άθροισμα της εναλλάσσουσας αρμονικής σειράς:

$$S_2 = \sum_{i=1}^{\infty} \frac{(-1)^{i+1}}{i} = \frac{1}{1} - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots$$

- Συνάρτηση `leibniz` [4] που να υπολογίζει το άθροισμα:

$$S_3 = \sum_{i=0}^{\infty} \frac{(-1)^{i}}{2i+1} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots$$

- Συνάρτηση `zeta_four` [5] που να υπολογίζει το άθροισμα:

$$S_4 = \sum_{i=1}^{\infty} \frac{1}{i^4} = \frac{1}{1^4} + \frac{1}{2^4} + \frac{1}{3^4} + \frac{1}{4^4} + \cdots$$

- Συνάρτηση `wallis` [6] που να υπολογίζει το γινόμενο:

$$P = \prod_{i=1}^{\infty} \frac{2i}{2i-1} \cdot \frac{2i}{2i+1} = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdots$$

- Συνάρτηση `ramanujan` [7] που να υπολογίζει το π χρησιμοποιώντας την σχέση:

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)! \cdot (1103 + 26390k)}{(k!)^4 \cdot 396^{4k}}$$

Υπόδειξη Κάθε σειρά είναι ένας βρόχος με έναν συσσωρευτή (αρχικοποιημένο σε 0 για άθροισμα, σε 1 για γινόμενο) και έναν τρέχοντα όρο. Προσοχή στην ακέραια διαίρεση: το `1/(i*i)` με ακέραιο `i` κάνει 0, οπότε ο υπολογισμός πρέπει να γίνεται σε `double`. Για τις εναλλασσόμενες σειρές κρατήστε ένα πρόσημο που αλλάζει σε κάθε βήμα αντί να καλείτε pow, και για το 1.3.1 ο έλεγχος τερματισμού συγκρίνει τον τρέχοντα όρο με το όριο.

Εργασίες (A6.14–A6.15)

A6.14 · Η εικασία Collatz

A6.14 · *Εργασία 0 (2023-24), Άσκηση 3* · ★★☆ · programming · `hw-2023-hw0-collatz`

Η εικασία Collatz είναι ένα από τα πιο διάσημα άλυτα προβλήματα των μαθηματικών. Η εικασία ισχυρίζεται ότι η επανάληψη δυο απλών αριθμητικών πράξεων με μια συγκεκριμένη διαδικασία μπορεί να μετατρέψει οποιονδήποτε θετικό ακέραιο στο 1. Η διαδικασία των πράξεων έχει ως εξής:

1. Διάλεξε οποιονδήποτε θετικό ακέραιο N.
2. Αν ο αριθμός είναι 1, τότε τερμάτισε την διαδικασία.
3. Αν είναι άρτιος, ο επόμενος αριθμός στην ακολουθία θα είναι ο N/2.
4. Αν είναι περιττός, ο επόμενος αριθμός στην ακολουθία θα είναι ο 3 × N + 1.
5. Επανάλαβε τα βήματα 2-4 μέχρι να φτάσουμε στο 1.

Για παράδειγμα, έστω N = 3. Η παραπάνω διαδικασία θα παράξει την ακολουθία: 3, 10, 5, 16, 8, 4, 2, 1. Για το 42 παίρνουμε την ακολουθία: 42, 21, 64, 32, 16, 8, 4, 2, 1.

Για κάθε θετικό ακέραιο N, ο αριθμός στοιχείων της ακολουθίας που παράγεται μέχρι να καταλήξουμε στο 1, λέγεται *μήκος ακολουθίας Collatz*. Για παράδειγμα, για N = 3, το μήκος της ακολουθίας είναι 8 (η ακολουθία έχει 8 στοιχεία: 3, 10, 5, 16, 8, 4, 2, 1). Αντίστοιχα για τον αριθμό 42, το μήκος ακολουθίας Collatz είναι 9. Αν N = 1, τότε έχουμε το ελάχιστο μήκος 1.

Για το ζητούμενο αυτής της άσκησης, καλείστε να γράψετε ένα πρόγραμμα που βρίσκει το μέγιστο μήκος ακολουθίας Collatz σε ένα εύρος αριθμών.

Τεχνικές Προδιαγραφές

- Repository Name: `progintro/hw0-<YourUsername>`
- Αρχείο C (Filepath): `collatz/src/collatz.c`
- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας

πρέπει να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr): `gcc -O0 -m32 -Wall -Wextra -Werror -pedantic -o collatz collatz.c`

- README Filepath: `collatz/README.md`
- **Ένα αρχείο που να περιέχει ένα στοιχείο εισόδου και ένα εξόδου *διαφορετικά* από αυτά της άσκησης. Συγκεκριμένα προτείνουμε να βάλετε έναν συνδυασμό που θεωρείται ότι είναι δύσκολος να γίνει σωστός κατά την υλοποίηση.**
  - input Filepath: `collatz/test/input`
  - output Filepath: `collatz/test/output`

  Για παράδειγμα, το περιεχόμενο του input αρχείου μπορεί να είναι: ”100 100000000” και του αντίστοιχου output: ”950”. Προσοχή: αυτό το παράδειγμα δεν θα γίνει δεκτό από την άσκηση επειδή αυτό το input-output ζευγάρι υπάρχει ήδη παραπάνω, επομένως πρέπει να διαλέξετε κάποιο άλλο.
- Όλοι οι ακέραιοι που θα δοθούν στο πρόγραμμά σας θα είναι στο εύρος [−100000000, 100000000]
- Το ελάχιστο κάτω όριο που είναι δεκτό: 1.
- Το μέγιστο άνω όριο που είναι δεκτό: 100.000.000.
- Αν δοθεί οποιοσδήποτε μη θετικός αριθμός στα όρια η είσοδος θεωρείται μη αποδεκτή το πρόγραμμα πρέπει να τυπώνει: ”0”.
- Πρέπει να ολοκληρώνει την εκτέλεση μέσα σε: 4 λεπτά.
- Μέγιστο μέγεθος αρχείου C που μπορεί να χρησιμοποιηθεί στην υποβολή: 8KB.

Παρακάτω παραθέτουμε την αλληλεπίδραση με μια ενδεικτική λύση:

```
thanassis@linux14:~$ hostname
linux14
thanassis@linux14:~$ gcc -O0 -m32 -Wall -Wextra -Werror -pedantic -o collatz
↪  collatz.c
thanassis@linux14:~$ ./collatz 1 10
20
thanassis@linux14:~$ ./collatz 900 1000
174
thanassis@linux14:~$ ./collatz 80000 100000
333
thanassis@linux14:~$ ./collatz 100 1000000
525
thanassis@linux14:~$ ./collatz 100 100000000
950
thanassis@linux14:~$ ./collatz -1 100
```

```
0
```

Ενδεικτικοί χρόνοι:

```
thanassis@linux14:~$ time ./collatz 100 1000000
525

real 0m1,343s
user 0m1,338s
sys 0m0,004s
thanassis@linux14:~$ time ./collatz 100 100000000
950

real 3m0,212s
user 3m0,206s
sys 0m0,000s
```

Ο παραπάνω χρόνος μπορεί να είναι και πιο γρήγορος ανάλογα με την υλοποίηση / μεταγλώττιση / μηχάνημα στο οποίο τρέχει. Αν επιθυμείτε να τα βελτιώσετε ίσως χρειαστεί λίγο παραπάνω έρευνα από μέρους σας σε θέματα για τα οποία δεν έχουμε μιλήσει ακόμα στο μάθημα, αλλά φυσικά πιο γρήγορες υποβολές είναι ευπρόσδεκτες:

```
thanassis@linux14:~$ time ./collatz_fast 100 100000000
950

real 0m8,034s
user 0m6,789s
sys 0m1,244s
```

Στο αρχείο `README.md` πρέπει να προσθέσετε οποιεσδήποτε παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης. Για την υποβολή με την καλύτερη απόδοση (πιο γρήγορη από πλευράς χρόνου) θα ζητήσουμε μια γρήγορη (5λεπτη) παρουσίαση στο μάθημα, για την οποία θα λάβει +100% της βαθμολογίας (+50 Μονάδες).

Υπόδειξη Γράψτε πρώτα μια συνάρτηση που υπολογίζει το μήκος για ένα N και μετά έναν βρόχο πάνω στο εύρος που κρατά το μέγιστο· τα όρια έρχονται από το `argv` με `atoi`. Προσέξτε ότι με `-m32` ο `int` είναι 32 bit, ενώ οι ενδιάμεσοι όροι της ακολουθίας για N κοντά στο `10^8` ξεπερνούν κατά πολύ το `2^32`: χρειάζεστε έναν ευρύτερο ακέραιο τύπο. Σκεφτείτε επίσης τι γίνεται αν το κάτω όριο είναι μεγαλύτερο από το άνω.

Δείτε επίσης

- Υποδειγματική υποβολή φοιτητή (περιέχει λύση: δείτε την αφού λύσετε την άσκηση)

A6.15 · Οι Ακολουθίες Aliquot

A6.15 · *Εργασία 0 (2025-26), Άσκηση 3* · ★★☆ · programming · `hw-2025-hw0-aliquot`

Στα μαθηματικά, η ακολουθία aliquot είναι μια ακολουθία θετικών ακεραίων στην οποία ο κάθε όρος είναι το άθροισμα των γνησίων διαιρετών του προηγουμένου όρου (γνήσιοι διαιρέτες ενός αριθμού n λέγονται όλοι οι ακέραιοι διαιρέτες του n πλην του εαυτού του). Αν η ακολουθία φτάσει στον αριθμό 1, τερματίζει, εφόσον το άθροισμα των γνησίων διαιρετών του 1 είναι 0. Συμβολίζοντας με $s(n)$ το άθροισμα των γνήσιων διαιρετών του n, για $n = 12$ έχουμε:

$$s(12) = 1 + 2 + 3 + 4 + 6 = 16$$

και συνεχίζοντας με τον ίδιο τρόπο $s(16) = 15$, $s(15) = 9$, $s(9) = 4$, $s(4) = 3$, $s(3) = 1$, $s(1) = 0$. Η ακολουθία τελικά φτάνει στο 0 μετά από 7 βήματα ($12 \to 16 \to 15 \to 9 \to 4 \to 3 \to 1 \to 0$). Ο αριθμός των βημάτων (ουσιαστικά ο αριθμός των μεταβάσεων $\to$) που απαιτούνται για να μεταβεί ένας ακέραιος n στο 0 λέγεται **μήκος aliquot** για το δοθέν n. Με τον συμβολισμό $s^m(n) = s(s(\cdots s(n)\cdots))$ (m φορές) γράφουμε $s^2(12) = 15$ ή $s^7(12) = 0$· ισοδύναμα, το μήκος aliquot l είναι ο μικρότερος ακέραιος l για τον οποίο $s^l(n) = 0$.

Δεν είναι όλες οι ακολουθίες aliquot φθίνουσες. Για τους τέλειους αριθμούς όπως το 6 ($s(6) = 1 + 2 + 3 = 6$) η ακολουθία δεν τερματίζει ποτέ, καθώς $s^m(6) = 6$ για κάθε m. Υπάρχουν και οι φίλιοι αριθμοί όπως οι 220 και 284 ($220 \to 284 \to 220 \to \cdots$) και οι κοινωνικοί αριθμοί με κύκλους μεγαλύτερης περιοδικότητας. Η εικασία Catalan-Dickson (1888) λέει ότι κάθε ακολουθία είτε φτάνει στο 0 είτε πέφτει σε κύκλο, και ακόμα δεν έχει αποδειχθεί. Για κάποιους σχετικά μικρούς αριθμούς, όπως ο 276, δεν έχουμε καταφέρει ακόμα να υπολογίσουμε την πλήρη ακολουθία!

Θα χρειαστεί να υλοποιήσετε ένα πρόγραμμα το οποίο να μπορεί να υπολογίζει τους όρους και τα μήκη της ακολουθίας aliquot για τους αριθμούς που επιλέγει ο χρήστης, με τους περιορισμούς των τεχνικών προδιαγραφών που ακολουθούν.

Τεχνικές Προδιαγραφές

- Repository Name: `progintro/hw0-<YourUsername>`
- Αρχείο C (Filepath): `aliquot/src/aliquot.c`
- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr): `gcc -O0 -m32 -Wall -Wextra -Werror -pedantic -o aliquot aliquot.c`
- README Filepath: `aliquot/README.md`
- Το πρόγραμμά σας πρέπει να διαβάζει από την πρότυπη είσοδο (stdin) 3 διαφορετικές τιμές:
  1. Τον θετικό ακέραιο από τον οποίο θέλετε να ξεκινήσετε την ακολουθία.

2. Το μέγιστο μήκος της ακολουθίας (το 0 θα συμβολίζει ότι επιτρέπεται ακολουθίες "άπειρου" μήκους) που θέλουμε να υπολογιστεί.
3. Έναν χαρακτήρα που να δηλώνει αν θέλετε το πρόγραμμά σας να τυπώσει την πλήρη ακολουθία `'f'` (full) ή να τυπώσει μόνο το μήκος της `'l'` (length).

- Δείτε παραδείγματα από τις πρότυπες εκτελέσεις παρακάτω για να δείτε την αναμενόμενη έξοδο.
- Όλοι οι ακέραιοι που θα δοθούν στο πρόγραμμά σας θα είναι ακέραιοι μικρότεροι από 10^{15}. Αν κατά την διαδικασία του υπολογισμού της ακολουθίας το πρόγραμμά σας φτάσει σε ακέραιο μεγαλύτερο του 10^{15}, το πρόγραμμά σας πρέπει να τερματίσει με μήνυμα λάθους και κωδικό εξόδου (exit code) 1.
- Αν δοθεί οποιαδήποτε είσοδος εκτός προδιαγραφών (για παράδειγμα ο χρήστης δεν δώσει κάποιον ακέραιο για να υπολογιστεί η ακολουθία) το πρόγραμμά σας πρέπει να τερματίζει με κωδικό εξόδου (exit code) 1.
- Για μια ακολουθία μήκους n, οι χρήστες αναμένουν πως το πρόγραμμά σας πρέπει να ολοκληρώνει την εκτέλεσή του μέσα σε λιγότερο από n δευτερόλεπτα.

Παρακάτω παραθέτουμε αλληλεπιδράσεις με μια ενδεικτική λύση:

```
$ hostname
linux14
$ gcc -O0 -m32 -Wall -Wextra -Werror -pedantic -o aliquot aliquot.c
$ ./aliquot
Please give the number to start the aliquot sequence from: 12
Provide the max aliquot length to look for (0 for unlimited): 0
Do you want to print the full sequence ('f') or just the length ('l')? f
12
16
15
9
4
3
1
0
$ ./aliquot
Please give the number to start the aliquot sequence from: 12
Provide the max aliquot length to look for (0 for unlimited): 0
Do you want to print the full sequence ('f') or just the length ('l')? l
Length of aliquot sequence: 7
$ echo $?
0
$ ./aliquot
Please give the number to start the aliquot sequence from: 138
Provide the max aliquot length to look for (0 for unlimited): 200
Do you want to print the full sequence ('f') or just the length ('l')? l
```

```
Length of aliquot sequence: 178
$ ./aliquot
Please give the number to start the aliquot sequence from: 6
Provide the max aliquot length to look for (0 for unlimited): 6
Do you want to print the full sequence ('f') or just the length ('l')? f
6
6
6
6
6
6
6
$ ./aliquot
Please give the number to start the aliquot sequence from: 276
Provide the max aliquot length to look for (0 for unlimited): 0
Do you want to print the full sequence ('f') or just the length ('l')? f
276
396
...
749365894850244
1414070378301756
Number exceeds maximum supported integer (1000000000000000). Stopping.
$ echo $?
1
```

Στο αρχείο `README.md` πρέπει να προσθέσετε οποιεσδήποτε παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.

Υπόδειξη Με `-m32` ο `long` έχει 32 bits, οπότε τιμές μέχρι 10^{15} θέλουν `long long` (και `%lld` στο `scanf`/`printf`). Για το $s(n)$ μη δοκιμάσεις όλους τους διαιρέτες μέχρι το n: οι διαιρέτες έρχονται σε ζεύγη d και n/d, άρα αρκεί να φτάσεις ως το $\sqrt{n}$, προσέχοντας τα τέλεια τετράγωνα και ότι ο ίδιος ο n δεν μετράει. Έλεγξε την τιμή επιστροφής κάθε `scanf` και σκέψου τις ακραίες περιπτώσεις: $n = 1$, μέγιστο μήκος 0, χαρακτήρας διαφορετικός από `f`/`l`.

Θέματα εξετάσεων (A6.16–A6.25)

A6.16 · Μέγιστος Κοινός Διαιρέτης

A6.16 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #6 (Crypto Themed), Θέμα 1* · ★☆☆ · programming · `exam-2023-fall-ex6-q1`

Πρόγραμμα: `gcd.c` (25 μονάδες)

Μέγιστος κοινός διαιρέτης ονομάζεται ο μεγαλύτερος ακέραιος που διαιρεί δύο ή περισσότερους ακέραιους αριθμούς. Για παράδειγμα ο μέγιστος κοινός διαιρέτης του 42 και του 27 είναι το 3 (καθώς διαιρεί ακριβώς και το 42/3=14 και το 27/3=9). Γράψτε ένα πρόγραμμα που διαβάζει δύο θετικούς ακεραίους σε δεκαδική μορφή από την γραμμή εντολών και τυπώνει τον μέγιστο κοινό διαιρέτη. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o gcd gcd.c
$ ./gcd 42 27
Greatest common divisor of 42 and 27 is 3
$ ./gcd 42 -27
Numbers must be positive
$ ./gcd 54 24
Greatest common divisor of 54 and 24 is 6
$ ./gcd 9827345 9182767
Greatest common divisor of 9827345 and 9182767 is 11
$ ./gcd 782936492674398272 982137984792892
Greatest common divisor of 782936492674398272 and 982137984792892 is 4
$ ./gcd 260056890954482 23809214813964
Greatest common divisor of 260056890954482 and 23809214813964 is 2982734
```

Υπόδειξη Ο αλγόριθμος του Ευκλείδη (με υπόλοιπα, όχι με διαδοχικές αφαιρέσεις) τελειώνει σε λίγα βήματα ακόμα και για πολύ μεγάλους αριθμούς, ενώ η δοκιμή όλων των πιθανών διαιρετών όχι. Τα παραδείγματα χρειάζονται 64-bit ακέραιους: διαβάστε τα ορίσματα με `strtoll`/`strtoull` και ελέγξτε ότι είναι έγκυροι θετικοί αριθμοί που χωράνε στον τύπο.

Α6.17 · Προσεγγίζοντας το π

A6.17 · *Κατατακτήριες Δεκεμβρίου 2024, Θέμα 1* · ★☆☆ · programming · `exam-2024-dec-q1`

Ο Άγγλος μαθηματικός John Wallis θεωρείται πως είναι ένας από τους πρώτους που εισήγαγε την έννοια του απείρου (∞) στα μαθηματικά. Το 1656 δημοσίευσε ένα αποτέλεσμα σύμφωνα με το οποίο μπορούμε να προσεγγίσουμε το π χρησιμοποιώντας το ακόλουθο απειρογινόμενο όρων t_i, γνωστό και ως γινόμενο Wallis προς τιμήν του:

$$\frac{\pi}{2} = \underbrace{\frac{2}{1}\cdot\frac{2}{3}}_{t_1}\cdot\underbrace{\frac{4}{3}\cdot\frac{4}{5}}_{t_2}\cdot\underbrace{\frac{6}{5}\cdot\frac{6}{7}}_{t_3}\cdot\underbrace{\frac{8}{7}\cdot\frac{8}{9}}_{t_4}\cdots$$

Χρησιμοποιώντας την παραπάνω σχέση μπορούμε να προσεγγίσουμε και εμείς το π όπως ο Wallis. Μάλιστα, όσους περισσότερους όρους t_i χρησιμοποιούμε, τόσο καλύτερη η προσέγγισή μας. Γράψτε ένα πρόγραμμα C το οποίο παίρνει έναν θετικό ακέραιο ως όρισμα από την γραμμή εντολών που αντιπροσωπεύει πόσους πρώτους όρους t_i να χρησιμοποιήσει στον υπολογισμό

του π και στην συνέχεια εκτυπώνει την προσέγγιση του π με 8 δεκαδικά ψηφία. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ ./wallis
Usage: ./wallis <number of terms to use>
$ echo $?
1
$ ./wallis 1
Using the first 1 terms, pi = 2.66666667
$ ./wallis 2
Using the first 2 terms, pi = 2.84444444
$ ./wallis 4
Using the first 4 terms, pi = 2.97215420
$ ./wallis 100
Using the first 100 terms, pi = 3.13378749
$ ./wallis 100000000
Using the first 100000000 terms, pi = 3.14159264
```

Υπόδειξη Γράψτε τον γενικό όρο t_i ως συνάρτηση του i και πολλαπλασιάστε τους σε έναν βρόχο, με μεταβλητή `double`: προσέξτε ότι οι διαιρέσεις πρέπει να γίνονται σε κινητή υποδιαστολή και ότι για i ως 10^8 το γινόμενο $2i \cdot 2i$ χρειάζεται τύπο αρκετά μεγάλο. Μην ξεχάσετε ότι το γινόμενο δίνει $\pi/2$, και ελέγξτε το `argc` ώστε να τυπώνετε το μήνυμα χρήσης και να επιστρέφετε κωδικό εξόδου 1.

A6.18 · Mystery

A6.18 · *Εξέταση Σεπτεμβρίου 2025, Θέμα 1* · ★☆☆ · trace · `exam-2025-sep-q1`

Mystery [10 Μονάδες]

Η συνάρτηση `mystery` δέχεται ως όρισμα έναν ακέραιο αριθμό που αποτελείται από τα 3 τελευταία ψηφία του sdi σας. Για παράδειγμα, αν ο sdi σας είναι ο sdi2400789 τότε καλούμε την συνάρτηση ως `mystery(789)`. Τι θα τυπώσει η συνάρτηση για τα ψηφία του δικού σας sdi; Αν δεν έχετε sdi, επιλέξτε έναν τυχαίο τριψήφιο. Αιτιολογήστε όπου νομίζετε πως χρειάζεται.

```c
int mystery(int number) {
  int s = 0, rem;
  printf("%s %d %d\n", "Starting loop", s, rem);
  do {
    rem = number % 10;
    printf("Inner: %d\n", rem);
    number /= 10;
    s += rem;
  } while (number);
```

```
  printf("Loop ended %d\n", s);
  return s;
}
```

Υπόδειξη Κρατήστε έναν πίνακα με τις τιμές των `number`, `rem` και `s` σε κάθε επανάληψη και θυμηθείτε ότι το σώμα της `do-while` εκτελείται τουλάχιστον μία φορά. Προσέξτε ιδιαίτερα την πρώτη `printf`: ποια τιμή έχει το `rem` εκείνη τη στιγμή; Σκεφτείτε επίσης τι γίνεται αν το τριψήφιό σας ξεκινά με 0 (π.χ. 042).

A6.19 · Mystery

A6.19 · *Εξέταση Σεπτεμβρίου 2026, Θέμα 1* · ★☆☆ · trace · `exam-2026-sep-q1`

Mystery [10 Μονάδες]

Η συνάρτηση `mystery` δέχεται ως όρισμα έναν ακέραιο αριθμό που αποτελείται από τα 3 τελευταία ψηφία του sdi σας. Για παράδειγμα, αν ο sdi σας είναι ο sdi2500789 τότε καλούμε την συνάρτηση ως `mystery(789)`. Τι θα τυπώσει η συνάρτηση για τα ψηφία του δικού σας sdi; Αιτιολογήστε όπου νομίζετε πως χρειάζεται.

```
int mystery(int number) {
  int i, total = 0;
  number += 100;
  printf("%s: %d, (%d)\n", "Computing total", total, i);
  for(i = 0 ; number ; i++) {
    total = 10 * total + number % 10;
    number /= 10;
    printf("%d: total: %d\n", i, total);
  }
  printf("%d: total: %d number: %d\n", i, total, number);
  return total;
}
```

Υπόδειξη Φτιάξτε έναν πίνακα με τις τιμές των `i`, `number` και `total` σε κάθε επανάληψη, ξεκινώντας από το `number` *αφού* προστεθεί το 100. Η συνθήκη του `for` είναι απλώς `number`: σκεφτείτε πότε γίνεται ψευδής. Προσέξτε την πρώτη `printf`: τι τιμή έχει το `i` εκείνη τη στιγμή; Και σκεφτείτε τι γίνεται με τα μηδενικά όταν τα ψηφία μπαίνουν στο `total`, π.χ. αν το άθροισμα είναι τετραψήφιο ή τελειώνει σε 0.

A6.20 · Κάντο όπως ο Βιετά

A6.20 · *Κατατακτήριες Δεκεμβρίου 2023, Θέμα 1* · ★★☆ · programming · `exam-2023-dec-q1`

Ο Γάλλος μαθηματικός Βιετά (François Viète) υπήρξε ο πρώτος μαθηματικός που χρησιμοποίησε ευρέως σύμβολα για να εκφράσει αριθμητικές ποσότητες. Το 1593 κατάφερε να εκφράσει και να υπολογίσει τον αριθμό π με ακρίβεια 9 δεκαδικών, βελτιώνοντας έτσι το σχετικό αποτέλεσμα του Αρχιμήδη. Για να υπολογίσει το π, ο Βιετά χρησιμοποίησε μια σχέση που χρησιμοποιεί ένα απειρογινόμενο όρων \$, γνωστό ως φόρμουλα Βιετά προς τιμήν του:

$$\frac{2}{\pi} = \underbrace{\frac{\sqrt{2}}{2}}_{t_1} \cdot \underbrace{\frac{\sqrt{2+\sqrt{2}}}{2}}_{t_2} \cdot \underbrace{\frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2}}_{t_3} \cdots$$

Λύνοντας την παραπάνω σχέση ως προς π, μπορούμε και εμείς να προσεγγίσουμε το π όπως ο Βιετά. Μάλιστα, όσους περισσότερους όρους \$ χρησιμοποιούμε, τόσο καλύτερη η προσέγγισή μας. Γράψτε ένα πρόγραμμα C το οποίο παίρνει έναν θετικό ακέραιο ως όρισμα από την γραμμή εντολών που αντιπροσωπεύει πόσους πρώτους όρους \$ να χρησιμοποιήσει στον υπολογισμό του π και στην συνέχεια εκτυπώνει την προσέγγιση του π με 9 δεκαδικά ψηφία. Παραδείγματα εκτέλεσης είναι τα εξής:

```
$ ./vieta 2
Multiplied first 2 ti terms, pi = 3.061467459
$ ./vieta 5
Multiplied first 5 ti terms, pi = 3.140331157
$ ./vieta 50
Multiplied first 50 ti terms, pi = 3.141592654
```

Υπόδειξη Παρατηρήστε ότι ο αριθμητής κάθε όρου προκύπτει από τον αριθμητή του προηγούμενου: αν `= \sqrt{2}`, $\tau\tau\varepsilon i + 1 = \sqrt{2+a_i}$. Κρατήστε λοιπόν σε έναν βρόχο τον τρέχοντα αριθμητή και το γινόμενο σε μεταβλητές `double` (με τη `sqrt` του `<math.h>`, μεταγλώττιση με `-lm`) και στο τέλος λύστε ως προς π. Μετατρέψτε το `argv[1]` σε ακέραιο και ελέγξτε ότι δόθηκε και είναι θετικός.

Α6.21 · Ασημένιο Κλάσμα

A6.21 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #10, Θέμα 2* · ★★☆ · programming · `exam-2023-fall-ex10-q2`

Πρόγραμμα: `silver.c`

Ένας τρόπος να προσεγγίσουμε την τιμή του $\sqrt{2}$ είναι μέσω της παράστασης:

$$1 + \sqrt{2} = 2 + \frac{1}{2 + \frac{1}{2 + \frac{1}{2 + \cdots}}}$$

Όπου η προσθήκη του κάθε όρου $\frac{1}{2+\cdots}$ βελτιώνει την προσέγγιση. Γράψτε ένα πρόγραμμα το οποίο παίρνει ως όρισμα έναν θετικό ακέραιο που αναπαριστά τον αριθμό των όρων $\frac{1}{2+\cdots}$ που θέλουμε να χρησιμοποιηθούν και τυπώνει την προσέγγιση του $\sqrt{2}$ με 20 δεκαδικά ψηφία ακριβείας. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ ./silver 1
sqrt(2) = 1.00000000000000000000
$ ./silver 2
sqrt(2) = 1.50000000000000000000
$ ./silver 3
sqrt(2) = 1.39999999999999991118
$ ./silver 4
sqrt(2) = 1.41666666666666651864
$ ./silver 5
sqrt(2) = 1.41379310344827580082
$ ./silver 1000
sqrt(2) = 1.41421356237309492343
```

Για έναν όρο η δεξιά παράσταση είναι (2 + 0) -> sqrt(2) = 1, για 2 όρους η δεξιά παράσταση είναι (2 + 1 / 2) -> sqrt(2) = 1.5, κοκ.

Υπόδειξη Το συνεχές κλάσμα υπολογίζεται εύκολα από μέσα προς τα έξω: ξεκινήστε από τον πιο εσωτερικό όρο και επαναλάβετε με έναν βρόχο. Προσέξτε πώς μετράνε οι όροι στα παραδείγματα (1 όρος δίνει 1) και ότι η παράσταση δίνει $1 + \sqrt{2}$, όχι $\sqrt{2}$. Ελέγξτε ότι το όρισμα είναι θετικός ακέραιος.

A6.22 · Κάντο όπως ο Βιετά

A6.22 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #13, Θέμα 1* · ★★☆ · programming · `exam-2023-fall-ex13-q1`

Πρόγραμμα: `vieta.c`

Ο Γάλλος μαθηματικός Βιετά (Francois Viete) υπήρξε ο πρώτος μαθηματικός που χρησιμοποίησε ευρέως σύμβολα για να εκφράσει αριθμητικές ποσότητες. Το 1593 κατάφερε να εκφράσει και να υπολογίσει τον αριθμό π με ακρίβεια 9 δεκαδικών, βελτιώνοντας έτσι το σχετικό αποτέλεσμα του Αρχιμήδη. Για να υπολογίσει το π, ο Βιετά χρησιμοποίησε μια σχέση που χρησιμοποιεί ένα απειρογινόμενο όρων t_i, γνωστό ως φόρμουλα Βιετά προς τιμήν του:

$$\frac{2}{\pi} = \underbrace{\frac{\sqrt{2}}{2}}_{t_1} \cdot \underbrace{\frac{\sqrt{2+\sqrt{2}}}{2}}_{t_2} \cdot \underbrace{\frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2}}_{t_3} \cdots$$

Λύνοντας την παραπάνω σχέση ως προς π, μπορούμε και εμείς να προσεγγίσουμε το π όπως ο Βιετά. Μάλιστα, όσους περισσότερους όρους t_i χρησιμοποιούμε, τόσο καλύτερη η προσέγγισή μας. Γράψτε ένα πρόγραμμα C το οποίο παίρνει έναν θετικό ακέραιο ως όρισμα από την γραμμή εντολών που αντιπροσωπεύει πόσους πρώτους όρους t_i να χρησιμοποιήσει στον υπολογισμό του π και στην συνέχεια εκτυπώνει την προσέγγιση του π με 9 δεκαδικά ψηφία. Παραδείγματα εκτέλεσης είναι τα εξής:

```
$ gcc -o vieta vieta.c -lm
$ ./vieta 2
Multiplied first 2 ti terms, pi = 3.061467459
$ ./vieta 5
Multiplied first 5 ti terms, pi = 3.140331157
$ ./vieta 50
Multiplied first 50 ti terms, pi = 3.141592654
```

Υπόδειξη Κάθε αριθμητής προκύπτει από τον προηγούμενο: αν ο αριθμητής του t_i είναι a, ο επόμενος είναι √(2 + a). Πολλαπλασιάστε τους πρώτους N όρους σε έναν `double` και λύστε τη σχέση ως προς π. Χρειάζεστε `math.h` και `-lm`, και έλεγχο ότι το όρισμα είναι θετικός ακέραιος.

A6.23 · Τυπώνοντας τον Πίνακα ASCII

A6.23 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #15, Θέμα 2* · ★★☆ · programming · `exam-2023-fall-ex15-q2`

Πρόγραμμα: `ascii.c`

Γράψτε ένα πρόγραμμα το οποίο δέχεται ως πρώτο όρισμα τον αριθμό στηλών που θέλουμε να χρησιμοποιήσει για τον πίνακα ASCII και στην συνέχεια τυπώνει τον πίνακα στην πρότυπη έξοδο. Οι χαρακτήρες πρέπει να τυπώνονται με την σειρά ο ένας κάτω από τον άλλο μέχρι να περάσουμε στην επόμενη στήλη. Για κάθε χαρακτήρα πρέπει να τυπώνεται: ”χαρακτήρας: δεκαεξαδική αναπαράσταση”. Για παράδειγμα ”A: 41”. Πρέπει να τυπωθούν όλοι οι χαρακτήρες στο εύρος [0-127]. Στην θέση των χαρακτήρων [0x08-0x0d] θέλουμε να τυπώνεται το 'x'. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o ascii ascii.c
$ ./ascii 5
    :   00  :   19  2:  32  K:  4b  d:  64  }:  7d
    :   01  □:  1a  3:  33  L:  4c  e:  65  ~:  7e
    :   02      1b  4:  34  M:  4d  f:  66  :   7f
    :   03  :   1c  5:  35  N:  4e  g:  67
    :   04  :   1d  6:  36  O:  4f  h:  68
    :   05  :   1e  7:  37  P:  50  i:  69
    :   06  :   1f  8:  38  Q:  51  j:  6a
```

```
    :   07   :  20  9:  39  R:  52  k:  6b
    x:  08  !:  21  ::  3a  S:  53  l:  6c
    x:  09  ":  22  ;:  3b  T:  54  m:  6d
    x:  0a  #:  23  <:  3c  U:  55  n:  6e
    x:  0b  $:  24  =:  3d  V:  56  o:  6f
    x:  0c  %:  25  >:  3e  W:  57  p:  70
    x:  0d  &:  26  ?:  3f  X:  58  q:  71
    :   0e  ':  27  @:  40  Y:  59  r:  72
    :   0f  (:  28  A:  41  Z:  5a  s:  73
    :   10  ):  29  B:  42  [:  5b  t:  74
    :   11  *:  2a  C:  43  \:  5c  u:  75
    :   12  +:  2b  D:  44  ]:  5d  v:  76
    :   13  ,:  2c  E:  45  ^:  5e  w:  77
    :   14  -:  2d  F:  46  _:  5f  x:  78
    :   15  .:  2e  G:  47  `:  60  y:  79
    :   16  /:  2f  H:  48  a:  61  z:  7a
    :   17  0:  30  I:  49  b:  62  {:  7b
    :   18  1:  31  J:  4a  c:  63  |:  7c
$ ./ascii 6
    :   00  :   15  *:  2a  ?:  3f  T:  54  i:  69  ~:  7e
    :   01  :   16  +:  2b  @:  40  U:  55  j:  6a  :   7f
    :   02  :   17  ,:  2c  A:  41  V:  56  k:  6b
    :   03  :   18  -:  2d  B:  42  W:  57  l:  6c
    :   04  :   19  .:  2e  C:  43  X:  58  m:  6d
    :   05  □:  1a  /:  2f  D:  44  Y:  59  n:  6e
    :   06      1b  0:  30  E:  45  Z:  5a  o:  6f
    :   07  :   1c  1:  31  F:  46  [:  5b  p:  70
    x:  08  :   1d  2:  32  G:  47  \:  5c  q:  71
    x:  09  :   1e  3:  33  H:  48  ]:  5d  r:  72
    x:  0a  :   1f  4:  34  I:  49  ^:  5e  s:  73
    x:  0b   :  20  5:  35  J:  4a  _:  5f  t:  74
    x:  0c  !:  21  6:  36  K:  4b  `:  60  u:  75
    x:  0d  ":  22  7:  37  L:  4c  a:  61  v:  76
    :   0e  #:  23  8:  38  M:  4d  b:  62  w:  77
    :   0f  $:  24  9:  39  N:  4e  c:  63  x:  78
    :   10  %:  25  ::  3a  O:  4f  d:  64  y:  79
    :   11  &:  26  ;:  3b  P:  50  e:  65  z:  7a
    :   12  ':  27  <:  3c  Q:  51  f:  66  {:  7b
    :   13  (:  28  =:  3d  R:  52  g:  67  |:  7c
    :   14  ):  29  >:  3e  S:  53  h:  68  }:  7d
```

Υπόδειξη Οι χαρακτήρες τυπώνονται κατά στήλες, αλλά το τερματικό γράφει κατά γραμμές: για τη γραμμή r και τη στήλη c ο χαρακτήρας είναι `c * rows + r`. Βρείτε από τα παραδείγματα

πώς προκύπτει το πλήθος γραμμών από το όρισμα και προσέξτε την ημιτελή τελευταία στήλη και τη μορφοποίηση με `%02x`.

A6.24 · Πετυχαίνοντας τον Στόχο

A6.24 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #9, Θέμα 2* · ★★☆ · programming · `exam-2023-fall-ex9-q2`

Πρόγραμμα: `legolas.c`

Γράψτε ένα πρόγραμμα το οποίο παίρνει ως ορίσματα από την γραμμή εντολών έναν ακέραιο-στόχο (goal) και στην συνέχεια ένα σύνολο υποψηφίων ακεραίων (candidates) και τυπώνει όλους τους συνδυασμούς 3 υποψηφίων των οποίων το άθροισμα ισούται με τον στόχο. Η σειρά με την οποία εκτυπώνονται τα αποτελέσματα δεν έχει σημασία για την ορθότητα του προγράμματος. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o legolas legolas.c
$ ./legolas 42 19 21 3 5 12
No combination of candidates leads to 42
$ ./legolas 42 19 21 3 5 12 11
Candidates combination found: 19 + 12 + 11 = 42
$ ./legolas 42 27 25 12 31 5 26 40 34 3 18
Candidates combination found: 27 + 12 + 3 = 42
Candidates combination found: 25 + 12 + 5 = 42
Candidates combination found: 5 + 34 + 3 = 42
$ ./legolas 37372082074 9238742398 82934723 27893492387 127863435 239847289
Candidates combination found: 9238742398 + 27893492387 + 239847289 = 37372082074
```

Υπόδειξη Τρεις φωλιασμένοι βρόχοι με δείκτες `i < j < k` δίνουν κάθε τριάδα υποψηφίων ακριβώς μία φορά. Το τελευταίο παράδειγμα έχει αριθμούς που δεν χωρούν σε `int`: χρησιμοποιήστε `long long` και κρατήστε αν βρέθηκε έστω ένας συνδυασμός για το τελικό μήνυμα.

A6.25 · Εύρεση Πρώτων Παραγόντων - factor

A6.25 · *Εξέταση Ιανουαρίου 2026, Θέμα 3* · ★★☆ · programming · `exam-2026-jan-q3`

Εύρεση Πρώτων Παραγόντων - factor [25 Μονάδες]

Κάθε φυσικός αριθμός μπορεί να γραφεί ως το γινόμενο των πρώτων παραγόντων του. Για παράδειγμα, ο αριθμός 42 μπορεί να γραφεί ως $2 \cdot 3 \cdot 7$. Γράψτε ένα πρόγραμμα το οποίο παίρνει έναν θετικό ακέραιο ως το πρώτο όρισμα στην γραμμή εντολών και τυπώνει στην έξοδο όλους τους πρώτους παράγοντές του. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ ./factor 42
Factors of 42: 2 3 7
$ ./factor 54
Factors of 54: 2 3 3 3
```

Υπόδειξη Δοκιμάστε διαιρέτες από το 2 και πάνω και, όσο ένας διαιρέτης διαιρεί τον αριθμό, τυπώστε τον και διαιρέστε· έτσι οι επαναλαμβανόμενοι παράγοντες (όπως το 3 στο 54) βγαίνουν σωστά και δεν χρειάζεται ξεχωριστός έλεγχος πρώτου. Σκεφτείτε πότε μπορείτε να σταματήσετε νωρίτερα (`d * d > n`), καθώς και τι γίνεται αν λείπει το όρισμα ή δεν είναι θετικός ακέραιος (π.χ. 0, 1, αρνητικός).

Κεφάλαιο 7: Επίλυση Προβλημάτων

Kahoot από το αμφιθέατρο (K7.1–K7.5)

K7.1 · Αξίζει να σχολιάζω τον κώδικα;

K7.1 · *Kahoot «Τύπωμα και Συναρτήσεις» (διάλεξη 3)* · ★☆☆ · multiple-choice · 91% σωστές · `kahoot-comments-good-practice`

Είναι καλή πρακτική να βάζω σχόλια στον κώδικά μου;

- True
- False

Υπόδειξη Σκεφτείτε ποιος θα διαβάσει τον κώδικά σας σε έναν μήνα (ίσως εσείς).

K7.2 · Η γλώσσα του README.md

K7.2 · *Kahoot «Επίλυση Προβλημάτων» (διάλεξη 7)* · ★☆☆ · multiple-choice · 84% σωστές · `kahoot-readme-markdown`

Η γλώσσα στην οποία γράφουμε το `README.md` μας λέγεται:

- Parkdown
- Markdown
- Smackdown
- Downtown

Υπόδειξη Η κατάληξη `.md` του αρχείου είναι η συντομογραφία της.

Κ7.3 · Εργασία χωρίς σχόλια

Κ7.3 · *Kahoot «Επίλυση Προβλημάτων» (διάλεξη 7)* · ★☆☆ · multiple-choice · 71% σωστές · `kahoot-homework-comments`

Θα βαθμολογηθεί η εργασία μου χωρίς σχόλια στον κώδικά μου;

- True
- False

Συχνή παρανόηση Το 25% απάντησε True, θεωρώντας τα σχόλια προαιρετικά· στο μάθημα όμως τα σχόλια και η τεκμηρίωση είναι μέρος αυτού που βαθμολογείται.

Υπόδειξη Θυμηθείτε τι είπαμε για τον σχολιασμό των προγραμμάτων και την ποιότητα του κώδικα που παραδίδετε.

Κ7.4 · Καλό σχόλιο;

Κ7.4 · *Kahoot «Επίλυση Προβλημάτων» (διάλεξη 7)* · ★★☆ · multiple-choice · 61% σωστές · `kahoot-good-comment`

Αυτό είναι ένα παράδειγμα καλού σχολίου:

```
printf("Hello"); // prints hello to the screen
```

- True
- False

Συχνή παρανόηση Το 34% απάντησε True· όμως ένα σχόλιο που επαναλαμβάνει τι κάνει ο κώδικας δεν προσθέτει τίποτα, ένα καλό σχόλιο εξηγεί το «γιατί».

Υπόδειξη Ρωτήστε τον εαυτό σας τι πληροφορία προσθέτει το σχόλιο σε κάποιον που ήδη ξέρει να διαβάζει C.

Κ7.5 · Πώς γράφουμε σχόλια

Κ7.5 · *Kahoot «Επίλυση Προβλημάτων» (διάλεξη 7)* · ★★☆ · multiple-choice · 46% σωστές · `kahoot-comment-syntax`

Πώς βάζω σχόλια στο πρόγραμμά μου; (Επιλέξτε όλες τις σωστές απαντήσεις.)

- `/* this way */`
- `*/ nope this is the way /*`
- `// wrong again!`

- `\\ the only correct answer is 42`

Υπόδειξη Η C έχει δύο είδη σχολίων: ένα για σχόλια πολλών γραμμών και ένα για σχόλια μέχρι το τέλος της γραμμής. Προσέξτε τη σειρά και τη φορά των χαρακτήρων.

Ζέσταμα: από τις διαφάνειες (A7.1–A7.2)

A7.1 · Τριψήφιοι άρτιοι σε φθίνουσα σειρά

A7.1 · *Διάλεξη 7: Επίλυση Προβλημάτων, διαφάνεια 14* · ★☆☆ · programming · `slides-lec07-even-descending`

Θέλω να τυπώσω όλους τους τριψήφιους άρτιους σε φθίνουσα σειρά (998 996 994 … 100). Πως;

Υπόδειξη Ποιος είναι ο πρώτος αριθμός που πρέπει να τυπωθεί, ποιος ο τελευταίος, και πόσο απέχουν δύο διαδοχικοί; Αυτά τα τρία δίνουν την αρχικοποίηση, τη συνθήκη και το βήμα μιας `for`, όπου η μεταβλητή μειώνεται.

A7.2 · Γινόμενο τριψήφιων περιττών πολλαπλασίων του 7

A7.2 · *Διάλεξη 7: Επίλυση Προβλημάτων, διαφάνεια 16* · ★★☆ · programming · `slides-lec07-odd-multiples-of-7`

Θέλω να βρω το γινόμενο όλων των τριψήφιων περιττών που διαιρούνται με το 7. Πως;

Υπόδειξη Μπορείτε να διατρέξετε τους τριψήφιους περιττούς και να κρατήσετε όσους έχουν `i % 7 == 0`, ή να βρείτε το πρώτο τέτοιο πολλαπλάσιο και το σταθερό βήμα ανάμεσα σε διαδοχικά. Προσέξτε από πού ξεκινά ο συσσωρευτής του γινομένου, και εκτιμήστε πόσο μεγάλο θα βγει το αποτέλεσμα πριν διαλέξετε τύπο.

Κεφάλαιο 8: Ροή Ελέγχου #2

Kahoot από το αμφιθέατρο (K8.1–K8.5)

K8.1 · goto στο διαγώνισμα

K8.1 · *Kahoot «Ροή Ελέγχου #2» (διάλεξη 8)* · ★☆☆ · multiple-choice · 97% σωστές · `kahoot-goto-in-exam`

Είναι καλή ιδέα να χρησιμοποιήσω `goto` στο τελικό διαγώνισμα;

- True
- False

Υπόδειξη Θυμηθείτε τι λέει ο δομημένος προγραμματισμός για τα άλματα σε ετικέτες.

K8.2 · Μόνο με break σταματά ένας βρόχος;

K8.2 · *Kahoot «Ροή Ελέγχου #2» (διάλεξη 8)* · ★☆☆ · multiple-choice · 94% σωστές · `kahoot-break-only-way`

Η εντολή `break` είναι ο μόνος τρόπος για να σταματήσουμε έναν βρόχο;

- True
- False

Υπόδειξη Σκεφτείτε πώς τελειώνει ένας συνηθισμένος βρόχος `for (i = 0; i < 10; i++)` και τι άλλο είδαμε, π.χ. μεταβλητές-σημαίες ή `return`.

K8.3 · switch και if-else

K8.3 · *Kahoot «Ροή Ελέγχου #2» (διάλεξη 8)* · ★★☆ · multiple-choice · 46% σωστές · `kahoot-switch-if-else-equivalence`

Οποιοδήποτε πρόγραμμα χρησιμοποιεί `switch` μπορούμε να το μετατρέψουμε σε `if-else` και το αντίστροφο.

- True
- False

Συχνή παρανόηση Το 53% απάντησε False, επειδή τα `case` δέχονται μόνο ακέραιες σταθερές· όμως μπορούμε να κάνουμε `switch` στην ίδια τη συνθήκη (`switch (x > 3.5)`), που είναι 0 ή 1.

Υπόδειξη Η `switch` δέχεται οποιαδήποτε ακέραια έκφραση· τι τιμή έχει μια σύγκριση όπως `x > 3.5`;

K8.4 · for χωρίς συνθήκη με break

K8.4 · *Kahoot «Ροή Ελέγχου #2» (διάλεξη 8)* · ★★★ · multiple-choice · 36% σωστές · `kahoot-for-break-overflow`

Θα τελειώσει αυτό το loop;

```
for (i = 1; ; i++)
    if (i == 0)
        break;
```

- Όχι δεν τελειώνει καθώς δεν έχει συνθήκη τερματισμού
- Όταν υπερχειλίσει ο ακέραιος i
- Όχι δεν τελειώνει καθώς ο ακέραιος i αρχικοποιείται στο 1 και αυξάνεται
- Μόνο όταν λιώσει η CPU

Συχνή παρανόηση Το 32% απάντησε ότι τελειώνει «μόνο όταν λιώσει η CPU», θεωρώντας ότι το `i` δεν θα γίνει ποτέ 0· όμως μετά τη μέγιστη τιμή του ο ακέραιος υπερχειλίζει σε αρνητικούς και φτάνει ξανά στο 0.

Υπόδειξη Ένας `int` έχει πεπερασμένο εύρος τιμών· σκεφτείτε τι γίνεται αν συνεχίσουμε να τον αυξάνουμε πέρα από τη μέγιστη τιμή του.

K8.5 · while(--i) με i = 0

K8.5 · *Kahoot «Επίλυση Προβλημάτων» (διάλεξη 7)* · ★★★ · multiple-choice · 17% σωστές · `kahoot-while-predecrement-overflow`

Παλιό θέμα (δύσκολο): τι θα εκτυπωθεί από το πρόγραμμα;

```
int i = 0;
while (--i);
printf("@");
```

- Σχεδόν ακαριαία, ένα @
- Μετά από κάποια δευτερόλεπτα εκτέλεσης, ένα @
- Θα εκτυπώνει συνεχώς @, επ' άπειρον
- Hello 42 World

Συχνή παρανόηση Το 43% απάντησε ότι θα τυπώνει @ επ' άπειρον, χάνοντας ότι το `;` είναι η (κενή) εντολή του βρόχου και ότι το `i` κάποτε υπερχειλίζει και φτάνει ξανά στο 0.

Υπόδειξη Προσέξτε το ερωτηματικό αμέσως μετά τη `while`, και σκεφτείτε τι γίνεται όταν ένας `int` μειώνεται ξανά και ξανά πέρα από την ελάχιστη τιμή του.

Ζέσταμα: από τις διαφάνειες (A8.1–A8.6)

A8.1 · Το αγγλικό όνομα κάθε ψηφίου

A8.1 · *Διάλεξη 8: Ροή Ελέγχου #2, διαφάνεια 13* · ★☆☆ · programming · `slides-lec08-digit-names`

Θέλω να τυπώσω το αγγλικό όνομα κάθε ψηφίου (`zero`, `one`, ..., `nine`), και `unknown` για οποιαδήποτε άλλη τιμή. Πώς;

Υπόδειξη Μια αλυσίδα `else if` δουλεύει, αλλά όλες οι συνθήκες συγκρίνουν την ίδια ακέραια μεταβλητή με σταθερές: αυτή είναι η περίπτωση της `switch`. Μην ξεχάσετε τι χρειάζεται στο τέλος κάθε `case` και πού πάνε οι τιμές που δεν είναι ψηφία.

A8.2 · Υπάρχει θέμα με αυτή την υλοποίηση;

A8.2 · *Διάλεξη 8: Ροή Ελέγχου #2, διαφάνεια 7* · ★☆☆ · debug · `slides-lec08-print-all-issue`

Θέλω να τυπώσω (αν υπάρχει) τον μικρότερο τριψήφιο που είναι πολλαπλάσιο του 2 και του 5 αλλά όχι του 4. Πώς;

```
for(i = 100 ; i < 1000 ; i++) {
  if (i % 2 == 0 && i % 5 == 0 && i % 4 != 0) {
      printf("%d\n", i);
  }
}
```

Υπάρχει κάποιο θέμα με αυτήν την υλοποίηση;

Υπόδειξη Τι γίνεται αφού ο βρόχος βρει και τυπώσει τον πρώτο κατάλληλο αριθμό; Μετρήστε πόσες γραμμές θα τυπωθούν και σκεφτείτε πώς σταματάμε έναν βρόχο νωρίς.

A8.3 · Γινόμενο τριψήφιων περιττών πολλαπλασίων του 7

A8.3 · *Διάλεξη 8: Ροή Ελέγχου #2, διαφάνεια 5* · ★☆☆ · programming · `slides-lec08-product-multiples-of-7`

Θέλω να βρω το γινόμενο όλων των τριψήφιων περιττών που διαιρούνται με το 7. Πώς;

Υπόδειξη Υπάρχουν δύο δρόμοι: διατρέξτε τους τριψήφιους περιττούς και κρατήστε με μια `if` όσους διαιρούνται με το 7, ή βρείτε τον πρώτο τέτοιο αριθμό και το σταθερό βήμα ανάμεσα σε διαδοχικούς, ώστε ο βρόχος να μη χρειάζεται `if`. Ελέγξτε με το χέρι τις πρώτες τιμές του μετρητή και σκεφτείτε αν το αποτέλεσμα χωράει σε `int`.

A8.4 · Αρνητικό γινόμενο σε βρόχο

A8.4 · *Διάλεξη 8: Ροή Ελέγχου #2, διαφάνεια 2* · ★☆☆ · debug · `slides-lec08-product-overflow`

Έτρεξα τον παρακάτω πολλαπλασιασμό:

```
for(prod = 1, i = 105 ; i <= 999 ; i += 14) {
    prod *= i;
}
```

και μου τύπωσε το παρακάτω. Τι συμβαίνει;

```
$ ./prod
-1187656959
```

Υπόδειξη Πόσους παράγοντες πολλαπλασιάζει ο βρόχος και πόσο μεγάλο είναι περίπου το σωστό γινόμενο; Συγκρίνετέ το με τη μέγιστη τιμή που χωράει σε έναν `int` των 32 bit.

A8.5 · Η εποχή κάθε μήνα

A8.5 · *Διάλεξη 8: Ροή Ελέγχου #2, διαφάνεια 17* · ★☆☆ · programming · `slides-lec08-season`

Θέλω να τυπώσω την εποχή κάθε μήνα (`winter`, `spring`, `summer`, ...), με τον μήνα ως ακέραιο από 1 έως 12. Πώς;

Υπόδειξη Με `switch` στον μήνα: κάθε εποχή αντιστοιχεί σε τρεις μήνες, οπότε γράψτε τα τρία `case` το ένα κάτω από το άλλο και τις κοινές εντολές μετά το τελευταίο. Προσέξτε ότι ο χειμώνας περιλαμβάνει τον Δεκέμβριο.

A8.6 · Ο μικρότερος τριψήφιος πολλαπλάσιο του 2 και του 5 αλλά όχι του 4

A8.6 · *Διάλεξη 8: Ροή Ελέγχου #2, διαφάνεια 6* · ★☆☆ · programming · `slides-lec08-smallest-three-digit`

Θέλω να τυπώσω (αν υπάρχει) τον μικρότερο τριψήφιο που είναι πολλαπλάσιο του 2 και του 5 αλλά όχι του 4. Πώς;

Υπόδειξη Διατρέξτε τους τριψήφιους σε αύξουσα σειρά και ελέγξτε τη συνθήκη με τον τελεστή `%`. Ο βρόχος πρέπει να σταματήσει μόλις βρει τον πρώτο αριθμό: με μια μεταβλητή-σημαία στη συνθήκη του ή με `break`.

Εργασίες (A8.7)

A8.7 · Η Μέθοδος Newton-Raphson

A8.7 · *Εργασία 1 (2023-24), Άσκηση 1* · ★★☆ · programming · `hw-2023-hw1-newton`

Για πολυώνυμα 5ου βαθμού και πάνω δεν υπάρχει κλειστή μορφή υπολογισμού των ριζών (Abel-Ruffini), οπότε καταφεύγουμε στις προσεγγιστικές μεθόδους της αριθμητικής ανάλυσης.

Η μέθοδος Newton-Raphson, γνωστή και πιο απλά ως μέθοδος Newton, μας επιτρέπει να βρίσκουμε προσεγγιστικές λύσεις σε πραγματικές συναρτήσεις. Η μέθοδος Newton ορίζεται ως εξής: Δεδομένης μιας πραγματικής συνάρτησης $f(x)$ και της παραγώγου της $f'(x)$, ξεκινώντας με ένα τυχαίο x_0 μία καλύτερη προσέγγιση μιας ρίζας του πολυωνύμου x_1 δίνεται από την σχέση:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Αντίστοιχα, η γενική αναδρομική σχέση της μεθόδου του Νεύτωνα είναι:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

όπου x_{n+1} είναι η προσεγγιστική τιμή μιας ρίζας της συνάρτησης $f(x)$ μετά από $n+1$ επαναλήψεις. Το Σχήμα 1 της εκφώνησης δείχνει πως η μέθοδος ξεκινάει από ένα σημείο της καμπύλης x_n και χρησιμοποιεί την κλίση της καμπύλης (την εφαπτομένη στο $(x_n, f(x_n))$) προκειμένου στην επόμενη επανάληψη το x_{n+1} (το σημείο όπου η εφαπτομένη τέμνει τον άξονα x) να είναι μια καλύτερη προσέγγιση μιας ρίζας του πολυωνύμου. Ισχύει αυτό πάντα; Μπορείτε να βρείτε κάποια περίπτωση που η x_{n+1} προσέγγιση είναι χειρότερη της αρχικής x_n;

Για το ζητούμενο αυτής της άσκησης, καλείστε να γράψετε ένα πρόγραμμα που υπολογίζει αυτόματα τις ρίζες (τα x για τα οποία μηδενίζεται) *οποιουδήποτε* πολυωνύμου 5ου βαθμού μέσα σε δευτερόλεπτα.

Τεχνικές Προδιαγραφές

- Repository Name: `progintro/hw1-<YourUsername>`
- C Filepath: `newton/src/newton.c`
- Το πρόγραμμά θα πρέπει να παίρνει επτά ορίσματα από την γραμμή εντολών στην μορφή `./newton a0 a1 a2 a3 a4 a5 x0`. Για την σημασιολογία των ορισμάτων, δείτε παρακάτω την χρήση του προγράμματος. Για οποιαδήποτε είσοδο δεν είναι μέσα στις προδιαγραφές το πρόγραμμα πρέπει να επιστρέφει με κωδικό εξόδου (exit code) 1.
- Για ένα πολυώνυμο 5ου βαθμού $a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5$ και μια αρχική ρίζα x_0 το πρόγραμμα θα εκτελεστεί ως εξής: `$ ./newton a_0 a_1 a_2 a_3 a_4 a_5 x_0`

- Όλες οι παράμετροι θα είναι σε δεκαδική μορφή τύπου `double`. Συνιστούμε να χρησιμοποιήσετε την συνάρτηση βιβλιοθήκης `strtod` για να κάνετε την μετατροπή από όρισμα σε double. Δεν χρειάζεται να ελέγξετε για συνθήκες σφάλματος κατά την μετατροπή σε `double` - εγγυώμαστε ότι όλα τα δεδομένα εισόδου θα είναι αριθμοί και ότι οι αριθμοί αυτοί θα είναι επαρκώς μικροί για να χωράνε σε double.
- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr): `gcc -O3 -Wall -Wextra -Werror -pedantic -o newton newton.c -lm`
- README Filepath: `newton/README.md`
- **Ένα αρχείο που να περιέχει στοιχεία εισόδου και ένα εξόδου διαφορετικά από αυτά της άσκησης. Συγκεκριμένα προτείνουμε να βάλετε έναν συνδυασμό που θεωρείται ότι είναι δύσκολος να γίνει σωστός**.
  - input Filepath: `newton/test/input`
  - output Filepath: `newton/test/output`

  Παράδειγμα που όμως δεν θα γίνει δεκτό από την άσκηση επειδή είναι ήδη στα παραδείγματα παρακάτω, για το input: `1.0 0.0 1.0 0.0 0.0 0.0 2.0` και για το output: `incomplete`.
- Όλα τα αποτελέσματα θα πρέπει να είναι δεκαδικοί με δύο ψηφία ακριβείας και πρόσημο.
- Συνθήκες τερματισμού: ο αλγόριθμος πρέπει να τερματίσει *όταν*:
  1. Ο αλγόριθμος έχει συγκλίνει και ισχύει: $|x_{n+1} - x_n| < 10^{-6}$. Σε αυτήν την περίπτωση τυπώνουμε το αποτέλεσμα x_{n+1} με ακρίβεια δύο δεκαδικών ψηφίων και πρόσημο.
  2. Ο αλγόριθμος αποκλίνει (π.χ., διαίρεση με 0). Σε αυτήν την περίπτωση τυπώνουμε το αποτέλεσμα `nan`.
  3. Ο αλγόριθμος δεν τερματίζει μετά από 1000 επαναλήψεις. Σε αυτήν την περίπτωση τυπώνουμε το αποτέλεσμα `incomplete`.
- Πρέπει να ολοκληρώνει την εκτέλεση μέσα σε: 10 δευτερόλεπτα.

Παρακάτω παραθέτουμε την αλληλεπίδραση με μια ενδεικτική λύση:

```
thanassis@linux14:~$ hostname
linux14
thanassis@linux14:~$ gcc -O3 -Wall -Wextra -Werror -pedantic -o newton newton.c
↪  -lm
thanassis@linux14:~$ ./newton 1.0 2.0 3.0 4.0 5.0 6.0 1.0
-0.67
thanassis@linux14:~$ ./newton 1.0 0.0 1.0 0.0 0.0 0.0 2.0
```

```
incomplete
thanassis@linux14:~$ ./newton 1.0 0.0 1.0 0.0 0.0 0.0 0.0
nan
```

Μπορούμε να επιβεβαιώσουμε τα αποτελέσματα των υπολογισμών μας αποτιμώντας το πολυώνυμο. Για παράδειγμα, το $a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5$ για $x = -0.67$ κάνει 0.00112899 (σχεδόν 0 - μπορούμε να πάρουμε ρίζες που προσεγγίζουν το μηδέν καλύτερα αυξάνοντας την ακρίβεια). Δοκιμάστε και εσείς με τα δικά σας πολυώνυμα για να δείτε αν όντως το πρόγραμμά σας τα καταφέρνει!

Στο αρχείο `README.md` πρέπει να προσθέσετε οποιεσδήποτε παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.

Υπόδειξη Γράψτε δύο μικρές συναρτήσεις, μία για την τιμή του πολυωνύμου και μία για την παράγωγό του (υπολογίστε τους συντελεστές της παραγώγου με το χέρι), και έναν βρόχο που σταματά με τρεις διαφορετικούς τρόπους: σύγκλιση, μηδενική παράγωγος, όριο επαναλήψεων. Ελέγξτε ότι το `argc` είναι ακριβώς 8. Για την έξοδο αρκεί η κατάλληλη μορφή του `printf` με υποχρεωτικό πρόσημο και δύο δεκαδικά· σκεφτείτε και τι θα τυπώσει ένα αποτέλεσμα όπως -0.001.

Δείτε επίσης

- Υποδειγματική υποβολή φοιτητή (περιέχει λύση: δείτε την αφού λύσετε την άσκηση)

Κεφάλαιο 9: Δεδομένα Εισόδου

Kahoot από το αμφιθέατρο (K9.1–K9.9)

K9.1 · Σύγκριση float με ==

K9.1 · *Kahoot «Δεδομένα Εισόδου» (διάλεξη 9)* · ★☆☆ · multiple-choice · 94% σωστές · `kahoot-float-equality`

Είναι καλή ιδέα να ελέγχω `float`/`double` με τον τελεστή ισότητας (`==`);

- True
- False

Υπόδειξη Σκεφτείτε αν αριθμοί όπως το 0.1 αναπαρίστανται ακριβώς στον υπολογιστή και τι σημαίνει αυτό για το αποτέλεσμα μιας σειράς πράξεων.

K9.2 · Πώς στέλνω EOF

K9.2 · *Kahoot «Δεδομένα Εισόδου» (διάλεξη 9)* · ★☆☆ · multiple-choice · 90% σωστές · `kahoot-eof-key`

Ποιος είναι ο συνδυασμός πλήκτρων για να στείλω EOF στο πρόγραμμά μου;

- Ctrl+Alt+Delete
- Ctrl+C
- Ctrl+D
- Ctrl+V

Υπόδειξη Ο ένας από τους συνδυασμούς διακόπτει βίαια το πρόγραμμα· εμείς θέλουμε να του πούμε ευγενικά ότι «δεν έχει άλλα δεδομένα».

K9.3 · Φτάνουν αμέσως τα δεδομένα;

K9.3 · *Kahoot «Δεδομένα Εισόδου» (διάλεξη 9)* · ★☆☆ · multiple-choice · 82% σωστές · `kahoot-input-buffering`

Τα δεδομένα που πληκτρολογούμε στέλνονται άμεσα στο πρόγραμμά μας.

- True
- False

Υπόδειξη Σκεφτείτε τι ρόλο παίζει το πλήκτρο Enter όταν ένα πρόγραμμα περιμένει είσοδο από το πληκτρολόγιο.

K9.4 · Η τιμή του EOF

K9.4 · *Kahoot «Δεδομένα Εισόδου» (διάλεξη 9)* · ★☆☆ · multiple-choice · 79% σωστές · `kahoot-eof-value`

Η τιμή `EOF` είναι συνήθως ίση με τον ακέραιο:

- 1
- 0
- -1
- 42

Υπόδειξη Η τιμή του `EOF` πρέπει να διαφέρει από κάθε έγκυρο χαρακτήρα, που διαβάζεται ως τιμή από 0 έως 255.

K9.5 · Ο τύπος επιστροφής της getchar

K9.5 · *Kahoot «Δεδομένα Εισόδου» (διάλεξη 9) και «Πολυπλοκότητα και άλλα» (διάλεξη 15)* · ★☆☆ · multiple-choice · 66% σωστές · `kahoot-getchar-return-type`

Η συνάρτηση `getchar()` επιστρέφει μια τιμή τύπου:

- int
- double
- char
- Εξαρτάται

Υπόδειξη Εκτός από κάθε δυνατό χαρακτήρα, η `getchar` πρέπει να μπορεί να επιστρέψει και μια ειδική τιμή για το τέλος της εισόδου.

K9.6 · Διάβασμα ακεραίου με scanf

K9.6 · *Kahoot «Δεδομένα Εισόδου» (διάλεξη 9)* · ★★☆ · multiple-choice · 53% σωστές · `kahoot-scanf-int-syntax`

Πώς διαβάζω έναν ακέραιο με την `scanf`;

- `scanf("&d", %num)`
- `scanf("d%", &num)`
- `scanf("%d", num);`
- `scanf("%d", &num)`

Υπόδειξη Το προσδιοριστικό μορφοποίησης γράφεται όπως στην `printf`, και η `scanf` χρειάζεται να ξέρει πού βρίσκεται η μεταβλητή στη μνήμη για να γράψει σε αυτήν.

K9.7 · Μετρητής χαρακτήρων μέχρι την αλλαγή γραμμής

K9.7 · *Kahoot «Δεδομένα Εισόδου» (διάλεξη 9)* · ★★★ · multiple-choice · 31% σωστές · `kahoot-charcount-eof`

Τι ισχύει για το πρόγραμμα:

```
while ((c = getchar()) != '\n')
    charcount++;
printf("%d", charcount);
```

- Θα τυπώνει 0 1 2 3 μέχρι να δώσει ο χρήστης αλλαγή γραμμής
- Δεν είναι δεκτή C
- Δεν θα τερματίσει ποτέ εάν ο χρήστης δώσει EOF

- tl;dr

Συχνή παρανόηση Το 36% επέλεξε ότι θα τυπώνει 0 1 2 3 ..., θεωρώντας ότι η `printf` είναι μέσα στον βρόχο· χωρίς άγκιστρα το σώμα της `while` είναι μόνο η `charcount++;`.

Υπόδειξη Δείτε ποια εντολή ανήκει στο σώμα του βρόχου, και τι επιστρέφει η `getchar` κάθε φορά που την καλούμε αφού η είσοδος έχει τελειώσει.

K9.8 · Πόσες τιμές επιστρέφει η getchar

K9.8 · *Kahoot «Δεδομένα Εισόδου» (διάλεξη 9)* · ★★★ · multiple-choice · 18% σωστές · `kahoot-getchar-257-values`

Πόσες διαφορετικές τιμές μπορεί να πάρει η τιμή επιστροφής της `getchar()`;

- 0
- 128
- 256
- 257

Συχνή παρανόηση Το 64% επέλεξε 256, μετρώντας μόνο τις τιμές ενός byte και ξεχνώντας την επιπλέον τιμή `EOF`, που είναι και ο λόγος που η `getchar` επιστρέφει `int`.

Υπόδειξη Μετρήστε πόσοι διαφορετικοί χαρακτήρες (bytes) μπορούν να διαβαστούν και μην ξεχάσετε ότι υπάρχει και μία ακόμη ειδική τιμή.

K9.9 · Τιμή επιστροφής της scanf με λάθος είσοδο

K9.9 · *Kahoot «Δεδομένα Εισόδου» (διάλεξη 9)* · ★★★ · multiple-choice · 11% σωστές · `kahoot-scanf-return-partial`

Έδωσα την ακολουθία `42 hello 43` στο πρόγραμμά μου:

```
scanf("%d%d", &n1, &n2);
```

Ποια η τιμή επιστροφής της `scanf`;

- 42 42
- 42 43
- 1
- EOF

Συχνή παρανόηση Το 42% επέλεξε `42 43`, μπερδεύοντας την τιμή επιστροφής της `scanf` με τις τιμές που αποθηκεύει στις μεταβλητές, και ξεχνώντας ότι σταματά στο `hello`.

Υπόδειξη Η `scanf` δεν επιστρέφει τις τιμές που διάβασε αλλά πόσες μετατροπές πέτυχαν. Σκεφτείτε πού σταματά όταν συναντήσει κάτι που δεν είναι ακέραιος.

Ζέσταμα: από τις διαφάνειες (A9.1–A9.10)

A9.1 · Μετρητής χαρακτήρων με getchar

A9.1 · *Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 20* · ★☆☆ · trace · `slides-lec09-charcount`

Διαδοχικές κλήσεις της `getchar()` διαβάζουν διαδοχικούς χαρακτήρες. Τι κάνει το παρακάτω πρόγραμμα;

```c
#include <stdio.h>
int main() {
  int ch, sum = 0;
  printf("Enter characters: ");
  while( (ch = getchar()) != EOF ) {
    printf("%c", ch);
    sum++;
  }
  printf("\nTotal characters: %d\n", sum);
  return 0;
}
```

Υπόδειξη Ακολουθήστε τον βρόχο για μια μικρή είσοδο, π.χ. `hi` και Enter, και μετά Ctrl+D. Μην ξεχάσετε ότι και το Enter φτάνει στο πρόγραμμα ως χαρακτήρας.

A9.2 · Τι κάνει το πρόγραμμα με τη scanf;

A9.2 · *Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 30* · ★☆☆ · trace · `slides-lec09-scanf-square`

Τι κάνει το παρακάτω πρόγραμμα;

```c
#include <stdio.h>
int main() {
  int n;
  printf("Gimme a number: ");
  scanf("%d", &n);
  printf("Square: %d\n", n * n);
```

```
  return 0;
}
```

Υπόδειξη Τι σημαίνει η προδιαγραφή %d στη scanf, και γιατί μπροστά από το n υπάρχει &; Τι γράφεται στο stdout αν δώσετε 16;

A9.3 · scanf με πολλά ορίσματα

A9.3 · *Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 36–37* · ★☆☆ · trace · slides-lec09-scanf-two

Τι κάνει το παρακάτω πρόγραμμα; Τι τυπώνει με είσοδο `2 4` (με επιπλέον κενά);

```
#include <stdio.h>
int main() {
  int n1, n2;
  printf("Gimme two numbers: ");
  scanf("%d %d", &n1, &n2);
  printf("Result: %d\n", n1 * n2);
  return 0;
}
```

Υπόδειξη Τι κάνει η %d με τους κενούς χαρακτήρες και τις αλλαγές γραμμής που συναντά πριν από το πρώτο ψηφίο;

A9.4 · Άθροισμα δύο αριθμών από την πρότυπη είσοδο

A9.4 · *Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 25–26* · ★★☆ · programming · slides-lec09-addnums

Θέλω να διαβάσω δύο αριθμούς από την πρότυπη είσοδο και να τους προσθέσω. Πώς;

```
$ ./addnums
Give me a number: 40
Give me another number: 2
Total: 42
```

Η διάλεξη δείχνει έναν τρόπο με τη getchar, φτιάχνοντας μια συνάρτηση getinteger που διαβάζει τους χαρακτήρες έναν-έναν και τους μετατρέπει σε αριθμό. Υπάρχει άλλος τρόπος να επιτύχουμε το ίδιο αποτέλεσμα;

Υπόδειξη Με τη getchar, κάθε νέο ψηφίο ch αλλάζει την τιμή σε `10 * val + (ch - '0')`, μέχρι να διαβαστεί το `'\n'`. Ο άλλος τρόπος είναι μια συνάρτηση της stdio.h που είναι η «συμμετρική» της printf· μην ξεχάσετε να ελέγξετε τι επιστρέφει.

A9.5 · Μια cat με char

A9.5 · *Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 23* · ★★☆ · debug · `slides-lec09-cat-char`

Τι πρόβλημα έχει η παρακάτω υλοποίηση της `cat`;

```c
#include <stdio.h>

int main() {
  char c;
  while((c = getchar()) != EOF)
    putchar(c);
  return 0;
}
```

Υπόδειξη Πόσες διαφορετικές τιμές μπορεί να επιστρέψει η `getchar` και πόσες χωράει ένα `char`; Δοκιμάστε την είσοδο `echo -e "hello\xffworld" | ./cat` και σκεφτείτε ποια τιμή παίρνει το `c` για το byte `0xff`.

A9.6 · OK ή Not OK;

A9.6 · *Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 3* · ★★☆ · trace · `slides-lec09-float-equality`

Τι θα τυπώσει το παρακάτω πρόγραμμα; Υπάρχει κάποια εξήγηση;

```c
#include <stdio.h>
int main() {
    float a = 3.1;

    if(a == 3.1)
       printf("OK\n");
    else
       printf("Not OK\n");
    return 0;
}
```

Υπόδειξη Ποιος είναι ο τύπος της σταθεράς `3.1` και ποιος του `a`; Αναπαρίσταται το `3.1` ακριβώς σε δυαδική μορφή, και πόσα ψηφία ακρίβειας κρατά ένα `float` σε σχέση με ένα `double`;

A9.7 · Η συνάρτηση getinteger

A9.7 · *Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 26, 38* · ★★☆ · trace · `slides-lec09-getinteger`

Τι κάνει το παρακάτω πρόγραμμα;

```
#define ERROR -1          // Return value for illegal character

int getinteger(int base) {
  int ch;                 // No need to declare ch as int - no EOF handling
  int val = 0;                                  // Initialize return value
  while ((ch = getchar()) != '\n')              // Read up to new line
    if (ch >= '0' && ch <= '0' + base - 1)      // Legal character?
      val = base * val + (ch - '0');            // Update return value
    else
      return ERROR;                             // Illegal character read
  return val;   // Everything OK - Return value of number read
}
```

Υπόδειξη Ιχνηλατήστε το `val` για την κλήση `getinteger(10)` με είσοδο `42` και για την `getinteger(2)` με είσοδο `101`. Τι τιμή έχει το `ch - '0'` για έναν χαρακτήρα-ψηφίο; Τι γίνεται αν εμφανιστεί γράμμα;

A9.8 · Μέτρηση γραμμάτων στην είσοδο

A9.8 · *Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 39* · ★★☆ · trace · `slides-lec09-letter-frequencies`

Τι κάνει το παρακάτω πρόγραμμα;

```
int i, ch, total = 0;
int letfr[26];  // Letter occurrences and frequencies array
for (i=0 ; i < 26 ; i++)
  letfr[i] = 0;
while ((ch = getchar()) != EOF) {
  if (ch >= 'A' && ch <= 'Z') {
    letfr[ch-'A']++;            // Found upper case letter
    total++;
  }
  if (ch >= 'a' && ch <= 'z') {
    letfr[ch-'a']++;            // Found lower case letter
    total++;
  }
}
```

Υπόδειξη Ποια θέση του πίνακα `letfr` αυξάνεται όταν διαβαστεί το `'C'` και ποια όταν διαβαστεί το `'c'`; Τι μετρά το `total` και τι γίνεται με τα ψηφία και τα κενά;

A9.9 · scanf χωρίς &

A9.9 · *Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 32* · ★★☆ · short-answer · `slides-lec09-scanf-no-ampersand`

Στο πρόγραμμα

```c
#include <stdio.h>
int main() {
  int n;
  printf("Gimme a number: ");
  scanf("%d", &n);
  printf("Square: %d\n", n * n);
  return 0;
}
```

τι θα γινόταν αν γράφαμε `scanf("%d", n);`; Γιατί;

```
$ ./scanf
Gimme a number: 3
Segmentation fault
```

Υπόδειξη Η C περνάει τα ορίσματα με τιμή. Τι παίρνει η `scanf` στη μία και τι στην άλλη περίπτωση, και πού προσπαθεί να γράψει τον αριθμό που διάβασε;

A9.10 · Είναι σωστό αυτό το πρόγραμμα;

A9.10 · *Διάλεξη 9: Δεδομένα Εισόδου, διαφάνεια 33–35* · ★★☆ · debug · `slides-lec09-scanf-return`

Είναι σωστό αυτό το πρόγραμμα;

```c
#include <stdio.h>
int main() {
  int n;
  printf("Gimme a number: ");
  scanf("%d", &n);
  printf("Square: %d\n", n * n);
  return 0;
}
```

```
$ ./scanf
Gimme a number: 16
Square: 256
$ ./scanf
Gimme a number: Square:
1068701481
$ ./scanf
Gimme a number: hello
Square: 1072038564
```

Υπόδειξη Τι επιστρέφει η `scanf` όταν πατήσετε Ctrl+D και τι όταν γράψετε `hello`; Ποια τιμή έχει το `n` σε αυτές τις περιπτώσεις, αφού δεν αρχικοποιήθηκε ποτέ;

Εργαστήριο (A9.11–A9.14)

A9.11 · Ένα απλό κομπιουτεράκι

A9.11 · *Εργαστήριο 2, Άσκηση 1* · ★☆☆ · programming · `lab-lab02-calc`

Η άσκηση αυτή ίσως σας βοηθήσει και στην Εργασία #0.

Γράψτε ένα πρόγραμμα C `calc.c` που δρα ως ένα κομπιουτεράκι (calculator) για να πραγματοποιεί πρόσθεση. Το πρόγραμμά σας πρέπει να ζητά δύο ακεραίους από την πρότυπη είσοδο (stdin) και στην συνέχεια να τυπώνει το άθροισμά τους. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ ./calc
Please enter the first number: 42
Please enter the second number: 58
The sum of the two numbers is: 100
$ echo $?
0
```

1.1 Υψηλή ακρίβεια (Προχωρημένο, Προαιρετικό)

Τροποποιήστε το παραπάνω πρόγραμμα προκειμένου να μπορεί να δέχεται μεγάλους αριθμούς μέχρι και 10^{18}. Το πρόγραμμά σας συνεχίζει να παράγει τα αναμενόμενα αποτελέσματα;

1.2 Περισσότερες πράξεις (Προχωρημένο, Προαιρετικό)

Τροποποιήστε το calc πρόγραμμά σας προκειμένου να πραγματοποιεί πρόσθεση Ή αφαίρεση ανάλογα με το τι προτιμάει ο χρήστης. Πώς θα σχεδιάζατε την εφαρμογή σας;

1.3 Χειρισμός σφαλμάτων (Προχωρημένο, Προαιρετικό)

Τροποποιήστε το calc πρόγραμμα προκειμένου να βγάζει κωδικό εξόδου 1 εάν οτιδήποτε πάει στραβά κατά την εκτέλεση του προγράμματος - για παράδειγμα εάν ο χρήστης δώσει το string ”hello” αντί για αριθμό.

Υπόδειξη Δύο κλήσεις της `scanf`, μία για κάθε αριθμό, με τη διεύθυνση της μεταβλητής ως όρισμα. Για το 1.1, ένας `int` σταματά γύρω στο $2 \cdot 10^9$: χρειάζεστε ευρύτερο τύπο και την αντίστοιχη προδιαγραφή σε `scanf`/`printf` (και σκεφτείτε αν χωράει το άθροισμα δύο τέτοιων αριθμών). Για το 1.3, ελέγξτε την τιμή που επιστρέφει η `scanf` και τι επιστρέφει η `main`.

A9.12 · Διάβασμα ακεραίου με scanf

A9.12 · *Εργαστήριο 2, Βήμα 3* · ★☆☆ · short-answer · `lab-lab02-readint`

Δημιουργήστε ένα αρχείο `readint.c` με τα ακόλουθα περιεχόμενα:

```c
/* File: readint.c */

#include <stdio.h>

int main() {
  int number;

  printf("Please give me a number: ");
  scanf("%d", &number);

  printf("I just read this number: %d\n", number);
  return 0;
}
```

Μεταγλωττίστε και τρέξτε το παραπάνω πρόγραμμα. Πώς συμπεριφέρεται;

Τι συμβαίνει αν αντί για αριθμό δώσετε μια λέξη ή μια κενή γραμμή;

Τι συμβαίνει αν αντί για αριθμό πληκτρολογήσουμε `Ctrl+D` (EOF);

Τι θα συμβεί αν τυπώσουμε την μεταβλητή `number` πριν τη διαβάσουμε;

Γιατί να μην δώσουμε μια σταθερή τιμή στην μεταβλητή `number` στο πρόγραμμά μας - γιατί να ζητήσουμε από τον χρήστη να δώσει μια τιμή;

Αν χρειαζόταν να διαβάσετε έναν αριθμό κινητής υποδιαστολής, πώς θα το κάνατε;

Υπόδειξη Η `scanf` επιστρέφει πόσες τιμές διάβασε επιτυχώς: δεν αλλάζει τη μεταβλητή όταν αποτυγχάνει. Αναρωτηθείτε λοιπόν ποια τιμή έχει μια τοπική μεταβλητή που δεν

αρχικοποιήθηκε ποτέ, και πώς η `scanf` χειρίζεται τα κενά και τις αλλαγές γραμμής πριν από έναν αριθμό.

A9.13 · Τροποποίηση κειμένου

A9.13 · *Εργαστήριο 4, Άσκηση 3* · ★☆☆ · programming · `lab-lab04-lowercase`

3.1 Αντιγράψτε το αρχείο `capitalize.c` παρακάτω, μεταγλωττίστε το και εκτελέστε το με είσοδο το ίδιο το πηγαίο αρχείο `capitalize.c`. Παρατηρήστε τη λειτουργία του.

```c
/* File: capitalize.c */

#include <stdio.h>

int main() {
  int ch; /* Be careful! Declare ch as int because of getchar() and EOF */
  ch = getchar(); /* Read first character */
  while (ch != EOF) { /* Go on if we didn't reach end of file */
    if (ch >= 'a' && ch <= 'z') { /* If lower case letter */
      ch = ch - ('a'-'A'); /* Move 'a'-'A' positions in the ASCII table */
    }
    putchar(ch); /* Print out character */
    ch = getchar(); /* Read next character */
  }
  return 0;
}
```

Η συνάρτηση `getchar()` διαβάζει έναν χαρακτήρα από την είσοδο και επιστρέφει τον ASCII κωδικό του χαρακτήρα. Αν δεν υπάρχει άλλος χαρακτήρας για διάβασμα στην είσοδο, επιστρέφει την ειδική ακέραια τιμή EOF που είναι ορισμένη στο stdio.h.

3.2 Γράψτε ένα πρόγραμμα `lowercase.c`, ώστε να κάνει το αντίστροφο, δηλαδή να μετατρέπει τους χαρακτήρες που εμφανίζονται με κεφαλαία γράμματα σε χαρακτήρες με μικρά γράμματα.

3.3 Τροποποιήστε το `lowercase.c`, ώστε να μετατρέπονται ταυτόχρονα και οι κεφαλαίοι χαρακτήρες σε μικρούς και οι μικροί σε κεφαλαίους.

Υπόδειξη Κρατήστε τον ίδιο σκελετό ανάγνωσης μέχρι το `EOF` και αλλάξτε μόνο τον έλεγχο και τη μετατόπιση στον πίνακα ASCII. Στο 3.3 προσέξτε να μη μετατρέψετε έναν χαρακτήρα δύο φορές: οι δύο περιπτώσεις πρέπει να αποκλείουν η μία την άλλη.

A9.14 · Κωδικοποίηση και αποκωδικοποίηση κειμένου

A9.14 · *Εργαστήριο 4, Άσκηση 4* · ★★☆ · programming · `lab-lab04-encode-decode`

Σε αυτήν την άσκηση θα γράψουμε προγράμματα για την δεκαεξαδική κωδικοποίηση και αποκωδικοποίηση ενός κειμένου.

4.1 Δημιουργήστε ένα αρχείο κειμένου με όνομα `text` και πληκτρολογήστε σε αυτό ένα τυχαίο κείμενο.

4.2 Γράψτε ένα πρόγραμμα `encode.c` που να διαβάζει από την είσοδο χαρακτήρες και, για καθένα απ' αυτούς, να εκτυπώνει στην έξοδο τον ASCII κωδικό που έχει σε δεκαεξαδική μορφή χρησιμοποιώντας ακριβώς δύο ψηφία. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ echo hello world | ./encode
68656c6c6f20776f726c640a
```

4.3 Εκτελέστε το πρόγραμμα encode με ανακατεύθυνση εισόδου από το αρχείο `text`.

4.4 Εκτελέστε το πρόγραμμα encode με ανακατεύθυνση εισόδου από το αρχείο `text` και ανακατεύθυνση εξόδου στο αρχείο `encoded_text`.

4.5 Γράψτε ένα πρόγραμμα `decode.c` που να διαβάζει από την είσοδο τα δεκαεξαδικά ψηφία για τον κάθε χαρακτήρα και για κάθε δύο από αυτά τα ψηφία να εκτυπώνει τον χαρακτήρα που αντιστοιχεί σε αυτόν τον δεκαεξαδικό αριθμό. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ echo 68656c6c6f20776f726c640a | ./decode
hello world
```

Ο αριθμός `68` αντιστοιχεί στον χαρακτήρα `'h'` κ.ο.κ. Για την αποκωδικοποίηση συνιστούμε να χρησιμοποιήσετε την συνάρτηση `getchar()`. Αν το επιθυμείτε μπορείτε να δοκιμάσετε και άλλες λύσεις όπως π.χ., την `scanf("%2x", ...)` αλλά μην ξεχάσετε να αποθηκεύσετε το αποτέλεσμα σε μεταβλητή τύπου `int` (και να μεταγλωττίσετε με το flag `-Wall` αλλιώς είναι πιθανό να σας ξεφύγουν διάφορα λαθάκια).

4.6 Μεταγλωττίστε το και εκτελέστε το με ανακατεύθυνση της εισόδου από το αρχείο `encoded_text`.

4.7 Εκτελέστε το πρόγραμμα `decode` για την αποκωδικοποίηση ενός κωδικοποιημένου κειμένου, έτσι ώστε να παίρνει την είσοδό του απ' ευθείας από το πρόγραμμα `encode`, το οποίο να καλείται έτσι ώστε να κωδικοποιεί το αρχείο `text`, χωρίς να χρησιμοποιήσετε το ενδιάμεσο αρχείο `encoded_text`. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ echo hello world | ./encode | ./decode
hello world
```

Υπόδειξη Για το `encode`, η `printf` με πλάτος πεδίου και μηδενικά μπροστά δίνει πάντα δύο δεκαεξαδικά ψηφία. Για το `decode` με `getchar`, διαβάστε τα ψηφία ανά ζεύγη, μετατρέψτε κάθε

ψηφίο (`0–9` ή `a–f`) στην τιμή του 0–15 και συνδυάστε τα ως 16 · πρώτο + δεύτερο. Σκεφτείτε τι κάνετε με την τελική αλλαγή γραμμής της εισόδου, που δεν είναι δεκαεξαδικό ψηφίο.

Εργασίες (A9.15–A9.17)

A9.15 · Συνεργασία (Prisoner's Dilemma)

A9.15 · *Εργασία 2 (2023-24), Άσκηση 3* · ★★☆ · programming · `hw-2023-hw2-coop`

Πολλά επιτραπέζια παιχνίδια (π.χ. το Uno) έχουν πολλούς γύρους όπου σε κάθε γύρο μπορείς να συνεργαστείς με τον διπλανό σου ή να τον τιμωρήσεις, και η επιλογή σου επηρεάζει το πώς θα συμπεριφερθούν οι άλλοι στη συνέχεια. Τέτοιες ερωτήσεις μελετά η θεωρία παιγνίων (game theory).

Ένας τρόπος να μοντελοποιήσουμε τέτοια παιχνίδια με πολλούς γύρους όπου οι παίκτες συνεργάζονται ή όχι είναι το Prisoner's Dilemma. Σε κάθε γύρο αυτού του παιχνιδιού, δύο παίκτες Α και Β αποφασίζουν αν θα συνεργαστούν (cooperate) ή όχι (defect). Αν και οι δύο συνεργαστούν τότε αποκομίζουν κάποιους πόντους (έστω Reward (R) = 3). Αν και οι δύο δεν συνεργαστούν τότε παίρνουν λιγότερους πόντους (έστω Payoff (P) = 1). Αν ο ένας (έστω ο Α) αποφασίσει να συνεργαστεί και ο Β τον "προδώσει" χωρίς να συνεργαστεί τότε ο Β θα αποκομίσει περισσότερους πόντους (έστω Temptation (T) = 5) ενώ ο Α δεν θα πάρει καθόλου πόντους (έστω Sucker (S) = 0). Ένας τρόπος να συνοψίσουμε το παραπάνω παιχνίδι πόντων είναι με έναν πίνακα (payoff matrix):

A / B	B συνεργάζεται	B δεν συνεργάζεται
A συνεργάζεται	A: 3, B: 3	A: 0, B: 5
A δεν συνεργάζεται	A: 5, B: 0	A: 1, B: 1

Όμως τα περισσότερα παιχνίδια δεν τελειώνουν στον πρώτο γύρο! Για παράδειγμα, αν στον πρώτο γύρο είδε ο Α ότι ο Β δεν συνεργάστηκε, ο Α μπορεί να αποφασίσει ότι δεν θα συνεργαστεί στον δεύτερο γύρο και θα περιμένει να δει πως θα "συμπεριφερθεί" ο Β στον δεύτερο γύρο. Αντίστοιχες αποφάσεις μπορεί να έχει στο μυαλό του και ο Β.

Για το ζητούμενο αυτής της άσκησης, ο στόχος είναι να γράψετε ένα πρόγραμμα που θα αποφασίζει πως να συμπεριφερθεί αποδοτικά σε τέτοια επαναληπτικά παιχνίδια αποφάσεων.

Τεχνικές Προδιαγραφές

- Repository Name: `progintro/hw2-<YourUsername>`
- C Filepath: `coop/src/coop.c`
- Το πρόγραμμα θα πρέπει να δέχεται από την πρότυπη είσοδο (stdin) 2 χαρακτήρες: 'C' (Cooperate), 'D' (Defect / Don't cooperate) - όλοι οι άλλοι χαρακτήρες αγνοούνται. Το πρόγραμμα θα πρέπει να τερματίζει με exit code 0 όταν λάβει EOF.

- Όταν ξεκινήσει το πρόγραμμά μας πρέπει να τυπώσει την εντολή που επιθυμεί **πριν** επιχειρήσει να διαβάσει την εντολή του άλλου παίκτη. Η εντολή του άλλου παίκτη παρέχεται μόνο αφού το πρόγραμμά σας τυπώσει την δική του εντολή.
- Το πρόγραμμα θα πρέπει να τυπώνει εντολές ως χαρακτήρες σε δύο μορφές: 'C' (Cooperate) και 'D' (Defect / Don't Cooperate). Μετά από κάθε εντολή το πρόγραμμά σας **πρέπει** να τυπώνει καινούρια γραμμή ('\n').
- Για την είσοδο και έξοδο του προγράμματος μπορούν να χρησιμοποιηθούν **μόνο** οι συναρτήσεις `getchar` και `putchar`.
- Το πρόγραμμά σας **πρέπει** να τυπώνει σε κάθε γύρο την απόφασή του ανεξαρτήτως του buffering που μπορεί να υπάρχει - συνίσταται η χρήση της συνάρτησης `fflush` μετά από κάθε κλήση της `putchar`.
- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr): `gcc -Os -Wall -Wextra -Werror -pedantic -o coop coop.c -lm`
- Το πρόγραμμά μας πρέπει να παίρνει τουλάχιστον 1 πόντο σε κάθε 1000 γύρους με έναν εκδικητικό παίκτη.
- Το πρόγραμμά μας πρέπει να παίρνει τουλάχιστον 3000 πόντους σε κάθε 1000 γύρους με έναν συνεργατικό παίκτη.
- Το πρόγραμμά μας πρέπει να παίζει τουλάχιστον μία φορά D και μία φορά C σε κάθε 1000 γύρους παιχνιδιού.
- README Filepath: `coop/README.md`
- Μέγιστος αριθμός γύρων: 10^6.
- Πρέπει να ολοκληρώνει την εκτέλεση 1000 εντολών μέσα σε: 1 δευτερόλεπτο.

Ας προσπαθήσουμε να παίξουμε με μια ενδεικτική λύση (όχι η καλύτερη):

```
$ gcc -Os -Wall -Wextra -Werror -pedantic -o coop coop.c -lm
$ ./coop
C
```

Παρατηρούμε ότι το πρόγραμμα έπαιξε ”C” (Cooperate). Σε απάντηση μπορούμε να απαντήσουμε D και μετά Enter:

```
$ gcc -Os -Wall -Wextra -Werror -pedantic -o coop coop.c -lm
$ ./coop
C
D
D
```

Παρατηρούμε ότι αφού γράψαμε ”D” (Defect) το πρόγραμμα μας απάντησε με ”D” για τον επόμενο γύρο. Έστω ότι απαντάμε D σε αυτόν τον γύρο και εμείς. Σύνολο σε αυτούς τους δύο γύρους επομένως εμείς (που γράφουμε από το stdin) συλλέξαμε 5 + 1 = 6 πόντους ενώ η ενδεικτική μας λύση (coop) μάζεψε 0 + 1 = 1 πόντους.

Έστω ότι θέλουμε να δούμε πως συμπεριφέρεται μια ενδεικτική (όχι η καλύτερη) λύση όταν παίζει με έναν "εκδικητικό" παίκτη που πάντα δεν συνεργάζεται:

```
$ echo > input; for i in `seq 1 1000`; do echo D >> input; done
$ ./coop < input | head -n -1 > output
$ grep -c D output
42
$ grep -c C output
958
```

Επομένως παρατηρούμε ότι το πρόγραμμά μας συνεργάστηκε 958 φορές και 42 δεν συνεργάστηκε, επομένως συγκέντρωσε $958 \cdot 0 + 42 = 42$ πόντους. Αντίστοιχα μπορούμε να ελέγξουμε πως συμπεριφέρεται με έναν "συνεργατικό" παίκτη:

```
$ echo > input; for i in `seq 1 1000`; do echo C >> input; done
$ ./coop < input | head -n -1 > output
$ grep -c D output
3
$ grep -c C output
997
```

Επομένως παρατηρούμε ότι το πρόγραμμά μας συνεργάστηκε 997 φορές και 3 δεν συνεργάστηκε, επομένως συγκέντρωσε $997 \cdot 3 + 3 \cdot 5 = 3006$ πόντους. Αν θέλετε να δοκιμάσετε διαφορετικές στρατηγικές και να δείτε πως θα "έπαιζε" με διαφορετικούς παίκτες παραθέτουμε ένα ενδεικτικό (όχι το τελικό!) script "διαιτητή" στο ακόλουθο URL: https://github.com/progintro/data/blob/main/scripts/referee.py που μπορεί να τρέξει διαφορετικές εκδοχές του προγράμματός σας για έναν αριθμό γύρων και να σας δώσει το score του πρώτου παίκτη. Για παράδειγμα:

```
##### Download referee
$ curl -O
 ↪ https://raw.githubusercontent.com/progintro/data/main/scripts/referee.py
##### Check what's my score against the `coop` implementation
$ python3 referee.py ./smart_coop ./coop 1000
3016
```

Στο αρχείο `README.md` πρέπει να προσθέσετε οποιεσδήποτε παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης. Για τις υποβολές με την καλύτερη απόδοση (αυτές που θα συγκεντρώσουν τους περισσότερους βαθμούς σε παιχνίδια με όλες τις άλλες υποβολές) θα υπάρχει bonus βαθμολογία: η πρώτη υποβολή θα λάβει +100%, η δεύτερη +70% και η τρίτη +40%.

Υπόδειξη Η δομή είναι ένας βρόχος «τύπωσε κίνηση, `fflush`, διάβασε την κίνηση του αντιπάλου με `getchar` αγνοώντας ό,τι δεν είναι 'C' ή 'D', μέχρι το `EOF`». Η στρατηγική μπορεί να κρατά λίγες μεταβλητές κατάστασης (π.χ. την τελευταία κίνηση του αντιπάλου ή μετρητές). Ελέγξτε ότι η στρατηγική σας ικανοποιεί και τους τρεις ελάχιστους όρους (εκδικητικός,

συνεργατικός παίκτης, τουλάχιστον ένα C και ένα D ανά 1000 γύρους), και διαβάστε για κλασικές στρατηγικές όπως το tit-for-tat.

Δείτε επίσης

- Υποδειγματική υποβολή φοιτητή (περιέχει λύση: δείτε την αφού λύσετε την άσκηση)

A9.16 · Το Πρόβλημα του Τρόλεϊ (trolley)

A9.16 · *Εργασία 0 (2024-25), Άσκηση 3* · ★★☆ · programming · `hw-2024-hw0-trolley`

Το Πρόβλημα του Τρόλεϊ (trolley - 50 Μονάδες)

Τα αυτοκινούμενα οχήματα (self-driving cars) πρέπει κάποια στιγμή να λύσουν μια εκδοχή του *Προβλήματος του Τρόλεϊ*, του ηθικού διλήμματος που πρότεινε η Philippa Foot το 1967: το όχημα φτάνει σε μια διασταύρωση, τα φρένα δεν λειτουργούν, και πρέπει να στρίψει αριστερά ή δεξιά, με κάθε επιλογή να έχει κάποιο κόστος.

Για τις ανάγκες αυτής της άσκησης, θα γράψετε ένα πρόγραμμα το οποίο λύνει επαναλαμβανόμενες εκδοχές του προβλήματος του τρόλεϊ όπως θα χρειαζόταν να τις λύσει και ένας αυτόματος πιλότος σε ένα αυτοκινούμενο όχημα. Επειδή η άσκηση μας είναι κυρίως προγραμματιστική, δεν θα αποπειραθούμε να λύσουμε τα ηθικά διλήμματα και θα ζητάμε από τον χρήστη να μας ενημερώσει για το κόστος της κάθε επιλογής μας αν πάμε αριστερά ή δεξιά.

Τεχνικές Προδιαγραφές

- Repository Name: progintro/hw0-<YourUsername>
- Αρχείο C (Filepath): trolley/src/trolley.c
- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr): `gcc -O0 -m32 -Wall -Wextra -Werror -pedantic -o trolley trolley.c`
- README Filepath: trolley/README.md
- Το πρόγραμμά σας πρέπει να διαβάζει όλα τα κόστη που έδωσε ο χρήστης μέχρι το τέλος αρχείου (End Of File - EOF).
- Όλα τα κόστη που θα δοθούν στο πρόγραμμά σας θα είναι δεκαδικοί ακέραιοι στο εύρος: -10^{18} έως 10^{18}.
- Σε κάθε επανάληψη του προβλήματος το πρόγραμμά σας θα πρέπει να ζητάει από τον χρήστη να δώσει τα κόστη για δύο επιλογές: αν πάει αριστερά (left) ή αν πάει δεξιά (right). Το πρόγραμμά σας πρέπει να ζητάει το κόστος για την αριστερή επιλογή πρώτα και στην συνέχεια για την δεξιά.
- Τα κόστη που θα δώσει ο χρήστης μπορούν να διαχωρίζονται με κενό (space), tab ή νέα γραμμή (new line).

- Για κάθε δύο κόστη που δίνονται, το πρόγραμμά σας πρέπει να τυπώνει στην πρότυπη έξοδο (stdout) είτε `Go left` είτε `Go right` - διαλέγοντας πάντα την επιλογή με το μικρότερο κόστος. Αν οι δύο επιλογές έχουν το ίδιο κόστος, το πρόγραμμά σας πρέπει να τυπώνει `Go left`.
- Αν ο χρήστης δώσει EOF αντί για το πρώτο κόστος το πρόγραμμά σας πρέπει να τερματίζει με κωδικό εξόδου (exit code) 0.
- Αν δοθεί οποιαδήποτε είσοδος εκτός προδιαγραφών (για παράδειγμα ο χρήστης δίνει μόνο το κόστος της αριστερής επιλογής) το πρόγραμμά σας πρέπει να τερματίζει με κωδικό εξόδου (exit code) 1.
- Για 10.000 επαναλήψεις του πειράματος, το πρόγραμμά σας πρέπει να ολοκληρώνει την εκτέλεση μέσα σε: 1 δευτερόλεπτο.

Παρακάτω παραθέτουμε αλληλεπιδράσεις με μια ενδεικτική λύση:

```
thanassis@linux14:~$ hostname
linux14
thanassis@linux14:~$ gcc -O0 -m32 -Wall -Wextra -Werror -pedantic -o trolley
 ↪  trolley.c
thanassis@linux14:~$ ./trolley
Please enter the cost of going left: 10
Please enter the cost of going right: 100
Go left.
Please enter the cost of going left: 41
Please enter the cost of going right: 42
Go left.
Please enter the cost of going left: 42
Please enter the cost of going right: 41
Go right.
Please enter the cost of going left: 1000000
Please enter the cost of going right: 1000000
Go left.
Please enter the cost of going left: Terminating.
thanassis@linux14:~$ echo $?
0
thanassis@linux14:~$ ./trolley
Please enter the cost of going left: 42
Please enter the cost of going right: No right cost provided.
thanassis@linux14:~$ echo $?
1
thanassis@linux14:~$ ./trolley
Please enter the cost of going left: 10000000000000
Please enter the cost of going right: 4
Go right.
Please enter the cost of going left: -4
```

```
Please enter the cost of going right: -4
Go left.
Please enter the cost of going left: 10000000000000
Please enter the cost of going right: 4000000000000
Go right.
Please enter the cost of going left: Terminating.
thanassis@linux14:~$ echo $?
0
```

Στο αρχείο README.md πρέπει να προσθέσετε οποιεσδήποτε παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.

Υπόδειξη Με `-m32` ο `long` έχει 32 bits, άρα για κόστη έως 10^{18} χρειάζεστε τύπο 64 bits και τον αντίστοιχο προσδιοριστή μορφής στη `scanf`. Η τιμή επιστροφής της `scanf` ξεχωρίζει τις τρεις περιπτώσεις που σας ενδιαφέρουν: διαβάστηκε αριθμός, ήρθε EOF, ή ήρθε κάτι που δεν είναι αριθμός. Προσέξτε ότι το EOF πριν από το αριστερό κόστος είναι κανονικός τερματισμός, ενώ πριν από το δεξί είναι σφάλμα.

Δείτε επίσης

- Υποδειγματική υποβολή φοιτητή (περιέχει λύση: δείτε την αφού λύσετε την άσκηση)
- Υποδειγματική υποβολή φοιτητή (2) (περιέχει λύση: δείτε την αφού λύσετε την άσκηση)

A9.17 · Επεξεργασία Ήχου (soundwave)

A9.17 · *Εργασία 1 (2025-26), Άσκηση 1* · ★★★ · programming · `hw-2025-hw1-soundwave`

Ο ήχος είναι ένα αναλογικό σήμα, μια συνάρτηση $f(t)$ της έντασης ως προς τον χρόνο, που για να αποθηκευθεί σε υπολογιστή ψηφιοποιείται καταγράφοντας την τιμή του σε διακριτές ισαπέχουσες χρονικές στιγμές (π.χ. 44.100 δείγματα ανά δευτερόλεπτο σε ένα μουσικό CD). Αντικείμενο αυτής της εργασίας είναι η δημιουργία *ενός συστήματος επεξεργασίας ήχου για αρχεία τύπου wav*.

Η μορφή των αρχείων wav Τα αρχεία wav είναι δυαδικά—δηλαδή τα περιεχόμενά τους δεν είναι άμεσα αναγνώσιμα ως κείμενο αλλά αντιθέτως περιέχουν μια σειρά από bytes διατεταγμένα σε ένα αρχείο στον δίσκο με συγκεκριμένες ερμηνείες. Τα αρχεία wav που θα κληθούμε να διαχειριστούμε στα πλαίσια της εργασίας έχουν την ακόλουθη δομή:

```
FileHeader | SampleData | OtherData
```

Δηλαδή αποτελούνται από μια κεφαλίδα (FileHeader) που περιέχει πληροφορίες για το συγκεκριμένο αρχείο ήχου wav, ακολουθούμενο από τα ίδια τα δείγματα του ήχου (SampleData)

και στην συνέχεια ακολουθούν προαιρετικά άλλα δεδομένα (OtherData). Τα αρχεία κεφαλίδας wav, περιέχουν τα ακόλουθα πεδία με την σειρά:

- 4 bytes με τους χαρακτήρες `RIFF` (αρχικά του Resource Interchange File Format, που είναι το πρότυπο που χρησιμοποιείται για τη δομή των wav αρχείων).
- 4 bytes, συμβολικά *SizeOfFile*, για το μέγεθος του αρχείου, που στην πραγματικότητα είναι το πλήθος των bytes που ακολουθούν μέχρι το τέλος του αρχείου. Ουσιαστικά, είναι το μέγεθος του αρχείου στον δίσκο μείον 8 bytes, δηλαδή αυτά που καταλαμβάνουν το τρέχον πεδίο και το προηγούμενο (RIFF). Σημειώνεται ότι η αναπαράσταση ακεραίων χωρίς πρόσημο γίνεται κατά σειρά από το λιγότερο σημαντικό byte προς το περισσότερο σημαντικό (πρότυπο little-endian). Δηλαδή, ο ακέραιος `0xA3870FB9` αναπαρίσταται κατά σειρά με τα bytes `0xB9, 0x0F, 0x87, 0xA3`.
- 4 bytes με τους χαρακτήρες `WAVE`, που υποδεικνύουν τη συγκεκριμένη υποκατηγορία αρχείων που ακολουθούν το πρότυπο RIFF.
- 4 bytes με τους χαρακτήρες `"fmt "` (προσοχή στον τελευταίο χαρακτήρα κενού διαστήματος), που υποδεικνύουν την έναρξη του τμήματος format (format chunk) του αρχείου.
- 4 bytes για τον χώρο που καταλαμβάνουν τα δεδομένα του τμήματος format που θα ακολουθήσουν. Στα αρχεία που θα χρησιμοποιηθούν στην εργασία, η τιμή αυτού του πεδίου θα είναι πάντα 16 (`0x00000010`). Και εδώ εφαρμόζεται το πρότυπο little-endian, δηλαδή ο αριθμός αυτός αποθηκεύεται με την ακολουθία από bytes `0x10, 0x00, 0x00, 0x00`.
- 2 bytes για τον τύπο του WAVE format. Για την εργασία, αυτός θα ισούται πάντα με 1 (`0x0001`) και θα αποθηκεύεται κατά little-endian, δηλαδή με τα bytes `0x01, 0x00`.
- 2 bytes, συμβολικά *MonoStereo*, για το αν ο ήχος είναι μονοφωνικός (τιμή `0x0001`) ή στερεοφωνικός με δύο κανάλια (τιμή `0x0002`). Προσοχή στο little-endian, όπως και σε όλες τις καταχωρήσεις ακεραίων που ακολουθούν.
- 4 bytes, συμβολικά *SampleRate*, για το ρυθμό δειγματοληψίας, δηλαδή πόσες τιμές, ανά δευτερόλεπτο, της συνάρτησης του αναλογικού ήχου έχουν καταγραφεί στο αρχείο.
- 4 bytes, συμβολικά *BytesPerSec*, για το πλήθος bytes ανά δευτερόλεπτο ήχου, που είναι καταχωρημένα στο αρχείο.
- 2 bytes, συμβολικά *BlockAlign*, για το πλήθος των bytes που απαιτούνται για την καταχώρηση της πληροφορίας του ήχου σε μία χρονική στιγμή, για όλα τα κανάλια. Σημειώστε ότι πάντοτε θα πρέπει να ισχύει ότι *BytesPerSec = SampleRate × BlockAlign*.
- 2 bytes, συμβολικά *BitsPerSample*, για το πλήθος των bits που απαιτούνται για την καταχώρηση της πληροφορίας του ήχου σε μία χρονική στιγμή, για ένα μόνο κανάλι. Στα πλαίσια της εργασίας, αυτή η τιμή θα είναι είτε 8 (`0x0008`), είτε 16 (`0x0010`). Για *BitsPerSample* = 8, η ένταση του ήχου είναι ένας ακέραιος χωρίς πρόσημο στο διάστημα [0,255], ενώ για *BitsPerSample* = 16, η ένταση του ήχου είναι ένας ακέραιος με πρόσημο στο διάστημα [-32768,32767]. Σημειώστε ότι πάντοτε θα πρέπει να ισχύει ότι *BlockAlign = BitsPerSample/8 × MonoStereo*.
- 4 bytes με τους χαρακτήρες `data`, που υποδεικνύουν την έναρξη του τμήματος data (data chunk) του αρχείου.

- 4 bytes, συμβολικά *SizeOfData*, για τον χώρο που καταλαμβάνουν τα δεδομένα του τμήματος data που θα ακολουθήσουν.

Μετά την κεφαλίδα του αρχείου, ακολουθούν τα ψηφιοποιημένα δεδομένα του ήχου με χρονική σειρά (SampleData). Στην περίπτωση στερεοφωνικού ήχου με δύο κανάλια, για κάθε χρονική στιγμή, υπάρχει ένα ζευγάρι δεδομένων, με το πρώτο στοιχείο να αντιστοιχεί στο αριστερό κανάλι και το δεύτερο στοιχείο στο δεξιό κανάλι. Μετά το τέλος των δεδομένων ήχου, ενδεχομένως να υπάρχουν και άλλα τμήματα (OtherData) που προβλέπονται για την κατηγορία WAVE, με τα οποία όμως δεν θα ασχοληθούμε στην εργασία.

Ας δούμε τώρα τα περιεχόμενα ενός πραγματικού αρχείου wav. Μπορούμε να δούμε τα δυαδικά περιεχόμενα οποιουδήποτε αρχείου χρησιμοποιώντας την εντολή `od` ως εξής (η εκφώνηση δείχνει και ένα σχήμα με τα ίδια bytes μέσω `xxd -g1`, όπου σημειώνονται τα πεδία):

```
$ od -tx1 good.wav
0000000 52 49 46 46 85 00 00 00 57 41 56 45 66 6d 74 20
0000020 10 00 00 00 01 00 01 00 44 ac 00 00 88 58 01 00
0000040 02 00 10 00 64 61 74 61 58 00 00 00 00 00 d4 07
0000060 a1 0f 5e 17 04 1f 8a 26 ea 2d 1c 35 18 3c d7 42
0000100 54 49 86 4f 69 55 f6 5a 27 60 f8 64 63 69 64 6d
0000120 f7 70 18 74 c5 76 fa 78 b6 7a f6 7b b9 7c ff 7c
0000140 c8 7c 12 7c e0 7a 33 79 0b 77 6c 74 58 71 d1 6d
0000160 dd 69 7e 65 b8 60 92 5b 0f 56 36 50 0c 4a 97 43
0000200 df 3c ea 35 45 78 74 72 61 44 61 74 61
0000215
```

> **Σημείωση**. Πέρα από την εντολή `od`, υπάρχουν και άλλες εντολές όπως οι: (1) `xxd`, (2) `hexdump`, (3) `hexedit` που προσφέρουν αντίστοιχες δυνατότητες σε Unix-like συστήματα (Linux, MacOS, WSL κοκ). Για να *συγκρίνετε* δύο δυαδικά αρχεία, μπορείτε να χρησιμοποιήσετε την εντολή `cmp` ή την εντολή `vbindiff`. Για περιβάλλοντα Windows, υπάρχουν ελεύθερα διαθέσιμα προγράμματα που λειτουργούν παρόμοια (π.χ. https://mh-nexus.de/en/hxd/).

Φτάσαμε λοιπόν στο ζητούμενο αυτής της άσκησης: να υλοποιήσουμε ένα πρόγραμμα επεξεργασίας αρχείων wav με δυνατότητες αλλαγής και παραγωγής ήχου—όλα σε γλώσσα C!

Τεχνικές Προδιαγραφές

- Repository Name: `progintro/hw1-<YourUsername>`
- README Filepath: `soundwave/README.md`
- Αρχείο C (Filepath): `soundwave/src/soundwave.c`
- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα

μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr): `gcc -Ofast -Wall -Wextra -Werror -pedantic -o soundwave soundwave.c -lm`

- Δύο αρχεία που να περιέχουν στοιχεία εισόδου και εξόδου διαφορετικά από αυτά της άσκησης για την εντολή `./soundwave rate 2.0`. Συγκεκριμένα προτείνουμε να βάλετε έναν συνδυασμό που θεωρείται ότι είναι δύσκολος να γίνει σωστός.
  - input Filepath: `soundwave/test/input.wav`
  - output Filepath: `soundwave/test/output.wav`
- Όλα τα περιεχόμενα των αρχείων wav θα δίνονται στο πρόγραμμά σας μέσω της πρότυπης εισόδου (stdin) και πρέπει να διαβάζονται με χρήση της συνάρτησης `getchar`. Δεν θα γίνουν δεκτές υλοποιήσεις όπου χρησιμοποιήθηκε κάποια άλλη συνάρτηση εισόδου εκτός της `getchar`.
- Όλες οι υποεντολές σας πρέπει να επιστρέφουν κωδικό εξόδου (exit code) 0 όταν ολοκλήρωσαν την επεξεργασία τους επιτυχώς - διαφορετικά πρέπει να επιστρέφουν με κωδικό εξόδου 1.
- Το πρόγραμμά σας θα πρέπει να υλοποιεί τις ακόλουθες 5 υποεντολές:
  - `info`: για τύπωμα των στοιχείων του wav και έλεγχο ορθότητας.
  - `rate`: για αλλαγή της ταχύτητας αναπαραγωγής (αντίστοιχο με το playback speed σε streaming υπηρεσίες).
  - `channel`: για επιλογή του καναλιού που θέλουμε.
  - `volume`: για μετατροπή της έντασης του ήχου.
  - `generate`: για παραγωγή ήχου.

  Αναλυτικές προδιαγραφές για την κάθε υποεντολή δίνονται στην συνέχεια.
- **Περιορισμοί Μνήμης**. Το πρόγραμμά σας πρέπει να χρησιμοποιεί λιγότερο από 8MB μνήμης ανεξαρτήτως της εντολής ή αρχείου ήχου που δίνεται.
- **Περιορισμοί Χρόνου**. Το πρόγραμμά σας πρέπει να μπορεί να διαχειρίζεται αρχεία των 10MB σε λιγότερο από 1 δευτερόλεπτο.
- **Περιορισμοί Εισόδου**. Το πρόγραμμά σας πρέπει να μπορεί να διαχειρίζεται ακολουθίες ήχου wav *οποιουδήποτε μεγέθους*. Το κάθε byte εισόδου θα μπορεί να διαβαστεί από την πρότυπη είσοδο *μόνο μία φορά* (δεν θα είναι εφικτή η χρήση `fseek`) χρησιμοποιώντας την γνωστή μας `getchar`.
- Απαγορεύεται ρητά η χρήση δομών (structs) σε αυτήν την εργασία.

Η Υποεντολή info (40% της βαθμολογίας) Όταν το πρόγραμμά σας κληθεί με την υποεντολή `info` ως πρώτο όρισμα, δηλαδή ως `./soundwave info` θα πρέπει να ελέγχει αν τα δεδομένα που διαβάστηκαν ακολουθούν το πρότυπο wav που περιγράφηκε παραπάνω και να εκτυπώνει τις σχετικές πληροφορίες του ήχου (χωρίς τα δεδομένα αυτά καθεαυτά). Παραδείγματα εκτέλεσης ακολουθούν:

```
$ ./soundwave info < good.wav
size of file: 133
size of format chunk: 16
WAVE type format: 1
mono/stereo: 1
sample rate: 44100
bytes/sec: 88200
block alignment: 2
bits/sample: 16
size of data chunk: 88
$ ./soundwave info < 1MB.wav
size of file: 1073210
size of format chunk: 16
WAVE type format: 1
mono/stereo: 2
sample rate: 8000
bytes/sec: 32000
block alignment: 4
bits/sample: 16
size of data chunk: 1072948
$ ./soundwave info < 2MB.wav
size of file: 2146158
size of format chunk: 16
WAVE type format: 1
mono/stereo: 2
sample rate: 16000
bytes/sec: 64000
block alignment: 4
bits/sample: 16
size of data chunk: 2145896
$ ./soundwave info < 5MB.wav
size of file: 5226758
size of format chunk: 16
WAVE type format: 1
mono/stereo: 2
sample rate: 44100
bytes/sec: 176400
block alignment: 4
bits/sample: 16
size of data chunk: 5226496
$ ./soundwave info < 10MB.wav
size of file: 10406730
size of format chunk: 16
```

```
WAVE type format: 1
mono/stereo: 2
sample rate: 44100
bytes/sec: 176400
block alignment: 4
bits/sample: 16
size of data chunk: 10406468
```

Όλα τα αρχεία που φαίνονται στα παραδείγματα της εκφώνησης της εργασίας μπορείτε να τα κατεβάσετε από το https://github.com/progintro/wav.

Οι εκτυπώσεις του προγράμματός σας θα γίνονται κανονικά στην πρότυπη έξοδο, **εκτός** αν παρουσιαστεί κάποιο σφάλμα (δείτε παρακάτω) οπότε και θα πρέπει να τυπώσετε στο πρότυπο ρεύμα εξόδου σφάλματος (stderr / standard error). Για να τυπώσετε στο stderr, μπορείτε να χρησιμοποιήσετε την συνάρτηση `fprintf` με πρώτο όρισμα το `stderr` εξής:

```
fprintf(stderr, <ίδια ακριβώς ορίσματα με αυτά της printf>);
```

Το πρόγραμμά σας πρέπει να είναι σε θέση να καταλαβαίνει λάθη που υπάρχουν στα δεδομένα ήχου που διαβάστηκαν και να σταματά στο πρώτο λάθος που βρίσκει εκτυπώνοντας το κατάλληλο διαγνωστικό μήνυμα στο stderr. Παραδείγματα εκτέλεσης σε προβληματικά δεδομένα εισόδου είναι τα ακόλουθα:

```
$ ./soundwave info < bad_riff.wav
Error! "RIFF" not found
$ echo $?
1
$ ./soundwave info < bad_riff.wav 2> error.txt
$ cat error.txt
Error! "RIFF" not found
$ ./soundwave info < bad_wave.wav
size of file: 133
Error! "WAVE" not found
$ ./soundwave info < bad_fmt.wav
size of file: 133
Error! "fmt " not found
$ ./soundwave info < bad_sfc.wav
size of file: 133
size of format chunk: 18
Error! size of format chunk should be 16
$ ./soundwave info < bad_wtf.wav
size of file: 133
size of format chunk: 16
WAVE type format: 2
Error! WAVE type format should be 1
```

```
$ ./soundwave info < bad_ms.wav
size of file: 133
size of format chunk: 16
WAVE type format: 1
mono/stereo: 3
Error! mono/stereo should be 1 or 2
$ ./soundwave info < bad_bys.wav
size of file: 133
size of format chunk: 16
WAVE type format: 1
mono/stereo: 1
sample rate: 44100
bytes/sec: 88202
block alignment: 2
Error! bytes/second should be sample rate x block alignment
$ ./soundwave info < bad_bis.wav
size of file: 133
size of format chunk: 16
WAVE type format: 1
mono/stereo: 1
sample rate: 44100
bytes/sec: 88200
block alignment: 2
bits/sample: 32
Error! bits/sample should be 8 or 16
$ ./soundwave info < bad_ba.wav
size of file: 133
size of format chunk: 16
WAVE type format: 1
mono/stereo: 2
sample rate: 44100
bytes/sec: 88200
block alignment: 2
bits/sample: 16
Error! block alignment should be bits per sample / 8 x mono/stereo
$ ./soundwave info < bad_data.wav
size of file: 133
size of format chunk: 16
WAVE type format: 1
mono/stereo: 1
sample rate: 44100
bytes/sec: 88200
block alignment: 2
```

```
bits/sample: 16
Error! "data" not found
$ ./soundwave info < bad_insfd.wav
size of file: 133
size of format chunk: 16
WAVE type format: 1
mono/stereo: 1
sample rate: 44100
bytes/sec: 88200
block alignment: 2
bits/sample: 16
size of data chunk: 88
Error! insufficient data
$ ./soundwave info < bad_sf.wav
size of file: 128
size of format chunk: 16
WAVE type format: 1
mono/stereo: 1
sample rate: 44100
bytes/sec: 88200
block alignment: 2
bits/sample: 16
size of data chunk: 88
Error! bad file size (found data past the expected end of file)
```

Στο τελευταίο παράδειγμα εκτέλεσης, παρατηρούμε ότι το `bad_sf.wav` αρχείο ισχυρίζεται ότι έχει μέγεθος 128 bytes (σύνολο 136 με τα αρχικά 8 bytes της κεφαλίδας), όμως αν ελέγξουμε τα συνολικά bytes του αρχείου:

```
$ wc -c ./bad_sf.wav
141 ./bad_sf.wav
```

παρατηρούμε ότι έχει και άλλα bytes στο τέλος τα οποία δεν συμπεριλαμβάνονται στο SizeOfFile και επομένως δεν περνάει τον έλεγχο ορθότητας (validation). Αντίστοιχα, μπορείτε να φτιάξετε και εσείς τα δικά σας παραδείγματα με λανθασμένα wav προκειμένου να ελέγξετε ότι το πρόγραμμά σας τα αναγνωρίζει επιτυχώς.

Η υποεντολή rate (40% της βαθμολογίας) Τώρα που το πρόγραμμά μας μπορεί να αναγνωρίζει αρχεία wav, ήρθε η ώρα να προσθέσουμε την πρώτη sound-editing υποεντολή μας, την υποεντολή `rate`, της οποίας η σύνταξη είναι `./soundwave rate fp_rate`, όπου `fp_rate` είναι ένας αριθμός κινητής υποδιαστολής (`double`) που αντιστοιχεί στην επιτάχυνση που επιθυμούμε στον ρυθμό αναπαραγωγής. Η υποεντολή `rate`, εκτός από τον έλεγχο ορθότητας των δεδομένων εισόδου θέλουμε να μεταφέρει στην έξοδο, με τη βοήθεια της συνάρτησης `putchar`, τα δεδομένα του ήχου, έχοντας αλλάξει την ταχύτητα αναπαραγωγής του, χωρίς να

αλλοιώσει τα δείγματα που αποτελούν τον ψηφιοποιημένο ήχο.

Αν το υλοποιήσουμε σωστά, καλώντας το πρόγραμμά μας ως `./soundwave rate 0.5 < input.wav > output.wav` περιμένουμε το `output.wav` αρχείο να αναπαράγεται με υποδιπλάσιο ρυθμό, δηλαδή στον διπλάσιο χρόνο από τον αρχικό. Σκεφτείτε ποιες παράμετροι των δεδομένων εισόδου θα πρέπει να αλλάξουν και πώς, ώστε να επιτευχθεί το ζητούμενο. Σε κάθε περίπτωση, η έξοδος του προγράμματος πρέπει να είναι μία νόμιμη ακολουθία από δεδομένα wav. Ακόμα και αν υπάρχουν στην είσοδο επιπλέον bytes μετά το τμήμα των δεδομένων του (OtherData), θα πρέπει και αυτά να μεταφερθούν στην έξοδο, προφανώς αυτούσια. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ ./soundwave info < LaughEvil.wav
size of file: 329266
size of format chunk: 16
WAVE type format: 1
mono/stereo: 1
sample rate: 44100
bytes/sec: 88200
block alignment: 2
bits/sample: 16
size of data chunk: 329230
$ ./soundwave rate 0.5 < LaughEvil.wav > LaughEvil2.wav
$ ./soundwave info < LaughEvil2.wav
size of file: 329266
size of format chunk: 16
WAVE type format: 1
mono/stereo: 1
sample rate: 22050
bytes/sec: 44100
block alignment: 2
bits/sample: 16
size of data chunk: 329230
$ ./soundwave rate 0.5 < ThePinkPantherTheme.wav > ThePinkPantherTheme2.wav
$ ./soundwave rate 0.5 < la.wav > la2.wav
$ ./soundwave rate 0.5 < 8bitstereo.wav > 8bitstereo2.wav
```

Στα παραπάνω παραδείγματα, η έξοδος του προγράμματος ανακατευθύνεται σε ένα νέο αρχείο ήχου. Για να τα ακούσετε, κάνετε διπλό κλικ στο αρχείο ή χρησιμοποιήστε μια εντολή όπως η `play` (Linux) ή η `afplay` (MacOS). Σε Linux μπορεί να αποφευχθεί η δημιουργία αρχείου με σωλήνωση, για παράδειγμα για ένα απομακρυσμένο αρχείο ήχου:

```
curl -s https://raw.githubusercontent.com/progintro/wav/refs/heads/main/la.wav |
  ./soundwave rate 0.5 | play -
```

Αντίστοιχα, μπορείτε να χρησιμοποιήσετε την παράμετρο της υποεντολής `rate` προκειμένου να

διπλασιάσετε την ταχύτητα αναπαραγωγής του. Διπλασιάζοντας την ταχύτητα αναπαραγωγής (2x speedup), ο χρόνος αναπαραγωγής θα υποδιπλασιασθεί, αφού τα δείγματα που αποτελούν τον ψηφιοποιημένο ήχο θα πρέπει και πάλι να μεταφερθούν αναλλοίωτα. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ ./soundwave info < LaughSarc.wav
size of file: 1541540
size of format chunk: 16
WAVE type format: 1
mono/stereo: 2
sample rate: 48000
bytes/sec: 192000
block alignment: 4
bits/sample: 16
size of data chunk: 1541504
$ ./soundwave rate 2 < LaughSarc.wav > LaughSarc3.wav
$ ./soundwave info < LaughSarc3.wav
size of file: 1541540
size of format chunk: 16
WAVE type format: 1
mono/stereo: 2
sample rate: 96000
bytes/sec: 384000
block alignment: 4
bits/sample: 16
size of data chunk: 1541504
$ ./soundwave rate 2.0 < ThePinkPantherTheme.wav > ThePinkPantherTheme3.wav
$ ./soundwave rate 2.0 < narc.wav > narc3.wav
$ ./soundwave rate 2.0 < narc.wav | ./soundwave info
size of file: 4956196
size of format chunk: 16
WAVE type format: 1
mono/stereo: 2
sample rate: 88200
bytes/sec: 352800
block alignment: 4
bits/sample: 16
size of data chunk: 4956160
$ ./soundwave rate 2.0 < 8bitmono.wav | tee 8bitmono3.wav | ./soundwave info
size of file: 44136
size of format chunk: 16
WAVE type format: 1
mono/stereo: 1
sample rate: 88200
```

```
bytes/sec: 88200
block alignment: 1
bits/sample: 8
size of data chunk: 44100
```

Η υποεντολή channel (20% της βαθμολογίας) Η υποεντολή `channel` επιτρέπει στον χρήστη να διαλέξει να αναπαράξει δεδομένα μόνο από το αριστερό ή δεξί κανάλι ήχου στην περίπτωση που ο ήχος της εισόδου είναι στερεοφωνικός. Η σύνταξη της εντολής είναι ως εξής: `./soundwave channel left` για να κρατήσει μόνο τα δεδομένα από το αριστερό κανάλι και `./soundwave channel right` για να κρατήσει μόνο τα δεδομένα από το δεξί κανάλι. Όπως προηγουμένως, το πρόγραμμά σας πρέπει να πραγματοποιεί έλεγχο ορθότητας των δεδομένων εισόδου και μεταφέρει στην έξοδο, με τη βοήθεια της συνάρτησης `putchar`, τα δεδομένα μόνο του καναλιού του ήχου που ζήτησε ο χρήστης. Αν ο ήχος στην είσοδο είναι μονοφωνικός, να μην προκύπτει σφάλμα, αλλά απλώς να μεταφέρεται στην έξοδο το μοναδικό κανάλι της εισόδου. Σκεφτείτε ποιες παράμετροι των δεδομένων εισόδου θα πρέπει να αλλάξουν και πώς και ποια από τα bytes του τμήματος δεδομένων της εισόδου θα πρέπει να μεταφερθούν στην έξοδο, ώστε να επιτευχθεί το ζητούμενο. Σε κάθε περίπτωση, η έξοδος του προγράμματος πρέπει να είναι μία νόμιμη ακολουθία από δεδομένα που συνιστούν έναν ήχο με βάση το πρότυπο που έχει περιγραφεί. Ακόμα και αν υπάρχουν στην είσοδο επιπλέον bytes μετά το τμήμα των δεδομένων του (OtherData), θα πρέπει και αυτά να μεταφερθούν στην έξοδο, προφανώς αυτούσια. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ ./soundwave info < 8bitstereo.wav
size of file: 176436
size of format chunk: 16
WAVE type format: 1
mono/stereo: 2
sample rate: 44100
bytes/sec: 88200
block alignment: 2
bits/sample: 8
size of data chunk: 176400
$ ./soundwave channel left < 8bitstereo.wav | tee 8bitstereo4.wav | ./soundwave
 ↪  info
size of file: 88236
size of format chunk: 16
WAVE type format: 1
mono/stereo: 1
sample rate: 44100
bytes/sec: 44100
block alignment: 1
bits/sample: 8
size of data chunk: 88200
```

```
$ ./soundwave channel left < CaliforniaDreamin.wav > CaliforniaDreamin4.wav
$ ./soundwave channel right < 8bitstereo.wav > 8bitstereo5.wav
$ ./soundwave channel right < CaliforniaDreamin.wav > CaliforniaDreamin5.wav
```

Η υποεντολή volume (Bonus 20% της βαθμολογίας) Η υποεντολή `volume` δέχεται έναν πολλαπλασιαστή για την ένταση του ήχου με την σύνταξη `./soundwave volume fp_multiplier` και παράγει ένα καινούριο αρχείο ήχου όπου η ένταση κάθε δείγματος πολλαπλασιάζεται με το καινούριο volume: trunc$(original \cdot fp_volume)$ (όπου trunc ο τελεστής αποκοπής σε ακέραιο). Για παράδειγμα, αν το πρόγραμμά σας κληθεί ως εξής: `./soundwave volume 0.125` περιμένουμε ότι το αρχείο ήχου εξόδου θα έχει το 1/8 της έντασης του αρχικού αρχείου. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ ./soundwave volume 0.125 < 8bitmono.wav > 8bitmono6.wav
$ ./soundwave volume 0.125 < 8bitstereo.wav > 8bitstereo6.wav
$ ./soundwave volume 0.125 < la.wav | tee la6.wav | ./soundwave info
size of file: 88236
size of format chunk: 16
WAVE type format: 1
mono/stereo: 1
sample rate: 44100
bytes/sec: 88200
block alignment: 2
bits/sample: 16
size of data chunk: 88200
```

Η υποεντολή generate (Bonus 30% της βαθμολογίας) Οι παραπάνω εντολές επιτρέπουν στο πρόγραμμά μας να επεξεργάζεται υπάρχοντα αρχεία ήχου. Τι κάνουμε όμως αν εμείς θέλουμε να παράξουμε τους δικούς μας ήχους; Σε αυτό ακριβώς το πρόβλημα έρχεται να δώσει λύση η υποεντολή `generate`. Η σύνταξη της εντολής generate είναι `./soundwave generate` και παράγει στην έξοδο δεδομένα τύπου wav, με βάση τον εξής μαθηματικό τύπο (το π είναι το γνωστό 3.14... και με το sin συμβολίζεται η συνάρτηση του ημιτόνου):

$$f(t) = \text{trunc}(amp \cdot \sin(2\pi f_c t - m_i \cdot \sin(2\pi f_m t)))$$

Για το σκοπό αυτό, ορίστε μία συνάρτηση C με πρωτότυπο το

```c
void mysound(int dur, int sr, double fm, double fc, double mi, double amp);
```

η οποία να καλείται με κατάλληλες τιμές στα ορίσματά της από τη `main` συνάρτηση, ώστε να παράγει τον κατάλληλο ήχο με βάση τον μαθηματικό τύπο που δόθηκε. Η υποεντολή, πέρα από τον βασικό τρόπο με τον οποίο καλείται (χωρίς άλλα ορίσματα) πρέπει να μπορεί να υποστηρίζει τα ακόλουθα προαιρετικά ορίσματα για τον καθορισμό των παραμέτρων:

- `--dur int_duration`: Το όρισμα dur είναι η διάρκεια του ήχου σε δευτερόλεπτα (ακέραιος). Default: 3.
- `--sr int_sr`: Το sr είναι το *SampleRate* για την ψηφιοποίηση του ήχου (ακέραιος). Default: 44100.
- `--fm fp_fm`: Το frequency modulation που χρησιμοποιείται στο σήμα f(t) (κινητής υποδιαστολής). Default: 2.0.
- `--fc fp_fc`: Το carrier frequency που χρησιμοποιείται στο σήμα f(t) (κινητής υποδιαστολής). Default: 1500.0.
- `--mi fp_mi`: Το modulation index που χρησιμοποιείται στο σήμα f(t) (κινητής υποδιαστολής). Default: 100.0.
- `--amp fp_amp`: Το amplitude (δηλαδή η ένταση) που χρησιμοποιείται στο σήμα f(t) (κινητής υποδιαστολής). Default: 30000.0.

Οι παραπάνω παράμετροι πρέπει να τηρούνται πλήρως κατά την παραγωγή του ηχητικού σήματος. Για το π μπορείτε να χρησιμοποιήσετε τη συμβολική σταθερά `M_PI`, που είναι ορισμένη στο `math.h`. Ο ήχος που θα παραχθεί να είναι μονοφωνικός (*MonoStereo* = 1) και να αναπαρίσταται με 2 bytes ανά δείγμα (*BitsPerSample* = 16). Καλέστε το πρόγραμμά σας με τις default παραμέτρους. Τι ήχος παράγεται; Ενδεικτική εκτέλεση:

```
$ ./soundwave generate > mysound.wav
$ ./soundwave info < mysound.wav
size of file: 264636
size of format chunk: 16
WAVE type format: 1
mono/stereo: 1
sample rate: 44100
bytes/sec: 88200
block alignment: 2
bits/sample: 16
size of data chunk: 264600
./soundwave generate --dur 5 --fm 1.2 --fc 300 --mi 100 --amp 15000 > space.wav
```

Διαγωνισμός Δημιουργικότητας και Τελική Υποβολή Σας συνιστούμε να πειραματιστείτε και με άλλες δυνατότητες—για παράδειγμα την προσθήκη ορισμάτων `--help` ώστε να μπορεί να βρούμε μια γρήγορη περιγραφή των διαφόρων δυνατοτήτων του προγράμματός σας. Για να λάβετε συμμετοχή στον διαγωνισμό δημιουργικότητας στο τέλος του εξαμήνου, μπορείτε να υλοποιήσετε μια δική σας υποεντολή `./soundwave dj` (με όποια ορίσματα θέλετε) η οποία θα είναι σε θέση να παράξει ενδιαφέροντες και δημιουργικούς ήχους! Αν κάποια υποβολή μας εκπλήξει, θα λάβει επιπλέον bonus μέχρι 50% της βαθμολογίας.

Τέλος, πριν την υποβολή της εργασίας, μην ξεχάσετε το `soundwave/README.md`, μέσα στο οποίο πρέπει να συμπεριλάβετε τις όποιες παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.

Υπόδειξη Γράψε πρώτα βοηθητικές συναρτήσεις που διαβάζουν (και γράφουν) έναν ακέραιο 2 ή 4 bytes little-endian με `getchar`/`putchar`, συνθέτοντας τα bytes με ολισθήσεις και `|`, και έλεγξε παντού για `EOF`. Επειδή δεν χωράει όλο το αρχείο στη μνήμη και δεν επιτρέπονται δομές, επεξεργάσου τα δείγματα σε ροή, ένα-ένα, και σκέψου ποια πεδία της κεφαλίδας αλλάζουν σε κάθε υποεντολή (π.χ. στο `channel` αλλάζουν τα μεγέθη, όχι μόνο τα δείγματα). Πρόσεξε ότι τα δείγματα των 8 bits είναι χωρίς πρόσημο ενώ των 16 bits με πρόσημο, κάτι που μετράει στο `volume`, και ότι η `generate` πρέπει να ξέρει εκ των προτέρων το *SizeOfData* για να γράψει την κεφαλίδα.

Θέματα εξετάσεων (A9.18–A9.31)

A9.18 · Ραβασάκι

A9.18 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #0 (Valentine's Themed), Θέμα 1* · ★☆☆ · programming · `exam-2023-fall-ex0-q1`

Πρόγραμμα: `ravasaki.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που διαβάζει το κείμενο που δίνεται από την πρότυπη είσοδο (stdin) και το τυπώνει στην πρότυπη έξοδο (stdout) προσθέτοντας μια κόκκινη καρδιά στο τέλος κάθε γραμμής. Παράδειγμα εκτέλεσης:

```
$ cat maravegias.txt
Και τυχαίο και γραφτό,
το μεγάλο μωρό που παίζει με τόξο και βέλη
που χτυπάει και φεύγει όλα τα ανατρέπει,
μα ο χάρτης τον δρόμο σου δείχνει
Του κρυμμένου θησαυρού το ταξίδι τελειώνει,
εκεί που είναι αδύνατο να ‘μαστε μόνοι
Σε κοιτάζω στα μάτια σαν καθρέφτες γυαλίζουν,
τον πιο ωραίο εαυτό μας αντικατοπτρίζουν
$ gcc -o ravasaki ravasaki.c
$ ./ravasaki < maravegias.txt
Και τυχαίο και γραφτό, <3
το μεγάλο μωρό που παίζει με τόξο και βέλη <3
που χτυπάει και φεύγει όλα τα ανατρέπει, <3
μα ο χάρτης τον δρόμο σου δείχνει <3
Του κρυμμένου θησαυρού το ταξίδι τελειώνει, <3
εκεί που είναι αδύνατο να ‘μαστε μόνοι <3
Σε κοιτάζω στα μάτια σαν καθρέφτες γυαλίζουν, <3
τον πιο ωραίο εαυτό μας αντικατοπτρίζουν <3
```

> Σημείωση: στο πρωτότυπο το παράδειγμα είναι στιγμιότυπο τερματικού (εικόνα), όπου το `<3` στο τέλος κάθε γραμμής εμφανίζεται με κόκκινο χρώμα.

Υπόδειξη Διαβάστε χαρακτήρα προς χαρακτήρα με `getchar` μέχρι το `EOF`: όταν συναντάτε `'\n'`, τυπώστε πρώτα την καρδιά και μετά την αλλαγή γραμμής. Για το κόκκινο χρώμα ψάξτε τις ANSI escape sequences του τερματικού (και θυμηθείτε να επαναφέρετε το χρώμα). Σκεφτείτε τι γίνεται αν η τελευταία γραμμή δεν τελειώνει σε `'\n'`.

A9.19 · Δεκαεξαδικοί

A9.19 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #1 (Coreutils Themed), Θέμα 1* · ★☆☆ · programming · `exam-2023-fall-ex1-q1`

Πρόγραμμα: `hex.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που διαβάζει το κείμενο που δίνεται από την πρότυπη είσοδο (stdin) και τυπώνει τον κάθε χαρακτήρα στην πρότυπη έξοδο (stdout) σε δεκαεξαδική μορφή χωρίζοντάς το σε γραμμές. Σε κάθε γραμμή πρέπει να τυπώνονται μέχρι 16 χαρακτήρες σε δεκαεξαδική μορφή και πρέπει να υπάρχει ένας κενός χαρακτήρας (' ') ανάμεσα σε κάθε χαρακτήρα στην ίδια γραμμή. Παράδειγμα εκτέλεσης:

```
$ cat domo.txt
□□□□□□□□ Mr. Roboto□
You're wondering who I am (secret, secret, I've got a secret)
Machine or mannequin? (Secret, secret, I've got a secret)
$ gcc -o hex hex.c
$ ./hex < domo.txt
 e3 81 a9 e3 81 86 e3 82 82 e3 81 82 e3 82 8a e3
 81 8c e3 81 a8 20 4d 72 2e 20 52 6f 62 6f 74 6f
 ff 0a 59 6f 75 27 72 65 20 77 6f 6e 64 65 72 69
 6e 67 20 77 68 6f 20 49 20 61 6d 20 28 73 65 63
 72 65 74 2c 20 73 65 63 72 65 74 2c 20 49 27 76
 65 20 67 6f 74 20 61 20 73 65 63 72 65 74 29 0a
 4d 61 63 68 69 6e 65 20 6f 72 20 6d 61 6e 6e 65
 71 75 69 6e 3f 20 28 53 65 63 72 65 74 2c 20 73
 65 63 72 65 74 2c 20 49 27 76 65 20 67 6f 74 20
 61 20 73 65 63 72 65 74 29 0a
```

Το αρχείο εισόδου: domo.txt.

Υπόδειξη Διαβάστε byte προς byte με `getchar` και τυπώστε το καθένα με `printf` και την κατάλληλη μορφή για δύο δεκαεξαδικά ψηφία. Ένας μετρητής σάς λέει πότε συμπληρώθηκαν 16 και χρειάζεται αλλαγή γραμμής· προσέξτε στο παράδειγμα πού ακριβώς μπαίνουν τα κενά και τι γίνεται με την τελευταία, ημιτελή γραμμή. Το byte `ff` δείχνει ότι η είσοδος δεν είναι απαραίτητα έγκυρο κείμενο.

A9.20 · Γραμμή και Γράμμα

A9.20 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #12, Θέμα 1* · ★☆☆ · programming · `exam-2023-fall-ex12-q1`

Πρόγραμμα: `letter.c`

Γράψτε ένα πρόγραμμα το οποίο διαβάζει από την πρότυπη είσοδο (stdin) μια σειρά από γραμμές που περιέχουν τα γράμματα μιας λέξης (ένα γράμμα ανά γραμμή με χρήση underscore '_' για όλα τα γράμματα που δεν αποκαλύπτονται σε εκείνη την γραμμή) και τυπώνει στην πρότυπη έξοδο (stdout) την λέξη που δόθηκε. Η είσοδος θα αποτελείται μόνο από λατινικούς (A-Za-z) χαρακτήρες, underscores και αλλαγές γραμμής. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ gcc -o letter letter.c
$ cat reveal.txt
__r_____
_______l
_a______
___n____
_____v__
c_______
______a_
____i___
$ ./letter < reveal.txt
carnival
```

Υπόδειξη Κάθε γραμμή έχει το ίδιο μήκος με τη λέξη και αποκαλύπτει ένα γράμμα στη θέση του. Διαβάστε χαρακτήρα-χαρακτήρα κρατώντας τη στήλη στην οποία βρίσκεστε και γράψτε κάθε γράμμα στη θέση του σε έναν πίνακα. Ελέγξτε για χαρακτήρες εκτός προδιαγραφών και γραμμές διαφορετικού μήκους.

A9.21 · Ορεκτικό

A9.21 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #14, Θέμα 1* · ★☆☆ · programming · `exam-2023-fall-ex14-q1`

Πρόγραμμα: `steak.c`

Γράψτε ένα πρόγραμμα που διαβάζει από την πρότυπη είσοδο (stdin) ένα κείμενο και το τυπώνει στην πρότυπη έξοδο (stdout) αφού πρώτα προσθέσει μια μπριζόλα (χαρακτήρας unicode U+1F969) στην αρχή και στο τέλος κάθε γραμμής. Το υπόλοιπο της πρότασης δεν πρέπει να αλλάζει. Το πρόγραμμά σας πρέπει να χειρίζεται γραμμές οποιουδήποτε μήκους. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ cat message.txt
```

```
Come savor the pleasure, of treasures beyond measure,
In our cozy abode, where gourmet is the code<U+FFFD>
We await your presence, for an experience quintessence,
At our steakhouse divine, where dining is fine!
$ gcc -o steak steak.c
$ ./steak  < message.txt
<U+0000>Come savor the pleasure, of treasures beyond measure,<U+0000>
<U+0000>In our cozy abode, where gourmet is the code<U+FFFD><U+0000>
<U+0000>We await your presence, for an experience quintessence,<U+0000>
<U+0000>At our steakhouse divine, where dining is fine!<U+0000>
```

> Σημείωση: στο πρωτότυπο το παράδειγμα δίνεται ως στιγμιότυπο τερματικού (steak.png) και εδώ μεταγράφεται ως κείμενο· ο χαρακτήρας <U+FFFD> είναι ένα byte του αρχείου που δεν είναι έγκυρο UTF-8.

Υπόδειξη Δεν χρειάζεται να αποθηκεύσετε γραμμές: με `getchar` τυπώστε τη μπριζόλα στην αρχή κάθε γραμμής και πριν από κάθε `'\n'`. Ο χαρακτήρας U+1F969 στο UTF-8 είναι μια ακολουθία 4 bytes που μπορείτε να γράψετε ως string. Σκεφτείτε την τελευταία γραμμή χωρίς `'\n'` και την κενή είσοδο.

Α9.22 · Πλαγιαστά Γράμματα

A9.22 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #15, Θέμα 1* · ★☆☆ · programming · `exam-2023-fall-ex15-q1`

Πρόγραμμα: `italics.c`

Γράψτε ένα πρόγραμμα που διαβάζει από την πρότυπη είσοδο (stdin) ένα κείμενο και το τυπώνει στην πρότυπη έξοδο (stdout) αφού πρώτα μετατρέψει τα γράμματα εντός των αστερίσκων (*) σε πλαγιαστά γράμματα (italics). Τα γράμματα που βρίσκονται εκτός αστερίσκων δεν χρειάζεται να αλλαχθούν. Οι αστερίσκοι θα παραλείπονται στην έξοδο του προγράμματος. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ cat ithaca.txt
As you set out for Ithaka
hope your road is a *long one*,
full of adventure, *full of discovery*.
Laistrygonians, Cyclops,
angry Poseidon *do not be afraid of them*.
$ gcc -o italics italics.c
$ ./italics < ithaca.txt
As you set out for Ithaka
hope your road is a long one,
full of adventure, full of discovery.
```

```
Laistrygonians, Cyclops,
angry Poseidon do not be afraid of them.
```

> Σημείωση: στο πρωτότυπο το παράδειγμα δίνεται ως στιγμιότυπο τερματικού (italics.png) και εδώ μεταγράφεται ως κείμενο: στην έξοδο τα κείμενα «long one», «full of discovery» και «do not be afraid of them» εμφανίζονται πλάγια.

Υπόδειξη Κρατήστε μια σημαία «είμαι μέσα σε αστερίσκους» που αλλάζει σε κάθε *, και τυπώστε τους υπόλοιπους χαρακτήρες όπως είναι. Το «έντονο» (και αντίστοιχα το πλάγιο ή το αναβόσβημα) στο τερματικό γίνεται με ακολουθίες διαφυγής ANSI (ANSI escape codes) που τυπώνετε πριν και μετά το κείμενο, π.χ. `\033[...m`· ψάξτε ποιος κωδικός αντιστοιχεί στο εφέ που θέλετε και ποιος τα επαναφέρει.

A9.23 · Μήνυμα από τον Καίσαρα

A9.23 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #6 (Crypto Themed), Θέμα 2* · ★☆☆ · programming · `exam-2023-fall-ex6-q2`

Πρόγραμμα: `caesar.c` (25 μονάδες)

Υπάρχει ένας τρόπος κρυπτογράφησης που αποδίδεται στον Καίσαρα (Caesar Cipher) όπου κάθε γράμμα της αλφαβήτου αντιστοιχίζεται σε ένα άλλο. Για παράδειγμα με μετάθεση γραμμάτων κατά 3 παίρνουμε την ακόλουθη αντιστοίχιση:

Εικόνα: Αντιστοίχιση γραμμάτων

Όπου το A αντιστοιχεί στο Z, το D στο A, κοκ. Γράψτε ένα πρόγραμμα που διαβάζει το κείμενο που δίνεται από την πρότυπη είσοδο (stdin) και το τυπώνει στην πρότυπη έξοδο (stdout) αφού το περάσει από κρυπτογράφηση με την μέθοδο του Καίσαρα όπου η μετατόπιση είναι 13, δηλαδή το A αντιστοιχεί στο N, το B αντιστοιχεί στο O κοκ. Το πρόγραμμα πρέπει να αντικαθιστά μόνο λατινικούς χαρακτήρες (A-Z, a-z) και όλοι οι υπόλοιποι πρέπει να μένουν ίδιοι. Παράδειγμα εκτέλεσης:

```
$ gcc -o caesar caesar.c
$ cat toto.txt
Vg'f tbaan gnxr n ybg gb qent zr njnl sebz lbh🤬 ,
Gurer'f abguvat gung n uhaqerq zra be zber pbhyq rire qb ,
V oyrff gur envaf qbja va Nsevpn ,
Tbaan gnxr fbzr gvzr gb qb gur guvatf jr arire unq !
$ ./caesar < toto.txt
It's gonna take a lot to drag me away from you🤬 ,
There's nothing that a hundred men or more could ever do ,
I bless the rains down in Africa ,
Gonna take some time to do the things we never had !
```

Υπόδειξη Διαβάστε χαρακτήρα προς χαρακτήρα· για κεφαλαία και πεζά ξεχωριστά, υπολογίστε τη θέση του γράμματος στο αλφάβητο (`c - 'A'`), προσθέστε τη μετατόπιση και κάντε αναδίπλωση με `%`. Όλοι οι άλλοι χαρακτήρες, ακόμη και bytes που δεν είναι έγκυρο κείμενο, περνάνε αυτούσιοι. Αφού το αλφάβητο έχει 26 γράμματα, τι παθαίνει ένα κείμενο αν του εφαρμόσετε ROT13 δύο φορές;

A9.24 · Ανάβεις Φωτιές

A9.24 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #7 (Star Wars Themed), Θέμα 1* · ★☆☆ · programming · `exam-2023-fall-ex7-q1`

Πρόγραμμα: `fire.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που διαβάζει από την πρότυπη είσοδο (stdin) ένα κείμενο και το τυπώνει στην πρότυπη έξοδο (stdout) αφού πρώτα προσθέσει μια φωτιά 🔥 (χαρακτήρας unicode U+1F525) ακολουθούμενη από κενό στην αρχή κάθε γραμμής. Το υπόλοιπο της πρότασης δεν πρέπει να αλλάζει. Το πρόγραμμά σας πρέπει να χειρίζεται γραμμές οποιουδήποτε μήκους. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ gcc -o fire fire.c
$ cat quotes.txt
May the force be with you!
Do... or do not. There is no try.
It's a trap
I find your lack of faith disturbing.
I have a very bad feeling about this.
$ ./fire < quotes.txt
🔥 May the force be with you!
🔥 Do... or do not. There is no try.
🔥 It's a trap
🔥 I find your lack of faith disturbing.
🔥 I have a very bad feeling about this.
```

Σημείωση: στο πρωτότυπο το παράδειγμα είναι στιγμιότυπο τερματικού (εικόνα).

Υπόδειξη Αφού οι γραμμές μπορεί να έχουν οποιοδήποτε μήκος, μην τις αποθηκεύετε: διαβάστε χαρακτήρα προς χαρακτήρα με `getchar` και θυμηθείτε μόνο αν βρίσκεστε στην αρχή γραμμής. Η φωτιά τυπώνεται πριν από τον πρώτο χαρακτήρα κάθε γραμμής, όχι μετά το τελευταίο `'\n'`. Ο χαρακτήρας U+1F525 σε UTF-8 είναι τέσσερα bytes, που μπορείτε να τα γράψετε μέσα σε ένα string literal.

A9.25 · ASCII Encoding

A9.25 · *Εξέταση Ιουλίου 2024, Θέμα 1* · ★☆☆ · programming · `exam-2024-jul-q1`

Γράψτε ένα πρόγραμμα το οποίο διαβάζει όλους τους χαρακτήρες που δίνονται από την πρότυπη είσοδο (stdin) και τυπώνει στην πρότυπη έξοδο (stdout) την δεκαεξαδική αναπαράστασή τους με μία εξαίρεση: οι χαρακτήρες νέας γραμμής (newline) πρέπει να τυπώνονται ως έχουν - χωρίς μετατροπή σε δεκαεξαδική μορφή. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ cat ithaka.txt
As you set out for Ithaka
hope your road is a long one,
full of adventure, full of discovery.
$ ./ascii < ithaka.txt
417320796f7520736574206f757420666f7220497468616b61
686f706520796f757220726f6164206973206120c6f6e67206f6e652c
66756c6c206f6620616476656e747572652c2066756c6c206f6620646973636f766572792e
```

Υπόδειξη Διαβάστε την είσοδο χαρακτήρα προς χαρακτήρα μέχρι το `EOF`, προσέχοντας τον τύπο της μεταβλητής που κρατά το αποτέλεσμα της `getchar`. Η `printf` έχει έτοιμο προσδιοριστή για δεκαεξαδική μορφή· σκεφτείτε πώς εξασφαλίζετε ότι κάθε χαρακτήρας βγαίνει πάντα με δύο ψηφία (π.χ. για χαρακτήρες με κωδικό μικρότερο του 16).

A9.26 · Αφαίρεση Μη Λατινικών Χαρακτήρων

A9.26 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #10, Θέμα 1* · ★★☆ · programming · `exam-2023-fall-ex10-q1`

Πρόγραμμα: `bold.c`

Γράψτε ένα πρόγραμμα το οποίο διαβάζει κείμενο από την πρότυπη είσοδο (stdin) και το τυπώνει στην πρότυπη έξοδο (stdout) με τους εξής περιορισμούς:

1. Οι αλλαγές γραμμής πρέπει να διατηρούνται.
2. Πρέπει να μετατρέψει κάθε μη λατινικό χαρακτήρα (A-Za-z) σε κενό (space) χαρακτήρα `' '`.
3. Η πρώτη λέξη σε κάθε γραμμή θέλουμε να φαίνεται **bold**.

Παράδειγμα εκτέλεσης ακολουθεί:

```
$ gcc -o bold bold.c
$ cat island.txt
Fifteen men on the Dead Man's Chest Yo-ho-ho,
and a bottle of rum!
```

```
Drink and the devil had done for the rest Yo-ho-ho,
and a bottle of rum!
$ ./bold < island.txt
Fifteen men on the Dead Man s Chest Yo ho ho
and a bottle of rum
Drink and the devil had done for the rest Yo ho ho
and a bottle of rum
```

> Σημείωση: στο πρωτότυπο το παράδειγμα δίνεται ως στιγμιότυπο τερματικού (bold.png) και εδώ μεταγράφεται ως κείμενο: στην έξοδο οι λέξεις Fifteen, and, Drink, and (η πρώτη κάθε γραμμής) εμφανίζονται με έντονα γράμματα.

Υπόδειξη Διαβάστε με `getchar` και κρατήστε μια κατάσταση: αν βρίσκεστε ακόμη στην πρώτη λέξη της γραμμής. Μη λατινικοί χαρακτήρες (εκτός από το `'\n'`) γίνονται κενά και η αλλαγή γραμμής μηδενίζει την κατάσταση. Το «έντονο» (και αντίστοιχα το πλάγιο ή το αναβόσβημα) στο τερματικό γίνεται με ακολουθίες διαφυγής ANSI (ANSI escape codes) που τυπώνετε πριν και μετά το κείμενο, π.χ. `\033[...m`· ψάξτε ποιος κωδικός αντιστοιχεί στο εφέ που θέλετε και ποιος τα επαναφέρει.

A9.27 · Αναζητώντας τον Blinky

A9.27 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #11, Θέμα 1* · ★★☆ · programming · `exam-2023-fall-ex11-q1`

Πρόγραμμα: `blinky.c`

Γράψτε ένα πρόγραμμα το οποίο διαβάζει κείμενο από την πρότυπη είσοδο (stdin) και το τυπώνει στην πρότυπη έξοδο (stdout) με τους εξής περιορισμούς:

1. Κάθε φορά που βρίσκει την λέξη Blinky - να την τοποθετεί ανάμεσα σε αστεράκια '*' και να την κάνει να αναβοσβήνει (blink).
2. Αν το string Blinky είναι μέρος άλλης λέξης δεν χρειάζεται κάποια αλλαγή.
3. Οι λέξεις αποτελούνται μόνο από λατινικούς χαρακτήρες (A-Za-z) οποιοσδήποτε άλλος χαρακτήρας θεωρείται διαχωριστικό λέξεων.

Παράδειγμα εκτέλεσης ακολουθεί:

[Εικόνα: asciicast](https://asciinema.org/a/js20ZyFDDS0t8ocszkBBo6b4J)

Αν για κάποιο λόγο δεν παίζει το παραπάνω link:

```
$ gcc -o blinky blinky.c
$ cat description.txt
In the Pac-Man universe, Blinky is the red ghost who
persistently chases the game's hero, earning the
```

```
nickname "Shadow" for his relentless pursuit.
Blinky's strategy intensifies as Pac-Man consumes
more dots, making Blinky a formidable challenge and
a central figure in game strategy. His vivid red hue
not only marks Blinky as the primary antagonist but
also signals the immediate threat he poses.
$ ./blinky < description.txt
In the Pac-Man universe, *Blinky* is the red ghost who
persistently chases the game's hero, earning the
nickname "Shadow" for his relentless pursuit.
*Blinky*'s strategy intensifies as Pac-Man consumes
more dots, making *Blinky* a formidable challenge and
a central figure in game strategy. His vivid red hue
not only marks *Blinky* as the primary antagonist but
also signals the immediate threat he poses.
```

Υπόδειξη Μαζέψτε κάθε λέξη (μέγιστη ακολουθία λατινικών χαρακτήρων) σε έναν buffer και, όταν τελειώσει, συγκρίνετέ τη ολόκληρη με το `Blinky`· έτσι το `Blinky` μέσα σε μεγαλύτερη λέξη δεν αλλάζει. Οι διαχωριστικοί χαρακτήρες τυπώνονται αυτούσιοι. Το «έντονο» (και αντίστοιχα το πλάγιο ή το αναβόσβημα) στο τερματικό γίνεται με ακολουθίες διαφυγής ANSI (ANSI escape codes) που τυπώνετε πριν και μετά το κείμενο, π.χ. `\033[...m`· ψάξτε ποιος κωδικός αντιστοιχεί στο εφέ που θέλετε και ποιος τα επαναφέρει.

A9.28 · Pikακίστικα

A9.28 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #2 (Pokémon Themed), Θέμα 1* · ★★☆ · programming · `exam-2023-fall-ex2-q1`

Πρόγραμμα: `pika.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που διαβάζει το κείμενο που δίνεται από την πρότυπη είσοδο (stdin) και τυπώνει την κάθε λέξη χαρακτήρα στην πρότυπη έξοδο (stdout) με το πρόθεμα ”pika”. Οι λέξεις χωρίζονται με κενούς (whitespace) χαρακτήρες. Αριθμοί ή σημεία στίξης *δεν* θεωρούνται λέξεις. Παράδειγμα εκτέλεσης:

```
$ cat ash.txt
    I wanna be the very best,
    Like no one ever was ,
    To catch them is my real-test ,
    To train them is my cause !
$ ./pika  < ash.txt
    pikaI pikawanna pikabe pikathe pikavery pikabest,
    pikaLike pikano pikaone pikaever pikawas ,
```

```
    pikaTo pikacatch pikathem pikais pikamy pikareal-test ,
    pikaTo pikatrain pikathem pikais pikamy pikacause !
```

Υπόδειξη Δεν χρειάζεται να αποθηκεύσετε ολόκληρες λέξεις: διαβάστε χαρακτήρα προς χαρακτήρα και κρατήστε μια μεταβλητή κατάστασης «είμαι μέσα σε λέξη ή όχι». Το πρόθεμα μπαίνει τη στιγμή που ξεκινά μια λέξη, και τα κενά (μαζί με τα tabs) αντιγράφονται όπως είναι. Κοιτάξτε προσεκτικά στο παράδειγμα ποιος χαρακτήρας κάνει ένα token «λέξη» (π.χ. `real-test` και `best`, αλλά όχι `,` ή `!`).

Α9.29 · Προσθήκη Νιφάδων

A9.29 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #3 (Frozen Themed), Θέμα 1* · ★★☆ · programming · `exam-2023-fall-ex3-q1`

Πρόγραμμα: `snow.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που διαβάζει το κείμενο που δίνεται από την πρότυπη είσοδο (stdin) και τυπώνει την κάθε λέξη χαρακτήρα στην πρότυπη έξοδο (stdout) εμπλουτισμένη με νιφάδες, δηλαδή με τον χαρακτήρα `*`. Νιφάδες μπαίνουν μόνο ανάμεσα στα γράμματα της κάθε λέξης. Λέξεις με μόνο ένα γράμμα δεν χρειάζονται κάποια προσθήκη. Οι λέξεις αποτελούνται από τα γράμματα A-Z και a-z. Αριθμοί ή σημεία στίξης *δεν* θεωρούνται λέξεις. Παράδειγμα εκτέλεσης:

```
$ cat elsa.txt
    Let it go, let it go,
    Turn away and slam the door ,
    I don't care what they're going to say ,
    Let the storm rage on ,
    The cold never bothered me anyway !
$ gcc -o snow snow.c
$ ./snow < elsa.txt
    L*e*t i*t g*o, l*e*t i*t g*o,
    T*u*r*n a*w*a*y a*n*d s*l*a*m t*h*e d*o*o*r ,
    I d*o*n't c*a*r*e w*h*a*t t*h*e*y'r*e g*o*i*n*g t*o s*a*y ,
    L*e*t t*h*e s*t*o*r*m r*a*g*e o*n ,
    T*h*e c*o*l*d n*e*v*e*r b*o*t*h*e*r*e*d m*e a*n*y*w*a*y !
```

Υπόδειξη Διαβάστε χαρακτήρα προς χαρακτήρα και θυμηθείτε μόνο αν ο προηγούμενος χαρακτήρας ήταν γράμμα: η νιφάδα μπαίνει ακριβώς όταν δύο γράμματα είναι διαδοχικά. Κοιτάξτε στο παράδειγμα τι συμβαίνει στο `don't` και στο `they're`. Προσέξτε ότι η `isalpha` εξαρτάται από το locale, ενώ η εκφώνηση ορίζει ρητά A-Z και a-z.

A9.30 · Αποκωδικοποίηση

A9.30 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #8, Θέμα 2* · ★★☆ · programming · `exam-2023-fall-ex8-q2`

Πρόγραμμα: `decode.c`

Γράψτε ένα πρόγραμμα που διαβάζει από την πρότυπη είσοδο (stdin) ένα κωδικοποιημένο (ως δεκαεξαδικοί χαρακτήρες) κείμενο και το τυπώνει στην πρότυπη έξοδο (stdout) αποκωδικοποιημένο. Με την δεκαεξαδική κωδικοποίηση κάθε δύο δεκαεξαδικά ψηφία αντιστοιχούν σε έναν χαρακτήρα. Για παράδειγμα τα ψηφία 4b αντιστοιχούν στον δεκαεξαδικό 0x4b που αντιστοιχούν στον χαρακτήρα ascii 'K'. Η αποκωδικοποίηση του υπόλοιπου κειμένου συνεχίζει με τον ίδιο τρόπο. Οποιοσδήποτε χαρακτήρας εισόδου δεν είναι δεκαεξαδικός (a-f, A-F, 0-9) μπορεί να αγνοηθεί. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ gcc -o decode decode.c
$ cat encoded_message.txt
4b65657020796F757220666616365206 16C7761797320
746f77617264207468652073756e7368696e652c2061
6e6420736861646f77732077696c6c2066616c6c2062
6568696e6420796f752e
$ ./decode < encoded_message.txt
Keep your face always toward the sunshine, and shadows will fall behind you.
```

Υπόδειξη Διαβάστε την είσοδο χαρακτήρα-χαρακτήρα με `getchar` και αγνοήστε ό,τι δεν είναι δεκαεξαδικό ψηφίο (και τις αλλαγές γραμμής). Μετατρέψτε κάθε ψηφίο στην τιμή του (0–15) και συνδυάστε ανά δύο: το πρώτο είναι τα υψηλά 4 bits. Σκεφτείτε τι κάνετε αν στο τέλος περισσέψει ένα μισό ζεύγος.

A9.31 · Μετρητής λέξεων

A9.31 · *Εξέταση Σεπτεμβρίου 2024, Θέμα 4* · ★★☆ · programming · `exam-2024-sep-q4`

Μετρητής Λέξεων [25 Μονάδες]

Γράψτε ένα πρόγραμμα `wordcount` το οποίο διαβάζει ένα κείμενο από την πρότυπη είσοδο (stdin) και τυπώνει στην πρότυπη έξοδο (stdout) τον αριθμό των λέξεων που διάβασε καθώς και το μέσο μήκος λέξης του κειμένου με ακρίβεια δύο δεκαδικών ψηφίων. Ως *λέξη*, ορίζουμε οποιαδήποτε ακολουθία συνεχόμενων χαρακτήρων A-Z, a-z και 0-9 (μετράμε μόνο λέξεις με λατινικούς χαρακτήρες) και ως *μήκος* τον αριθμό των συνεχόμενων χαρακτήρων. Για παράδειγμα, το κείμενο ”I'm a good person” έχει σύνολο 5 λέξεις και μέσο μήκος (1 + 1 + 1 + 4 + 6)/5 = 2.6 χαρακτήρες. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ echo "I'm a good person" | ./wordcount
Total words: 5
```

```
Average word length: 2.60
```

```
$ cat quote.txt
"Success is not final, failure is not fatal: It is the courage to continue that
 ↪  counts."
$ ./wordcount < quote.txt
Total words: 16
Average word length: 4.25
```

Υπόδειξη Διαβάστε χαρακτήρα-χαρακτήρα μέχρι το `EOF` και κρατήστε μια κατάσταση «είμαι μέσα σε λέξη ή όχι»: μια λέξη ξεκινά όταν περνάτε από μη αλφαριθμητικό σε αλφαριθμητικό χαρακτήρα. Δεν χρειάζεται να αποθηκεύσετε το κείμενο, μόνο μετρητές. Προσέξτε την ακέραια διαίρεση στον μέσο όρο και τι θα τυπώσετε αν η είσοδος δεν έχει καμία λέξη.

Κεφάλαιο 10: Πίνακες

Kahoot από το αμφιθέατρο (K10.1–K10.10)

K10.1 · Πρόσβαση εκτός ορίων

K10.1 · *Kahoot «Πίνακες» (διάλεξη 10)* · ★☆☆ · multiple-choice · 85% σωστές · `kahoot-out-of-bounds`

Δεν υπάρχει πρόβλημα αν προσπελάσω στοιχεία εκτός των ορίων ενός πίνακα.

- True
- False

Υπόδειξη Ελέγχει η C τα όρια ενός πίνακα; Τι υπάρχει στη μνήμη αμέσως μετά το τελευταίο στοιχείο;

K10.2 · Αντιγραφή πίνακα με ανάθεση

K10.2 · *Kahoot «Πίνακες» (διάλεξη 10)* · ★☆☆ · multiple-choice · 84% σωστές · `kahoot-array-assignment`

Ο πιο αποδοτικός τρόπος για να αντιγράψω έναν πίνακα 100 στοιχείων είναι με ανάθεση:

```c
int a[100];
int b[100];
b = a;
```

- True

- False

Υπόδειξη Μπορεί ένα όνομα πίνακα να βρεθεί στα αριστερά μιας ανάθεσης;

K10.3 · Ο δείκτης του πρώτου στοιχείου

K10.3 · *Kahoot «Πίνακες» (διάλεξη 10)* · ★☆☆ · multiple-choice · 82% σωστές · `kahoot-first-index`

Το πρώτο στοιχείο ενός πίνακα με 10 στοιχεία βρίσκεται στη θέση (index):

- 1
- 10
- 9
- 0

Υπόδειξη Ο δείκτης μετράει πόσα στοιχεία απέχουμε από την αρχή του πίνακα.

K10.4 · Το τελευταίο στοιχείο πίνακα

K10.4 · *Kahoot «Πίνακες» (διάλεξη 10)* · ★☆☆ · multiple-choice · 81% σωστές · `kahoot-last-element`

Το τελευταίο στοιχείο ενός πίνακα με ορισμό `int students[100];` είναι:

- `students[100]`
- `students[101]`
- `students[99]`
- `students[-1]`

Υπόδειξη Αν η αρίθμηση ξεκινά από το 0, ποιος είναι ο τελευταίος από 100 διαδοχικούς δείκτες;

K10.5 · Αρχικοποίηση με περισσότερες τιμές

K10.5 · *Kahoot «Πίνακες» (διάλεξη 10)* · ★☆☆ · multiple-choice · 71% σωστές · `kahoot-excess-initializers`

Μπορούμε να αρχικοποιήσουμε έναν πίνακα με 10 ακεραίους ως εξής:

```
int numbers[10] = {0,1,2,3,4,5,6,7,8,9,10};
```

- True

- False

Υπόδειξη Μετρήστε πόσες τιμές έχει η λίστα αρχικοποίησης και συγκρίνετε με το μέγεθος του πίνακα.

K10.6 · Δείκτης μέσα σε δείκτη

K10.6 · *Kahoot «Πίνακες» (διάλεξη 10)* · ★★☆ · multiple-choice · 69% σωστές · `kahoot-nested-index`

Τι θα τυπώσει ο παρακάτω κώδικας;

```
int a[] = {5, 4, 3, 1, 2, 6};
printf("%d\n", a[ a[2] ]);
```

- 1
- 4
- 3
- 6

Υπόδειξη Υπολογίστε πρώτα την εσωτερική έκφραση και χρησιμοποιήστε το αποτέλεσμα ως θέση.

K10.7 · Μέγεθος πίνακα int

K10.7 · *Kahoot «Πίνακες» (διάλεξη 10)* · ★★☆ · multiple-choice · 49% σωστές · `kahoot-int-array-bytes`

Ένας πίνακας `int numbers[1024];` πόσα bytes καταλαμβάνει στη μνήμη;

- 1024
- 4096
- 8192
- Εξαρτάται

Συχνή παρανόηση Το 35% επέλεξε `4096`, θεωρώντας ότι ένας `int` είναι πάντα 4 bytes. Το πρότυπο της C δεν ορίζει το ακριβές μέγεθος του `int`· εξαρτάται από το σύστημα.

Υπόδειξη Το μέγεθος του πίνακα είναι πλήθος στοιχείων × μέγεθος στοιχείου· είναι το μέγεθος του `int` ίδιο παντού;

Κ10.8 · Διεύθυνση στοιχείου πίνακα int

Κ10.8 · *Kahoot «Πίνακες» (διάλεξη 10)* · ★★☆ · multiple-choice · 47% σωστές · `kahoot-int-array-address`

Ο πίνακας `int numbers[200]` με `sizeof(int) == 4` ξεκινάει στη διεύθυνση 1000. Πού βρίσκεται το στοιχείο `numbers[42]`;

- `1042`
- `1046`
- `1842`
- `1168`

Υπόδειξη Τα στοιχεία είναι συνεχόμενα στη μνήμη και το καθένα πιάνει `sizeof(int)` bytes.

Κ10.9 · Χαρακτήρας σε συμβολοσειρά

Κ10.9 · *Kahoot «Πίνακες» (διάλεξη 10)* · ★★★ · multiple-choice · 27% σωστές · `kahoot-string-char-index`

Τι θα τυπώσει ο παρακάτω κώδικας;

```
char hello[] = "Hello World";
printf("%c", hello[6]);
```

- `H`
- `W`
- `o`
- (κενό)

Συχνή παρανόηση Το 47% επέλεξε «(κενό)», μετρώντας τις θέσεις από το 1. Το κενό είναι το `hello[5]`, γιατί η αρίθμηση στους πίνακες ξεκινά από το 0.

Υπόδειξη Μετρήστε τους χαρακτήρες ξεκινώντας από το 0, μετρώντας και το κενό.

Κ10.10 · Διεύθυνση στοιχείου πίνακα char

Κ10.10 · *Kahoot «Πίνακες» (διάλεξη 10)* · ★★★ · multiple-choice · 19% σωστές · `kahoot-char-array-address`

Αν ο πίνακας `char array[100];` ξεκινάει στη διεύθυνση 1000, σε ποια διεύθυνση βρίσκεται το στοιχείο `array[42]`;

- `1000`

- 1042
- 1041
- 1043

Συχνή παρανόηση Το 52% επέλεξε 1041, μετρώντας τις θέσεις σαν να ξεκινούσαν από το 1. Αφού το `array[0]` είναι στο 1000 και κάθε `char` πιάνει 1 byte, το `array[i]` είναι στο 1000 + i.

Υπόδειξη Ποια είναι η διεύθυνση του `array[0]`, και πόσα bytes πιάνει κάθε `char`;

Ζέσταμα: από τις διαφάνειες (A10.1–A10.14)

A10.1 · Πρόσθεση δύο αριθμών από την είσοδο

A10.1 · *Διάλεξη 10, διαφάνεια 5* · ★☆☆ · programming · `slides-lec10-addnums`

Θέλω να διαβάσω δύο αριθμούς από την πρότυπη είσοδο και να τους προσθέσω. Πώς;

```
$ ./addnums
Give me a number: 40
Give me another number: 2
Total: 42
```

Ένας τρόπος είναι να χρησιμοποιήσουμε την `getchar` και να φτιάξουμε μια συνάρτηση `getinteger` που διαβάζει τους χαρακτήρες έναν-έναν και τους μετατρέπει σε αριθμό. Υπάρχει άλλος τρόπος να επιτύχουμε το ίδιο αποτέλεσμα; Γράψτε το πρόγραμμα `addnums.c`.

Υπόδειξη Η πρότυπη βιβλιοθήκη έχει μια «αδελφή» της `printf` για διάβασμα. Θυμηθείτε ότι πρέπει να της δώσετε πού να αποθηκεύσει κάθε τιμή, και ελέγξτε τι επιστρέφει.

A10.2 · Ανάθεση πίνακα σε πίνακα

A10.2 · *Διάλεξη 10, διαφάνεια 42* · ★☆☆ · short-answer · `slides-lec10-array-assign`

Μπορώ να αναθέσω τα στοιχεία ενός πίνακα σε άλλον;

```
int a[3] = {1, 2, 3};
int b[3] = {4, 5, 6};
b = a;
```

Υπόδειξη Δοκιμάστε να το μεταγλωττίσετε με `gcc`. Αν δεν επιτρέπεται, πώς θα αντιγράφατε τα στοιχεία;

A10.3 · Ποιος πίνακας πιάνει περισσότερη μνήμη;

A10.3 · *Διάλεξη 10, διαφάνεια 40* · ★☆☆ · short-answer · `slides-lec10-array-memory`

Πίνακες μπορούν να οριστούν για όλους τους τύπους της C. Παράδειγμα:

```
int a[1024];
char b[2048];
double c[512];
```

Ποιος από τους παραπάνω πίνακες καταλαμβάνει περισσότερη μνήμη;

Υπόδειξη Ένας πίνακας πιάνει «πλήθος στοιχείων × μέγεθος τύπου» bytes. Θυμηθείτε τα συνηθισμένα `sizeof` των `int`, `char` και `double`.

A10.4 · Μέσος όρος πίνακα

A10.4 · *Διάλεξη 10, διαφάνειες 45-46* · ★☆☆ · programming · `slides-lec10-average`

Θέλω μια συνάρτηση που να δέχεται έναν πίνακα 100 ακεραίων και να επιστρέφει τον μέσο όρο. Πώς;

Υπόδειξη Χρησιμοποιήστε έναν συσσωρευτή που ξεκινά από 0. Προσέξτε τι τύπο επιστρέφετε και αν η διαίρεση είναι ακέραια.

A10.5 · Αναζήτηση στοιχείου σε πίνακα

A10.5 · *Διάλεξη 10, διαφάνειες 47-48* · ★☆☆ · programming · `slides-lec10-find`

Θέλω μια συνάρτηση που να δέχεται έναν πίνακα 100 ακεραίων και έναν ακέραιο και να γυρνάει τη θέση του στοιχείου αν το βρήκε ή -1. Πώς;

Υπόδειξη Ελέγξτε τα στοιχεία ένα-ένα και επιστρέψτε μόλις βρείτε αυτό που ψάχνετε. Πότε είστε σίγουροι ότι το στοιχείο δεν υπάρχει;

A10.6 · Το μεγαλύτερο αρκουδάκι

A10.6 · *Διάλεξη 10, διαφάνειες 21-26 και 37* · ★☆☆ · programming · `slides-lec10-find-max`

Έστω ότι έχω 100 αρκουδάκια και ψάχνω να βρω το μεγαλύτερο. Τι μπορώ να κάνω για να το βρω; Στόχος: ελαχιστοποίηση του κόπου.

Αναπαριστούμε κάθε αρκουδάκι με έναν ακέραιο (`int`). Γράψτε μια συνάρτηση `find_max` που να παίρνει έναν πίνακα 100 στοιχείων και να γυρίζει το μέγιστο.

Υπόδειξη Κρατήστε το «μεγαλύτερο μέχρι τώρα» και κοιτάξτε κάθε στοιχείο μία φορά. Με τι τιμή πρέπει να ξεκινά αυτό, ώστε η συνάρτηση να δουλεύει και αν όλα τα στοιχεία είναι αρνητικά;

A10.7 · Πρόσβαση εκτός ορίων πίνακα

A10.7 · *Διάλεξη 10, διαφάνεια 44* · ★☆☆ · short-answer · `slides-lec10-out-of-bounds`

Για τον πίνακα `int bears[100];`, τι θα συμβεί αν προσπελάσω εκτός ορίων, π.χ. `bears[100]` ή `bears[-1]`;

Υπόδειξη Ποιες είναι οι έγκυρες θέσεις; Σκεφτείτε αν η C ελέγχει τα όρια και πώς κατατάσσει το standard αυτή τη χρήση.

A10.8 · Τι κάνει το πρόγραμμα με τη scanf;

A10.8 · *Διάλεξη 10, διαφάνειες 9-10* · ★☆☆ · trace · `slides-lec10-scanf-square`

Τι κάνει το παρακάτω πρόγραμμα;

```c
#include <stdio.h>

int main() {
  int n;
  printf("Gimme a number: ");
  scanf("%d", &n);
  printf("Square: %d\n", n * n);
  return 0;
}
```

Υπόδειξη Διαβάστε τη `scanf` ως την «αντίστροφη» της `printf`. Τι ρόλο παίζει το `&` μπροστά από το `n`;

A10.9 · scanf με πολλά ορίσματα

A10.9 · *Διάλεξη 10, διαφάνειες 15-16* · ★☆☆ · trace · `slides-lec10-scanf-two-numbers`

Τι κάνει το παρακάτω πρόγραμμα; Τι θα τυπώσει αν δώσουμε ως είσοδο `2 4` (με επιπλέον κενά);

```c
#include <stdio.h>

int main() {
  int n1, n2;
  printf("Gimme two numbers: ");
  scanf("%d %d", &n1, &n2);
  printf("Result: %d\n", n1 * n2);
  return 0;
}
```

Υπόδειξη Τι κάνει η `scanf` με τους κενούς χαρακτήρες όταν ψάχνει για δεκαδικό ψηφίο;

A10.10 · Η συνάρτηση atoi

A10.10 · *Διάλεξη 10, διαφάνειες 49-50* · ★★☆ · programming · `slides-lec10-atoi`

Θέλω μια συνάρτηση `atoi` που να παίρνει ένα πίνακα χαρακτήρων (μόνο ψηφία) και να επιστρέφει έναν ακέραιο. Πώς;

Αφού τη γράψετε: τι μπορεί να πάει στραβά με αυτήν τη συνάρτηση;

Υπόδειξη Πώς ξέρετε πού τελειώνει ο πίνακας χαρακτήρων; Η μετατροπή είναι ίδια με της `getinteger` σε βάση 10. Για το δεύτερο σκέλος, σκεφτείτε εισόδους που δεν είναι «μόνο ψηφία» ή είναι πολύ μεγάλες.

A10.11 · Τι κάνει η getinteger;

A10.11 · *Διάλεξη 10, διαφάνεια 17* · ★★☆ · trace · `slides-lec10-getinteger`

Τι κάνει το παρακάτω πρόγραμμα;

```c
#define ERROR -1

int getinteger(int base) {
  int ch;
  int val = 0;
  while ((ch = getchar()) != '\n')
    if (ch >= '0' && ch <= '0' + base - 1)
      val = base * val + (ch - '0');
    else
```

```
      return ERROR;
  return val;
}
```

Υπόδειξη Εκτελέστε τη με το χέρι για `base = 10` και είσοδο `472`, και μετά για `base = 2` και είσοδο `101`. Ποιοι χαρακτήρες θεωρούνται νόμιμοι για κάθε βάση;

A10.12 · Τι κάνει ο πίνακας letfr;

A10.12 · *Διάλεξη 10, διαφάνεια 18* · ★★☆ · trace · `slides-lec10-letter-frequency`

Τι κάνει το παρακάτω πρόγραμμα;

```
int i, ch, total = 0;
int letfr[26];  // Letter occurrences and frequencies array
for (i=0 ; i < 26 ; i++)
  letfr[i] = 0;
while ((ch = getchar()) != EOF) {
  if (ch >= 'A' && ch <= 'Z') {
    letfr[ch-'A']++;            // Found upper case letter
    total++;
  }
  if (ch >= 'a' && ch <= 'z') {
    letfr[ch-'a']++;            // Found lower case letter
    total++;
  }
}
```

Υπόδειξη Υπολογίστε τι τιμή έχει η έκφραση `ch - 'A'` για `ch = 'A'`, `'B'` και `'Z'`. Τι μετράει λοιπόν το `letfr[i]` και τι το `total`;

A10.13 · Είναι σωστό αυτό το πρόγραμμα;

A10.13 · *Διάλεξη 10, διαφάνειες 12-14* · ★★☆ · debug · `slides-lec10-scanf-correct`

Είναι σωστό αυτό το πρόγραμμα;

```
#include <stdio.h>

int main() {
  int n;
  printf("Gimme a number: ");
```

```
  scanf("%d", &n);
  printf("Square: %d\n", n * n);
  return 0;
}
```

Μερικές εκτελέσεις:

```
$ ./scanf
Gimme a number: 16
Square: 256
$ ./scanf
Gimme a number: Square:
1068701481
$ ./scanf
Gimme a number: hello
Square: 1072038564
```

Υπόδειξη Η `scanf` επιστρέφει κάτι. Τι επιστρέφει όταν τελειώσει η είσοδος και τι όταν η είσοδος δεν είναι αριθμός; Τι τιμή έχει η `n` σε αυτές τις περιπτώσεις;

A10.14 · scanf χωρίς &

A10.14 · *Διάλεξη 10, διαφάνεια 11* · ★★☆ · debug · `slides-lec10-scanf-no-ampersand`

Στο πρόγραμμα:

```
#include <stdio.h>

int main() {
  int n;
  printf("Gimme a number: ");
  scanf("%d", &n);
  printf("Square: %d\n", n * n);
  return 0;
}
```

Τι θα γινόταν αν γράφαμε `scanf("%d", n);`; Γιατί;

```
$ ./scanf
Gimme a number: 3
Segmentation fault
```

Υπόδειξη Σκεφτείτε τι παίρνει μια συνάρτηση όταν της περνάμε μια μεταβλητή με τιμή, και τι τιμή έχει η `n` τη στιγμή της κλήσης. Πού θα προσπαθήσει να γράψει η `scanf`;

Εργαστήριο (A10.15–A10.17)

A10.15 · Πίνακες και συναρτήσεις

A10.15 · *Εργαστήριο 6, Άσκηση 4* · ★☆☆ · programming · `lab-lab06-judgement`

Σε ένα αγώνισμα υπάρχουν 10 κριτές. Η βαθμολογία του κάθε αγωνιζόμενου βγαίνει από τον μέσο όρο 8 κριτών, αφού εξαιρεθεί αυτός με τη μεγαλύτερη και αυτός με τη μικρότερη βαθμολογία (για να αποφευχθούν φαινόμενα μεροληψίας, είτε υπέρ είτε κατά ενός αθλητή).

4.1 Ορίστε τη συνάρτηση `int max_array(int *array, int n)` που επιστρέφει στο όνομά της το μέγιστο στοιχείο του πίνακα ακεραίων `array` διάστασης `n`.

4.2 Ορίστε τη συνάρτηση `int min_array(int *array, int n)` που επιστρέφει στο όνομά της το ελάχιστο στοιχείο του πίνακα ακεραίων `array` διάστασης `n`.

4.3 Ορίστε τη συνάρτηση `int sum_array(int *array, int n)` που επιστρέφει στο όνομά της το άθροισμα των στοιχείων του πίνακα ακεραίων `array` διάστασης `n`.

4.4 Κατασκευάστε το πρόγραμμα `judgement.c` το οποίο να διαβάζει τις 10 βαθμολογίες με χρήση της `scanf()` και να τυπώνει τη βαθμολογία με ακρίβεια 2 δεκαδικών ψηφίων σύμφωνα με τον τύπο:

```
Βαθμός = (Άθροισμα - Μέγιστο - Ελάχιστο) / 8
```

Έστω ένα αρχείο `grades.txt` με τις παρακάτω βαθμολογίες:

```
8 2 3 9 0 10 8 3 4 5
```

Παράδειγμα εκτέλεσης ακολουθεί:

```
$ ./judgement < grades.txt
Sum: 52
Min: 0
Max: 10
Average: 5.25
```

Υπόδειξη Για μέγιστο και ελάχιστο, αρχικοποιήστε με το πρώτο στοιχείο του πίνακα (όχι με 0) και συγκρίνετε με τα υπόλοιπα. Στον τελικό τύπο και τα τρία μεγέθη είναι ακέραια: αν δεν μετατρέψετε σε `double` πριν τη διαίρεση με το 8, θα χάσετε το δεκαδικό μέρος.

A10.16 · Το κόσκινο του Ερατοσθένη

A10.16 · *Εργαστήριο 6, Άσκηση 2* · ★☆☆ · programming · `lab-lab06-sieve`

Το κόσκινο του Ερατοσθένη είναι ένας αλγόριθμος για την εύρεση όλων των πρώτων αριθμών σε ένα εύρος τιμών (από 2 έως N - 1). Είναι ένας από τους αρχαιότερους γνωστούς αλγορίθμους και οφείλεται στον Έλληνα μαθηματικό και αστρονόμο Ερατοσθένη (276-194 π.Χ.). Ο αλγόριθμος εξετάζει διαδοχικά όλους τους ακεραίους και για κάθε αριθμό που συναντά διαγράφει όλα τα πολλαπλάσιά του (αφού σίγουρα δεν είναι πρώτοι).

Παρατηρήστε τα τρία πρώτα βήματα του αλγορίθμου, για N=20.

Όταν ξεκινά ο αλγόριθμος, όλοι οι αριθμοί θεωρούνται πιθανοί πρώτοι (αρχικοποιημένοι στο 1):

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Ο «2» είναι πρώτος, διαγραφή των 4, 6, 8, 10, 12, 14, 16, 18, 20 (το σημειώνουμε με 0):

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Ο «3» είναι πρώτος, διαγραφή των 6, 9, 12, 15, 18:

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	1	0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	1	0

Ο «4» δεν εξετάζεται γιατί έχει αφαιρεθεί σε προηγούμενο βήμα.

Ορίστε έναν πίνακα N θέσεων και αρχικοποιήστε τον με μονάδες. Η διάσταση του πίνακα να ορίζεται μέσω `#define` με τιμή ίση με 50. Υλοποιήστε το κόσκινο του Ερατοσθένη σύμφωνα με τον αλγόριθμο όπως εκφράζεται παρακάτω σε ψευδογλώσσα:

```
Για i=2 έως N-1 επανάλαβε
  Θέσε A[i]=1

Για i=2 έως N-1 επανάλαβε
  Αν A[i]!=0
    Για j=2*i έως N-1 με βήμα i επανάλαβε
      Θέσε A[j]=0

Για i=2 έως N-1 επανάλαβε
  Αν το A[i]==1 τύπωσε "i is a prime number"
```

Εκτελέστε το πρόγραμμά σας και επιβεβαιώστε την ορθότητα του αποτελέσματος.

Υπόδειξη Η ψευδογλώσσα μεταφράζεται σχεδόν γραμμή προς γραμμή σε βρόχους `for`. Προσέξτε τα όρια: ο πίνακας έχει θέσεις 0 έως N-1, άρα «έως N-1» σημαίνει συνθήκη `< N`, και ο δείκτης του πίνακα είναι ο ίδιος ο αριθμός που εξετάζετε.

A10.17 · Εντοπισμός σφάλματος μνήμης με τον gdb

A10.17 · *Εργαστήριο 7, Άσκηση 6* · ★☆☆ · debug · `lab-lab07-my_prog`

Το παρακάτω πρόγραμμα `my_prog.c` προσπαθεί να μετρήσει πόσοι αριθμοί στον πίνακα `p` είναι θετικοί και να υπολογίσει το άθροισμά τους (παρατηρήστε ότι το μέγεθος του πίνακα δεν είναι καθορισμένο με άμεσο τρόπο).

```c
#include <stdio.h>

int main(int argc, char *argv[]) {
    int p[] = {10, -1, 8, 6, 9, 13, 0, -9, 6};
    int count = 0, sum = 0, i = 0;
    do {
        if (p[i] > 0) {
            count++;
            sum += p[count];
        }
        i++;
    } while (1);
    printf("There are %d positive numbers with sum %d\n", count, sum);
    return 0;
}
```

Τρέχοντας αυτό το πρόγραμμα, αργά ή γρήγορα θα μας δώσει segmentation fault. Μεταγλωττίζοντας με `gcc -g3 -o my_prog my_prog.c` και τρέχοντάς το μέσα στον gdb (`gdb ./my_prog` και μετά `r`), παίρνουμε κάτι σαν:

```
Program received signal SIGSEGV, Segmentation fault.
0x080483ac in main () at my_prog.c:7
7   if (p[i] > 0) {
```

```
(gdb) p p[i]
Cannot access memory at address 0xbf8de000
(gdb) p i
$3 = 1279
```

Δεδομένου ότι το μέγεθος του πίνακα `p` δεν δηλώνεται ρητά, πώς θα αντιμετωπίζατε αυτό το πρόβλημα; Ήταν αυτό αρκετό για να λειτουργήσει σωστά το πρόγραμμα; Αν φταίει και κάτι άλλο, προσπαθήστε να το βρείτε με τη χρήση του gdb.

Με βάση όσα είδατε παραπάνω, διορθώστε το `my_prog.c` ώστε να μην ξεπερνά τα όρια του πίνακα και να τερματίζει σωστά.

Υπόδειξη Το πλήθος των στοιχείων ενός πίνακα που δηλώθηκε με αρχικοποίηση βγαίνει από το `sizeof` όλου του πίνακα διά το `sizeof` ενός στοιχείου, και αυτό πρέπει να μπει στη συνθήκη του βρόχου. Μετά τη διόρθωση των ορίων, ελέγξτε με `p` στον `gdb` ποιο στοιχείο προστίθεται στο `sum` σε κάθε βήμα: είναι αυτό που μόλις ελέγξατε;

Θέματα εξετάσεων (A10.18–A10.24)

A10.18 · Αναποδογύρισμα

A10.18 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #5 (HP Themed), Θέμα 1* · ★☆☆ · programming · `exam-2023-fall-ex5-q1`

Πρόγραμμα: `flip.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που διαβάζει το κείμενο που δίνεται από την πρότυπη είσοδο (stdin) και τυπώνει την κάθε γραμμή στην πρότυπη έξοδο (stdout) αντίστροφα, δηλαδή τυπώνοντας τους χαρακτήρες όπως θα διαβάζονταν από τα δεξιά προς τα αριστερά. Παράδειγμα εκτέλεσης:

```
$ gcc -o flip flip.c
$ cat riddle.txt
, esiugsid ni sevil ohw nosrep eht fo kniht tsriF
. seil tub thguan sllet dna sterces ni slaed ohW
, dnem ot gniht tsal eht syawla s'tahw em llet ,txeN
? dne eht fo dne dna elddim fo elddim ehT
 draeh netfo dnuos eht em evig yllanif dnA
 drow dnif-ot-drah a rof hcraes eht gniruD
, siht em rewsna dna ,rehtegot meht gnirts woN
? ssik ot gnilliwnu eb uoy dluow erutaerc hcihW
$ ./flip < riddle.txt
First think of the person who lives in disguise ,
Who deals in secrets and tells naught but lies .
Next, tell me what's always the last thing to mend ,
The middle of middle and end of the end ?
And finally give me the sound often heard
During the search for a hard-to-find word 
Now string them together, and answer me this ,
Which creature would you be unwilling to kiss ?
```

Υπόδειξη Μαζέψτε τους χαρακτήρες κάθε γραμμής σε έναν πίνακα μέχρι το `'\n'` και τυπώστε τους από το τέλος προς την αρχή, αφήνοντας την αλλαγή γραμμής στη θέση της. Αν οι γραμμές

μπορεί να είναι οσοδήποτε μεγάλες, ο πίνακας πρέπει να μεγαλώνει δυναμικά (`realloc`). Μην ξεχάσετε την τελευταία γραμμή χωρίς `'\n'`.

A10.19 · Η συνάρτηση cons

A10.19 · *Εξέταση Σεπτεμβρίου 2026, Θέμα 2* · ★☆☆ · trace · `exam-2026-sep-q2`

Η συνάρτηση cons (15 Μονάδες)

Ζητήσαμε από την Κλαυδία να προγραμματίσει κάτι στα γρήγορα και χρησιμοποίησε την συνάρτηση `cons` στον κώδικά της:

```
int cons(int a[], int n) {
    int best = 1;
    int current = 1;

    for (int i = 1; i <= n; i++) {
        if (a[i] == a[i - 1])
            current++;
        else
            current = 1;

        if (current > best)
            best = current;
    }

    return best;
}
```

Τι κάνει η συνάρτηση `cons` (μέχρι 15 λέξεις εξήγηση);

Τι θα τυπώσει η παρακάτω ακολουθία εντολών;

```
    int arg[] = {4, 4, 2, 6, 7, 7, 7, 3};
    printf("%d", cons(arg, 6));
```

Υπάρχει κάποιο σφάλμα στην `cons`, αν `n` είναι το μέγεθος του πίνακα;

Υπόδειξη Ιχνηλατήστε τον βρόχο με το `current` και το `best` για κάθε `i` και δείτε ποια ζευγάρια γειτονικών στοιχείων συγκρίνονται. Στην κλήση, το `n` είναι 6 ενώ ο πίνακας έχει 8 στοιχεία: μετρήστε ακριβώς ποια στοιχεία διαβάζει ο βρόχος. Για το σφάλμα, ρωτήστε ποιος είναι ο μεγαλύτερος έγκυρος δείκτης ενός πίνακα μεγέθους `n` και αν η συνάρτηση δουλεύει για πολύ μικρά `n` (π.χ. 0).

A10.20 · Τοποθέτηση του Pacman

A10.20 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #11, Θέμα 2* · ★★☆ · programming · `exam-2023-fall-ex11-q2`

Πρόγραμμα: `position.c`

Γράψτε ένα πρόγραμμα που παίρνει ως πρώτο όρισμα την διάσταση ενός τετραγωνικού πλέγματος και ως υπόλοιπα ορίσματα τις θέσεις των φαντασμάτων (ghosts) του pacman και τυπώνει όλες τις θέσεις στο πλέγμα που θα μπορούσε να τοποθετηθεί ο pacman. Ο pacman δεν μπορεί να τοποθετηθεί στην ίδια γραμμή ή στήλη με κάποιο φάντασμα. Όλες οι διαστάσεις δίνονται ως "x,y" ορίσματα (όπου x, y είναι ακέραιοι). Εάν δεν υπάρχει δυνατή τοποθέτηση το πρόγραμμά σας θα πρέπει να τυπώνει ανάλογο μήνυμα. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o position position.c
$ ./position 4 0,4 3,2 2,1 1,3
Pacman cannot be positioned anywhere
$ ./position 4 0,4 3,2 2,1 0,3
Pacman can be positioned at: 1,0
$ ./position 4 0,4 3,2 2,1
Pacman can be positioned at: 1,0
Pacman can be positioned at: 1,3
$ ./position 4 1,2 2,1
Pacman can be positioned at: 0,0
Pacman can be positioned at: 0,3
Pacman can be positioned at: 3,0
Pacman can be positioned at: 3,3
```

Υπόδειξη Δεν χρειάζεστε ολόκληρο το πλέγμα: αρκούν δύο πίνακες σημαιών, ένας για τις γραμμές και ένας για τις στήλες που «καλύπτει» κάποιο φάντασμα. Κοιτάξτε στα παραδείγματα τι γίνεται με συντεταγμένες έξω από το πλέγμα (π.χ. `0,4` σε πλέγμα 4).

A10.21 · Φορτωμένο Έλκηθρο

A10.21 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #3 (Frozen Themed), Θέμα 2* · ★★☆ · programming · `exam-2023-fall-ex3-q2`

Πρόγραμμα: `sled.c` (25 μονάδες)

Ο Kristoff θέλει να πάρει τρία αντικείμενα μαζί του και να ελέγξει αν γεμίζουν πλήρως (από πλευράς βάρους) το έλκηθρό του. Γράψτε ένα πρόγραμμα που δέχεται ως πρώτο όρισμα την χωρητικότητα του ελκήθρου και στην συνέχεια το βάρος του κάθε αντικειμένου που θα μπορούσε να πάρει. Περιορισμοί: (1) τα βάρη είναι όλα ακέραιοι, (2) πρέπει να πάρει

αναγκαστικά 3 αντικείμενα, (3) το κάθε αντικείμενο έχει μοναδικό βάρος και (4) τα βάρη πρέπει να γεμίζουν *πλήρως* το έλκηθρο. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ ./sled 42 18 1 4 81 47 35 22 41 2 3 17
You can take 3 objects: 18 + 22 + 2 = 42
You can take 3 objects: 4 + 35 + 3 = 42
You can take 3 objects: 22 + 3 + 17 = 42
$ ./sled 42 18 1 4 81 47 35 41 2 17
No 3 objects found to fill up your sled.
```

Υπόδειξη Μετατρέψτε τα ορίσματα σε πίνακα ακεραίων (με έλεγχο ότι είναι έγκυροι ακέραιοι και ότι δεν επαναλαμβάνονται) και δοκιμάστε κάθε τριάδα δεικτών `i < j < k`, ώστε κάθε συνδυασμός να τυπωθεί μία φορά. Η σειρά εξόδου του παραδείγματος προκύπτει ακριβώς από αυτή τη σειρά απαρίθμησης. Αν θέλετε κάτι ταχύτερο από $O(n^3)$, σκεφτείτε ταξινόμηση και δύο δείκτες, αλλά τότε αλλάζει η σειρά εξόδου.

A10.22 · Αναγράμματα

A10.22 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #4 (Rick Astley Themed), Θέμα 1* · ★★☆ · programming · `exam-2023-fall-ex4-q1`

Πρόγραμμα: `anagram.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που δέχεται δύο ορίσματα και ελέγχει αν το ένα είναι ένα ανάγραμμα του άλλου. Δύο φράσεις λέγονται αναγράμματα όταν περιέχουν τους ίδιους χαρακτήρες αλλά πιθανώς με διαφορετική σειρά π.χ., οι φράσεις ”desert you” και ”you rested” είναι αναγράμματα η μία της άλλης. Παράδειγμα εκτέλεσης:

```
$ gcc -o anagram anagram.c
$ ./anagram "desert you!" "you rested!"
"desert you!" is an anagram of "you rested!".
$ ./anagram "never gonna give" "never gonna give"
"never gonna give" is an anagram of "never gonna give".
$ ./anagram "never gonna give" "gonna run around"
"never gonna give" is NOT an anagram of "gonna run around".
```

Υπόδειξη Μετρήστε πόσες φορές εμφανίζεται κάθε χαρακτήρας σε κάθε φράση, με έναν πίνακα 256 μετρητών που δεικτοδοτείται από την τιμή του byte (ως `unsigned char`), και συγκρίνετε τα δύο ιστογράμματα. Αυτό είναι $O(n)$, σε αντίθεση με την ταξινόμηση ή τη σύγκριση όλων με όλους. Ελέγξτε ότι δόθηκαν ακριβώς δύο ορίσματα.

A10.23 · Μαγικό Ζευγάρι

A10.23 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #5 (HP Themed), Θέμα 2* · ★★☆ · programming · `exam-2023-fall-ex5-q2`

Πρόγραμμα: `magicpair.c` (25 μονάδες)

Δύο θετικοί ακέραιοι αποτελούν μαγικό ζευγάρι εάν αποτελούνται από τα ίδια ψηφία αλλά πιθανόν με διαφορετική σειρά. Για παράδειγμα, οι αριθμοί 1980 και 1908 αποτελούν ένα μαγικό ζευγάρι, ενώ οι αριθμοί 1980 και 198 δεν είναι. Γράψτε ένα πρόγραμμα που δέχεται ακεραίους σε δεκαδική μορφή ως ορίσματα και τυπώνει αν οι αριθμοί αποτελούν ένα μαγικό ζευγάρι. Παραδείγματα εκτέλεσης:

```
$ ./magicpair 1980 1908
1980 and 1908 are a magic pair.
$ ./magicpair 1980 198
Not a magic pair
$ ./magicpair -18 -18
Not a magic pair
$ ./magicpair 4321 1234
4321 and 1234 are a magic pair.
$ ./magicpair 4231444232 3244214243
4231444232 and 3244214243 are a magic pair.
```

Υπόδειξη Μετρήστε πόσες φορές εμφανίζεται κάθε ψηφίο 0-9 σε κάθε αριθμό (ένας πίνακας 10 μετρητών ανά αριθμό) και συγκρίνετε. Τα παραδείγματα ξεπερνούν τον `int`, οπότε ή δουλέψτε με 64-bit ακέραιους ή απευθείας με τους χαρακτήρες του ορίσματος, αφού ελέγξετε ότι είναι μόνο ψηφία. Τι απαντάτε για αρνητικούς αριθμούς, για το 0 και για είσοδο που δεν είναι αριθμός;

A10.24 · Η συνάρτηση paws

A10.24 · *Εξέταση Ιουνίου 2026, Θέμα 2* · ★★☆ · debug · `exam-2026-jun-q2`

Η συνάρτηση paws (15 Μονάδες)

Ζητήσαμε από τον Κλαύδιο να προγραμματίσει κάτι στα γρήγορα και συμπεριέλαβε την ακόλουθη συνάρτηση paws στον κώδικά του:

```c
void paws(char *a, size_t len) {
    int lo = 0, hi = len;
    while(lo < hi) {
      char tmp = a[lo];
      a[lo++] = a[hi];
      a[hi--] = tmp;
    }
```

```
    return;
}
```

1. Τι κάνει η συνάρτηση `paws` (μέχρι 15 λέξεις εξήγηση);
2. Τι θα τυπωθεί στην πρότυπη έξοδο μετά την εκτέλεση της παρακάτω ακολουθίας εντολών;

   ```
   char buffer[] = { 33, 0x43, 'B', 0x41, '!', 0 };
   paws(buffer, 3);
   printf("Result: %s\n", buffer);
   ```

3. Μπορεί να προκύψει κάποιο σφάλμα στην συνάρτηση `paws` όπως είναι γραμμένη; Αν ναι, εξηγήστε γιατί και πως θα το διορθώνατε. Αν όχι, εξηγήστε γιατί αυτή η υλοποίηση είναι ορθή.

Υπόδειξη Μετατρέψτε πρώτα τα στοιχεία του `buffer` σε χαρακτήρες με τον πίνακα ASCII (33 και 0x43, 0x41 σε δεκαδικό). Στη συνέχεια εκτελέστε τον βρόχο με το χέρι, γράφοντας τα `lo`, `hi` και τα περιεχόμενα του πίνακα σε κάθε βήμα. Για το σφάλμα, αναρωτηθείτε ποιος είναι ο δείκτης (index) του τελευταίου έγκυρου στοιχείου ενός πίνακα μήκους `len`, τι γίνεται όταν `len` είναι 0, και τι σημαίνει η μετατροπή `size_t` σε `int`.

Κεφάλαιο 11: Δείκτες και Αναδρομή

Kahoot από το αμφιθέατρο (K11.1–K11.14)

K11.1 · Τερματισμός αναδρομής

K11.1 · *Kahoot «Δείκτες και Αναδρομή» (διάλεξη 11) και «Δείκτες Παντού!» (διάλεξη 12)* · ★☆☆ · multiple-choice · 87% σωστές · `kahoot-recursion-terminates`

Η αναδρομή πρέπει να τερματίζει.

- True
- False

Υπόδειξη Τι γίνεται στη στοίβα σε κάθε νέα αναδρομική κλήση;

K11.2 · Πίνακας και δείκτης

K11.2 · *Kahoot «Δείκτες και Αναδρομή» (διάλεξη 11) και «Δείκτες Παντού!» (διάλεξη 12)* · ★☆☆ · multiple-choice · 71% σωστές · `kahoot-array-vs-pointer`

Μια μεταβλητή πίνακα συμπεριφέρεται ακριβώς όπως ένας δείκτης.

- True
- False

Υπόδειξη Συγκρίνετε τι επιστρέφει το `sizeof` και αν μπορείτε να αναθέσετε νέα τιμή σε καθένα από τα δύο.

K11.3 · *(ptr + 4) και ptr[4]

K11.3 · *Kahoot «Δείκτες και Αναδρομή» (διάλεξη 11)* · ★★☆ · multiple-choice · 67% σωστές · `kahoot-ptr-offset-equals-index`

Η έκφραση `*(ptr + 4)` είναι ισοδύναμη με την έκφραση `ptr[4]`.

- True
- False

Συχνή παρανόηση Το 26% απάντησε False, θεωρώντας ότι οι αγκύλες είναι κάτι διαφορετικό από την αριθμητική δεικτών. Στην C το `ptr[n]` ορίζεται ακριβώς ως `*(ptr + n)`.

Υπόδειξη Θυμηθείτε πώς ορίζει η C τον τελεστή `[]` με βάση την αριθμητική δεικτών.

K11.4 · Η τιμή του NULL

K11.4 · *Kahoot «Δείκτες και Αναδρομή» (διάλεξη 11) και «Δείκτες Παντού!» (διάλεξη 12)* · ★★☆ · multiple-choice · 65% σωστές · `kahoot-null-is-zero`

Η τιμή δείκτη `NULL` στα περισσότερα σημερινά συστήματα ισούται με τον αριθμό 0.

- True
- False

Συχνή παρανόηση Το 28% απάντησε False, θεωρώντας το `NULL` ειδική τιμή χωρίς αριθμητική αναπαράσταση. Στην πράξη το `NULL` είναι η διεύθυνση 0.

Υπόδειξη Ένας δείκτης είναι μια διεύθυνση, δηλαδή ένας αριθμός· ποια διεύθυνση διαλέξαμε να σημαίνει «πουθενά»;

Κ11.5 · Η τιμή του ptr + 1

Κ11.5 · *Kahoot «Δείκτες και Αναδρομή» (διάλεξη 11)* · ★★☆ · multiple-choice · 61% σωστές · `kahoot-pointer-plus-one`

Έστω ότι η τιμή του δείκτη `int *ptr` είναι 42. Η τιμή του δείκτη `(ptr + 1)` είναι:

- `43`
- `46`
- `42`
- Εξαρτάται

Υπόδειξη Πόσα bytes προχωράει ένας δείκτης σε `int` όταν του προσθέτουμε 1, και είναι αυτό το ίδιο σε κάθε σύστημα;

Κ11.6 · Διεύθυνση και ακέραιος

Κ11.6 · *Kahoot «Δείκτες και Αναδρομή» (διάλεξη 11)* · ★★☆ · multiple-choice · 50% σωστές · `kahoot-address-is-integer`

Η διεύθυνση μιας μεταβλητής είναι *πάντα* ένας ακέραιος αριθμός.

- True
- False

Συχνή παρανόηση Το 47% απάντησε False, ίσως επειδή η διεύθυνση έχει τύπο δείκτη ή επειδή τη βλέπουμε σε δεκαεξαδικό. Όμως κάθε διεύθυνση είναι ο αύξων αριθμός ενός byte στη μνήμη, δηλαδή ακέραιος.

Υπόδειξη Η μνήμη είναι μια σειρά από bytes αριθμημένα από το 0· τι είδους αριθμός είναι η θέση ενός byte;

Κ11.7 · Αλλαγή μεταβλητής και δείκτης

Κ11.7 · *Kahoot «Δείκτες και Αναδρομή» (διάλεξη 11)* · ★★☆ · multiple-choice · 48% σωστές · `kahoot-variable-increment-via-ptr`

Τι θα τυπώσει ο παρακάτω κώδικας;

```c
int x = 41;
int *ptr = &x;
x++;
printf("%d", *ptr);
```

- 41
- 42
- 43
- Που να ξέρω

Υπόδειξη Ο δείκτης δεν κρατά αντίγραφο της τιμής του x αλλά τη διεύθυνσή του.

K11.8 · Μέγεθος δεικτών

K11.8 · *Kahoot «Δείκτες και Αναδρομή» (διάλεξη 11)* · ★★☆ · multiple-choice · 46% σωστές · `kahoot-pointer-size`

Οι μεταβλητές `double *x;` και `char *c;` καταλαμβάνουν τον ίδιο χώρο στη μνήμη.

- True
- False

Συχνή παρανόηση Το 49% απάντησε False, μπερδεύοντας το μέγεθος του δείκτη με το μέγεθος του τύπου στον οποίο δείχνει. Κάθε δείκτης κρατά μια διεύθυνση, άρα όλοι έχουν το ίδιο μέγεθος (8 bytes σε 64-bit συστήματα).

Υπόδειξη Τι αποθηκεύει μια μεταβλητή δείκτη, και εξαρτάται το μέγεθος αυτού που αποθηκεύει από τον τύπο στον οποίο δείχνει;

K11.9 · Αύξηση του ίδιου του δείκτη

K11.9 · *Kahoot «Δείκτες και Αναδρομή» (διάλεξη 11) και «Δείκτες Παντού!» (διάλεξη 12)* · ★★★ · multiple-choice · 37% σωστές · `kahoot-pointer-increment-scalar`

Τι θα τυπώσει ο παρακάτω κώδικας;

```c
int x = 41;
int *ptr = &x;
ptr++;
printf("%d", *ptr);
```

- 41
- 42
- 43
- Που να ξέρω

Συχνή παρανόηση Το 25% επέλεξε `42`, θεωρώντας ότι το `ptr++` αυξάνει την τιμή του `x`. Στην πραγματικότητα αλλάζει τη διεύθυνση που κρατά ο δείκτης, όχι την τιμή στην οποία δείχνει.

Υπόδειξη Μετά το `ptr++` πού δείχνει ο `ptr`; Υπάρχει εκεί κάποια μεταβλητή που ξέρουμε;

K11.10 · Η έκφραση `*&x`

K11.10 · *Kahoot «Δείκτες και Αναδρομή» (διάλεξη 11) και «Δείκτες Παντού!» (διάλεξη 12)* · ★★★ · multiple-choice · 35% σωστές · `kahoot-deref-address-of`

Παλιό θέμα: η εντολή `y = *&x;` στη C είναι:

- Ισοδύναμη με την εντολή `y = x;`
- Ισοδύναμη με την εντολή `y = &*x;`
- Ισοδύναμη με την εντολή `y = *x;`
- Συντακτικά λανθασμένη

Υπόδειξη Οι τελεστές `*` και `&` είναι συμπληρωματικοί: τι κάνει ο καθένας στο αποτέλεσμα του άλλου;

K11.11 · Αύξηση μέσω δείκτη

K11.11 · *Kahoot «Δείκτες και Αναδρομή» (διάλεξη 11) και «Δείκτες Παντού!» (διάλεξη 12)* · ★★★ · multiple-choice · 34% σωστές · `kahoot-deref-increment`

Τι θα τυπώσει ο παρακάτω κώδικας;

```
int x = 41;
int *ptr = &x;
(*ptr)++;
printf("%d", *ptr);
```

- `41`
- `42`
- `43`
- Που να ξέρω

Υπόδειξη Οι παρενθέσεις ορίζουν σε τι εφαρμόζεται το `++`: στον ίδιο τον δείκτη ή σε αυτό που δείχνει;

Κ11.12 · Δείκτης συν 4 σε δεκαεξαδικό

Κ11.12 · *Kahoot «Δείκτες και Αναδρομή» (διάλεξη 11)* · ★★★ · multiple-choice · 31% σωστές · `kahoot-pointer-plus-int-hex`

Έστω ότι `sizeof(int) == 4` και η τιμή του `int *ptr` είναι `0x42`. Ποια η τιμή της έκφρασης `ptr + 4`;

- `0x46`
- `42`
- `0x52`
- `0x4C`

Συχνή παρανόηση Το 32% επέλεξε `0x4C` και το 22% `0x46`. Οι δεύτεροι ξέχασαν ότι το `+ 4` σημαίνει 4 × `sizeof(int)` bytes, ενώ οι πρώτοι πρόσθεσαν σωστά 16 bytes, αλλά μπέρδεψαν τη δεκαεξαδική πρόσθεση (16 = `0x10`).

Υπόδειξη Η αριθμητική δεικτών μετράει σε στοιχεία, όχι σε bytes· προσέξτε και σε ποια βάση κάνετε την πρόσθεση.

Κ11.13 · Άθροισμα δύο δεικτών

Κ11.13 · *Kahoot «Δείκτες και Αναδρομή» (διάλεξη 11) και «Δείκτες Παντού!» (διάλεξη 12)* · ★★★ · multiple-choice · 28% σωστές · `kahoot-pointer-addition`

Έστω ότι `int *p, *q` είναι δύο δείκτες και ο καθένας έχει τιμή 42. Τότε η παράσταση `p + q` είναι:

- `84`
- `166`
- Μη δεκτή στην C
- `0`

Συχνή παρανόηση Το 28% επέλεξε `166` και το 22% `84`, θεωρώντας ότι δύο δείκτες προστίθενται σαν ακέραιοι. Η C επιτρέπει δείκτη ± ακέραιο και διαφορά δύο δεικτών, όχι όμως άθροισμα δεικτών.

Υπόδειξη Σκεφτείτε ποιες πράξεις ανάμεσα σε δείκτες και ακεραίους έχουν νόημα για μια διεύθυνση μνήμης.

K11.14 · Δείκτης στη μέση πίνακα

K11.14 · *Kahoot «Δείκτες και Αναδρομή» (διάλεξη 11) και «Δείκτες Παντού!» (διάλεξη 12)* · ★★★ · short-answer · 12% σωστές · `kahoot-ptr-index-offset`

Τι θα τυπώσει ο παρακάτω κώδικας;

```
int a[] = {10, 20, 30, 40};
int *ptr = &a[1];
printf("%d", ptr[1]);
```

Γράψτε την απάντηση.

Υπόδειξη Το `ptr[n]` σημαίνει `*(ptr + n)`, και μετράει από εκεί που δείχνει ο `ptr`, όχι από την αρχή του πίνακα.

Ζέσταμα: από τις διαφάνειες (A11.1–A11.13)

A11.1 · Πρόσβαση σε μεταβλητή μόνο μέσω της διεύθυνσής της

A11.1 · *Διάλεξη 11, διαφάνεια 20* · ★☆☆ · short-answer · `slides-lec11-access-by-address`

Ο δείκτης *δείχνει* σε μια μεταβλητή. Μπορώ να προσπελάσω τη μεταβλητή έχοντας μόνο τη διεύθυνσή της;

Υπόδειξη Ο `&` πάει από τη μεταβλητή στη διεύθυνση. Υπάρχει τελεστής που κάνει την αντίστροφη διαδρομή; Σκεφτείτε αν η έκφραση που προκύπτει μπορεί να χρησιμοποιηθεί και για ανάγνωση και για ανάθεση.

A11.2 · Αναφορά σε στοιχεία πίνακα μέσω δείκτη

A11.2 · *Διάλεξη 11, διαφάνειες 29–30* · ★☆☆ · trace · `slides-lec11-array-pointer-trace`

Αναφορά σε στοιχεία πίνακα: τι θα τυπώσει;

```
#include <stdio.h>
int main() {
  int *ptr, arr[] = {10, 20, 30};
  ptr = &arr[0];
  printf("mem[%p], %d\n", ptr, *ptr);
  ptr += 2;
  printf("mem[%p], %d\n", ptr, *ptr);
  return 0;
}
```

Υπόδειξη Τις ακριβείς διευθύνσεις δεν μπορείτε να τις ξέρετε, αλλά μπορείτε να πείτε πόσο διαφέρουν οι δύο. Κατά πόσα bytes μετακινεί το `ptr += 2` έναν `int *`, και σε ποιο στοιχείο του πίνακα δείχνει μετά;

A11.3 · Αναθέσεις μέσω δεικτών

A11.3 · *Διάλεξη 11, διαφάνεια 24* · ★☆☆ · trace · `slides-lec11-assign-through-pointers`

Τι τυπώνει το παρακάτω πρόγραμμα;

```c
#include <stdio.h>
int main() {
  int a = 100, b = 200, c;
  int *ptr_a = &a, *ptr_b = &b, *ptr_c = &c;
  *ptr_c = a;
  *ptr_a = b;
  *ptr_b = *ptr_c;
  printf("%d %d %d", a, b, c);
  return 0;
}
```

Υπόδειξη Αντικαταστήστε κάθε `*ptr_x` με τη μεταβλητή στην οποία δείχνει ο δείκτης και εκτελέστε τις τρεις αναθέσεις με τη σειρά, κρατώντας έναν πίνακα με τις τιμές των `a`, `b`, `c` μετά από κάθε γραμμή.

A11.4 · Μέσος όρος πίνακα 100 ακεραίων

A11.4 · *Διάλεξη 11, διαφάνειες 37–38* · ★☆☆ · programming · `slides-lec11-average`

Θέλω μια συνάρτηση που να δέχεται έναν πίνακα 100 ακεραίων και να επιστρέφει τον μέσο όρο. Πώς;

Υπόδειξη Η παράμετρος μπορεί να δηλωθεί ως πίνακας (`int grades[100]`). Αθροίστε τα στοιχεία με έναν βρόχο `for` και διαιρέστε στο τέλος· σκεφτείτε αν θέλετε ακέραια διαίρεση ή αποτέλεσμα με δεκαδικά.

A11.5 · Το παραγοντικό αναδρομικά σε C

A11.5 · *Διάλεξη 11, διαφάνειες 44–45* · ★☆☆ · programming · `slides-lec11-factorial`

Στα μαθηματικά το παραγοντικό ενός φυσικού αριθμού n, συμβολίζεται με $n!$ και είναι το γινόμενο όλων των θετικών ακεραίων μικρότερων ή ίσων του n. Ο ορισμός του παραγοντικού στα μαθηματικά είναι αναδρομικός (recursive):

$$n! = 1 \text{ if } n = 0, \qquad n! = n \times (n-1)! \text{ if } n > 0$$

Πώς θα το γράφαμε σε C; Το πρόγραμμα παίρνει τον αριθμό από τη γραμμή εντολών:

```
$ ./fact 5
5! = 120
```

Υπόδειξη Οι δύο γραμμές του ορισμού αντιστοιχούν στις δύο περιπτώσεις μιας αναδρομικής συνάρτησης: η πρώτη είναι η βάση τερματισμού, η δεύτερη η κλήση της συνάρτησης στον εαυτό της με μικρότερο όρισμα. Για την είσοδο, μετατρέψτε το `argv[1]` σε ακέραιο με την `atoi`.

A11.6 · Πόσες αναδρομικές κλήσεις κάνει το factorial

A11.6 · *Διάλεξη 11, διαφάνεια 46* · ★☆☆ · short-answer · `slides-lec11-factorial-calls`

Με τη συνάρτηση

```c
int factorial(int number) {
  if (number == 0) return 1;
  else return number * factorial(number - 1);
}
```

πόσες αναδρομικές (στον εαυτό της) κλήσεις κάνει η κλήση `factorial(5)`; Πόσες η κλήση `factorial(N)`;

Υπόδειξη Γράψτε την αλυσίδα κλήσεων που ξεκινά από το `factorial(5)` μέχρι τη βάση τερματισμού και μετρήστε τους κρίκους της, προσέχοντας αν μετράτε και την αρχική κλήση. Μετά γενικεύστε για `N`.

A11.7 · Αρνητικό παραγοντικό

A11.7 · *Διάλεξη 11, διαφάνεια 46* · ★☆☆ · short-answer · `slides-lec11-factorial-overflow`

Για κάποιους αριθμούς το αποτέλεσμα του αναδρομικού προγράμματος παραγοντικού (με `int`) είναι αρνητικό. Τι συμβαίνει;

```
$ ./fact 20
20! = -2102132736
```

Υπόδειξη Ποια είναι η μεγαλύτερη τιμή που χωρά σε έναν `int` των 32 bit; Υπολογίστε πρόχειρα πόσο γρήγορα μεγαλώνει το $n!$ και βρείτε το πρώτο n για το οποίο το ξεπερνά.

A11.8 · Θέση στοιχείου σε πίνακα ή -1

A11.8 · *Διάλεξη 11, διαφάνειες 39–40* · ★☆☆ · programming · `slides-lec11-find`

Θέλω μια συνάρτηση που να δέχεται έναν πίνακα 100 ακεραίων και έναν ακέραιο και να γυρνάει τη θέση του στοιχείου αν το βρήκε ή `-1`. Πώς;

Υπόδειξη Διατρέξτε τον πίνακα από την αρχή· μόλις βρείτε το στοιχείο μπορείτε να επιστρέψετε αμέσως από μέσα από τον βρόχο. Τι πρέπει να γίνει αν ο βρόχος τελειώσει χωρίς να το βρει;

A11.9 · Το μέγεθος δεικτών διαφορετικών τύπων

A11.9 · *Διάλεξη 11, διαφάνεια 18* · ★☆☆ · trace · `slides-lec11-pointer-sizes`

Τι θα τυπώσει το παρακάτω πρόγραμμα;

```c
#include <stdio.h>

int main() {
  int * ipointer;
  char * cpointer;
  double * dpointer;
  printf("%d %d %d\n", sizeof(ipointer), sizeof(cpointer), sizeof(dpointer));
  return 0;
}
```

Υπόδειξη Τι ακριβώς αποθηκεύει μια μεταβλητή τύπου δείκτη; Εξαρτάται το μέγεθος αυτού που αποθηκεύει από τον τύπο των δεδομένων στα οποία δείχνει ή από το σύστημα (32 ή 64 bit) για το οποίο μεταγλωττίζεται το πρόγραμμα;

A11.10 · Πώς τυπώνω μόνο το World

A11.10 · *Διάλεξη 11, διαφάνειες 34–35* · ★☆☆ · programming · `slides-lec11-print-world`

Από την προηγούμενη φορά, πώς μπορώ να τυπώσω `"World\n"`; Συμπληρώστε το `...`:

```
char hello[] = "Hello World\n";
char *world = ...;
printf("%s", world);
```

Υπόδειξη Το `%s` τυπώνει χαρακτήρες ξεκινώντας από τη διεύθυνση που του δίνετε, μέχρι το `'\0'`. Μετρήστε σε ποια θέση του πίνακα βρίσκεται το `'W'` και σκεφτείτε πώς παίρνετε τη διεύθυνση αυτού του στοιχείου.

A11.11 · Η δική μας atoi

A11.11 · *Διάλεξη 11, διαφάνειες 41–42* · ★★☆ · programming · `slides-lec11-atoi`

Θέλω μια συνάρτηση `atoi` που να παίρνει έναν πίνακα χαρακτήρων (μόνο ψηφία) και να επιστρέφει έναν ακέραιο. Πώς;

Υπόδειξη Διατρέξτε τους χαρακτήρες μέχρι το `'\0'`. Η τιμή ενός χαρακτήρα-ψηφίου προκύπτει αφαιρώντας του το `'0'`· σκεφτείτε πώς αλλάζει το μέχρι τώρα αποτέλεσμα όταν «προσθέτετε» ένα ψηφίο στα δεξιά του (π.χ. από 12 σε 123).

A11.12 · Τι μπορεί να πάει στραβά με την atoi

A11.12 · *Διάλεξη 11, διαφάνεια 42* · ★★☆ · debug · `slides-lec11-atoi-pitfalls`

Τι μπορεί να πάει στραβά με αυτήν τη συνάρτηση;

```
int atoi(char digits[]) {
  int result = 0;
  for (int i = 0; digits[i]; i++) {
    result = 10 * result + digits[i] - '0';
  }
  return result;
}
```

Υπόδειξη Δοκιμάστε νοερά εισόδους που παραβιάζουν τις υποθέσεις της: χαρακτήρες που δεν είναι ψηφία (π.χ. πρόσημο), έναν πολύ μεγάλο αριθμό, έναν πίνακα χωρίς `'\0'` στο τέλος. Σκεφτείτε επίσης αν το όνομα της συνάρτησης συγκρούεται με κάτι από την standard library.

A11.13 · Αρνητικό όρισμα στο αναδρομικό παραγοντικό

A11.13 · *Διάλεξη 11, διαφάνεια 46* · ★★☆ · short-answer · `slides-lec11-factorial-negative`

Τι θα συμβεί αν δώσουμε έναν αρνητικό αριθμό στη συνάρτησή μας;

```
int factorial(int number) {
  if (number == 0) return 1;
  else return number * factorial(number - 1);
}
```

Υπόδειξη Ακολουθήστε τις τιμές του `number` στις διαδοχικές κλήσεις ξεκινώντας από `-1`. Φτάνει ποτέ στη βάση τερματισμού; Τι κοστίζει στη μνήμη κάθε κλήση που δεν έχει ακόμα επιστρέψει; Πώς θα άλλαζε η συνθήκη της βάσης για να προστατεύεται η συνάρτηση;

Εργαστήριο (A11.14–A11.18)

A11.14 · Πέρασμα δεδομένων μέσω δεικτών

A11.14 · *Εργαστήριο 6, Άσκηση 1* · ★☆☆ · programming · `lab-lab06-myprog`

1.1 Γράψτε την παρακάτω συνάρτηση μέσα σ' ένα αρχείο `myprog.c` και μεταγλωττίστε το:

```
void badf(int x, int y, int sum, int diff) {
  sum = x + y;
  diff = x - y;
}
```

Γράψτε και μία συνάρτηση `main` μέσα σ' ένα αρχείο `myprog.c` η οποία να υλοποιεί την παρακάτω διαδικασία (εκφρασμένη σε ψευδογλώσσα) και μεταγλωττίστε το:

```
Όρισε τις ακέραιες μεταβλητές a, b, sum, diff

Θέσε a=5, b=4, sum=0, diff=0

Κάλεσε badf(a, b, sum, diff)

Τύπωσε sum, diff
```

Δημιουργήστε το εκτελέσιμο πρόγραμμα `myprog` και εκτελέστε το. Τι παρατηρείτε; Μπορείτε να εξηγήσετε το αποτέλεσμα;

1.2 Τροποποιήστε τη συνάρτηση `void badf(int x, int y, int sum, int diff)` ώστε οι `sum` και `diff` να είναι δείκτες σε ακεραίους, και μετονομάστε την σε `goodf`. Τροποποιήστε την `main` του προγράμματος ώστε να καλεί τη συνάρτηση `goodf` αντί για την `badf`. Τι παρατηρείτε;

1.3 Χρησιμοποιήστε τη συνάρτηση `scanf()` για να διαβάσετε τους ακέραιους αριθμούς `a`, `b` που εμφανίζονται στην `main`.

Υπενθυμίζουμε ότι η συνάρτηση `scanf()` διαβάζει από την είσοδο δεδομένα προς επεξεργασία. Συντάσσεται με αντίστοιχο τρόπο με την `printf()` δηλαδή:

```
scanf("%y", &var);
```

όπου y είναι σύμβολο που αντιστοιχεί στον τύπο δεδομένων της μεταβλητής var (π.χ. `d` για `int`, `f` για `float`, κλπ.). Παρατηρούμε ότι στο δεύτερο όρισμα περνιέται η διεύθυνση της μεταβλητής `var`, για να αποθηκευθεί η καταχώρηση και μετά το πέρας της κλήσης.

Υπόδειξη Στη C τα ορίσματα περνούν με τιμή: η `badf` αλλάζει τα δικά της αντίγραφα. Για να αλλάξει μια συνάρτηση μεταβλητές της `main`, πρέπει να πάρει τις διευθύνσεις τους (`&`) και να γράψει μέσω του δείκτη (`*`). Είναι ακριβώς ο λόγος που η `scanf` θέλει `&var`.

A11.15 · Η εικασία του Collatz

A11.15 · *Εργαστήριο 5, Άσκηση 1* · ★★☆ · programming · `lab-lab05-collatz`

Η εικασία Collatz είναι ένα από τα πιο διάσημα άλυτα προβλήματα των μαθηματικών. Πολλοί μαθηματικοί έχουν επιχειρήσει κατά καιρούς να την αποδείξουν και μέχρι στιγμής δεν έχουμε ακόμα μια ευρέως αποδεκτή απόδειξη. Η εικασία ισχυρίζεται κάτι πολύ απλό: ότι η επανάληψη δυο απλών αριθμητικών πράξεων με μια συγκεκριμένη διαδικασία μπορεί να μετατρέψει οποιονδήποτε θετικό ακέραιο στο 1. Η διαδικασία των πράξεων έχει ως εξής:

1. Διάλεξε οποιονδήποτε θετικό ακέραιο N.
2. Αν ο αριθμός είναι 1, τότε τερμάτισε την διαδικασία.
3. Αν είναι άρτιος, ο επόμενος αριθμός στην ακολουθία θα είναι ο $N/2$.
4. Αν είναι περιττός, ο επόμενος αριθμός στην ακολουθία θα είναι ο $3 \times N + 1$.
5. Επανάλαβε τα βήματα 2-4 μέχρι να φτάσουμε στο 1.

Για παράδειγμα, έστω $N = 3$. Η παραπάνω διαδικασία θα παράξει την ακολουθία: 3, 10, 5, 16, 8, 4, 2, 1. Μπορείτε να δοκιμάσετε το ίδιο με τον αγαπημένο σας θετικό ακέραιο και να ελέγξετε την ακολουθία που παράγεται. Ας πάρουμε για παράδειγμα το 42, που οδηγεί στην ακολουθία: 42, 21, 64, 32, 16, 8, 4, 2, 1. Λογικά και η δική σας επιλογή κατέληξε στο 1 μετά από μερικά βήματα - αν όχι ίσως έχετε την ευκαιρία να γίνετε εκατομμυριούχοι (ποιος είπε ότι τα μαθηματικά δεν έχουν λεφτά!).

Για κάθε θετικό ακέραιο N, ο αριθμός στοιχείων της ακολουθίας που παράγεται μέχρι να καταλήξουμε στο 1, λέγεται *μήκος ακολουθίας Collatz*. Για παράδειγμα, για $N = 3$, το μήκος της ακολουθίας είναι 8 (η ακολουθία έχει 8 στοιχεία: 3, 10, 5, 16, 8, 4, 2, 1). Αντίστοιχα για τον αριθμό 42, το μήκος ακολουθίας Collatz είναι 9. Αν $N = 1$, τότε έχουμε το ελάχιστο μήκος 1.

Σε αυτήν την άσκηση, καλείστε να γράψετε ένα πρόγραμμα `collatz.c` που να βρίσκει το μήκος της ακολουθίας collatz για έναν αριθμό.

1.1 Κατασκευάστε τη συνάρτηση `int isodd(int n)` που δέχεται σαν όρισμα έναν ακέραιο `n` και επιστρέφει 1, αν ο αριθμός είναι περιττός, ή 0, αν ο αριθμός είναι άρτιος.

1.2 Γράψτε σε γλώσσα C μια συνάρτηση `int collatz_it(int n)` που υπολογίζει το μήκος ακολουθίας collatz - δηλαδή το πλήθος των αριθμών της ακολουθίας που ξεκινά από τον `n` και καταλήγει στο 1 (συμπεριλαμβανομένων και των δύο). Ολοκληρώστε την υλοποίηση με δομή επανάληψης.

> **Ορισμός συνάρτησης**
>
> ```
> <τύπος επιστροφής> <όνομα συνάρτησης>(<τυπικές παράμετροι>)
> {
>     <εντολές και δηλώσεις>
> }
> ```
>
> Ο **τύπος επιστροφής** είναι ο τύπος της τιμής που επιστρέφει η συνάρτηση μέσω της εντολής `return <παράσταση>;`. Αν η συνάρτηση δεν επιστρέφει τιμή, ο τύπος επιστροφής της ορίζεται ως `void`.
>
> Οι **τυπικές παράμετροι** είναι οι μεταβλητές - μαζί με τους τύπους τους - που χρησιμοποιεί η συνάρτηση και παίρνουν τιμές από την καλούσα συνάρτηση. Γράφονται χωρισμένες με κόμμα.

Αν μια συνάρτηση καλείται πριν οριστεί, πρέπει να έχει προηγηθεί η προαναγγελία του **πρωτοτύπου** της - ο ορισμός της χωρίς σώμα, με ερωτηματικό στο τέλος:

```c
int collatz_it(int n);   /* πρωτότυπο: προαναγγέλλει τη συνάρτηση */

int main(void)
{
    /* εδώ μπορούμε πλέον να καλέσουμε την collatz_it */
}

int collatz_it(int n)    /* ο ορισμός της συνάρτησης */
{
    /* ... */
}
```

1.3 Υλοποιήστε τον ίδιο αλγόριθμο σε μια συνάρτηση `int collatz(int n)` κάνοντας χρήση αναδρομής και **χωρίς** να χρησιμοποιήσετε άλλη δομή επανάληψης.

1.4 Ελέγξτε τα αποτελέσματά σας ώστε να βεβαιωθείτε ότι το πρόγραμμά σας λειτουργεί σωστά για διάφορες αρχικές τιμές:

```
$ ./collatz
Number to find the length of the collatz sequence: 42
Iterative result: 9
Recursive result: 9
```

```
$ ./collatz
Number to find the length of the collatz sequence: 950000001
Iterative result: 199
Recursive result: 199
```

Κάποιο από τα αποτελέσματά μας δεν συμφωνεί; Τι μπορεί να πηγαίνει στραβά;

1.5 (Προχωρημένο, Προαιρετικό) Κάνετε το πρόγραμμά σας να δέχεται τον ακέραιο από την γραμμή εντολών, για παράδειγμα:

```
$ ./collatz 950000001
Iterative result: 199
Recursive result: 199
```

Υπόδειξη Για την αναδρομική εκδοχή, διατυπώστε το μήκος για τον `n` μέσω του μήκους για τον επόμενο αριθμό της ακολουθίας, με βάση την περίπτωση $n = 1$. Για το 1.4, υπολογίστε με το χέρι πόσο γίνεται το $3 \cdot 950000001 + 1$ και συγκρίνετέ το με το μέγιστο `int`: τι τύπος χρειάζεται για τους ενδιάμεσους όρους;

A11.16 · Η ακολουθία Fibonacci

A11.16 · *Εργαστήριο 5, Άσκηση 2* · ★★☆ · programming · `lab-lab05-fib`

Η ακολουθία Fibonacci είναι από τις πιο διάσημες μαθηματικές ακολουθίες. Ο αναδρομικός ορισμός της συνάρτησης είναι ιδιαίτερα απλός:

$$fib(n) = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ fib(n-1) + fib(n-2) & \text{if } n \geq 2. \end{cases}$$

και προκύπτει η ακολουθία (συνδέεται και με την χρυσή τομή):

`0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...`

Σε αυτήν την άσκηση θα υλοποιήσετε ένα πρόγραμμα `fib.c` που υπολογίζει τον n-οστό αριθμό της ακολουθίας Fibonacci.

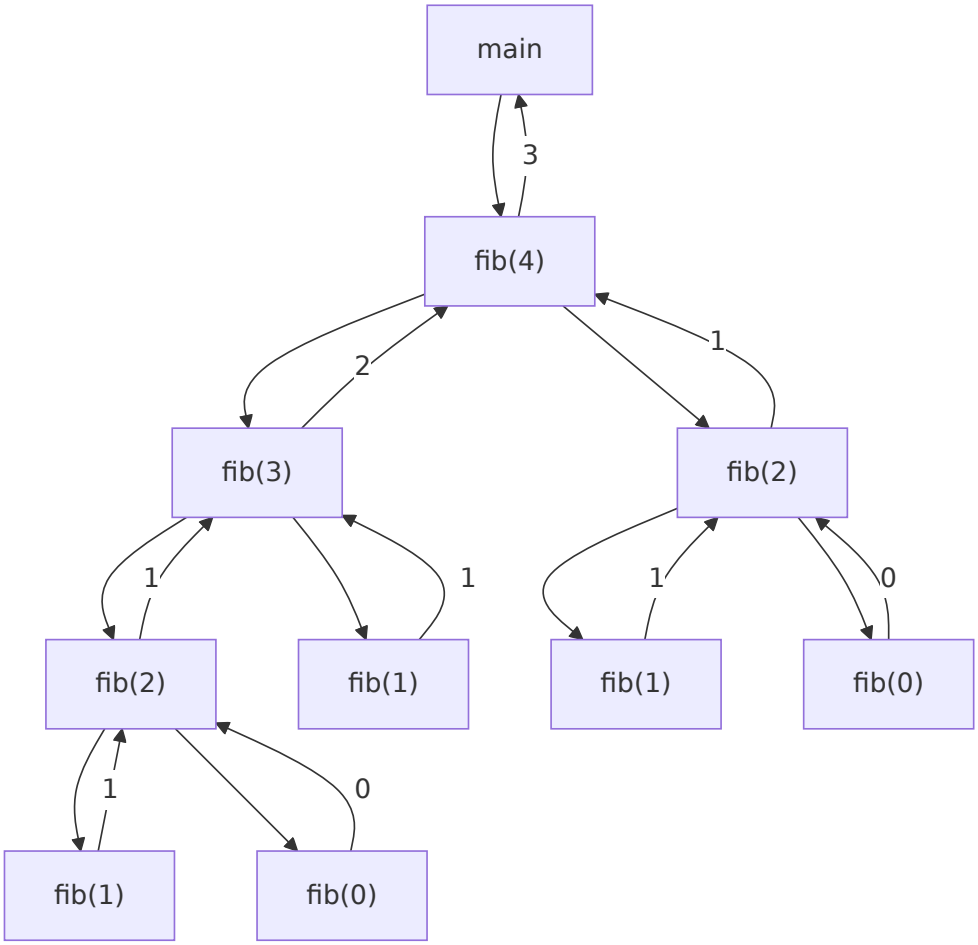
2.1 Ορίστε την αναδρομική συνάρτηση `int fib(int n)` που να υλοποιεί τον υπολογισμό του n-οστού όρου της ακολουθίας Fibonacci.

2.2 Μετατρέψτε το πρόγραμμά σας έτσι ώστε να δέχεται από τον χρήστη το πόσους αριθμούς Fibonacci θέλει να εκτυπωθούν και στην συνέχεια θα καλεί την συνάρτηση `fib` για να εκτυπώσει τον κάθε όρο. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ ./fib
How many fibonacci terms would you like: 35
fib(0) = 0
fib(1) = 1
fib(2) = 1
fib(3) = 2
fib(4) = 3
fib(5) = 5
fib(6) = 8
fib(7) = 13
fib(8) = 21
fib(9) = 34
...
fib(31) = 1346269
fib(32) = 2178309
fib(33) = 3524578
fib(34) = 5702887
```

Τρέξτε το πρόγραμμά σας για περισσότερους όρους, για παράδειγμα 45. Τι παρατηρείτε;

2.3 Μετρήστε τις αναδρομικές κλήσεις για κάθε εκτέλεση του παραπάνω προγράμματος χρησιμοποιώντας μία καθολική (global) μεταβλητή. Εκτυπώστε το πλήθος των αναδρομικών κλήσεων μετά από κάθε εκτέλεση.



Σχήμα: το δένδρο αναδρομικών κλήσεων για `fib(4)`*, με τις τιμές που επιστρέφει κάθε κλήση.*

2.4 Παρατηρήστε το δένδρο αναδρομικών κλήσεων του προγράμματος για n=4 και εξηγήστε την αύξηση του χρόνου υπολογισμού του προγράμματος σε σχέση με τον όρο που υπολογίζεται.

2.5 Ο n-οστός όρος της ακολουθίας Fibonacci μπορεί να υπολογισθεί και επαναληπτικά, αν χρησιμοποιήσουμε δύο μεταβλητές για να αποθηκεύουμε σε κάθε βήμα τους δύο προηγούμενους αριθμούς, ώστε προσθέτοντας τους, να υπολογίζουμε τον επόμενο. Γράψτε μια συνάρτηση `int fib_it(int n)` η οποία να υπολογίζει τον n-οστό όρο της ακολουθίας Fibonacci ακολουθώντας την παραπάνω επαναληπτική διαδικασία. Μετατρέψτε το πρόγραμμά σας ώστε να χρησιμοποιεί την επαναληπτική διαδικασία και ξανατρέξτε το παραπάνω πείραμα. Τι παρατηρείτε;

2.6 (Προχωρημένο, Προαιρετικό) Μπορείτε να υπολογίσετε τον n-οστό όρο Fibonacci αναδρομικά με απόδοση κοντά στον επαναληπτικό αλγόριθμο; Αν ναι, υλοποιήστε την λύση σας σε μια συνάρτηση `fib_rec_fast` και μετατρέψτε το πρόγραμμά σας ώστε να χρησιμοποιεί την συνάρτηση `fib_rec_fast`.

Υπόδειξη Στο δένδρο κλήσεων, πόσες φορές υπολογίζεται το `fib(2)`; Το ίδιο συμβαίνει σε κάθε επίπεδο, γι' αυτό το πλήθος των κλήσεων μεγαλώνει εκθετικά με το n. Ο μετρητής της 2.3 είναι μια μεταβλητή δηλωμένη έξω από κάθε συνάρτηση, που αυξάνεται στην αρχή της `fib`. Για το 2.6, σκεφτείτε μια αναδρομική συνάρτηση που «κουβαλά» τους δύο τελευταίους όρους ως ορίσματα, ή που θυμάται όσα έχει ήδη υπολογίσει.

A11.17 · Πίνακες και αριθμητική δεικτών

A11.17 · *Εργαστήριο 6, Άσκηση 3* · ★★☆ · trace · `lab-lab06-pointers`

3.1 Ορίστε τη συνάρτηση `void print_array(int *array, int n)` που δέχεται σαν όρισμα έναν πίνακα ακεραίων και τη διάστασή του και εκτυπώνει τα περιεχόμενά του σε μία γραμμή χωρισμένα με στηλογνώμονα (tab).

3.2 Δημιουργήστε το πρόγραμμα `pointers.c` που δημιουργεί έναν πίνακα 8 θέσεων και κάνει πράξεις πάνω σε αυτόν, ακολουθώντας το παρακάτω τμήμα κώδικα:

```
01: int i, a[8], *pa;
02:
03: for (i=0; i<8; i++)
04:   a[i] = i*i;
05:
06: pa = &a[0];
07: a[6] = *(a+4);
08: *(pa+3) = a[5];
09: a[0] = *((pa++)+2);
10: *((++pa)+5) = a[1];
11: *(&a[5]-1) = *(--pa);
```

3.3 Εκτυπώστε τα περιεχόμενα του πίνακα, χρησιμοποιώντας τη συνάρτηση `print_array()` μετά από τα βήματα 04, 07, 08, 09, 10 και 11 για να παρακολουθήσετε την επίδραση των εντολών στον πίνακα.

Υπόδειξη Πριν τρέξετε το πρόγραμμα, προβλέψτε με το χέρι τον πίνακα μετά από κάθε γραμμή και μετά συγκρίνετε. Το `*(p+k)` είναι το ίδιο με `p[k]`. Το δύσκολο σημείο είναι οι γραμμές 09–11: κρατήστε σημείωση σε ποιο στοιχείο δείχνει το `pa` κάθε στιγμή, και θυμηθείτε ότι το `pa++` χρησιμοποιεί την παλιά τιμή ενώ το `++pa` και το `--pa` τη νέα.

A11.18 · Υπολογισμός μισθών

A11.18 · *Εργαστήριο 8, Άσκηση 4* · ★★☆ · debug · `lab-lab08-wages`

Το παρακάτω πρόγραμμα δημιουργεί έναν πίνακα με τυχαίους μισθούς και βρίσκει το μέσο μισθό, παραλείποντας όσα κελιά έχουν τιμή μικρότερη ή ίση του μηδενός. Όμως το πρόγραμμα δίνει segmentation fault, ενώ έχει και λογικά λάθη. Αποσφαλματώστε το με τη βοήθεια ενός debugger και διορθώστε το.

```c
#include <stdio.h>

#define N 100

int main(int argc, char ** argv) {
  int i = 0, sum = 0;
  int *wages = NULL;
  srand(N);
  for (i = 0 ; i < N ; i++)
    wages[i] = 700 + (rand()%2301) - 1000;
  while (wages[i] > 0 && i < N) {
    sum += wages[i];
    i++;
  }
  printf("Average wage is %0.2f\n", sum/N);
  return 0;
}
```

Συμβουλές

Όποτε δουλεύετε με δείκτες στα προγράμματά σας, να έχετε πάντα στο νου σας τα εξής:

1. Ένας δείκτης δεν αρχικοποιείται σε NULL, αλλά δείχνει σε κάποια τυχαία θέση μνήμης. Είναι πολύ σημαντικό όποτε δηλώνουμε ένα δείκτη να του δίνουμε τιμή NULL, έτσι ώστε όποτε χρησιμοποιείται, να ελέγχουμε πρώτα αν η τιμή του είναι NULL.
2. Κάθε συμβολοσειρά `char *` πρέπει να έχει αρκετό χώρο για να χωρέσει όλους τους χαρακτήρες που θέλουμε να βάλουμε συν το τελικό `\0`.
3. Όποτε χρησιμοποιούμε πίνακες, πάντα προσέχουμε να μη βγούμε έξω από τα όριά τους.

Υπόδειξη Τρέξτε το στον `gdb` και δείτε σε ποια γραμμή σκάει και πού δείχνει ο `wages`. Το segfault είναι μόνο το πρώτο πρόβλημα: ιχνηλατήστε μετά τον βρόχο `while` (με ποια τιμή του `i` ξεκινά, τι κάνει όταν συναντήσει μη θετικό μισθό) και τον υπολογισμό του μέσου όρου, και μεταγλωττίστε με `-Wall` για να δείτε τις προειδοποιήσεις.

Εργασίες (A11.19–A11.21)

A11.19 · Ο Αλγόριθμος του Ευκλείδη (gcd)

A11.19 · *Εργασία 1 (2024-25), Άσκηση 1* · ★★☆ · programming · `hw-2024-hw1-gcd`

Ο Αλγόριθμος του Ευκλείδη (gcd - 50 Μονάδες)

Στις προτάσεις 1-2 του 7ου τόμου των "Στοιχείων", ο Ευκλείδης περιγράφει έναν αλγόριθμο ο οποίος χρησιμοποιείται ακόμα και σήμερα και λέγεται προς τιμήν του *Αλγόριθμος του Ευκλείδη* (Euclidean Algorithm). Ο αλγόριθμος μας βοηθάει να λύσουμε το πρόβλημα του *Μέγιστου Κοινού Διαιρέτη* (ΜΚΔ) ή *Greatest Common Divisor* (GCD) στα Αγγλικά: δοθέντων δύο αριθμών a και b ο μέγιστος κοινός διαιρέτης τους είναι ο μέγιστος αριθμός d ο οποίος *διαιρεί* τους a και b χωρίς να αφήνει υπόλοιπο. Πιο επίσημα, έχουμε τους ορισμούς:

Ορισμός 1. Αν ο a και ο b είναι ακέραιοι με $a \neq 0$, λέμε ότι *ο a διαιρεί τον b* αν υπάρχει ακέραιος c έτσι ώστε $b = a \cdot c$. Όταν ο a διαιρεί τον b λέμε ότι ο a είναι παράγοντας του b και ότι ο b είναι πολλαπλάσιο του a. Ο συμβολισμός $a \mid b$ σημαίνει ότι ο a διαιρεί τον b ($b \bmod a = 0$). Αντίθετα $a \nmid b$ συμβολίζει ότι ο a δεν διαιρεί τον b ($b \bmod a \neq 0$).

Ορισμός 2. Έστω ότι οι a και b είναι ακέραιοι, όχι μηδενικοί και οι δύο. Ο μεγαλύτερος ακέραιος d έτσι ώστε να είναι $d \mid a$ και $d \mid b$ ονομάζεται *μέγιστος κοινός διαιρέτης* των a και b και συμβολίζεται με $gcd(a, b)$.

Για να δούμε πως μπορούμε να λύσουμε ένα τέτοιο πρόβλημα. Έστω ότι οι δύο ακέραιοι είναι: το 42 και το 18. Στο Δημοτικό, προκειμένου βρούμε το ΜΚΔ παραγοντοποιούσαμε τους δύο αριθμούς και ο ΜΚΔ ήταν το γινόμενο των κοινών παραγόντων. Η παραγοντοποίηση του 18 είναι: $2 \cdot 3^2$ ενώ του 42 είναι $2 \cdot 3 \cdot 7$ και επομένως οι $2 \cdot 3$ είναι κοινοί παράγοντες και $gcd(42, 18) = 6$.

Η παραγοντοποίηση δεν είναι κακή μέθοδος αλλά είναι γενικά δύσκολη. Ο αλγόριθμος του Ευκλείδη προτείνει κάτι σχετικά πιο εύκολο (να αναφέρουμε ότι ο αυθεντικός αλγόριθμος του Ευκλείδη ήταν διαφορετικός, εδώ χρησιμοποιούμε μια πιο μοντέρνα εκδοχή του):

$$\gcd(a, b) = \begin{cases} b & \text{if } a \bmod b = 0 \\ \gcd(b, a \bmod b) & \text{otherwise} \end{cases}$$

Για να "τρέξουμε" την αναδρομική εξίσωση για το παράδειγμά μας: $\gcd(42, 18)$. Έχουμε ότι $42 \bmod 18 = 6$ (διάφορο του 0) και επομένως παίρνουμε τον δεύτερο κλάδο της συνάρτησης και υπολογίζουμε το $\gcd(18, 42 \bmod 18) = \gcd(18, 6)$. Τώρα όμως έχουμε ότι $18 \bmod 6 = 0$ και επομένως από τον πρώτο κλάδο της συνάρτησης έχουμε: $\gcd(18, 6) = 6$. Συνεπώς πήραμε και πάλι το αναμενόμενο αποτέλεσμα: $\gcd(42, 18) = 6$. Εύκολο; Θα δείξει!

Για το ζητούμενο αυτής της άσκησης λοιπόν, καλείστε να γράψετε ένα πρόγραμμα `gcd` που υπολογίζει αυτόματα και αποδοτικά τον μέγιστο κοινό διαιρέτη δύο ακεραίων αριθμών χρησιμοποιώντας τον αναδρομικό αλγόριθμο του Ευκλείδη.

Τεχνικές Προδιαγραφές

- Repository Name: progintro/hw1-<YourUsername>
- C Filepath: gcd/src/gcd.c

- Το πρόγραμμά θα πρέπει να παίρνει δύο ακεραίους αριθμούς στο δεκαδικό σύστημα ως ορίσματα από την γραμμή εντολών στην μορφή `./gcd num0 num1`. Αν το πρόγραμμα εκτελεστεί με ορίσματα που δεν ακολουθούν τις παραπάνω προδιαγραφές, πρέπει να εκτυπώσει αντίστοιχο μήνυμα όπως στα παρακάτω παραδείγματα και να επιστρέφει με κωδικό εξόδου (exit code) 1.
- Όλες οι παράμετροι θα είναι στο εύρος $[-10^{18}, 10^{18}]$. Σε περίπτωση που δοθεί το μηδέν (0) το πρόγραμμά σας πρέπει να επιστρέφει με κωδικό εξόδου 1.
- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr): `gcc -O3 -Wall -Wextra -Werror -pedantic -o gcd gcd.c`
- README Filepath: gcd/README.md
- **Ένα αρχείο που να περιέχει στοιχεία εισόδου και ένα εξόδου διαφορετικά από αυτά της εκφώνησης. Συγκεκριμένα προτείνουμε να βάλετε έναν συνδυασμό που θεωρείται ότι είναι δύσκολος να γίνει σωστός**.
  - input Filepath: `gcd/test/input.txt`
  - output Filepath: `gcd/test/output.txt`

  Παράδειγμα που όμως δεν θα γίνει δεκτό από την άσκηση επειδή είναι ήδη στα παραδείγματα παρακάτω, για το input.txt: ”`942 1042`” και για το output.txt: ”2”.
- Πρέπει να ολοκληρώνει την εκτέλεση μέσα σε: 1 δευτερόλεπτο.

Παρακάτω παραθέτουμε την αλληλεπίδραση με μια ενδεικτική λύση:

```
$ hostname
linux14
$ gcc -O3 -Wall -Wextra -Werror -pedantic -o gcd gcd.c
$ ./gcd
Usage: ./gcd <num1> <num2>
$ echo $?
1
$ ./gcd 1
Usage: ./gcd <num1> <num2>
$ ./gcd 18 42
gcd(18, 42) = 6
$ ./gcd 42 18
gcd(42, 18) = 6
$ ./gcd -42 18
gcd(-42, 18) = 6
$ ./gcd 982451653 776531401
```

```
gcd(982451653, 776531401) = 1
$ ./gcd 78423360000000000 35241600000000000
gcd(78423360000000000, 35241600000000000) = 960000000000
$ ./gcd 68719476736 84767329979727872
gcd(68719476736, 84767329979727872) = 68719476736
$ echo $?
0
$ time ./gcd 1000000000000000000 999999999999999999
gcd(1000000000000000000, 999999999999999999) = 1

real    0m0.009s
user    0m0.004s
sys     0m0.004s
```

Στο αρχείο README.md πρέπει να προσθέσετε οποιεσδήποτε παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.

Υπόδειξη Η αναδρομική συνάρτηση γράφεται σχεδόν αυτούσια από τον τύπο· η δυσκολία είναι γύρω της. Μετατρέψτε τα ορίσματα με `strtoll` και ελέγξτε ότι όλο το string ήταν αριθμός, απορρίψτε το μηδέν, και σκεφτείτε τι κάνει ο τελεστής `%` με αρνητικούς τελεστέους ώστε το αποτέλεσμα να βγαίνει θετικό (δείτε το `gcd(-42, 18) = 6`). Στην έξοδο τυπώνονται τα ορίσματα όπως δόθηκαν.

A11.20 · Άψογα Τετράγωνα (Bonus)

A11.20 · *Εργασία 1 (2023-24), Άσκηση 3 (Bonus)* · ★★★ · programming · `hw-2023-hw1-flawless`

Αυτή η άσκηση είναι Bonus, δηλαδή η λύση της *δεν* είναι απαραίτητη για να πάρει κάποιος/α όλες τις μονάδες της Εργασίας 1. Σε αυτήν την άσκηση θα ασχοληθούμε με μια κατηγορία αριθμών που ονομάσαμε *άψογα τετράγωνα*.

Ορισμός 6. Ένας φυσικός αριθμός λέγεται *άψογο τετράγωνο* (flawless square), όταν είναι τέλειο τετράγωνο και ταυτόχρονα ισούται με το τετράγωνο του αθροίσματος *διαδοχικών* ψηφίων του. Τονίζουμε ότι όλα τα ψηφία πρέπει να χρησιμοποιηθούν. Για παράδειγμα, ο αριθμός 1 είναι ένα άψογο τετράγωνο καθώς είναι το τετράγωνο ενός αριθμού (1) και ταυτόχρονα το άθροισμα των ψηφίων του στο τετράγωνο (1^2) ισούται με τον αρχικό αριθμό. Αντίστοιχα, ο αριθμός 81 είναι ένα άψογο τετράγωνο, καθώς είναι τέλειο (9^2) και ταυτόχρονα το άθροισμα των ψηφίων του στο τετράγωνο ισούται με τον αρχικό αριθμό ($(8 + 1)^2 = 81$). Παρακάτω παραθέτουμε μερικά ακόμα παραδείγματα άψογων αριθμών:

```
1296    = (1 + 29 + 6)^2                = 36^2
3025    = (30 + 25)^2                   = 55^2
```

```
8281    = (8 + 2 + 81)^2 = (82 + 8 + 1)^2 = 91^2
998001  = (998 + 0 + 0 + 1)^2             = 999^2
4941729 = (494 + 1729)^2                  = 2223^2
```

Για το ζητούμενο αυτής της άσκησης, καλείστε να γράψετε ένα πρόγραμμα το οποίο να υπολογίζει το άθροισμα όλων των άψογων τετραγώνων που βρίσκονται σε ένα εύρος φυσικών αριθμών.

Τεχνικές Προδιαγραφές

- Repository Name: `progintro/hw1-<YourUsername>`
- C Filepath: `flawless/src/flawless.c`
- Το πρόγραμμά θα πρέπει να παίρνει δύο ορίσματα από την γραμμή εντολών στην μορφή `./flawless low high`, με το πρώτο (`low`) να είναι το κάτω όριο και το δεύτερο να είναι το άνω όριο (`high`). Τα δύο όρια είναι κλειστά, δηλαδή η αναζήτηση πρέπει να γίνει στο σύνολο `[low, high]`. Για οποιαδήποτε είσοδο δεν είναι μέσα στις προδιαγραφές το πρόγραμμα πρέπει να επιστρέφει με κωδικό εξόδου (exit code) 1. Εάν δοθεί άνω όριο χαμηλότερο του κάτω ορίου το πρόγραμμα πρέπει να τερματίσει με κωδικό εξόδου 1.
- Όλοι οι ακέραιοι που θα δοθούν στο πρόγραμμά σας θα είναι στο εύρος: [1, 10^12]. Εάν δοθούν μη-θετικοί ακέραιοι ή ακέραιοι άνω του 10^12 το πρόγραμμά σας πρέπει να τερματίζει με κωδικό εξόδου 1. Οποιαδήποτε άλλη μη ακέραια είσοδος είναι εκτός προδιαγραφών και δεν θα ελεγχθεί.
- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr): `gcc -O3 -Wall -Wextra -Werror -pedantic -o flawless flawless.c -lm`
- Μορφοποίηση: το αρχείο που θα υποβληθεί πρέπει να έχει μορφοποίηση σύμφωνη με το C/C++ στυλ της Google. Για να μορφοποιήσετε το αρχείο σας, μπορείτε να τρέξετε την ακόλουθη εντολή: `clang-format -i -style=Google flawless.c` σε έναν υπολογιστή εργαστηρίου. Μη μορφοποιημένα προγράμματα δεν θα εξεταστούν.
- README Filepath: `flawless/README.md`
- **Ένα αρχείο που να περιέχει ένα στοιχείο εισόδου και ένα εξόδου *διαφορετικά* από αυτά της άσκησης. Συγκεκριμένα προτείνουμε να βάλετε έναν συνδυασμό που θεωρείται ότι είναι δύσκολος να γίνει σωστός κατά την υλοποίηση**.
  - input Filepath: `flawless/test/input`
  - output Filepath: `flawless/test/output`

  Για παράδειγμα, το περιεχόμενο του input αρχείου μπορεί να είναι: ”1 100000” και του αντίστοιχου output: ”184768”. Προσοχή: αυτό το παράδειγμα δεν θα γίνει δεκτό από την

άσκηση επειδή αυτό το input-output ζευγάρι υπάρχει ήδη παρακάτω, επομένως πρέπει να διαλέξετε κάποιο άλλο.

- Πρέπει να ολοκληρώνει την εκτέλεση μέσα σε: 10 δευτερόλεπτα.
- **Δεν επιτρέπεται να γίνει χρήση πινάκων/δεικτών πέρα από το διάβασμα των ορισμάτων της** `main`.
- **Δεν επιτρέπεται χρήση προϋπολογισμένων αποτελεσμάτων**. Το πρόγραμμά σας πρέπει να υπολογίζει το αποτέλεσμα χωρίς "πρότερη γνώση", δηλαδή χωρίς να έχετε ήδη κωδικοποιήσει τα άψογα τετράγωνα που έχετε βρει από προηγούμενους υπολογισμούς σας μέσα στον κώδικα.
- Καθώς αυτή η εργασία είναι Bonus, θα ελέγξουμε και ποιοτικά χαρακτηριστικά της υποβολής. Μια ιδιαίτερα δυσνόητη ή μη διαχειρίσιμη λύση (π.χ., 6 nested if-else, 35 μεταβλητές στην ίδια συνάρτηση κτλ) θα οδηγήσει σε αφαίρεση μονάδων κατά την κρίση του εξεταστή.

Παρακάτω παραθέτουμε την αλληλεπίδραση με μια ενδεικτική λύση:

```
thanassis@linux14:~$ gcc -O3 -Wall -Wextra -Werror -pedantic -o flawless
↪  flawless.c -lm
thanassis@linux14:~$ ./flawless 1 1000
182
thanassis@linux14:~$ ./flawless 1 100000
184768
thanassis@linux14:~$ ./flawless 1 10000000
30940314
thanassis@linux14:~$ time ./flawless 1 10000000000
499984803178

real    0m0,204s
user    0m0,203s
sys     0m0,001s
```

Στο αρχείο `README.md` πρέπει να προσθέσετε οποιεσδήποτε παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.

Υπόδειξη Διατρέξτε τις ρίζες k (από $\sqrt{\text{low}}$ έως $\sqrt{\text{high}}$) αντί για όλους τους αριθμούς του εύρους, και για κάθε k^2 ρωτήστε: «μπορώ να κόψω τα ψηφία του σε διαδοχικά κομμάτια με άθροισμα k;». Αυτό γράφεται φυσικά αναδρομικά: αφαιρέστε ένα κομμάτι από το τέλος του αριθμού (με `%` και `/` σε δύναμη του 10) και ρωτήστε το ίδιο για το υπόλοιπο με μικρότερο στόχο. Κόψτε νωρίς τους κλάδους όπου το μερικό άθροισμα ήδη ξεπέρασε το k, και προσέξτε τα ενδιάμεσα μηδενικά (π.χ. 998001).

Δείτε επίσης

- Υποδειγματική υποβολή φοιτητή (περιέχει λύση: δείτε την αφού λύσετε την άσκηση)

A11.21 · Ο Αλγόριθμος RSA (rsa)

A11.21 · *Εργασία 1 (2024-25), Άσκηση 2* · ★★★ · programming · `hw-2024-hw1-rsa`

Ο Αλγόριθμος RSA (rsa - 50 Μονάδες)

Κάθε φορά που συνδεόμαστε σε μια υπηρεσία (ssh, webmail, instagram) τα bytes του κωδικού μας ταξιδεύουν στο δίκτυο, και η κρυπτογραφία φροντίζει να μην μπορεί να τα διαβάσει οποιοσδήποτε. Σε αυτήν την άσκηση, θα ασχοληθούμε με έναν από τους πιο φημισμένους αλγορίθμους κρυπτογραφίας, τον αλγόριθμο RSA (Rivest-Shamir-Adleman, 1977). Χρειαζόμαστε: (1) μια συνάρτηση *encrypt* που να "κρύβει" το μήνυμά μας ώστε να μην μπορεί κάποιος άλλος να το διαβάσει καθώς το στέλνουμε και (2) μια συνάρτηση *decrypt* η οποία να παίρνει το κρυπτογραφημένο μήνυμά μας και να το μετατρέπει (αποκρυπτογραφεί) στο αρχικό.

Πως όμως μπορούμε να "κρύψουμε" το μήνυμά μας; Η βασική ιδέα του RSA είναι η εξής: έστω ότι το μήνυμα που θέλουμε να στείλουμε είναι ένας ακέραιος m. Τότε για να κρύψουμε το μήνυμά μας αρκεί να το υψώσουμε σε μια μεγάλη δύναμη: m^x. Η αποκρυπτογράφηση μπορεί να γίνει εξίσου απλά, αρκεί να βρούμε έναν αριθμό y έτσι ώστε $(m^x)^y = m$. Βλέποντας αυτό το παράδειγμα, ίσως σκέφτεστε ότι $x = 2$ και $y = \frac{1}{2}$ είναι μια πιθανή λύση στο πρόβλημά μας. Η σκέψη σας είναι σωστή, αλλά επειδή οποιοσδήποτε μπορεί να υπολογίσει την τετραγωνική ρίζα ενός αριθμού δεν μπορούμε να χρησιμοποιήσουμε αυτές τις πράξεις και αριθμούς για ασφαλή κρυπτογράφηση. Για να δούμε ποιους αριθμούς και πράξεις μπορούμε να χρησιμοποιήσουμε, χρειαζόμαστε πρώτα κάποιους ορισμούς:

Ορισμός 3. Ένας φυσικός αριθμός p μεγαλύτερος του 1 ονομάζεται *πρώτος (prime)* όταν έχει σαν μόνους διαιρέτες (το υπόλοιπο της διαίρεσης είναι 0) το 1 και το p. Για παράδειγμα, το 17 είναι πρώτος αριθμός, ενώ το 42 δεν είναι, αφού έχει σαν διαιρέτες το 2, το 3 και το 7, εκτός από τους 1 και 42. Μπορείτε να επιβεβαιώσετε ότι οι πρώτοι αριθμοί που είναι μικρότεροι από το 100 είναι οι εξής:

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97$$

Μαθηματικοί στο παρελθόν έχουν ασχοληθεί με πολλά γνωστά προβλήματα όπως η κατανομή τους. Μέχρι σήμερα, δεν έχει βρεθεί μια συνάρτηση που να παράγει τους πρώτους αριθμούς αποδοτικά.

Ορισμός 4. Δύο ακέραιοι a και b είναι *πρώτοι μεταξύ τους ή σχετικά πρώτοι (coprime)* όταν ο μέγιστος κοινός διαιρέτης τους είναι το 1, δηλαδή $\gcd(a, b) = 1$. Για παράδειγμα, οι αριθμοί 8 και 9 είναι coprime εφόσον το $\gcd(8, 9) = 1$ παρόλο που κανείς από τους δυο τους δεν είναι ο ίδιος πρώτος.

Ορισμός 5. Η συνάρτηση $\phi : \mathbb{N} \to \mathbb{N}$ του Euler (γνωστή και ως totient function) δέχεται έναν φυσικό αριθμό n και επιστρέφει το πλήθος των φυσικών αριθμών που είναι μικρότεροι του n και comprime με το n. Για παράδειγμα, $\phi(9) = 6$, εφόσον υπάρχουν ακριβώς έξι coprime με το 9: 1, 2, 4, 5, 7 και 8. Η συνάρτηση ϕ έχει την πολλαπλασιαστική ιδιότητα, δηλαδή για κάθε δύο φυσικούς a, b με $\gcd(a, b) = 1$ ισχύει ότι $\phi(a \cdot b) = \phi(a) \cdot \phi(b)$. Επίσης, αν ο αριθμός a είναι πρώτος, τότε ισχύει πως $\phi(a) = a - 1$. Για παράδειγμα: $\phi(5) = 4$, εφόσον οι αριθμοί 1, 2, 3, 4 είναι coprime ως προς το 5.

Έχοντας τους παραπάνω ορισμούς, μπορούμε επιτέλους να ορίσουμε τους περιορισμούς για να λειτουργήσει σωστά ο αλγόριθμος RSA:

1. Έστω οι ακέραιοι e, d, p, q (το μυστικό) και ο ακέραιος m (το μήνυμα)
2. Έστω ο ακέραιος $N = p \cdot q$.
3. Περιορισμός: όλοι οι ακέραιοι πρέπει να είναι θετικοί.
4. Περιορισμός: το μήνυμα m πρέπει να είναι μικρότερο του N.
5. Περιορισμός: οι ακέραιοι p και q είναι πρώτοι.
6. Περιορισμός: ο ακέραιος e είναι coprime με το $\phi(N)$.
7. Περιορισμός: οι ακέραιοι e και d είναι αντίστροφοι, δηλαδή: $e \cdot d \bmod \phi(N) = 1$.

Με βάση τους παραπάνω περιορισμούς, μπορούμε πλέον να ορίσουμε την συνάρτηση κρυπτογράφησης encrypt ως:

$$encrypt(m) = m^e \bmod N$$

Αντίστοιχα αν μας δώσουν έναν ακέραιο c ($= m^e \bmod N$) που είναι το κρυπτογραφημένο μήνυμα, μπορούμε να το αποκρυπτογραφήσουμε, χρησιμοποιώντας την συνάρτηση decrypt:

$$decrypt(c) = c^d \bmod N$$

Το γιατί η παραπάνω πράξη μας δίνει το αρχικό μας μηνύμα είναι μεγάλη ιστορία ($c^d \bmod N = (m^e)^d \bmod N = m^{e \cdot d} \bmod N = m^{1+k\phi(N)} \bmod N = m \bmod N$) αλλά μπορείτε να βρείτε όλα τα βήματα της απόδειξης στην Wikipedia.

Φτάσαμε επιτέλους στο ζητούμενο αυτής της άσκησης: να γράψετε ένα πρόγραμμα το οποίο να μπορεί να κρυπτογραφεί και να αποκρυπτογραφεί μηνύματα χρησιμοποιώντας τον παραπάνω αλγόριθμο RSA.

Τεχνικές Προδιαγραφές

- Repository Name: progintro/hw1-<YourUsername>
- C Filepath: rsa/src/rsa.c
- Το πρόγραμμά θα πρέπει να παίρνει 5 ορίσματα από την γραμμή εντολών στην μορφή `./rsa op e d p q`. Το πρώτο op επιτρέπεται να είναι ”enc” (για encryption) και ”dec” για decryption. Τα υπόλοιπα ορίσματα θα είναι ακέραιοι αριθμοί και θα είναι στο

εύρος $[-10^{18}, 10^{18}]$. Αν το πρόγραμμα εκτελεστεί με ορίσματα που δεν ακολουθούν τις παραπάνω προδιαγραφές, πρέπει να εκτυπώσει αντίστοιχο μήνυμα όπως στα παρακάτω παραδείγματα και να επιστρέφει με κωδικό εξόδου (exit code) 1.

- Το πρόγραμμα θα πρέπει να διαβάζει το μήνυμα m από την πρότυπη είσοδο (standard input) ως έναν δεκαδικό αριθμό επίσης στο εύρος $[-10^{18}, 10^{18}]$. Γενικά, δεν θα σας ζητηθεί να χρησιμοποιήσετε ακεραίους με περισσότερα από 64 bits.
- Αν η είσοδος του χρήστη δεν πληροί τους περιορισμούς του αλγορίθμου RSA το πρόγραμμά σας πρέπει να τερματίζει με κωδικό εξόδου 1 και αντίστοιχο μήνυμα λάθους όπως δείχνουμε παρακάτω στις ενδεικτικές εκτελέσεις.
- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr): `gcc -O3 -Wall -Wextra -Werror -pedantic -o rsa rsa.c`
- README Filepath: rsa/README.md
- Πρέπει να ολοκληρώνει την εκτέλεση μέσα σε: 1 δευτερόλεπτο.

Παρακάτω παραθέτουμε αλληλεπιδράσεις με μια ενδεικτική λύση. Ας δοκιμάσουμε να στείλουμε το μήνυμα ”42” ($m = 42$) χρησιμοποιώντας το μυστικό ”257 257 173 193” ($e = 257$, $d = 257$, $p = 173$, $q = 193$):

```
$ echo 42 | ./rsa enc 257 257 173 193
6990
$ echo $?
0
```

Άρα το κρυπτογραφημένο μήνυμα που μπορούμε να στείλουμε είναι ο αριθμός $c = 6990$. Είναι σωστός; Πρέπει να κάνουμε την πράξη $42^{257} \bmod (173 \cdot 193)$ το οποίο όντως αν χρησιμοποιήσουμε ένα online calculator φαίνεται σωστό (αντίστοιχα μπορείτε να κάνετε πειράματα με τους δικούς σας συνδυασμούς αριθμών). Για να δούμε αν μπορούμε να αποκρυπτογραφήσουμε το αρχικό μας μήνυμα:

```
$ echo 6990 | ./rsa dec 257 257 173 193
42
$ echo $?
0
```

Πήραμε όντως πίσω το αρχικό μας μήνυμα! Παρακάτω δοκιμάζουμε να κρυπτογραφήσουμε και να αποκρυπτογραφήσουμε διάφορους συνδυασμούς που καλύπτουν τις προδιαγραφές παραπάνω:

```
$ ./rsa
Usage: ./rsa enc|dec <exp_exp> <priv_exp> <prime1> <prime2>
$ echo $?
1
$ ./rsa pop 1 2 3 4
```

```
First argument must be 'enc' or 'dec'
$ echo $?
1
$ ./rsa enc 1 2 -3 4
Negative numbers are not allowed
$ echo $?
1
$ ./rsa enc 1 2 3 4
p and q must be prime
$ echo $?
1
$ ./rsa enc 3 6 17 19
e is not coprime with phi(N)
$ echo $?
1
$ ./rsa enc 5 6 17 19
e * d mod phi(N) is not 1
$ echo $?
1
$ echo 500 | ./rsa enc 5 173 17 19
Message is larger than N
$ echo $?
1
$ echo -42 | ./rsa enc 5 173 17 19
Negative numbers are not allowed
$ echo $?
1
$ echo 42 | ./rsa enc 5 173 17 19
264
$ echo 42 | ./rsa enc 17 26153 131 229
27187
$ echo 27187 | ./rsa dec 257 257 173 193
5343
$ echo 27187 | ./rsa dec 17 26153 131 229
42
$ echo 117 | ./rsa enc 17 26153 131 229 | ./rsa dec 17 26153 131 229
117
$ echo 43434343 | ./rsa enc 65537 2278459553 62971 38609 |
  ./rsa dec 65537 2278459553 62971 38609
43434343
$ echo 42 | ./rsa enc 65537 2278459553 62971 38609 > enc_msg
$ cat enc_msg
741088023
```

```
$ time ./rsa dec 65537 2278459553 62971 38609 < enc_msg
42

real    0m0.011s
user    0m0.004s
sys     0m0.008s
```

Καθώς οι εκθέτες του μυστικού στον οποίο υψώνουμε το μήνυμα γίνονται μεγαλύτεροι, είναι πιθανό πως το πρόγραμμά σας θα αρχίσει να παίρνει όλο και περισσότερο χρόνο για να τερματίσει. Αν παρατηρήσετε πως κάτι τέτοιο συμβαίνει και στον δικό σας αλγόριθμο, μπορείτε να δοκιμάσετε βελτιωμένους αλγόριθμους υπολογισμού της δύναμης ενός ακεραίου (Repeated Squaring, Modular Exponentiation).

Στο αρχείο README.md πρέπει να προσθέσετε οποιεσδήποτε παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.

Υπόδειξη Μην υπολογίσετε ποτέ ολόκληρο το m^e: παίρνετε το υπόλοιπο mod N μετά από κάθε πολλαπλασιασμό και, για μεγάλους εκθέτες, χρησιμοποιείτε ύψωση σε δύναμη με επαναλαμβανόμενο τετραγωνισμό (λογαριθμικά βήματα). Προσοχή στην υπερχείλιση: το γινόμενο δύο υπολοίπων κοντά στο N μπορεί να μη χωρά σε 64 bits. Κάντε τους ελέγχους με τη σειρά που δείχνουν τα παραδείγματα (ορίσματα, αρνητικοί, πρώτοι, coprime, αντίστροφοι, μέγεθος μηνύματος) και επαναχρησιμοποιήστε τον ΜΚΔ της άσκησης gcd.

Θέματα εξετάσεων (A11.22)

A11.22 · Ο Στέργιος Ξαναχτυπά

A11.22 · *Εξέταση Ιανουαρίου 2026, Θέμα 6* · ★☆☆ · trace · `exam-2026-jan-q6`

Ο Στέργιος Ξαναχτυπά [10 Μονάδες]

Ο Στέργιος κουράστηκε με τους μαθηματικούς γρίφους και αποφάσισε να ασχοληθεί με κώδικα συστημάτων. Ο μπαμπάς του—ο Μάκης—του έδωσε το ακόλουθο κομμάτι κώδικα:

```c
#include <stdio.h>

int main(void) {
    char *p1 = NULL;
    char *p2 = NULL;
    int n = 16;
    while (n-- > 0) {
        if (*p1++ != *p2++) {
            printf("Not Equal\n");
```

```
        }
    }
    printf("Equal\n");
    return 0;
}
```

Τι θα κάνει το παραπάνω κομμάτι κώδικα; Θα τυπώσει Equal, Not Equal, δεν κάνει compile, θα έχει κάποιο runtime error ή δεν μπορούμε να γνωρίζουμε; Αιτιολογήστε την απάντησή σας. Μη σωστά αιτιολογημένες απαντήσεις δεν θα λάβουν βαθμολογία.

Υπόδειξη Ξεκινήστε από το ερώτημα αν ο κώδικας περνάει από τον compiler: είναι συντακτικά σωστός; Μετά σκεφτείτε τι σημαίνει να διαβάσετε μέσω `*p1` όταν ο `p1` είναι `NULL`, τι λέει το πρότυπο της C γι' αυτό και τι συμβαίνει συνήθως στην πράξη. Αναρωτηθείτε επίσης αν ο compiler (π.χ. με `-O2`) έχει δικαίωμα να υποθέσει κάτι για έναν κώδικα που θα ήταν απροσδιόριστη συμπεριφορά.

Κεφάλαιο 12: Δείκτες και Πίνακες

Kahoot από το αμφιθέατρο (K12.1–K12.9)

K12.1 · Διαστάσεις πίνακα

K12.1 · *Kahoot «Δείκτες Παντού!» (διάλεξη 12) και «Πίνακες και Δείκτες» (διάλεξη 12)* · ★☆☆ · multiple-choice · 93% σωστές · `kahoot-3d-array-dims`

Ο πίνακας `double two[2][2][2]` είναι δισδιάστατος.

- True
- False

Υπόδειξη Οι διαστάσεις ενός πίνακα μετριούνται από το πλήθος των αγκυλών, όχι από τους αριθμούς μέσα τους.

K12.2 · Στατικοί και δυναμικοί πίνακες

K12.2 · *Kahoot «Δείκτες Παντού!» (διάλεξη 12) και «Πίνακες και Δείκτες» (διάλεξη 12)* · ★☆☆ · multiple-choice · 92% σωστές · `kahoot-static-vs-dynamic-arrays`

Δεν υπάρχει διαφορά ανάμεσα σε στατικούς και δυναμικούς πίνακες.

- True
- False

Υπόδειξη Σκεφτείτε πού βρίσκεται η μνήμη του καθενός, πότε ορίζεται το μέγεθός του και ποιος την αποδεσμεύει.

K12.3 · Πλήθος στοιχείων δισδιάστατου πίνακα

K12.3 · *Kahoot «Δείκτες Παντού!» (διάλεξη 12) και «Πίνακες και Δείκτες» (διάλεξη 12)* · ★☆☆ · multiple-choice · 81% σωστές · `kahoot-2d-array-count`

Ένας πίνακας `int array[4][5];` περιέχει πόσα `int` στοιχεία;

- `20`
- `12`
- `80`
- Εξαρτάται

Υπόδειξη Ένας δισδιάστατος πίνακας είναι ένας πίνακας από γραμμές, καθεμία με τις δικές της στήλες.

K12.4 · sizeof δισδιάστατου πίνακα

K12.4 · *Kahoot «Πίνακες και Δείκτες» (διάλεξη 12)* · ★★☆ · multiple-choice · 68% σωστές · `kahoot-sizeof-2d-double`

Έστω `sizeof(double) == 8` και πίνακας `double coeffs[10][10]`. Πόσο είναι το `sizeof(coeffs)`;

- `800`
- `100`
- `160`
- Εξαρτάται

Υπόδειξη Πόσα στοιχεία έχει συνολικά ο πίνακας, και πόσα bytes πιάνει το καθένα;

K12.5 · Το τελευταίο στοιχείο δισδιάστατου πίνακα

K12.5 · *Kahoot «Πίνακες και Δείκτες» (διάλεξη 12)* · ★★☆ · multiple-choice · 67% σωστές · `kahoot-2d-last-element`

Το στοιχείο της τελευταίας στήλης της τελευταίας γραμμής του πίνακα `int array[100][200];` είναι:

- `array[0][0]`

- `array[100][200]`
- `array[99][199]`
- `array[199][99]`

Υπόδειξη Ποια διάσταση είναι οι γραμμές και ποια οι στήλες, και από πού ξεκινά η αρίθμηση;

K12.6 · Ο τύπος του argv

K12.6 · *Kahoot «Δείκτες Παντού!» (διάλεξη 12) και «Πίνακες και Δείκτες» (διάλεξη 12)* · ★★☆ · multiple-choice · 44% σωστές · `kahoot-argv-type`

Η μεταβλητή `char *argv[]` είναι:

- Ένας δισδιάστατος πίνακας από χαρακτήρες
- Ένας δείκτης σε πίνακες από χαρακτήρες
- Ένας πίνακας από δείκτες σε χαρακτήρες
- Εξαρτάται

Συχνή παρανόηση Το 34% επέλεξε «Ένας δείκτης σε πίνακες από χαρακτήρες», διαβάζοντας τη δήλωση ανάποδα. Οι αγκύλες δένουν πιο ισχυρά από το `*`, άρα το `argv` είναι πρώτα πίνακας, και τα στοιχεία του είναι `char *`.

Υπόδειξη Διαβάστε τη δήλωση ξεκινώντας από το όνομα: πρώτα εφαρμόζονται οι αγκύλες και μετά το `*`.

K12.7 · sizeof μιας γραμμής

K12.7 · *Kahoot «Πίνακες και Δείκτες» (διάλεξη 12)* · ★★☆ · multiple-choice · 42% σωστές · `kahoot-sizeof-row`

Έστω `sizeof(char) == 1` και πίνακας `char map[10][10]`. Πόσο είναι το `sizeof(map[5])`;

- `1`
- `10`
- `100`
- Εξαρτάται

Συχνή παρανόηση Το 26% επέλεξε `1`, θεωρώντας ότι το `map[5]` είναι ένας χαρακτήρας. Στην πραγματικότητα το `map[5]` είναι ολόκληρη η 6η γραμμή, ένας πίνακας 10 `char`.

Υπόδειξη Με έναν μόνο δείκτη σε δισδιάστατο πίνακα, τι παίρνουμε: ένα στοιχείο ή μια ολόκληρη γραμμή;

K12.8 · Χαρακτήρας από πίνακα συμβολοσειρών

K12.8 · *Kahoot «Δείκτες Παντού!» (διάλεξη 12) και «Πίνακες και Δείκτες» (διάλεξη 12)* · ★★★ · multiple-choice · 38% σωστές · `kahoot-string-array-element`

Έστω ότι

```c
char *strings[] = { "Hello", "fine", "world", NULL };
```

Το στοιχείο `strings[1][2]` είναι:

- `'e'`
- `NULL`
- `"fine"`
- `'n'`

Συχνή παρανόηση Το 23% επέλεξε `"fine"` και άλλο 22% `'e'`: οι πρώτοι σταμάτησαν στο `strings[1]` αγνοώντας τον δεύτερο δείκτη, οι δεύτεροι μέτρησαν τις θέσεις από το 1.

Υπόδειξη Ο πρώτος δείκτης διαλέγει συμβολοσειρά και ο δεύτερος χαρακτήρα μέσα σε αυτή· και οι δύο μετράνε από το 0.

K12.9 · Διεύθυνση στοιχείου δισδιάστατου πίνακα

K12.9 · *Kahoot «Πίνακες και Δείκτες» (διάλεξη 12)* · ★★★ · multiple-choice · 33% σωστές · `kahoot-2d-element-address`

Έστω ότι η διεύθυνση του `int array[10][10]` είναι 100 και `sizeof(int) == 4`. Η διεύθυνση του `array[5][5]` είναι:

- `320`
- `145`
- `155`
- Εξαρτάται

Συχνή παρανόηση Το 29% επέλεξε `155`: μέτρησαν σωστά ότι προηγούνται 55 στοιχεία, αλλά ξέχασαν να τα πολλαπλασιάσουν με `sizeof(int)` για να τα κάνουν bytes.

Υπόδειξη Ο πίνακας αποθηκεύεται κατά γραμμές: πόσα στοιχεία προηγούνται του `array[5][5]`, και πόσα bytes είναι το καθένα;

Ζέσταμα: από τις διαφάνειες (A12.1–A12.7)

A12.1 · Στοιχείο δισδιάστατου πίνακα

A12.1 · *Διάλεξη 12, διαφάνεια 14* · ★☆☆ · trace · `slides-lec12-2d-element`

Δίνεται ο πίνακας:

```
int array[2][4] = {
    {1, 4, 7, 10},
    {3, 6, 9, 12},
};
```

Ποιο είναι το στοιχείο `a[1][1]` (δηλαδή `array[1][1]`);

Υπόδειξη Ο πρώτος δείκτης επιλέγει τη γραμμή και ο δεύτερος τη στήλη, και οι δύο μετρούν από το 0. Γράψτε τον πίνακα σε μορφή γραμμών και στηλών και βρείτε το κελί.

A12.2 · Πίνακας από δείκτες

A12.2 · *Διάλεξη 12, διαφάνεια 29* · ★☆☆ · trace · `slides-lec12-array-of-pointers`

Οι δείκτες είναι μεταβλητές και μπορούν να μπουν σε πίνακες. Για παράδειγμα, τι θα τυπώσει το ακόλουθο;

```
#include <stdio.h>
int main() {
  int *ptr[3], a = 100, b = 200, c = 300;
  ptr[0] = &a;
  ptr[1] = &b;
  ptr[2] = &c;
  printf("%d %d %d\n", *ptr[2], *ptr[1], *ptr[0]);
  return 0;
}
```

Υπόδειξη Το `*ptr[2]` διαβάζεται ως `*(ptr[2])`: πρώτα παίρνουμε το στοιχείο του πίνακα (έναν δείκτη) και μετά την τιμή όπου αυτός δείχνει. Προσέξτε τη σειρά των ορισμάτων στο `printf`.

A12.3 · Δείκτης σε δείκτη

A12.3 · *Διάλεξη 12, διαφάνεια 27* · ★☆☆ · trace · `slides-lec12-pointer-to-pointer`

Τι θα τυπώσει το παρακάτω;

```
int x = 42;
int * ptr = &x;
int ** ptr2 = &ptr;
printf("%p %d", *ptr2, **ptr2);
```

Στη διαφάνεια η μνήμη σχεδιάζεται ως εξής: το x βρίσκεται στη διεύθυνση 100 με τιμή 42, ο `ptr` στη διεύθυνση 200 με τιμή 100, και ο `ptr2` στη διεύθυνση 400 με τιμή 200.

Υπόδειξη Κάθε `*` «ακολουθεί ένα βέλος»: ξεκινήστε από την τιμή του `ptr2` και βρείτε τι υπάρχει στη διεύθυνση όπου δείχνει, μία φορά για το `*ptr2` και δύο φορές για το `**ptr2`. Σκεφτείτε επίσης σε τι μορφή τυπώνει το `%p`.

A12.4 · Παράδειγμα endianness

A12.4 · *Διάλεξη 12, διαφάνεια 42* · ★★☆ · trace · `slides-lec12-endianness`

Τι θα τυπώσει το παρακάτω;

```
#include <stdio.h>
int main() {
  int x = 42;
  char * bytes = (char*)&x;
  int i;
  for(i = 0; i < sizeof(int) / sizeof(char); i++)
     printf("%02x\n", bytes[i]);
  return 0;
}
```

Υπόδειξη Γράψτε το 42 σε δεκαεξαδικό ως ακέραιο 4 bytes. Ο δείκτης `bytes` διατρέχει τη μνήμη του x ένα byte τη φορά, από τη χαμηλότερη διεύθυνση· η απάντηση εξαρτάται από το αν ο υπολογιστής σας είναι little ή big endian (οι x86 και οι περισσότεροι ARM είναι little endian).

A12.5 · Πόση μνήμη δεσμεύει η malloc και τι λέει το sizeof

A12.5 · *Διάλεξη 12, διαφάνειες 39–40* · ★★☆ · short-answer · `slides-lec12-malloc-sizeof`

```
int * nums = malloc(100 * sizeof(int));
double * coeffs = malloc(100 * sizeof(double));
```

```
char * str = malloc(100 * sizeof(char));
```

1. Πόση μνήμη δεσμεύεται με καθεμιά από τις παραπάνω κλήσεις;
2. Τι θα τυπώσει το ακόλουθο;

```
printf("%d %d %d\n", sizeof(nums), sizeof(coeffs), sizeof(str));
```

Υπόδειξη Για το (1), η `malloc` μετρά bytes: υπολογίστε `100 * sizeof(τύπος)` με τα συνηθισμένα μεγέθη των τύπων. Για το (2), αναρωτηθείτε ποιος είναι ο τύπος των `nums`, `coeffs` και `str`: πίνακες ή δείκτες; Και τι μέγεθος έχει μια διεύθυνση σε ένα σύστημα 64 bit;

A12.6 · Είναι απαραίτητοι οι πολυδιάστατοι πίνακες;

A12.6 · *Διάλεξη 12, διαφάνεια 23* · ★★☆ · short-answer · `slides-lec12-multidim-necessary`

Πίνακες υψηλότερων διαστάσεων λειτουργούν με τον ίδιο τρόπο, προσθέτοντας τον επιθυμητό αριθμό διαστάσεων, π.χ. ένας κύβος του Rubik:

```
int rubiksCube[6][3][3] = {
    {{0, 0, 0}, {0, 0, 0}, {0, 0, 0}}, // Face 1 (e.g., Red)
    {{1, 1, 1}, {1, 1, 1}, {1, 1, 1}}, // Face 2 (e.g., Green)
    {{2, 2, 2}, {2, 2, 2}, {2, 2, 2}}, // Face 3 (e.g., Blue)
    {{3, 3, 3}, {3, 3, 3}, {3, 3, 3}}, // Face 4 (e.g., Yellow)
    {{4, 4, 4}, {4, 4, 4}, {4, 4, 4}}, // Face 5 (e.g., Orange)
    {{5, 5, 5}, {5, 5, 5}, {5, 5, 5}}  // Face 6 (e.g., White)
};
```

Είναι απαραίτητοι οι πολυδιάστατοι πίνακες;

Υπόδειξη Θυμηθείτε πώς αποθηκεύεται ένας δισδιάστατος πίνακας στη μνήμη και τον τύπο `i * Y + j` για τη θέση του `a[i][j]`. Τι θα χρειαζόταν για να κρατήσετε τα ίδια δεδομένα σε έναν μονοδιάστατο πίνακα, και τι κερδίζετε όταν τον λογαριασμό τον κάνει ο μεταγλωττιστής;

A12.7 · Περιεχόμενα του x μετά από βρόχο με δείκτη

A12.7 · *Διάλεξη 12, διαφάνεια 25* · ★★☆ · trace · `slides-lec12-pointer-copy-loop`

Ποια είναι τα περιεχόμενα του x μετά την εκτέλεση;

```
int * p;
int x[] = {5, 7, 2, 3, 6, 0, 1, 4};
p = x;
while (*p = *(p+2))
  p++;
```

Υπόδειξη Η συνθήκη του `while` είναι ανάθεση, όχι σύγκριση: αντιγράφει στο `*p` την τιμή δύο θέσεις πιο μετά, και ο βρόχος σταματά όταν η τιμή που αντιγράφηκε είναι 0. Ιχνηλατήστε βήμα-βήμα σε πίνακα τη θέση του `p` και τα περιεχόμενα του `x`, θυμίζοντας στον εαυτό σας ότι ο πίνακας αλλάζει καθώς προχωρά ο βρόχος.

Εργαστήριο (A12.8–A12.11)

A12.8 · Δυναμική δέσμευση μνήμης για μονοδιάστατο πίνακα

A12.8 · *Εργαστήριο 7, Άσκηση 2* · ★☆☆ · programming · `lab-lab07-array`

2.1 Κατασκευάστε το πρόγραμμα `array.c` που να διαβάζει από την είσοδο τη διάσταση ενός μονοδιάστατου πίνακα ακεραίων (έστω `N`), να δεσμεύει χώρο `N` θέσεων δυναμικά και έπειτα να τον αρχικοποιεί διαβάζοντας από την είσοδο ακέραιους αριθμούς.

> **Δυναμική δέσμευση μνήμης** για μονοδιάστατο πίνακα
>
> Συνάρτηση `void *malloc(size_t size)`
>
> Χρήση:
>
> - `ΤΔ *p;` — δήλωση ενός δείκτη σε στοιχεία τύπου *ΤΔ*
> - `p = malloc(N * sizeof(ΤΔ));` — δέσμευση μνήμης για N στοιχεία τύπου *ΤΔ*
>
> Η `malloc` επιστρέφει `NULL` σε περίπτωση αποτυχίας δέσμευσης της αιτούμενης μνήμης.
>
> **Αποδέσμευση μνήμης** για δυναμικά δεσμευμένους πίνακες
>
> Συνάρτηση `void free(void *p)`
>
> Χρήση: `free(p);` όπου `p` είναι δείκτης σε θέσεις μνήμης που έχουν δεσμευθεί δυναμικά.

Όταν χρησιμοποιούνται οι συναρτήσεις `malloc()` και `free()`, πρέπει να γίνεται συμπερίληψη του αρχείου επικεφαλίδας `stdlib.h`.

Γράψτε μια συνάρτηση `print_array` που να εκτυπώνει τα στοιχεία του πίνακα σε μία γραμμή χωρισμένα με στηλογνώμονα (tab).

2.2 Κατασκευάστε το αρχείο `input.txt` το οποίο να περιέχει `N+1` ακέραιους αριθμούς ως εξής: Ο πρώτος αριθμός είναι το πλήθος των στοιχείων (`N`) και ακολουθούν οι `N` ακέραιοι χωρισμένοι με κενά.

2.3 Εκτελέστε το πρόγραμμα `array` με ανακατεύθυνση εισόδου από το αρχείο `input.txt`.

2.4 Επεκτείνετε το πρόγραμμά σας, με μια συνάρτηση `print_average` ώστε να εκτυπώνεται ο μέσος όρος των στοιχείων του πίνακα.

Υπόδειξη Διαβάστε πρώτα το `N`, δεσμεύστε και ελέγξτε αμέσως αν η `malloc` επέστρεψε `NULL`. Μετά ο δείκτης χρησιμοποιείται ακριβώς σαν πίνακας (`p[i]`). Μην ξεχάσετε την `free` στο τέλος, και τη μετατροπή σε `double` στον μέσο όρο.

A12.9 · Ορίσματα γραμμής εντολής

A12.9 · *Εργαστήριο 8, Άσκηση 2* · ★☆☆ · programming · `lab-lab08-argcalc`

2.1 Κατασκευάστε το πρόγραμμα `argcalc.c` που να εκτελεί απλές αριθμητικές πράξεις (πρόσθεση, αφαίρεση, πολλαπλασιασμό, πηλίκο διαίρεσης και υπόλοιπο διαίρεσης) μεταξύ ακεραίων. Οι πράξεις που θα γίνονται να δίνονται σαν ορίσματα στη γραμμή εντολής.

Παραδείγματα εκτέλεσης ακολουθούν:

```
$ ./argcalc 12 + 18
30
$ ./argcalc 70 % 12
10
```

> Προσοχή: τον χαρακτήρα `*` τον επεκτείνει το shell πριν φτάσει στο πρόγραμμά μας, οπότε για τον πολλαπλασιασμό τον γράφουμε σε εισαγωγικά (`./argcalc 6 '*' 7`) ή με ανάποδη κάθετο (`./argcalc 6 \* 7`).

Ορίσματα Γραμμής Εντολής

Ορισμός της συνάρτησης `main`:

```
int main(int argc, char *argv[])
```

- Η μεταβλητή `argc` αρχικοποιείται με το πλήθος των ορισμάτων `N + 1`, τα οποία βρίσκονται στις θέσεις `argv[0]`, ..., `argv[N]` του πίνακα συμβολοσειρών `argv`. Το `argv[0]` είναι το όνομα του προγράμματος και το `argv[N+1]` είναι `NULL`.
- Η συνάρτηση `int atoi(const char *s)` επιστρέφει στο όνομά της την αριθμητική τιμή που αντιστοιχεί στην (αριθμητική) συμβολοσειρά `s`.

Υπόδειξη Ελέγξτε πρώτα ότι το `argc` είναι 4. Οι αριθμοί είναι τα `argv[1]` και `argv[3]` (ως συμβολοσειρές, άρα χρειάζονται μετατροπή), και ο τελεστής είναι ο πρώτος χαρακτήρας του `argv[2]`, πάνω στον οποίο μπορείτε να κάνετε `switch`. Σκεφτείτε τι πρέπει να γίνει όταν ο διαιρέτης είναι 0.

A12.10 · Δισδιάστατοι πίνακες

A12.10 · *Εργαστήριο 7, Άσκηση 1* · ★★☆ · programming · `lab-lab07-twodim`

1.1 Κατασκευάστε το πρόγραμμα `twodim.c` που ορίζει στατικά έναν πίνακα A διαστάσεων 6 x 10 και αρχικοποιεί το στοιχείο `A[i][j]` που βρίσκεται στη γραμμή i και στη στήλη j, σύμφωνα με τον τύπο:

$$A[i][j] = 5 \cdot (5 - i) + j \cdot (9 - j)$$

Φτιάξτε μια συνάρτηση `print_array` που να εκτυπώνει τον πίνακα κατά γραμμές, με τα στοιχεία κάθε γραμμής χωρισμένα με στηλογνώμονα (tab).

1.2 Επεκτείνετε το πρόγραμμα `twodim.c` με μια συνάρτηση `print_transposed` ώστε να εκτυπώνει *και* τον ανάστροφο του πίνακα, δηλαδή αυτόν που έχει στήλες τις γραμμές του αρχικού και γραμμές τις στήλες του αρχικού (αυτή είναι μια κατεξοχήν άσκηση προγραμματιστικής συνέντευξης).

Επομένως, αν ένας αρχικός πίνακας 3 x 5 που εκτυπώνεται έτσι:

```
1 4 6 7 8
5 6 7 1 3
1 0 4 7 2
```

όταν τον τυπώνετε ανάστροφα, θέλουμε να τυπώνεται 5 x 3, δηλαδή ως εξής:

```
1 5 1
4 6 0
6 7 4
7 1 7
8 3 2
```

1.3 Επεκτείνετε το πρόγραμμα `twodim.c` με μια συνάρτηση `print_reverse` ώστε να εκτυπώνει τα στοιχεία κάθε γραμμής του αρχικού πίνακα με αντίστροφη σειρά. Για παράδειγμα, αν ο αρχικός σας πίνακας είναι και πάλι:

```
1 4 6 7 8
5 6 7 1 3
1 0 4 7 2
```

όταν τυπωθεί με αντίστροφη σειρά, θέλουμε να τυπώσει κάτι τέτοιο:

```
8 7 6 4 1
3 1 7 6 5
2 7 4 0 1
```

1.4 Επεκτείνετε το πρόγραμμα `twodim.c` ώστε να εκτυπώνει όλα τα στοιχεία του πίνακα σε μία σειρά, διασχίζοντάς τον με έναν οφιοειδή τρόπο, όπως φαίνεται στο παράδειγμα:

Ο αρχικός πίνακας:

```
1 4 6 7 8
```

```
5 6 7 1 3
1 0 4 7 2
```

θέλουμε να εκτυπωθεί ως εξής:

```
1 4 6 7 8 3 1 7 6 5 1 0 4 7 2
```

Υπόδειξη Όταν περνάτε στατικό δισδιάστατο πίνακα σε συνάρτηση, η δεύτερη διάσταση πρέπει να φαίνεται στον τύπο της παραμέτρου (ορίστε τις διαστάσεις με `#define`). Όλες οι παραλλαγές είναι δύο εμφωλευμένοι βρόχοι: στον ανάστροφο αλλάζει ποιος δείκτης είναι εξωτερικός, στην αντίστροφη η εσωτερική μέτρηση πάει προς τα πίσω, και στην οφιοειδή η φορά της γραμμής εξαρτάται από το αν η γραμμή είναι άρτια ή περιττή.

A12.11 · Πετυχαίνοντας τον στόχο (Παλιό θέμα)

A12.11 · *Εργαστήριο 8, Άσκηση 3* · ★★☆ · programming · `lab-lab08-legolas`

Γράψτε ένα πρόγραμμα το οποίο παίρνει ως ορίσματα από την γραμμή εντολών έναν ακέραιο-στόχο (goal το `argv[1]`) και στην συνέχεια ένα σύνολο υποψηφίων ακεραίων (candidates) και τυπώνει όλους τους συνδυασμούς 3 υποψηφίων των οποίων το άθροισμα ισούται με τον στόχο. Η σειρά με την οποία εκτυπώνονται τα αποτελέσματα δεν έχει σημασία για την ορθότητα του προγράμματος. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o legolas legolas.c
$ ./legolas 42 19 21 3 5 12
No combination of candidates leads to 42
$ ./legolas 42 19 21 3 5 12 11
Candidates combination found: 19 + 12 + 11 = 42
$ ./legolas 42 27 25 12 31 5 26 40 34 3 18
Candidates combination found: 27 + 12 + 3 = 42
Candidates combination found: 25 + 12 + 5 = 42
Candidates combination found: 5 + 34 + 3 = 42
$ ./legolas 37372082074 9238742398 82934723 27893492387 127863435 239847289
Candidates combination found: 9238742398 + 27893492387 + 239847289 = 37372082074
```

Υπόδειξη Διαλέξτε τριάδες διαφορετικών θέσεων με τρεις εμφωλευμένους βρόχους όπου κάθε δείκτης ξεκινά μετά τον προηγούμενο, ώστε κάθε συνδυασμός να εμφανίζεται μία φορά. Το τελευταίο παράδειγμα έχει αριθμούς που δεν χωρούν σε `int`, οπότε η μετατροπή από συμβολοσειρά πρέπει να γίνει σε `long long`. Μην ξεχάσετε το μήνυμα όταν δεν βρεθεί κανένας συνδυασμός.

Εργασίες (A12.12)

A12.12 · FauxtoShop: περιστροφή εικόνας BMP

A12.12 · *Εργασία 2 (2023-24), Άσκηση 1* · ★★★ · programming · `hw-2023-hw2-fauxtoshop`

Πολύ συχνά βγάζουμε φωτογραφίες που δεν έχουν τον σωστό προσανατολισμό και θέλουμε να τις περιστρέψουμε, χωρίς τους περιορισμούς (κόστος, μέγεθος εικόνας) των προγραμμάτων επεξεργασίας εικόνας.

Το ζητούμενο σε αυτήν την άσκηση είναι να υλοποιήσουμε το δικό μας `fauxtoshop` που μας επιτρέπει να περιστρέψουμε εικόνες τύπου BMP (BitMaP) δεξιόστροφα κατά 90° (με την φορά του ρολογιού). Προκειμένου να το καταφέρουμε αυτό, θα χρειαστεί να μάθουμε λίγο παραπάνω για την μορφή των αρχείων BMP. Σε αυτήν την άσκηση θα ασχοληθούμε με μια κατηγορία αρχείων BMP τα οποία έχουν την μορφή κεφαλίδων (headers) ακολουθούμενη από κάποια bytes και έναν δισδιάστατο πίνακα εικονοστοιχείων (pixels):

```
Headers (H bytes) | Other Data (M bytes) | Pixel 2-D array (N bytes)
```

Τα Σχήματα 1 και 2 της εκφώνησης παρουσιάζουν την ανάλυση των περιεχομένων ενός ενδεικτικού αρχείου `color6x6.bmp`. Για τις ανάγκες της άσκησης, μπορείτε να υποθέσετε ότι $H \geq 54$ και ότι τα δεδομένα ανάμεσα στις κεφαλίδες Other Data δεν χρειάζονται αλλαγή αλλά πρέπει να αντιγραφούν όπως είναι.

Σχήμα 1 (τα 54 πρώτα bytes της κεφαλίδας, με την εντολή `xxd`):

```
$ xxd -l 54 -g 1 -c 14 < color6x6.bmp
00000000: 42 4d ae 00 00 00 00 00 00 00 36 00 00 00  BM........6...
0000000e: 28 00 00 00 06 00 00 00 06 00 00 00 01 00  (.............
0000001c: 18 00 00 00 00 00 78 00 00 00 c4 0e 00 00  ......x.......
0000002a: c4 0e 00 00 00 00 00 00 00 00 00 00        ............
```

Τα σχόλια του σχήματος εξηγούν τα καίρια πεδία (απαραίτητα για την άσκηση):

- Τα αρχεία bmp αρχίζουν με τους χαρακτήρες 'B' και 'M' (magic number 0x42 0x4d).
- Στο 2ο byte της κεφαλίδας περιέχεται το μέγεθος του αρχείου (εδώ 174 bytes).
- Στο 10ο byte της κεφαλίδας περιέχεται το offset του πίνακα των pixels σε μορφή little endian (εδώ 54).
- Στο 18ο byte της κεφαλίδας περιέχεται το πλάτος (width) της εικόνας σε pixels σε μορφή little endian.
- Στο 22ο byte της κεφαλίδας περιέχεται το ύψος (height) της εικόνας σε pixels σε μορφή little endian.
- Στο 34ο byte της κεφαλίδας περιέχεται το συνολικό μέγεθος της εικόνας σε bytes σε μορφή little endian.

Σχήμα 2 (μόνο τα bytes των pixels, παραλείποντας τα 54 πρώτα):

```
$ xxd -s 54 -c 20 -g 3 < color6x6.bmp
```

```
00000036: 00ffff 00ffff 00ffff ff0000 ff0000 ff0000 0000
0000004a: 00ffff 00ffff 00ffff ff0000 ff0000 ff0000 0000
0000005e: 00ffff 00ffff 00ffff ff0000 ff0000 ff0000 0000
00000072: 0000ff 0000ff 0000ff 00ff00 00ff00 00ff00 0000
00000086: 0000ff 0000ff 0000ff 00ff00 00ff00 00ff00 0000
0000009a: 0000ff 0000ff 0000ff 00ff00 00ff00 00ff00 0000
```

Κάθε pixel είναι 3 bytes με σειρά B, G, R. Κάθε γραμμή έχει στο τέλος 2 bytes padding ώστε κάθε γραμμή να είναι πολλαπλάσιο του 4. Η πρώτη γραμμή του αρχείου είναι η κάτω γραμμή της εικόνας: το `pixels[0][0]` είναι το κάτω αριστερά pixel (κίτρινο) και το `pixels[4][1]` ένα κόκκινο pixel της πάνω αριστερά περιοχής. Η εικόνα έχει κίτρινο τεταρτημόριο κάτω αριστερά, μπλε κάτω δεξιά, κόκκινο πάνω αριστερά και πράσινο πάνω δεξιά. Δίνουμε **προσοχή στο padding** που μπορεί να έχει το κάθε αρχείο bmp το οποίο δεν έχει απεικόνιση για τους χρήστες.

Τεχνικές Προδιαγραφές

- Repository Name: `progintro/hw2-<YourUsername>`
- C Filepath: `fauxtoshop/src/fauxtoshop.c`
- Ένα αρχείο εισόδου/εξόδου BMP είναι δεκτό μόνο εάν:
  - Ξεκινάει την μαγική κεφαλίδα με τα bytes: 'B' 'M'.
  - Χρησιμοποιεί 24-bit για την αναπαράσταση χρώματος (μονοχρωματικά BMP δεν είναι δεκτά).
  - Χρησιμοποιεί τουλάχιστον 54 (14 + 40) bytes για την αναπαράσταση των κεφαλίδων ακολουθώντας το Windows 3.x format (Bitmap file header + DIB header). Για παράδειγμα, αρχεία που είναι συμβατά με την μορφή κεφαλίδας που δείχνουμε στο Σχήμα 1 είναι δεκτά.
  - Έχει σωστό μέγεθος αρχείου (τα μεγέθη στις κεφαλίδες συμφωνούν με τα περιεχόμενα του αρχείου).
  - Έχει σωστό μήκος, πλάτος και padding.
- Το πρόγραμμά θα πρέπει να παίρνει είσοδο από το standard input. Για οποιαδήποτε είσοδο δεν είναι μέσα στις προδιαγραφές το πρόγραμμα πρέπει να επιστρέφει με κωδικό εξόδου (exit code) 1.
- Ένα αρχείο `image.bmp` μπορεί να περαστεί στο πρόγραμμά μας με ανακατεύθυνση (redirection) στο stdin και αντίστοιχα η έξοδος μπορεί να γραφτεί σε ένα αρχείο `rotated.bmp` και πάλι με ανακατεύθυνση του stdout: `$ ./fauxtoshop < image.bmp > rotated.bmp`
- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr): `gcc -O3 -Wall -Wextra -Werror -pedantic -o fauxtoshop fauxtoshop.c`
- README Filepath: `fauxtoshop/README.md`
- **Ένα αρχείο που να περιέχει στοιχεία εισόδου και ένα εξόδου διαφορετικά από αυτά της άσκησης. Συγκεκριμένα προτείνουμε να βάλετε έναν συνδυασμό που θεωρείται**

ότι είναι δύσκολος να γίνει σωστός.
  - input Filepath: `fauxtoshop/test/input.bmp`
  - output Filepath: `fauxtoshop/test/output.bmp`
- Πρέπει να ολοκληρώνει την εκτέλεση μέσα σε: 10 δευτερόλεπτα.
- **Δεν επιτρέπεται** η χρήση δομών/εγγραφών (`struct`).

Παρακάτω παραθέτουμε την αλληλεπίδραση με μια ενδεικτική λύση (το Σχήμα 3 της εκφώνησης δείχνει την εικόνα `Apparition_of_Face_and_Fruit.bmp` πριν και μετά την περιστροφή):

```
$ gcc -O3 -Wall -Wextra -Werror -pedantic -o fauxtoshop fauxtoshop.c
$ ./fauxtoshop < ./fauxtoshop
Error: not a BMP file
$ echo $?
1
$ ./fauxtoshop < color6x6.bmp > color6x6.90.bmp
$ ./fauxtoshop < color6x6.90.bmp > color6x6.180.bmp
$ ./fauxtoshop < color6x6.180.bmp > color6x6.270.bmp
$ ./fauxtoshop < color6x6.270.bmp > color6x6.360.bmp
$ md5sum color6x6*.bmp
755c07991c51c0b3af855ba7f668c7d0  color6x6.180.bmp
5f19168fa9511a6520ec5dba2db4b478  color6x6.270.bmp
33855d8642cf110f74464f8ff78449aa  color6x6.360.bmp
6d62a0c4264dbfa962b372b6e5c8b9be  color6x6.90.bmp
33855d8642cf110f74464f8ff78449aa  color6x6.bmp
$ wc -c color6x6*.bmp
174 color6x6.180.bmp
174 color6x6.270.bmp
174 color6x6.360.bmp
174 color6x6.90.bmp
174 color6x6.bmp
$ ./fauxtoshop < Apparition_of_Face_and_Fruit.bmp > app90.bmp
$ md5sum Apparition_of_Face_and_Fruit.bmp app90.bmp
841d3634ba8c6af641d86b63ac1b6dfc  Apparition_of_Face_and_Fruit.bmp
96e8c33582f8a922a6665d4891e62a77  app90.bmp
$ wc -c Apparition_of_Face_and_Fruit.bmp app90.bmp
1440054 Apparition_of_Face_and_Fruit.bmp
1440054 app90.bmp
2880108 total
$ ./fauxtoshop < school_of_athens.bmp > school_of_athens90.bmp
$ md5sum school_of_athens*.bmp
282d01bbd23044a7e6ca91f0dda2238f  school_of_athens90.bmp
112ce7f3a60265992dc939949232257d  school_of_athens.bmp
$ wc -c school_of_athens*.bmp
15252534 school_of_athens.bmp
```

```
15257654 school_of_athens90.bmp
30510188 total
$ file school_of_athens.bmp school_of_athens90.bmp
school_of_athens.bmp:   PC bitmap, Windows 3.x format, 2560 x 1986 x 24,
                        image size 15252480, cbSize 15252534, bits offset 54
school_of_athens90.bmp: PC bitmap, Windows 3.x format, 1986 x 2560 x 24,
                        image size 15257600, cbSize 15257654, bits offset 54
$ ./fauxtoshop < school_of_athens.bmp |./fauxtoshop | ./fauxtoshop |
          ./fauxtoshop > school_of_athens360.bmp
$ md5sum school_of_athens*.bmp
112ce7f3a60265992dc939949232257d  school_of_athens360.bmp
282d01bbd23044a7e6ca91f0dda2238f  school_of_athens90.bmp
112ce7f3a60265992dc939949232257d  school_of_athens.bmp
```

Τα παραπάνω bmp αρχεία είναι στο: https://github.com/progintro/data/tree/main/bmp άλλα μπορείτε να δοκιμάσετε και άλλα παραδείγματα εικόνων που βρίσκετε online. Εκτός από `scp` για να τα αντιγράψετε στο Linux lab, μπορείτε να τα κατεβάσετε και μέσω κονσόλας χρησιμοποιώντας `curl`, για παράδειγμα:

```
$ curl -O https://raw.githubusercontent.com/progintro/data/main/bmp/color6x6.bmp
```

Ως συνήθως, στο αρχείο `README.md` πρέπει να προσθέσετε οποιεσδήποτε παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.

Υπόδειξη Διαβάστε ολόκληρο το stdin σε έναν δυναμικά δεσμευμένο πίνακα από `unsigned char` και γράψτε μια βοηθητική συνάρτηση που διαβάζει ακέραιο 4 bytes σε little endian από δοσμένο offset. Υπολογίστε το μέγεθος κάθε γραμμής με το padding (στρογγυλοποίηση του 3·width στο επόμενο πολλαπλάσιο του 4) για την είσοδο *και* για την έξοδο, όπου πλάτος και ύψος αλλάζουν θέση· ενημερώστε στην κεφαλίδα της εξόδου πλάτος, ύψος, μέγεθος εικόνας και αρχείου. Δουλέψτε πρώτα την αντιστοίχιση θέσεων (γραμμή, στήλη) στο χαρτί με τη μικρή `color6x6.bmp`, θυμηθείτε ότι οι γραμμές αποθηκεύονται από κάτω προς τα πάνω, και δοκιμάστε με μια εικόνα μη τετράγωνη με πλάτος που δεν είναι πολλαπλάσιο του 4.

Δείτε επίσης

- Υποδειγματική υποβολή φοιτητή (περιέχει λύση: δείτε την αφού λύσετε την άσκηση)

Θέματα εξετάσεων (A12.13–A12.22)

A12.13 · Αλλαγή Τέρματος

A12.13 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #13, Θέμα 2* · ★☆☆ · programming · exam-

`2023-fall-ex13-q2`

Πρόγραμμα: `change.c`

Γράψτε ένα πρόγραμμα το οποίο διαβάζει από την πρότυπη είσοδο (stdin) την μορφή ενός ποδοσφαιρικού γηπέδου σε ορθογώνια μορφή και την τυπώνει στην πρότυπη έξοδο (stdout) μετά από περιστροφή 180 μοιρών. Στην πρώτη γραμμή δίνονται οι ακέραιες διαστάσεις του γηπέδου (πλάτος και μήκος) και στην συνέχεια δίνονται τα περιεχόμενα του γηπέδου ως ακέραιοι - οι παίκτες της 1ης ομάδας αναπαρίστανται με 1, της 2ης ομάδας με 2 και ο κενός χώρος με 0. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ gcc -o change change.c
$ cat field.txt
9 31
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0
0 0 0 0 2 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
$ ./change < field.txt
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 2 0 0 0 0 0
0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Υπόδειξη Η περιστροφή κατά 180 μοίρες αντιστρέφει και τη σειρά των γραμμών και τη σειρά των στοιχείων κάθε γραμμής. Διαβάστε τις διαστάσεις, δεσμεύστε δυναμικά χώρο και ελέγξτε ότι κάθε τιμή είναι 0, 1 ή 2 και ότι το πλήθος τους είναι σωστό.

A12.14 · Τουρνουά

A12.14 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #2 (Pokémon Themed), Θέμα 2* · ★☆☆ · programming · `exam-2023-fall-ex2-q2`

Πρόγραμμα: `tournament.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που δέχεται ως ορίσματα όλα τα pokémon που διαγωνίζονται στο τουρνουά και στην συνέχεια τυπώνει στην πρότυπη έξοδο όλους τους αγώνες που θα γίνουν. Κάθε pokémon πρέπει να παίξει με κάθε άλλο ακριβώς δύο φορές. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ ./tournament charmander dratini mew
charmander vs dratini
charmander vs mew
dratini vs charmander
dratini vs mew
mew vs charmander
mew vs dratini
```

Υπόδειξη Δύο εμφωλευμένοι βρόχοι πάνω στο `argv` αρκούν· σκεφτείτε ποιο ζεύγος δεικτών πρέπει να παραλείψετε ώστε κάθε ζευγάρι να παίζει ακριβώς δύο φορές (μία «εντός» και μία «εκτός»). Τι πρέπει να γίνει αν δοθούν λιγότερα από δύο pokémon ή το ίδιο όνομα δύο φορές;

A12.15 · Μέση Τιμή Τυχαίων Μεταβλητών - mean

A12.15 · *Εξέταση Σεπτεμβρίου 2025, Θέμα 3* · ★☆☆ · programming · `exam-2025-sep-q3`

Μέση Τιμή Τυχαίων Μεταβλητών - mean [25 Μονάδες]

Για μια τυχαία μεταβλητή X που παίρνει τις τιμές $x_1, x_2, \dots, x_n$, με αντίστοιχες πιθανότητες $p_1, p_2, \dots, p_n$, η μέση τιμή $E[X]$ ορίζεται ως:

$$E[X] = \sum_{i=1}^{n} x_i \cdot p_i = p_1 \cdot x_1 + p_2 \cdot x_2 + \cdots + x_n \cdot p_n$$

Γράψτε ένα πρόγραμμα το οποίο παίρνει τις ακέραιες τιμές x_i μιας τυχαίας μεταβλητής X μαζί με τις πιθανότητες p_i σε δεκαδική μορφή ($0 \leq p_i \leq 1$) και υπολογίζει την μέση τιμή με ακρίβεια 8 δεκαδικών ψηφίων. Τα ορίσματα θα δίνονται ως εξής: το κάθε x_i θα ακολουθείται από το p_i που του αντιστοιχεί. Για παράδειγμα, αν $x_1 = 10$, $p_1 = 0.5$, $x_2 = 20$, $p_2 = 0.3$, $x_3 = 40$, $p_3 = 0.20$ ($E[X] = 10 \cdot 0.5 + 20 \cdot 0.3 + 40 \cdot 0.20 = 19.0$) το πρόγραμμα καλείται ως εξής:

```
$ ./mean 10 0.5 20 0.30 40 0.20
The mean of the random variable is: 19.00000000
```

Όλα τα ορίσματα δίνονται από την γραμμή εντολών και το αποτέλεσμα δεν είναι ανάγκη να είναι ακέραιο (ακολουθεί παράδειγμα).

```
./mean 42 0.222 5 0.0009999 100 0.777 100000 0.0000001
The mean of the random variable is: 87.03899950
```

Υπόδειξη Διατρέξτε το `argv` ανά ζεύγη (τιμή, πιθανότητα), μετατρέποντας το πρώτο σε ακέραιο και το δεύτερο σε `double` (π.χ. με `atoi`/`atof` ή `strtol`/`strtod`). Σκεφτείτε τι πρέπει να γίνει αν το πλήθος των ορισμάτων δεν είναι άρτιο ή είναι μηδέν, και χρησιμοποιήστε `%.8f` για την έξοδο.

A12.16 · Περιστροφή Πίνακα

A12.16 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #1 (Coreutils Themed), Θέμα 3* · ★★☆ · programming · `exam-2023-fall-ex1-q3`

Πρόγραμμα: `rotate.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που διαβάζει ένα πίνακα ακεραίων και τον περιστρέφει αντίστροφα από την φορά των δεικτών του ρολογιού. Τα δεδομένα περιέχονται σε αρχείο που δίνεται ως πρώτο όρισμα. Το αρχείο θα περιέχει τον αριθμό των γραμμών και τις στηλών του πίνακα και θα ακολουθούν τα στοιχεία του πίνακα. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ cat array.txt
3 5
2 9 8 3 2
2 8 1 0 8
7 1 2 4 3
$ gcc -o rotate rotate.c
$ ./rotate array.txt
2 8 3
3 0 4
8 1 2
9 8 1
2 2 7
```

Υπόδειξη Διαβάστε τις διαστάσεις με `fscanf` και δεσμεύστε δυναμικά χώρο για `R*C` ακεραίους. Ο περιστραμμένος πίνακας έχει `C` γραμμές και `R` στήλες: βρείτε, από το παράδειγμα, ποιο στοιχείο του αρχικού πίνακα καταλήγει στη γραμμή `i`, στήλη `j` της εξόδου, και τυπώστε με αυτή τη σειρά χωρίς να φτιάξετε δεύτερο πίνακα. Ελέγξτε αρνητικές ή μηδενικές διαστάσεις και αρχεία με λιγότερα στοιχεία από όσα δηλώνουν.

A12.17 · Κινήσεις σε Πλέγμα

A12.17 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #11, Θέμα 3* · ★★☆ · programming · `exam-2023-fall-ex11-q3`

Πρόγραμμα: `pacman.c`

Γράψτε ένα πρόγραμμα που παίρνει δύο ορίσματα: (1) το όνομα ενός αρχείου που περιέχει μια πίστα pacman και (2) τις κινήσεις που πρέπει να κάνει ο pacman πάνω στο πλέγμα και τυπώνει την τελική κατάσταση του παιχνιδιού μετά από αυτές τις κινήσεις. Το αρχείο θα έχει την ακόλουθη μορφή:

1. Διάσταση του πλέγματος του παιχνιδιού (δεκαδικός ακέραιος).
2. Τετράγωνο πλέγμα με χαρακτήρες '.' και ένα 'P' για την αρχική θέση του pacman. Όλοι οι άλλοι χαρακτήρες αγνοούνται. Οι κινήσεις που πρέπει να κάνει ο pacman θα αποτελούνται από 4 χαρακτήρες 'U' (Up), 'D' (Down), 'L' (Left), 'R' (Right). Όταν ο pacman κινείται σε κάποια θέση, τότε η τελεία ('.') σε εκείνη την θέση εξαφανίζεται. Η πίστα είναι τόρος, δηλαδή αν είσαι στην τελευταία γραμμή της και κινηθείς προς τα κάτω (D) τότε μεταφέρεσαι στην πρώτη γραμμή. Αντίστοιχα για κινήσεις αριστερά-δεξιά. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o pacman pacman.c
$ cat level.txt
7
.......
.......
..P....
.......
.......
.......
.......
$ ./pacman level.txt RRD
.......
.......
..   ..
....P..
.......
.......
.......
$ ./pacman level.txt RRDDLL
.......
.......
..   ..
.... ..
..P  ..
.......
.......
$ ./pacman level.txt RRDDLLUU
.......
.......
..P  ..
```

```
.. . ..
..   ..
.......
.......
$ ./pacman level.txt RRDDLLUULLLL
.......
.......
     P
.. . ..
..   ..
.......
.......
```

Υπόδειξη Φορτώστε την πίστα σε δισδιάστατο πίνακα, βρείτε το `P` και εφαρμόστε τις κινήσεις μία-μία, αφήνοντας κενό όπου περνάει ο pacman. Η αναδίπλωση του τόρου γίνεται με υπόλοιπο διαίρεσης· προσέξτε τις αρνητικές τιμές (`(x - 1 + n) % n`). Άγνωστος χαρακτήρας κίνησης είναι σφάλμα.

A12.18 · Πολλαπλασιασμός Πινάκων

A12.18 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #8, Θέμα 3* · ★★☆ · programming · `exam-2023-fall-ex8-q3`

Πρόγραμμα: `mat_mult.c`

Γράψτε ένα πρόγραμμα που παίρνει ως ορίσματα δύο ονόματα αρχείων που περιέχουν πίνακες αριθμών κινητής υποδιαστολής και στην συνέχεια τυπώνει το γινόμενο των δύο πινάκων με ακρίβεια δύο δεκαδικών ψηφίων. Στην πρώτη γραμμή κάθε αρχείου περιέχονται οι διαστάσεις του πίνακα (γραμμές και στήλες). Σύμφωνα με τον ορισμό, το γινόμενο δύο πινάκων A (a_{ij}) και B (b_{jk}) είναι ένας πίνακας C όπου

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Παράδειγμα εκτέλεσης ακολουθεί:

```
$ gcc -o mat_mult mat_mult.c
$ cat mat1.txt
2 3
1.12 2.01 3.22
4.08 5.02 6.05
$ cat mat2.txt
3 2
7      7.99
```

```
 9.09 10.0
11.11 12.21
$ ./mat_mult mat1.txt mat2.txt
61.89 68.37
141.41 156.67
```

Υπόδειξη Διαβάστε πρώτα τις διαστάσεις και δεσμεύστε δυναμικά τον πίνακα με το σωστό μέγεθος· ελέγξτε ότι το αρχείο ανοίγει και ότι περιέχει όσα στοιχεία δηλώνουν οι διαστάσεις. Το γινόμενο ορίζεται μόνο όταν οι στήλες του πρώτου ισούνται με τις γραμμές του δεύτερου· αλλιώς είναι είσοδος εκτός προδιαγραφών.

A12.19 · Πολύτιμοι Πίνακες

A12.19 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #9, Θέμα 3* · ★★☆ · programming · `exam-2023-fall-ex9-q3`

Πρόγραμμα: `precious.c`

Ένας πίνακας ακεραίων λέγεται πολύτιμος όταν το άθροισμα των στοιχείων της κάθε γραμμής ισούται με το άθροισμα των στοιχείων της κάθε στήλης. Γράψτε ένα πρόγραμμα που δέχεται το όνομα ενός αρχείου που περιέχει έναν πίκανα ως πρώτο όρισμα και τυπώνει εάν ο πίνακας είναι πολύτιμος. Το αρχείο που περιέχει τον πίνακα ξεκινά με τις διαστάσεις του πίνακα και συνεχίζει με τους ακεραίους που αποτελούν τα στοιχεία του. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o precious precious.c
$ cat ring.txt
3 3
8 1 6
3 5 7
4 9 2
$ ./precious ring.txt
My precious!
$ cat not-ring.txt
3 4
6 1 6 2
7 5 1 2
4 9 8 9
$ ./precious not-ring.txt
This matrix is NOT precious.
```

Για παράδειγμα ο πρώτος πίνακας είναι πολύτιμος καθώς κάθε γραμμή και κάθε στήλη αθροίζει στον ίδιο ακέραιο (το 15).

Υπόδειξη Διαβάστε πρώτα τις διαστάσεις και δεσμεύστε δυναμικά τον πίνακα με το σωστό μέγεθος· ελέγξτε ότι το αρχείο ανοίγει και ότι περιέχει όσα στοιχεία δηλώνουν οι διαστάσεις. Υπολογίστε το άθροισμα μιας γραμμής ως αναφορά και συγκρίνετε με αυτό όλα τα αθροίσματα γραμμών και στηλών (ο πίνακας δεν χρειάζεται να είναι τετράγωνος).

A12.20 · Στατιστικές

A12.20 · *Εξέταση Ιουλίου 2024, Θέμα 2* · ★★☆ · programming · `exam-2024-jul-q2`

Γράψτε ένα πρόγραμμα το οποίο δέχεται ακεραίους ως ορίσματα από την γραμμή εντολών και τυπώνει στην πρότυπη έξοδο (stdout) τον μέσο όρο τους και την τυπική απόκλιση με 3 δεκαδικά ψηφία ακρίβεια. Υπενθυμίζεται ότι ο μέσος όρος μ για N όρους δίνεται από τον τύπο: $\mu = \frac{\sum x_i}{N}$ και η τυπική απόκλιση δίνεται από τον τύπο: $\sigma = \sqrt{\frac{\sum (x_i - \mu)^2}{N}}$. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ ./stats 8 4 12
Mean: 8.000, Standard deviation: 3.266
$ ./stats 8 4 12 23 2 3 42 432 2 22 3 4 2 3 2 32 3 23
Mean: 34.556, Standard deviation: 97.112
```

Υπόδειξη Τα ορίσματα βρίσκονται στο `argv` ως συμβολοσειρές, οπότε χρειάζεστε μετατροπή σε ακέραιο, και το πλήθος τους είναι `argc - 1`. Η τυπική απόκλιση θέλει τον μέσο όρο ήδη υπολογισμένο, άρα κάντε δύο περάσματα πάνω στα ορίσματα. Κάντε τις πράξεις σε `double` (όχι ακέραια διαίρεση), θυμηθείτε το `-lm` για την `sqrt` και σκεφτείτε τι γίνεται όταν δεν δοθεί κανένα όρισμα.

A12.21 · Κινούμενος Μέσος Όρος - sma

A12.21 · *Εξέταση Ιανουαρίου 2025, Θέμα 3* · ★★☆ · programming · `exam-2025-jan-q3`

Κινούμενος Μέσος Όρος - sma [25 Μονάδες]

Ο κινούμενος μέσος όρος είναι παρεμφερής με τον κανονικό μέσο όρο, με μια διαφορά: ο κινούμενος μέσος όρος απαιτεί και ένα ”παράθυρο” (window), δηλαδή τον αριθμό των τιμών (μετρώντας από το τέλος) που θέλουμε να λάβουμε υπόψη μας στον υπολογισμό του μέσου όρου. Για παράδειγμα, ας υποθέσουμε ότι έχουμε αυτές τις 10 τιμές:

$$9 \quad 7 \quad 7 \quad 1 \quad 4 \quad 4 \quad 4 \quad 38 \quad 8 \quad 4$$

Ο μέσος όρος αυτών των τιμών είναι $\frac{9+7+7+1+4+4+4+38+8+4}{10} = 8.60$. Για να υπολογίσουμε τον κινούμενο μέσο, πρέπει να επιλέξουμε ένα παράθυρο, έστω 3. Τότε ο κινούμενος μέσος με παράθυρο 3 για αυτά τα στοιχεία λαμβάνει υπόψη του μόνο τα τελευταία 3:

$$9 \quad 7 \quad 7 \quad 1 \quad 4 \quad 4 \quad 4 \quad \underbrace{38 \quad 8 \quad 4}_{SMA_3}$$

και επομένως $SMA_3 = \frac{38+8+4}{3} = 16.67$. Αντίστοιχα, μπορούν να οριστούν κινούμενοι μέσοι με μικρότερα παράθυρα (π.χ., παράθυρο 1 σημαίνει μόνο το τελευταίο στοιχείο) ή μεταλύτερα παράθυρα (π.χ., παράθυρο 10 θα λάμβανε υπόψη του όλες τις τιμές παραπάνω). Γράψτε ένα πρόγραμμα που υπολογίζει τον κινούμενο μέσο όρο με παράθυρο 10 (SMA_{10}) και ακρίβεια 2 δεκαδικών ψηφίων για όλους τους ακεραίους που θα δοθούν ως ορίσματα στο πρόγραμμα από την γραμμή εντολών. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ ./sma 42 0 -4 9 7 7 1 4 4 4 38 8 4
The simple moving average (window 10) is: 8.60
```

Υπόδειξη Τα ορίσματα βρίσκονται στο `argv` ως συμβολοσειρές, οπότε χρειάζεστε μετατροπή σε ακέραιο (π.χ. `atoi`), και τα τελευταία 10 είναι στις τελευταίες θέσεις του `argv`: υπολογίστε από ποιον δείκτη ξεκινάτε με βάση το `argc`. Αποφασίστε τι κάνετε όταν δίνονται λιγότερα από 10 ορίσματα ή κανένα, και διαιρέστε σε `double` για να μη χάσετε τα δεκαδικά.

A12.22 · Το μεγαλύτερο άλμα - polevault

A12.22 · *Εξέταση Σεπτεμβρίου 2026, Θέμα 3* · ★★☆ · programming · `exam-2026-sep-q3`

Το μεγαλύτερο άλμα - polevault [25 Μονάδες]

Μας κάλεσαν από το Diamond League να γράψουμε ένα πρόγραμμα υπολογισμού της καλύτερης επίδοσης σε άλματα επί κοντώ. Το πρόγραμμα θα παίρνει τις επιδόσεις των αθλητών/τριών με την σειρά με τις οποίες τις πέτυχαν ως ορίσματα με τον εξής τρόπο: το όνομα του/της κάθε αθλητή/τριας θα ακολουθείται από την επίδοση, ενώ στην πρότυπη έξοδο περιμένουμε να δούμε το άτομο με την καλύτερη επίδοση. Σε περίπτωση ισοβαθμίας, το άτομο που πέτυχε την υψηλότερη επίδοση πρώτο παίρνει το χρυσό. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ ./polevault Kendricks 5.72 Guttormsen 5.82 Karalis 6.12 Marschall 5.82
Out of 4 athletes, the gold goes to Karalis with a jump to 6.12!
```

Υπόδειξη Τα ορίσματα έρχονται σε ζεύγη (όνομα, επίδοση) μετά το `argv[0]`, οπότε διατρέξτε το `argv` με βήμα 2 και μετατρέψτε την επίδοση σε αριθμό κινητής υποδιαστολής (π.χ. με `atof`). Αρκεί να κρατάτε τη θέση του καλύτερου μέχρι στιγμής· η ισοβαθμία λύνεται από το αν η σύγκριση είναι αυστηρή ή όχι. Σκεφτείτε τι γίνεται αν το πλήθος των ορισμάτων είναι μονό ή μηδέν, και πώς θα τυπώσετε την επίδοση με δύο δεκαδικά.

Κεφάλαιο 13: Μνήμη

Kahoot από το αμφιθέατρο (K13.1–K13.11)

K13.1 · Διπλό free

K13.1 · *Kahoot «Μνήμη» (διάλεξη 13)* · ★☆☆ · multiple-choice · 92% σωστές · `kahoot-double-free`

Για να είμαι βέβαιος ότι απελευθέρωσα τη μνήμη στον σωρό, τρέχω `free` δύο φορές χωρίς έλεγχο.

- True
- False

Υπόδειξη Τι συμβαίνει αν δώσετε στην `free` έναν δείκτη σε μνήμη που έχει ήδη αποδεσμευτεί;

K13.2 · Κατηγορίες μνήμης

K13.2 · *Kahoot «Δείκτες Παντού!» (διάλεξη 12) και «Μνήμη» (διάλεξη 13)* · ★☆☆ · multiple-choice · 87% σωστές · `kahoot-memory-categories`

Υπάρχουν μόνο δύο κατηγορίες μνήμης: η στοίβα και ο σωρός.

- True
- False

Υπόδειξη Πού αποθηκεύονται οι παγκόσμιες μεταβλητές και τα string literals;

K13.3 · Αφαίρεση από στοίβα

K13.3 · *Kahoot «Δείκτες Παντού!» (διάλεξη 12) και «Μνήμη» (διάλεξη 13)* · ★☆☆ · multiple-choice · 80% σωστές · `kahoot-stack-lifo`

Όταν αφαιρούμε ένα στοιχείο από μια στοίβα με πολλά στοιχεία, αυτό είναι:

- πάντα το τελευταίο που βάλαμε στην στοίβα
- πάντα το πρώτο που βάλαμε στην στοίβα
- οποιοδήποτε στοιχείο της στοίβας
- το 42ο στην σειρά

Υπόδειξη Σκεφτείτε μια στοίβα από πιάτα, ή τη σειρά με την οποία επιστρέφουν οι κλήσεις συναρτήσεων.

Κ13.4 · Είναι συνεχόμενος ο σωρός;

Κ13.4 · *Kahoot «Δείκτες Παντού!» (διάλεξη 12) και «Μνήμη» (διάλεξη 13)* · ★☆☆ · multiple-choice · 75% σωστές · `kahoot-heap-not-contiguous`

Τα στοιχεία του σωρού αποθηκεύονται σε μια συνεχόμενη περιοχή της μνήμης.

- True
- False

Υπόδειξη Σκεφτείτε τι γίνεται όταν δεσμεύουμε και αποδεσμεύουμε κομμάτια με τυχαία σειρά.

Κ13.5 · Πάντα πετυχαίνει η malloc;

Κ13.5 · *Kahoot «Δείκτες Παντού!» (διάλεξη 12) και «Μνήμη» (διάλεξη 13)* · ★☆☆ · multiple-choice · 70% σωστές · `kahoot-malloc-may-fail`

Έχω 16KB ελεύθερη μνήμη και τρέχω `malloc(8000)`: αυτή η κλήση σίγουρα θα πετύχει.

- True
- False

Υπόδειξη Η ελεύθερη μνήμη δεν είναι απαραίτητα ένα ενιαίο κομμάτι, και γι' αυτό ελέγχουμε πάντα την τιμή επιστροφής.

Κ13.6 · Αποδέσμευση από τον σωρό

Κ13.6 · *Kahoot «Μνήμη» (διάλεξη 13)* · ★★☆ · multiple-choice · 67% σωστές · `kahoot-heap-free-any`

Όταν αποδεσμεύουμε τη μνήμη ενός στοιχείου από έναν σωρό, αυτό το στοιχείο είναι:

- πάντα το τελευταίο που βάλαμε στον σωρό
- πάντα το πρώτο που βάλαμε στον σωρό
- οποιοδήποτε στοιχείο του σωρού
- το 42ο στην σειρά

Υπόδειξη Σε αντίθεση με τη στοίβα, ο σωρός δεν επιβάλλει σειρά· ποιος αποφασίζει πότε ελευθερώνεται κάτι;

K13.7 · Δέσμευση 10^9 double

K13.7 · *Kahoot «Δείκτες Παντού!» (διάλεξη 12) και «Μνήμη» (διάλεξη 13)* · ★★☆ · multiple-choice · 66% σωστές · `kahoot-malloc-doubles`

Θέλω να αποθηκεύσω 10^9 στοιχεία `double`. Τι κάνω;

- `int * array = malloc(1000000000);`
- `double array[1000000000];`
- `double * array = malloc(1000000000);`
- `double * array = malloc(1000000000 * sizeof(double));`

Υπόδειξη Η `malloc` μετράει σε bytes, όχι σε στοιχεία· και θυμηθείτε πόσο χωράει η στοίβα.

K13.8 · Πού βάζω έναν μεγάλο πίνακα

K13.8 · *Kahoot «Δείκτες Παντού!» (διάλεξη 12) και «Μνήμη» (διάλεξη 13)* · ★★☆ · multiple-choice · 65% σωστές · `kahoot-large-array-placement`

Έχω να δημιουργήσω έναν μεγάλο πίνακα (10^7 στοιχεία). Επιλέγω:

- Στοίβα
- Σωρό
- Στίβο
- Να μειώσω τον αριθμό των στοιχείων

Υπόδειξη Θυμηθείτε πόσο μεγάλη είναι συνήθως η στοίβα (`ulimit -s`) σε σχέση με τη μνήμη που μπορεί να δώσει ο σωρός.

K13.9 · malloc για πίνακα 10000 int

K13.9 · *Kahoot «Ταξινόμηση και Δομές»* · ★★☆ · multiple-choice · 60% σωστές · `kahoot-malloc-int-array`

Έστω ότι οι `int` θέλουν 4 bytes για την αναπαράστασή τους. Για να δεσμεύσω μνήμη για έναν πίνακα 10000 `int` κάνω:

- `int * array = malloc(10000);`
- `int * array = malloc(4);`
- `int * array = malloc(40000);`
- Εξαρτάται

Υπόδειξη Σε τι μονάδες μετράει το όρισμα της `malloc`; (Στον δικό σας κώδικα, γράψτε το με `sizeof`.)

K13.10 · malloc, calloc, realloc και free

K13.10 · *Kahoot «Μνήμη» (διάλεξη 13)* · ★★★ · multiple-choice · 27% σωστές · `kahoot-memory-functions`

Επιλέξτε όλα όσα είναι σωστά.

- Αποδεσμεύω μνήμη στην στοίβα με την free
- Αλλάζω το μέγεθος δεσμευμένης μνήμης στον σωρό με την realloc
- Η calloc είναι σαν την malloc απλά μηδενίζει τα στοιχεία της μνήμης
- Η malloc μπορεί να σμικρύνει την μνήμη στην οποία δείχνει ένας δείκτης

Συχνή παρανόηση Περίπου οι μισοί (50%) συμπεριέλαβαν ότι αποδεσμεύουμε μνήμη της στοίβας με την `free`. Η `free` είναι μόνο για μνήμη του σωρού· οι τοπικές μεταβλητές αποδεσμεύονται αυτόματα όταν επιστρέφει η συνάρτηση.

Υπόδειξη Ποια μνήμη διαχειριζόμαστε εμείς με τις συναρτήσεις της `stdlib.h`, και ποια αποδεσμεύεται αυτόματα;

K13.11 · Βάθος αναδρομής μέχρι να γεμίσει η στοίβα

K13.11 · *Kahoot «Μνήμη» (διάλεξη 13)* · ★★★ · multiple-choice · 20% σωστές · `kahoot-stack-overflow-depth`

Έστω ότι `ulimit -s` δίνει 8192 και το activation record της `foo` είναι 1024 bytes. Πόσες αναδρομικές κλήσεις μέχρι να «σκάσει» το πρόγραμμα;

- ~8
- 42
- ~400
- ~8000

Συχνή παρανόηση Το 48% επέλεξε `~8`, θεωρώντας ότι το 8192 είναι bytes. Το `ulimit -s` μετράει σε KB, άρα η στοίβα είναι 8 MB.

Υπόδειξη Ελέγξτε σε ποια μονάδα μετράει το `ulimit -s` το μέγεθος της στοίβας.

Ζέσταμα: από τις διαφάνειες (A13.1–A13.7)

A13.1 · Τι γίνεται μετά την free;

A13.1 · *Διάλεξη 13, διαφάνειες 47–51* · ★☆☆ · short-answer · `slides-lec13-after-free`

Μάθαμε να δεσμεύουμε μνήμη με την `malloc`: μας λείπει κάτι;

```
int * array = malloc(4 * sizeof(int));
if (!array) {
  // fail gracefully
}
free(array);
```

1. Τι θα συμβεί στη μνήμη (στη στοίβα και στον σωρό) μετά την `free`;
2. Τι θα συμβεί αν προσπαθήσουμε να προσπελάσουμε μνήμη που έχουμε αποδεσμεύσει, π.χ. με `array[0] = 4;` μετά την `free`;
3. Τι θα συμβεί αν προσπαθήσουμε να αποδεσμεύσουμε μνήμη που έχουμε ήδη αποδεσμεύσει, δηλαδή με δεύτερο `free(array);`;

Υπόδειξη Ξεχωρίστε τον pointer `array` (που ζει στη στοίβα) από το μπλοκ στο οποίο δείχνει (στον σωρό): ποιο από τα δύο αλλάζει η `free`; Για τα 2 και 3, σκεφτείτε τι σημαίνει να χρησιμοποιούμε μνήμη που δεν μας ανήκει πια, στην καλύτερη και στη χειρότερη περίπτωση.

A13.2 · Γιατί η αναδρομή πρέπει να τελειώνει;

A13.2 · *Διάλεξη 13, διαφάνεια 27* · ★☆☆ · short-answer · `slides-lec13-infinite-recursion`

Είχαμε πει «η αναδρομή πρέπει να τελειώνει». Γιατί;

```
void recurse() {
  recurse();
}

int main() {
  recurse();
  return 0;
}
```

```
$ gcc -o rec rec.c
$ ./rec
Segmentation fault
```

Υπόδειξη Σκεφτείτε τι μπαίνει στη στοίβα σε κάθε κλήση της `recurse` και πότε αφαιρείται. Η στοίβα έχει πεπερασμένο μέγεθος (δείτε `ulimit -s`).

A13.3 · Τα stack frames της equalIgnoreCase

A13.3 · *Διάλεξη 13, διαφάνειες 20–26* · ★☆☆ · short-answer · `slides-lec13-stack-frames`

```
char tolower(char c) {
    if (c >= 'A' && c <= 'Z')
        return c + ('a' - 'A');
    return c;
}
int equalIgnoreCase(char char1, char char2) {
    char lowerChar1 = tolower(char1);
    char lowerChar2 = tolower(char2);
    return lowerChar1 == lowerChar2;
}
```

Η εκτέλεση βρίσκεται μέσα στην `equalIgnoreCase`, και η στοίβα περιέχει το stack frame της (`char1`, `char2`, `lowerChar1`, `lowerChar2` και προσωρινά δεδομένα του μεταγλωττιστή).

1. Τι θα συμβεί όταν κληθεί η συνάρτηση `tolower` την πρώτη φορά;
2. Τι θα συμβεί όταν εκτελεστεί η εντολή `return` (της `tolower`);
3. Τι θα συμβεί όταν εκτελεστεί η δεύτερη κλήση `tolower`;
4. Τι θα συμβεί όταν εκτελεστεί η `return` της `equalIgnoreCase`;

Υπόδειξη Θυμηθείτε ότι η στοίβα λειτουργεί με σειρά LIFO και ότι κάθε κλήση συνάρτησης παίρνει το δικό της activation record με τα ορίσματα, τις τοπικές μεταβλητές και προσωρινά δεδομένα. Σχεδιάστε τη στοίβα ως κουτάκια μετά από κάθε βήμα.

A13.4 · Δυναμικός δισδιάστατος πίνακας MxN

A13.4 · *Διάλεξη 13, διαφάνειες 56–58* · ★★☆ · programming · `slides-lec13-dynamic-2d`

Θέλω να δημιουργήσω έναν δισδιάστατο πίνακα ακεραίων `MxN` δυναμικά. Πώς; Θέλω μετά να αποδεσμεύσω τη μνήμη του. Πώς;

Γράψτε τον κώδικα και για τα δύο, ώστε μετά τη δέσμευση να μπορείτε να γράφετε `array[i][j]` όπως σε έναν στατικό πίνακα. Για `M = 5`, `N = 6` και τον πίνακα γεμάτο με 1, 2, 3, … γραμμή-γραμμή, το `array[1][3]` πρέπει να είναι 10 και το `array[2][2]` 15.

Υπόδειξη Ο `array` είναι `int **`: δεσμεύστε πρώτα έναν πίνακα από `M` pointers και μετά μία γραμμή των `N` ακεραίων για καθέναν, ελέγχοντας κάθε κλήση για `NULL`. Για την αποδέσμευση σκεφτείτε τη σειρά: τι χάνετε αν αποδεσμεύσετε πρώτα τον πίνακα των pointers;

A13.5 · Ένας τεράστιος πίνακας στον σωρό

A13.5 · *Διάλεξη 13, διαφάνειες 43–46* · ★★☆ · trace · `slides-lec13-heap-bomb`

Τι θα κάνει το ακόλουθο πρόγραμμα;

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
  char *bomb = malloc(sizeof(char) * 9000000);
  printf("Hello World\n");
  return 0;
}
```

Τι θα κάνει αν η `malloc` ζητήσει `sizeof(char) * 900000000000L` bytes; Και τι αν μετά από αυτή την κλήση προσθέσουμε τη γραμμή `bomb[0] = 'A';` πριν το `printf`; Πώς πρέπει να διορθωθεί το πρόγραμμα;

Υπόδειξη Συγκρίνετε κάθε μέγεθος με τη μνήμη του υπολογιστή σας. Τι επιστρέφει η `malloc` όταν δεν υπάρχει αρκετή μνήμη, και τι γίνεται όταν γράφουμε σε αυτή τη διεύθυνση;

A13.6 · Κουίζ: πίνακας 5x5 σε συνάρτηση

A13.6 · *Διάλεξη 13, διαφάνειες 61–63* · ★★☆ · debug · `slides-lec13-quiz-5x5`

Θέλω να δημιουργήσω έναν πίνακα 5x5. Για καθεμία από τις τρεις υλοποιήσεις, είναι σωστή; Αν όχι, τι πρόβλημα έχει;

Υλοποίηση 1

```c
void bar() {
  int ** array = malloc(5 * sizeof(int*));
  int i;
  for(i = 0 ; i < 5 ; i++)
    array[i] = malloc(5 * sizeof(int));
  // process loop here
}
```

Υλοποίηση 2

```c
void bar() {
  int ** array = malloc(5 * sizeof(int*));
  if (!array) {
    return;
  }
  int i;
```

```
  for(i = 0 ; i < 5 ; i++) {
    array[i] = malloc(5 * sizeof(int));
    if (!array[i]) {
      return;
    }
  }
  // process loop here
}
```

Υλοποίηση 3

```
void bar() {
  int ** array = malloc(5 * sizeof(int*));
  if (!array) {
    return;
  }
  int i;
  for(i = 0 ; i < 5 ; i++) {
    array[i] = malloc(5 * sizeof(int));
    if (!array[i]) {
      return;
    }
  }
  // process loop here
  for(i = 0 ; i < 5 ; i++) {
    free(array[i]);
  }
  free(array);
}
```

Υπόδειξη Για κάθε `malloc` ρωτήστε: ελέγχεται για `NULL`; Αποδεσμεύεται με `free` σε **κάθε** δρόμο εξόδου από τη συνάρτηση, και στον κανονικό και σε αυτόν μιας αποτυχίας στη μέση του βρόχου;

A13.7 · Ένας τεράστιος πίνακας στη στοίβα

A13.7 · *Διάλεξη 13, διαφάνειες 28–31* · ★★☆ · trace · `slides-lec13-stack-bomb`

Τι θα κάνει το ακόλουθο πρόγραμμα;

```
#include <stdio.h>

int main() {
  char bomb[9000000];
```

```
  printf("Hello World\n");
  return 0;
}
```

```
$ ulimit -s
8192
```

Τι αλλάζει αν πρώτα εκτελέσουμε `ulimit -s unlimited`; Και τι κάνουμε όταν η στοίβα δεν είναι αρκετή και χρειάζεται να χρησιμοποιήσουμε πίνακες μεγάλου μεγέθους;

Υπόδειξη Η τιμή του `ulimit -s` είναι σε KB. Συγκρίνετε τη με το μέγεθος του τοπικού πίνακα και θυμηθείτε πού αποθηκεύονται οι τοπικές μεταβλητές. Για το τελευταίο ερώτημα, σκεφτείτε ποια άλλη κατηγορία μνήμης μπορεί να δεσμεύσει όλη τη διαθέσιμη μνήμη.

Εργαστήριο (A13.8–A13.9)

A13.8 · Δυναμική δέσμευση μνήμης για δισδιάστατο πίνακα

A13.8 · *Εργαστήριο 7, Άσκηση 3* · ★★☆ · programming · `lab-lab07-mines`

3.1 Κατασκευάστε το πρόγραμμα `mines.c` που να διαβάζει από την είσοδο τις διαστάσεις ενός δισδιάστατου πίνακα χαρακτήρων (έστω `N x M`), να δεσμεύει χώρο `N x M` θέσεων δυναμικά και έπειτα να τον αρχικοποιεί διαβάζοντας από την είσοδο χαρακτήρες.

Στην συνέχεια παραθέτουμε τρόπους για να δεσμεύσουμε και να αποδεσμεύσουμε μνήμη δυναμικά για δισδιάστατους πίνακες. Αντίστοιχα μπορείτε να διαχειριστείτε πίνακες με περισσότερες διαστάσεις. Στην τελική εξέταση θα χρειαστεί να διαχειριστείτε δεδομένα δυναμικής φύσης (δηλαδή δεδομένα που πρέπει να χωρέσουν σε πίνακες των οποίων τα μεγέθη δεν γνωρίζουμε εκ των προτέρων) και επομένως είναι καλό να εξοικειωθείτε με την διαδικασία δέσμευσης/αποδέσμευσης μνήμης.

Δυναμική δέσμευση μνήμης για δισδιάστατο πίνακα διάστασης N x M:

```
    int i;
    TΔ **p;
    p = malloc(N * sizeof(TΔ *));
    if (p == NULL) {
        fprintf(stderr, "Failed to allocate rows\n");
        exit(1);
    }
    for (i = 0 ; i < N ; i++) {
        p[i] = malloc(M * sizeof(TΔ));
        if (p[i] == NULL) {
            fprintf(stderr, "Failed to allocate row %d\n", i);
            exit(1);
```

```
        }
    }
```

Αποδέσμευση μνήμης για δυναμικά δεσμευμένο πίνακα N x M:

```
    int i;
    for (i = 0 ; i < N ; i++)
        free(p[i]);
    free(p);
```

3.2 Κατασκευάστε το αρχείο `mines.txt` που αναπαριστά τα δεδομένα ενός ναρκοπεδίου. Συγκεκριμένα, στην πρώτη γραμμή του αρχείου υπάρχουν οι διαστάσεις του ναρκοπεδίου (`N` και `M`) και μετά ακολουθούν γραμμή-γραμμή τα περιεχόμενα των κελιών, που είναι ο χαρακτήρας `.` όταν δεν υπάρχει νάρκη και ο χαρακτήρας `*` όταν υπάρχει νάρκη.

```
3 4
..*.
.**.
*.*.
```

3.3 Επεκτείνετε το πρόγραμμά σας, ώστε να εκτυπώνονται τα περιεχόμενα του πίνακα που διαβάστηκε. Εκτελέστε το πρόγραμμα σας με ανακατεύθυνση εισόδου από το αρχείο `mines.txt`.

3.4 Επεκτείνετε το πρόγραμμα σας, ώστε να εκτυπώνεται μια τροποποιημένη μορφή του ναρκοπεδίου, στην οποία, στα κελιά που υπάρχει νάρκη να εμφανίζεται πάλι το `*`, ενώ στα κελιά που δεν υπάρχει νάρκη να φαίνεται ένας αριθμός που δείχνει σε *πόσα γειτονικά κελιά* υπάρχει νάρκη. Σαν γειτονικά θεωρούνται όχι μόνο συνεχόμενα οριζόντια ή κατακόρυφα κελιά, αλλά και συνεχόμενα σε διαγώνια κατεύθυνση. Για παράδειγμα, για το ναρκοπέδιο που είδαμε, θα πρέπει να εμφανίζεται η έξοδος:

```
13*2
2**3
*4*2
```

Μπορείτε να δοκιμάσετε το πρόγραμμά σας και με μεγαλύτερα πλέγματα προκειμένου να επιβεβαιώσετε ότι λειτουργεί σωστά.

Υπόδειξη Κατά την ανάγνωση των κελιών προσέξτε τις αλλαγές γραμμής: αγνοήστε όποιον χαρακτήρα δεν είναι `.` ή `*`. Για κάθε κελί εξετάστε τα έως 8 γειτονικά του με δύο μικρούς βρόχους για μετατόπιση $-1, 0, +1$ σε γραμμή και στήλη, ελέγχοντας πάντα ότι ο γείτονας είναι μέσα στα όρια του πίνακα (οι γωνίες και οι άκρες έχουν λιγότερους γείτονες).

A13.9 · Κινήσεις σε πλέγμα (Παλιό θέμα)

A13.9 · *Εργαστήριο 7, Άσκηση 5* · ★★★ · programming · `lab-lab07-pacman`

Γράψτε ένα πρόγραμμα που παίρνει δύο εισόδους: (1) το περιεχόμενο μιας πίστας pacman από την πρότυπη είσοδο και (2) τις κινήσεις που πρέπει να κάνει ο pacman πάνω στο πλέγμα ως το πρώτο όρισμα και τυπώνει την τελική κατάσταση του παιχνιδιού μετά από αυτές τις κινήσεις. Η πίστα pacman θα έχει την ακόλουθη μορφή:

1. Διάσταση του πλέγματος του παιχνιδιού (δεκαδικός ακέραιος).
2. Τετράγωνο πλέγμα με χαρακτήρες '.' και ένα 'P' για την αρχική θέση του pacman. Όλοι οι άλλοι χαρακτήρες αγνοούνται.

Οι κινήσεις που πρέπει να κάνει ο pacman θα αποτελούνται από τους 4 χαρακτήρες 'U' (Up), 'D' (Down), 'L' (Left), 'R' (Right). Όταν ο pacman κινείται σε κάποια θέση, τότε η τελεία ('.') σε εκείνη την θέση εξαφανίζεται. Η πίστα είναι τόρος, δηλαδή αν είσαι στην τελευταία γραμμή της και κινηθείς προς τα κάτω (D) τότε μεταφέρεσαι στην πρώτη γραμμή. Αντίστοιχα για κινήσεις αριστερά-δεξιά. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o pacman pacman.c
$ cat level.txt
7
.......
.......
..P....
.......
.......
.......
.......
$ ./pacman RRD < level.txt
.......
.......
..   ..
....P..
.......
.......
.......
$ ./pacman RRDDLL < level.txt
.......
.......
..   ..
.... ..
..P  ..
.......
.......
$ ./pacman RRDDLLUU < level.txt
.......
.......
```

```
..P  ..
.. . ..
..   ..
.......
.......
$ ./pacman RRDDLLUULLLL < level.txt
.......
.......
     P
.. . ..
..   ..
.......
.......
```

Υπόδειξη Κρατήστε την πίστα σε δισδιάστατο πίνακα χαρακτήρων δεσμευμένο δυναμικά (η διάσταση διαβάζεται πρώτη) και διαβάστε τα κελιά αγνοώντας κάθε χαρακτήρα εκτός από `.` και `P`. Διατρέξτε το `argv[1]` χαρακτήρα-χαρακτήρα: σε κάθε κίνηση αδειάστε το τρέχον κελί και μετακινήστε τη θέση. Για την αναδίπλωση του τόρου χρησιμοποιήστε υπόλοιπο διαίρεσης, προσέχοντας ότι στη C το υπόλοιπο αρνητικού αριθμού είναι αρνητικό.

Θέματα εξετάσεων (A13.10–A13.11)

A13.10 · Debugging

A13.10 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #1 (Coreutils Themed), Θέμα 4* · ★★☆ · debug · `exam-2023-fall-ex1-q4`

Πρόγραμμα: `broken.c` (25 μονάδες)

Το πρόγραμμα broken.c δεν λειτουργεί για κάποιον λόγο - όταν το τρέχουμε κρασάρει. Βρείτε όποιο σφάλμα υπάρχει και διορθώστε το χωρίς να εισάγετε καινούρια σφάλματα στο πρόγραμμα. Παράδειγμα επιτυχούς εκτέλεσης ακολουθεί:

```
$ gcc -o broken broken.c
$ ./broken
yesss! -> yee -> bruh -> NULL
listdelete: 1
yesss! -> yee -> NULL
```

Το πρόγραμμα:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```c
typedef struct node {
    char value[128];
    struct node * next;
} Elem;

int listdelete(char value[], Elem ** head) {
    Elem * tmp;

    while (*head && strcmp((*head)->value, value)) {
        head = &((*head)->next);
    }

    if (*head) {
        tmp=*head;
        *head = tmp->next;
        free(tmp);
        return 1;
    }
    return -1;
}

int listadd(char value[], Elem ** head) {
    Elem * new = malloc(sizeof(Elem));
    if (new == NULL) {
        return 0;
    }
    Elem * tmp;
    tmp=*head;
    *head=new;
    new->next=tmp;
    strncpy(new->value, value, 127);
    return 1;
}

void listprint(Elem * list) {
  while(list) {
    printf("%s -> ", list->value);
    list = list->next;
  }
  printf("NULL\n");
}
```

```c
int main() {

    Elem elem1 = {"bruh", NULL};
    Elem * head = &elem1;

    listadd("yee", &head);
    listadd("yesss!", &head);
    listprint(head);
    printf("listdelete: %d\n", listdelete("bruh", &head));
    listprint(head);
    return 0;
}
```

Υπόδειξη Εκτελέστε το με `valgrind` ή `gcc -fsanitize=address` και δείτε σε ποια γραμμή σκάει. Αναρωτηθείτε για κάθε κόμβο της λίστας πού ζει στη μνήμη (στοίβα ή σωρός) και ποιοι κόμβοι επιτρέπεται να δοθούν στην `free`. Η διόρθωση πρέπει να κρατήσει τη λίστα συνεπή και να μην αφήνει διαρροές.

A13.11 · Κάδρο

A13.11 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #9, Θέμα 4* · ★★★ · programming · `exam-2023-fall-ex9-q4`

Πρόγραμμα: `frame.c`

Γράψτε ένα πρόγραμμα που διαβάζει από την πρότυπη είσοδο (stdin) ένα κείμενο λατινικών χαρακτήρων και το τυπώνει στην πρότυπη έξοδο (stdout) εντός ενός τετράγωνου κάδρου. Το κάδρο αποτελείται από τον χαρακτήρα '*' (42 στο δεκαδικό). Το πρόγραμμά σας πρέπει να μπορεί να χειριστεί οποιονδήποτε αριθμό γραμμών κάθε μία από τις οποίες μπορεί να είναι οποιουδήποτε μεγέθους. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ cat message.txt
    One Ring to
   Rule them all
    One Ring to
     find them
    One Ring to
   Bring them all
and in the darkness
    Bind them
$ gcc -o frame frame.c
$ ./frame < message.txt
**********************
```

```
*    One Ring to    *
*   Rule them all   *
*    One Ring to    *
*     find them     *
*    One Ring to    *
*   Bring them all  *
*and in the darkness*
*    Bind them      *
**********************
```

Υπόδειξη Το πλάτος του κάδρου εξαρτάται από τη μεγαλύτερη γραμμή, άρα πρέπει να διαβάσετε όλη την είσοδο πριν τυπώσετε οτιδήποτε. Δεν ξέρετε εκ των προτέρων πόσες γραμμές ή πόσο μεγάλη θα είναι η καθεμία, οπότε χρειάζεστε δυναμικούς πίνακες που μεγαλώνουν με `realloc`. Κάθε γραμμή συμπληρώνεται με κενά ως το μέγιστο μήκος.

Κεφάλαιο 14: Εμβέλεια, Μνήμη και Συμβολοσειρές

Kahoot από το αμφιθέατρο (K14.1–K14.8)

K14.1 · Πού δηλώνονται οι παγκόσμιες μεταβλητές

K14.1 · *Kahoot «Εμβέλεια, Μνήμη και Συμβολοσειρές» (διάλεξη 14)* · ★☆☆ · multiple-choice · 93% σωστές · `kahoot-global-declaration`

Οι μεταβλητές παγκόσμιας εμβέλειας (global scope) δηλώνονται:

- Μέσα στην συνάρτηση main
- Έξω από οποιαδήποτε συνάρτηση
- Μέσα σε οποιαδήποτε συνάρτηση
- Δεν υπάρχουν μεταβλητές παγκόσμιας εμβέλειας

Υπόδειξη Η εμβέλεια καθορίζεται από το σημείο της δήλωσης· για να είναι ορατή από κάθε συνάρτηση, πού πρέπει να βρίσκεται η δήλωση;

K14.2 · Δύο είδη εμβέλειας

K14.2 · *Kahoot «Δυαδική Αναζήτηση, Ταξινόμηση, Πολυπλοκότητα και άλλα» και «Εμβέλεια, Μνήμη και Συμβολοσειρές» (διάλεξη 14)* · ★★☆ · multiple-choice · 63% σωστές · `kahoot-two-kinds-of-scope`

Στη C υπάρχουν μόνο δύο είδη εμβέλειας για μεταβλητές: τοπικές και παγκόσμιες.

- True
- False

Συχνή παρανόηση Το 31% απάντησε False, πιθανόν θεωρώντας τις `static` μεταβλητές τρίτο είδος εμβέλειας· η `static` αλλάζει τον χρόνο ζωής και τη μνήμη, όχι το από πού είναι ορατή μια τοπική μεταβλητή.

Υπόδειξη Η εμβέλεια εξαρτάται από το σημείο της δήλωσης· ξεχωρίστε την από τον χρόνο ζωής και από τη μνήμη όπου αποθηκεύεται μια μεταβλητή.

K14.3 · Διαδοχικές strcat

K14.3 · *Kahoot «Εμβέλεια, Μνήμη και Συμβολοσειρές» (διάλεξη 14)* · ★★☆ · multiple-choice · 60% σωστές · `kahoot-strcat-sequence`

Τι θα τυπώσει ο παρακάτω κώδικας;

```
char str[100];
strcpy(str, "Hi");
strcat(str, "How");
strcat(str, " are you doing");
printf("%s", str);
```

- Hi
- How are you doing
- HiHow are you doing
- Θα κρασάρει

Υπόδειξη Θυμηθείτε ποια συνάρτηση αντικαθιστά το περιεχόμενο του προορισμού και ποια προσθέτει στο τέλος του, και προσέξτε πού υπάρχουν κενά.

K14.4 · Πού αποθηκεύεται το char str[]

K14.4 · *Kahoot «Δυαδική Αναζήτηση, Ταξινόμηση, Πολυπλοκότητα και άλλα» και «Εμβέλεια, Μνήμη και Συμβολοσειρές» (διάλεξη 14)* · ★★☆ · multiple-choice · 49% σωστές · `kahoot-char-array-storage`

Παλιό θέμα: Η εντολή `char str[] = "hello";` αποθηκεύει τον πίνακα `str`:

- Στην στοίβα
- Στον σωρό
- Στην στατική μνήμη

- Στην στοίβα ή στατική μνήμη ανάλογα με το σημείο δήλωσης

Συχνή παρανόηση Το 31% απάντησε «Στην στοίβα», υποθέτοντας σιωπηρά ότι η δήλωση είναι μέσα σε συνάρτηση· αν είναι παγκόσμια, ο πίνακας βρίσκεται στη στατική μνήμη.

Υπόδειξη Η εντολή δεν λέει πού βρίσκεται: μέσα σε συνάρτηση ή έξω από κάθε συνάρτηση; Θυμηθείτε πού ζει μια τοπική και πού μια παγκόσμια μεταβλητή.

K14.5 · Η τιμή της strcmp για ίσες συμβολοσειρές

K14.5 · *Kahoot «Δυαδική Αναζήτηση, Ταξινόμηση, Πολυπλοκότητα και άλλα» και «Εμβέλεια, Μνήμη και Συμβολοσειρές» (διάλεξη 14)* · ★★☆ · multiple-choice · 48% σωστές · `kahoot-strcmp-equal`

Ποια η τιμή της έκφρασης `strcmp("Hodor", "Hodor");`

- 0
- 1
- 42
- Hodor

Συχνή παρανόηση Το 30% επέλεξε 1, διαβάζοντας την `strcmp` σαν έλεγχο ισότητας που επιστρέφει «αληθές» όταν οι συμβολοσειρές είναι ίδιες.

Υπόδειξη Η `strcmp` δεν απαντά «ναι/όχι» αλλά συγκρίνει λεξικογραφικά: το πρόσημο του αποτελέσματος λέει ποια συμβολοσειρά προηγείται.

K14.6 · strcpy από literal σε πίνακα

K14.6 · *Kahoot «Εμβέλεια, Μνήμη και Συμβολοσειρές» (διάλεξη 14)* · ★★★ · multiple-choice · 32% σωστές · `kahoot-strcpy-literal-to-array`

Τι θα τυπώσει ο παρακάτω κώδικας;

```
char *s1 = "World", s2[] = "Hello";
strcpy(s2, s1);
printf("%s\n", s2);
```

- Hello
- World
- HelloWorld
- Θα κρασάρει

Συχνή παρανόηση Το 24% απάντησε ότι θα κρασάρει, μπερδεύοντας τους ρόλους: η `strcpy` γράφει στον πίνακα `s2`, που είναι εγγράψιμος και χωράει τα 6 bytes του `"World"`· το literal `s1` μόνο διαβάζεται.

Υπόδειξη Ποιος από τους δύο είναι πίνακας που μπορούμε να γράψουμε και ποιος δείχνει σε string literal; Σε ποιον γράφει η `strcpy`, και χωράει εκεί το αποτέλεσμα;

K14.7 · Η strcat με τον εαυτό της

K14.7 · *Kahoot «Δυαδική Αναζήτηση, Ταξινόμηση, Πολυπλοκότητα και άλλα» και «Εμβέλεια, Μνήμη και Συμβολοσειρές» (διάλεξη 14)* · ★★★ · multiple-choice · 24% σωστές · `kahoot-strcat-self-overflow`

Τι θα τυπώσει ο παρακάτω κώδικας;

```
char fruit[10];
strcpy(fruit, "banana");
strcat(fruit, fruit);
strcat(fruit, fruit);
printf("%s", fruit);
```

- banana
- bananabanana
- bananabananabananabanana
- Πιθανόν να κρασάρει

Συχνή παρανόηση Το 34% επέλεξε bananabananabananabanana και το 24% bananabanana, κοιτάζοντας μόνο τη λογική της συνένωσης: ο πίνακας έχει 10 bytes, οπότε ήδη η πρώτη `strcat` γράφει εκτός ορίων, και η επικάλυψη πηγής και προορισμού είναι από μόνη της απροσδιόριστη συμπεριφορά.

Υπόδειξη Μετρήστε πόσα bytes χρειάζεται το αποτέλεσμα, μαζί με το `'\0'`, και συγκρίνετε με το μέγεθος του `fruit`. Σκεφτείτε επίσης τι σημαίνει για την `strcat` να διαβάζει από τον ίδιο πίνακα στον οποίο γράφει.

K14.8 · Πού αποθηκεύονται οι τοπικές μεταβλητές

K14.8 · *Kahoot «Εμβέλεια, Μνήμη και Συμβολοσειρές» (διάλεξη 14)* · ★★★ · multiple-choice · 23% σωστές · `kahoot-local-variable-storage`

Πού αποθηκεύονται οι τοπικές μεταβλητές μιας συνάρτησης;

- Στην στοίβα
- Στον σωρό
- Στην στατική μνήμη
- Στην στοίβα ή στην στατική μνήμη ανάλογα με το αν δηλώθηκε static

Συχνή παρανόηση Το 55% απάντησε «Στην στοίβα», ξεχνώντας τις τοπικές `static` μεταβλητές, που έχουν τοπική εμβέλεια αλλά ζουν στη στατική μνήμη για όλη την εκτέλεση.

Υπόδειξη Μια τοπική μεταβλητή μπορεί να κρατά την τιμή της από κλήση σε κλήση· σκεφτείτε ποια λέξη-κλειδί το κάνει αυτό και πού πρέπει να ζει μια τέτοια μεταβλητή.

Ζέσταμα: από τις διαφάνειες (A14.1–A14.8)

A14.1 · Είναι το πρώτο όρισμα ”--boo”;

A14.1 · *Διάλεξη 14, διαφάνεια 42* · ★☆☆ · programming · `slides-lec14-check-boo`

Θέλω να ελέγξω αν το πρώτο όρισμα του προγράμματός μου είναι `"--boo"`. Πώς;

Υπόδειξη Το πρώτο όρισμα είναι το `argv[1]`, αλλά υπάρχει μόνο αν το `argc` είναι αρκετά μεγάλο. Ο τελεστής `==` σε δύο string συγκρίνει διευθύνσεις, όχι περιεχόμενα· χρειάζεστε μια συνάρτηση της `string.h` και να θυμάστε τι επιστρέφει όταν τα string είναι ίσα.

A14.2 · Αύξηση παραμέτρου μέσα σε συνάρτηση

A14.2 · *Διάλεξη 14, διαφάνειες 11–12* · ★☆☆ · trace · `slides-lec14-local-param`

Τι θα επιστρέψει αυτό το πρόγραμμα;

```c
void foo(int i) {
  i++;
}
int main() {
  int i = 42;
  foo(i);
  return i;
}
```

```
$ gcc -o local2 local2.c
$ ./local2
$ echo $?
```

Υπόδειξη Η παράμετρος μιας συνάρτησης είναι κι αυτή τοπική μεταβλητή, που αρχικοποιείται κατά την κλήση με την τιμή του ορίσματος. Σκεφτείτε ποια μεταβλητή αυξάνει το `i++` και ποια επιστρέφει η `main`.

A14.3 · Τοπική μεταβλητή με το ίδιο όνομα σε δύο συναρτήσεις

A14.3 · *Διάλεξη 14, διαφάνειες 9–10* · ★☆☆ · trace · `slides-lec14-local-same-name`

Η τοπική μεταβλητή μιας συνάρτησης δεν είναι ορατή σε κάποια άλλη συνάρτηση και επομένως μπορούμε να δηλώσουμε μεταβλητές με το ίδιο όνομα σε άλλες συναρτήσεις και να αναφέρονται σε άλλες θέσεις μνήμης.

Τι θα επιστρέψει αυτό το πρόγραμμα;

```
void foo() {
  int i = 43;
}
int main() {
  int i = 42;
  foo();
  return i;
}
```

```
$ gcc -o local local.c
$ ./local
$ echo $?
```

Υπόδειξη Αναρωτηθείτε αν το `i` της `foo` και το `i` της `main` είναι η ίδια μεταβλητή: πού είναι η εμβέλεια του καθενός και σε ποιο frame της στοίβας ζει; Θυμηθείτε ότι το `echo $?` τυπώνει την τιμή που επέστρεψε η `main`.

A14.4 · Παλινδρομικό string

A14.4 · *Διάλεξη 14, διαφάνειες 43–44* · ★☆☆ · programming · `slides-lec14-palindrome`

Θέλω να ελέγξω αν ένα string είναι παλινδρομικό. Πώς;

Γράψτε μια συνάρτηση `int isPalindrome(char *str)` που επιστρέφει 1 αν το `str` διαβάζεται ίδιο από την αρχή και από το τέλος, και 0 αλλιώς.

Υπόδειξη Χρησιμοποιήστε δύο δείκτες θέσης που ξεκινούν από τις δύο άκρες του string και κινούνται ο ένας προς τον άλλο. Προσέξτε πού είναι ο τελευταίος χαρακτήρας: το `strlen` δεν μετρά το null byte.

A14.5 · Επισκίαση παγκόσμιας μεταβλητής

A14.5 · *Διάλεξη 14, διαφάνειες 14–16* · ★☆☆ · trace · `slides-lec14-shadowing`

Όταν μια δήλωση μεταβλητής βρίσκεται εντός εμβέλειας άλλης μεταβλητής με το ίδιο όνομα, τότε η εσωτερική μεταβλητή επισκιάζει (shadows) την άλλη. Τι θα τυπώσει;

```c
#include <stdio.h>
int baz = 42;
int main() {
  int baz = 43;
  printf("baz: %d\n", baz);
  return 0;
}
```

Υπόδειξη Βρείτε την εμβέλεια της παγκόσμιας `baz` και της τοπικής `baz` της `main`. Στο σημείο της `printf` είναι ορατές και οι δύο· ποια δήλωση είναι η πιο «εσωτερική»;

A14.6 · Στατική και τοπική μεταβλητή σε τρεις κλήσεις

A14.6 · *Διάλεξη 14, διαφάνειες 17–18* · ★☆☆ · trace · `slides-lec14-static-counter`

Μια μεταβλητή λέγεται στατική (static), όταν διατηρεί την τιμή της ανάμεσα σε κλήσεις συναρτήσεων μέχρι το τέλος του προγράμματος. Οι στατικές μεταβλητές αρχικοποιούνται μόνο στην πρώτη κλήση της συνάρτησης. Τι θα τυπώσει;

```c
#include <stdio.h>
void foo() {
  static int counter = 0; int i = 0;
  counter++; i++;
  printf("%d %d\n", counter, i);
}
int main() {
  foo(); foo(); foo();
  return 0;
}
```

Υπόδειξη Κρατήστε δύο στήλες, μία για το `counter` και μία για το `i`, και ενημερώστε τις σε κάθε κλήση. Ποια από τις δύο αρχικοποιήσεις εκτελείται ξανά σε κάθε κλήση και ποια μόνο μία φορά;

A14.7 · Στατική μεταβλητή με ανάθεση εκτός δήλωσης

A14.7 · *Διάλεξη 14, διαφάνεια 19* · ★★☆ · trace · `slides-lec14-static-assign`

Τι θα τυπώσει το πρόγραμμα, όπου η `counter` είναι στατική αλλά παίρνει την τιμή 0 με ξεχωριστή εντολή;

```c
#include <stdio.h>
void foo() {
  static int counter; int i = 0; counter = 0;
  counter++; i++;
  printf("%d %d\n", counter, i);
}
int main() {
  foo(); foo(); foo();
  return 0;
}
```

Υπόδειξη Συγκρίνετε με την εκδοχή όπου το `= 0` είναι μέρος της δήλωσης `static int counter = 0;`. Μια αρχικοποίηση στη δήλωση μιας στατικής μεταβλητής γίνεται μία φορά· μια εντολή ανάθεσης όμως εκτελείται κάθε φορά που τη φτάνει η ροή.

A14.8 · Η έκφραση *str++

A14.8 · *Διάλεξη 14, διαφάνεια 34* · ★★☆ · short-answer · `slides-lec14-str-plus-plus`

Η υλοποίηση της `strlen` στις διαφάνειες είναι:

```c
size_t strlen(char * str) {
  size_t length = 0;
  while(*str++) length++;
  return length;
}
```

Πώς αποτιμάται η έκφραση `*str++`;

- Έχει διαφορά από την έκφραση `(*str)++`;
- Έχει διαφορά από την έκφραση `*(str++)`;
- Τελικά έχει σημασία η προτεραιότητα τελεστών / χρήση παρενθέσεων;

Υπόδειξη Βρείτε στον πίνακα προτεραιότητας ποιος τελεστής δένει πιο σφιχτά, ο μεταθεματικός `++` ή το μοναδιαίο `*`. Για κάθε έκφραση απαντήστε σε δύο ερωτήσεις: ποια τιμή δίνει, και τι αλλάζει στη μνήμη, ο δείκτης ή ο χαρακτήρας όπου δείχνει;

Εργαστήριο (A14.9)

A14.9 · Επεξεργασία συμβολοσειρών

A14.9 · *Εργαστήριο 8, Άσκηση 1* · ★★☆ · programming · `lab-lab08-string`

1.1 Κατασκευάστε το πρόγραμμα `string.c`, το οποίο θα συμπεριλαμβάνει το αρχείο επικεφαλίδας `string.h` και θα πραγματοποιεί το σενάριο που ακολουθεί.

1.2 Ορίστε τις συμβολοσειρές `strA` και `strB` με στατική ή δυναμική δέσμευση μνήμης 80 χαρακτήρων.

1.3 Αντιγράψτε στην `strA` τη συμβολοσειρά `"This  is  a  string."` και στην `strB` τη συμβολοσειρά `"This is another string."` χρησιμοποιώντας την συνάρτηση `strcpy`.

```
char *strcpy(char *s1, const char *s2)
```

Η συνάρτηση `strcpy` αντιγράφει τη συμβολοσειρά `s2` στην `s1` και επιστρέφει την τιμή του δείκτη (char *) στον οποίο έκανε την αντιγραφή (σε αυτό το παράδειγμα επιστρέφει την τιμή της μεταβλητής `s1`). Χρησιμοποιώντας την `printf("%p %p %p\n", ...)` τυπώστε τους τρεις δείκτες που συμμετέχουν στην κλήση αυτής της συνάρτησης (`s1`, `s2` και την τιμή επιστροφής της `strcpy`).

1.4 Υλοποιήστε τη συνάρτηση `int mystrlen(char *str)` η οποία δέχεται σαν όρισμα μία συμβολοσειρά και επιστρέφει το μήκος της (χωρίς να συμπεριλαμβάνεται ο χαρακτήρας τέλους συμβολοσειράς `'\0'`).

1.5 Υλοποιήστε τη συνάρτηση `char *mystrcat(char *s1, char *s2)` η οποία προσαρτά ένα αντίγραφο της συμβολοσειράς `s2` στο τέλος της `s1` και επιστρέφει στο όνομά της έναν δείκτη στην `s1`.

1.6 Εκτυπώστε τις δύο συμβολοσειρές και το μήκος τους. Υπολογίστε το μήκος της `strA` μέσω της συνάρτησης `mystrlen` που υλοποιήσατε παραπάνω και το μήκος της `strB` μέσω της συνάρτησης `strlen` της C.

```
size_t strlen(const char *s)
```

Η συνάρτηση `strlen` επιστρέφει στο όνομά της το μήκος της συμβολοσειράς `s` (χωρίς να μετράται το τελικό `'\0'`).

1.7 Συγκρίνετε αλφαβητικά τις συμβολοσειρές `strA` και `strB` εκτυπώνοντας κατάλληλο μήνυμα.

```
int strcmp(const char *s1, const char *s2)
```

Η συνάρτηση `strcmp` συγκρίνει τις συμβολοσειρές `s1` και `s2` χαρακτήρα προς χαρακτήρα με βάση τους αντίστοιχους ASCII κωδικούς. Επιστρέφει:

1. `= 0`, αν οι συμβολοσειρές είναι ίδιες
2. `> 0`, αν η `s1` είναι λεξικογραφικά «μεγαλύτερη» της `s2`
3. `< 0`, αν η `s1` είναι λεξικογραφικά «μικρότερη» της `s2`

1.8 Προσαρτήστε τη συμβολοσειρά `strB` στο τέλος της `strA` (χρησιμοποιώντας τη συνάρτηση `mystrcat` που υλοποιήσατε παραπάνω) και εκτυπώστε το αποτέλεσμα της προσάρτησης. Στη συνέχεια, προσαρτήστε τη νέα τιμή της συμβολοσειράς `strA` στο τέλος της `strB` (χρησιμοποιώντας τη συνάρτηση `strcat` της C) και εκτυπώστε το αποτέλεσμα της προσάρτησης.

```
char *strcat(char *s1, const char *s2)
```

Η `strcat` προσαρτά / συνενώνει (concatenates) ένα αντίγραφο της συμβολοσειράς `s2` στο τέλος της `s1` και επιστρέφει στο όνομά της έναν δείκτη στην `s1`.

1.9 Χρησιμοποιήστε τη συνάρτηση `strtok` για να εκτυπώσετε μία προς μία τις λέξεις που εμφανίζονται στην τελική συμβολοσειρά `strB`, χωρίς τους χαρακτήρες στίξης.

```
char *strtok(char *string, const char *delim)
```

Αν το `string` δεν είναι `NULL`, η `strtok` ψάχνει στο string για την πρώτη εμφάνιση συμβολοσειράς που περιορίζεται από έναν από τους χαρακτήρες που εμφανίζονται στη συμβολοσειρά `delim`. Αν υπάρχει, αντικαθιστά τον χαρακτήρα που βρέθηκε στο string, με `'\0'` και επιστρέφει έναν δείκτη στην αρχή του `string`.

Σε κάθε επόμενη κλήση της, η `strtok` καλείται με `NULL` στο πρώτο όρισμα και συνεχίζει τη λειτουργία της από το σημείο που η τελευταία κλήση βρήκε τον χαρακτήρα διαχωρισμού. Ένα παράδειγμα χρήσης ακολουθεί με ένα quote:

```
char *p, s[] = "Little by little, one Little travels far.";
p = strtok(s, " ,.");
while(p != NULL) {
  printf("%s\n", p);
  p = strtok(NULL, " ,.");
}
```

Προσθέστε το παραπάνω σε μια συνάρτηση `strtok_example` και τρέξτε την από την `main`. Στο `stdout` θα πρέπει να δείτε μια ακολουθία της μορφής:

```
Little
by
little
one
Little
travels
far
```

Υπόδειξη Μια συμβολοσειρά τελειώνει στο `'\0'`: η `mystrlen` προχωρά μέχρι να το βρει, και η `mystrcat` βρίσκει πρώτα το τέλος της `s1` και από εκεί αντιγράφει την `s2` μαζί με το τελικό `'\0'`. Στο 1.8 λογαριάστε τα μήκη: χωράει το αποτέλεσμα των δύο συνενώσεων στους 80 χαρακτήρες της `strB`; Αν όχι, γράφετε έξω από τον πίνακα.

Εργασίες (A14.10)

A14.10 · Το Δικό σου Chatbot (jason)

A14.10 · *Εργασία 2 (2024-25), Άσκηση 3* · ★★★ · programming · `hw-2024-hw2-jason`

Το Δικό σου Chatbot - (jason - 50 Μονάδες)

Η τεχνητή νοημοσύνη έχει μπει για τα καλά στην ζωή μας, και χρήστες και προγραμματιστές/τριες βρίσκουν συνεχώς νέες εφαρμογές της. Πόσο δύσκολο είναι όμως να φτιάξουμε ένα εργαλείο που να μπορεί να αξιοποιήσει τις υπηρεσίες τεχνητής νοημοσύνης και να προσφέρει λύσεις σε χρήστες; Αυτή είναι η ερώτηση που θα μας απασχολήσει σε αυτήν την άσκηση.

Προκειμένου να μιλήσουμε με απομακρυσμένες υπηρεσίες, πρέπει να μπορούμε να μιλήσουμε την γλώσσα τους. Τυχαίνει σήμερα πάμπολλες εφαρμογές στο internet να μεταφέρουν δεδομένα χρησιμοποιώντας μια μορφοποίηση (data format) που λέγεται JSON (JavaScript Object Notation). Για να δούμε ένα παράδειγμα. Έστω ότι έχουμε ένα αρχείο `person.json` με τα ακόλουθα περιεχόμενα:

```
{
  "first_name": "John",
  "last_name": "Smith",
  "age": 27,
  "address": {
    "street_address": "21 2nd Street",
    "city": "New York",
    "state": "NY",
  },
  "phone_numbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [
    "Catherine",
    "Thomas",
    "Trevor"
  ],
```

```
  "spouse": null
}
```

Το παραπάνω παράδειγμα JSON παρουσιάζει ιεραρχικά τα δεδομένα για ένα άτομο. Παρατηρούμε ότι όλα τα δεδομένα είναι μέσα σε ένα object (περικλείονται από αγκύλες { και }) και μέσα σε αυτό έχει διάφορα πεδία που κλείνονται μέσα σε διπλά quotes ": first_name, last_name, ..., children, spouse. Πέρα από αυτό παρατηρούμε ότι κάθε πεδίο μπορεί να έχει ως τιμή (ότι ακολουθεί την άνω και κάτω τελεία ":") ένα από τα ακόλουθα:

1. έναν αριθμό (π.χ., `"age": 27`)
2. ένα string (`"first_name": "John"`)
3. μια λίστα από δεδομένα (`"children": ["Catherine", "Thomas", "Trevor"]`)
4. ένα object με δικά του πεδία (`"address": { ... }`)

Παρατηρήστε ότι ο παραπάνω ορισμός είναι αυτο-αναφορικός (ένα JSON object μπορεί να περιέχει ένα πεδίο τύπου JSON object, που μπορεί να περιέχει ένα JSON object κοκ).

Τι σχέση έχουν όλα αυτά με την τεχνητή νοημοσύνη; Προκειμένου να χρησιμοποιήσουμε τις υπηρεσίες τεχνητής νοημοσύνης πρέπει να μπορούμε να εξάγουμε κάποια συγκεκριμένα δεδομένα από ένα JSON. Για παράδειγμα, δείτε το παρακάτω παράδειγμα-απάντηση του gpt-4o-mini μοντέλου τεχνητής νοημοσύνης στην ερώτηση "Tell me a joke":

```
{
  "id": "chatcmpl-Ag5hb4g6PYiVpvBp3tuiJjqX9KjqN",
  "object": "chat.completion",
  "created": 1734595059,
  "model": "gpt-4o-mini-2024-07-18",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "Why do programmers always mix up Halloween
                    and Christmas?\nBecause Oct 31 == Dec 25!\n",
        "refusal": null
      },
      "logprobs": null,
      "finish_reason": "stop"
    }
  ],
  "usage": {
    "prompt_tokens": 10,
    "completion_tokens": 23,
    "total_tokens": 33,
    "prompt_tokens_details": {
```

```
      "cached_tokens": 0,
      "audio_tokens": 0
    },
    "completion_tokens_details": {
      "reasoning_tokens": 0,
      "audio_tokens": 0,
      "accepted_prediction_tokens": 0,
      "rejected_prediction_tokens": 0
    }
  },
  "system_fingerprint": "fp_6fc10e10eb"
}
```

Το παραπάνω JSON φαίνεται χαοτικό στην αρχή, αλλά αν το κοιτάξετε καλύτερα θα βρείτε το περιεχόμενο της απάντησης. Για να την ανιχνεύσουμε μαζί:

- Στο πάνω επίπεδο, έχουμε ένα JSON object με ένα πεδίο ”choices”.
  - Το πεδίο choices είναι τύπου λίστα (ξεκινά με ’[’) και το πρώτο στοιχείο (index = 0) είναι επίσης JSON object.
    * Το JSON object της λίστας choices έχει μέσα ένα πεδίο που λέγεται ”message” το οποίο είναι επίσης τύπου JSON object.
      · Τέλος, το message έχει μέσα ένα πεδίο το οποίο λέγεται ”content”, το οποίο περιέχει το string με την απάντησή μας μέσα σε double quotes: ”Why do programmers always mix up Halloween and Christmas?\n Because Oct 31 == Dec 25!\n” - παρατηρήστε ότι οι αλλαγές γραμμής είναι escaped όπως θα ήταν σε ένα string που θα γράφαμε σε πρόγραμμα σε γλώσσα C.

Ένας πιο γρήγορος τρόπος να συμβολίσουμε το παραπάνω πεδίο είναι ως εξής: `json.choices[0].message.con` - μετάφραση: στο JSON αρχείο που μας δίνεται, θέλουμε να βρούμε την λίστα choices, να πάρουμε το πρώτο στοιχείο της, από εκεί να πάρουμε το message και στο τέλος να πάρουμε την τιμή του content. Αν είχαμε την δυνατότητα να πάρουμε οποιοδήποτε αρχείο JSON και να εξάγουμε αυτήν την πληροφορία θα μπορούσαμε να φτιάξουμε το δικό μας chatbot!

Αυτό θα είναι και ο στόχος αυτής της άσκησης, θα υλοποιήσουμε ένα πρόγραμμα το οποίο θα εξάγει την παραπάνω πληροφορία από αρχεία JSON και θα την αξιοποιεί για να δώσει απαντήσεις στον χρήστη.

Τεχνικές Προδιαγραφές

- Repository Name: progintro/hw2-<YourUsername>
- C Filepath: jason/src/jason.c
- Το πρόγραμμά θα έχει δύο modes:
  - **Extraction mode**. Το πρόγραμμά σας μπαίνει σε extraction mode όταν ο χρήστης δώσει το option `--extract`. Το option `--extract` περιμένει στην συνέχεια το όνομα

του αρχείου που περιέχει τα περιεχόμενα JSON και από τα οποία θα πρέπει να εξάγετε το `json.choices[0].message.content` και να το τυπώσετε στην πρότυπη έξοδο. Εάν δωθεί οποιοδήποτε αρχείο που δεν είναι valid JSON, το πρόγραμμά σας πρέπει να τυπώσει μήνυμα ίδιο με τα παραδείγματα παρακάτω στο stderr.

  - **Conversation mode**. Το πρόγραμμά σας μπαίνει σε διαδικασία συζήτησης με τον χρήστη όταν δωθεί το option `--bot` χωρίς άλλα ορίσματα. Στην διαδικασία συζήτησης, το πρόγραμμά σας ρωτάει επαναλαμβανόμενα τον χρήστη `> What would you like to know?` μέχρι να στείλει ο χρήστης End-of-File (EOF). Κάθε ερώτηση του χρήστη τελειώνει με καινούρια γραμμή και πρέπει να στέλνεται αυτούσια στην κατάλληλη συνάρτηση της βιβλιοθήκης neurolib (δες παρακάτω).

- Για τις διαδράσεις με το σύστημα τεχνητής νοημοσύνης (conversation mode) είναι υποχρεωτικό να χρησιμοποιήσετε την βιβλιοθήκη neurolib (neurolib.c και neurolib.h) που σας δίνεται στον φάκελο jason/src. Δυστυχώς κατεβάσαμε αυτήν την βιβλιοθήκη από ένα online project χωρίς καθόλου documentation / testing. Προκειμένου να την αξιοποιήσετε θα χρειαστεί να διαβάσετε τις διαθέσιμες συναρτήσεις στο header file της και να ανακαλύψετε την χρήση τους. Ο στόχος σας είναι να στείλετε queries στην υπηρεσία τεχνητής νοημοσύνης και να πάρετε JSON απαντήσεις.

- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με τις ακόλουθες εντολές σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr) - για παράδειγμα το linux14:

  ```
  gcc -Wall -Wextra -Werror -pedantic -c neurolib.c
  gcc -Wall -Wextra -Werror -pedantic -c jason.c
  gcc -o jason neurolib.o jason.o -lssl -lcrypto
  ```

- README Filepath: jason/README.md

- Τα αρχεία JSON θα είναι μέχρι 1MB σε μέγεθος.

- Πρέπει να ολοκληρώνει την εκτέλεση μέσα σε: 1 δευτερόλεπτο.

- Το πρόγραμμά σας θα πρέπει να έχει τα λιγότερα δυνατά memory leaks.

Παραδείγματα εκτέλεσης ακολουθούν, πρώτα σε extraction mode:

```
$ ./jason --extract json/1.json
Why do programmers always mix up Halloween and Christmas?
Because Oct 31 == Dec 25!
$ echo $?
0
$ ./jason --extract json/2.json 2> stderr
$ echo $?
1
$ cat stderr
```

```
Not an accepted JSON!
```

Και στην συνέχεια σε conversation mode:

```
$ ./jason --bot
> What would you like to know? What is the last digit of pi?
I'd answer that, but I don't want to ruin the surprise.
> What would you like to know? What is the age of the universe?
I could tell you, but then I'd have to awkwardly dance away without explaining why.
> What would you like to know? Terminating
```

Ή μπορείτε να αλληλεπιδράσετε με μια πραγματική υπηρεσία τεχνητής νοημοσύνης (αν αγοράσετε tokens):

```
$ export OPENAI_API_KEY=... # enter your key here
$ ./jason --bot
> What would you like to know? What is the last digit of pi?
Pi (pi) is an irrational number, which means it has an infinite
number of decimal places and does not terminate. Therefore, it
does not have a last digit. The decimal representation of pi begins
with 3.14159 and continues indefinitely without repeating.
> What would you like to know? What is the age of the universe?
As of my last knowledge update in October 2023, the age of the
universe is estimated to be about 13.8 billion years. This estimate
is based on measurements of the cosmic microwave background radiation,
the expansion rate of the universe (Hubble constant), and observations
of the oldest known star clusters. However, scientific understanding is
always evolving, so it's possible that new discoveries could refine
this estimate in the future.
> What would you like to know? Terminating
```

Αν φτάσατε μέχρι εδώ: (1) συγχαρητήρια, (2) μην ξεχάσετε το README.md και (3) όπως πάντα ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.

Υπόδειξη Διαβάστε όλο το αρχείο σε ένα δυναμικό buffer και γράψτε έναν μικρό αναδρομικό parser (recursive descent) με μία συνάρτηση ανά είδος τιμής (object, array, string, αριθμός, literal), που προχωρά έναν δείκτη πάνω στο κείμενο και αποτυγχάνει μόλις κάτι δεν ταιριάζει. Δεν χρειάζεται να χτίσετε όλο το δέντρο: αρκεί να ακολουθήσετε τη διαδρομή `choices` → `[0]` → `message` → `content` και να μετατρέψετε τα escapes (`\n`, `\"`, `\\`, ...) κατά την εκτύπωση. Για το `--bot`, ξεκινήστε από το `neurolib.h`, και ελέγξτε με valgrind ότι ελευθερώνετε κάθε απάντηση.

Θέματα εξετάσεων (A14.11–A14.22)

A14.11 · Η συνάρτηση dog

A14.11 · *Εξέταση Ιανουαρίου 2025, Θέμα 2* · ★☆☆ · trace · `exam-2025-jan-q2`

Η συνάρτηση dog (10 Μονάδες)

```
char *dog(const char *str1, const char *str2) {
    size_t len1 = strlen(str1);
    size_t len2 = strlen(str2);
    char *result = malloc(len1 + len2 + 1);
    if (!result) return NULL;
    char *ptr = result;
    while (*str1) *ptr++ = *str1++;
    while (*str2) *ptr++ = *str2++;
    *ptr = '\0';
    return result;
}
```

Τι κάνει η συνάρτηση dog (μέχρι 15 λέξεις εξήγηση); Τι θα τυπώσουν οι παρακάτω εντολές;

```
char arg1[] = {71, 111, 111, 100, 32, 0};
char arg2[5] = {'j', 111, 98, '!', 0};
printf("%s\n", dog(arg1, arg2));
```

Υπόδειξη Παρακολουθήστε πού δείχνει ο `ptr` μετά από κάθε βρόχο και γιατί η `malloc` ζητά `len1 + len2 + 1` bytes. Για την έξοδο, μετατρέψτε τους αριθμούς σε χαρακτήρες με τον πίνακα ASCII (το 0 είναι ο τερματικός χαρακτήρας και το 32 το κενό). Σκεφτείτε επίσης τι γίνεται με τη μνήμη που επιστρέφει η dog σε αυτή την κλήση.

A14.12 · Η συνάρτηση transform

A14.12 · *Εξέταση Σεπτεμβρίου 2025, Θέμα 2* · ★☆☆ · trace · `exam-2025-sep-q2`

Η συνάρτηση transform (15 Μονάδες)

```
char *transform(char * str) {
    int length = strlen(str);
    char * lo = str;
    char * hi = str + length - 1;
    char tmp;
    while (lo < hi) {
      tmp = *lo;
      *lo++ = *hi;
```

```
      *hi-- = tmp;
    }
    return str;
}
```

Τι κάνει η συνάρτηση transform (μέχρι 15 λέξεις εξήγηση); Τι θα τυπωθεί κατά την εκτέλεση των δύο ακόλουθων εντολών;

```
char arg[] = {'!', 121, 107, 99, 117, 76, 0};
printf("%s\n", transform(arg));
```

Σημείωση: στο τέλος της εξέτασης δίνεται πίνακας ASCII ως βοήθημα.

Υπόδειξη Παρακολουθήστε πώς κινούνται οι δείκτες `lo` και `hi` και τι ανταλλάσσουν σε κάθε επανάληψη, και πότε σταματά ο βρόχος. Για την έξοδο, μετατρέψτε πρώτα κάθε αριθμό του `arg` σε χαρακτήρα με τον πίνακα ASCII· το `0` στο τέλος είναι ο τερματικός χαρακτήρας.

A14.13 · Ταιριαστές Καρδιές

A14.13 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #0 (Valentine's Themed), Θέμα 2* · ★★☆ · programming · `exam-2023-fall-ex0-q2`

Πρόγραμμα: `hearts.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που για κάθε όρισμα από την γραμμή εντολών ελέγχει αν η συμβολοακολουθία του ορίσματος περιέχει μόνο *ταιριαστές* καρδιές. Οι ταιριαστές καρδιές ορίζονται ως εξής:

1. Μπορούν να περιέχουν μόνο χαρακτήρες '<' και '3'.
2. Κάθε καρδιά που κλείνει ('3') πρέπει πρώτα να έχει ανοίξει ('<').
3. Όλες οι καρδιές που άνοιξαν κλείνουν.

Παράδειγμα εκτέλεσης:

```
$ gcc -o hearts hearts.c
$ ./hearts '<3' '<.3>' '<<33' '<<<3' '<333' '<<33<3<<<<3<<33333'
<3: yes
<.3>: no
<<33: yes
<<<3: no
<333: no
<<33<3<<<<3<<33333: yes
```

Υπόδειξη Είναι το κλασικό πρόβλημα των ισορροπημένων παρενθέσεων: αρκεί ένας μετρητής «ανοιχτών» καρδιών που ανεβαίνει με `'<'` και κατεβαίνει με `'3'`. Ποιες δύο συνθήκες

πρέπει να ελέγξετε, μία κατά τη διάρκεια της σάρωσης και μία στο τέλος; Μην ξεχάσετε τους άκυρους χαρακτήρες.

A14.14 · Εντοπισμός Διπλών Ορισμάτων

A14.14 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #1 (Coreutils Themed), Θέμα 2* · ★★☆ · programming · `exam-2023-fall-ex1-q2`

Πρόγραμμα: `duplicate.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που για κάθε όρισμα από την γραμμή εντολών που επαναλαμβάνεται δύο ή περισσότερες φορές τυπώνεται ως πολλαπλό (”dup”) μία φορά στην πρότυπη έξοδο (stdout). Παράδειγμα εκτέλεσης ακολουθεί:

```
$ gcc -o duplicate duplicate.c
$ ./duplicate foo bar foo baz bank zonk bazinga foo zonk
dup: foo
dup: zonk
```

Υπόδειξη Συγκρίνετε τα ορίσματα ανά δύο με `strcmp`. Το δύσκολο σημείο είναι το «μία φορά»: το `foo` εμφανίζεται τρεις φορές αλλά τυπώνεται μόνο μία. Ένας τρόπος είναι να αποφασίζετε για κάθε όρισμα μόνο στην πρώτη του εμφάνιση· άλλος είναι να ταξινομήσετε πρώτα, αλλά τότε σκεφτείτε αν αλλάζει η σειρά εξόδου.

A14.15 · Προσεγγίζοντας την Λέξη

A14.15 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #12, Θέμα 2* · ★★☆ · programming · `exam-2023-fall-ex12-q2`

Πρόγραμμα: `check.c`

Γράψτε ένα πρόγραμμα που να δέχεται ως πρώτο όρισμα μια λέξη στόχο (goal) ακολουθούμενο από μια σειρά από προσπάθειες του χρήστη να μαντέψει την λέξη στόχο και να τυπώνει στην πρότυπη έξοδο την εγγύτητα της κάθε προσπάθειας στην λέξη στόχο. Η εγγύτητα της κάθε λέξης εξαρτάται από δύο μετρικές: (1) πόσα σωστά γράμματα μάντεψε ο χρήστης και είναι σε λάθος θέση και (2) πόσα σωστά γράμματα μάντεψε ο χρήστης και είναι στην σωστή θέση. Όλες οι λέξεις θα αποτελούνται από λατινικούς χαρακτήρες και θα έχουν το ίδιο μήκος. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o check check.c
$ ./check unlit legal vinyl sling inlet unlit
legal has 0 correct letters in correct position and 1 correct letters in an
 ↪  incorrect position
```

```
vinyl has 0 correct letters in correct position and 3 correct letters in an
 ↪  incorrect position
sling has 0 correct letters in correct position and 3 correct letters in an
 ↪  incorrect position
inlet has 3 correct letters in correct position and 1 correct letters in an
 ↪  incorrect position
unlit has 5 correct letters in correct position and 0 correct letters in an
 ↪  incorrect position
$ ./check hello world worll wrllo
world has 1 correct letters in correct position and 1 correct letters in an
 ↪  incorrect position
worll has 1 correct letters in correct position and 2 correct letters in an
 ↪  incorrect position
wrllo has 3 correct letters in correct position and 0 correct letters in an
 ↪  incorrect position
```

Υπόδειξη Μετρήστε πρώτα τις ακριβείς θέσεις. Για τα γράμματα σε λάθος θέση, προσέξτε τα διπλά γράμματα: φτιάξτε πίνακες συχνοτήτων (26 θέσεις) μόνο για τα γράμματα που δεν ταίριαξαν ακριβώς, σε στόχο και προσπάθεια, και αθροίστε το ελάχιστο ανά γράμμα. Λέξεις με διαφορετικό μήκος είναι σφάλμα.

A14.16 · Κρεμάλα

A14.16 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #12, Θέμα 3* · ★★☆ · programming · `exam-2023-fall-ex12-q3`

Πρόγραμμα: `hangman.c`

Γράψτε ένα πρόγραμμα που παίρνει ως πρώτο όρισμα το όνομα ενός αρχείου που περιέχει όλες τις επιτρεπτές λέξεις που μπορεί να βάλει κάποιος (μία ανά γραμμή) και ως δεύτερο όρισμα τα γράμματα της λέξης που έχουμε βρει σε ένα παιχνίδι κρεμάλας και τυπώνει στην πρότυπη έξοδο όλες τις πιθανές λέξεις που μπορεί να ταιριάζουν. Εάν δεν υπάρχει λέξη που να ταιριάζει, το πρόγραμμά σας θα πρέπει να τυπώνει ανάλογο μήνυμα. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o hangman hangman.c
$ cat words.txt
cat
DOG
Giraffe
tennis
turtle
$ ./hangman words.txt t_____
tennis
```

```
turtle
$ ./hangman words.txt __r___
turtle
$ ./hangman words.txt ___
cat
DOG
$ ./hangman words.txt d__
DOG
$ ./hangman words.txt _e_e__
No words matching guess found
$ ./hangman /usr/share/dict/words __ou________ing
troubleshooting
```

Υπόδειξη Διαβάστε το λεξικό γραμμή-γραμμή και κρατήστε μόνο λέξεις με το ίδιο μήκος με το μοτίβο. Το _ ταιριάζει με οποιοδήποτε γράμμα, ενώ τα γράμματα συγκρίνονται χωρίς διάκριση πεζών-κεφαλαίων (`tolower`). Μην ξεχάσετε να αφαιρέσετε το `'\n'` στο τέλος κάθε γραμμής.

A14.17 · Δυνατότητες

A14.17 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #15, Θέμα 4* · ★★☆ · programming · `exam-2023-fall-ex15-q4`

Πρόγραμμα: `phone.c`

Σε κάποια τηλέφωνα, ίσως έχετε παρατηρήσει πως ο κάθε αριθμός αντιστοιχεί σε συγκεκριμένα γράμματα:

```
1       2 ABC   3 DEF
4 GHI   5 JKL   6 MNO
7 PQRS  8 TUV   9 WXYZ
*       0       #
```

Σημείωση: στο πρωτότυπο το πληκτρολόγιο δίνεται ως εικόνα (phone.jpeg).

(2 -> ABC, 3 -> DEF, ... κοκ). Λέγοντας έναν αριθμό, μπορείς να περιγράψεις ένα σύνολο από λέξεις και το αντίστροφο: μια λέξη προσδιορίζει έναν συγκεκριμένο αριθμό. Για παράδειγμα, η λέξη CAT προσδιορίζει το 228 (C -> 2, A -> 2, T -> 8). Γράψτε ένα πρόγραμμα, το οποίο παίρνει ως πρώτο όρισμα ένα αρχείο με λέξεις (μία ανά γραμμή) και έναν αριθμό ως το δεύτερο όρισμα και τυπώνει όλες τις λέξεις που μπορούν να εκφραστούν με αυτόν τον αριθμό. Οποιοδήποτε ψηφίο δεν ταιριάζει σε κάποιο γράμμα (0, 1) θεωρείται ”μπαλαντέρ”, δηλαδή ταιριάζει με οποιοδήποτε γράμμα. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o phone phone.c
$ ./phone words.txt
```

```
Usage: ./phone dictionary phone
$ cat words.txt
cat
dog
deep
purple
smoke
water
rain
spain
plain
$ ./phone words.txt 228
cat
$ ./phone words.txt 70000
smoke
spain
plain
$ ./phone words.txt 71246
spain
plain
$ ./phone /usr/share/dict/words 71246
Rabin
Robin
Rubin
Sabin
Spahn
Spain
plain
robin
slain
stain
swain
```

Υπόδειξη Φτιάξτε έναν πίνακα 26 θέσεων που αντιστοιχίζει κάθε γράμμα σε ψηφίο. Για κάθε λέξη του λεξικού ίδιου μήκους με τον αριθμό, συγκρίνετε θέση-θέση χωρίς διάκριση πεζών-κεφαλαίων, με τα 0 και 1 να ταιριάζουν με οτιδήποτε. Προσέξτε λέξεις με μη λατινικούς χαρακτήρες και το `'\n'` στο τέλος κάθε γραμμής.

A14.18 · Εύρεση Λέξεων

A14.18 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #3 (Frozen Themed), Θέμα 3* · ★★☆ · programming · `exam-2023-fall-ex3-q3`

Πρόγραμμα: `find.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που παίρνει δύο ορίσματα: (1) το όνομα του αρχείου που περιέχει το σύνολο των λέξεων που έχουμε διαθέσιμες και (2) την υπακολουθία που ψάχνουμε μέσα σε αυτές τις λέξεις. Το πρόγραμμά μας πρέπει να τυπώνει όσες λέξεις περιέχουν την υπακολουθία (για παράδειγμα το ”snow” περιέχεται στο ”snowman” αλλά δεν περιέχεται στο ”noman”) και το μέρος που ταιριάζει στην υπακολουθία πρέπει να τονιστεί με **bold** χαρακτήρες. Προσοχή: μια υπακολουθία μπορεί να υπάρχει περισσότερες από μία φορές σε μια λέξη. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ cat test.txt
Anna
Arendelle
Sven
Hans
Ice
Snow
$ gcc -o find find.c
$ ./find test.txt e
Arendelle
Sven
Ice
$ ./find /usr/share/dict/words snowm
snowman
snowman's
snowmen
snowmobile
snowmobile's
snowmobiled
snowmobiles
snowmobiling
```

> Σημείωση: στο πρωτότυπο το παράδειγμα είναι στιγμιότυπο τερματικού (εικόνα), όπου τα τμήματα που ταιριάζουν (όλα τα `e` στο `Arendelle`, `Sven`, `Ice` και το `snowm` στις υπόλοιπες) εμφανίζονται με έντονους χαρακτήρες.

Υπόδειξη Διαβάστε το αρχείο γραμμή-γραμμή με `fgets` (αφαιρώντας το `'\n'`) και σε κάθε λέξη ψάξτε όλες τις εμφανίσεις της υπακολουθίας, π.χ. με επαναλαμβανόμενες κλήσεις της `strstr` που συνεχίζουν μετά το προηγούμενο ταίριασμα. Τυπώστε τα κομμάτια ανάμεσα στα ταιριάσματα κανονικά και τα ταιριάσματα ανάμεσα στις ANSI escape sequences για bold και επαναφορά. Σκεφτείτε τι σημαίνει κενή υπακολουθία και τι γίνεται με επικαλυπτόμενα ταιριάσματα (π.χ. `aa` μέσα στο `aaa`).

A14.19 · Τα Πάνω Κάτω

A14.19 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #4 (Rick Astley Themed), Θέμα 2* · ★★☆ · programming · `exam-2023-fall-ex4-q2`

Πρόγραμμα: `updown.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που διαβάζει το κείμενο που δίνεται από την πρότυπη είσοδο (stdin) και τυπώνει την κάθε λέξη χαρακτήρα στην πρότυπη έξοδο (stdout) έχοντας κάνει την εξής αντικατάσταση: κάθε λέξη "up" πρέπει να αντικατασταθεί με την λέξη "down" και κάθε "down" πρέπει να αντικατασταθεί με την λέξη "up". Οι λέξεις χωρίζονται με κενά (whitespace). Η αντικατάσταση πρέπει να γίνεται σε ολόκληρες λέξεις, όχι μέρος τους. Παράδειγμα εκτέλεσης:

```
$ cat rick.txt
upload and download this :
never gonna give you up ,
never gonna let you down ,
never gonna run around and desert you !
$ gcc -o updown updown.c
$ ./updown < rick.txt
upload and download this :
never gonna give you down ,
never gonna let you up ,
never gonna run around and desert you !
```

Υπόδειξη Συγκεντρώστε κάθε λέξη (μέγιστη ακολουθία χαρακτήρων που δεν είναι whitespace) σε έναν buffer και, μόλις τελειώσει, συγκρίνετέ την με `strcmp` με τα `"up"` και `"down"` πριν την τυπώσετε· τα whitespace αντιγράφονται αυτούσια. Προσοχή στις πολύ μεγάλες λέξεις (δεν χωράνε σε σταθερό buffer, αλλά αρκεί να ξέρετε ότι δεν είναι ούτε up ούτε down) και στην τελευταία λέξη όταν το αρχείο δεν τελειώνει σε αλλαγή γραμμής.

A14.20 · Σπάσε το PIN

A14.20 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #6 (Crypto Themed), Θέμα 3* · ★★☆ · programming · `exam-2023-fall-ex6-q3`

Πρόγραμμα: `pin.c` (25 μονάδες)

Το αρχείο pin.c προσομοιάζει τον έλεγχο ενός συστήματος που ελέγχει αν το pin σου είναι σωστό και αν επιτρέπεται η πρόσβαση. Βρείτε το pin και προσθέστε το στο πρόγραμμα προκειμένου να τυπώσει "Access granted". Δεν επιτρέπεται να αλλάξετε τις ρουτίνες υπολογισμού της ορθότητας του pin αλλά μπορείτε να κάνετε όποιες άλλες δοκιμές / αλλαγές θέλετε. Γνωρίζουμε ότι ο pin είναι τετραψήφιος. Βεβαιωθείτε ότι καταγράψατε στο αρχείο όλον τον κώδικα που γράψατε για να βρείτε το pin. Παράδειγμα επιτυχούς εκτέλεσης ακολουθεί:

```
$ gcc -o pin pin.c
$ ./pin
Access granted
```

Το αρχείο pin.c:

```c
// Program that checks whether a pin is correct.

#include <stdio.h>

// Password converter routine, cannot be changed
int converter(const char *str) {
    int num = 5381;
    int c;
    while ((c = *str++)) {
        num = ((num << 5) + num) + c;
    }
    return num;
}

int main() {

    // Γνωρίζουμε ότι το πρώτο ψηφίο του pin είναι το 7,
    // αλλά δεν γνωρίζουμε την συνέχεια. Αλλάξτε το pin προκειμένου
    // να μπορέσουμε να πάρουμε πρόσβαση. Γράψτε όσον κώδικα χρειαστήκατε
    // και όποιες μεθόδους χρησιμοποιήσατε για να το βρείτε σε αυτό το
    // αρχείο. Μπορείτε να βάλετε εδώ το σωστό pin όταν το βρείτε.
    char pin[] = "7...";



    // Cannot change the checks below
    if (converter(pin) == 2088511771) {
        printf("Access granted\n");
    } else {
        printf("Access denied\n");
    }

    return 0;
}
```

Υπόδειξη Η `converter` είναι μια συνάρτηση κατακερματισμού που δεν αντιστρέφεται εύκολα, αλλά οι υποψήφιοι τετραψήφιοι κωδικοί είναι ελάχιστοι: δοκιμάστε τους όλους (ωμή βία). Χρειάζεται να φτιάχνετε για κάθε υποψήφιο την αντίστοιχη συμβολοσειρά τεσσάρων

ψηφίων (π.χ. με `sprintf` ή γεμίζοντας τις θέσεις του πίνακα με `'0' + ψηφίο`) και να κρατήσετε τον κώδικα αναζήτησης στο αρχείο, όπως ζητά η εκφώνηση.

A14.21 · Συνένωση Αλφαριθμητικών - join

A14.21 · *Εξέταση Ιανουαρίου 2025, Θέμα 5* · ★★☆ · programming · `exam-2025-jan-q5`

Συνένωση Αλφαριθμητικών - join [20 Μονάδες]

Γράψτε μία *συνάρτηση* `join` η οποία να λαμβάνει ως ορίσματα έναν πίνακα elements από αλφαριθμητικά (strings) και ένα delimiter αλφαριθμητικό (string) και να επιστρέφει ένα νέο αλφαριθμητικό με όλα τα αλφαριθμητικά συνενωμένα μεταξύ τους με το αλφαριθμητικό delimiter. Για παράδειγμα, αν δώσουμε ως elements τον πίνακα `{"hall", "oates"}` και ως delimiter το `" & "`, τότε η συνάρτηση θέλουμε να μας επιστρέψει το αλφαριθμητικό `"hall & oates"`. Αντίστοιχα, αν δώσουμε ως elements τον πίνακα `{"A", "Bing", "Boom"}` και ως delimiter το `"Bada"`, τότε πρέπει να μας επιστρέψει `"ABadaBingBadaBoom"`. Η συνάρτηση μπορεί να έχει οποιαδήποτε διεπαφή (τύπο επιστροφής / επιπλέον ορίσματα) επιθυμείτε. Τι χρονική και χωρική πολυπλοκότητα έχει ο αλγόριθμός σας (5/20 της βαθμολογίας);

Υπόδειξη Η συνάρτηση πρέπει να ξέρει πόσα στοιχεία έχει ο πίνακας, οπότε περάστε το πλήθος ως επιπλέον όρισμα. Υπολογίστε πρώτα το ακριβές μήκος του αποτελέσματος (τα μήκη των στοιχείων, `n - 1` φορές τον delimiter και τον τερματικό χαρακτήρα), κάντε μία `malloc` και μετά αντιγράψτε κρατώντας έναν δείκτη στο τέλος. Προσέξτε ότι επαναλαμβανόμενη `strcat` από την αρχή του αποτελέσματος χαλάει την πολυπλοκότητα, και σκεφτείτε την περίπτωση άδειου πίνακα.

A14.22 · Μεταμορφώσιμες Προτάσεις

A14.22 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #5 (HP Themed), Θέμα 4* · ★★★ · programming · `exam-2023-fall-ex5-q4`

Πρόγραμμα: `meta.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που παίρνει 2 προτάσεις ως ορίσματα από την γραμμή εντολών και τυπώνει αν η μία πρόταση μπορεί να μεταμορφωθεί στην άλλη κάνοντας αντικαταστάσεις λέξεων καθώς και ποιες αντικαταστάσεις απαιτούνται. Για να μπορεί να μεταμορφωθεί μια πρόταση σε μια δεύτερη πρέπει *όλες* οι λέξεις της πρώτης να ταιριάξουν με λέξεις της δεύτερης μία προς μία. Οι προτάσεις θα έχουν μόνο κενά και λατινικούς χαρακτήρες (A-Z, a-z). Παραδείγματα εκτέλεσης ακολουθούν:

```
$ ./meta "I solemnly swear I am up to no good" "Magic glimmers brightly Magic
 ↪  creatures stand under the moonlight"
You can metamorphose the first sentence into the second sentence
```

```
I <-> Magic
solemnly <-> glimmers
swear <-> brightly
am <-> creatures
up <-> stand
to <-> under
no <-> the
good <-> moonlight
$ ./meta "I solemnly swear I am up to no good" "Magic glimmers brightly magic
 ↪  creatures stand under the moonlight"
You cannot metamorphose the first sentence into the second sentence
```

Υπόδειξη Χωρίστε κάθε πρόταση σε πίνακα λέξεων και απαιτήστε ίδιο πλήθος λέξεων. Η αντιστοίχιση πρέπει να είναι ένα-προς-ένα και προς τις δύο κατευθύνσεις: η ίδια λέξη της πρώτης πρότασης πρέπει πάντα να πηγαίνει στην ίδια λέξη της δεύτερης, και δύο διαφορετικές λέξεις δεν μπορούν να πάνε στην ίδια. Προσέξτε ότι η σύγκριση κάνει διάκριση πεζών-κεφαλαίων (δεύτερο παράδειγμα) και ότι κάθε ζεύγος τυπώνεται μία φορά, με τη σειρά πρώτης εμφάνισης.

Κεφάλαιο 15: Πολυπλοκότητα και Προεπεξεργαστής

Kahoot από το αμφιθέατρο (K15.1–K15.8)

K15.1 · Γραμμική αναζήτηση σε αμφιθέατρο n × n

K15.1 · *Kahoot «Δυαδική Αναζήτηση, Ταξινόμηση, Πολυπλοκότητα και άλλα» και «Πολυπλοκότητα και άλλα» (διάλεξη 15)* · ★☆☆ · multiple-choice · 75% σωστές · `kahoot-seat-search-grid`

Ψάχνω έναν γνωστό μου σε αμφιθέατρο διαστάσεων n × n, κοιτώντας μία-μία τις θέσεις. Ο αλγόριθμός μου είναι σε χρόνο:

- O(n)
- O(f(x))
- $O(n^2)$
- O(1)

Υπόδειξη Πόσες θέσεις έχει συνολικά το αμφιθέατρο, και πόσες θα κοιτάξετε στη χειρότερη περίπτωση;

Κ15.2 · Ορισμός μακροεντολής

Κ15.2 · *Kahoot «Πολυπλοκότητα και άλλα» (διάλεξη 15)* · ★☆☆ · multiple-choice · 74% σωστές · `kahoot-define-macro`

Για να ορίσω μια μακροεντολή (macro) χρησιμοποιώ την εντολή:

- `#ifdef`
- `#include`
- `#define`
- `#ifndef`

Υπόδειξη Από τις τέσσερις οδηγίες, δύο αφορούν μεταγλώττιση υπό συνθήκη και μία την εισαγωγή αρχείου.

Κ15.3 · Τι σημαίνει O(n)

Κ15.3 · *Kahoot «Δυαδική Αναζήτηση, Ταξινόμηση, Πολυπλοκότητα και άλλα» και «Πολυπλοκότητα και άλλα» (διάλεξη 15)* · ★★☆ · multiple-choice · 62% σωστές · `kahoot-big-o-worst-case`

Ένας αλγόριθμος έχει χρονική πολυπλοκότητα O(n). Αυτό σημαίνει ότι χρειάζεται γραμμικό χρόνο:

- στην καλύτερη (πιο γρήγορη) περίπτωση
- στην χειρότερη (πιο αργή) περίπτωση
- στην μέση περίπτωση
- σε κάθε περίπτωση

Υπόδειξη Το Big-O είναι άνω φράγμα· σκεφτείτε για ποια από τις περιπτώσεις ένα άνω φράγμα δίνει εγγύηση για όλες τις εισόδους.

Κ15.4 · Πολυπλοκότητα πολλαπλασιασμού πινάκων

Κ15.4 · *Kahoot «Δυαδική Αναζήτηση, Ταξινόμηση, Πολυπλοκότητα και άλλα» και «Πολυπλοκότητα και άλλα» (διάλεξη 15)* · ★★☆ · multiple-choice · 49% σωστές · `kahoot-matrix-multiplication`

Μια συνάρτηση πολλαπλασιάζει δύο πίνακες `m1`, `m2` διαστάσεων N × N (με τον συνήθη αλγόριθμο, γραμμή επί στήλη). Η χρονική πολυπλοκότητά της είναι:

- $O(n^2)$
- O(n!)
- O(n)
- O(τρέχα γύρευε)

- O(log n)
- O(n³)

Συχνή παρανόηση Το 23% επέλεξε O(n²), μετρώντας μόνο τα N² στοιχεία του αποτελέσματος και όχι το εσωτερικό άθροισμα N γινομένων για το καθένα.

Υπόδειξη Μετρήστε πόσα στοιχεία έχει ο πίνακας-αποτέλεσμα και πόσους πολλαπλασιασμούς χρειάζεται ο υπολογισμός του καθενός.

K15.5 · Πολυπλοκότητα μέτρησης bit

K15.5 · *Kahoot «Πολυπλοκότητα και άλλα» (διάλεξη 15)* · ★★☆ · multiple-choice · 43% σωστές · `kahoot-count-set-bits`

Μια συνάρτηση παίρνει έναν αριθμό n και επιστρέφει πόσα bit είναι 1. Πόσο γρήγορη μπορώ να κάνω αυτήν τη συνάρτηση;

- O(n³)
- O(n²)
- O(n)
- O(f(x))
- O(log n)

Συχνή παρανόηση Το 32% επέλεξε O(n), μπερδεύοντας την τιμή του n με το πλήθος των bit του, που είναι περίπου $\log_2 n$.

Υπόδειξη Πόσα bit χρειάζονται για να γράψετε τον αριθμό n στο δυαδικό;

K15.6 · Πολυπλοκότητα μέτρησης ψηφίων

K15.6 · *Kahoot «Δυαδική Αναζήτηση, Ταξινόμηση, Πολυπλοκότητα και άλλα» και «Πολυπλοκότητα και άλλα» (διάλεξη 15)* · ★★★ · multiple-choice · 38% σωστές · `kahoot-count-digits`

Μετράω τα ψηφία οποιουδήποτε αριθμού n μου δοθεί. Η χρονική πολυπλοκότητα του αλγορίθμου μου είναι:

- O(n)
- O(n²)
- O(f(x))
- O(log n)

Συχνή παρανόηση Το 45% επέλεξε O(n), μπερδεύοντας την τιμή του n με το μέγεθος της εισόδου: ο αλγόριθμος κάνει μία επανάληψη ανά ψηφίο, όχι ανά μονάδα της τιμής.

Υπόδειξη Πόσες φορές μπορείτε να διαιρέσετε το n με το 10 πριν φτάσει στο 0; Συγκρίνετε το πλήθος των ψηφίων του 1000 και του 1000000.

K15.7 · Η έξοδος του προεπεξεργαστή

K15.7 · *Kahoot «Δυαδική Αναζήτηση, Ταξινόμηση, Πολυπλοκότητα και άλλα» και «Πολυπλοκότητα και άλλα» (διάλεξη 15)* · ★★★ · multiple-choice · 38% σωστές · `kahoot-preprocessor-output`

Η έξοδος του προεπεξεργαστή είναι ένα αρχείο:

- Δυαδικού κώδικα
- Κώδικα assembly
- Πηγαίου Κώδικα
- Εξαρτάται

Συχνή παρανόηση Το 30% απάντησε «Εξαρτάται», χωρίς να έχει ξεκαθαρίσει τη θέση του προεπεξεργαστή στη σειρά προεπεξεργασία → μεταγλώττιση → συμβολομετάφραση → σύνδεση: ο προεπεξεργαστής κάνει πάντα μόνο αντικαταστάσεις κειμένου.

Υπόδειξη Σκεφτείτε τι κάνουν οι οδηγίες `#include` και `#define`: αντικαθιστούν κείμενο. Τι παραλαμβάνει στη συνέχεια ο μεταγλωττιστής; Δοκιμάστε `gcc -E`.

K15.8 · Πολυπλοκότητα O(n² + log n)

K15.8 · *Kahoot «Πολυπλοκότητα και άλλα» (διάλεξη 15)* · ★★★ · multiple-choice · 36% σωστές · `kahoot-big-o-sum-of-terms`

Ένας αλγόριθμος μπορεί να έχει πολυπλοκότητα $O(n^2 + \log n)$.

- True
- False

Συχνή παρανόηση Το 60% απάντησε False, θεωρώντας ότι το O(·) πρέπει να περιέχει έναν μόνο όρο· η έκφραση είναι σωστή, απλώς ισοδυναμεί με $O(n^2)$, γιατί ο κυρίαρχος όρος είναι το n^2.

Υπόδειξη Το $O(\cdot)$ περιγράφει ένα άνω φράγμα με μια συνάρτηση· σκεφτείτε αν είναι λάθος να γράψουμε μια συνάρτηση που δεν έχει απλοποιηθεί, και σε τι απλοποιείται αυτή.

Ζέσταμα: από τις διαφάνειες (A15.1–A15.15)

A15.1 · Πολυπλοκότητα της atoi

A15.1 · *Διάλεξη 15, διαφάνεια 15* · ★☆☆ · short-answer · `slides-lec15-complexity-atoi`

Θέλω μια συνάρτηση `atoi` που να παίρνει ένα πίνακα χαρακτήρων (μόνο ψηφία) και να επιστρέφει έναν ακέραιο. Πώς;

```
int atoi(char digits[]) {
  int result = 0;
  for(int i = 0; digits[i]; i++) {
    result = 10 * result + digits[i] - '0';
  }
  return result;
}
```

Τι χρονική και τι χωρική πολυπλοκότητα έχει η συνάρτηση;

Υπόδειξη Ορίστε πρώτα τι είναι το μέγεθος N της εισόδου εδώ (τι ακριβώς διατρέχει ο βρόχος;). Μετά μετρήστε τις επαναλήψεις και τις μεταβλητές που χρειάζονται.

A15.2 · Πολυπλοκότητα εύρεσης μέγιστου σε πίνακα N x N

A15.2 · *Διάλεξη 15, διαφάνεια 23* · ★☆☆ · short-answer · `slides-lec15-complexity-find-max-2d`

Εύρεση Μέγιστου Στοιχείου σε Πίνακα N x N. Τι χρονική και τι χωρική πολυπλοκότητα έχει η συνάρτηση;

```
int find_max(int **matrix, size_t n) {
  int i, j, max = -1;
  for(i = 0; i < n; i++) {
    for(j = 0; j < n; j++) {
      if (matrix[i][j] > max) max = matrix[i][j];
    }
  }
  return max;
}
```

Υπόδειξη Πόσες φορές τρέχει ο εσωτερικός βρόχος για *κάθε* επανάληψη του εξωτερικού; Μετά σκεφτείτε αν η αρχική τιμή `max = -1` είναι σωστή για κάθε πίνακα.

A15.3 · Πολυπλοκότητα υπολογισμού βαθμολογίας

A15.3 · *Διάλεξη 15, διαφάνεια 21* · ★☆☆ · short-answer · `slides-lec15-complexity-grade`

Το γνωστό μας παράδειγμα: Υπολογισμός Βαθμολογίας. Τι χρονική και τι χωρική πολυπλοκότητα έχει η συνάρτηση;

```
// Compute grades using the class formula
int grade(int final_exam, int homework, int lab, int year) {
  if (year <= 1) {
    return final_exam * 50 / 100 + homework * 30 / 100 + lab * 20 / 100;
  } else {
    return final_exam * 70 / 100 + homework * 30 / 100;
  }
}
```

Υπόδειξη Υπάρχει κάποια ποσότητα της εισόδου που κάνει τη συνάρτηση να εκτελεί περισσότερα βήματα; Αν ο αριθμός των πράξεων δεν εξαρτάται από την είσοδο, ποια κλάση είναι;

A15.4 · Πολυπλοκότητα δυναμικού πίνακα με malloc

A15.4 · *Διάλεξη 15, διαφάνεια 19* · ★☆☆ · short-answer · `slides-lec15-complexity-malloc`

Με την βοήθεια των δεικτών, μπορούμε να χρησιμοποιήσουμε δυναμικούς πίνακες, πίνακες των οποίων το μέγεθος καθορίζεται δυναμικά, δηλαδή την στιγμή που τρέχει το πρόγραμμα. Παράδειγμα:

```
int * array = malloc(N * sizeof(int));
for(int i = 0 ; i < N ; i++)
  array[i] = i * i;
```

Τι χρονική και τι χωρική πολυπλοκότητα έχει ο κώδικας ως προς το `N`;

Υπόδειξη Ο χρόνος και ο χώρος δεν έχουν πάντα την ίδια πολυπλοκότητα. Εδώ αναρωτηθείτε πόση μνήμη ζητά η `malloc` συναρτήσει του `N`.

A15.5 · Πολυπλοκότητα: γινόμενο περιττών πολλαπλασίων του 7

A15.5 · *Διάλεξη 15, διαφάνεια 13* · ★☆☆ · short-answer · `slides-lec15-complexity-odd-multiples-of-7`

Θέλω να βρω το γινόμενο όλων των περιττών από το 1 μέχρι το N που διαιρούνται με το 7. Πώς;

Ένα for loop με μια μεταβλητή που αυξάνεται και έλεγχο για mod 7 = 0:

```
for(product = 1, i = 1; i < N ; i += 2) {
  if ( i % 7 == 0 )
    product *= i;
}
```

Τι χρονική και τι χωρική πολυπλοκότητα έχει ο βρόχος ως προς το `N`;

Υπόδειξη Μετρήστε πόσες φορές εκτελείται το σώμα του βρόχου συναρτήσει του `N`, και θυμηθείτε τι κάνει ο συμβολισμός O στους σταθερούς παράγοντες. Για τον χώρο, ρωτήστε αν κάποια δομή μεγαλώνει με το `N`. Ελέγξτε επίσης αν ο βρόχος εξετάζει το ίδιο το `N`.

A15.6 · Πολυπλοκότητα της strlen

A15.6 · *Διάλεξη 15, διαφάνεια 29* · ★☆☆ · short-answer · `slides-lec15-complexity-strlen`

Η συνάρτηση strlen. Μια πιθανή υλοποίηση:

```
size_t strlen(char * str) {
  size_t length = 0;
  while(*str++) length++;
  return length;
}
```

Τι πολυπλοκότητας είναι η συνάρτηση strlen ως προς το μέγεθος της συμβολοσειράς;

Υπόδειξη Πότε σταματά ο βρόχος και πόσους χαρακτήρες έχει περάσει μέχρι τότε; Υπάρχει τρόπος να βρεθεί το μήκος χωρίς να διατρέξουμε τη συμβολοσειρά;

A15.7 · Τι κάνει το πρόγραμμα με την getchar

A15.7 · *Διάλεξη 15, διαφάνεια 17* · ★☆☆ · trace · `slides-lec15-getchar-count`

Διαδοχικές κλήσεις της `getchar()` διαβάζουν διαδοχικούς χαρακτήρες. Τι κάνει το παρακάτω πρόγραμμα; Τι χρονική και τι χωρική πολυπλοκότητα έχει;

```c
#include <stdio.h>

int main() {
  int ch, sum = 0;
  printf("Enter characters: ");
  while( (ch = getchar()) != '\n' && ch != EOF ) {
    printf("%c", ch);
    sum++;
  }
  printf("\nTotal characters: %d\n", sum);
  return 0;
}
```

Υπόδειξη Ακολουθήστε τον βρόχο για την είσοδο `hello` και Enter: πότε σταματά και τι τυπώνεται σε κάθε επανάληψη; Για τον χώρο, ρωτήστε αν το πρόγραμμα κρατά κάπου όλη τη γραμμή.

A15.8 · Τι επιστρέφει το πρόγραμμα με τη μακροεντολή PROD

A15.8 · *Διάλεξη 15, διαφάνεια 47* · ★☆☆ · trace · `slides-lec15-macro-prod`

Η μακροεντολή αντικαθίσταται στον κώδικα πριν την μεταγλώττιση. Τι θα επιστρέψει το παρακάτω πρόγραμμα;

```c
#define PROD 2*5
int main() {
  return 20 / PROD;
}
```

Υπόδειξη Γράψτε τη γραμμή `return` όπως θα τη δει ο μεταγλωττιστής μετά την αντικατάσταση (ή τρέξτε `cpp` στο αρχείο), και υπολογίστε την παράσταση με τους κανόνες προτεραιότητας και προσεταιριστικότητας. Την τιμή επιστροφής τη βλέπετε με `echo $?`.

A15.9 · Σε τι προεπεξεργάζεται το #if 0

A15.9 · *Διάλεξη 15, διαφάνεια 50* · ★☆☆ · trace · `slides-lec15-preprocess-if-else`

Οι εντολές `#if` `#else` `#endif` έχουν μορφή παρόμοια με την δομή ελέγχου if στην C αλλά δρουν στο επίπεδο του κώδικα. Σε τι θα προεπεξεργαστεί το ακόλουθο πρόγραμμα;

```c
int main() {
#if 0
```

```
  return 42;
#else
  return 1;
#endif
}
```

Υπόδειξη Η συνθήκη του `#if` είναι σταθερά: είναι αληθής ή ψευδής; Κρατήστε μόνο τις γραμμές του κλάδου που επιλέγεται και σβήστε όλες τις γραμμές που αρχίζουν με `#`. Επιβεβαιώστε με `gcc -E`.

A15.10 · Πολυπλοκότητα του αναδρομικού παραγοντικού

A15.10 · *Διάλεξη 15, διαφάνεια 25* · ★★☆ · short-answer · `slides-lec15-complexity-factorial`

Η συνάρτηση παραγοντικό (factorial). Τι χρονική και τι χωρική πολυπλοκότητα έχει;

```
int factorial(int n) {
  if (n == 0) return 1;
  return n * factorial(n - 1);
}
```

Υπόδειξη Για τον χρόνο, μετρήστε τις κλήσεις. Για τον χώρο, μην κοιτάξετε μόνο τις μεταβλητές μιας κλήσης: πόσες κλήσεις είναι ενεργές ταυτόχρονα στη στοίβα;

A15.11 · Πολυπλοκότητα του αναδρομικού Fibonacci

A15.11 · *Διάλεξη 15, διαφάνεια 27* · ★★☆ · short-answer · `slides-lec15-complexity-fibonacci`

Η συνάρτηση fibonacci. Τι χρονική και τι χωρική πολυπλοκότητα έχει;

```
int fib(int n) {
  if (n == 0 || n == 1) return 1;
  return fib(n - 1) + fib(n - 2);
}
```

Υπόδειξη Σχεδιάστε το δέντρο κλήσεων του `fib(4)` ή του `fib(5)`: πόσα παιδιά έχει κάθε κόμβος και πόσο βαθύ είναι το δέντρο; Ο χρόνος εξαρτάται από το πλήθος των κόμβων, ο χώρος από το μακρύτερο μονοπάτι.

A15.12 · Πολυπλοκότητα εύρεσης κατόπτρου ακεραίου

A15.12 · *Διάλεξη 15, διαφάνεια 37* · ★★☆ · short-answer · `slides-lec15-complexity-mirror`

Εύρεση του κατόπτρου ενός ακεραίου. Τι χρονική και τι χωρική πολυπλοκότητα έχει η συνάρτηση ως προς την τιμή του n;

```
int mirror(int n) {
  int result = 0, tmp;
  while(n > 0) {
    tmp = n % 10;
    result = 10 * result + tmp;
    n /= 10;
  }
  return result;
}
```

Υπόδειξη Τι παθαίνει το n σε κάθε επανάληψη; Πόσες επαναλήψεις γίνονται για `n = 9`, `99`, `999`; Ποια συνάρτηση του n μετρά αυτό το πλήθος;

A15.13 · Άθροισμα τέλειων τετραγώνων: δύο εκδοχές

A15.13 · *Διάλεξη 15, διαφάνεια 33* · ★★☆ · short-answer · `slides-lec15-complexity-perfect-squares`

Άθροισμα τέλειων τετραγώνων. Τι χρονική και τι χωρική πολυπλοκότητα έχει η πρώτη εκδοχή;

```
int isPerfectSquare(int num) {
  int root = sqrt(num);
  return root * root == num;
}
int low = atoi(argv[1]);
int high = atoi(argv[2]);
int i, sum = 0;
for(i = low ; i <= high ; i++) {
  if (isPerfectSquare(i))
    sum += i;
}
```

Και η δεύτερη;

```
int low = atoi(argv[1]);
int high = atoi(argv[2]);
int i, sum = 0;
for(i = sqrt(low) ; i <= sqrt(high) ; i++)
```

```
  sum += i*i ;
```

Υπόδειξη Και στις δύο, μετρήστε τις επαναλήψεις συναρτήσει του μεγέθους του διαστήματος (θεωρήστε την `sqrt` ένα βήμα). Η δεύτερη διατρέχει άλλες τιμές από την πρώτη: ποιες; Δοκιμάστε επίσης τη δεύτερη με `low = 5`, `high = 20` και συγκρίνετε με την πρώτη.

A15.14 · Πολυπλοκότητα της strcmp

A15.14 · *Διάλεξη 15, διαφάνεια 31* · ★★☆ · short-answer · `slides-lec15-complexity-strcmp`

Η συνάρτηση strcmp. Μια πιθανή υλοποίηση:

```
int strcmp(char * str1, char * str2) {
  while(*str1 && (*str1 == *str2)) {
    str1++;
    str2++;
  }
  return *str1 - *str2;
}
```

Τι πολυπλοκότητας είναι η συνάρτηση strcmp ως προς το μέγεθος των συμβολοσειρών (έστω n και m);

Υπόδειξη Βρείτε όλους τους λόγους για τους οποίους σταματά ο βρόχος. Ποια είναι η χειρότερη περίπτωση, και ποιο από τα δύο μήκη την περιορίζει;

A15.15 · Πολυπλοκότητα ελέγχου αν ένας αριθμός είναι πρώτος

A15.15 · *Διάλεξη 15, διαφάνεια 39* · ★★☆ · short-answer · `slides-lec15-prime-complexity`

Έλεγχος αν ένας αριθμός είναι πρώτος - τι χρονική πολυπλοκότητα έχει;

Υπόδειξη Γράψτε πρώτα τον απλό αλγόριθμο που δοκιμάζει διαιρέτες και μετρήστε τις δοκιμές για αριθμό n. Μετά σκεφτείτε: αν ο n έχει διαιρέτη μεγαλύτερο από $\sqrt{n}$, τι ισχύει για τον «συμπληρωματικό» του διαιρέτη; Πώς αλλάζει αυτό το όριο του βρόχου;

Εργασίες (A15.16–A15.17)

A15.16 · Κατοπτρικά Πρώτα Τετράγωνα

A15.16 · *Εργασία 1 (2023-24), Άσκηση 2* · ★★★ · programming · `hw-2023-hw1-mirror`

Σε αυτήν την άσκηση, καλούμαστε να εντοπίσουμε κάποιους αριθμούς που λέγονται *κατοπτρικά πρώτα τετράγωνα* σε ένα εύρος ακεραίων. Πρώτα όμως, πρέπει να βεβαιωθούμε ότι χρησιμοποιούμε κοινό λεξιλόγιο. Στους ακόλουθους ορισμούς, εξηγούμε τους βασικούς όρους.

Ορισμός 1. Ένας φυσικός αριθμός λέγεται *τέλειο τετράγωνο* (perfect square), όταν μπορεί να γραφεί σαν το τετράγωνο ενός άλλου φυσικού αριθμού. Για παράδειγμα τα 4 (= 2^2), 16 (= 4^2), 625 (= 25^2) είναι όλα τέλεια τετράγωνα. Αντίθετα οι αριθμοί 7, 8, 624 και ένα σωρό άλλοι δεν είναι.

Ορισμός 2. Ένας φυσικός αριθμός μεγαλύτερος του 1 ονομάζεται *πρώτος* (prime) όταν έχει σαν μόνους διαιρέτες (το υπόλοιπο της διαίρεσης είναι 0) το 1 και τον εαυτό του. Για παράδειγμα, το 17 είναι πρώτος αριθμός, ενώ το 42 δεν είναι, αφού έχει σαν διαιρέτες το 2, το 3 και το 7, εκτός από τους 1 και 42. Μπορείτε να επιβεβαιώσετε ότι οι πρώτοι αριθμοί που είναι μικρότεροι από το 100 είναι οι εξής:

```
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,
↪  79, 83, 89, 97
```

Ορισμός 3. Ένας φυσικός αριθμός λέγεται *παλινδρομικός* (palindromic) όταν διαβάζεται το ίδιο είτε ευθέως (από αριστερά προς τα δεξιά) είτε αντιστρόφως. Κάποια παραδείγματα παλινδρομικών αριθμών είναι τα ακόλουθα: 1, 9, 151, 484, 12321, 98766789.

Ορισμός 4. Ένα ζευγάρι αριθμών λέγεται *κατοπτρικό* (mirror) εάν τα ψηφία του πρώτου διαβάζοντάς τα από αριστερά προς τα δεξιά ταυτίζονται με τα ψηφία του δεύτερου όταν τα διαβάζουμε από τα δεξιά προς τα αριστερά και αντίστροφα. Για παράδειγμα, τα ακόλουθα ζεύγη αριθμών είναι κατοπτρικά: 1234 και 4321, 18437 και 73481, 837465 και 564738. Σε ένα κατοπτρικό ζεύγος αριθμών, λέμε ότι ο ένας είναι *κάτοπτρο* του άλλου.

Ορισμός 5. Ένας φυσικός αριθμός είναι ένα *κατοπτρικό πρώτο τετράγωνο* όταν:

1. Είναι τέλειο τετράγωνο ενός πρώτου αριθμού.
2. Το κάτοπτρό του είναι επίσης τέλειο τετράγωνο ενός πρώτου αριθμού.
3. Δεν είναι παλινδρομικός.

Για παράδειγμα, ο αριθμός 12769 είναι ένα κατοπτρικό πρώτο τετράγωνο, καθώς: (α) είναι το τετράγωνο ενός πρώτου αριθμού (113^2 = 12769), (β) το κάτοπτρό του 96721 είναι επίσης το τετράγωνο ενός πρώτου αριθμού (311^2 = 96721) και (γ) δεν είναι παλινδρομικός.

Για το ζητούμενο αυτής της άσκησης, καλείστε να γράψετε ένα πρόγραμμα το οποίο να υπολογίζει το άθροισμα όλων των κατοπτρικών πρώτων τετραγώνων που βρίσκονται σε ένα εύρος φυσικών αριθμών.

Τεχνικές Προδιαγραφές

- Repository Name: `progintro/hw1-<YourUsername>`
- C Filepath: `mirror/src/mirror.c`

- Το πρόγραμμά θα πρέπει να παίρνει δύο ορίσματα από την γραμμή εντολών στην μορφή `./mirror low high`, με το πρώτο (`low`) να είναι το κάτω όριο και το δεύτερο να είναι το άνω όριο (`high`). Τα δύο όρια είναι κλειστά, δηλαδή η αναζήτηση πρέπει να γίνει στο σύνολο `[low, high]`. Για οποιαδήποτε είσοδο δεν είναι μέσα στις προδιαγραφές το πρόγραμμα πρέπει να επιστρέφει με κωδικό εξόδου (exit code) 1. Εάν δοθεί άνω όριο χαμηλότερο του κάτω ορίου το πρόγραμμα πρέπει να τερματίσει με κωδικό εξόδου 1.
- Όλοι οι φυσικοί που θα δοθούν στο πρόγραμμά σας θα είναι στο εύρος: [1, 10^15]. Εάν δοθούν μη-θετικοί ακέραιοι ή ακέραιοι άνω του 10^15 το πρόγραμμά σας πρέπει να τερματίζει με κωδικό εξόδου 1. Οποιαδήποτε άλλη μη ακέραια είσοδος -- για παράδειγμα το string ”mpampis” -- είναι εκτός προδιαγραφών και δεν θα ελεγχθεί.
- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr): `gcc -O3 -Wall -Wextra -Werror -pedantic -o mirror mirror.c -lm`
- README Filepath: `mirror/README.md`
- **Ένα αρχείο που να περιέχει ένα στοιχείο εισόδου και ένα εξόδου *διαφορετικά* από αυτά της άσκησης. Συγκεκριμένα προτείνουμε να βάλετε έναν συνδυασμό που θεωρείται ότι είναι δύσκολος να γίνει σωστός κατά την υλοποίηση**.
  - input Filepath: `mirror/test/input`
  - output Filepath: `mirror/test/output`

  Για παράδειγμα, το περιεχόμενο του input αρχείου μπορεί να είναι: ”1 100000” και του αντίστοιχου output: ”110620”. Προσοχή: αυτό το παράδειγμα δεν θα γίνει δεκτό από την άσκηση επειδή αυτό το input-output ζευγάρι υπάρχει ήδη παρακάτω, επομένως πρέπει να διαλέξετε κάποιο άλλο.
- Πρέπει να ολοκληρώνει την εκτέλεση μέσα σε: 60 δευτερόλεπτα.
- **Δεν επιτρέπεται να γίνει χρήση πινάκων/δεικτών πέρα από το διάβασμα των ορισμάτων της** `main`.
- **Δεν επιτρέπεται χρήση προϋπολογισμένων αποτελεσμάτων**. Το πρόγραμμά σας πρέπει να υπολογίζει το αποτέλεσμα χωρίς ”πρότερη γνώση”, δηλαδή χωρίς να έχετε ήδη κωδικοποιήσει τα κατοπτρικά πρώτα τετράγωνα που έχετε βρει από προηγούμενους υπολογισμούς σας μέσα στον κώδικα.

Παρακάτω παραθέτουμε την αλληλεπίδραση με μια ενδεικτική λύση:

```
thanassis@linux14:~$ gcc -O3 -Wall -Wextra -Werror -pedantic -o mirror mirror.c
↪  -lm
thanassis@linux14:~$ ./mirror 1 100000
110620
```

```
thanassis@linux14:~$ ./mirror 1 10000000
15633754
thanassis@linux14:~$ ./mirror 1 1000000000000
53009475688
thanassis@linux14:~$ time ./mirror 1 1000000000000
53009475688

real    0m0,025s
user    0m0,023s
sys     0m0,000s
```

Κατά την διάρκεια της αποσφαλμάτωσης του προγράμματος, μπορούμε να επιβεβαιώσουμε τα επιμέρους αποτελέσματα τυπώνοντας τους αριθμούς που αποτελούν κάθε άθροισμα. Για παράδειγμα, το πρώτο αποτέλεσμα είναι 110620 = 169 + 961 + 12769 + 96721 - και αυτοί οι 4 αριθμοί είναι τα μόνα κατοπτρικά πρώτα τετράγωνα στο εύρος [1, 100000].

Στο αρχείο `README.md` πρέπει να προσθέσετε οποιεσδήποτε παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.

Υπόδειξη Μην διατρέχετε όλους τους αριθμούς του εύρους (μέχρι 10^15): διατρέξτε τους υποψήφιους πρώτους p με p^2 στο εύρος, δηλαδή μέχρι την τετραγωνική ρίζα του `high`. Χρειάζεστε μικρές συναρτήσεις για «είναι πρώτος», «κάτοπτρο αριθμού» (με `%` και `/`) και «είναι τέλειο τετράγωνο» (προσοχή στη στρογγυλοποίηση της `sqrt` σε μεγάλους αριθμούς). Οι τιμές και το άθροισμα δεν χωράνε σε `int`, και θυμηθείτε ότι δεν επιτρέπονται πίνακες.

Δείτε επίσης

- Υποδειγματική υποβολή φοιτητή (περιέχει λύση: δείτε την αφού λύσετε την άσκηση)

A15.17 · Παραγοντοποίηση ημιπρώτων (factor)

A15.17 · *Εργασία 1 (2024-25), Άσκηση 3 (Bonus)* · ★★★ · programming · `hw-2024-hw1-factor`

Παραγοντοποίηση (factor - Bonus 50 Μονάδες)

Αυτή η άσκηση είναι Bonus, δηλαδή η λύση της *δεν* είναι απαραίτητη για να πάρει κάποιος/α όλες τις μονάδες της Εργασίας 1. Οι μονάδες από την όποια υποβολή για αυτήν την άσκηση θα προστεθούν στον τελικό σας βαθμό του μαθήματος που περιλαμβάνει τις ασκήσεις.

Κάθε μη πρώτος (σύνθετος/composite) αριθμός μπορεί να γραφτεί σαν γινόμενο δύο ή περισσότερων πρώτων αριθμών. Επομένως η παραγοντοποίηση (factorization) ενός φυσικού αριθμού n είναι το πρόβλημα της εύρεσης των πρώτων παραγόντων που το γινόμενό τους μας δίνει το n. Για παράδειγμα, η παραγοντοποίηση του αριθμού 42 είναι $2 \cdot 3 \cdot 7$. Συγκεκριμένα, σε

αυτήν την άσκηση θα ασχοληθούμε με την παραγοντοποίηση μιας συγκεκριμένης κατηγορίας σύνθετων αριθμών, τους ημιπρώτους (semiprimes).

Ορισμός 6. Ένας φυσικός αριθμός λέγεται ημιπρώτος (semiprime), όταν είναι το γινόμενο ακριβώς δύο πρώτων αριθμών. Για παράδειγμα, ο αριθμός 46 είναι semiprime ($2 \cdot 23$) όπως και ο αριθμός 9 ($3 \cdot 3$). Αντίθετα, ο αριθμός 42 δεν είναι semiprime.

Το πρόβλημα φαίνεται απλό, αλλά κανείς δεν έχει βρει μέχρι στιγμής αποδοτική λύση, και σε αυτό στηρίζεται η ασφάλεια του RSA. Σε αυτήν την άσκηση λοιπόν, καλείστε να γράψετε ένα αποδοτικό πρόγραμμα το οποίο να παραγοντοποιεί ημιπρώτους (semiprimes).

Τεχνικές Προδιαγραφές

- Repository Name: progintro/hw1-<YourUsername>
- C Filepath: factor/src/factor.c
- Το πρόγραμμά θα πρέπει να παίρνει ένα όρισμα από την γραμμή εντολών στην μορφή `./factor semiprime`, με το πρώτο (`semiprime`) να είναι ο ημιπρώτος που θέλουμε να παραγοντοποιήσουμε. Αν το πρόγραμμα εκτελεστεί με ορίσματα που δεν ακολουθούν τις παραπάνω προδιαγραφές, πρέπει να εκτυπώσει αντίστοιχο μήνυμα όπως στα παρακάτω παραδείγματα και να επιστρέφει με κωδικό εξόδου (exit code) 1.
- Όλοι οι δεκαδικοί ακέραιοι που θα δοθούν στο πρόγραμμά σας θα είναι ημιπρώτοι και στο εύρος: $[0, 2^{127}]$. Για οποιαδήποτε άλλη είσοδο, το πρόγραμμά σας πρέπει να τερματίζει με κωδικό εξόδου 1.
- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr): `gcc -O3 -Wall -Wextra -Werror -o factor factor.c -lm`
- Μορφοποίηση: το αρχείο που θα υποβληθεί πρέπει να έχει μορφοποίηση σύμφωνη με το C/C++ στυλ της Google. Για να μορφοποιήσετε το αρχείο σας, μπορείτε να τρέξετε την ακόλουθη εντολή: `clang-format -i -style=Google factor.c` σε έναν υπολογιστή εργαστηρίου. Μη μορφοποιημένα προγράμματα δεν θα εξεταστούν.
- README Filepath: factor/README.md
- **Ένα αρχείο που να περιέχει ένα στοιχείο εισόδου και ένα εξόδου *διαφορετικά* από αυτά της εκφώνησης. Συγκεκριμένα προτείνουμε να βάλετε έναν συνδυασμό που θεωρείται ότι είναι δύσκολος να γίνει σωστός κατά την υλοποίηση**.
  - input Filepath: `factor/test/input.txt`
  - output Filepath: `factor/test/output.txt`

Για παράδειγμα, το περιεχόμενο του input.txt αρχείου μπορεί να είναι: "93" και του αντίστοιχου output.txt: "3 31". Προσοχή: αυτό το παράδειγμα δεν θα γίνει δεκτό από την άσκηση επειδή αυτό το input-output ζευγάρι υπάρχει ήδη παρακάτω, επομένως πρέπει να διαλέξετε κάποιο άλλο.

- Πρέπει να ολοκληρώνει την εκτέλεση μέσα σε: 10 δευτερόλεπτα.
- **Δεν επιτρέπεται χρήση προϋπολογισμένων αποτελεσμάτων**. Το πρόγραμμά σας πρέπει να υπολογίζει το αποτέλεσμα χωρίς "πρότερη γνώση", δηλαδή χωρίς να έχετε ήδη κωδικοποιήσει παραγοντοποιήσεις ημιπρώτων που έχετε βρει από προηγούμενους υπολογισμούς σας μέσα στον κώδικα.
- Καθώς αυτή η εργασία είναι Bonus, θα ελέγξουμε και ποιοτικά χαρακτηριστικά της υποβολής. Μια ιδιαίτερα δυσνόητη ή μη διαχειρίσιμη λύση (π.χ., 6 nested if-else, 35 μεταβλητές στην ίδια συνάρτηση κτλ) θα οδηγήσει σε αφαίρεση μονάδων κατά την κρίση του εξεταστή.

Παρακάτω παραθέτουμε την αλληλεπίδραση με μια ενδεικτική λύση:

```
$ ./factor
Usage: ./factor <semiprime>
$ echo $?
1
$ ./factor 93
Factors: 3 31
$ echo $?
0
$ ./factor 9827348119
Factors: 613 16031563
$ # level: very hard
$ ./factor 2524891914334062643
Factors: 1175747593 2147477851
$ # level: extremely hard
$ ./factor 809724910412139638697047
Factors: 783108713587 1033987869581
$ # level: this is impossible
$ ./factor 661621452399004520128701898758039 61
Factors: 206547667773749927 320323855277677343
```

Στο αρχείο README.md πρέπει να προσθέσετε οποιεσδήποτε παρατηρήσεις κάνατε κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης. Οι 10 γρηγορότερες λύσεις θα μοιραστούν ένα bonus (extra) 500 μονάδων κατανεμημένων αναλογικά με τον παράγοντα $\frac{1}{T}$ (μέχρι το max: 200 μονάδες ανά υποβολή), όπου T είναι ο συνολικός χρόνος που απαιτεί ο αλγόριθμος της υλοποίησης για να παραγοντοποιήσει όλους τους ακεραίους που του δόθηκαν. Αν δούμε μια ιδιαίτερα ενδιαφέρουσα/αποδοτική λύση, θα ζητήσουμε μια παρουσίαση από το παιδί που την

υλοποίησε.

Υπόδειξη Αριθμοί έως 2^{127} δεν χωρούν σε `long long`: χρειάζεστε ακέραιο 128 bits (π.χ. την επέκταση `__int128` του gcc) και δική σας μετατροπή από/προς δεκαδικό string, αφού η `printf`/`strtoll` δεν τον υποστηρίζουν. Η δοκιμαστική διαίρεση έως $\sqrt{n}$ αρκεί για τα πρώτα παραδείγματα αλλά όχι για τα «very hard» και πάνω· ψάξτε πιθανοτικούς αλγορίθμους όπως τον Pollard's rho σε συνδυασμό με έλεγχο πρώτων Miller-Rabin, και προσέξτε την υπερχείλιση στον πολλαπλασιασμό modulo.

Θέματα εξετάσεων (A15.18)

A15.18 · Δίδυμοι Πρώτοι

A15.18 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #7 (Star Wars Themed), Θέμα 3* · ★★★ · programming · `exam-2023-fall-ex7-q3`

Πρόγραμμα: `leia.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που παίρνει ένα εύρος θετικών ακεραίων ως ορίσματα από την κονσόλα (το εύρος είναι κλειστό, δηλαδή περιέχει τα όρια) και τυπώνει πόσα ζεύγη *δίδυμων πρώτων* (twin primes) υπάρχουν σε αυτό το διάστημα. Ένα ζεύγος ακεραίων (n, n+2) είναι δίδυμοι πρώτοι αν και ο n και ο n+2 είναι πρώτοι. Για παράδειγμα, στο διάστημα [1, 10] υπάρχουν μόλις δύο ζεύγη δίδυμων πρώτων (3, 5) και (5, 7). Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o leia leia.c
$ ./leia 1 10
2
$ ./leia 10 100
6
$ ./leia 1 10000000
58980
$ ./leia 100000000000 100000100000
182
```

Υπόδειξη Ένας έλεγχος πρώτου με δοκιμαστική διαίρεση για κάθε αριθμό του [1, 10^7] είναι αργός· το κόσκινο του Ερατοσθένη σε δυναμικά δεσμευμένο πίνακα είναι η κλασική λύση. Για διαστήματα όπως το [10^11, 10^11 + 10^5] δεν χωράει κόσκινο μέχρι το πάνω όριο: σκεφτείτε ένα «τμηματικό» κόσκινο, που χρησιμοποιεί τους πρώτους μέχρι τη ρίζα του πάνω ορίου για να σημαδέψει μόνο το ζητούμενο διάστημα. Προσέξτε ότι και τα δύο μέλη του ζεύγους πρέπει να είναι μέσα στο διάστημα και ότι οι αριθμοί θέλουν 64 bits.

Κεφάλαιο 16: Επίλυση Προβλημάτων #2

Ζέσταμα: από τις διαφάνειες (A16.1–A16.15)

A16.1 · Μέσος όρος πίνακα και πολυπλοκότητα

A16.1 · *Διάλεξη 16, διαφάνειες 8–9* · ★☆☆ · programming · `slides-lec16-average-complexity`

Θέλω μια συνάρτηση που να δέχεται έναν πίνακα 100 ακεραίων και να επιστρέφει τον μέσο όρο. Πώς; Χρονική και χωρική πολυπλοκότητα;

Υπόδειξη Ένας συσσωρευτής και ένα πέρασμα από τον πίνακα αρκούν. Για την πολυπλοκότητα, μετρήστε πόσες φορές εκτελείται το σώμα του βρόχου αν ο πίνακας είχε n στοιχεία, και πόσες βοηθητικές μεταβλητές χρειάζεστε. Προσέξτε αν η διαίρεση στο τέλος είναι ακέραια.

A16.2 · Αναζήτηση σε πίνακα και πολυπλοκότητα

A16.2 · *Διάλεξη 16, διαφάνειες 10 και 15* · ★☆☆ · programming · `slides-lec16-find-complexity`

Θέλω μια συνάρτηση που να δέχεται έναν πίνακα 100 ακεραίων και έναν ακέραιο και να γυρνάει τη θέση του στοιχείου αν το βρήκε ή -1. Πώς; Χρονική και χωρική πολυπλοκότητα;

Υπόδειξη Ελέγξτε τα στοιχεία ένα-ένα και επιστρέψτε μόλις βρείτε αυτό που ψάχνετε. Για την πολυπλοκότητα, σκεφτείτε ποια είσοδος κάνει τη συνάρτηση να δουλέψει περισσότερο.

A16.3 · Η συνάρτηση get_two_chars

A16.3 · *Διάλεξη 16, διαφάνεια 20* · ★☆☆ · programming · `slides-lec16-get-two-chars`

Θέλω να γράψω μια συνάρτηση `get_two_chars` που να διαβάζει *δύο* χαρακτήρες και να τους επιστρέφει. Πώς;

Υπόδειξη Το `return` επιστρέφει μόνο μία τιμή. Πώς μπορεί μια συνάρτηση να γράψει σε μεταβλητές του καλούντος; Σκεφτείτε τι θα επιστρέφει το ίδιο το `return` και τι γίνεται όταν η είσοδος τελειώσει.

A16.4 · Παλινδρομικός αριθμός

A16.4 · *Διάλεξη 16, διαφάνεια 21* · ★☆☆ · programming · `slides-lec16-palindrome-number`

Θέλω να βρω αν ένας αριθμός είναι παλινδρομικός. Πώς μπορώ να το βρω;

Υπόδειξη Ένας αριθμός είναι παλινδρομικός αν διαβάζεται ίδια ανάποδα. Έχετε ήδη τρόπο να αντιστρέψετε τα ψηφία ενός αριθμού· εναλλακτικά, βάλτε τα ψηφία σε πίνακα και συγκρίνετέ τα από τα δύο άκρα. Αποφασίστε τι ισχύει για αρνητικούς.

A16.5 · Αντιστροφή ψηφίων αριθμού

A16.5 · *Διάλεξη 16, διαφάνεια 13* · ★☆☆ · programming · `slides-lec16-reverse-digits`

Θέλω μια συνάρτηση που να αντιστρέφει τα ψηφία ενός αριθμού, για παράδειγμα αν δοθεί το 123, θέλω να επιστρέφει το 321. Πώς;

Υπόδειξη Ποια πράξη σας δίνει το τελευταίο ψηφίο ενός αριθμού και ποια τον αριθμό χωρίς αυτό; Χτίστε το αποτέλεσμα ψηφίο-ψηφίο, «σπρώχνοντας» ό,τι έχετε ήδη μία θέση αριστερά. Σκεφτείτε τι γίνεται με το 1200 και με πολύ μεγάλους αριθμούς.

A16.6 · yes αν ένα στοιχείο υπάρχει σε δισδιάστατο πίνακα

A16.6 · *Διάλεξη 16, διαφάνεια 24* · ★☆☆ · programming · `slides-lec16-search-2d`

Θέλω να τυπώσω ”yes” αν ένα στοιχείο βρίσκεται σε έναν δισδιάστατο πίνακα. Πώς;

Υπόδειξη Δύο εμφωλευμένοι βρόχοι διατρέχουν τον πίνακα. Η δυσκολία είναι να σταματήσετε μόλις το βρείτε: από πόσους βρόχους βγάζει ένα `break`; Σκεφτείτε μια βοηθητική συνάρτηση ή μια μεταβλητή-σημαία, ώστε το ”yes” να τυπωθεί ακριβώς μία φορά.

A16.7 · Η συνάρτηση swap

A16.7 · *Διάλεξη 16, διαφάνειες 22–23* · ★☆☆ · programming · `slides-lec16-swap`

Γράψτε μια συνάρτηση `swap` που ανταλλάζει δύο ακεραίους.

```c
#include <stdio.h>
int main() {
  int a = 100, b = 200;
  printf("%d %d\n", a, b);
  swap( ... );
  printf("%d %d\n", a, b);
  return 0;
}
```

Υπόδειξη Τα ορίσματα περνούν κατά τιμή, οπότε μια `swap(a, b)` θα άλλαζε μόνο αντίγραφα. Τι πρέπει να περάσει η `main` ώστε η `swap` να μπορεί να γράψει στις δικές της `a` και `b`; Θα χρειαστείτε και μια προσωρινή μεταβλητή.

A16.8 · Γιατί εξάσκηση στην επίλυση προβλημάτων;

A16.8 · *Διάλεξη 16, διαφάνειες 5–6* · ★☆☆ · short-answer · `slides-lec16-why-practice`

Γιατί κάνουμε εξάσκηση στην επίλυση προβλημάτων;

Υπόδειξη Σκεφτείτε τι κερδίζετε από κάθε πρόβλημα που λύνετε, πέρα από τη συγκεκριμένη λύση: τι μένει για το επόμενο πρόβλημα, πού θα ξανασυναντήσετε τα ίδια μοτίβα, και τι σημαίνει να συνδυάζετε γνωστές ιδέες σε κάτι νέο.

A16.9 · Η atoi και τι μπορεί να πάει στραβά

A16.9 · *Διάλεξη 16, διαφάνειες 16–17* · ★★☆ · programming · `slides-lec16-atoi`

Θέλω μια συνάρτηση `atoi` που να παίρνει ένα πίνακα χαρακτήρων (μόνο ψηφία) και να επιστρέφει έναν ακέραιο. Πώς;

Η λύση των διαφανειών είναι η εξής. Τι μπορεί να πάει στραβά με αυτήν τη συνάρτηση;

```
int atoi(char digits[]) {
  int result = 0;
  for(int i = 0; digits[i]; i++) {
    result = 10 * result + digits[i] - '0';
  }
  return result;
}
```

Υπόδειξη Γράψτε τις υποθέσεις που κάνει η συνάρτηση για την είσοδό της και δοκιμάστε νοερά εισόδους που τις παραβιάζουν: πρόσημο ή γράμματα, έναν πολύ μεγάλο αριθμό, έναν πίνακα χωρίς `'\0'`. Σκεφτείτε και το όνομά της.

A16.10 · char *array[], char **array και char array[10][10]

A16.10 · *Διάλεξη 16, διαφάνεια 19* · ★★☆ · short-answer · `slides-lec16-char-pointer-arrays`

Ποια η διαφορά ενός `char * array[]` και ενός `char ** array`; Έχει διαφορά από έναν `char array[10][10]`;

Υπόδειξη Σχεδιάστε τη μνήμη για καθεμία: πόσα bytes πιάνει (τι δίνει το `sizeof`), πού υπάρχουν αποθηκευμένοι pointers και πού σκέτοι χαρακτήρες. Μετά σκεφτείτε σε ποιον τύπο μετατρέπεται το καθένα όταν περνά ως όρισμα σε συνάρτηση.

A16.11 · Αποδοτική αναδρομική Fibonacci

A16.11 · *Διάλεξη 16, διαφάνεια 26* · ★★☆ · programming · `slides-lec16-fibonacci-efficient`

Θέλω μια αναδρομική υλοποίηση που να υπολογίζει τους fibonacci αριθμούς αποδοτικά. Γίνεται;

Υπόδειξη Σχεδιάστε το δέντρο κλήσεων της απλής αναδρομής για ένα μικρό `n` και μετρήστε πόσες φορές υπολογίζεται η ίδια τιμή. Τι θα γινόταν αν θυμόσασταν τα αποτελέσματα που έχετε ήδη βρει; Εναλλακτικά, τι θα έπρεπε να «κουβαλά» κάθε κλήση ώστε να αρκεί μία αναδρομική κλήση;

A16.12 · Το στοιχείο που υπάρχει δύο φορές

A16.12 · *Διάλεξη 16, διαφάνεια 11* · ★★☆ · programming · `slides-lec16-find-duplicate`

Θέλουμε μια συνάρτηση που παίρνει ως όρισμα έναν πίνακα ακεραίων και βρίσκει το στοιχείο που υπάρχει δύο φορές - όλα τα υπόλοιπα στοιχεία θα υπάρχουν ακριβώς μία φορά. Για παράδειγμα, αν δοθεί ο πίνακας {8, 1, 5, 42, 7, 3, 42} θα πρέπει να επιστρέψει την τιμή 42.

Υπόδειξη Η απλή λύση συγκρίνει κάθε ζεύγος στοιχείων· προσέξτε να μη συγκρίνετε ένα στοιχείο με τον εαυτό του. Μετά σκεφτείτε πώς θα γινόταν ταχύτερα: τι αλλάζει αν ο πίνακας είναι ταξινομημένος, ή αν οι τιμές έχουν μικρό εύρος και μπορείτε να ξοδέψετε μνήμη;

A16.13 · Δισδιάστατος πίνακας στον σωρό

A16.13 · *Διάλεξη 16, διαφάνεια 25* · ★★☆ · programming · `slides-lec16-heap-2d-array`

Θέλω να δημιουργήσω έναν δισδιάστατο πίνακα στον σωρό. Πώς μπορώ να το κάνω αυτό;

Υπόδειξη Δύο δρόμοι: ένας πίνακας από pointers γραμμών, όπου κάθε γραμμή έχει τη δική της `malloc`, ή ένα ενιαίο μπλοκ για όλα τα στοιχεία, όπου υπολογίζετε μόνοι σας τη θέση του στοιχείου `[i][j]`. Μην ξεχάσετε τον έλεγχο για `NULL` και τη σωστή σειρά των `free`.

A16.14 · Το στοιχείο χωρίς ζευγάρι

A16.14 · *Διάλεξη 16, διαφάνεια 14* · ★★☆ · programming · `slides-lec16-single-unpaired`

Έχω έναν πίνακα ακεραίων και όλα τα στοιχεία του είναι διπλά εκτός από ένα. Πώς το βρίσκω αποδοτικά;

Υπόδειξη Υπάρχει τελεστής bit για τον οποίο ένας αριθμός «ακυρώνει» τον εαυτό του και το 0 είναι ουδέτερο στοιχείο. Στοχεύστε σε ένα πέρασμα από τον πίνακα, χωρίς βοηθητικό πίνακα και χωρίς ταξινόμηση.

A16.15 · Άθροισμα τέλειων τετραγώνων σε διάστημα

A16.15 · *Διάλεξη 16, διαφάνεια 18* · ★★☆ · programming · `slides-lec16-sum-perfect-squares`

Θέλω να βρω το άθροισμα των τέλειων τετραγώνων μέσα σε ένα διάστημα αριθμών. Πώς το βρίσκω αποδοτικά;

Υπόδειξη Αντί να ελέγχετε κάθε αριθμό του διαστήματος, σκεφτείτε πόσα τέλεια τετράγωνα υπάρχουν μέχρι το b και απαριθμήστε απευθείας αυτά. Υπάρχει και κλειστός τύπος για το $1^2 + 2^2 + \cdots + m^2$. Προσέξτε την υπερχείλιση.

Εργαστήριο (A16.16–A16.17)

A16.16 · Σκαλί-σκαλί (Παλιό θέμα, Προαιρετικό)

A16.16 · *Εργαστήριο 5, Άσκηση 3* · ★★★ · programming · `lab-lab05-ladder`

Ένα παιδί ανεβαίνει μια σκάλα και σε κάθε βήμα του ανεβαίνει 1, 2 ή 3 σκαλιά. Έστω ότι η σκάλα έχει 5 σκαλιά. Ένας τρόπος να τα ανέβει είναι 1-1-1-1-1, δηλαδή ένα σκαλί την φορά. Ένας άλλος είναι ο 2-2-1 (δύο σκαλιά στα πρώτα δύο βήματα και μετά ένα σκαλί). Άλλοι πιθανοί τρόποι είναι οι 3-2, 2-3, 3-1-1, κ.ο.κ - σε κάθε περίπτωση ο συνολικός αριθμός των σκαλιών που ανέβηκε πρέπει να ισούται με τον αριθμό των σκαλιών της σκάλας. Γράψτε ένα πρόγραμμα `ladder.c` το οποίο διαβάζει τον συνολικό αριθμό των σκαλοπατιών και επιστρέφει τον αριθμό των διαφορετικών τρόπων με τους οποίους το παιδί μπορεί να ανέβει την σκάλα. Το πρόγραμμά σας θέλουμε να βγάζει σωστά αποτελέσματα για σκάλες μέχρι 50 σκαλιά. Λάβετε υπόψη σας ότι στην C δεν μπορούν να αναπαρασταθούν ακέραιοι μεγαλύτεροι από το $2^{64} - 1$, δεδομένου ότι ο ευρύτερος τύπος ακεραίου που μπορούμε να έχουμε είναι ο `unsigned long long` (των 8 bytes). Ακολουθούν ενδεικτικές εκτελέσεις:

```
$ ./ladder
Please provide the number of steps: 4
```

```
There are 7 different ways to climb the ladder
$ ./ladder
Please provide the number of steps: 5
There are 13 different ways to climb the ladder
$ ./ladder
Please provide the number of steps: 6
There are 24 different ways to climb the ladder
$ ./ladder
Please provide the number of steps: 50
There are 10562230626642 different ways to climb the ladder
```

Υπόδειξη Σκεφτείτε το τελευταίο βήμα του παιδιού: ήταν 1, 2 ή 3 σκαλιά, οπότε οι τρόποι για n σκαλιά προκύπτουν από τους τρόπους για λιγότερα σκαλιά (όπως στο Fibonacci, αλλά με τρεις όρους). Προσέξτε τις βασικές περιπτώσεις για μικρά n. Η αφελής αναδρομή δεν θα τελειώσει ποτέ για 50 σκαλιά, και το αποτέλεσμα δεν χωράει σε `int`.

A16.17 · Χτίζοντας έναν χιονάνθρωπο (Παλιό θέμα)

A16.17 · *Εργαστήριο 7, Άσκηση 4* · ★★★ · programming · `lab-lab07-olaf`

Βρισκόμαστε σε έναν χιονισμένο τετράγωνο κήπο διαστάσεων NxN και θέλουμε να φτιάξουμε έναν χιονάνθρωπο. Δυστυχώς το χιόνι είναι μαζεμένο σε συγκεκριμένους σωρούς μέσα στον κήπο (δεν είναι παντού). Γράψτε ένα πρόγραμμα που αποφασίζει πού πρέπει να φτιάξουμε τον χιονάνθρωπό μας ώστε να κάνουμε τον ελάχιστο δυνατό κόπο. Για λόγους απλοποίησης θεωρούμε ότι όλες οι συντεταγμένες στον κήπο είναι ακέραιες και ότι οι κινήσεις μας μπορούν να είναι μόνο πάνω-κάτω-αριστερά-δεξιά (όχι διαγώνια). Το μέγεθος του κήπου και οι τοποθεσίες των σωρών χιονιού δίνονται στο πρόγραμμα μέσω αρχείου το οποίο δίνουμε με ανακατεύθυνση στην πρότυπη είσοδο του προγράμματος. Το αρχείο θα περιέχει την διάσταση του κήπου (N) ακολουθούμενη από τις συντεταγμένες του κάθε σωρού. Παράδειγμα επιτυχούς εκτέλεσης ακολουθεί:

```
$ gcc -o olaf olaf.c
$ cat map.txt
10
1 1
6 7
2 8
7 2
$ ./olaf < map.txt
We will position the snowman on (4, 4) with a minimum cost of 22.
```

Στην οπτικοποίηση της επιλογής μας ο χιονάνθρωπος βρίσκεται στην θέση ”καρότο” ενώ οι σωροί με τα χιόνια απεικονίζονται με τους κύκλους. Από τους δύο πάνω σωρούς παρατηρούμε

ότι χρειαζόμαστε 6 βήματα για να φτάσουμε τον χιονάνθρωπο ενώ από τους δύο κάτω χρειαζόμαστε 5 - σύνολο (6 + 6 + 5 + 5 =) 22. Παρατηρήστε ότι μπορεί να υπάρχουν περισσότερες από μία βέλτιστες τοποθετήσεις, χρειάζεται να βρούμε μόνο μία από αυτές. Για λόγους απλότητας ο χιονάνθρωπος μπορεί να τοποθετηθεί πάνω σε έναν σωρό (και τα βήματα που απαιτούνται σε αυτήν την περίπτωση είναι 0).

Υπόδειξη Η απόσταση με κινήσεις μόνο πάνω-κάτω-αριστερά-δεξιά είναι η απόσταση Manhattan, $|x_1 - x_2| + |y_1 - y_2|$. Το πλήθος των σωρών δεν είναι γνωστό από πριν, οπότε διαβάστε τις συντεταγμένες μέχρι το τέλος της εισόδου σε πίνακα που μεγαλώνει δυναμικά. Μια εξαντλητική λύση δοκιμάζει κάθε θέση του κήπου και κρατά την καλύτερη. Για μεγάλο N, σκεφτείτε ότι οι δύο συντεταγμένες βελτιστοποιούνται ανεξάρτητα η μία από την άλλη.

Εργασίες (A16.18)

A16.18 · Ο Γρίφος του Στέργιου

A16.18 · *Bonus #0 (2025-26, προαιρετική)* · ★★★ · programming · `hw-2025-bonus0-stergios`

Ο Στέργιος μόλις έμαθε πρόσθεση και αφαίρεση στο σχολείο και βρήκε ένα καινούριο παιχνίδι: διαλέγει στην τύχη τρεις αριθμούς, π.χ., το $(1, 2, 6)$ και σε κάθε βήμα μπορεί να αντικαταστήσει έναν από αυτούς (όποιον θέλει—έστω το 2) με το άθροισμα των άλλων $(1 + 6)$ δύο φορές μείον τον εαυτό του $(1+6+1+6-2 = 12)$ και παίρνει μια καινούρια τριάδα $(1, 12, 6)$. Αναρωτιέται αν κάνοντας τέτοια βήματα / επιλογές επανειλημμένα θα καταφέρει να μηδενίσει έναν από τους αριθμούς. Πέρα από αυτό, αναρωτιέται αν μπορεί να βρει την συντομότερη σειρά επιλογών προκειμένου να μηδενίσει έναν εξ αυτών. Με δοκιμές παρατήρησε ότι για τους αριθμούς $(1, 2, 6)$ η καλύτερη επιλογή είναι να διαλέξει το 6 πρώτα καθώς σε μόνο ένα βήμα θα πάρει την τριάδα $(1, 2, 0)$.

Έστω `stergios` η μαγική συνάρτηση που δίνει τον *ελάχιστο αριθμό βημάτων* προκειμένου να μηδενιστεί ένας από τους αριθμούς της αρχικής τριάδας. Για παράδειγμα, $stergios(0, 7, 9) = 0$, $stergios(1, 2, 6) = 1$, $stergios(1000, 216, 102) = 3$ (πως;). Αν *δεν* υπάρχει αριθμός βημάτων που να την μηδενίζει, τότε η συνάρτηση `stergios` πάλι επιστρέφει μηδέν. Για παράδειγμα, $stergios(74, 214, 540) = 0$ (ο Στέργιος δεν βρήκε κάποιον τρόπο να φτάσει στο μηδέν με αυτούς τους τρεις αριθμούς όσο και αν προσπάθησε). Είναι εφικτό να υπολογίσουμε αθροίσματα της συνάρτησης `stergios` για όλους τους αριθμούς αν κάποιες από τις αρχικές μας επιλογές είναι μεγάλοι ακέραιοι; Συγκεκριμένα, ο Στέργιος αναρωτιέται αν μπορούμε να υπολογίσουμε το άθροισμα:

$$Solution = \sum_{i=0}^{\infty} stergios(i, 100000000000000000, 3656158440062976)$$

Ο μπαμπάς του, ο Μάκης, προσπάθησε, πρώτα μόνος και μετά με βοήθεια, αλλά δεν μπόρεσε

να δώσει μια ικανοποιητική απάντηση. Μήπως μπορείτε εσείς; Τα τελευταία βράδια ο Μάκης βλέπει διαρκώς στο όνειρό του ότι $\sum_{i=0}^{\infty} stergios(1000, 216, i) = 1051$ και επίσης πως $stergios(1000, 216, 109218) = 108$ αλλά δεν έχει καμία βεβαιότητα. Το μόνο που ξέρει είναι πως η τιμή του $Solution$ τυχαίνει να είναι και το password του χρήστη `impossible0` στο σύστημα `35.169.96.19` της Εργασίας 0.

Η πρώτη σωστή λύση στον παραπάνω γρίφο θα οδηγήσει σε Bonus μέχρι 360 μονάδες. Οι επόμενες λύσεις θα λάβουν επίσης bonus αναλογικά (η 2η μέχρι 360/2=180 μονάδες, η 3η μέχρι 360/3=120 μονάδες κοκ). Το πρόβλημα θα κλείσει μόλις (εάν) βρεθούν 10 λύσεις ή τελειώσει το εξάμηνο—ότι από τα δύο συμβεί πιο γρήγορα.

Υπόδειξη Ξεκίνα με ένα πρόγραμμα ωμής βίας (αναζήτηση κατά πλάτος στις τριάδες) για μικρούς αριθμούς και επιβεβαίωσε τα παραδείγματα και τις «ονειρικές» τιμές του Μάκη· μετά ψάξε δομή στα αποτελέσματα. Παρατήρησε ότι το βήμα $x \to 2(y+z) - x$ είναι «ανάκλαση» και ότι κάνοντάς το δύο φορές στην ίδια θέση επιστρέφεις εκεί που ήσουν· σκέψου ποια ποσότητα των τριών αριθμών μένει αναλλοίωτη και ποιο βήμα μικραίνει την τριάδα, ώστε να μην εξερευνάς τυφλά. Το άθροισμα είναι άπειρο αλλά μόνο πεπερασμένα i δίνουν μη μηδενικό αποτέλεσμα, και οι τιμές ξεπερνούν τα 64 bits, άρα χρειάζεσαι προσοχή με τους τύπους (π.χ. `__int128` ή δική σου αριθμητική).

Θέματα εξετάσεων (A16.19–A16.26)

A16.19 · Εαυτοί Αριθμοί

A16.19 · *Κατατακτήριες Δεκεμβρίου 2023, Θέμα 2* · ★★☆ · programming · `exam-2023-dec-q2`

Στην θεωρία αριθμών, ένας φυσικός αριθμός $ λέγεται *εαυτός* (self) όταν δεν υπάρχει κάποιος φυσικός αριθμός $, έτσι ώστε το $ να ισούται με το άθροισμα του $ και των ψηφίων του $ (με βάση το 10 σε αυτό το θέμα). Έστω (n)$ η συνάρτηση που υπολογίζει το άθροισμα ενός φυσικού $ και των ψηφίων του. Τότε ένας αριθμός $ είναι εαυτός αν και μόνο αν $\nexists m.F(m) = n$. Για παράδειγμα, (15) = 15 + 1 + 5 = 21$ και επομένως ο αριθμός 21 δεν είναι εαυτός. Αντίθετα, ο αριθμός 20 είναι εαυτός καθώς δεν υπάρχει φυσικός αριθμός $ έτσι ώστε (m) = 20$. Υπάρχουν μόλις 13 εαυτοί αριθμοί μικρότεροι του 100:

$$1, 3, 5, 7, 9, 20, 31, 42, 53, 64, 75, 86, 97$$

Γράψτε ένα πρόγραμμα C το οποίο βρίσκει και τυπώνει όλους τους εαυτούς αριθμούς σε ένα εύρος φυσικών αριθμών 0$ (το εύρος είναι κλειστό, δηλαδή τα άκρα συμπεριλαμβάνονται), όπου οι αριθμοί δίνονται από την γραμμή εντολών. Ακολουθεί παράδειγμα εκτέλεσης για να βρούμε όλους τους εαυτούς αριθμούς στο διάστημα 10000$:

```
$ ./self 9900 10000
```

```
Self numbers: 9903 9914 9925 9927 9938 9949 9960 9971 9982 9993
Found 10 total
```

Υπόδειξη Γράψτε πρώτα μια συνάρτηση για το (m)$, με `% 10` και `/ 10` σε βρόχο. Για κάθε $ του εύρους δεν χρειάζεται να δοκιμάσετε όλα τα : $\alpha\varphi o(m) \geq m$ και το άθροισμα ψηφίων ενός αριθμού είναι μικρό (το πολύ 9 ανά ψηφίο), αρκεί να ελέγξετε λίγα $ ακριβώς κάτω από το $. Προσέξτε ότι τα όρια δίνονται ως συμβολοσειρές στο `argv` και ότι πρέπει να μετρήσετε πόσοι βρέθηκαν.

A16.20 · Αγαπήσιμοι Αριθμοί

A16.20 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #0 (Valentine's Themed), Θέμα 4* · ★★☆ · programming · `exam-2023-fall-ex0-q4`

Πρόγραμμα: `lovable.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που διαβάζει ένα εύρος αριθμών από την κονσόλα (το εύρος είναι κλειστό, δηλαδή περιέχει τα όρια) και τυπώνει όλους τους αγαπήσιμους (lovable) αριθμούς. Ένας αριθμός είναι lovable αν είναι τέλειος κύβος (δηλαδή ένας άλλος ακέραιος υψωμένος στην 3η δύναμη) και ταυτόχρονα παλινδρομικός (δηλαδή διαβάζεται το ίδιο από αριστερά προς τα δεξιά και από δεξιά προς τα αριστερά). Ένα παράδειγμα εκτέλεσης ακολουθεί:

```
$ gcc -o lovable lovable.c -lm
$ ./lovable 100000 100000000
1030301
1367631
```

Ο 1030301 είναι lovable καθώς είναι παλινδρομικός και ταυτόχρονα τέλειος κύβος (= 101^3). Το πρόγραμμά σας πρέπει να βγάζει σωστά αποτελέσματα για όρια μέχρι 10^18.

Υπόδειξη Μέχρι το 10^18 δεν μπορείτε να ελέγξετε κάθε αριθμό: απαριθμήστε αντί γι' αυτό τους κύβους `i*i*i`, που είναι μόνο περίπου ένα εκατομμύριο, και ελέγξτε αν είναι παλινδρομικοί αντιστρέφοντας τα ψηφία τους. Χρησιμοποιήστε 64-bit ακέραιους (`long long` / `unsigned long long`) και προσέξτε την ακρίβεια αν υπολογίσετε την κυβική ρίζα του κάτω ορίου με `cbrt`.

A16.21 · Clyde Πρώτοι

A16.21 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #11, Θέμα 4* · ★★☆ · programming · `exam-2023-fall-ex11-q4`

Πρόγραμμα: `clyde.c`

Ένας ακέραιος λέγεται πρώτος όταν διαιρείται μόνο με τον εαυτό του και την μονάδα. Ένας πρώτος λέγεται clyde πρώτος όταν υπάρχει μια περιστροφή των ψηφίων του (με βάση το 10) που είναι επίσης πρώτος. Γράψτε ένα πρόγραμμα που παίρνει έναν ακέραιο από την κονσόλα και αποφαίνεται αν είναι Clyde πρώτος ή όχι. Παραδείγματα εκτέλεσης:

```
$ gcc -o clyde clyde.c -lm
$ ./clyde 13
13 is a clyde prime because 31 is prime too
$ ./clyde 99923
99923 is a clyde prime because 92399 is prime too
$ ./clyde 41
41 is NOT a clyde prime
$ ./clyde 99924
99924 is NOT a clyde prime
$ ./clyde 90000520793
90000520793 is a clyde prime because 93900005207 is prime too
```

Υπόδειξη Γράψτε μια συνάρτηση ελέγχου πρώτου που δοκιμάζει διαιρέτες ως τη ρίζα του αριθμού, με `long long` αφού τα παραδείγματα ξεπερνούν το `int`. Οι περιστροφές παράγονται αριθμητικά με `/` και `%` δυνάμεων του 10· σκεφτείτε τι γίνεται όταν μια περιστροφή ξεκινά με 0 και αν μετράει ο ίδιος ο αριθμός.

A16.22 · Εαυτοί Αριθμοί

A16.22 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #12, Θέμα 4* · ★★☆ · programming · `exam-2023-fall-ex12-q4`

Πρόγραμμα: `self.c`

Στην θεωρία αριθμών, ένας φυσικός αριθμός n λέγεται εαυτός (self) όταν δεν υπάρχει κάποιος φυσικός αριθμός m, έτσι ώστε το n να ισούται με το άθροισμα του m και των ψηφίων του m (με βάση το 10 σε αυτό το θέμα). Έστω $F(n)$ η συνάρτηση που υπολογίζει το άθροισμα ενός φυσικού n και των ψηφίων του. Τότε ένας αριθμός n είναι εαυτός αν και μόνο αν $\nexists m.F(m) = n$. Για παράδειγμα, $F(15) = 15 + 1 + 5 = 21$ και επομένως ο αριθμός 21 δεν είναι εαυτός. Αντίθετα, ο αριθμός 20 είναι εαυτός καθώς δεν υπάρχει φυσικός αριθμός m έτσι ώστε $F(m) = 20$. Υπάρχουν μόλις 13 εαυτοί αριθμοί μικρότεροι του 100:

$$1, 3, 5, 7, 9, 20, 31, 42, 53, 64, 75, 86, 97$$

Γράψτε ένα πρόγραμμα C το οποίο βρίσκει και τυπώνει όλους τους εαυτούς αριθμούς σε ένα εύρος φυσικών αριθμών $[low, high]$ (το εύρος είναι κλειστό, δηλαδή τα άκρα συμπεριλαμβάνονται), όπου οι αριθμοί δίνονται από την γραμμή εντολών. Ακολουθεί παράδειγμα εκτέλεσης για να βρούμε όλους τους εαυτούς αριθμούς στο διάστημα $[9900, 10000]$:

```
$ gcc -o self self.c
$ ./self 9900 10000
Self numbers: 9903 9914 9925 9927 9938 9949 9960 9971 9982 9993
Found 10 total
```

Υπόδειξη Γράψτε τη συνάρτηση F(m) με άθροισμα ψηφίων. Αντί να ψάχνετε για κάθε n ένα m, σημαδέψτε σε έναν πίνακα όλες τις τιμές F(m) που πέφτουν στο [low, high]· αρκεί να δοκιμάσετε m λίγο μικρότερα από το low (το άθροισμα ψηφίων είναι μικρό). Ελέγξτε ότι low ≤ high.

A16.23 · Τυχεροί Αριθμοί

A16.23 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #13, Θέμα 4* · ★★☆ · programming · `exam-2023-fall-ex13-q4`

Πρόγραμμα: `lucky.c`

Γράψτε ένα πρόγραμμα που να ελέγχει για κάθε ένα ακέραιο όρισμα που του δίνεται εάν ο αριθμός αυτός είναι ”τυχερός” (lucky). Ένας ακέραιος λέγεται τυχερός, εάν μπορεί να εκφραστεί ως το άθροισμα των κύβων δύο θετικών ακεραίων. Για παράδειγμα, ο αριθμός 28 είναι τυχερός (3^3 + 1^3) ενώ ο αριθμός 27 δεν είναι. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ gcc -o lucky lucky.c -lm
$ ./lucky 28 95 128 298374 508167898112
28 is lucky: 1^3 + 3^3 = 28
No positive i, j exist such that i^3+j^3 = 95
128 is lucky: 4^3 + 4^3 = 128
No positive i, j exist such that i^3+j^3 = 298374
508167898112 is lucky: 2344^3 + 7912^3 = 508167898112
```

Υπόδειξη Για κάθε i με $i^3 \le n/2$ αρκεί να ελέγξετε αν το $n - i^3$ είναι τέλειος κύβος. Αν χρησιμοποιήσετε `cbrt`, στρογγυλέψτε και επαληθεύστε με ακέραια αριθμητική, γιατί η κινητή υποδιαστολή κάνει λάθη στρογγύλευσης. Το τελευταίο παράδειγμα χρειάζεται `long long`.

A16.24 · Οι Καλύτεροι Αριθμοί, Παμψηφεί

A16.24 · *Κατατακτήριες Δεκεμβρίου 2024, Θέμα 2* · ★★☆ · programming · `exam-2024-dec-q2`

Στα μαθηματικά, ένας φυσικός αριθμός n λέγεται *πανψηφιακός* (pandigital) ως προς μια βάση b όταν περιέχει *τουλάχιστον* μία φορά όλα τα ψηφία από το 0 μέχρι το $b - 1$. Για παράδειγμα, ο αριθμός 1234567890 είναι πανψηφιακός με βάση το 10 όπως και ο αριθμός 90123456789 (τα ψηφία μπορούν να επαναλαμβάνονται και η σειρά τους δεν έχει σημασία).

Γράψτε ένα πρόγραμμα C το οποίο βρίσκει και τυπώνει όλους τους πανψηφιακούς αριθμούς με βάση το 10 σε ένα εύρος φυσικών αριθμών $[low, high]$ (το εύρος είναι κλειστό, δηλαδή τα άκρα συμπεριλαμβάνονται), όπου οι αριθμοί δίνονται από την γραμμή εντολών. Οι αριθμοί του εύρους δεν θα υπερβούν το $2^{64} - 1$. Ακολουθεί παράδειγμα εκτέλεσης για να βρούμε όλους τους πανψηφιακούς αριθμούς στο διάστημα $[1000000000, 1300000000]$:

```
$ ./pandigital 1000000000 1300000000
All pandigital numbers in the range 1000000000 to 1300000000 follow:
1023456789
1023456798
1023456879
1023456897
...
1298765304
1298765340
1298765403
1298765430
```

Υπόδειξη Τα όρια χωράνε μόνο σε `unsigned long long`, οπότε διαβάστε τα με `strtoull` και προσέξτε τον βρόχο όταν `high` είναι η μέγιστη τιμή του τύπου (ο μετρητής δεν πρέπει να υπερχειλίσει). Για κάθε αριθμό βγάλτε τα ψηφία του με `% 10` και `/ 10` και σημειώστε ποια εμφανίστηκαν, π.χ. σε μια μάσκα 10 bit. Αριθμοί με λιγότερα από 10 ψηφία δεν μπορούν να είναι πανψηφιακοί.

A16.25 · Χτίζοντας έναν Χιονάνθρωπο

A16.25 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #3 (Frozen Themed), Θέμα 4* · ★★★ · programming · `exam-2023-fall-ex3-q4`

Πρόγραμμα: `olaf.c` (25 μονάδες)

Βρισκόμαστε σε έναν χιονισμένο τετράγωνο κήπο διαστάσεων NxN και Θέλουμε να φτιάξουμε έναν χιονάνθρωπο. Δυστυχώς το χιόνι είναι μαζεμένο σε συγκεκριμένους σωρούς μέσα στον κήπο (δεν είναι παντού). Γράψτε ένα πρόγραμμα που αποφασίζει που πρέπει να φτιάξουμε τον χιονάνθρωπό μας ώστε να κάνουμε τον ελάχιστο δυνατό κόπο. Για λόγους απλοποίησης θεωρούμε ότι όλες οι συντεταγμένες στον κήπο είναι ακέραιοι και ότι οι κινήσεις μας μπορούν να είναι μόνο πάνω-κάτω-αριστερά-δεξιά (όχι διαγώνια). Το μέγεθος του κήπου και οι τοποθεσίες των σωρών χιονιού δίνονται στο πρόγραμμα μέσω αρχείου του οποίου το όνομα θα είναι το πρώτο όρισμα του προγράμματος. Το αρχείο θα περιέχει την διάσταση του κήπου (N) ακολουθούμενη από τις συντεταγμένες του κάθε σωρού. Παράδειγμα επιτυχούς εκτέλεσης ακολουθεί:

```
$ gcc -o olaf olaf.c
```

```
$ cat map.txt
10
1 1
6 7
2 8
7 2
$ ./olaf map.txt
We will position the snowman on (4, 4) with a minimum cost of 22.
```

Παρακάτω βλέπουμε την οπτικοποίηση της επιλογής μας - ο χιονάνθρωπος βρίσκεται στην θέση "καρότο" ενώ οι σωροί με τα χιόνια απεικονίζονται με τους κύκλους. Από τους δύο πάνω σωρούς παρατηρούμε ότι χρειαζόμαστε 6 βήματα για να φτάσουμε τον χιονάνθρωπο ενώ από τους δύο κάτω χρειαζόμαστε 5 - σύνολο (6 + 6 + 5 + 5 =) 22. Παρατηρήστε ότι μπορεί να υπάρχουν πάνω από μία βέλτιστες τοποθετήσεις, χρειάζεται να βρούμε μόνο μία από αυτές. Για λόγους απλότητας ο χιονάνθρωπος μπορεί να τοποθετηθεί πάνω σε έναν σωρό (και τα βήματα που απαιτούνται σε αυτήν την περίπτωση είναι 0).

Εικόνα: path to the snowman

Υπόδειξη Το κόστος μιας θέσης είναι το άθροισμα των αποστάσεων Manhattan `|x - xi| + |y - yi|` από όλους τους σωρούς. Μια λύση ωμής βίας δοκιμάζει κάθε κελί του κήπου, αλλά κοστίζει $O(N^2 \cdot K)$· παρατηρήστε ότι οι δύο συντεταγμένες ανεξαρτητοποιούνται και ρωτήστε τον εαυτό σας ποιο σημείο ελαχιστοποιεί το άθροισμα απολύτων αποστάσεων σε μία διάσταση. Ελέγξτε ότι οι συντεταγμένες είναι μέσα στον κήπο και ότι υπάρχει τουλάχιστον ένας σωρός.

A16.26 · Μετρώντας τα Αστέρια - stars

A16.26 · *Εξέταση Ιανουαρίου 2025, Θέμα 6* · ★★★ · programming · `exam-2025-jan-q6`

Μετρώντας τα Αστέρια - stars [25 Μονάδες]

Είμαστε στην διαδικασία προγραμματισμού του καινούριου διαστημοπλοίου του DI και για την λειτουργία του είναι απαραίτητο ένα υποσύστημα που να υπολογίζει το σύνολο των αστεριών σε ένα τμήμα του ορίζοντα. Το Σχήμα 1 δείχνει ένα παράδειγμα:

0	3	0	1	0
0	0	0	8	0
0	**42**	**0**	**0**	0
7	**1**	**0**	**5**	4
0	2	0	0	0

Σχήμα 1: Μετρώντας τα αστέρια σε μια υποπεριοχή του ορίζοντα (πλέγμα 5x5). Ο ακέραιος στο κάθε κελί υποδεικνύει έναν αριθμό αστεριών. Στο παράδειγμα επιλέξαμε ένα παραλληλόγραμμο με συντεταγμένες από το (2, 1) (άνω αριστερό άκρο) έως και το (3, 3) (κάτω δεξί άκρο). Στην επιλεγμένη περιοχή (με έντονα) το σύνολο των αστεριών είναι 42 + 1 + 5 = 48.

Θεωρούμε πως ο ορίζοντας είναι ένα τετραγωνικό πλέγμα και πως σε κάθε κελί του υπάρχει ένας ακέραιος αριθμός από αστέρια. Οι περιοχές που μπορεί να επιλέξει ο χρήστης είναι αποκλειστικά παραλληλόγραμμα σε σχήμα, τα οποία προσδιορίζονται από τις συντεταγμένες της άνω αριστερής και κάτω δεξιάς γωνίας τους.

Γράψτε ένα πρόγραμμα το οποίο παίρνει ως όρισμα το όνομα αρχείου που περιέχει την διάσταση του πλέγματος καθώς και τους αριθμούς των αστεριών ανά κελί. Στην συνέχεια, το πρόγραμμά σας πρέπει να διαβάζει από την πρότυπη είσοδο (stdin) τις συντεταγμένες του χρήστη και να υπολογίζει τον αριθμό αστεριών *αποδοτικά* για κάθε περιοχή που θα ζητηθεί. Το πρόγραμμά σας πρέπει να είναι σε θέση να απαντάει σε επαναλαμβανόμενες ερωτήσεις από τον χρήση μέχρι να λάβει EOF. Αν η διάσταση του πλέγματος είναι N και ο αριθμός των ερωτήσεων του χρήστη είναι Q, ποια η χρονική και χωρική πολυπλοκότητα της λύσης σας (8/25 της βαθμολογίας); Παράδειγμα εκτέλεσης ακολουθεί:

```
$ cat horizon.txt
5
0 3 0 1 0
0 0 0 8 0
0 42 0 0 0
7 1 0 5 4
0 2 0 0 0
$ ./stars horizon.txt
Provide top-left x, y coordinates: 2 1
Provide bottom-right x, y coordinates: 3 3
Total number of stars in region (2, 1) - (3, 3) is 48
Provide top-left x, y coordinates: 0 0
Provide bottom-right x, y coordinates: 4 4
Total number of stars in region (0, 0) - (4, 4) is 73
Provide top-left x, y coordinates: 3 0
Provide bottom-right x, y coordinates: 3 0
Total number of stars in region (3, 0) - (3, 0) is 7
Provide top-left x, y coordinates: Terminating
```

Υπόδειξη Από το παράδειγμα, η πρώτη συντεταγμένη είναι η γραμμή και η δεύτερη η στήλη. Αν αθροίζετε κάθε περιοχή κελί προς κελί, κάθε ερώτηση κοστίζει $O(N^2)$: σκεφτείτε ποια πληροφορία μπορείτε να προϋπολογίσετε μία φορά μετά το διάβασμα του αρχείου ώστε κάθε ερώτηση να απαντιέται με λίγες πράξεις (χρόνος έναντι μνήμης). Δεσμεύστε το πλέγμα δυναμικά αφού διαβάσετε το N, και χρησιμοποιήστε την τιμή επιστροφής της `scanf` για να σταματήσετε στο EOF.

Κεφάλαιο 17: Δυαδική Αναζήτηση και Ταξινόμηση

Kahoot από το αμφιθέατρο (K17.1–K17.8)

K17.1 · Δυαδική αναζήτηση σε μη ταξινομημένο πίνακα

K17.1 · *Kahoot «Δυαδική Αναζήτηση και Ταξινόμηση» (διάλεξη 17)* · ★☆☆ · multiple-choice · 94% σωστές · `kahoot-binary-search-unsorted`

Έχει νόημα να τρέξω δυαδική αναζήτηση σε μη ταξινομημένο πίνακα;

- True
- False

Υπόδειξη Σκεφτείτε με ποιο κριτήριο η δυαδική αναζήτηση πετάει τη μισή ακολουθία σε κάθε βήμα, και τι προϋποθέτει αυτό.

K17.2 · Αναζήτηση σε μη ταξινομημένο πίνακα

K17.2 · *Kahoot «Δυαδική Αναζήτηση και Ταξινόμηση» (διάλεξη 17) και «Δυαδική Αναζήτηση, Ταξινόμηση, Πολυπλοκότητα και άλλα»* · ★☆☆ · multiple-choice · 85% σωστές · `kahoot-linear-vs-binary-search`

Ψάχνω να βρω ένα στοιχείο σε έναν μη ταξινομημένο πίνακα n θέσεων. Χρησιμοποιώ:

- Γραμμική Αναζήτηση
- Δυαδική Αναζήτηση

Υπόδειξη Ρωτήστε τον εαυτό σας τι προϋπόθεση έχει ο καθένας από τους δύο αλγορίθμους για τη σειρά των στοιχείων.

K17.3 · Πολυπλοκότητα της bubblesort

K17.3 · *Kahoot «Δυαδική Αναζήτηση και Ταξινόμηση» (διάλεξη 17)* · ★☆☆ · multiple-choice · 75% σωστές · `kahoot-bubblesort-complexity`

Ποια η χρονική πολυπλοκότητα της bubblesort;

- O(n)
- $O(n^2)$
- O(log n)

- O(n log n)

Υπόδειξη Μετρήστε πόσα περάσματα κάνει ο αλγόριθμος στη χειρότερη περίπτωση και πόσες συγκρίσεις γίνονται σε κάθε πέρασμα.

K17.4 · Ταξινόμηση ενός εκατομμυρίου ακεραίων

K17.4 · *Kahoot «Δυαδική Αναζήτηση και Ταξινόμηση» (διάλεξη 17) και «Δυαδική Αναζήτηση, Ταξινόμηση, Πολυπλοκότητα και άλλα»* · ★★☆ · multiple-choice · 68% σωστές · `kahoot-sort-million-ints`

Ποιος είναι χρονικά ο πιο αποδοτικός τρόπος να ταξινομήσεις 10^6 ακεραίους 32-bit;

- bubblesort
- insertion sort
- mergesort
- selection sort

Υπόδειξη Συγκρίνετε την πολυπλοκότητα χειρότερης περίπτωσης των τεσσάρων αλγορίθμων και υπολογίστε χονδρικά πόσες πράξεις σημαίνει η καθεμία για $n = 10^6$.

K17.5 · Μέση πολυπλοκότητα της quicksort

K17.5 · *Kahoot «Δυαδική Αναζήτηση και Ταξινόμηση» (διάλεξη 17)* · ★★☆ · multiple-choice · 66% σωστές · `kahoot-quicksort-average`

Η χρονική πολυπλοκότητα στη μέση περίπτωση (average case) της quicksort είναι:

- O(n²)
- O(n log n)
- O(n)
- O(1)

Υπόδειξη Σκεφτείτε πόσα επίπεδα διαμέρισης έχει η quicksort όταν ο pivot χωρίζει τον πίνακα περίπου στη μέση, και πόση δουλειά γίνεται σε κάθε επίπεδο.

K17.6 · Βήματα δυαδικής αναζήτησης σε 2^50 στοιχεία

K17.6 · *Kahoot «Δυαδική Αναζήτηση και Ταξινόμηση» (διάλεξη 17) και «Δυαδική Αναζήτηση, Ταξινόμηση, Πολυπλοκότητα και άλλα»* · ★★☆ · multiple-choice · 52% σωστές · `kahoot-binary-search-steps`

Αναζητώ έναν ακέραιο σε ταξινομημένο πίνακα με $10^{15} \approx 2^{50}$ στοιχεία. Με δυαδική αναζήτηση θα χρειαστώ τόσα βήματα:

- 10^{15}
- $10^{7.5}$
- 2^{10}
- 50

Υπόδειξη Κάθε βήμα της δυαδικής αναζήτησης υποδιπλασιάζει το διάστημα που απομένει· πόσες φορές πρέπει να διαιρέσετε το 2^{50} με το 2 για να φτάσετε στο 1;

K17.7 · Αναζήτηση αριθμημένης θέσης σε αίθουσα

K17.7 · *Kahoot «Δυαδική Αναζήτηση και Ταξινόμηση» (διάλεξη 17) και «Δυαδική Αναζήτηση, Ταξινόμηση, Πολυπλοκότητα και άλλα»* · ★★☆ · multiple-choice · 48% σωστές · `kahoot-find-seat-sorted`

Ψάχνω αν η αριθμημένη θέση μου J-13 υπάρχει (κάποιες λείπουν) στην αίθουσα προβολής N×N θέσεων. Χρειάζομαι τόσα βήματα:

- $O(n^2)$
- $O(\log n)$
- $O(2^n)$
- 0 - μετά τον κόβιντ δεν έχει βγει ταινία της προκοπής

Συχνή παρανόηση Το 33% επέλεξε $O(n^2)$, σαν να έπρεπε να ελέγξει κάθε θέση μία-μία. Επειδή όμως οι θέσεις είναι ταξινομημένες, δεν χρειάζεται να τις κοιτάξετε όλες.

Υπόδειξη Οι θέσεις είναι αριθμημένες με σειρά (γραμμή και αριθμός)· σκεφτείτε ποιον αλγόριθμο αναζήτησης σας επιτρέπει αυτή η διάταξη.

K17.8 · Η πιο γρήγορη ταξινόμηση στη χειρότερη περίπτωση

K17.8 · *Kahoot «Δυαδική Αναζήτηση και Ταξινόμηση» (διάλεξη 17)* · ★★★ · multiple-choice · 28% σωστές · `kahoot-sort-worst-case`

Ποια μέθοδος ταξινόμησης είναι πιο γρήγορη στη χειρότερη περίπτωση;

- insertion sort
- selection sort
- quicksort
- bubblesort

- Καμία, όλες είναι $O(n^2)$

Συχνή παρανόηση Το 35% επέλεξε την quicksort, που είναι O(n log n) στη μέση περίπτωση αλλά $O(n^2)$ στη χειρότερη (κακή επιλογή pivot). Ένα 27% επέλεξε την bubblesort, ίσως επειδή τελειώνει γρήγορα σε ήδη ταξινομημένο πίνακα, που όμως είναι η καλύτερη και όχι η χειρότερη περίπτωση.

Υπόδειξη Για κάθε αλγόριθμο, σκεφτείτε ποια είσοδος είναι η χειρότερη δυνατή, και για την quicksort ειδικά, τι γίνεται όταν ο pivot είναι πάντα το μικρότερο ή το μεγαλύτερο στοιχείο.

Ζέσταμα: από τις διαφάνειες (A17.1–A17.9)

A17.1 · Πολυπλοκότητα της δυαδικής αναζήτησης

A17.1 · *Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 18* · ★☆☆ · short-answer · `slides-lec17-binary-search-complexity`

Τι πολυπλοκότητα (χρόνου και χώρου) έχει αυτός ο αλγόριθμος;

```
int binary_search(int elem, int *array, int n) {
  int mid, low = 0, high = n - 1;
  while (low <= high) {
    mid = low + (high - low) / 2;
    if (array[mid] == elem)
      return 1;
    else if (array[mid] < elem)
      low = mid + 1;
    else
      high = mid - 1;
  }
  return 0;
}
```

Υπόδειξη Πόσα στοιχεία μένουν στο διάστημα `[low, high]` μετά από κάθε γύρο του βρόχου; Μετρήστε πόσες φορές μπορείτε να διαιρέσετε το n με το 2 μέχρι να φτάσετε στο 1. Για τον χώρο, μετρήστε τις μεταβλητές που χρησιμοποιεί η συνάρτηση.

A17.2 · Γρηγορότερα από O(n);

A17.2 · *Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 11* · ★☆☆ · short-answer · `slides-lec17-faster-than-linear`

Η γραμμική αναζήτηση βρίσκει ένα στοιχείο σε μια ακολουθία n στοιχείων σε χρόνο $O(n)$.

Μπορούμε να βρούμε ένα στοιχείο πιο γρήγορα από $O(n)$;

Υπόδειξη Σκεφτείτε πρώτα τι θα μπορούσε να συμβαίνει σε ένα στοιχείο που δεν κοιτάξατε, αν ο πίνακας δεν έχει καμία δομή. Έπειτα σκεφτείτε πώς ψάχνετε μια λέξη σε ένα λεξικό: ποια ιδιότητα των δεδομένων σάς επιτρέπει να μην τα κοιτάξετε όλα;

A17.3 · Κρυμμένος αριθμός έως 10^9

A17.3 · *Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 6* · ★☆☆ · short-answer · `slides-lec17-guess-billion`

Έχω έναν αριθμό στο νου μου στο διάστημα $[1, 10^9]$.

Θέλετε να τον βρείτε. Μπορείτε να με ρωτήσετε ό,τι ερώτηση θέλετε και εγώ απαντάω ναι ή όχι.

Τι θα με ρωτήσετε; Πόσες προσπάθειες χρειάζεστε για να βρείτε τον αριθμό μου;

Υπόδειξη Διαλέξτε ερωτήσεις που χωρίζουν τους υποψήφιους αριθμούς σε δύο ίσα μέρη. Μετά από k τέτοιες ερωτήσεις πόσοι υποψήφιοι μένουν; Βρείτε το μικρότερο k για το οποίο το 2^k ξεπερνά το 10^9.

A17.4 · Αναζήτηση χρήστη στο Instagram

A17.4 · *Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 19* · ★☆☆ · short-answer · `slides-lec17-instagram`

Το Instagram έχει 2 δισεκατομμύρια χρήστες σε έναν πίνακα ακεραίων (έστω ένας ακέραιος ανά χρήστη). Πόσα βήματα (χρονική πολυπλοκότητα) χρειάζεστε για να βρείτε αν ο χρήστης 424242 βρίσκεται στον πίνακα;

Υπόδειξη Η απάντηση εξαρτάται από το αν ο πίνακας είναι ταξινομημένος. Υπολογίστε τα βήματα της γραμμικής αναζήτησης στη χειρότερη περίπτωση και της δυαδικής, βρίσκοντας τη μικρότερη δύναμη του 2 που ξεπερνά το $2 \cdot 10^9$.

A17.5 · Αναζήτηση σε πίνακα 100 ακεραίων

A17.5 · *Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 9* · ★☆☆ · programming · `slides-lec17-linear-search`

Θέλω μια συνάρτηση που να δέχεται έναν πίνακα 100 ακεραίων και έναν ακέραιο και να γυρνάει την θέση του στοιχείου αν το βρήκε ή -1. Πως;

```
int find(int haystack[100], int needle);
```

Ποια είναι η χρονική πολυπλοκότητα και ο χώρος μνήμης που χρειάζεται;

Υπόδειξη Διατρέξτε τον πίνακα με έναν βρόχο και επιστρέψτε μόλις βρείτε ίσο στοιχείο· το `-1` επιστρέφεται μόνο αφού τελειώσει ο βρόχος. Για την πολυπλοκότητα, μετρήστε τις συγκρίσεις στη χειρότερη περίπτωση.

A17.6 · Η συνάρτηση swap

A17.6 · *Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 21* · ★☆☆ · programming · `slides-lec17-swap`

Γράψτε μια συνάρτηση `swap` που ανταλλάζει δύο ακεραίους.

```
#include <stdio.h>
int main() {
  int a = 100, b = 200;
  printf("%d %d\n", a, b);
  swap( ... );
  printf("%d %d\n", a, b);
  return 0;
}
```

Υπόδειξη Τα ορίσματα στη C περνούν με τιμή, άρα μια συνάρτηση που παίρνει δύο `int` αλλάζει μόνο τα δικά της αντίγραφα. Περάστε στη `swap` τις διευθύνσεις των `a` και `b` και χρησιμοποιήστε μια βοηθητική μεταβλητή.

A17.7 · Δυαδική αναζήτηση σε ταξινομημένο πίνακα

A17.7 · *Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 14* · ★★☆ · programming · `slides-lec17-binary-search`

Θέλω μια συνάρτηση που να δέχεται έναν πίνακα n ταξινομημένων ακεραίων σε αύξουσα σειρά και έναν ακέραιο που ψάχνουμε και να γυρνάει αν υπάρχει στον πίνακα ή όχι. Πως;

```
int binary_search(int elem, int *array, int n);
```

Υπόδειξη Κρατήστε δύο δείκτες θέσης, `low` και `high`, για το τμήμα όπου μπορεί ακόμη να βρίσκεται το στοιχείο, και συγκρίνετε με το μεσαίο στοιχείο του τμήματος. Σκεφτείτε προσεκτικά τη συνθήκη του βρόχου (τι γίνεται όταν μένει ένα στοιχείο;) και ποια είναι τα νέα όρια, ώστε το τμήμα να μικραίνει πάντα.

A17.8 · Το σφάλμα της δυαδικής αναζήτησης

A17.8 · *Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 16* · ★★☆ · debug · `slides-lec17-binary-search-bug`

Αυτός ο αλγόριθμος έχει σφάλμα, μπορούμε να το βρούμε και να το διορθώσουμε;

```
int binary_search(int elem, int *array, int n) {
  int mid, low = 0, high = n - 1;
  while (low <= high) {
    mid = (low + high) / 2;
    if (array[mid] == elem)
      return 1;
    else if (array[mid] < elem)
      low = mid + 1;
    else
      high = mid - 1;
  }
  return 0;
}
```

Υπόδειξη Η λογική του αλγορίθμου είναι σωστή για μικρούς πίνακες. Σκεφτείτε έναν πίνακα με δύο δισεκατομμύρια στοιχεία: ποιες τιμές μπορούν να πάρουν τα `low` και `high`, και χωράει πάντα το άθροισμά τους σε `int`; Βρείτε μια έκφραση για το μέσο που δεν αθροίζει δύο μεγάλες θέσεις.

A17.9 · Ο γρίφος του κρυμμένου αριθμού

A17.9 · *Διάλεξη 17: Δυαδική Αναζήτηση και Ταξινόμηση, διαφάνεια 5* · ★★☆ · short-answer · `slides-lec17-guess-100`

Έχω έναν αριθμό στο νου μου στο διάστημα $[1, 100]$. Θέλετε να τον βρείτε. Μπορείτε να με ρωτήσετε ό,τι ερώτηση θέλετε και εγώ απαντάω ναι ή όχι.

- Αν τον βρείτε στην 1η προσπάθεια => κερδίζετε 5€
- Αν τον βρείτε στην 2η προσπάθεια => κερδίζετε 4€
- Αν τον βρείτε στην 3η προσπάθεια => κερδίζετε 3€

- Αν τον βρείτε στην 4η προσπάθεια => κερδίζετε 2€
- Αν τον βρείτε στην 5η προσπάθεια => κερδίζετε 1€
- Αν τον βρείτε στην 6η προσπάθεια => κερδίζετε 0€
- Αν τον βρείτε στην 7η προσπάθεια => μου δίνετε 1€

Παίζουμε;

(Microsoft Interview Question by Steve Ballmer.)

Υπόδειξη Βρείτε πρώτα πόσες ερωτήσεις χρειάζεται στη χειρότερη περίπτωση η στρατηγική που μοιράζει κάθε φορά το διάστημα στη μέση, και πόσους αριθμούς μπορείτε να εντοπίσετε με 1, 2, 3, ... ερωτήσεις. Έπειτα σκεφτείτε ότι τον αριθμό τον διαλέγει κάποιος που ξέρει τη στρατηγική σας.

Θέματα εξετάσεων (A17.10–A17.13)

A17.10 · Η συνάρτηση what

A17.10 · *Εξέταση Σεπτεμβρίου 2024, Θέμα 2* · ★☆☆ · trace · `exam-2024-sep-q2`

Η συνάρτηση what (10 Μονάδες)

```c
int what(int *array, int x, int y) {
  int mid, low = 0, high = y - 1;
  while (low <= high) {
    mid = low + (high - low) / 2;
    printf("%d\n", mid);
    if (array[mid] == x) return 1;
    else if (array[mid] < x) low = mid + 1;
    else high = mid - 1;
  }
  return 0;
}
```

Τι κάνει η συνάρτηση `what` (μέχρι 10 λέξεις εξήγηση); Τι θα τυπώσει αν κληθεί με ορίσματα `{57, 98, 105, 241}, 36, 4`;

Υπόδειξη Αναγνωρίστε το μοτίβο: δύο όρια `low`/`high` που πλησιάζουν μεταξύ τους και ένα μεσαίο στοιχείο που συγκρίνεται με το `x`. Για την έξοδο, κρατήστε έναν πίνακα με τις τιμές `low`, `high`, `mid` σε κάθε επανάληψη και προσέξτε ότι το 36 είναι μικρότερο από όλα τα στοιχεία του πίνακα.

A17.11 · Πλησιάζοντας στον Στόχο

A17.11 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #10, Θέμα 4* · ★★☆ · programming · `exam-2023-fall-ex10-q4`

Πρόγραμμα: `treasure.c`

Γράψτε ένα πρόγραμμα το οποίο παίρνει ως ορίσματα από την γραμμή εντολών μια συντεταγμένη στόχο (goal) και στην συνέχεια ένα σύνολο άλλων συντεταγμένων (starting points) και τυπώνει όλες τις συντεταγμένες ταξινομημένες σε αύξουσα σειρά με βάση την απόστασή τους από τον στόχο. Όλοι οι αριθμοί κινητής υποδιαστολής θέλουμε να τυπώνονται με ακρίβεια με δύο δεκαδικών ψηφίων. Όλες οι συντεταγμένες είναι της μορφής x,y όπου x,y είναι αριθμοί κινητής υποδιαστολής. Η απόσταση δύο σημείων x1,y1 και x2,y2 δίνεται από την έκφραση:

$$\sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

Παράδειγμα εκτέλεσης ακολουθεί:

```
$ gcc -o treasure treasure.c -lm
$ ./treasure 17,42 3.01,50.5 27.27,7 32,65.1 25,4 77,42.42 50,50 30.30,12
1. Point 3.01,50.50 is 16.37 steps away from the goal
2. Point 32.00,65.10 is 27.54 steps away from the goal
3. Point 30.30,12.00 is 32.82 steps away from the goal
4. Point 50.00,50.00 is 33.96 steps away from the goal
5. Point 27.27,7.00 is 36.48 steps away from the goal
6. Point 25.00,4.00 is 38.83 steps away from the goal
7. Point 77.00,42.42 is 60.00 steps away from the goal
```

Υπόδειξη Αναλύστε κάθε όρισμα `x,y` σε δύο `double` (π.χ. με `sscanf` ή `strtod`) και απορρίψτε όσα δεν έχουν αυτή τη μορφή. Κρατήστε κάθε σημείο σε ένα `struct` μαζί με την απόστασή του από τον στόχο και ταξινομήστε με `qsort`.

A17.12 · Εύρεση μηδενός σε πίνακα

A17.12 · *Εξέταση Σεπτεμβρίου 2024, Θέμα 3* · ★★☆ · programming · `exam-2024-sep-q3`

Εύρεση Μηδενός σε Πίνακα [15 Μονάδες]

Γράψτε μία συνάρτηση `find_zero` η οποία λαμβάνει ως όρισμα έναν πίνακα από τιμές `double` ταξινομημένες σε αύξουσα σειρά και επιστρέφει την θέση (index) όπου υπάρχει το 0 ή -1 αν δεν το βρει. Η συνάρτηση μπορεί να έχει οποιαδήποτε διεπαφή (ορίσματα / τύπο επιστροφής) επιθυμείτε. Για παράδειγμα αν δοθεί ο πίνακας `{-8.1, -2.3, 0.0, 1.9}` θέλουμε να επιστραφεί

η τιμή 2. Αντίστοιχα, αν δοθεί ο πίνακας `{1.0}` θέλουμε να επιστραφεί -1. Τι χρονική και χωρική πολυπλοκότητα έχει ο αλγόριθμός σας (5/15 της βαθμολογίας);

Υπόδειξη Ο πίνακας είναι ταξινομημένος, άρα μια σειριακή αναζήτηση αφήνει βαθμούς στο τραπέζι: σκεφτείτε ποιος αλγόριθμος εκμεταλλεύεται τη διάταξη για λογαριθμικό χρόνο. Αφού ο πίνακας δεν «ξέρει» το μέγεθός του, η διεπαφή πρέπει να περνά και το πλήθος των στοιχείων. Ελέγξτε τις ακραίες περιπτώσεις: πίνακας ενός στοιχείου, μηδέν στην πρώτη ή στην τελευταία θέση, όλα θετικά ή όλα αρνητικά.

A17.13 · Η συνάρτηση compute

A17.13 · *Εξέταση Ιανουαρίου 2026, Θέμα 2* · ★★☆ · trace · `exam-2026-jan-q2`

Η συνάρτηση compute (15 Μονάδες)

```
int *compute(const int *a, size_t na, const int *b, size_t nb) {
    size_t i = 0, j = 0, k = 0;
    int * out = malloc((na + nb) * sizeof(int));
    while (i < na && j < nb) {
        if (a[i] <= b[j])
            out[k++] = a[i++];
        else
            out[k++] = b[j++];
    }
    while (i < na)
        out[k++] = a[i++];
    while (j < nb)
        out[k++] = b[j++];
    return out;
}
```

1. Τι κάνει η συνάρτηση compute (μέχρι 15 λέξεις εξήγηση);
2. Ποιο θα είναι το περιεχόμενο της μεταβλητής result μετά την εκτέλεση της παρακάτω ακολουθίας εντολών και σε ποια κατηγορία μνήμης είναι αποθηκευμένο;

   ```
   int arg1[] = {71, 111, 111, 100, 32, 0};
   int arg2[5] = {'j', 111, 98, '!', 0};
   int * result = compute(arg1, 3, arg2, 2);
   ```

3. Υπάρχει κάποιο σφάλμα στην συνάρτηση compute; Αιτιολογήστε την απάντησή σας.

Στο τέλος του φυλλαδίου δίνεται πίνακας ASCII ως βοήθημα.

Υπόδειξη Δείτε το βήμα που επαναλαμβάνεται: συγκρίνει τα δύο ”τρέχοντα” στοιχεία και παίρνει το μικρότερο· σε ποιον αλγόριθμο ταξινόμησης είναι αυτό το βασικό βήμα και τι υποθέτει για τις εισόδους; Για το `result` προσέξτε ότι χρησιμοποιούνται μόνο τα πρώτα `na` και `nb` στοιχεία, και ξεχωρίστε πού ζει ο δείκτης από πού ζουν τα δεδομένα στα οποία δείχνει. Για το σφάλμα, σκεφτείτε τι μπορεί να επιστρέψει η `malloc`.

Κεφάλαιο 18: Ταξινόμηση και Δεδομένα Εισόδου #2

Kahoot από το αμφιθέατρο (K18.1–K18.5)

K18.1 · stdin και αρχεία

K18.1 · *Kahoot «Δομές + Αρχεία», «Δυαδική Αναζήτηση και Ταξινόμηση» (διάλεξη 17)* · ★☆☆ · multiple-choice · 84% σωστές · `kahoot-stdin-vs-files`

Η πρότυπη είσοδος `stdin` και τα ανοιγμένα αρχεία στην C δεν έχουν καμία σχέση.

- True
- False

Υπόδειξη Κοιτάξτε ποιος είναι ο τύπος του `stdin` και αν μπορείτε να το δώσετε σε συναρτήσεις όπως η `fscanf`.

K18.2 · Ανάγνωση int σε little endian

K18.2 · *Kahoot «Δομές + Αρχεία»* · ★★☆ · multiple-choice · 62% σωστές · `kahoot-little-endian-read`

Το αρχείο `input.txt` περιέχει τα bytes `0xfe 0xca 0xef 0xbe`. Αν διαβάσουμε έναν `int` σε μορφή little endian, θα πάρουμε:

- 0xfecaefbe
- 0xefbefeca
- 0xbeefcafe
- 0xcafebeef

Υπόδειξη Στο little endian το πρώτο byte στη μνήμη (ή στο αρχείο) είναι το λιγότερο σημαντικό byte του αριθμού.

K18.3 · Ο file descriptor του stderr

K18.3 · *Kahoot «Δομές + Αρχεία», «Δυαδική Αναζήτηση και Ταξινόμηση» (διάλεξη 17)* · ★★☆ · multiple-choice · 51% σωστές · `kahoot-stderr-fd`

Το `stderr` έχει συνήθως file descriptor με αριθμό:

- 0
- 1
- 2
- 3

Συχνή παρανόηση Το 29% επέλεξε 1, που είναι ο αριθμός του `stdout`· τα τρία πρότυπα ρεύματα έχουν τους πρώτους αριθμούς με τη σειρά `stdin`, `stdout`, `stderr`.

Υπόδειξη Θυμηθείτε τη σειρά των τριών ρευμάτων που ανοίγουν αυτόματα και ότι η αρίθμηση ξεκινάει από το 0 (σκεφτείτε και το `2>` του shell).

K18.4 · Αποτυχία της fopen

K18.4 · *Kahoot «Δομές + Αρχεία»* · ★★☆ · multiple-choice · 50% σωστές · `kahoot-fopen-failure`

Η συνάρτηση `fopen` μόλις απέτυχε να ανοίξει ένα αρχείο. Επιστρέφει:

- 1
- -1
- NULL
- Τίποτα, απλά κρασάρει

Υπόδειξη Η `fopen` επιστρέφει δείκτη σε `FILE`· ποια τιμή δείκτη σημαίνει «δεν δείχνω πουθενά»;

K18.5 · Συνάρτηση σύγκρισης για αύξουσα ταξινόμηση

K18.5 · *Kahoot «Ταξινόμηση και Δομές»* · ★★★ · multiple-choice · 12% σωστές · `kahoot-qsort-compare-ascending`

Θέλω να ταξινομήσω σε αύξουσα σειρά βαθμολογίας τον πίνακα `struct student[1024]`. Χρησιμοποιώ τη συνάρτηση σύγκρισης:

- `int comp(... s1, ... s2) { return s1.grade - s2.grade; }`
- `int comp(... s1, ... s2) { return s2.grade - s1.grade; }`
- `int comp(... s1, ... s2) { return s1.grade == s2.grade; }`

- `int comp(... s1, ... s2) { return s1.grade ^ s2.grade; }`

Συχνή παρανόηση Το 43% επέλεξε `s2.grade - s1.grade`, αντιστρέφοντας τη σύμβαση: αρνητική τιμή σημαίνει ότι το `s1` μπαίνει πρώτο, οπότε η αφαίρεση `s2 - s1` δίνει φθίνουσα σειρά. Οι επιλογές με `==` και `^` (20% η καθεμία) δεν δίνουν καν πρόσημο που να λέει ποιο στοιχείο προηγείται.

Υπόδειξη Η συνάρτηση σύγκρισης της `qsort` επιστρέφει αρνητικό, μηδέν ή θετικό· σκεφτείτε ποιο πρόσημο σημαίνει «το πρώτο όρισμα πάει πριν από το δεύτερο» και δοκιμάστε με δύο βαθμούς, π.χ. 5 και 8.

Ζέσταμα: από τις διαφάνειες (A18.1–A18.6)

A18.1 · Μέγεθος δισδιάστατου πίνακα και μιας γραμμής του

A18.1 · *Διάλεξη 18, διαφάνεια 2* · ★☆☆ · short-answer · `slides-lec18-2d-sizeof`

Πόσος χώρος δεσμεύεται από τη δήλωση `int array[5][10];`;

Ποιο είναι το `sizeof(array[3])`;

Υπόδειξη Ένας δισδιάστατος πίνακας είναι πίνακας από γραμμές: σκεφτείτε πόσα `int` έχει όλος ο πίνακας και τι τύπου είναι το `array[3]`. Εκφράστε τις απαντήσεις με `sizeof(int)`.

A18.2 · Γιατί μπορεί να αποτύχει η fopen;

A18.2 · *Διάλεξη 18, διαφάνειες 43–44* · ★☆☆ · short-answer · `slides-lec18-fopen-fail`

Ελέγχουμε πάντα το αποτέλεσμα της `fopen` αν είναι `NULL`. Για ποιο λόγο μπορεί να αποτύχει;

```c
#include <stdio.h>
int main() {
  FILE *fileToRead, *fileToWrite;
  fileToRead = fopen("input.txt", "r");
  if (!fileToRead) {
    return 1;
  }
  fileToWrite = fopen("output.txt", "w");
  if (!fileToWrite) {
    return 1;
  }
  return 0;
```

```
}
```

Υπόδειξη Σκεφτείτε ξεχωριστά το άνοιγμα για διάβασμα και για γράψιμο: τι πρέπει να υπάρχει ήδη, τι πρέπει να επιτρέπεται στον χρήστη και ποιους πόρους του συστήματος καταναλώνει ένα ανοιχτό αρχείο.

A18.3 · Υποτείνουσα με scanf

A18.3 · *Διάλεξη 18, διαφάνειες 29–30* · ★☆☆ · trace · `slides-lec18-hypotenuse`

Τι κάνει το παρακάτω πρόγραμμα; Τι τυπώνει αν ο χρήστης δώσει `3.0 4.0`;

```c
#include <stdio.h>
#include <math.h>
int main() {
  double d1, d2;
  printf("Gimme two doubles: ");
  scanf("%lf %lf", &d1, &d2);
  printf("Hypotenuse: %.1f\n", sqrt(d1 * d1 + d2 * d2));
  return 0;
}
```

Υπόδειξη Το `%lf` διαβάζει `double` και το `%.1f` τυπώνει με ένα δεκαδικό. Αναγνωρίστε τον τύπο που υπολογίζεται μέσα στη `sqrt` (Πυθαγόρειο θεώρημα).

A18.4 · Ανακατεύθυνση της stderr

A18.4 · *Διάλεξη 18, διαφάνεια 56* · ★☆☆ · tooling · `slides-lec18-stderr-redirect`

Σύγκρινε το `find / -name foo` με το `find / -name foo 2> error.txt`.

Υπόδειξη Τρέξτε και τις δύο εντολές και δείτε τι εμφανίζεται στην οθόνη και τι στο `error.txt`. Ποιον file descriptor έχει η `stderr`, και σε ποιο ρεύμα γράφει η `find` τα μηνύματα «Permission denied»;

A18.5 · Ακέραιοι από αρχείο κειμένου με fread

A18.5 · *Διάλεξη 18, διαφάνειες 50–51* · ★★☆ · trace · `slides-lec18-fread-int`

Το αρχείο `input.txt` δημιουργήθηκε με `echo hello > input.txt`. Εξηγήστε την έξοδο του προγράμματος («What?!»).

```c
#include <stdio.h>
int main() {
  FILE *fileToRead, *fileToWrite;
  fileToRead = fopen("input.txt", "r");
  if (!fileToRead) return 1;
  int buffer[1024];
  size_t integersRead = fread(buffer, sizeof(int), 1023, fileToRead);
  printf("# of integers read: %d\n", integersRead);
  printf("Integer read: %d %08x\n", buffer[0], buffer[0]);
  fclose(fileToRead);
  return 0;
}
```

```
$ ./freadint
##### of integers read: 1
Integer read: 1819043176 6c6c6568
$ hexdump -C input.txt
00000000  68 65 6c 6c 6f 0a  |hello.|
```

Υπόδειξη Η `fread` αντιγράφει bytes χωρίς καμία μετατροπή και μετρά μόνο ολόκληρα δεδομένα μεγέθους `sizeof(int)`. Συγκρίνετε τα bytes του `hexdump` με το δεκαεξαδικό που τυπώθηκε και θυμηθείτε το endianness.

A18.6 · Μια λέξη σε char[7] με scanf

A18.6 · *Διάλεξη 18, διαφάνειες 31–35* · ★★☆ · debug · `slides-lec18-scanf-string`

Τι κάνει το παρακάτω πρόγραμμα;

```c
#include <stdio.h>
int main() {
  char message[7];
  printf("Say something: ");
  scanf("%s", message);
  printf("%s\n", message);
  return 0;
}
```

```
$ ./message
Say something: hello!
hello!
```

1. Δεν βάλαμε `&` πριν τη μεταβλητή `message`. Πώς και λειτουργεί;

2. Μπορεί να πάει κάτι στραβά εδώ; Τι θα συμβεί αν ο χρήστης γράψει `Houston, we've had a problem here.`; Πώς το διορθώνουμε;

Υπόδειξη Θυμηθείτε σε τι μετατρέπεται το όνομα ενός πίνακα όταν το περνάμε σε συνάρτηση. Για το δεύτερο σκέλος, μετρήστε πόσους χαρακτήρες (μαζί με το `'\0'`) διαβάζει το `%s` από την πρώτη λέξη και πόσοι χωρούν στον πίνακα· η `scanf` δέχεται και πλάτος πεδίου.

Εργαστήριο (A18.7–A18.10)

A18.7 · Μέτρηση στατιστικών αρχείων

A18.7 · *Εργαστήριο 10, Άσκηση 4* · ★☆☆ · programming · `lab-lab10-count`

Στην άσκηση αυτή θα υλοποιήσουμε ένα υποσύνολο της εντολής `wc` του Unix. Συγκεκριμένα, θα κατασκευάσουμε πρόγραμμα που θα μετράει το πλήθος των χαρακτήρων και το πλήθος των γραμμών ενός αρχείου κειμένου.

4.1 Κατασκευάστε το πρόγραμμα `count.c` που να δέχεται ως όρισμα γραμμής εντολής το όνομα ενός αρχείου κειμένου, να το ανοίγει για διάβασμα, να μετράει το πλήθος των χαρακτήρων του αρχείου και να προβάλλει το αποτέλεσμα στην οθόνη.

4.2 Τροποποιήστε το πρόγραμμά σας, ώστε να μετράει και το πλήθος των γραμμών του αρχείου.

Υπόδειξη Διαβάστε το αρχείο χαρακτήρα-χαρακτήρα με `getc` μέχρι το `EOF`, μετρώντας κάθε χαρακτήρα και ξεχωριστά κάθε `'\n'`. Συγκρίνετε τα αποτελέσματά σας με τα `wc -c` και `wc -l` στο ίδιο αρχείο.

A18.8 · Σύγκριση αρχείων

A18.8 · *Εργαστήριο 10, Άσκηση 3* · ★☆☆ · programming · `lab-lab10-filediff`

Δημιουργήστε το πρόγραμμα `filediff.c`, το οποίο να δέχεται στη γραμμή εντολής τα ονόματα δύο αρχείων και να ελέγχει αν τα αρχεία αυτά είναι ίδια byte προς byte.

Υπόδειξη Ανοίξτε και τα δύο αρχεία (ελέγχοντας για `NULL`) και διαβάστε από έναν χαρακτήρα από το καθένα ταυτόχρονα, μέχρι να βρείτε διαφορά ή να τελειώσουν. Μην ξεχάσετε την περίπτωση όπου το ένα αρχείο είναι πρόθεμα του άλλου: τα αρχεία είναι ίδια μόνο αν φτάσουν στο `EOF` ταυτόχρονα.

A18.9 · Δυαδικά αρχεία

A18.9 · *Εργαστήριο 10, Άσκηση 2* · ★★☆ · programming · `lab-lab10-bgrades`

2.1 Κατασκευάστε το πρόγραμμα `bgrades.c`, το οποίο να ανοίγει το αρχείο `grades.dat` για γράψιμο σε δυαδική μορφή. Στη συνέχεια, να διαβάζει ένα όνομα (συμβολοσειρά) από την πρότυπη είσοδο και έναν βαθμό και να τα γράφει στο αρχείο. Το πρόγραμμα να τερματίζει όταν επισημανθεί, με κάποιο τρόπο, το τέλος της εισόδου.

2.2 Στη συνέχεια, επεκτείνετε το πρόγραμμά σας, ώστε να ανοίγει το αρχείο `grades.dat` για διάβασμα, να διαβάζει τα δεδομένα που έχουν γραφεί σ' αυτό και να τα προβάλλει στην οθόνη.

Συναρτήσεις για Δυαδική Εισαγωγή/Εξαγωγή:

1. `size_t fread(void *ptr, size_t size, size_t count, FILE *fp)`: Διαβάζει από το ρεύμα `fp` το πολύ `count` δεδομένα μεγέθους `size` το καθένα και τα αποθηκεύει στη διεύθυνση `ptr`.
2. `size_t fwrite(const void *ptr, size_t size, size_t count, FILE *fp)`: Γράφει στο ρεύμα `fp` το πολύ `count` δεδομένα μεγέθους `size` το καθένα, από τη διεύθυνση `ptr`.

Για να δείτε τα περιεχόμενα του αρχείου `grades.dat`, αν δουλεύετε σ' ένα Unix σύστημα, χρησιμοποιήστε την εντολή `od -tu1c grades.dat` ή `hexdump -C grades.dat`.

Υπόδειξη Σε δυαδικό αρχείο δεν υπάρχουν «γραμμές», οπότε το πρόγραμμα που διαβάζει πρέπει να ξέρει ακριβώς πόσα bytes έγραψε αυτό που έγραψε: το πιο απλό είναι μια εγγραφή σταθερού μεγέθους (π.χ. πίνακας χαρακτήρων σταθερού μήκους για το όνομα και ένας αριθμός), που γράφεται και διαβάζεται ολόκληρη. Χρησιμοποιήστε τα modes με `b` και σταματήστε το διάβασμα όταν η `fread` επιστρέψει λιγότερα στοιχεία από όσα ζητήσατε.

A18.10 · Αρχεία κειμένου

A18.10 · *Εργαστήριο 10, Άσκηση 1* · ★★☆ · programming · `lab-lab10-more`

Κατασκευάστε το πρόγραμμα `more.c` που δέχεται ως όρισμα γραμμής εντολής το όνομα ενός αρχείου κειμένου και προβάλλει τις γραμμές του αρχείου ανά 20, προτρέποντας τον χρήστη να συνεχίσει, αν επιθυμεί, έως ότου συναντήσει το τέλος του αρχείου.

Συναρτήσεις Εισόδου/Εξόδου στην C

Άνοιγμα και Κλείσιμο Αρχείων

```
FILE *fopen(const char *filename, const char *mode);
int fclose(FILE *fp);
```

- `fopen`: Ανοίγει το αρχείο με όνομα `filename` με τρόπο προσπέλασης που καθορίζεται από το `mode`. Επιστρέφει ένα ρεύμα μέσω του οποίου μπορούμε στη συνέχεια να αναφερόμαστε στο αρχείο, ή `NULL` αν δεν ήταν δυνατόν να ανοίξει το αρχείο.

  - `mode` **Παραδείγματα**:
    - `"r"`: Διάβασμα από υπάρχον αρχείο.
    - `"w"`: Γράψιμο σε αρχείο. Δημιουργείται αν δεν υπάρχει ή διαγράφονται τα περιεχόμενα αν υπάρχει.
    - `"a"`: Προσάρτηση δεδομένων στο τέλος του αρχείου.
    - `"r+"`, `"w+"`, `"a+"`: Διάφοροι συνδυασμοί ανάγνωσης και εγγραφής.
- `fclose`: Κλείνει το ρεύμα `fp`. Επιστρέφει `0` σε περίπτωση επιτυχίας.

Άλλες Συναρτήσεις

1. `int feof(FILE *fp)`: Επιστρέφει τιμή διαφορετική από το 0 αν το προηγούμενο διάβασμα απέτυχε λόγω τέλους αρχείου, αλλιώς επιστρέφει 0.
2. `int fprintf(FILE *fp, ...)`: Αντίστοιχη της `printf`, αλλά γράφει στο ρεύμα `fp` αντί του `stdout`.
3. `int fscanf(FILE *fp, ...)`: Αντίστοιχη της `scanf`, αλλά διαβάζει από το ρεύμα `fp` αντί του `stdin`.
4. `char *fgets(char *buf, int max, FILE *fp)`: Διαβάζει το πολύ `max-1` χαρακτήρες από το ρεύμα `fp` μέχρι την αλλαγή γραμμής και τους φυλάσσει στο `buf`. Αν δεν υπάρχουν δεδομένα, επιστρέφει `NULL`.
5. `int getc(FILE *fp)`: Επιστρέφει τον επόμενο χαρακτήρα από το ρεύμα `fp`, ή `EOF` αν φτάσει στο τέλος του αρχείου.

Υπόδειξη Ελέγξτε το `argc` και την τιμή που επιστρέφει η `fopen` πριν διαβάσετε οτιδήποτε. Διαβάζετε με `fgets` και μετράτε γραμμές· κάθε 20 γραμμές ρωτάτε τον χρήστη από το `stdin`, που είναι διαφορετικό ρεύμα από το αρχείο. Προσέξτε ότι μια γραμμή μεγαλύτερη από τον buffer σας έρχεται σε περισσότερα από ένα κομμάτια.

Εργασίες (A18.11)

A18.11 · Προβλέποντας το Μέλλον (future)

A18.11 · *Εργασία 2 (2024-25), Άσκηση 1* · ★★☆ · programming · `hw-2024-hw2-future`

Προβλέποντας το Μέλλον (future - 50 Μονάδες)

Όλες οι προβλέψεις, από τον καιρό μέχρι την επόμενη λέξη σε μια πρόταση, αξιολογούνται με ένα κοινό κριτήριο: το πόσο ακριβείς είναι. Σε αυτήν την άσκηση, θα υλοποιήσουμε έναν αλγόριθμο που χρησιμοποιείται κατεξοχήν για να κάνουμε προβλέψεις, τον *κινούμενο μέσο όρο (Simple Moving Average - SMA)*. Ο κινούμενος μέσος όρος είναι παρεμφερής με τον κανονικό μέσο όρο, με μια διαφορά: ο κινούμενος μέσος όρος απαιτεί και ένα ”παράθυρο” (window), δηλαδή τον αριθμό των τιμών (μετρώντας από το τέλος) που θέλουμε να λάβουμε υπόψη μας στον υπολογισμό του μέσου όρου. Για παράδειγμα, ας υποθέσουμε ότι έχουμε αυτές τις 10 τιμές:

```
9  7  7  1  4  4  4  38  8  4
```

Ο μέσος όρος αυτών των τιμών είναι $\frac{9+7+7+1+4+4+4+38+8+4}{10} = 8.60$. Για να υπολογίσουμε τον κινούμενο μέσο, πρέπει να επιλέξουμε ένα παράθυρο, έστω 3. Τότε ο κινούμενος μέσος με παράθυρο 3 για αυτά τα στοιχεία λαμβάνει υπόψη του μόνο τα τελευταία 3:

$$9 \quad 7 \quad 7 \quad 1 \quad 4 \quad 4 \quad 4 \quad \underbrace{38 \quad 8 \quad 4}_{SMA_3}$$

και επομένως $SMA_3 = \frac{38+8+4}{3} = 16.67$. Αντίστοιχα, μπορούν να οριστούν κινούμενοι μέσοι με μικρότερα παράθυρα (π.χ., παράθυρο 1 σημαίνει μόνο το τελευταίο στοιχείο) ή μεγαλύτερα παράθυρα (π.χ., παράθυρο 10 θα λάμβανε υπόψη του όλες τις τιμές παραπάνω).

Σε αυτήν την άσκηση λοιπόν, καλείστε να υλοποιήσετε ένα πρόγραμμα το οποίο θα υπολογίζει τον κινούμενο μέσο όρο για μια σειρά από δεδομένα και να τον εκτυπώνει ως πρόβλεψη.

Τεχνικές Προδιαγραφές

- Repository Name: progintro/hw2-<YourUsername>
- C Filepath: future/src/future.c
- Το πρόγραμμά θα πρέπει να παίρνει ως κυρίως όρισμα το όνομα του αρχείου που περιέχει τα δεδομένα μας. Προαιρετικά, μπορεί να παίρνει και άλλα δύο ορίσματα προκειμένου να ορίσουμε το μέγεθος του "παραθύρου" που θέλουμε για τον κινούμενο μέσο όρο. Για παράδειγμα αν το τρέξουμε ως `./future data.txt --window 42`, σημαίνει υπολόγισε τον κινούμενο μέσο όρο με παράθυρο 42 για τα δεδομένα του αρχείου data.txt. Αν δεν δοθεί παράμετρος για το μέγεθος του παραθύρου, τότε το (default) παράθυρο πρέπει να είναι μεγέθους 50. Αν το πρόγραμμα εκτελεστεί με ορίσματα που δεν ακολουθούν τις παραπάνω προδιαγραφές, πρέπει να εκτυπώσει αντίστοιχο μήνυμα όπως στα παρακάτω παραδείγματα και να επιστρέφει με κωδικό εξόδου (exit code) 1.
- Το αρχείο με τα δεδομένα θα περιέχει αριθμούς κινητής υποδιαστολής με μέχρι δύο ψηφία ακριβείας. Γενικά, δεν θα σας ζητηθεί να χρησιμοποιήσετε μεγαλύτερη ακρίβεια από double.
- Αν το παράθυρο είναι μικρότερο από 1 ή μεγαλύτερο από τον συνολικό αριθμό δεδομένων, το πρόγραμμά σας πρέπει να τερματίζει με μήνυμα σφάλματος όπως στα παραδείγματα. Το μήνυμά σας πρέπει να τυπώνεται στο stderr του προγράμματος. Το μέγεθος του παραθύρου δεν θα υπερβεί το 10^{18}.
- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr): `gcc -Os -Wall -Wextra -Werror -pedantic -o future future.c`
- README Filepath: future/README.md
- Πρέπει να ολοκληρώνει την εκτέλεση μέσα σε: 1 δευτερόλεπτο.

Παραδείγματα εκτέλεσης ακολουθούν:

```
$ ./future
Usage: ./future <filename> [--window N (default: 50)]
$ cat values.txt
9 7 7 1 4 4 4 38 8 4
$ ./future values.txt --window 10
8.60
$ echo $?
0
$ ./future values.txt --window 3
16.67
$ ./future values.txt --window 1
4.00
$ ./future values.txt --window 0
Window too small!
$ echo $?
1
$ ./future values.txt --window 0 2> out
$ cat out
Window too small!
$ ./future values.txt --window 12
Window too large!
$ echo $?
1
$ ./future values.txt --window 1000000000000
Failed to allocate window memory
```

Φυσικά οι ίδιοι έλεγχοι μπορούν να γίνουν με πραγματικά δεδομένα και να δείτε αν οι προβλέψεις του προγράμματός μας είναι καλές (Το αρχείο dow_jones.txt βρίσκεται στο https://github.com/progintro/data):

```
$ ./future dow_jones.txt
43471.71
$ ./future dow_jones.txt --window 200
40677.19
$ ./future dow_jones.txt --window 1000
35273.61
$ wc -l dow_jones.txt
8302 dow_jones.txt
$ ./future dow_jones.txt --window 8000
15447.33
```

Στο αρχείο README.md πρέπει να προσθέσετε οποιεσδήποτε παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.

Bonus (Προαιρετικό) Πιστεύετε πως μπορείτε να φτιάξετε αλγορίθμους που προβλέπουν πως θα κινηθεί ένα οικονομικό asset καλύτερα από τον κινούμενο μέσο; Αν ναι, προσθέστε ένα easter egg option στο πρόγραμμά σας ονόματι `--compete` και θα μπείτε αυτόματα στον διαγωνισμό μας. Η αξιολόγηση θα γίνει με πραγματικά δεδομένα από χρηματιστήρια του κόσμου. Η διεπαφή θα είναι παρόμοια με παραπάνω, απλά υπολογίζετε την καλύτερή σας πρόβλεψη για την επόμενη τιμή σε μια σειρά δεδομένων:

```
$ ./future dow_jones.txt --compete
43455.32
```

Οι τρεις καλύτερες υποβολές θα λάβουν 100%, 70% και 40% έξτρα βαθμολογία σε αυτήν την άσκηση.

Υπόδειξη Χρειάζεστε μόνο τις τελευταίες N τιμές: ένας κυκλικός πίνακας (ring buffer) μεγέθους ίσου με το παράθυρο, δεσμευμένος με `malloc`, κρατά αυτές τις τιμές όσο διαβάζετε το αρχείο, χωρίς να ξέρετε από πριν πόσες είναι. Ελέγξτε την τιμή επιστροφής της `malloc` (δείτε το παράθυρο 10^{12}) και της `fopen`, διαβάστε το παράθυρο ως 64-bit ακέραιο, και θυμηθείτε ότι τα μηνύματα λάθους πάνε στο `stderr`.

Θέματα εξετάσεων (A18.12–A18.23)

A18.12 · Κρυφό Μήνυμα

A18.12 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #7 (Star Wars Themed), Θέμα 2* · ★☆☆ · programming · `exam-2023-fall-ex7-q2`

Πρόγραμμα: `r2d2.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που παίρνει ως ορίσματα δύο ονόματα αρχείων και στην συνέχεια τυπώνει το xor των bytes των δύο αρχείων. Για κάθε byte του πρώτου αρχείου a0, a1, a2 και κάθε byte του δεύτερου αρχείου b0, b1, b2 το πρόγραμμά σας θα πρέπει να τυπώνει τα bytes `a0 xor b0`, `a1 xor b1`, `a2 xor b2`. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ gcc -o r2d2 r2d2.c
$ hexdump -C message.txt
00000000  3b 04 9f da fe db 32 10  28 6a b7 2c c0 a8 bb ee  |;.....2.(j.,....|
00000010  12 fa 32 1a 3e c7 a8 ec  dd 56 a7 11 c7 e0 07 95  |..2.>....V......|
...
00000140  2d cd fe 38 24                                    |-..8$|
00000145
$ hexdump -C key.txt
00000000  68 70 ed b3 95 b2 5c 77  08 0c c5 43 ad 88 da ce  |hp....\w...C....|
00000010  74 95 40 6e 4c a2 db 9f  fd 3e ce 75 a3 85 69 b5  |t.@nL....>.u..i.|
...
```

```
00000140  48 bb 9b 4a 0a                                       |H..J.|
00000145
$ ./r2d2 message.txt key.txt
Striking from a fortress hidden among the billion stars of the galaxy, rebel
↪ spaceships have won their first victory in a battle with the powerful Imperial
↪ Starfleet. The EMPIRE fears that another defeat could bring a thousand more
↪ solar systems into the rebellion, and Imperial control over the galaxy would
↪ be lost forever.
```

Παρατηρήστε ότι το a0 είναι 0x3b και το b0 είναι 0x68 και το 0x3b xor 0x68 κάνει 0x53 ('S'). Το υπόλοιπο μήνυμα προκύπτει με αντίστοιχο τρόπο.

> Σημείωση: τα δύο hexdump (0x145 = 325 bytes το καθένα) συντομεύτηκαν εδώ· τα αρχεία είναι τα message.txt και key.txt.

Υπόδειξη Ανοίξτε και τα δύο αρχεία σε δυαδική λειτουργία (`"rb"`), διαβάστε ένα byte από το καθένα σε κάθε βήμα με `fgetc` και γράψτε το αποτέλεσμα του τελεστή ^ με `putchar`. Ελέγξτε το `EOF` πριν το χρησιμοποιήσετε ως byte, και αποφασίστε τι σημαίνει το να έχουν τα δύο αρχεία διαφορετικό μήκος.

A18.13 · Κόψιμο Αρχείων

A18.13 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #7 (Star Wars Themed), Θέμα 4* · ★☆☆ · programming · `exam-2023-fall-ex7-q4`

Πρόγραμμα: `cut.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα το οποίο παίρνει ως όρισμα το όνομα ενός αρχείου που περιέχει δεδομένα και έναν ακέραιο αριθμό από bytes τα οποία θέλουμε να κόψουμε από το τέλος του αρχείου και τυπώνει το νέο αρχείο - έχοντας ”κόψει” αυτά τα bytes - στην πρότυπη έξοδο. Τα αρχεία θα είναι μέχρι 100MB. Παράδειγμα επιτυχούς εκτέλεσης ακολουθεί:

```
$ gcc -o cut cut.c
$ echo hello world > msg.txt
$ hexdump -C msg.txt
00000000  68 65 6c 6c 6f 20 77 6f  72 6c 64 0a              |hello world.|
0000000c
$ ./cut msg.txt 6 > msg-cut.txt
$ hexdump -C msg-cut.txt
00000000  68 65 6c 6c 6f 20                                 |hello |
00000006
$ wc -c imperial.mp3
2846267 imperial.mp3
$ ./cut imperial.mp3 2256267 > imperial-cut.mp3
```

```
$ wc -c imperial-cut.mp3
590000 imperial-cut.mp3
```

Υπόδειξη Βρείτε πρώτα το μέγεθος του αρχείου (π.χ. `fseek` στο τέλος και `ftell`, σε αρχείο ανοιγμένο με `"rb"`), υπολογίστε πόσα bytes πρέπει να κρατήσετε και αντιγράψτε ακριβώς τόσα στην έξοδο, κατά προτίμηση σε μπλοκ με `fread`/`fwrite`. Τι κάνετε αν ζητηθεί να κοπούν περισσότερα bytes από όσα έχει το αρχείο, ή αν ο αριθμός είναι αρνητικός ή δεν είναι αριθμός;

A18.14 · Έλεγχος Εκτελέσιμου

A18.14 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #9, Θέμα 1* · ★☆☆ · programming · `exam-2023-fall-ex9-q1`

Πρόγραμμα: `elf.c`

Γράψτε ένα πρόγραμμα που παίρνει ως όρισμα το όνομα ενός αρχείου και αποφαίνεται για το αν το αρχείο είναι ένα Linux εκτελέσιμο ELF (Executable and Linking Format) ή όχι. Ένα αρχείο θεωρείται ELF όταν ξεκινάει με τους χαρακτήρες 0x7f, E, L, F. Οποιοδήποτε άλλο αρχείο δεν θεωρείται ELF. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o elf elf.c
$ ./elf foo
Error: Unable to read file
$ ./elf message.txt
message.txt is NOT an ELF file.
$ ./elf ./elf.c
./elf.c is NOT an ELF file.
$ ./elf ./elf
./elf is an ELF file.
$ ./elf /bin/ls
/bin/ls is an ELF file.
$ ./elf /lib/x86_64-linux-gnu/libc.so.6
/lib/x86_64-linux-gnu/libc.so.6 is an ELF file.
```

Υπόδειξη Ανοίξτε το αρχείο σε δυαδική μορφή (`"rb"`) και διαβάστε μόνο τα πρώτα 4 bytes. Σκεφτείτε τις ακραίες περιπτώσεις: αρχείο που δεν ανοίγει, αρχείο με λιγότερα από 4 bytes, λάθος πλήθος ορισμάτων.

A18.15 · Κρυμμένο Μήνυμα

A18.15 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #10, Θέμα 3* · ★★☆ · programming · `exam-2023-fall-ex10-q3`

Πρόγραμμα: `hidden.c`

Γράψτε ένα πρόγραμμα που παίρνει ως όρισμα το όνομα ενός αρχείου που περιέχει ένα κείμενο με λέξεις που αποτελούνται από λατινικούς χαρακτήρες (a-zA-Z) και θετικούς ακεραίους που αντιστοιχούν στην θέση της κάθε λέξης μέσα στο κείμενο και τυπώνει αυτές τις λέξεις με την σειρά που δίνονται ως ορίσματα στο πρόγραμμα. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o hidden hidden.c
$ cat message.txt
In the quiet glade, where whispers of the past weave,
Beneath the vault of heaven, a secret to conceive.
The oak, ancient and large, guards a mystery so deep,
Within its shadowed roots, a treasure lies asleep.
Hidden by time's embrace, under earth's tender keep,
Near the tree, love's promise, forever to reap
$ ./hidden message.txt 19 34 35 42 1 23 20 49
The treasure lies under the large oak tree
$ ./hidden message.txt 19 34 35 29
The treasure lies Within
```

Παρατηρούμε ότι η λέξη ”The” τυπώνεται πρώτη καθώς είναι η 20η λέξη στο κείμενο, δηλαδή βρίσκεται στην θέση 19 και αυτός είναι ο πρώτος ακέραιος που δόθηκε ως όρισμα στην γραμμή εντολών μετά το αρχείο. Αντίστοιχα τυπώνονται και οι υπόλοιπες λέξεις.

Υπόδειξη Χωρίστε το κείμενο σε λέξεις από λατινικούς χαρακτήρες (οτιδήποτε άλλο είναι διαχωριστικό, άρα το `time's` δίνει δύο λέξεις) και αποθηκεύστε τες σε δυναμικό πίνακα, ώστε να έχετε άμεση πρόσβαση με τη θέση. Οι θέσεις μετράνε από το 0· ελέγξτε θέσεις εκτός ορίων.

A18.16 · Μετρήσεις Θερμοκρασίας

A18.16 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #14, Θέμα 3* · ★★☆ · programming · `exam-2023-fall-ex14-q3`

Πρόγραμμα: `templog.c`

Γράψτε ένα πρόγραμμα το οποίο παίρνει ως όρισμα ένα αρχείο με μετρήσεις ενός θερμομέτρου στην διάρκεια του χρόνου το οποίο να τυπώνει στην πρότυπη έξοδο την μέση θερμοκρασία όλων των μετρήσεων και στην συνέχεια να τυπώνει όλες τις θερμοκρασίες ταξινομημένες με αύξουσα σειρά με βάση την απόστασή τους από την μέση θερμοκρασία. Το αρχείο με τις μετρήσεις θα έχει την μορφή ”ώρα μέτρησης,θερμοκρασία” όπου η θερμοκρασία θα είναι ένας αριθμός κινητής υποδιαστολής. Η μέση θερμοκρασία θέλουμε να τυπωθεί με 3 δεκαδικά ψηφία. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ cat log.txt
```

```
2024-03-07 10:18:27,205.00
2024-03-07 10:19:27,220.50
2024-03-07 10:20:27,221.00
2024-03-07 10:21:27,223.80
2024-03-07 10:22:27,229.32
2024-03-07 10:23:27,230.50
2024-03-07 10:24:27,223.00
2024-03-07 10:25:27,223.50
2024-03-07 10:26:27,220.75
2024-03-07 10:27:27,219.50
$ gcc -o templog templog.c
$ ./templog log.txt
Average temperature: 221.687
2024-03-07 10:20:27,221.00
2024-03-07 10:26:27,220.75
2024-03-07 10:19:27,220.50
2024-03-07 10:24:27,223.00
2024-03-07 10:25:27,223.50
2024-03-07 10:21:27,223.80
2024-03-07 10:27:27,219.50
2024-03-07 10:22:27,229.32
2024-03-07 10:23:27,230.50
2024-03-07 10:18:27,205.00
```

Υπόδειξη Διαβάστε κάθε γραμμή, χωρίστε στο κόμμα και κρατήστε την αρχική γραμμή μαζί με τη θερμοκρασία σε ένα `struct`. Υπολογίστε τον μέσο όρο και ταξινομήστε με `qsort` κατά `fabs(t - μέσος)`. Το πλήθος των μετρήσεων είναι άγνωστο, οπότε ο πίνακας μεγαλώνει με `realloc`.

A18.17 · Αλλαγή Μεγέθους

A18.17 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #4 (Rick Astley Themed), Θέμα 4* · ★★☆ · programming · `exam-2023-fall-ex4-q4`

Πρόγραμμα: `crop.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα το οποίο παίρνει 3 ορίσματα: (1) το όνομα ενός αρχείου, (2) την τοποθεσία (offset) σε αριθμό bytes μέσα στο αρχείο και (3) έναν αριθμό των δύο bytes (short) και τυπώνει στην πρότυπη έξοδο (stdout) το περιεχόμενο του αρχείου με τα δύο bytes στην τοποθεσία που αναφέρθηκε να έχουν αντικατασταθεί από τα bytes του αριθμού που δόθηκε από την γραμμή εντολών (όρισμα 3). Το υπόλοιπο αρχείο δεν πρέπει να αλλαχθεί. Τα αρχεία που θα δοθούν θα είναι μέχρι 100MB. Η εκτέλεση `./crop file 2 13362` σημαίνει διάβασε το αρχείο

`file` και στο byte στην θέση 2 τοποθέτησε τα δύο bytes του αριθμού 13362. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o crop crop.c
$ echo example > file
$ hexdump -C file
00000000  65 78 61 6d 70 6c 65 0a                           |example.|
00000008
$ ./crop file 2 13362 > file2
$ hexdump -C file2
00000000  65 78 34 32 70 6c 65 0a                           |ex42ple.|
00000008
$ file rick.jpg
rick.jpg: JPEG image data, JFIF standard 1.01, resolution (DPI), density 96x96,
 ↪  segment length 16, comment: "CREATOR: gd-jpeg v1.0 (using IJG JPEG v62),
 ↪  quality = 80", baseline, precision 8, 640x436, components 3
$ ./crop rick.jpg 224 218 > rick2.jpg
$ file rick2.jpg
rick2.jpg: JPEG image data, JFIF standard 1.01, resolution (DPI), density 96x96,
 ↪  segment length 16, comment: "CREATOR: gd-jpeg v1.0 (using IJG JPEG v62),
 ↪  quality = 80", baseline, precision 8, 640x218, components 3
```

Παρακάτω δείχνουμε το αποτέλεσμα της εκτέλεσης στην εικόνα rick.jpg (πριν):

Εικόνα: rick before

Και μετά:

Εικόνα: rick after

Υπόδειξη Ανοίξτε το αρχείο σε δυαδική λειτουργία (`"rb"`) και αντιγράψτε το στην έξοδο byte προς byte ή σε μπλοκ, αντικαθιστώντας μόνο τα bytes στις θέσεις `offset` και `offset + 1`. Γράψτε το 13362 στο δεκαεξαδικό και συγκρίνετε με το hexdump για να δείτε με ποια σειρά μπαίνουν τα δύο bytes (endianness)· τα βγάζετε με ολισθήσεις και μάσκες, όχι με `memcpy` από ένα `short`. Ελέγξτε ότι ο αριθμός χωράει σε δύο bytes και ότι το offset δεν ξεπερνά το μέγεθος του αρχείου.

A18.18 · Επιλογή

A18.18 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #5 (HP Themed), Θέμα 3* · ★★☆ · programming · `exam-2023-fall-ex5-q3`

Πρόγραμμα: `sorting.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα το οποίο παίρνει ως όρισμα το όνομα ενός αρχείου που περιέχει ονόματα παιδιών και το σκορ ταιριάσματος του κάθε παιδιού με έναν κοιτώνα του Hogwarts. Κάθε γραμμή περιέχει ακριβώς ένα όνομα ακολουθούμενο από παύλα (dash -) και από το σκορ ταιριάσματος (ένας αριθμός κινητής υποδιαστολής από το 0 μέχρι το 5). Η αντιστοιχία έχει ως εξής: 1 -> Gryffindor, 2 -> Hufflepuff, 3 -> Ravenclaw, 4 -> Slytherin. Το πρόγραμμα πρέπει να τυπώνει στην πρότυπη έξοδο το κάθε όνομα ακολουθούμενο από το όνομα του κοιτώνα που ταιριάζει καλύτερα. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ cat names.txt
Harry Potter - 1.01
Hermione Granger - 1
Draco Malfoy - 4.9
Cho Chang - 3.5
Cedric Diggory - 1.50001
Luna Lovegood - 2.76
$ gcc -o sorting sorting.c
$ ./sorting names.txt
Harry Potter -> Gryffindor
Hermione Granger -> Gryffindor
Draco Malfoy -> Slytherin
Cho Chang -> Ravenclaw
Cedric Diggory -> Hufflepuff
Luna Lovegood -> Ravenclaw
```

Υπόδειξη Διαβάστε το αρχείο γραμμή-γραμμή με `fgets` και χωρίστε όνομα και σκορ στην παύλα (το όνομα έχει κενά, οπότε η `strrchr` βοηθά), μετατρέποντας το σκορ με `strtod`. Ο «καλύτερος» κοιτώνας είναι ο πλησιέστερος στο σκορ: μελετήστε τα `4.9`, `3.5` και `1.50001` του παραδείγματος για να καταλάβετε τι γίνεται στα άκρα (0 και 5) και στις ισοπαλίες. Γραμμές χωρίς παύλα ή με σκορ εκτός [0, 5] είναι είσοδος εκτός προδιαγραφών.

A18.19 · Ταξινόμηση Αρχείων Καταγραφής

A18.19 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #6 (Crypto Themed), Θέμα 4* · ★★☆ · programming · `exam-2023-fall-ex6-q4`

Πρόγραμμα: `sortlog.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα το οποίο παίρνει ως όρισμα το όνομα ενός αρχείου που περιέχει δεδομένα από αρχεία καταγραφής (logs) και τυπώνει όλα τα μηνύματα ταξινομημένα με βαση την ημερομηνία καταγραφής σε αύξουσα σειρά (δηλαδή πρώτα οι ημερομηνίες που είναι νωρίτερα). Κάθε γραμμή καταγραφής έχει την μορφή ημερομηνίας με ακρίβεια λεπτού (μορφής ”χρόνος-μήνας-ημέρα T ώρες:λεπτά”) ακολουθούμενη από : και μετά το μήνυμα της καταγραφής. Παράδειγμα επιτυχούς εκτέλεσης ακολουθεί:

```
$ gcc -o sortlog sortlog.c
$ cat logs.txt
2024-03-02T13:20: Info: New user 'developer' created successfully.
2000-07-18T07:00: System startup sequence completed successfully.
2024-12-12T11:25: 'admin' initiated system reboot after security patch
↪  application.
2024-11-07T08:15: Scheduled backup started for database 'TestDB'.
2024-12-01T09:00: System startup sequence initiated.
2024-03-15T05:45: System shutdown sequence initiated.
2024-03-02T10:30: User 'admin' logged in from IP address 192.168.1.105.
2024-02-05T16:30: Warning: Disk usage exceeds 85% on drive C:.
2004-03-09T21:00: Error: Network timeout while connecting to server
↪  'backup.example.com'.
2025-03-04T14:45: Database connection established successfully.
$ ./sortlog logs.txt
2000-07-18T07:00: System startup sequence completed successfully.
2004-03-09T21:00: Error: Network timeout while connecting to server
↪  'backup.example.com'.
2024-02-05T16:30: Warning: Disk usage exceeds 85% on drive C:.
2024-03-02T10:30: User 'admin' logged in from IP address 192.168.1.105.
2024-03-02T13:20: Info: New user 'developer' created successfully.
2024-03-15T05:45: System shutdown sequence initiated.
2024-11-07T08:15: Scheduled backup started for database 'TestDB'.
2024-12-01T09:00: System startup sequence initiated.
2024-12-12T11:25: 'admin' initiated system reboot after security patch
↪  application.
2025-03-04T14:45: Database connection established successfully.
```

Υπόδειξη Διαβάστε όλες τις γραμμές σε έναν πίνακα από δείκτες σε δυναμικά δεσμευμένες συμβολοσειρές (δεν ξέρετε ούτε πόσες είναι ούτε πόσο μεγάλες) και ταξινομήστε τον με `qsort`. Παρατηρήστε ότι στη μορφή `EEEE-MM-ΗΗΤΩΩ:ΛΛ` η λεξικογραφική σύγκριση των πρώτων 16 χαρακτήρων συμπίπτει με τη χρονολογική. Ελέγξτε ότι κάθε γραμμή έχει πράγματι αυτή τη μορφή, και σκεφτείτε αν σας ενδιαφέρει η σειρά γραμμών με την ίδια ώρα.

A18.20 · Καλύτερο Ταίριασμα

A18.20 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #0 (Valentine's Themed), Θέμα 3* · ★★★ · programming · `exam-2023-fall-ex0-q3`

Πρόγραμμα: `bestmatch.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που διαβάζει ένα σύνολο από ονόματα ακολουθούμενα από έναν αριθμό που δηλώνει πόσο καλά μας ταιριάζει αυτό το άτομο και τυπώνει τα ονόματα από το μεγαλύτερο

στο μικρότερο ταξινομημένα. Τα δεδομένα περιέχονται σε αρχείο που δίνεται ως πρώτο όρισμα. Το αρχείο θα περιέχει κάθε όνομα σε μια γραμμή της μορφής ”όνομα,σκορ”. Ένα παράδειγμα εκτέλεσης ακολουθεί:

```
$ cat names.txt
Alex,3.4
Casey,9.1
Jordan,2
Taylor,5.3
Morgan,1.7
Riley Avery,6.16
Jamie,0
Quinn,-1.1
Skyler,5.88
$ gcc -o bestmatch bestmatch.c
$ ./bestmatch names.txt
Casey,9.1
Riley Avery,6.16
Skyler,5.88
Taylor,5.3
Alex,3.4
Jordan,2
Morgan,1.7
Jamie,0
Quinn,-1.1
```

Υπόδειξη Κρατήστε κάθε εγγραφή σε ένα `struct` (όνομα, σκορ) μέσα σε πίνακα που μεγαλώνει δυναμικά, αφού δεν ξέρετε πόσες γραμμές έχει το αρχείο, και ταξινομήστε με `qsort` και δική σας συνάρτηση σύγκρισης σε φθίνουσα σειρά. Τα ονόματα μπορεί να έχουν κενά, οπότε χωρίστε στο κόμμα. Το σκορ τυπώνεται όπως γράφτηκε (`2`, `6.16`), άρα ίσως σας βολεύει να κρατήσετε και το αρχικό κείμενο. Ελέγξτε το πλήθος των ορισμάτων, αν άνοιξε το αρχείο και τις κακοσχηματισμένες γραμμές.

A18.21 · Ταξινομώντας τα Άλματα

A18.21 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #15, Θέμα 3* · ★★★ · programming · `exam-2023-fall-ex15-q3`

Πρόγραμμα: `longjump.c`

Γράψτε ένα πρόγραμμα το οποίο παίρνει ως όρισμα ένα αρχείο με τα άλματα που έκανε κάθε αθλητής και εκτυπώνει τους αθλητές σε φθίνουσα σειρά με βάση τα άλματα που έκαναν. Το αρχείο θα είναι σε μορφή JSON, δηλαδή θα αρχίζει με τον χαρακτήρα '{' και θα τελειώνει με τον

χαρακτήρα '}' και για κάθε όνομα αθλητή θα τυπώνει `"όνομα":` μήκος όπου το μήκος θα είναι ένας αριθμός κινητής υποδιαστολής. Ανάμεσα σε κάθε αθλητή-άλμα θα υπάρχει ο χαρακτήρας ','. Δεν επιτρέπεται χρήση εξωτερικών βιβλιοθηκών. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ cat jumps.json
{
    "mcleod": 8.20,
    "furlani": 8.21,
    "montler": 7.80,
    "batz": 8.06,
    "williams": 7.83,
    "lawson": 8.05,
    "gayle": 7.89,
    "tentoglou": 8.22
}
$ gcc -o longjump longjump.c
$ ./longjump jumps.json
tentoglou: 8.22
furlani: 8.21
mcleod: 8.20
batz: 8.06
lawson: 8.05
gayle: 7.89
williams: 7.83
montler: 7.80
```

Υπόδειξη Διαβάστε το αρχείο χαρακτήρα-χαρακτήρα ή με `fscanf` και αναλύστε το: `{`, μετά ζεύγη `"όνομα": αριθμός` χωρισμένα με `,`, και `}`. Κρατήστε κάθε αθλητή σε `struct` και ταξινομήστε φθίνουσα με `qsort`. Αρχείο που δεν ακολουθεί τη μορφή είναι είσοδος εκτός προδιαγραφών.

A18.22 · Μίνι Βάση Δεδομένων

A18.22 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #2 (Pokémon Themed), Θέμα 3* · ★★★ · programming · `exam-2023-fall-ex2-q3`

Πρόγραμμα: `pokedex.c` (25 μονάδες)

Γράψτε ένα πρόγραμμα που παίρνει δύο ορίσματα: (1) το όνομα του αρχείο που περιέχει όλα τα pokémon που έχουμε πιάσει με όλα τα δεδομένα τους και (2) την ημερομηνία που αναζητούμε και επιστρέφει αναλυτικές πληροφορίες για όλα τα pokémon που πιάστηκαν *μετά* από εκείνη την ημερομηνία ταξινομημένα. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ cat pokedex.db
```

```
name, date_caught, level
geodude, 15-02-2024, 15
slowpoke, 16-01-2024, 20
pidgeotto, 19-02-2024, 5
bulbasaur, 01-01-2005, 37
onix, 31-07-2021, 21
gyarados, 20-12-2018, 19
$ gcc -o pokedex pokedex.c
$ ./pokedex pokedex.db 19-02-2024
pidgeotto was caught on 19-02-2024 and is level 5
$ ./pokedex pokedex.db 01-01-2024
slowpoke was caught on 16-01-2024 and is level 20
geodude was caught on 15-02-2024 and is level 15
pidgeotto was caught on 19-02-2024 and is level 5
```

Υπόδειξη Παραλείψτε τη γραμμή-επικεφαλίδα, διαβάστε κάθε εγγραφή σε ένα `struct` με `fgets` και `sscanf`, και μετατρέψτε την ημερομηνία `HH-MM-EEEE` σε έναν ακέραιο που συγκρίνεται σωστά (π.χ. έτος, μετά μήνας, μετά μέρα). Φιλτράρετε, ταξινομήστε κατά ημερομηνία με `qsort` και προσέξτε από το πρώτο παράδειγμα αν η ημερομηνία αναζήτησης περιλαμβάνεται. Επικυρώστε την ημερομηνία του ορίσματος (μορφή και εύρος μήνα/ημέρας).

A18.23 · Ταξινόμηση Πακέτων

A18.23 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #8, Θέμα 4* · ★★★ · programming · `exam-2023-fall-ex8-q4`

Πρόγραμμα: `packet.c`

Γράψτε ένα πρόγραμμα το οποίο παίρνει ως όρισμα το όνομα ενός αρχείου που περιέχει πακέτα ακολουθούμενα από κόμμα `,` και στην συνέχεια έναν αύξοντα αριθμό (packet id) που δείχνει που έπρεπε να βρίσκονται κανονικά στην σειρά των πακέτων και τυπώνει τα πακέτα ταξινομημένα σε αύξουσα σειρά με βάση τους αύξοντες αριθμούς. Παράδειγμα επιτυχούς εκτέλεσης ακολουθεί:

```
$ gcc -o packet packet.c
$ cat unordered.txt
Hello from five,5
This is the first packet,1
Continuing with seven,7
Almost there,9
Middle of the sequence,6
Starting to make sense,8
The very last packet,10
```

```
Second in line,2
Getting closer,4
Third packet coming through,3
$ ./packet unordered.txt
This is the first packet,1
Second in line,2
Third packet coming through,3
Getting closer,4
Hello from five,5
Middle of the sequence,6
Continuing with seven,7
Starting to make sense,8
Almost there,9
The very last packet,10
```

Υπόδειξη Αποθηκεύστε κάθε γραμμή μαζί με το packet id της (το κείμενο μετά το τελευταίο κόμμα) και ταξινομήστε με `qsort` και κατάλληλη συνάρτηση σύγκρισης. Δεν ξέρετε εκ των προτέρων πόσες γραμμές ή πόσο μεγάλη θα είναι η καθεμία, οπότε χρειάζεστε δυναμικούς πίνακες που μεγαλώνουν με `realloc`.

Κεφάλαιο 19: Δομές

Kahoot από το αμφιθέατρο (K19.1–K19.9)

K19.1 · Δικοί μας τύποι

K19.1 · *Kahoot «Δομές + Αρχεία», «Ταξινόμηση και Δομές»* · ★☆☆ · multiple-choice · 81% σωστές · `kahoot-custom-types`

Μπορώ να ορίσω δικούς μου τύπους στην C.

- True
- False

Υπόδειξη Σκεφτείτε τι κάνουν οι λέξεις-κλειδιά `struct` και `typedef`.

K19.2 · Σε τι χρησιμεύει το typedef

K19.2 · *Kahoot «Δομές + Αρχεία», «Ταξινόμηση και Δομές»* · ★☆☆ · multiple-choice · 79% σωστές · `kahoot-typedef-purpose`

Σε τι χρησιμοποιούμε το `typedef` στην C;

- Για να ορίσουμε συνώνυμα τύπων
- Για να ορίσουμε μεταβλητές
- Για να διαγράψουμε από την μνήμη ένα struct
- Για να γράψουμε λίγο παραπάνω κώδικα

Υπόδειξη Μετά από `typedef struct point Point;` τι σημαίνει η λέξη `Point`;

K19.3 · Δομή χωρίς αρχικοποίηση

K19.3 · *Kahoot «Δομές + Αρχεία», «Ταξινόμηση και Δομές»* · ★☆☆ · multiple-choice · 74% σωστές · `kahoot-struct-uninitialized`

Έστω `struct {int x; int y;} point;` (τοπική μεταβλητή). Ποια είναι η τιμή του `point.y`;

- 0xFFFFFFFF
- 0
- 1
- Ότι έτυχε να έχει η μνήμη σε εκείνη την θέση

Υπόδειξη Μια τοπική μεταβλητή χωρίς αρχικοποιητή δεν παίρνει καμία τιμή αυτόματα, είτε είναι `int` είτε δομή.

K19.4 · sizeof ενός typedef πίνακα

K19.4 · *Kahoot «Δομές + Αρχεία», «Ταξινόμηση και Δομές»* · ★★☆ · multiple-choice · 66% σωστές · `kahoot-typedef-array-size`

Έστω `typedef double trouble[1024];` και `sizeof(double) == 8`. Ποιο είναι το `sizeof(trouble)`;

- 1024
- 4096
- 8192
- Εξαρτάται

Συχνή παρανόηση Το 24% επέλεξε «Εξαρτάται», ενώ το `typedef` απλώς δίνει όνομα στον τύπο «πίνακας 1024 `double`», που έχει σταθερό μέγεθος.

Υπόδειξη Το `typedef` δεν φτιάχνει μεταβλητή, μόνο ένα συνώνυμο για τον τύπο· ποιος είναι αυτός ο τύπος και πόσα bytes πιάνει;

K19.5 · Σύγκριση δομών με ==

K19.5 · *Kahoot «Δομές + Αρχεία», «Ταξινόμηση και Δομές»* · ★★☆ · multiple-choice · 57% σωστές · `kahoot-struct-comparison`

Έστω `struct {int x;} cafe = {1}, bar = {2};`. Η παράσταση `cafe == bar` αποτιμάται σε:

- 1
- 0
- Δεν αποτιμάται
- 42

Συχνή παρανόηση Το 24% επέλεξε `0`, υποθέτοντας ότι το `==` συγκρίνει τις δομές πεδίο προς πεδίο, όπως το `=` τις αντιγράφει.

Υπόδειξη Η ανάθεση δομών επιτρέπεται, αλλά ελέγξτε ποιοι τελεστές ορίζονται για τύπους δομής· τι θα έλεγε ο μεταγλωττιστής;

K19.6 · Μέγεθος δομής και padding

K19.6 · *Kahoot «Δομές + Αρχεία», «Ταξινόμηση και Δομές»* · ★★☆ · multiple-choice · 52% σωστές · `kahoot-struct-padding-size`

Ποιο είναι το μέγεθος του `struct {char c; int x; double d;}`;

- 1 + 4 + 8 = 13
- 1 + 8 + 8 = 17
- 4 + 4 + 8 = 16
- Εξαρτάται

Συχνή παρανόηση Το 24% επέλεξε `4 + 4 + 8 = 16`, την τιμή που δίνει συνήθως ένας 64-bit υπολογιστής· όμως το padding ανάμεσα στα πεδία και τα μεγέθη των τύπων δεν ορίζονται από το πρότυπο.

Υπόδειξη Θυμηθείτε ότι ο μεταγλωττιστής μπορεί να προσθέσει padding για τη στοίχιση των πεδίων, και ότι ούτε το `sizeof(int)` είναι ίδιο παντού.

K19.7 · Ανάθεση δομών

K19.7 · *Kahoot «Δομές + Αρχεία», «Ταξινόμηση και Δομές»* · ★★☆ · multiple-choice · 49% σωστές · `kahoot-struct-assignment`

Έστω `struct {int x;} cafe = {1}, bar = {2}; bar = cafe;`. Η παράσταση `bar.x` αποτιμάται σε:

- 1
- 2
- Δεν αποτιμάται
- 42

Συχνή παρανόηση Το 28% επέλεξε 2, σαν η ανάθεση `bar = cafe` να μην άλλαζε τα πεδία της `bar`· στην πραγματικότητα η ανάθεση δομών αντιγράφει όλα τα πεδία.

Υπόδειξη Αναρωτηθείτε αν ο τελεστής = επιτρέπεται ανάμεσα σε δύο δομές του ίδιου τύπου και τι αντιγράφει.

K19.8 · Ο τελεστής ->

K19.8 · *Kahoot «Προχωρημένες Δομές» (διάλεξη 20), «Προχωρημένες Δομές #2»* · ★★☆ · multiple-choice · 48% σωστές · `kahoot-arrow-operator`

Η έκφραση `robin->hood` στην C είναι ισοδύναμη με:

- `robin.(*hood)`
- `*robin.hood`
- `hood.robin`
- `(*robin).hood`

Υπόδειξη Το `robin` είναι δείκτης σε δομή· πρώτα πρέπει να φτάσετε στη δομή και μετά στο πεδίο, και θυμηθείτε ότι η `.` έχει μεγαλύτερη προτεραιότητα από το *.

K19.9 · Μερική αρχικοποίηση δομής

K19.9 · *Kahoot «Δομές + Αρχεία», «Ταξινόμηση και Δομές»* · ★★☆ · multiple-choice · 42% σωστές · `kahoot-struct-partial-init`

Έστω `struct {int x; int y;} point = {42};`. Ποια είναι η τιμή του `point.y`;

- 42
- 0

- 1
- Ότι έτυχε να έχει η μνήμη σε εκείνη την θέση

Συχνή παρανόηση Το 26% επέλεξε 42, πιστεύοντας ότι η μία τιμή αντιγράφεται σε όλα τα πεδία, και άλλο 26% «ό,τι έτυχε να έχει η μνήμη», ξεχνώντας ότι με αρχικοποιητή τα πεδία που λείπουν μηδενίζονται.

Υπόδειξη Σκεφτείτε τι κάνει η C με τα πεδία που δεν αναφέρονται μέσα σε έναν αρχικοποιητή `{...}`, όπως ακριβώς και στους πίνακες.

Ζέσταμα: από τις διαφάνειες (A19.1–A19.8)

A19.1 · Ανάθεση και χρήση πεδίων δομής

A19.1 · *Διάλεξη 19, διαφάνειες 12–13* · ★☆☆ · trace · `slides-lec19-field-assignment`

Με τη δομή

```
struct student {
  char first_name[128];
  char last_name[128];
  unsigned int year;
  double grade;
};
```

τι τυπώνει αυτό το πρόγραμμα;

```
int main() {
  struct student st1;
  st1.year = 1;
  st1.grade = 7.54;
  strncpy(st1.first_name, "Thanos", sizeof(st1.first_name) - 1);
  st1.first_name[sizeof(st1.first_name) - 1] = '\0';
  strncpy(st1.last_name, "Barbounis", sizeof(st1.last_name) - 1);
  st1.last_name[sizeof(st1.last_name) - 1] = '\0';
  printf("%s %s: %f [%u year]\n", st1.first_name, st1.last_name, st1.grade,
 ↪  st1.year);
  return 0;
}
```

Υπόδειξη Κάθε πεδίο συμπεριφέρεται όπως μια μεταβλητή του τύπου του. Προσέξτε πόσα δεκαδικά τυπώνει το `%f` χωρίς ακρίβεια.

A19.2 · Πώς αναπαριστούμε 100 φοιτητές;

A19.2 · *Διάλεξη 19, διαφάνειες 6–7* · ★☆☆ · short-answer · `slides-lec19-many-students`

Για ένα πρόγραμμα διαχείρισης μιας λίστας φοιτητών θα χρειαστείτε μεταβλητές για τα ακόλουθα:

```
char first_name[128];
char last_name[128];
unsigned int year;
double grade;
```

Έστω ότι έχουμε 100 φοιτητές, τι θα κάνουμε για να τους αναπαραστήσουμε στο πρόγραμμά μας;

Υπόδειξη Σκεφτείτε πρώτα τη λύση μόνο με πίνακες: πόσους πίνακες χρειάζεστε και τι πρέπει να ισχύει για τη θέση `i` σε καθέναν; Μετά σκεφτείτε πώς θα μπορούσαν τα τέσσερα πεδία ενός φοιτητή να γίνουν ένας τύπος.

A19.3 · Ανάθεση με δομές

A19.3 · *Διάλεξη 19, διαφάνειες 29–30* · ★☆☆ · trace · `slides-lec19-struct-assign`

Για να αντιγραφούν τα περιεχόμενα μιας δομής σε μια άλλη, χρησιμοποιούμε τον τελεστή ανάθεσης. Τι θα τυπώσει αυτό το πρόγραμμα;

```
#include <stdio.h>
struct point { int x;  int y; };
int main() {
  struct point pt1 = { 3, 4 };
  struct point pt2;
  printf("%d %d\n", pt2.x, pt2.y);
  pt2 = pt1;
  printf("%d %d\n", pt2.x, pt2.y);
  return 0;
}
```

Υπόδειξη Εξετάστε χωριστά τα δύο `printf`: τι περιέχει το `pt2` πριν από την ανάθεση, αφού δεν αρχικοποιήθηκε; Και τι αντιγράφει η ανάθεση ανάμεσα σε δύο δομές ίδιου τύπου;

A19.4 · Σύγκριση με δομές

A19.4 · *Διάλεξη 19, διαφάνεια 32* · ★☆☆ · short-answer · `slides-lec19-struct-compare`

Μπορώ να συγκρίνω δομές με τελεστές σύγκρισης;

```
struct point pt1 = { 3, 4 };
struct point pt2;
pt2 = pt1;
if (pt1 == pt2) printf("impossible\n");
```

Υπόδειξη Δοκιμάστε να το μεταγλωττίσετε και διαβάστε το μήνυμα του gcc. Αν η ανάθεση επιτρέπεται, ισχύει το ίδιο για το ==; Πώς αλλιώς μπορείτε να ελέγξετε αν δύο σημεία είναι ίδια;

A19.5 · Δείκτες σε δομές: διαφορά ημερομηνιών

A19.5 · *Διάλεξη 19, διαφάνειες 42–43* · ★★☆ · trace · `slides-lec19-date-pointers`

Οι δείκτες μπορούν να συνδυαστούν με δομές όπως όλοι οι άλλοι τύποι. Τι θα τυπώσει το παρακάτω πρόγραμμα;

```
#include <stdio.h>
#include <stdlib.h>
typedef struct { int day; int month; int year; } Date;
int main() {
  Date d1 = {1, 10, 2023};
  Date * d2 = &d1;
  (*d2).day = 2;
  Date * d3 = malloc(sizeof(Date));
  *d3 = *d2;
  (*d3).month = 12; (*d3).day = 11;
  printf("Diff: %d/%d/%d\n", (*d3).day - d1.day, (*d3).month - d1.month,
 ↪  (*d3).year - d1.year);
  return 0;
}
```

Υπόδειξη Ξεχωρίστε πού δείχνει κάθε δείκτης: το d2 σε υπάρχουσα μεταβλητή, το d3 σε νέα μνήμη. Αλλάζει το d1 όταν γράφετε μέσω του d2; Τι αντιγράφει το `*d3 = *d2`;

A19.6 · Padding: η σειρά των πεδίων μετράει

A19.6 · *Διάλεξη 19, διαφάνειες 25–27* · ★★☆ · trace · `slides-lec19-padding-interleaved`

Τι θα τυπώσει αυτό το πρόγραμμα (μεταγλώττιση με `gcc -m32 -o struct3 struct3.c`);

```
#include <stdio.h>
int main() {
```

```
  struct pixel_tag {
     char red; int alpha;
     char green; int beta;
     char blue; int gamma;
  } pixel;
  printf("%zu\n", sizeof(pixel));
  return 0;
}
```

Υπόδειξη Ζωγραφίστε τη δομή byte-byte με τη σειρά της δήλωσης και βάλτε κάθε `int` σε διεύθυνση πολλαπλάσιο του 4. Πόσα από τα bytes χρησιμοποιούνται στην πραγματικότητα;

A19.7 · Padding: το μέγεθος ενός pixel

A19.7 · *Διάλεξη 19, διαφάνειες 22–24* · ★★☆ · trace · `slides-lec19-padding-pixel`

Τι θα τυπώσει αυτό το πρόγραμμα (μεταγλώττιση με `gcc -m32 -o struct2 struct2.c`);

```
#include <stdio.h>
int main() {
  struct pixel_tag {
      char red;
      char green;
      char blue;
      int alpha;
  } pixel = {0xFF, 0xFF, 0xFF, 42};
  printf("%zu\n", sizeof(pixel));
  return 0;
}
```

Υπόδειξη Τα πεδία πιάνουν 7 bytes, αλλά η απάντηση δεν είναι 7. Σκεφτείτε σε ποια διεύθυνση προτιμά ο επεξεργαστής να ξεκινά ένα `int` (memory alignment).

A19.8 · Το μέγεθος του struct student

A19.8 · *Διάλεξη 19, διαφάνειες 20–21, 28* · ★★☆ · trace · `slides-lec19-sizeof-student`

```
struct student {
  char first_name[128];
  char last_name[128];
  unsigned int year;
  double grade;
```

```
};
```

Τι θα τυπώσει το ακόλουθο, όταν το πρόγραμμα μεταγλωττιστεί με `gcc -m32`;

```
printf("%zu\n", sizeof(struct student));
```

Ποιο είναι, γενικά, το μέγεθος του `struct student`;

Υπόδειξη Αθροίστε τα μεγέθη των πεδίων με τη σειρά της δήλωσης. Για το γενικό ερώτημα, σκεφτείτε αν ο μεταγλωττιστής είναι υποχρεωμένος να τα τοποθετήσει κολλητά (τι αλλάζει για ένα `double` σε σύστημα 64 bit;).

Εργαστήριο (A19.9–A19.10)

A19.9 · Δομές και συναρτήσεις

A19.9 · *Εργαστήριο 9, Άσκηση 1* · ★☆☆ · programming · `lab-lab09-point`

1.1 Κατασκευάστε το αρχείο `point.c` και ορίστε σε αυτό τη δομή `point` που αποθηκεύει τις συντεταγμένες (τύπου `double`) ενός σημείου στον δισδιάστατο χώρο.

1.2 Ορίστε τη συνάρτηση:

```
struct point middle(struct point a, struct point b);
```

Η συνάρτηση θα υπολογίζει και θα επιστρέφει το σημείο που βρίσκεται στο μέσο του ευθυγράμμου τμήματος με άκρα τα σημεία `a` και `b`.

1.3 Υπολογίστε και τυπώστε το μέσο του ευθυγράμμου τμήματος με άκρα τα σημεία `(1.2,5.4)` και `(7.3,1.8)`.

Υπόδειξη Οι δομές περνούν και επιστρέφονται με τιμή, όπως οι απλές μεταβλητές: η `middle` φτιάχνει μια τοπική `struct point`, συμπληρώνει τα πεδία της με τον μέσο όρο των αντίστοιχων συντεταγμένων και την επιστρέφει. Στα πεδία φτάνετε με τον τελεστή `..`

A19.10 · Δομές και δείκτες

A19.10 · *Εργαστήριο 9, Άσκηση 2* · ★★☆ · programming · `lab-lab09-person`

2.1 Δημιουργήστε το αρχείο `person.c` και ορίστε τη δομή `person` που αποθηκεύει το όνομα, το επώνυμο και το πατρώνυμο ενός ατόμου.

```
struct person {
    char *fname;
    char *lname;
```

```
    char *mname;
};
```

2.2 Κατασκευάστε τη συνάρτηση:

```
struct person *person_init(char *firstname, char *lastname, char *middlename);
```

Η συνάρτηση θα δεσμεύει χώρο για μία δομή τύπου `person`, θα την αρχικοποιεί και θα επιστρέφει τη διεύθυνσή της. Καλέστε τη συνάρτηση από την `main` για να καταχωρίσετε τα στοιχεία του πατέρα σας.

2.3 Ορίστε τη συνάρτηση:

```
struct person *childof(struct person father, char *newname);
```

Η συνάρτηση θα καταχωρεί τα στοιχεία ενός παιδιού με μικρό όνομα `newname`, χρησιμοποιώντας τον πατέρα του `father`. Καλέστε τη συνάρτηση από την `main` για να καταχωρίσετε τα στοιχεία σας.

Υπόδειξη Τα πεδία είναι δείκτες: αποφασίστε αν θα αντιγράψετε τα ονόματα σε δική τους δυναμική μνήμη (`malloc` με `strlen + 1` και `strcpy`) ή αν απλώς θα κρατήσετε τους δείκτες που σας δόθηκαν, και σκεφτείτε τι γίνεται στη δεύτερη περίπτωση αν αλλάξει ο αρχικός πίνακας. Στην `childof`, ποιο όνομα του πατέρα γίνεται πατρώνυμο του παιδιού και ποιο κρατιέται ως επώνυμο;

Θέματα εξετάσεων (A19.11–A19.12)

A19.11 · Πρωτάθλημα

A19.11 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #13, Θέμα 3* · ★★★ · programming · `exam-2023-fall-ex13-q3`

Πρόγραμμα: `league.c`

Γράψτε ένα πρόγραμμα το οποίο παίρνει ως όρισμα ένα αρχείο με αποτελέσματα αγώνων ανάμεσα σε ομάδες και στο τέλος τυπώνει όλες τις ομάδες ταξινομημένες σε αύξουσα σειρά με βάση την βαθμολογία τους. Σε περίπτωση ισοβαθμίας, χρησιμοποιούμε την διαφορά γκολ για να αποφασίσουμε ποια ομάδα είναι πρώτη. Μια ομάδα παίρνει 3 βαθμούς σε περίπτωση νίκης, 1 βαθμό σε περίπτωση ισοπαλίας και 0 βαθμούς εάν χάσει. Το αρχείο θα περιέχει έναν αγώνα ανά γραμμή και η μορφή του θα είναι: ΟΜΑΔΑ1-ΟΜΑΔΑ2,ΓΚΟΛ1-ΓΚΟΛ2 όπου ΓΚΟΛ1 είναι τα γκολ που έβαλε η ΟΜΑΔΑ1 και αντίστοιχα για τα ΓΚΟΛ2/ΟΜΑΔΑ2. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ gcc -o league league.c
$ cat games.txt
ΑΕΚ-Πανιώνιος,4-2
```

```
Ιωνικός-Παναχαϊκή,3-0
Ξάνθη-Ολυμπιακός,1-5
Παναθηναϊκός-Ακράτητος,3-1
Άρης-Ηρακλής,2-2
ΠΑΟΚ-ΠΑΣ Γιάννινα,2-3
ΠΑΟΚ-Παναχαϊκή,2-0
Ξάνθη-Ηρακλής,1-2
Ολυμπιακός-ΠΑΣ Γιάννινα,1-0
Πανιώνιος-Παναθηναϊκός,0-2
Άρης-ΑΕΚ,2-2
Ιωνικός-Ακράτητος,3-3
$ ./league games.txt
Ολυμπιακός, 6, (6 - 1)
Παναθηναϊκός, 6, (5 - 1)
Ιωνικός, 4, (6 - 3)
ΑΕΚ, 4, (6 - 4)
Ηρακλής, 4, (4 - 3)
ΠΑΟΚ, 3, (4 - 3)
ΠΑΣ Γιάννινα, 3, (3 - 3)
Άρης, 2, (4 - 4)
Ακράτητος, 1, (4 - 6)
Πανιώνιος, 0, (2 - 6)
Ξάνθη, 0, (2 - 7)
Παναχαϊκή, 0, (0 - 5)
```

Υπόδειξη Κρατήστε ένα `struct` ανά ομάδα (όνομα, βαθμοί, γκολ υπέρ, γκολ κατά) σε δυναμικό πίνακα, και για κάθε αγώνα βρείτε ή προσθέστε τις δύο ομάδες. Τα ονόματα είναι UTF-8 και μπορεί να έχουν κενά, οπότε χωρίστε τη γραμμή στα `-` και `,` και όχι στα κενά. Ταξινομήστε με `qsort` με βαθμούς και μετά διαφορά γκολ.

A19.12 · World Cup 2026

A19.12 · *Εξέταση Ιουνίου 2026, Θέμα 4* · ★★★ · programming · `exam-2026-jun-q4`

World Cup 2026 (35 Μονάδες)

Το παγκόσμιο κύπελλο ποδοσφαίρου μόλις έφτασε! Όλοι και όλες είμαστε έτοιμοι/ες για να δούμε τους αγώνες αλλά διαπιστώσαμε πως κάτι λείπει: δεν έχουμε ένα γρήγορο τρόπο να υπολογίζουμε την κατάταξη των ομάδων μετά τα αποτελέσματα των αγώνων. Αυτό είναι και το πρόβλημα που καλούμαστε να λύσουμε σε αυτό το θέμα.

Γράψτε ένα πρόγραμμα C το οποίο διαβάζει από την πρότυπη είσοδο όλα τα αποτελέσματα αγώνων ανάμεσα σε ομάδες και στο τέλος τυπώνει τις ομάδες ταξινομημένες σε φθίνουσα σειρά

με βάση την βαθμολογία τους. Σε περίπτωση ισοβαθμίας, χρησιμοποιούμε την διαφορά γκολ για να αποφασίσουμε ποια ομάδα είναι πρώτη (σε περίπτωση ίδιας διαφοράς γκολ, οι ομάδες ταξινομούνται αλφαβητικά). Μια ομάδα παίρνει 3 βαθμούς σε περίπτωση νίκης, 1 βαθμό σε περίπτωση ισοπαλίας και 0 βαθμούς εάν χάσει. Το αρχείο θα περιέχει έναν αγώνα ανά γραμμή και η μορφή του θα είναι: `ΟΜΑΔΑ1-ΟΜΑΔΑ2,ΓΚΟΛ1-ΓΚΟΛ2` όπου ΓΚΟΛ1 είναι τα γκολ που έβαλε η ΟΜΑΔΑ1 και αντίστοιχα για τα ΓΚΟΛ2/ΟΜΑΔΑ2.

Ποια είναι η χρονική και η χωρική πολυπλοκότητα του αλγορίθμου σας (10/30 της βαθμολογίας); Αιτιολογήστε την απάντησή σας και εξηγήστε την σημασία κάθε μεταβλητής που θα χρησιμοποιήσετε.

Παράδειγμα εκτέλεσης ακολουθεί:

```
$ cat games.txt
Germany-South Korea,4-2
Mexico-New Zealand,3-0
Ecuador-France,1-5
Argentina-Morocco,3-1
Turkey-Croatia,2-2
Portugal-Senegal,2-3
Portugal-New Zealand,2-0
Ecuador-Croatia,1-2
France-Senegal,1-0
South Korea-Argentina,0-2
Turkey-Germany,2-2
Mexico-Morocco,3-3
$ ./scoreboard < games.txt
France, 6, (6 - 1)
Argentina, 6, (5 - 1)
Mexico, 4, (6 - 3)
Germany, 4, (6 - 4)
Croatia, 4, (4 - 3)
Portugal, 3, (4 - 3)
Senegal, 3, (3 - 3)
Turkey, 2, (4 - 4)
Morocco, 1, (4 - 6)
South Korea, 0, (2 - 6)
Ecuador, 0, (2 - 7)
New Zealand, 0, (0 - 5)
```

Υπόδειξη Κρατήστε έναν πίνακα από `struct` (όνομα, βαθμοί, γκολ υπέρ, γκολ κατά) και, για κάθε γραμμή, βρείτε ή προσθέστε τις δύο ομάδες. Τα ονόματα περιέχουν κενά (π.χ. "South Korea"), οπότε διαβάστε ολόκληρη τη γραμμή με `fgets` και χωρίστε την στα `-` και `,` αντί για `%s`. Για την ταξινόμηση γράψτε μια συνάρτηση σύγκρισης με τρία κριτήρια (βαθμοί, διαφορά γκολ,

`strcmp`) για το `qsort`, και σκεφτείτε το κόστος της αναζήτησης ομάδας για την πολυπλοκότητα.

Κεφάλαιο 20: Προχωρημένες Δομές

Kahoot από το αμφιθέατρο (K20.1–K20.8)

K20.1 · Κόμβος με left και right

K20.1 · *Kahoot «Προχωρημένες Δομές #2»* · ★★☆ · multiple-choice · 67% σωστές · `kahoot-node-binary-tree`

Η δομή `struct node {int x; struct node *left; struct node *right;}` αντιστοιχεί σε:

- single linked list (απλά συνδεδεμένη λίστα)
- cyclic graph (κυκλικός γράφος)
- binary tree (δυαδικό δέντρο)
- bitfield (πεδίο δυφίων)

Υπόδειξη Μετρήστε πόσους δείκτες προς άλλους κόμβους έχει κάθε κόμβος και τι δηλώνουν τα ονόματά τους.

K20.2 · Σε τι χρησιμεύει η ένωση

K20.2 · *Kahoot «Προχωρημένες Δομές» (διάλεξη 20), «Προχωρημένες Δομές #2»* · ★★☆ · multiple-choice · 65% σωστές · `kahoot-union-purpose`

Σε τι χρησιμεύει η ένωση (`union`);

- Είναι ο μόνος τρόπος να δεσμεύσουμε χώρο για πίνακες
- Θέτει όριο στον αριθμό των πεδίων μιας δομής
- Είναι εξαιρετική ομάδα
- Εξοικονομεί χώρο μνήμης

Υπόδειξη Μια ένωση κρατάει μία τιμή τη φορά από πολλούς πιθανούς τύπους· τι κερδίζουμε σε σχέση με μια δομή με όλα αυτά τα πεδία;

K20.3 · Δομή ή ένωση;

K20.3 · *Kahoot «Προχωρημένες Δομές» (διάλεξη 20), «Προχωρημένες Δομές #2»* · ★★☆ · multiple-choice · 63% σωστές · `kahoot-struct-vs-union`

Ποια η διαφορά ανάμεσα σε μια δομή (`struct`) και μια ένωση (`union`);

- Στην δομή κάθε πεδίο έχει την δική του μνήμη, στην ένωση την μοιράζονται
- Η δομή μπορεί να έχει πολλά πεδία, η ένωση μόνο ένα
- Η δομή μπορεί να έχει άλλες δομές εντός, ενώ η ένωση όχι
- Trick question, δεν υπάρχει διαφορά

Υπόδειξη Συγκρίνετε το `sizeof` μιας δομής και μιας ένωσης με τα ίδια πεδία: τι σας λέει για το πού βρίσκεται κάθε πεδίο;

K20.4 · Αυτοαναφορική δομή

K20.4 · *Kahoot «Προχωρημένες Δομές» (διάλεξη 20), «Προχωρημένες Δομές #2»* · ★★☆ · multiple-choice · 56% σωστές · `kahoot-self-referential-struct`

Μια αυτοαναφορική δομή:

- περιέχει τον εαυτό της
- είναι ένα πεδίο τύπου δείκτη
- περιέχει ένα πεδίο τύπου δείκτη που δείχνει σε δομή ίδιου τύπου
- περιέχει αυτοαναφορικούς ακεραίους

Συχνή παρανόηση Το 22% επέλεξε «περιέχει τον εαυτό της», κάτι αδύνατο: μια δομή που περιέχει αντίγραφο του εαυτού της θα είχε άπειρο μέγεθος.

Υπόδειξη Σκεφτείτε πόσα bytes θα χρειαζόταν μια δομή που περιέχει ολόκληρη μια δομή του ίδιου τύπου, και τι μπορεί να περιέχει αντί γι' αυτό.

K20.5 · Κόμβος με next

K20.5 · *Kahoot «Προχωρημένες Δομές #2»* · ★★☆ · multiple-choice · 52% σωστές · `kahoot-node-linked-list`

Το `struct node {int x; struct node *next;}` είναι δομικό στοιχείο για:

- union (ένωση)
- binary tree (δυαδικό δέντρο)
- single linked list (απλά συνδεδεμένη λίστα)
- graph (γράφος)

Υπόδειξη Κάθε κόμβος δείχνει σε έναν μόνο επόμενο κόμβο· τι σχήμα φτιάχνουν τέτοιοι κόμβοι αν τους αλυσοδέσετε;

K20.6 · Τιμές απαρίθμησης

K20.6 · *Kahoot «Προχωρημένες Δομές» (διάλεξη 20), «Προχωρημένες Δομές #2»* · ★★☆ · multiple-choice · 48% σωστές · `kahoot-enum-values`

Έστω `enum TrafficLight {Red = 0, Yellow, Green};`. Ποια η τιμή του `Green`;

- 0
- 1
- 2
- 'G'

Υπόδειξη Οι σταθερές μιας απαρίθμησης είναι ακέραιοι· αν δεν δοθεί τιμή, κάθε σταθερά παίρνει την τιμή της προηγούμενης συν 1.

K20.7 · Περιορισμοί πεδίων bit

K20.7 · *Kahoot «Προχωρημένες Δομές» (διάλεξη 20), «Προχωρημένες Δομές #2»* · ★★☆ · multiple-choice · 44% σωστές · `kahoot-bitfield-limits`

Τι δεν μπορώ να κάνω σε bitfields (πεδία δυφίων) στην C;

- Να αναφερθώ στη διεύθυνση ενός bitfield
- Να βρω το μέγεθος με `sizeof`
- Να αποθηκεύσω τιμές μεγαλύτερες του 2^n - 1, όπου n ο αριθμός των bits
- Όλα τα παραπάνω

Συχνή παρανόηση Το 26% επέλεξε μόνο το `sizeof`, χωρίς να δει ότι ένα πεδίο bit δεν έχει ούτε δική του διεύθυνση (δεν ξεκινάει απαραίτητα σε byte) ούτε χωράει τιμές πέρα από τα n bits του.

Υπόδειξη Ένα πεδίο bit μπορεί να πιάνει λιγότερο από ένα byte· σκεφτείτε τι σημαίνει αυτό για τη διεύθυνση, το μέγεθος και το εύρος τιμών του.

K20.8 · Μέγεθος ένωσης

K20.8 · *Kahoot «Προχωρημένες Δομές» (διάλεξη 20), «Προχωρημένες Δομές #2»* · ★★★ · multiple-choice · 36% σωστές · `kahoot-union-size`

Έστω `union station {int x; char name[20]; double d;};`. Ποιο είναι το `sizeof(union station)`;

- sizeof(double)
- sizeof(int) + sizeof(name) + sizeof(double)
- max(sizeof(x), sizeof(name), sizeof(double))
- 42

Συχνή παρανόηση Το 22% επέλεξε `sizeof(double)`, θεωρώντας τον `double` το μεγαλύτερο πεδίο, ενώ ο πίνακας `name` πιάνει 20 bytes· και το 18% πρόσθεσε τα μεγέθη, όπως σε μια δομή.

Υπόδειξη Σε μια ένωση όλα τα πεδία ξεκινούν από την ίδια διεύθυνση· πόσος χώρος χρειάζεται ώστε να χωράει οποιοδήποτε από αυτά;

Ζέσταμα: από τις διαφάνειες (A20.1–A20.12)

A20.1 · Μέγεθος δομής με πεδία bit

A20.1 · *Διάλεξη 20, διαφάνεια 7* · ★☆☆ · trace · `slides-lec20-bitfield-sizeof`

Τι τυπώνει το παρακάτω κομμάτι κώδικα; Τι θα τύπωνε αν οι τρεις τύποι `int` γίνονταν `char`; Και αν, επιπλέον, αφαιρούσαμε τα `: 1`, `: 3`, `: 4`;

```
struct student_status {
  int registered : 1;
  int year : 3;
  int grade : 4;
};
printf("%zu\n", sizeof(struct student_status));
```

Υπόδειξη Μετρήστε πόσα bits χρειάζονται συνολικά τα τρία πεδία και σκεφτείτε σε πόσες «μονάδες» του τύπου τους χωράνε. Χωρίς πεδία bit, κάθε μέλος πιάνει ολόκληρο τον τύπο του.

A20.2 · Βρόχος με απαρίθμηση

A20.2 · *Διάλεξη 20, διαφάνεια 27* · ★☆☆ · trace · `slides-lec20-enum-loop`

Τι θα τυπώσει το ακόλουθο πρόγραμμα;

```
#include <stdio.h>

enum weekday {Mon = 1, Tue, Wed, Thu, Fri, Sat, Sun};
```

```c
int main() {
  for (enum weekday day = Mon; day <= Sun; day++) {
    if (day == Mon || day == Fri)
      printf("We have class on day # %d of the week\n", day);
  }
  return 0;
}
```

Υπόδειξη Βρείτε πρώτα την ακέραια τιμή κάθε σταθεράς, αφού τώρα η Mon έχει ρητή τιμή. Μετά τρέξτε τον βρόχο σαν να ήταν ένας απλός ακέραιος μετρητής.

A20.3 · Τιμές απαρίθμησης

A20.3 · *Διάλεξη 20, διαφάνεια 25* · ★☆☆ · trace · `slides-lec20-enum-values`

Τι θα τυπώσει το ακόλουθο πρόγραμμα;

```c
#include <stdio.h>

enum weekday {Mon, Tue, Wed, Thu, Fri, Sat, Sun};

int main() {
  enum weekday day1 = Mon, day2;
  day2 = Fri;
  printf("%d %d\n", day1, day2);
  return 0;
}
```

Υπόδειξη Θυμηθείτε από ποια τιμή ξεκινά η πρώτη σταθερά μιας απαρίθμησης όταν δεν της δίνουμε τιμή, και πόσο αυξάνεται κάθε επόμενη.

A20.4 · Γενεαλογικό δέντρο

A20.4 · *Διάλεξη 20, διαφάνεια 43* · ★☆☆ · short-answer · `slides-lec20-family-tree`

Δίνεται η δομή:

```c
struct person {
  char name[128];
  struct person *parent1;
  struct person *parent2;
```

```
};
```

Με αυτήν φτιάχνουμε το γενεαλογικό δέντρο: ο Angel δείχνει στους Logan και River, ο Logan στους Avery και Andie, ο River στους Hayden και Riley, κ.ο.κ. Μας θυμίζει κάτι;

Υπόδειξη Κάθε κόμβος έχει το πολύ δύο δείκτες προς άλλους κόμβους του ίδιου τύπου, και υπάρχει ένας «πρώτος» κόμβος από τον οποίο ξεκινούν όλα. Ποια δομή δεδομένων έχει αυτά τα χαρακτηριστικά;

A20.5 · Γονικοί φάκελοι

A20.5 · *Διάλεξη 20, διαφάνεια 33* · ★☆☆ · trace · `slides-lec20-folder-parent`

Τι θα τυπώσει το πρόγραμμα;

```c
#include <stdio.h>

typedef struct folder {
  char name[128];
  struct folder *parent;
} Folder;

int main() {
  Folder root = {"/", NULL};
  Folder home = {"home/", &root};
  Folder user = {"thanos/", &home};
  Folder hw0 = {"hw0/", &user}, hw1 = {"hw1/", &user};
  Folder newton = {"newton/", &hw0};
  printf("%s -> %s -> %s\n", newton.name, newton.parent->name,
         newton.parent->parent->name);
  return 0;
}
```

Υπόδειξη Ακολουθήστε τους δείκτες `parent` ένα βήμα κάθε φορά, ξεκινώντας από τον `newton`. Γράψτε δίπλα σε κάθε φάκελο σε ποια μεταβλητή δείχνει.

A20.6 · Δομή για φάκελο

A20.6 · *Διάλεξη 20, διαφάνεια 31* · ★☆☆ · short-answer · `slides-lec20-folder-struct`

Θέλω να φτιάξω μια δομή (struct) που να αναπαριστά έναν φάκελο σε ένα σύστημα αρχείων. Πώς θα το κάνω;

Κάθε φάκελος έχει ένα όνομα και βρίσκεται μέσα σε κάποιον άλλο (parent) φάκελο. Ο αρχικός φάκελος (root `/`) δεν βρίσκεται μέσα σε κάποιο φάκελο. Για παράδειγμα: `/` ⟶ `home/` ⟶ `thanos/` ⟶ `hw0/`, `hw1/`, `hw2/`, και `hw0/` ⟶ `newton/` ⟶ `src/`.

Υπόδειξη Χρειάζεστε ένα μέλος για το όνομα και ένα μέλος που να «δείχνει» στον γονικό φάκελο. Σκεφτείτε γιατί αυτό το μέλος δεν μπορεί να είναι ολόκληρη δομή του ίδιου τύπου, και τι τιμή πρέπει να έχει για τη ρίζα.

A20.7 · Η διάταξη των φακέλων

A20.7 · *Διάλεξη 20, διαφάνεια 38* · ★☆☆ · short-answer · `slides-lec20-list-layout`

Η πραγματική διάταξη (layout) των φακέλων `newton/`, `hw0/`, `thanos/`, `home/`, `/` στη μνήμη μπορεί να είναι περίπλοκη. Σε αφηρημένη μορφή είναι κάπως έτσι:

```
"newton" | &hw0  ->  "hw0" | &user  ->  "thanos" | &home
  ->  "home/" | &root  ->  "/" | NULL
```

Μας θυμίζει κάτι αυτή η διάταξη;

Υπόδειξη Κάθε στοιχείο κρατά μια τιμή και δείχνει στο επόμενο, και το τελευταίο δείχνει στο `NULL`. Ποια δομή δεδομένων ορίζεται ακριβώς έτσι;

A20.8 · Ανάθεση σε πεδία bit

A20.8 · *Διάλεξη 20, διαφάνεια 12* · ★★☆ · trace · `slides-lec20-bitfield-assign`

Τι θα τυπώσει το πρόγραμμα;

```c
#include <stdio.h>

typedef struct {
  unsigned char registered : 1;
  unsigned char year : 3;
  unsigned char grade : 4;
} status;

int main() {
  status st = {1, 1, 10};
  printf("Status: %u %u %u\n", st.registered, st.year, st.grade);
  st.year = 2;
  printf("Status: %u %u %u\n", st.registered, st.year, st.grade);
  st.year += 7;
```

```
  printf("Status: %u %u %u\n", st.registered, st.year, st.grade);
  return 0;
}
```

Υπόδειξη Οι δύο πρώτες γραμμές είναι απλές. Για την τρίτη, σκεφτείτε ποια είναι η μεγαλύτερη τιμή που χωράει σε 3 bits και τι απομένει όταν κρατήσετε μόνο τα 3 χαμηλά bits του αποτελέσματος.

A20.9 · Σύστημα αρχείων με υποφακέλους

A20.9 · *Διάλεξη 20, διαφάνεια 47* · ★★☆ · short-answer · `slides-lec20-filesystem-children`

Θέλω να αναπαραστήσω ένα σύστημα αρχείων (filesystem) ώστε να βρίσκω άμεσα τους υποφακέλους και τον parent φάκελο. Πώς;

Υπόδειξη Ξεκινήστε από τη `struct folder` με όνομα και δείκτη `parent`. Τι πρέπει να προσθέσετε ώστε να φτάνετε και προς τα κάτω; Ένας φάκελος μπορεί να έχει οσουσδήποτε υποφακέλους, οπότε σκεφτείτε μια δομή που μεγαλώνει (π.χ. δυναμικός πίνακας δεικτών ή λίστα).

A20.10 · Διάσχιση φακέλων

A20.10 · *Διάλεξη 20, διαφάνεια 35* · ★★☆ · trace · `slides-lec20-folder-iterate`

Τι θα τυπώσει το πρόγραμμα;

```
#include <stdio.h>

typedef struct folder {
  char name[128];
  struct folder *parent;
} Folder;

int main() {
  Folder root = {"/", NULL};
  Folder home = {"home/", &root};
  Folder user = {"thanos/", &home};
  Folder hw0 = {"hw0/", &user}, hw1 = {"hw1/", &user};
  Folder newton = {"newton/", &hw0};
  for (Folder *iterator = &newton; iterator; iterator = iterator->parent)
    printf("folder: %s\n", iterator->name);
  return 0;
```

```
}
```

Υπόδειξη Ο βρόχος ξεκινά με δείκτη στον `newton` και σε κάθε βήμα μετακινείται στον γονέα. Ρωτήστε: πότε γίνεται ψευδής η συνθήκη `iterator`; Τυπώνεται ποτέ ο `hw1`;

A20.11 · Χρήση ένωσης και μέγεθος

A20.11 · *Διάλεξη 20, διαφάνεια 19* · ★★☆ · trace · `slides-lec20-union-size`

Τι θα τυπώσει το πρόγραμμα;

```c
#include <stdio.h>

union anything {
  char c;  int i;  float f;  double d;
};

int main() {
  union anything a1;
  a1.i = 0x42;
  printf("%d %c\n", a1.i, a1.c);
  a1.c = 'C';
  printf("%d %c\n", a1.i, a1.c);
  printf("%zu\n", sizeof(a1));
  return 0;
}
```

Υπόδειξη Όλα τα μέλη της ένωσης ξεκινούν από το ίδιο byte, οπότε το `c` είναι το πρώτο byte του `i`. Σκεφτείτε ποιο byte ενός ακεραίου είναι πρώτο σε ένα little-endian μηχάνημα, και ότι το μέγεθος της ένωσης καθορίζεται από το μεγαλύτερο μέλος.

A20.12 · Χάρτης ως γράφος

A20.12 · *Διάλεξη 20, διαφάνεια 46* · ★★★ · short-answer · `slides-lec20-graph-map`

Θέλω να αναπαραστήσω έναν χάρτη σε μορφή γράφου με αυτοαναφορικές δομές. Πώς;

Ο χάρτης έχει τέσσερις κόμβους A, B, C, D και κάθε ζεύγος συνδέεται με ακμή που έχει βάρος (απόσταση): A–B 20, A–C 42, A–D 35, B–C 30, B–D 34, C–D 12.

Υπόδειξη Σκεφτείτε τι πρέπει να κρατά κάθε κόμβος: ένα όνομα και τους γειτονές του. Επειδή σε έναν γράφο το πλήθος των γειτόνων δεν είναι σταθερό και κάθε ακμή έχει βάρος, δεν αρκεί ένας σταθερός αριθμός δεικτών όπως στη λίστα ή στο δυαδικό δέντρο.

Κεφάλαιο 21: Λίστες και Δέντρα

Kahoot από το αμφιθέατρο (K21.1–K21.3)

K21.1 · Ανάγνωση διαγραμμένου στοιχείου

K21.1 · *Kahoot «Λίστες, Δέντρα and Beyond»* · ★☆☆ · multiple-choice · 92% σωστές · `kahoot-list-use-after-free`

Μόλις διέγραψα ένα στοιχείο από μια λίστα. Τι θα συμβεί αν προσπαθήσω να το διαβάσω;

- Το πρόγραμμά μου θα σκάσει (Segmentation Fault)
- Όλα θα πάνε καλά και θα διαβάσω κανονικά το στοιχείο

Υπόδειξη Μετά το `free` η μνήμη του κόμβου δεν ανήκει πια στο πρόγραμμά σας· σκεφτείτε αν είναι ασφαλές να τη διαβάσετε.

K21.2 · Εργαλείο για memory leaks

K21.2 · *Kahoot «Δυαδική Αναζήτηση και Ταξινόμηση» (διάλεξη 17)* · ★☆☆ · multiple-choice · 86% σωστές · `kahoot-valgrind`

Το εργαλείο που μας βοηθάει να βρούμε memory leaks στο πρόγραμμά μας λέγεται:

- Grindenwald
- Valhalla
- Valgrind
- Mjölnir

Υπόδειξη Είναι το εργαλείο που τρέχει το πρόγραμμά σας και στο τέλος αναφέρει μνήμη «definitely lost».

K21.3 · Εισαγωγή στη μέση λίστας

K21.3 · *Kahoot «Λίστες, Δέντρα and Beyond»* · ★★★ · multiple-choice · 39% σωστές · `kahoot-list-insert-middle`

Η προσθήκη ενός στοιχείου στη μέση μιας απλά συνδεδεμένης λίστας γίνεται σε χρόνο:

- O(n^2)
- O(f(x))
- O(n)
- O(1)

Συχνή παρανόηση Το 26% επέλεξε O(1), μετρώντας μόνο την αλλαγή των δύο δεικτών και ξεχνώντας ότι πρώτα πρέπει να φτάσουμε στη μέση διατρέχοντας τη λίστα από την αρχή (άλλο ένα 32% διάλεξε το αστείο O(f(x))).

Υπόδειξη Σε μια απλά συνδεδεμένη λίστα έχετε μόνο τον δείκτη στην κεφαλή· πόσα βήματα χρειάζονται για να φτάσετε στη μέση;

Ζέσταμα: από τις διαφάνειες (A21.1–A21.6)

A21.1 · Προσθήκη στοιχείου σε λίστα

A21.1 · *Διάλεξη 21, διαφάνειες 11–12* · ★☆☆ · short-answer · `slides-lec21-insert-how`

Δίνεται η απλά συνδεδεμένη λίστα:

```
list -> 6 -> 7 -> 8 -> NULL
```

Θέλω να προσθέσω ένα στοιχείο (π.χ., το 5) σε λίστα. Πως;

Υπόδειξη Σκεφτείτε πού είναι φθηνότερο να μπει ο νέος κόμβος, ώστε να μη χρειαστεί να διασχίσετε τη λίστα. Ποιοι δείκτες πρέπει να αλλάξουν, και με ποια σειρά, ώστε να μη χαθεί κανένας κόμβος; Από πού θα πάρετε μνήμη για τον νέο κόμβο;

A21.2 · Τι τυπώνει η εισαγωγή σε λίστα;

A21.2 · *Διάλεξη 21, διαφάνειες 13–16* · ★☆☆ · trace · `slides-lec21-insert-output`

Τι θα τυπώσει το πρόγραμμα;

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct listnode {int value; struct listnode * next;} * List;
void insert(List * list, int value) {
  List current_head = *list;
  List new_head = malloc(sizeof(struct listnode));
  new_head->value = value;
```

```
  new_head->next = current_head;
  *list = new_head;
}
void print(List list) {
  printf("list: ");
  while(list) {
    printf(" -> %d", list->value);
    list = list->next;
  }
  printf(" -> NULL\n");
}
int main() {
  List list = NULL;
  insert(&list, 42); insert(&list, 43); insert(&list, 44);
  print(list);
  return 0;
}
```

Υπόδειξη Ζωγραφίστε τη λίστα μετά από κάθε κλήση της `insert`: σε ποια θέση μπαίνει κάθε νέος κόμβος; Προσέξτε και τα κενά που τυπώνουν τα δύο πρώτα `printf`.

A21.3 · Αναδρομικό μήκος λίστας

A21.3 · *Διάλεξη 21, διαφάνεια 18* · ★☆☆ · programming · `slides-lec21-recursive-length`

Η παρακάτω συνάρτηση βρίσκει το μήκος μιας λίστας επαναληπτικά:

```
int length(List list) {
  int counter = 0;
  while(list) {
    counter++;
    list = list->next;
  }
  return counter;
}
```

Πως θα το κάναμε αναδρομικά;

Υπόδειξη Ποιο είναι το μήκος της κενής λίστας (`NULL`); Αν ξέρατε το μήκος της λίστας που ξεκινά από το `list->next`, πώς θα βρίσκατε το μήκος ολόκληρης της λίστας;

A21.4 · Αφαίρεση στοιχείου από λίστα

A21.4 · *Διάλεξη 21, διαφάνειες 23–25* · ★★☆ · short-answer · `slides-lec21-delete-how`

Δίνεται η απλά συνδεδεμένη λίστα:

```
list -> 5 -> 6 -> 7 -> 8 -> NULL
```

Θέλω να αφαιρέσω ένα στοιχείο (π.χ., το 6). Πως;

Υπόδειξη Ποιος δείκτης δείχνει σήμερα στον κόμβο 6, και πού πρέπει να δείχνει μετά την αφαίρεση; Τι γίνεται με τη μνήμη του κόμβου; Σκεφτείτε ξεχωριστά την περίπτωση που το στοιχείο είναι η κεφαλή: γιατί η συνάρτηση χρειάζεται `List *` και όχι `List`;

A21.5 · Τι τυπώνει η find σε λίστα;

A21.5 · *Διάλεξη 21, διαφάνειες 21–22* · ★★☆ · trace · `slides-lec21-find-output`

Τι θα τυπώσει το πρόγραμμα; (Η `insert` είναι η εισαγωγή στην αρχή της λίστας της διάλεξης.)

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct listnode {int value; struct listnode * next;} * List;
List find(List list, int value) {
  while(list && list->value != value) {
    list = list->next;
  }
  return list;
}
int main() {
  List list = NULL;
  insert(&list, 42); insert(&list, 43); insert(&list, 44);
  printf("Found 43: %x\n", find(list, 43));
  printf("Found 34: %x\n", find(list, 34));
  return 0;
}
```

Ποια η πολυπλοκότητα της `find` ως προς χρόνο και χώρο;

Υπόδειξη Τι επιστρέφει η `find` όταν βρει την τιμή και τι όταν φτάσει στο τέλος της λίστας; Η μία από τις δύο γραμμές δεν μπορεί να προβλεφθεί ακριβώς: γιατί; Σκεφτείτε επίσης αν το `%x` είναι ο σωστός προσδιοριστής για δείκτη.

A21.6 · Πολυπλοκότητα του μήκους λίστας

A21.6 · *Διάλεξη 21, διαφάνειες 19–20* · ★★☆ · short-answer · `slides-lec21-length-complexity`

Δίνονται δύο υλοποιήσεις της `length`:

```
int length(List list) {
  int counter = 0;
  while(list) {
    counter++;
    list = list->next;
  }
  return counter;
}
```

```
int length(List list) {
  if (!list) return 0;
  return 1 + length(list->next);
}
```

Ποια η πολυπλοκότητα των παραπάνω ως προς χρόνο και χώρο;

Υπόδειξη Για τον χρόνο, μετρήστε πόσες φορές επισκέπτεται κάθε εκδοχή κάθε κόμβο. Για τον χώρο, σκεφτείτε πόσες κλήσεις της συνάρτησης είναι ταυτόχρονα ενεργές στη στοίβα όταν φτάνουμε στο τέλος της λίστας.

Εργαστήριο (A21.7)

A21.7 · Συνδεδεμένες λίστες

A21.7 · *Εργαστήριο 9, Άσκηση 3* · ★★☆ · programming · `lab-lab09-grades`

3.1 Κατασκευάστε το πρόγραμμα `grades.c` και ορίστε μία αυτοαναφορική δομή λίστας ακεραίων αριθμών.

```
typedef struct listnode *Listptr;

struct listnode {
    int data;
    Listptr next;
};
```

3.2 Κατασκευάστε τη συνάρτηση:

```
void insert_at_start(Listptr *ptr, int grade);
```

Η συνάρτηση θα προσθέτει έναν βαθμό στην αρχή της λίστας. Τροποποιήστε τη `main` για να διαβάζει βαθμούς από το πληκτρολόγιο και να τους προσθέτει στη λίστα.

3.3 Κατασκευάστε τη συνάρτηση:

```
float average(Listptr ptr);
```

Η συνάρτηση θα διασχίζει τα περιεχόμενα της λίστας και θα υπολογίζει τον μέσο όρο των βαθμών που είναι αποθηκευμένοι σε αυτήν.

Υπόδειξη Η `insert_at_start` αλλάζει την κεφαλή της λίστας, γι' αυτό παίρνει δείκτη στον δείκτη της κεφαλής: δεσμεύει νέο κόμβο, τον κάνει να δείχνει στην παλιά κεφαλή και μετά ενημερώνει την κεφαλή. Η `average` προχωρά ακολουθώντας τα `next` μέχρι το `NULL`, μετρώντας και αθροίζοντας. Σκεφτείτε τι πρέπει να επιστρέψει για κενή λίστα.

Θέματα εξετάσεων (A21.8–A21.10)

A21.8 · Αντιστροφή λίστας

A21.8 · *Εξέταση Σεπτεμβρίου 2024, Θέμα 5* · ★★☆ · programming · `exam-2024-sep-q5`

Αντιστροφή Λίστας [25 Μονάδες]

Γράψτε μια συνάρτηση `reverse_list` η οποία παίρνει ως όρισμα μια λίστα ακεραίων τύπου `List` και επιστρέφει μια νέα λίστα με τους κόμβους της σε αντίστροφη σειρά σε σχέση με την αρχική. Για παράδειγμα, αν δοθεί η ακόλουθη λίστα:

```
[12|•]□□>[99|•]□□>[37|•]□□>NULL
```

περιμένουμε να επιστραφεί η ανεστραμμένη χωρίς παρενέργειες (side-effects) στην αρχική:

```
[37|•]□□>[99|•]□□>[12|•]□□>NULL
```

Ποια είναι η χρονική και η χωρική πολυπλοκότητα του αλγορίθμου σας (8/25 της βαθμολογίας); Ο τύπος `List` δίνεται παρακάτω:

```
typedef struct node {
  int value;
  struct node * next;
} * List;
```

Υπόδειξη «Χωρίς παρενέργειες» σημαίνει ότι δεν πειράζετε τους δείκτες `next` της αρχικής λίστας: χρειάζεστε νέους κόμβους με `malloc`. Παρατηρήστε ότι αν διατρέξετε την αρχική λίστα από την αρχή και εισάγετε κάθε αντίγραφο στην *αρχή* της νέας, η σειρά βγαίνει αντεστραμμένη. Σκεφτείτε την κενή λίστα και τι γίνεται αν αποτύχει η `malloc`.

A21.9 · Μεσαίο Στοιχείο Λίστας

A21.9 · *Εξέταση Σεπτεμβρίου 2025, Θέμα 4* · ★★☆ · programming · `exam-2025-sep-q4`

Μεσαίο Στοιχείο Λίστας [25 Μονάδες]

Γράψτε μια συνάρτηση `middle_element` η οποία παίρνει ως όρισμα μια λίστα ακεραίων τύπου `List` και επιστρέφει μια νέα λίστα με μοναδικό στοιχείο το μεσαίο στοιχείο της λίστας. Αν ο αριθμός των στοιχείων είναι άρτιος, μπορεί να επιστραφεί οποιοδήποτε από τα μεσαία στοιχεία. Για παράδειγμα, αν δοθεί η ακόλουθη λίστα:

```
12 -> 99 -> 42 -> 3 -> 17 -> NULL
```

περιμένουμε να επιστραφεί η ακόλουθη λίστα:

```
42 -> NULL
```

Ποια είναι η χρονική και η χωρική πολυπλοκότητα του αλγορίθμου σας (8/25 της βαθμολογίας); Ο τύπος `List` δίνεται παρακάτω:

```c
typedef struct node {
  int value;
  struct node * next;
} * List;
```

Bonus: Αν ο αλγόριθμός σας βρίσκει το μεσαίο στοιχείο της λίστας χωρίς να την διατρέξει πάνω από μία φορές.

Υπόδειξη Η απλή λύση μετρά πρώτα τους κόμβους και μετά ξαναπερπατά ως τη μέση· για το bonus, σκεφτείτε δύο δείκτες που ξεκινούν μαζί αλλά προχωρούν με διαφορετική ταχύτητα. Μην ξεχάσετε ότι πρέπει να επιστραφεί **νέα** λίστα (ένας νέος κόμβος με `malloc`), και ελέγξτε τι επιστρέφετε για κενή λίστα ή λίστα ενός στοιχείου.

A21.10 · N-οστό Στοιχείο Λίστας

A21.10 · *Εξέταση Σεπτεμβρίου 2026, Θέμα 4* · ★★☆ · programming · `exam-2026-sep-q4`

N-οστό Στοιχείο Λίστας [25 Μονάδες]

Γράψτε μια συνάρτηση `nth` η οποία παίρνει ως όρισμα μια λίστα από πίνακες χαρακτήρων τύπου `List` μαζί με έναν αριθμό `n` και επιστρέφει μια νέα λίστα με μοναδικό στοιχείο το n-οστό στοιχείο της λίστας. Αν δεν υπάρχει n-οστό στοιχείο, το πρόγραμμα θέλουμε να τερματίζει με κωδικό εξόδου 1. Για παράδειγμα, αν δοθεί η ακόλουθη λίστα:

```
"foo" -> "bar" -> "zonk" -> "baz" -> "hello" -> NULL
```

και `n=3` περιμένουμε να επιστραφεί η ακόλουθη νέα λίστα (0-indexing όπως στην C):

```
"baz" -> NULL
```

Ποια είναι η χρονική και η χωρική πολυπλοκότητα του αλγορίθμου σας (8/25 της βαθμολογίας); Ο τύπος `List` δίνεται παρακάτω:

```
typedef struct node {
  char value[32];
  struct node * next;
} * List;
```

Υπόδειξη Προχωρήστε `n` βήματα από την κεφαλή ακολουθώντας τα `next`, ελέγχοντας σε κάθε βήμα μήπως τελείωσε η λίστα (και τι γίνεται αν η λίστα είναι κενή ή το `n` αρνητικό). Η νέα λίστα είναι ένας καινούργιος κόμβος από `malloc`: αντιγράψτε το `value` (δεν ανατίθεται ως πίνακας) και βάλτε `next = NULL`, αντί να επιστρέψετε δείκτη μέσα στην παλιά λίστα. Για την πολυπλοκότητα, μετρήστε πόσους κόμβους επισκέπτεστε και πόση επιπλέον μνήμη δεσμεύετε.

Κεφάλαιο 22: Δέντρα

Kahoot από το αμφιθέατρο (K22.1–K22.9)

K22.1 · Κόμβοι τέλειου δυαδικού δέντρου

K22.1 · *Kahoot «Λίστες, Δέντρα and Beyond»* · ★☆☆ · multiple-choice · 81% σωστές · `kahoot-perfect-tree-nodes`

Έχω ένα τέλειο δυαδικό δέντρο βάθους n. Πόσους κόμβους έχει;

- ~n
- ~log n
- ~2^n
- Εξαρτάται

Υπόδειξη Μετρήστε τους κόμβους ανά επίπεδο: κάθε επίπεδο έχει διπλάσιους κόμβους από το προηγούμενο.

K22.2 · Ο καλύτερος αλγόριθμος αναζήτησης

K22.2 · *Kahoot «Λίστες, Δέντρα and Beyond»* · ★☆☆ · multiple-choice · 75% σωστές · `kahoot-best-search-algorithm`

Ποιος αλγόριθμος αναζήτησης είναι ο καλύτερος;

- Breadth-First Search

- Depth-First Search
- Beam Search
- Εξαρτάται

Υπόδειξη Θυμηθείτε τη συζήτηση «DFS ή BFS;»: πότε κερδίζει ο καθένας, ανάλογα με το σχήμα του δέντρου και το πού βρίσκεται η λύση;

K22.3 · Ταξινομημένη εκτύπωση BST

K22.3 · *Kahoot «Λίστες, Δέντρα and Beyond»* · ★★☆ · multiple-choice · 66% σωστές · `kahoot-bst-sorted-traversal`

Έστω ότι έχω ένα δυαδικό δέντρο αναζήτησης. Για να τυπώσω τα στοιχεία του ταξινομημένα, χρησιμοποιώ διάσχιση:

- preorder
- inorder
- postorder
- Δεν γίνεται

Υπόδειξη Σε ένα BST τα μικρότερα στοιχεία είναι αριστερά και τα μεγαλύτερα δεξιά· πότε πρέπει να τυπώνεται η ρίζα σε σχέση με τα δύο υποδέντρα;

K22.4 · Κόμβοι δέντρου με n επίπεδα

K22.4 · *Kahoot «Λίστες, Δέντρα and Beyond»* · ★★☆ · multiple-choice · 53% σωστές · `kahoot-tree-levels-nodes`

Έχω ένα δέντρο με n επίπεδα. Πόσους κόμβους έχει;

- 2^n
- log n
- n
- Εξαρτάται

Συχνή παρανόηση Το 37% επέλεξε 2^n, που είναι μόνο το μέγιστο για ένα τέλειο δυαδικό δέντρο· ένα τυχαίο δέντρο μπορεί να έχει από έναν κόμβο ανά επίπεδο μέχρι πολύ περισσότερους.

Υπόδειξη Η ερώτηση δεν λέει τι είδους δέντρο είναι· σκεφτείτε ένα εκφυλισμένο δέντρο και ένα τέλειο με τα ίδια επίπεδα.

K22.5 · Ελάχιστο βάθος δυαδικού δέντρου

K22.5 · *Kahoot «Προχωρημένες Δομές #2»* · ★★☆ · multiple-choice · 47% σωστές · `kahoot-tree-min-depth`

Έστω ένα δυαδικό δέντρο (binary tree) με n στοιχεία. Το ελάχιστο δυνατό βάθος του δέντρου είναι:

- O(n)
- O(f(x))
- O(log n)
- O(1)

Υπόδειξη Για να είναι το βάθος ελάχιστο, κάθε επίπεδο πρέπει να είναι γεμάτο· πόσα επίπεδα χρειάζονται για n κόμβους αν κάθε επίπεδο διπλασιάζεται;

K22.6 · Φύλλα τέλειου δυαδικού δέντρου

K22.6 · *Kahoot «Λίστες, Δέντρα and Beyond»* · ★★☆ · multiple-choice · 45% σωστές · `kahoot-perfect-tree-leaves`

Έχω ένα τέλειο δυαδικό δέντρο με n κόμβους. Πόσα φύλλα έχει;

- ~2n
- ~n
- ~n/2
- Εξαρτάται

Υπόδειξη Συγκρίνετε το πλήθος των κόμβων του τελευταίου επιπέδου με το άθροισμα όλων των προηγούμενων επιπέδων.

K22.7 · BFS σε τέλειο δυαδικό δέντρο

K22.7 · *Kahoot «Λίστες, Δέντρα and Beyond»* · ★★★ · multiple-choice · 37% σωστές · `kahoot-bfs-perfect-tree`

MIT (3-ετείς): έστω T ένα τέλειο δυαδικό δέντρο με n κόμβους. Μπορώ να βρω ένα στοιχείο με BFS σε O(log n);

- True
- False

Συχνή παρανόηση Το 61% απάντησε True, μπερδεύοντας το ύψος του δέντρου (log n) με το κόστος της αναζήτησης· η BFS σε δέντρο που δεν είναι BST μπορεί να χρειαστεί να επισκεφθεί όλους τους κόμβους.

Υπόδειξη Η BFS δεν ξέρει προς ποιο παιδί να πάει· σκεφτείτε πόσους κόμβους επισκέπτεται στη χειρότερη περίπτωση, αν το στοιχείο είναι στο τελευταίο επίπεδο.

K22.8 · Αποτίμηση δέντρου εκφράσεων

K22.8 · *Kahoot «Λίστες, Δέντρα and Beyond»* · ★★★ · multiple-choice · 35% σωστές · `kahoot-expression-tree-eval`

Θέλω να γράψω έναν αποτιμητή εκφράσεων που αναπαρίστανται με δέντρα. Προτιμώ διάσχιση:

- preorder
- inorder
- postorder
- Δεν χρειάζομαι διάσχιση, το κάνω σε O(1)

Συχνή παρανόηση Το 28% επέλεξε preorder, όμως για να εφαρμόσουμε τον τελεστή της ρίζας χρειαζόμαστε πρώτα τις τιμές και των δύο υποδέντρων.

Υπόδειξη Σε έναν κόμβο + με παιδιά τις υποεκφράσεις, τι πρέπει να έχετε υπολογίσει πριν κάνετε την πρόσθεση;

K22.9 · Χειρότερη περίπτωση αναζήτησης σε BST

K22.9 · *Kahoot «Λίστες, Δέντρα and Beyond»* · ★★★ · multiple-choice · 29% σωστές · `kahoot-bst-worst-case`

Έχω ένα δυαδικό δέντρο αναζήτησης (BST) με n κόμβους. Μπορεί να χρειαστώ O(n) χρόνο για να βρω ένα στοιχείο:

- Όταν το δέντρο είναι τέλειο και έχει όλα τα φύλλα
- Όταν το δέντρο είναι εκφυλισμένο
- Όταν το δέντρο είναι ισορροπημένο και δεν ξέρω προς τα που να ψάξω
- Δεν γίνεται, σε BST πάντα χρειάζομαι O(logn)

Συχνή παρανόηση Το 30% επέλεξε το ισορροπημένο δέντρο, ενώ σε ισορροπημένο BST η σύγκριση με κάθε κόμβο δείχνει πάντα προς τα πού να ψάξουμε, άρα αρκούν O(log n) βήματα.

Υπόδειξη Το κόστος της αναζήτησης σε BST είναι ανάλογο του ύψους του· πότε το ύψος γίνεται ίσο με n;

Ζέσταμα: από τις διαφάνειες (A22.1–A22.13)

A22.1 · Ποιος αλγόριθμος αναζήτησης είναι καλύτερος;

A22.1 · *Διαλέξεις 21–22: Δέντρα, διαφάνεια 38 (διάλεξη 21: διαφάνεια 60)* · ★☆☆ · short-answer · `slides-lec22-best-search`

Ποιος αλγόριθμος αναζήτησης είναι καλύτερος; Η αναζήτηση κατά βάθος (DFS) ή η αναζήτηση κατά πλάτος (BFS);

Υπόδειξη Συγκρίνετε τη σειρά με την οποία επισκέπτεται τους κόμβους ο καθένας και πόση προσωρινή μνήμη χρειάζεται (τη στοίβα της αναδρομής έναντι του μετώπου). Σκεφτείτε ένα πρόβλημα όπου πλεονεκτεί ο καθένας.

A22.2 · Έλεγχος ύπαρξης σε δυαδικό δέντρο αναζήτησης

A22.2 · *Διαλέξεις 21–22: Δέντρα, διαφάνειες 35–37 (διάλεξη 21: διαφάνειες 57–59)* · ★☆☆ · programming · `slides-lec22-bst-exists`

Ένα Δυαδικό Δέντρο Αναζήτησης (Binary Search Tree - BST ή Ordered Tree) είναι ένα ταξινομημένο δέντρο έτσι ώστε για κάθε κόμβο, όλοι οι κόμβοι στο αριστερό του υποδέντρο να είναι μικρότεροι σε τιμή και όλοι οι κόμβοι στο δεξί του υποδέντρο να είναι μεγαλύτεροι.

Έλεγχος αν υπάρχει (exists) ένα στοιχείο σε δυαδικό δέντρο. Πως;

Γράψτε τη συνάρτηση

```
int exists(Tree t, int value);
```

που επιστρέφει 1 αν η τιμή `value` υπάρχει στο BST `t` και 0 αλλιώς, με χρόνο $O(\log n)$ για ισορροπημένο δέντρο.

Υπόδειξη Σε κάθε κόμβο, η σύγκριση του `value` με την τιμή του κόμβου σας λέει σε ποιο από τα δύο υποδέντρα *μπορεί* να βρίσκεται η τιμή· το άλλο δεν χρειάζεται να το ψάξετε. Μην ξεχάσετε τη βασική περίπτωση του άδειου δέντρου.

A22.3 · Κόμβοι τέλειου δυαδικού δέντρου

A22.3 · *Διαλέξεις 21–22: Δέντρα, διαφάνεια 10 (διάλεξη 21: διαφάνεια 32)* · ★☆☆ · short-answer · `slides-lec22-perfect-tree-nodes`

Ένα **τέλειο δυαδικό δέντρο** (perfect binary tree) είναι ένα δυαδικό δέντρο που όλοι οι εσωτερικοί κόμβοι έχουν δύο παιδιά και όλα τα φύλλα βρίσκονται στο ίδιο επίπεδο (η ρίζα βρίσκεται στο επίπεδο 1).

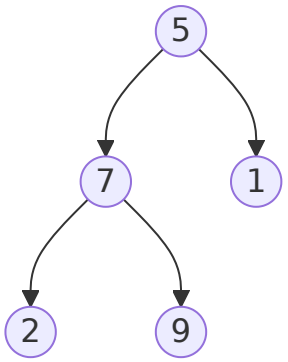
Πόσοι κόμβοι σε n επίπεδα;

Υπόδειξη Μετρήστε πόσους κόμβους έχει κάθε επίπεδο ξεχωριστά: πόσους έχει το επίπεδο 1, πόσους το 2, και πώς σχετίζεται κάθε επίπεδο με το προηγούμενο; Μετά αθροίστε τη γεωμετρική πρόοδο.

A22.4 · Διασχίσεις pre-order, in-order, post-order

A22.4 · *Διαλέξεις 21–22: Δέντρα, διαφάνειες 22–28 (διάλεξη 21: διαφάνειες 44–50)* · ★☆☆ · trace · `slides-lec22-traversals`

Δίνεται το δέντρο με ρίζα το 5· τα παιδιά του 5 είναι το 7 (αριστερά) και το 1 (δεξιά), και τα παιδιά του 7 είναι το 2 (αριστερά) και το 9 (δεξιά).



Τι τυπώνει καθεμία από τις παρακάτω εκδοχές της `print` όταν καλείται με τη ρίζα;

```
void print(Tree t) {       /* Pre-order */
  if (t == NULL) return;
  printf("%d ", t->value);
  print(t->left);
  print(t->right);
}
```

```
void print(Tree t) {       /* In-order */
```

```
  if (t == NULL) return;
  print(t->left);
  printf("%d ", t->value);
  print(t->right);
}

void print(Tree t) {       /* Post-order */
  if (t == NULL) return;
  print(t->left);
  print(t->right);
  printf("%d ", t->value);
}
```

Υπόδειξη Εφαρμόστε τον ορισμό αναδρομικά, ξεκινώντας από τη ρίζα: η διάσχιση του 5 αποτελείται από τη διάσχιση του υποδέντρου του 7, τη διάσχιση του υποδέντρου του 1 και το ίδιο το 5, με τη σειρά που ορίζει η θέση του `printf`.

A22.5 · Μέγιστο στοιχείο δέντρου: BFS ή DFS;

A22.5 · *Διαλέξεις 21–22: Δέντρα, διαφάνεια 41 (διάλεξη 21: διαφάνεια 63)* · ★☆☆ · short-answer · `slides-lec22-tree-max`

Θέλεις να βρεις το μέγιστο στοιχείο ενός δέντρου. Προτιμάς BFS ή DFS; Γιατί;

Υπόδειξη Χρειάζεται να επισκεφθείτε όλους τους κόμβους ούτως ή άλλως, άρα ο χρόνος είναι ίδιος. Συγκρίνετε λοιπόν την προσωρινή μνήμη που χρειάζεται ο καθένας και πόσο απλός είναι ο κώδικας.

A22.6 · Πολυπλοκότητα της depth

A22.6 · *Διαλέξεις 21–22: Δέντρα, διαφάνεια 20 (διάλεξη 21: διαφάνειες 41–42)* · ★★☆ · short-answer · `slides-lec22-depth-complexity`

Δίνεται η συνάρτηση που υπολογίζει το βάθος ενός δυαδικού δέντρου:

```
int depth(Tree t) {
  if (t == NULL) return -1;
  int left_depth = depth(t->left);
  int right_depth = depth(t->right);
  return 1 + ((left_depth > right_depth) ? left_depth : right_depth);
}
```

Ποια η πολυπλοκότητα για ένα τέλειο δυαδικό δέντρο; (χρόνος και χώρος)

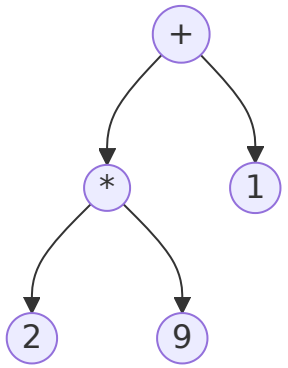
Υπόδειξη Για τον χρόνο, μετρήστε πόσες φορές καλείται η `depth` για κάθε κόμβο. Για τον χώρο, σκεφτείτε πόσες κλήσεις μπορεί να βρίσκονται ταυτόχρονα στη στοίβα, και πώς σχετίζεται αυτό με το ύψος ενός τέλειου δέντρου με n κόμβους.

A22.7 · Διάσχιση για αποτιμητή εκφράσεων

A22.7 · *Διαλέξεις 21–22: Δέντρα, διαφάνεια 29 (διάλεξη 21: διαφάνεια 51)* · ★★☆ · short-answer · `slides-lec22-expression-evaluator`

Θέλω να γράψω έναν αποτιμητή εκφράσεων (calculator / evaluator / interpreter). Ποια διάσχιση θα χρησιμοποιήσω;

Το δέντρο της διαφάνειας έχει ρίζα `+`· το αριστερό παιδί του είναι `*` (με παιδιά `2` και `9`) και το δεξί είναι `1`.



Υπόδειξη Για να υπολογίσετε τον τελεστή ενός κόμβου χρειάζεστε ήδη τις τιμές των δύο υποδέντρων του. Ποια από τις τρεις διασχίσεις (pre-order, in-order, post-order) επεξεργάζεται τον κόμβο αφού έχει τελειώσει με τα παιδιά του;

A22.8 · Λίστα για το BFS και λίστες κάθε τύπου

A22.8 · *Διαλέξεις 21–22: Δέντρα, διαφάνεια 33 (διάλεξη 21: διαφάνεια 55)* · ★★☆ · short-answer · `slides-lec22-generic-list`

Δίνεται η συνάρτηση που τυπώνει ένα δέντρο με διάσχιση κατά πλάτος (BFS):

```
void bfs(Tree t) {
  List worklist = NULL;
  Tree tmp;
  insert(&worklist, t);
```

```
  while(worklist) {
    tmp = pop_last(&worklist);
    printf("%d ", tmp->value);
    if (tmp->left) insert(&worklist, tmp->left);
    if (tmp->right) insert(&worklist, tmp->right);
  }
}
```

Σημείωση: Ο τύπος List παραπάνω δεν περιέχει πλέον ακεραίους. Τι περιέχει; Πως θα έπρεπε να αλλάξει προκειμένου να υποστηρίξει μια τέτοια υλοποίηση; Τι θα κάνατε ώστε οι λίστες σας να δουλεύουν με κάθε τύπου δεδομένα;

Υπόδειξη Κοιτάξτε τον τύπο του δεύτερου ορίσματος της `insert` και της τιμής που επιστρέφει η `pop_last`. Για το τελευταίο ερώτημα, σκεφτείτε έναν τύπο δείκτη της C που μπορεί να δείχνει σε δεδομένα οποιουδήποτε τύπου, και διαβάστε για τους αφηρημένους τύπους δεδομένων (ΑΤΔ).

A22.9 · Συντομότερο μονοπάτι σε δέντρα-λαβύρινθους: BFS ή DFS;

A22.9 · *Διαλέξεις 21–22: Δέντρα, διαφάνεια 40 (διάλεξη 21: διαφάνεια 62)* · ★★☆ · short-answer · `slides-lec22-maze-shortest-path`

Γράφεις ένα πρόγραμμα που βρίσκει λύσεις σε δέντρα-λαβυρίνθους. Αναζητάς το συντομότερο μονοπάτι για την έξοδο. BFS ή DFS; Γιατί;

Υπόδειξη Σκεφτείτε με ποια σειρά απόστασης από την αρχή επισκέπτεται τους κόμβους ο κάθε αλγόριθμος. Όταν ο καθένας βρει για πρώτη φορά μια έξοδο, είναι εγγυημένα η πιο κοντινή;

A22.10 · Αποθήκευση και αναζήτηση σε 1 PetaByte

A22.10 · *Διαλέξεις 21–22: Δέντρα, διαφάνεια 39 (διάλεξη 21: διαφάνεια 61)* · ★★☆ · short-answer · `slides-lec22-petabyte`

Έχεις να αποθηκεύσεις 1 PetaByte (10^6 GB) δεδομένων για μια υπηρεσία (service). Πως θα αποθηκεύσεις τα δεδομένα σου; Πως θα κάνεις αναζήτηση;

Υπόδειξη Υπολογίστε πόσα βήματα θα χρειαζόταν μια σειριακή αναζήτηση σε τέτοιο όγκο και συγκρίνετε με μια δομή όπου κάθε σύγκριση απορρίπτει το μισό των δεδομένων. Σκεφτείτε επίσης τι γίνεται όταν προστίθενται νέα δεδομένα.

A22.11 · Δέντρο καταστάσεων τρίλιζας

A22.11 · *Διάλεξη 22: Δέντρα, διαφάνεια 15* · ★★☆ · short-answer · `slides-lec22-tictactoe-tree`

Είναι συνηθισμένο να αναπαριστούμε καταστάσεις χρησιμοποιώντας δομές όπως τα δέντρα, κάποιες φορές με **περισσότερα** από 2 παιδιά κόμβους. Η διαφάνεια δείχνει ένα δέντρο καταστάσεων για την τρίλιζα (tic-tac-toe): η ρίζα είναι μια θέση του παιχνιδιού, κάθε παιδί είναι η θέση μετά από μία κίνηση, και τα φύλλα έχουν βαθμολογία (+10, -10 ή 0).

Αυτό το δέντρο καταστάσεων είναι σωστό ή έχει σφάλματα; Πως θα τα βρίσκαμε με έναν αλγόριθμο αυτόματα;

Υπόδειξη Σκεφτείτε ποιες σχέσεις πρέπει να ισχύουν ανάμεσα σε έναν κόμβο και κάθε παιδί του (πόσα σύμβολα προστίθενται, ποιος παίκτης παίζει, αν αλλάζουν τα ήδη γεμάτα τετράγωνα) και ποια βαθμολογία πρέπει να έχει κάθε φύλλο. Ένας αλγόριθμος μπορεί να επισκεφθεί κάθε κόμβο με μια διάσχιση του δέντρου και να ελέγξει αυτούς τους κανόνες τοπικά.

A22.12 · Προσθήκη στοιχείου σε δυαδικό δέντρο

A22.12 · *Διαλέξεις 21–22: Δέντρα, διαφάνεια 16 (διάλεξη 21: διαφάνεια 38)* · ★★☆ · programming · `slides-lec22-tree-insert`

Στις βασικές λειτουργίες με δυαδικά δέντρα, η διάλεξη αφήνει ως άσκηση:

- insert: Προσθήκη στοιχείου στο δέντρο (μόνοι σας)

Υλοποιήστε την προσθήκη ενός ακεραίου σε δυαδικό δέντρο αναζήτησης (BST) με κόμβους

```
typedef struct treenode {
  int value;
  struct treenode *left;
  struct treenode *right;
} *Tree;
```

ώστε το δέντρο να παραμένει ταξινομημένο. Ελέγξτε το αποτέλεσμα τυπώνοντας το δέντρο με in-order διάσχιση.

Υπόδειξη Ακολουθήστε την ίδια διαδρομή με την αναζήτηση στο BST· όταν φτάσετε σε `NULL`, εκεί ανήκει ο νέος κόμβος, που δεσμεύεται με `malloc`. Για να αλλάξει ο δείκτης του γονιού (ή η ρίζα, αν το δέντρο είναι άδειο), είτε επιστρέψτε το νέο δέντρο είτε περάστε δείκτη σε `Tree`, όπως στην `insert` των λιστών.

A22.13 · Αφαίρεση στοιχείου από δυαδικό δέντρο

A22.13 · *Διαλέξεις 21–22: Δέντρα, διαφάνεια 16 (διάλεξη 21: διαφάνεια 38)* · ★★★ · programming · `slides-lec22-tree-delete`

Στις βασικές λειτουργίες με δυαδικά δέντρα, η διάλεξη αφήνει ως άσκηση:

- delete: Αφαίρεση στοιχείου από δέντρο (μόνοι σας)

Υλοποιήστε την αφαίρεση μιας τιμής από δυαδικό δέντρο αναζήτησης (BST), ώστε το δέντρο να παραμένει ταξινομημένο και η μνήμη του κόμβου που αφαιρείται να αποδεσμεύεται.

Υπόδειξη Χωρίστε σε περιπτώσεις ανάλογα με το πόσα παιδιά έχει ο κόμβος που αφαιρείται: 0, 1 ή 2. Στην τελευταία, σκεφτείτε ποια τιμή του δέντρου μπορεί να πάρει τη θέση του χωρίς να χαλάσει η διάταξη (κοιτάξτε το δεξί υποδέντρο του).

Εργαστήριο (A22.14)

A22.14 · Δυαδικά δένδρα

A22.14 · *Εργαστήριο 9, Άσκηση 4* · ★★☆ · programming · `lab-lab09-tree`

4.1 Δημιουργήστε το αρχείο `tree.c` και ορίστε μία αυτοαναφορική δομή δυαδικού δένδρου ακεραίων αριθμών.

```
typedef struct tnode *Treeptr;

struct tnode {
    int data;
    Treeptr left;
    Treeptr right;
};
```

4.2 Ένα ταξινομημένο δυαδικό δένδρο είναι ένα δυαδικό δένδρο στο οποίο κάθε κόμβος έχει στο αριστερό του υποδένδρο αριθμούς μικρότερους από τον ίδιο και στο δεξί του υποδένδρο αριθμούς μεγαλύτερους από τον ίδιο. Η ιδιότητα αυτή ισχύει αναδρομικά για κάθε υποδένδρο.

Ορίστε την αναδρομική συνάρτηση:

```
Treeptr addtree(Treeptr p, int x);
```

Η συνάρτηση προσθέτει έναν αριθμό `x` στο ταξινομημένο δυαδικό δένδρο `p`, διατηρώντας το ταξινομημένο, και επιστρέφει το νέο δένδρο. Ο αλγόριθμος λειτουργεί ως εξής:

- Αν το δένδρο `p` είναι κενό (`NULL`), δημιουργείται ένας νέος κόμβος που περιέχει τον αριθμό `x`.

- Αν ο αριθμός που περιέχει ο τρέχων κόμβος είναι μεγαλύτερος από το x, η συνάρτηση καλείται για το αριστερό παιδί του κόμβου.
- Αν ο αριθμός είναι μικρότερος από το x, η συνάρτηση καλείται για το δεξί παιδί.
- Αν ο αριθμός είναι ίσος με το x, δεν γίνεται εισαγωγή.

Τροποποιήστε τη συνάρτηση `main` ώστε να διαβάζει αριθμούς από το πληκτρολόγιο μέχρι το τέλος της εισόδου και να τους προσθέτει στο δένδρο.

4.3 Κατασκευάστε την αναδρομική συνάρτηση:

```
void treeprint(Treeptr p);
```

Η συνάρτηση δέχεται ως όρισμα το δένδρο και εκτυπώνει τα περιεχόμενά του με in-order διάσχιση. Ο αλγόριθμος λειτουργεί ως εξής:

1. Αν το δένδρο είναι κενό (`NULL`), η συνάρτηση επιστρέφει.
2. Διαφορετικά, εκτελούνται τα εξής βήματα:
   - Καλείται η `treeprint` για το αριστερό παιδί.
   - Εκτυπώνεται η τιμή του τρέχοντος κόμβου.
   - Καλείται η `treeprint` για το δεξί παιδί.

Καλέστε τη `treeprint` από τη συνάρτηση `main` για να εκτυπώσετε το δένδρο που κατασκευάσατε.

Υπόδειξη Το κλειδί στην `addtree` είναι ότι επιστρέφει το (πιθανώς νέο) υποδένδρο: την τιμή της αναδρομικής κλήσης πρέπει να την αναθέσετε πίσω στο `left` ή στο `right` του κόμβου, αλλιώς ο νέος κόμβος χάνεται. Νέοι κόμβοι ξεκινούν με `NULL` παιδιά. Η in-order διάσχιση ταξινομημένου δένδρου τυπώνει τους αριθμούς σε αύξουσα σειρά, κάτι που σας δίνει έναν εύκολο έλεγχο ορθότητας.

Εργασίες (A22.15–A22.17)

A22.15 · Zoomba: συντομότερη διαδρομή σε δωμάτιο

A22.15 · *Εργασία 3 (2023-24), Άσκηση 1* · ★★★ · programming · `hw-2023-hw3-zoomba`

Η Εργασία 3 είναι προαιρετικά ομαδική (ομάδες μέχρι 2 άτομα) και κάθε repository πρέπει να έχει ένα αρχείο `AUTHORS` με μία γραμμή για κάθε άτομο (sdi, GitHub username, όνομα).

Οι ρομποτικές σκούπες Roomba κάνουν τυχαίες βόλτες στον χώρο αντί να πάνε κατευθείαν στο σημείο που χρειάζεται καθάρισμα, με αποτέλεσμα να κολλάνε (Σχήμα 1 της εκφώνησης) ή να αδειάζει η μπαταρία τους. Η δική μας ρομποτική σκούπα, η Zoomba, διαθέτει ραντάρ που σκανάρει το δωμάτιο, δημιουργεί τον ”χάρτη” του και εντοπίζει το σημείο που χρειάζεται επειγόντως καθαρισμό. Αυτό που μας λείπει είναι ένα υποσύστημα που παίρνει τον χάρτη του δωματίου και το σημείο-στόχο καθαρισμού και επιστρέφει τις κινήσεις που πρέπει να

ακολουθήσει η Zoomba μας. Η γραφή αυτού του υποσυστήματος είναι το ζητούμενο αυτής της άσκησης.

Τεχνικές Προδιαγραφές

- Repository Name: `progintro/hw3-<YourTeamName>`
- README Filepath: `zoomba/README.md`
- C Filepath: `zoomba/src/zoomba.c`
- Το πρόγραμμα θα πρέπει να δέχεται από την πρότυπη είσοδο (stdin) τα δεδομένα του δωματίου στο οποίο βρίσκεται. Συγκεκριμένα, η είσοδος θα έχει την ακόλουθη γενική μορφή:

  ```
  ROOM_DIMENSION_N
  ZOOMBA_X ZOOMBA_Y ZOOMBA_TARGET_X ZOOMBA_TARGET_Y
  ROOM_ENCODED_AS_1s_AND_0s
  ```

  Η πρώτη γραμμή θα περιέχει την διάσταση του τρέχοντος δωματίου ως ακέραιο (μόνο τετράγωνα δωμάτια είναι δεκτά για το πρωτότυπό μας), την τρέχουσα θέση της zoomba σε καρτεσιανές συντεταγμένες (ακέραιοι), την θέση στόχο την οποία πρέπει να καθαρίσει σε καρτεσιανές συντεταγμένες (ακέραιοι) και τέλος τον χάρτη του τρέχοντος δωματίου κωδικοποιημένο ως 1 και 0 - όπου 1 δηλώνει την παρουσία κάποιου εμποδίου ενώ 0 δηλώνει ανοιχτό χώρο. Οποιαδήποτε είσοδος δεν ακολουθεί την παραπάνω μορφή θα πρέπει να κάνει το πρόγραμμα να τερματίζει με κωδικό εξόδου (exit code) 1. Ομοίως και αν οι συντεταγμένες που δόθηκαν είναι λανθασμένες - για παράδειγμα αν η θέση της zoomba ή του στόχου είναι πάνω σε εμπόδιο - το πρόγραμμα πρέπει να τερματίζει με κωδικό εξόδου

  1. Για παράδειγμα, μια πιθανή είσοδος είναι η ακόλουθη - μπορείτε να την συγκρίνετε με την απεικόνιση στο Σχήμα 2:

  ```
  10
  1 8 8 1
  0000000000
  0110000000
  0110000000
  0100000000
  0110000000
  0111111100
  0000000000
  0000000000
  0000001110
  0000000000
  ```

  Το Σχήμα 2 της εκφώνησης δείχνει τον χάρτη ως πλέγμα 10 × 10 με τα εμπόδια, την αρχική θέση της Zoomba (ένα γρανάζι, πάνω δεξιά) και την περιοχή που χρειάζεται καθάρισμα

(κόκκινο X, κάτω αριστερά). Μια από τις λύσεις που θέλουμε να επιλέξει η Zoomba αποτελείται από 14 βήματα: `DDDDDDLLLLLLD`. Παρατηρήστε ότι υπάρχουν πάνω από μία βέλτιστες λύσεις.

- Στην έξοδό του το πρόγραμμά σας πρέπει να τυπώνει τις κινήσεις που πρέπει να πραγματοποιήσει η zoomba προκειμένου να φτάσει στον στόχο στην πρότυπη έξοδο (stdout). Για εξοικονόμηση υλικών, η zoomba μας μπορεί να κινηθεί μόνο πάνω (U - Up), κάτω (D - Down), αριστερά (L - Left) και δεξιά (R - Right) οπότε όλες οι λύσεις πρέπει να εκφραστούν με αυτούς τους τέσσερις χαρακτήρες. Η λύση **πρέπει να είναι η συντομότερη δυνατή σε αριθμό κινήσεων** της zoomba (θεωρούμε ότι όλες οι κινήσεις έχουν το ίδιο ενεργειακό κόστος). Για παράδειγμα, η λύση που εικονίζεται στο Σχήμα 2 θα έχει την ακόλουθη μορφή:

  ```
  DDDDDDLLLLLLD
  ```

  Παρατηρήστε ότι είναι η βέλτιστη από απόψεως αριθμού κινήσεων - φτάνει στον στόχο μόλις σε 14 βήματα. Αφού τυπώσει την λύση, το πρόγραμμα πρέπει να τερματίζει με κωδικό εξόδου (exit code) 0. Εάν δεν υπάρχει καμία λύση επειδή η θέση δεν είναι προσβάσιμη, το πρόγραμμά μας πρέπει να τυπώνει ”0” και πάλι να τερματίζει με κωδικό εξόδου 0.

- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr): `gcc -Ofast -Wall -Wextra -Werror -pedantic -o zoomba zoomba.c -lm`
- **Ένα αρχείο που να περιέχει στοιχεία εισόδου και ένα εξόδου διαφορετικά από αυτά της άσκησης. Συγκεκριμένα προτείνουμε να βάλετε έναν συνδυασμό που θεωρείται ότι είναι δύσκολος να γίνει σωστός**.
  - input Filepath: `zoomba/test/input`
  - output Filepath: `zoomba/test/output`
- Μέγιστη διάσταση δωματίου: 10000.
- Πρέπει να ολοκληρώνει την εκτέλεση μέσα σε: 30 δευτερόλεπτα.

Παρακάτω παραθέτουμε την αλληλεπίδραση με μια ενδεικτική λύση:

```
$ echo | ./zoomba
Incorrect room input provided.
$ echo $?
1
$ cat impossible
10
1 8 8 1
0100000000
```

```
0110000000
0110000000
0100000000
0110000000
0111111111
0000000000
0000000000
0000001110
0000000000
$ ./zoomba < impossible
0
$ echo $?
0
$ cat input
10
1 8 8 1
0000000000
0110000000
0110000000
0100000000
0110000000
0111111100
0000000000
0000000000
0000001110
0000000000
$ ./zoomba < input
DDDDDDLLLLDLLL
```

Παρατηρήστε ότι η λύση είναι διαφορετική απ' ό,τι εικονίζεται στο Σχήμα 2 αλλά παρόλα αυτά είναι σωστή και μία από τις βέλτιστες. Παρακάτω παραθέτουμε μερικά ακόμα παραδείγματα. Φυσικά καλό είναι να δοκιμάσετε και εσείς το πρόγραμμά σας με διάφορα δωμάτια που είναι εντός προδιαγραφών ώστε να ελέγξετε την ορθότητα και την αποδοτικότητα της λύσης σας.

```
$ cat maze
10
1 8 8 1
1111111101
0000000101
0111110101
0001000101
1101011101
0001010001
0111010111
```

```
0101010101
0001010101
0001000001
$ ./zoomba < maze
DDDDLLDDDDLLUUUUUURRUULLLLLLDDRRDDLLDDDR
```

Στον φάκελο https://github.com/progintro/data/tree/main/zoomba μπορείτε να βρείτε και μεγαλύτερα παραδείγματα (τα μεγαλύτερα είναι συμπιεσμένα για λόγους χώρου, μπορείτε να τα αποσυμπιέσετε χρησιμοποιώντας την εντολή `tar`). Μερικές ενδεικτικές εκτελέσεις ακολουθούν (έχουμε "κόψει" το output για λόγους χώρου):

```
$ time ./zoomba < tricky1000
UUUUUUU...UUUUUUUUULUURRRRR...RRRRRRRRRRR

real    0m0.197s
user    0m0.145s
sys     0m0.052s
$ time ./zoomba < impossible1000
0

real    0m0.155s
user    0m0.121s
sys     0m0.034s

time ./zoomba < maze100
DDDDLLL...DDLLDDD

real    0m0.003s
user    0m0.003s
sys     0m0.000s
$ ./zoomba < maze100 | wc -c
2472
$ ./zoomba < maze100 | md5sum -
5b2b80b22c12408e82e10b2bbb625c48
$ ./zoomba < guernica10000 | wc -c
19173
$ time ./zoomba < guernica10000 | md5sum -
873bb6d49f1d6a6e36dc261cf3d79788  -

real    0m11.406s
user    0m10.233s
sys     0m1.176s
```

Προκειμένου να βελτιστοποιήσετε την λύση σας ίσως χρειαστεί (όχι αναγκαστικά!) να εμβαθύνετε σε θέματα αναζήτησης πέρα από αυτά που συζητήσαμε στο μάθημα - οι ενότητες

2-4 της Τεχνητής Νοημοσύνης της σχολής είναι ένα παράδειγμα από πηγές που μπορούν να σας βοηθήσουν σε αυτήν την κατεύθυνση.

Στο αρχείο `README.md` πρέπει να προσθέσετε οποιεσδήποτε παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης. Για τις υποβολές με την καλύτερη χρονική απόδοση θα υπάρχει bonus βαθμολογία: η πρώτη υποβολή θα λάβει +100%, η δεύτερη +70% και η τρίτη +40%.

Υπόδειξη Το δωμάτιο είναι ένας γράφος με κόμβους τα ελεύθερα κελιά και ακμές τις τέσσερις κινήσεις· η συντομότερη διαδρομή σε γράφο χωρίς βάρη βρίσκεται με αναζήτηση κατά πλάτος (BFS) με ουρά, αποθηκεύοντας για κάθε κελί από πού ήρθατε ώστε να ανακατασκευάσετε τη διαδρομή στο τέλος (ανάποδα). Για N = 10000 ο χάρτης έχει 10^8 κελιά: δεσμεύστε δυναμικά συμπαγείς πίνακες (π.χ. ένα byte ανά κελί) και αποφύγετε την αναδρομή, που θα γέμιζε τη στοίβα. Από το παράδειγμα του Σχήματος 2 βγάλτε ποια συντεταγμένη είναι η γραμμή και ποια η στήλη, και ελέγξτε προσεκτικά κάθε παράβαση της μορφής εισόδου.

A22.16 · Νέα Μηχανή Σκακιού (chess engine)

A22.16 · *Εργασία 3 (2024-25), Άσκηση 1* · ★★★ · programming · `hw-2024-hw3-chess`

Νέα Μηχανή Σκακιού (chess engine - 100 Μονάδες)

Η εργασία είναι **προαιρετικά ομαδική με ομάδες μέχρι 2 άτομα**. Για να είναι ξεκάθαρο ποια άτομα συνεργάστηκαν, **κάθε repository πρέπει να έχει ένα AUTHORS αρχείο** με μία γραμμή για κάθε άτομο, με πρώτο το sdi σας, μετά το github username και τέλος το όνομά σας:

```
$ cat AUTHORS
sdi2400998,mourmourakis-2006,ΘΑΝΟΣ ΜΟΥΡΜΟΥΡΑΚΗΣ
sdi2400999,xifias-2006,ΚΩΣΤΑΣ ΞΙΦΙΑΣ
```

Υποβολές χωρίς σωστό AUTHORS αρχείο δεν θα εξεταστούν. Αυτό ισχύει και για ατομικές υποβολές.

Τα περισσότερα παιχνίδια έχουν παρεμφερή δομή: (1) το παιχνίδι ξεκινάει σε μια αρχική κατάσταση, (2) ένας από τους παίκτες παίζει πρώτος επιλέγοντας μια από τις δυνατές κινήσεις που έχει στην διάθεσή του αλλάζοντας την κατάσταση του παιχνιδιού και (3) στην συνέχεια δίνει την σειρά του στον επόμενο παίκτη. Ο χώρος καταστάσεων ενός παιχνιδιού μπορεί να φανταστεί σαν ένα m-αδικό δέντρο βάθους n (m οι επιλογές σε κάθε θέση, n οι γύροι), με περίπου m^n καταστάσεις. Η τρίλιζα και η ντάμα έχουν ”λυθεί” (διατρέξαμε όλον τον χώρο καταστάσεών τους), αλλά το σκάκι όχι: ο Claude Shannon εκτίμησε τις δυνατές παρτίδες στο 10^{120} και τις θέσεις της σκακιέρας στο 10^{40}. Οι μηχανές σκακιού εξερευνούν αυτόν τον χώρο με τεχνικές αναζήτησης / ευριστικές (minimax, negamax) και με ιδιαίτερα αποδοτικές υλοποιήσεις, συνήθως σε C και C++.

Πόσο εύκολο είναι να φτιάξουμε μια μηχανή σκακιού σήμερα; Μπορεί να κερδίσει σχετικά απλούς αντιπάλους που απλά παίζουν τυχαίες κινήσεις; Αυτό θα είναι και το αντικείμενο αυτής της άσκησης, όπου καλείστε να υλοποιήσετε τον πυρήνα μιας μηχανής σκακιού. Ευτυχώς, δεν χρειάζεται να ξεκινήσετε από το μηδέν, μπορείτε να βρείτε στο διαδίκτυο πολλές πληροφορίες για το πως δομούνται σκακιστικές μηχανές καθώς και ποιες βελτιστοποιήσεις είναι δημοφιλείς (Chess Programming Wiki).

Τεχνικές Προδιαγραφές

- Repository Name: progintro/hw3-<YourTeamName>
- README Filepath: README.md
- Όλα τα αρχεία κώδικα (.c και .h) που θα υλοποιήσετε πρέπει να τοποθετηθούν μέσα σε έναν φάκελο `src`.
- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr): `make`

  Στο repository της άσκησης σας δίνεται μια πιθανή οργάνωση κώδικα μαζί με ένα Makefile. Μπορείτε να αλλάξετε τα πάντα στο project σας, αλλά θέλουμε να επιβεβαιώσετε πως η εντολή `make` συνεχίζει να δουλεύει και να παράγει εκτελέσιμα από την μηχανή σκακιού σας.
- Το πρόγραμμά σας πρέπει να προσφέρει τις ακόλουθες δύο διεπαφές:
  1. **Μέσω γραμμής εντολών**. Να δέχεται 3 ορίσματα από την γραμμή εντολών: fen moves timeout. Το πρώτο όρισμα (fen) θα είναι η τρέχουσα κατάσταση της σκακιέρας σε Forsyth-Edwards Notation. Το δεύτερο όρισμα (moves) θα είναι όλες οι δυνατές κινήσεις για αυτήν την θέση σε standard algebraic notation και χωρισμένες με τον κενό χαρακτήρα " ". Τέλος, το 3ο όρισμα θα είναι ο χρόνος σε δευτερόλεπτα που έχει το πρόγραμμά σας για να αποφασίσει ποια από τις δοθείσες κινήσεις επιλέγει. Το πρόγραμμά σας πρέπει να τυπώνει την θέση (index) της κίνησης που επιλέγει και στην συνέχεια να επιστρέφει με exit code 0.
  2. **Μέσω κλήσης συνάρτησης (για χρήση σε Web Assembly)**. Το πρόγραμμά σας πρέπει να περιέχει μια συνάρτηση `int choose_move(char * fen, char * moves, int timeout)` η οποία θα δέχεται ορίσματα με σημασιολογία όμοια με παραπάνω και θα επιστρέφει το index της κίνησης που επιλέγει. Όταν γίνεται compile σε Web Assembly, το πρόγραμμά σας θα πρέπει να αποφεύγει να τυπώνει στο stdout/stderr καθώς αυτά δεν είναι επιτρεπτά σε αυτό το περιβάλλον.
- Το πρόγραμμά σας πρέπει να κερδίζει αποφασιστικά (άνω του 60%) των παιχνιδιών με αντίπαλο μια μηχανή Random, δηλαδή μια μηχανή που επιλέγει κινήσεις τυχαία.
- Το πρόγραμμά σας όταν μεταγλωττιστεί δεν θα πρέπει να ξεπερνά το 1MB στον δίσκο.

- Το πρόγραμμά / συνάρτησή σας πρέπει να ολοκληρώνει την εκτέλεση μέσα στον αριθμό των δευτερολέπτων που δίνονται ως όρισμα.

Παρακάτω παραθέτουμε την αλληλεπίδραση με μια ενδεικτική λύση:

```
$ make TARGET=engine
$ ./engine "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1" \
           "a3 a4 b3 b4 c3 c4 d3 d4 e3 e4 f3 f4 g3 g4 h3 h4 Na3 Nc3 Nf3 Nh3" \
           3
9
```

Παρατηρούμε ότι δώσαμε στο πρόγραμμά μας (το οποίο ονομάσαμε engine) 3 ορίσματα: (1) την αρχική μας σκακιέρα σε FEN format, (2) τις κινήσεις που μπορεί να παίξει χωρισμένες με κενό (20 στον αριθμό) και (3) 3 δευτερόλεπτα για να αποφασίσει ποια κίνηση επιθυμεί να παίξει. Το πρόγραμμά μας αποφάσισε να παίξει την κίνηση στην θέση 9 (προσοχή οι κινήσεις ξεκινάνε από το 0) και επομένως επέλεξε την κίνηση e4 (να κινήσει επομένως το λευκό πιόνι κατά δύο τετράγωνα μπροστά). Αυτό ήταν! Αν το πρόγραμμά σας υποστηρίζει την παραπάνω δυνατότητα (να τυπώνει τον σωστό ακέραιο), είναι σε θέση να χρησιμοποιηθεί ως μηχανή σκακιού. Το πόσο καλά θα τα πάει εξαρτάται από τις επιλογές που θα κάνει.

Στο αρχείο README.md πρέπει να προσθέσετε την περιγραφή του project σας καθώς και ποιες πηγές χρησιμοποιήσατε κατά την υλοποίηση. Περιμένουμε να δούμε write ups υψηλής ποιότητας, καθώς αυτό είναι ένα project που θα μπορούσατε να δημοσιεύσετε μετά το πέρας της άσκησης και να το βάλετε στο portfolio σας. Σκακιστικές μηχανές που επιτυγχάνουν υψηλότερες επιδόσεις, θα πάρουν bonus μονάδες με μετρική που θα ανακοινωθεί στην συνέχεια, (πιθανώς να χρησιμοποιήσουμε την μετρική ELO).

Υπόδειξη Χωρίστε το project σε ενότητες (αναπαράσταση σκακιέρας από FEN, παραγωγή και εκτέλεση κινήσεων, αξιολόγηση θέσης, αναζήτηση) με δικό του `.c`/`.h` η καθεμία. Ακόμη και μια απλή αξιολόγηση με το υλικό (άθροισμα αξιών κομματιών) μαζί με minimax/negamax λίγων κινήσεων βάθους αρκεί για να κερδίζει μια τυχαία μηχανή· το alpha-beta pruning και η επαναληπτική εκβάθυνση (iterative deepening) σας επιτρέπουν να σέβεστε το timeout. Προσοχή: οι δοσμένες κινήσεις είναι σε SAN, οπότε πρέπει να τις αντιστοιχίσετε στις δικές σας, και η `choose_move` δεν πρέπει να τυπώνει τίποτα.

A22.17 · Νέα Μηχανή Go (goteam)

A22.17 · *Εργασία 3 (2025-26), Άσκηση 1* · ★★★ · programming · `hw-2025-hw3-goteam`

Η τρίτη και τελευταία εργασία είναι **προαιρετικά ομαδική, με ομάδες μέχρι 2 άτομα**. Αν δουλέψετε ως ομάδα, ένας/μία από εσάς αποδέχεται την πρόσκληση και δημιουργεί την ομάδα με συγκεκριμένο όνομα και ο/η συνεργάτης επιλέγει να προστεθεί σε αυτήν (**προσοχή**: **μην επιλέξετε λάθος ομάδα**)· το repository είναι κοινό, οπότε προσοχή στα conflicts! Κάθε

repository **πρέπει** να έχει ένα αρχείο `AUTHORS` με μία γραμμή για κάθε άτομο, με πρώτο το sdi σας, μετά το github username και τέλος το όνομά σας:

```
$ cat AUTHORS
sdi2500998,mourmourakis-2007,ΘΑΝΟΣ ΜΟΥΡΜΟΥΡΑΚΗΣ
sdi2500999,lavrakis-2007,ΤΑΚΗΣ ΛΑΒΡΑΚΗΣ
```

Υποβολές χωρίς σωστό AUTHORS αρχείο δεν θα εξεταστούν. Αυτό ισχύει και για ατομικές υποβολές.

Τα περισσότερα παιχνίδια ξεκινούν από μια αρχική κατάσταση και οι παίκτες εναλλάσσονται επιλέγοντας κινήσεις μέχρι κάποιο κριτήριο να δείξει νίκη, ισοπαλία ή ήττα· ο χώρος καταστάσεων μοιάζει με ένα m-αδικό δέντρο βάθους n με περίπου m^n καταστάσεις. Η τρίλιζα λύνεται εξαντλητικά σε κλάσματα δευτερολέπτου και η ντάμα λύθηκε το 2007, όμως το Go, σε ταμπλό 19x19 με περίπου 10^{170} θέσεις, θεωρούνταν άφταστο για την τεχνητή νοημοσύνη μέχρι που ο AlphaGo της DeepMind κέρδισε 4–1 τον Lee Sedol το 2016. Σε αυτήν την άσκηση καλείστε να υλοποιήσετε τον πυρήνα μιας μηχανής Go ικανό να παίζει με αντίπαλο είτε έναν άνθρωπο, τον χρήστη του προγράμματος, είτε ένα άλλο πρόγραμμα, μέσω ενός ελεγκτή/διαιτητή. Μπορείτε να βρείτε στο διαδίκτυο πολλές πληροφορίες για το πως δομούνται μηχανές Go.

Τεχνικές Προδιαγραφές

- Repository Name: `progintro/hw3-<YourTeamName>`
- README Filepath: `README.md`
- Όλα τα αρχεία κώδικα (`.c` και `.h`) που θα υλοποιήσετε πρέπει να τοποθετηθούν μέσα σε έναν φάκελο `src`.
- Το project C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με την εντολή `make` σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr).

  Στο repository της άσκησης σας δίνεται μια πιθανή οργάνωση κώδικα μαζί με ένα Makefile το οποίο μπορείτε να συμπληρώσετε όπως θέλετε. Μπορείτε να αλλάξετε τα πάντα στο project σας, αλλά θέλουμε να επιβεβαιώσετε πως η εντολή `make` συνεχίζει να δουλεύει και να παράγει εκτελέσιμα για την Go μηχανή σας. **Το εκτελέσιμό σας πρέπει να λέγεται "goteam"**—άλλα ονόματα δεν θα γίνουν δεκτά. Ακολουθήστε τις συμβάσεις που δίνονται στο αρχικό Makefile.
- Το πρόγραμμά σας πρέπει να υποστηρίζει τις εντολές του πρωτοκόλλου Go Text Protocol (GTP) προκειμένου να επικοινωνεί με τον χρήστη ή το πρόγραμμα διαιτητή. Δείτε τις ενδεικτικές εκτελέσεις στην συνέχεια για τις βασικές δυνατότητες που πρέπει να υποστηρίζει.

- Υπάρχουν διαφορετικές εκδοχές κανόνων για το Go, στα πλαίσια της άσκησης θα ακολουθήσουμε την πιο δημοφιλή εκδοχή, δηλαδή τους κινέζικους κανόνες.
- Προκειμένου να συμμετέχετε στον διαγωνισμό Go που θα γίνει στο τέλος του εξαμήνου, οι εντολές του πρωτοκόλλου GTP που θα υλοποιήσετε πρέπει να είναι επαρκείς για να είναι σε θέση το πρόγραμμά σας να συνεργάζεται με τους ελεγκτές/διαιτητές που θα βρείτε στην διεύθυνση https://github.com/progintro/goref. Προκειμένου να κατεβάσετε την τελευταία έκδοση του goref τοπικά (**προσοχή**: **θα υπάρξουν αρκετές ανανεώσεις του goref μέχρι το τέλος του διαγωνισμού** προκειμένου να διαχειριστούμε την όποια ... εφευρετικότητα έχουν οι λύσεις σας), μπορείτε να τρέξετε τις ακόλουθες εντολές:

  ```
  curl -O https://raw.githubusercontent.com/progintro/goref/main/goref.py
  chmod +x goref.py
  ```

Μία ενδεικτική εκτέλεση του προγράμματος (έστω ότι το εκτελέσιμο ονομάζεται `goteam`) είναι η εξής:

```
$ ./goteam
boardsize 5
=

komi 0.5
=

clear_board
=

showboard
=
   A B C D E
 5 . . . . . 5
 4 . . . . . 4
 3 . . . . . 3
 2 . . . . . 2
 1 . . . . . 1
   A B C D E

genmove black
= C3

showboard
=
   A B C D E
 5 . . . . . 5
 4 . . . . . 4
```

```
 3 . . X . . 3
 2 . . . . . 2
 1 . . . . . 1
   A B C D E

play white C4
=

genmove black
= B4

showboard
=
   A B C D E
 5 . . . . . 5
 4 . X O . . 4
 3 . . X . . 3
 2 . . . . . 2
 1 . . . . . 1
   A B C D E

play white B3
=

genmove black
= B2

showboard
=
   A B C D E
 5 . . . . . 5
 4 . X O . . 4
 3 . O X . . 3
 2 . X . . . 2
 1 . . . . . 1
   A B C D E

play white D3
=

genmove black
= A3
```

```
showboard
=
   A B C D E
 5 . . . . . 5
 4 . X O . . 4
 3 X . X O . 3
 2 . X . . . 2
 1 . . . . . 1
   A B C D E

play white D5
=

genmove black
= E4

showboard
=
   A B C D E
 5 . . . O . 5
 4 . X O . X 4
 3 X . X O . 3
 2 . X . . . 2
 1 . . . . . 1
   A B C D E

play white C2
=

genmove black
= B3

showboard
=
   A B C D E
 5 . . . O . 5
 4 . X O . X 4
 3 X X X O . 3
 2 . X O . . 2
 1 . . . . . 1
   A B C D E

play white D1
```

```
=

genmove black
= D4

showboard
=
   A B C D E
 5 . . . O . 5
 4 . X O X X 4
 3 X X X O . 3
 2 . X O . . 2
 1 . . . O . 1
   A B C D E

play white E3
=

genmove black
= C5

showboard
=
   A B C D E
 5 . . X O . 5
 4 . X . X X 4
 3 X X X O O 3
 2 . X O . . 2
 1 . . . O . 1
   A B C D E

play white E2
=

genmove black
= B1

showboard
=
   A B C D E
 5 . . X O . 5
 4 . X . X X 4
 3 X X X O O 3
```

```
 2 . X O . O 2
 1 . X . O . 1
   A B C D E

play white C1
=

genmove black
= PASS

showboard
=
   A B C D E
 5 . . X O . 5
 4 . X . X X 4
 3 X X X O O 3
 2 . X O . O 2
 1 . X O O . 1
   A B C D E

play white pass
=

final_score
= B+8.5

quit
=

$
```

Κύπελλο Go (BONUS βαθμολογία). Οι μηχανές που θα υποβληθούν θα συμμετάσχουν σε κύπελλο Go που θα διεξαχθεί μετά το τέλος της προθεσμίας. Για τις τρεις καλύτερες υλοποιήσεις, θα υπάρξει επιβράβευση στη βαθμολογία τους κατά 100%, 70% και 40%, κατά σειρά. Μπορείτε να δοκιμάσετε τις δυνατότητες της υλοποίησής σας χρησιμοποιώντας προγράμματα διαιτητές όπως το goref παραπάνω ως εξής:

```
$ ./goref.py --black "gnugo --mode gtp --chinese-rules" \
    --white "./goteam" --games 10 --main-time 60 --verbose
[blackcmd:B >>] name
[blackcmd:B <<] = GNU Go
[blackcmd:B <<]
...
[whitecmd:W >>] name
```

```
[whitecmd:W <<] = Go Team
[whitecmd:W <<]
[whitecmd:W(Go Team) >>] version
[whitecmd:W(Go Team) <<] = 1.0
[whitecmd:W(Go Team) <<]
...
[whitecmd:W(Go Team) v1.0 >>] time_settings 60 0 0
[whitecmd:W(Go Team) v1.0 <<] =
[whitecmd:W(Go Team) v1.0 <<]
...
[whitecmd:W(Go Team) v1.0 >>] time_left B 60 0
[whitecmd:W(Go Team) v1.0 <<] =
[whitecmd:W(Go Team) v1.0 <<]
[blackcmd:B(GNU Go) v3.8 >>] genmove B
[blackcmd:B(GNU Go) v3.8 <<] = Q16
[blackcmd:B(GNU Go) v3.8 <<]
[whitecmd:W(Go Team) v1.0 >>] play B Q16
[whitecmd:W(Go Team) v1.0 <<] =
[whitecmd:W(Go Team) v1.0 <<]
...
[whitecmd:W(Go Team) v1.0 >>] time_left W 60 0
[whitecmd:W(Go Team) v1.0 <<] =
[whitecmd:W(Go Team) v1.0 <<]
...
[whitecmd:W(Go Team) v1.0 >>] genmove W
[whitecmd:W(Go Team) v1.0 <<] = D4
[whitecmd:W(Go Team) v1.0 <<]
...
Games: 10
Overall (by command slot):
  black_cmd: 8
  white_cmd: 2
  draws: 0
...
```

Η παραπάνω εκτέλεση ελέγχει την υλοποίησή μας με λευκά ενάντια στην μηχανή GNU Go με μαύρα (`apt install gnugo` σε σύστημα Ubuntu) και μας επιτρέπει να ελέγξουμε την πληρότητα αλλά και την αποδοτικότητα της υλοποίησής μας. Αν θέλετε να οπτικοποιήσετε τα αποτελέσματα της μηχανής σας (ή να παίξετε ένα παιχνίδι με graphical user interface) υπάρχουν αρκετές υλοποιήσεις ανοιχτού κώδικα που μπορείτε να χρησιμοποιήσετε, π.χ. το gogui.

Επίλογος Πριν την υποβολή της εργασίας, μην ξεχάσετε το `README.md`, μέσα στο οποίο πρέπει να συμπεριλάβετε τις όποιες παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης! Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.

Υπόδειξη Χώρισε το project σε επίπεδα: ανάγνωση και απάντηση εντολών GTP (γραμμή-γραμμή από το stdin, απάντηση = ή ? και κενή γραμμή), αναπαράσταση ταμπλό, κανόνες, και τέλος επιλογή κίνησης. Ο πυρήνας των κανόνων είναι η εύρεση μιας ομάδας συνδεδεμένων πετρών και των ελευθεριών της με BFS/DFS σε πλέγμα· με αυτήν υλοποιείς αιχμαλωσίες, απαγόρευση αυτοκτονίας, ko και την καταμέτρηση περιοχής των κινέζικων κανόνων. Ξεκίνα με μια μηχανή που παίζει απλώς νόμιμες κινήσεις (και ξέρει να κάνει pass) και βελτίωσέ την μετά, π.χ. με προσομοιώσεις τυχαίων παρτίδων, προσέχοντας τον χρόνο που δίνει το `time_left`.

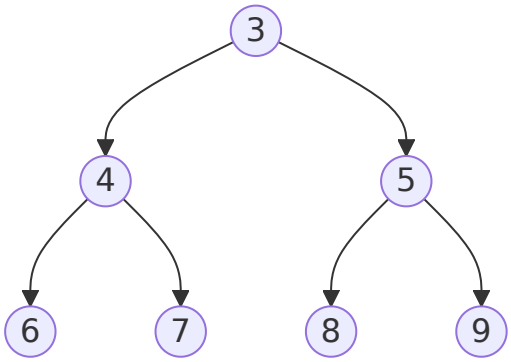
Θέματα εξετάσεων (A22.18–A22.20)

A22.18 · Αθροιστής Δέντρων - sumtree

A22.18 · *Εξέταση Ιανουαρίου 2025, Θέμα 4* · ★☆☆ · programming · `exam-2025-jan-q4`

Αθροιστής Δέντρων - sumtree [20 Μονάδες]

Γράψτε μια συνάρτηση `sumtree` η οποία παίρνει ως όρισμα ένα δέντρο ακεραίων τύπου `Tree` και επιστρέφει το άθροισμα όλων των κόμβων του δέντρου. Για παράδειγμα, για το ακόλουθο δέντρο:



Σχήμα: το δέντρο του παραδείγματος.

περιμένουμε να μας επιστρέψει την τιμή: 42 = 3 + 4 + 5 + 6 + 7 + 8 + 9. Ποια είναι η χρονική και η χωρική πολυπλοκότητα του αλγορίθμου σας (6/20 της βαθμολογίας); Ο τύπος `Tree` δίνεται παρακάτω:

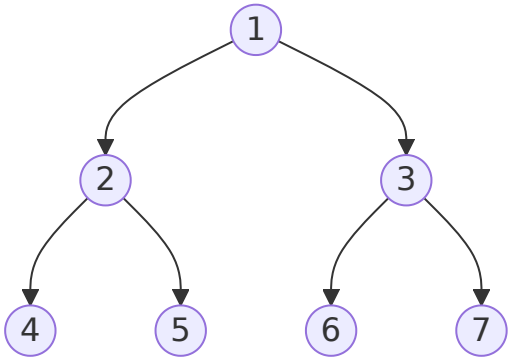
```c
typedef struct node {
  int value;
  struct node * left;
  struct node * right;
} * Tree;
```

Υπόδειξη Το άθροισμα ενός δέντρου ορίζεται φυσικά αναδρομικά, μέσω των αθροισμάτων του αριστερού και του δεξιού υποδέντρου. Ξεκινήστε από τη βασική περίπτωση του κενού δέντρου (`NULL`). Για τη χωρική πολυπλοκότητα, σκεφτείτε το βάθος της στοίβας των αναδρομικών κλήσεων στη χειρότερη περίπτωση (π.χ. ένα δέντρο σαν λίστα).

A22.19 · Reverse Inorder Traversal

A22.19 · *Εξέταση Ιουλίου 2024, Θέμα 4* · ★★☆ · programming · `exam-2024-jul-q4`

Γράψτε μια συνάρτηση `reverse_inorder` η οποία παίρνει ως όρισμα ένα δέντρο ακεραίων τύπου `Tree` και τυπώνει τους αριθμούς των κόμβων σε σειρά reverse inorder traversal, δηλαδή όπως βλέπουμε τους αριθμούς από δεξιά προς τα αριστερά (πρώτο το δεξί παιδί, μετά ο γονιός και στην συνέχεια το αριστερό). Για παράδειγμα, για το ακόλουθο δέντρο:



Σχήμα: το δέντρο του παραδείγματος.

περιμένουμε να εκτυπωθεί η ακολουθία: 7 3 6 1 5 2 4. Ποια είναι η χρονική και η χωρική πολυπλοκότητα του αλγορίθμου σας (8/25 της βαθμολογίας); Ο τύπος `Tree` δίνεται παρακάτω:

```c
typedef struct node {
  int value;
  struct node * left;
  struct node * right;
} * Tree;
```

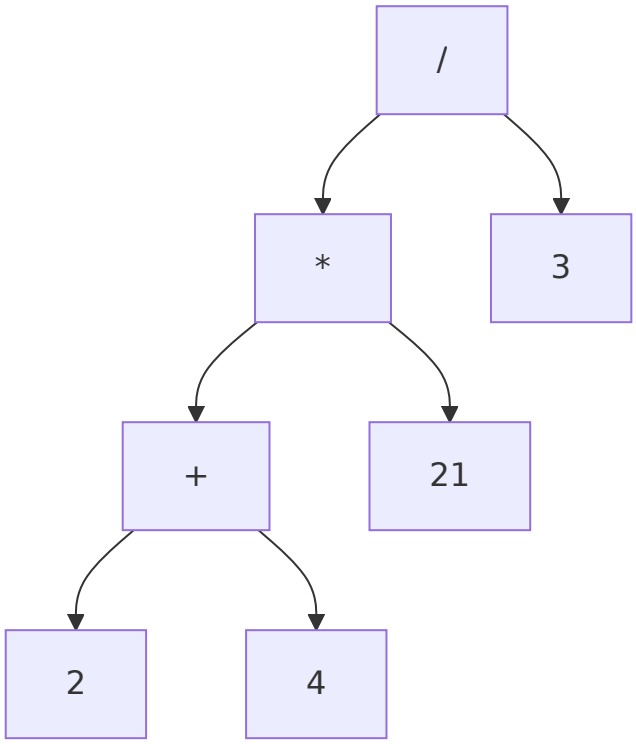
Υπόδειξη Ξεκινήστε από την αναδρομική inorder διάσχιση και αλλάξτε τη σειρά με την οποία επισκέπτεστε τα δύο υποδέντρα. Μην ξεχάσετε τη βασική περίπτωση του κενού δέντρου (`NULL`). Για τη χωρική πολυπλοκότητα, σκεφτείτε πόσο βαθιά μπορεί να φτάσει η στοίβα των αναδρομικών κλήσεων στη χειρότερη περίπτωση.

A22.20 · Διερμηνέας Αριθμητικών Εκφράσεων - eval

A22.20 · *Εξέταση Ιανουαρίου 2026, Θέμα 4* · ★★☆ · programming · `exam-2026-jan-q4`

Διερμηνέας Αριθμητικών Εκφράσεων - eval [25 Μονάδες]

Γράψτε μια συνάρτηση `eval` η οποία παίρνει ως όρισμα ένα δέντρο ακεραίων εκφράσεων τύπου `Expr` και επιστρέφει το αποτέλεσμα της αποτίμησης της έκφρασης. Για παράδειγμα, για την έκφραση (2 + 4) * 21 / 3 το δέντρο έκφρασης δείχνει ως εξής:



και αν δοθεί στην συνάρτηση eval, περιμένουμε να μας επιστραφεί η τιμή: 42 = (2 + 4) * 21 / 3.

1. Ποια είναι η χρονική και η χωρική πολυπλοκότητα του αλγορίθμου σας; (8/25)
2. Ποιον αλγόριθμο διάσχισης επιλέξατε και γιατί; (2/25)

Ο ορισμός του τύπου `Expr` δίνεται παρακάτω:

```
typedef enum {
```

```
  VALUE,  // current node is an integer value
  ADD,    // current node is addition of two nodes
  SUB,    // current node is subtraction of left minus right
  MUL,    // current node is the multiplication of two nodes
  DIV     // current node is the division of left by right
} exp_type;

typedef struct node {
  exp_type type;
  int value;
  struct node * left;
  struct node * right;
} * Expr;
```

Υπόδειξη Η φυσική λύση είναι αναδρομική: ένας κόμβος `VALUE` είναι η βάση της αναδρομής, ενώ για έναν τελεστή πρέπει πρώτα να ξέρετε τις τιμές *και των δύο* παιδιών· ποια διάσχιση (preorder, inorder, postorder) αντιστοιχεί σε αυτή τη σειρά; Ένα `switch` στο `type` ταιριάζει καλά. Για την πολυπλοκότητα σκεφτείτε πόσες φορές επισκέπτεστε κάθε κόμβο και πόσο βαθιά μπορεί να φτάσει η στοίβα κλήσεων· μην ξεχάσετε τη διαίρεση με το μηδέν και τον δείκτη `NULL`.

Κεφάλαιο 23: Οργάνωση Κώδικα

Ζέσταμα: από τις διαφάνειες (A23.1–A23.4)

A23.1 · Κλήση πριν από τον ορισμό

A23.1 · *Διάλεξη 23, διαφάνειες 19-21* · ★☆☆ · debug · `slides-lec23-implicit-declaration`

Το παρακάτω πρόγραμμα `prototype.c` μεταγλωττίζεται με μήνυμα `implicit declaration of function ‘print_err’`. Γιατί; Διορθώστε το χωρίς να μετακινήσετε τον ορισμό της `print_err`.

```
#include <stdio.h>

int main() {
  print_err("hello");
  return 0;
}
int print_err(char * msg) {
  return fprintf(stderr, "%s\n", msg);
}
```

```
$ gcc -o prototype prototype.c
prototype.c: In function ‘main’:
```

```
prototype.c:4:3: warning: implicit declaration of function ‘print_err’
[-Wimplicit-function-declaration]
    4 |   print_err("hello");
      |   ^~~~~~~~~
```

Μετά τη διόρθωση:

```
$ gcc -o prototype prototype.c
$ ./prototype
hello
```

Υπόδειξη Ο μεταγλωττιστής της C διαβάζει το αρχείο μία φορά, από πάνω προς τα κάτω. Τι χρειάζεται να ξέρει για την `print_err` όταν φτάσει στη γραμμή 4, και πώς του το λέτε χωρίς να γράψετε το σώμα της;

A23.2 · Ένα σύστημα 40.000 γραμμών

A23.2 · *Διαλέξεις 23–24, διαφάνειες 11 και 14 (διάλεξη 24: διαφάνειες 12 και 15)* · ★☆☆ · short-answer · `slides-lec23-organize-40kloc`

Υλοποιούμε ένα καινούριο σύστημα και περιμένουμε να έχει ~40 χιλιάδες γραμμές κώδικα. Τι κάνουμε; Ποιος είναι ο καλύτερος τρόπος να οργανώσουμε τον κώδικά μας;

Συγκρίνετε δύο λύσεις: (1) όλος ο κώδικας σε ένα αρχείο C και (2) οργάνωση του κώδικα σε πολλά αρχεία, και δώστε θετικά και αρνητικά για την καθεμία.

Υπόδειξη Σκεφτείτε τι συμβαίνει όταν ψάχνετε κάτι σε ένα αρχείο με δεκάδες χιλιάδες γραμμές και όταν αλλάζετε μία μόνο γραμμή και πρέπει να ξαναμεταγλωττίσετε. Για τη δεύτερη λύση, θυμηθείτε την ιδέα της αφαίρεσης: τι ρόλο παίζουν τα αρχεία `.c` και τα `.h`;

A23.3 · Αλλάζοντας κάτι const

A23.3 · *Διάλεξη 23, διαφάνειες 23-24* · ★★☆ · trace · `slides-lec23-const-violations`

Τι θα συμβεί όταν μεταγλωττίσουμε και τρέξουμε καθένα από τα δύο προγράμματα; Σε ποιο στάδιο (μεταγλώττιση ή εκτέλεση) εμφανίζεται το πρόβλημα, και γιατί;

`const1.c`:

```c
const char message[] = "Hello";
int main() {
  const int x = 42;
  const char const * msg_ptr = message;
  x++;
```

```
  return 0;
}
```

`const2.c`:

```
const char message[] = "Hello";
int main() {
  const int x = 42;
  char * bad = (char*)message;
  bad[1] = 'o';
  return 0;
}
```

Υπόδειξη Στο πρώτο πρόγραμμα, ο μεταγλωττιστής ξέρει ότι το `x` είναι `const`: τι θα κάνει με το `x++`; Στο δεύτερο, το cast κρύβει το `const` από τον μεταγλωττιστή, όμως ο πίνακας `message` εξακολουθεί να βρίσκεται σε μνήμη που προορίζεται μόνο για ανάγνωση.

A23.4 · Εξαρτήσεις ανάμεσα σε αρχεία

A23.4 · *Διαλέξεις 23–24, διαφάνεια 15 (διάλεξη 24: διαφάνεια 16)* · ★★☆ · short-answer · `slides-lec23-dependencies`

Ένα πρόγραμμα έχει τρία αρχεία:

```
/* err.c */
int print_err(char *msg) {
  ...
}
```

```
/* err.h */
int print_err(char *);
```

```
/* init.c */
#include "err.h"
...
print_err("hi");
```

Το `err.c` υλοποιεί τη διεπαφή `err.h` και το `init.c` τη χρησιμοποιεί.

1. Αν αλλάξω ένα αρχείο, τι επηρεάζεται; Απαντήστε για καθένα από τα τρία αρχεία.
2. Με τι μοιάζουν αυτές οι εξαρτήσεις;

Υπόδειξη Για κάθε αρχείο, αναρωτηθείτε ποια άλλα αρχεία «βλέπουν» το περιεχόμενό του κατά τη μεταγλώττιση. Σχεδιάστε τα αρχεία ως κόμβους και τις σχέσεις «υλοποιεί» και «χρησιμοποιεί» ως βέλη: ποια δομή δεδομένων από τις προηγούμενες διαλέξεις σχηματίζεται;

Εργαστήριο (A23.5)

A23.5 · Σπάστε το πρόγραμμά σας σε αρθρώματα

A23.5 · *Εργαστήριο 10, Άσκηση 5* · ★★☆ · tooling · `lab-lab10-more-modules`

Πάρτε το πρόγραμμα `more.c` της Άσκησης 1 (προβολή ενός αρχείου κειμένου ανά 20 γραμμές) και οργανώστε το σε πολλαπλά αρχεία:

1. Μεταφέρετε τη λογική ανάγνωσης και προβολής γραμμών σε ένα άρθρωμα `pager.c` με αντίστοιχο `pager.h`, αφήνοντας στη `main` μόνο τον χειρισμό των ορισμάτων γραμμής εντολής και το άνοιγμα του αρχείου.

2. Βάλτε include guard στο `pager.h`.

3. Κάντε τη σταθερά των 20 γραμμών `static` μέσα στο `pager.c`, ώστε να μην είναι ορατή από το `main.c`.

4. Μεταγλωττίστε πρώτα με το χέρι (`gcc -c` και σύνδεση) και μετά γράψτε ένα `Makefile` που κάνει το ίδιο. Επιβεβαιώστε ότι μετά από αλλαγή μόνο στο `main.c` το `make` ξαναμεταγλωττίζει μόνο αυτό.

5. Προσθέστε στο `Makefile` έναν στόχο `clean`.

Υπόδειξη Ακολουθήστε το παράδειγμα `collatz.h` / `collatz.c` / `main.c` του παραρτήματος του εργαστηρίου: στο `.h` μόνο πρωτότυπα μέσα σε `#ifndef`/`#define`/`#endif`, και `#include "pager.h"` με εισαγωγικά και στα δύο `.c`. Στο `Makefile`, κάθε `.o` πρέπει να εξαρτάται και από το `.h` που συμπεριλαμβάνει, και οι εντολές ξεκινούν με tab. Για τον έλεγχο του 4, αλλάξτε το `main.c` (ή κάντε `touch`) και ξανατρέξτε `make`.

Εργασίες (A23.6)

A23.6 · Η Newton-Raphson Ξαναχτυπά! (Bonus)

A23.6 · *Εργασία 3 (2023-24), Άσκηση 2 (Bonus) και 2.1 (Bonus)* · ★★★ · programming · `hw-2023-hw3-fractal`

Στην Εργασία 1 δοκιμάσαμε να γράψουμε ένα πρόγραμμα που έβρισκε αυτόματα ρίζες σε πολυώνυμα, αλλά δυστυχώς δεν είχαν όλα τα πολυώνυμα ρίζες. Για παράδειγμα, το πολυώνυμο $x^2 + 1$ δεν είχε ρίζες καθώς η διακρίνουσα είναι αρνητική. Με τους φανταστικούς αριθμούς όμως, όπου η φανταστική μονάδα i όταν υψωθεί στο τετράγωνο μας κάνει −1 ($i^2 = -1$), μπορούμε να πούμε ότι το $x = i$ είναι μια ρίζα του παραπάνω πολυωνύμου. Αντίστοιχα, μπορεί

η ρίζα ενός πολυωνύμου να είναι ένας συνδυασμός πραγματικών και φανταστικών, δηλαδή κάτι που ονομάζεται μιγαδικός αριθμός (complex number).

Θα δοκιμάσουμε να βρούμε λύσεις σε πολυώνυμα στο μιγαδικό επίπεδο. Η μέθοδος Newton παραμένει η ίδια, απλά αλλάζουμε τον ορισμό του τύπου που παίρνει η f: Δεδομένης μιας μιγαδικής συνάρτησης $f(z)$ και της παραγώγου της $f'(z)$, ξεκινώντας με ένα τυχαίο z_0 μία καλύτερη προσέγγιση μιας ρίζας του πολυωνύμου z_1 δίνεται από την σχέση:

$$z_1 = z_0 - \frac{f(z_0)}{f'(z_0)}$$

Αντίστοιχα, η γενική αναδρομική σχέση της μεθόδου του Νεύτωνα είναι:

$$z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}$$

όπου z_{n+1} είναι η προσεγγιστική τιμή μιας ρίζας της συνάρτησης $f(z)$ μετά από $n+1$ επαναλήψεις. Σε αντίθεση με την προηγούμενη άσκηση, *όλες οι πράξεις* που κάναμε με αριθμούς κινητής υποδιαστολής και μας δίνονταν ως βασικοί τελεστές (built in primitives/operators), τώρα θα πρέπει να οριστούν και να υλοποιηθούν για complex αριθμούς. Κάποιες από τις βασικές πράξεις για μιγαδικούς που μπορεί να χρειαστείτε περιλαμβάνουν (ελέγξτε την ορθότητά τους για την αποφυγή ορθογραφικών με άλλες πηγές):

$$(a + bi) + (c + di) = (a + c) + (b + d)i$$

$$(a + bi) \cdot (c + di) = (ac - bd) + (ad + bc)i$$

$$\frac{a + bi}{c + di} = \frac{ac + bd}{c^2 + d^2} + \frac{bc - ad}{c^2 + d^2}i$$

$$|a + bi| = \sqrt{a^2 + b^2}$$

Για κάθε αρχικό z_0, η μέθοδος Newton μπορεί να βρει μια ρίζα του πολυωνύμου - χωρίς να μπορούμε να προβλέψουμε ποια - και οι υπόλοιπες ρίζες μας διαφεύγουν. Μια συχνή λύση για να βρούμε περισσότερες ρίζες είναι να δοκιμάσουμε πολλά και διαφορετικά z_0 και να τρέξουμε πολλές φορές την μέθοδο Newton.

Για το ζητούμενο αυτής της άσκησης, καλείστε να γράψετε ένα πρόγραμμα που υπολογίζει αυτόματα τις ρίζες (τα z για τα οποία μηδενίζεται) *οποιουδήποτε* πολυωνύμου για ένα σύνολο από μιγαδικούς προκειμένου να μπορούμε να εντοπίσουμε περισσότερες από μία ρίζες.

Τεχνικές Προδιαγραφές

- Repository Name: `progintro/hw3-<YourTeamName>`
- Σε αντίθεση με όλες τις άλλες ασκήσεις μέχρι τώρα όπου όλος ο κώδικας του προγράμματός μας ήταν σε ένα αρχείο, σε αυτήν την εργασία, ο κώδικάς σας θα πρέπει να είναι χωρισμένος σε πολλά αρχεία και συγκεκριμένα:
    1. C Filepath: `fractal/src/complexlib.c` - το αρχείο στο οποίο θα βρίσκεται η υλοποίηση για όλες τις βασικές πράξεις με μιγαδικούς: πρόσθεση, αφαίρεση, διαίρεση κτλ.
    2. Header Filepath: `fractal/src/complexlib.h` - το αρχείο κεφαλίδων στο οποίο θα βρίσκονται οι δηλώσεις όλων των βασικών συναρτήσεων μιγαδικών που υλοποιήσατε.
    3. C Filepath: `fractal/src/fractal.c` - το αρχείο στο οποίο θα βρίσκεται η `main` σας και θα κάνει include το `"complexlib.h"` προκειμένου να χρησιμοποιήσει όλες τις συναρτήσεις μιγαδικών που θα χρειαστείτε.
- Το πρόγραμμά θα πρέπει να παίρνει ως πρώτο και κύριο όρισμα από την γραμμή εντολών το όνομα ενός αρχείου που περιέχει την είσοδο για το πρόγραμμα - στην μορφή `./fractal filename`. Για το περιεχόμενο του αρχείου δείτε παρακάτω. Αν το αρχείο δεν μπορεί να ανοιχτεί το πρόγραμμα πρέπει να επιστρέφει με κωδικό εξόδου (exit code) 1.
- Η βασική είσοδος για το πρόγραμμα θα είναι μέσω των περιεχομένων του αρχείου (το όνομα του αρχείου θα δίνεται ως πρώτο όρισμα όπως αναφέρθηκε παραπάνω). Το αρχείο θα έχει την ακόλουθη μορφή:

  ```
  POLYNOMIAL_DEGREE_N
  A0 A1 A2 ... AN
  MIN_REAL MIN_IMAG MAX_REAL MAX_IMAG STEP
  ```

  Η πρώτη γραμμή περιέχει τον βαθμό του πολυωνύμου. Η δεύτερη περιέχει τους πραγματικούς όρους του πολυωνύμου γραμμένους ως double. Η τρίτη και τελευταία γραμμή περιέχει δύο double όρους που καθορίζουν το παράθυρο μέσα στο οποίο θα αναζητήσουμε λύσεις με την μέθοδο Newton (δες παρακάτω στην ενδεικτική λύση). Για παράδειγμα, για ένα πολυώνυμο 5ου βαθμού $0.0 + 1.0x + 2.0x^2 + 3.0x^3 + 4.0x^4 + 5.0x^5$ και μέγεθος παραθύρου το $[-1.0 - i, 1.01 + 1.01i]$ με βήμα 0.1 η είσοδος θα είναι:

  ```
  5
  0.0 1.0 2.0 3.0 4.0 5.0
  -1.0 -1.0 1.01 1.01 0.1
  ```

- Όλες οι παράμετροι θα είναι σε δεκαδική μορφή τύπου `double` και επαρκούς ακρίβειας ώστε να χωράνε σε μεταβλητές double. Συνιστούμε να χρησιμοποιήσετε την συνάρτηση βιβλιοθήκης `fscanf` για να διαβάσετε τους αριθμούς από το αρχείο με τους κατάλληλους μετατροπείς.
- Η λύση σας πρέπει να κάνει χρήση ενός τύπου `complex` προκειμένου να μοντελοποιήσει τους μιγαδικούς αριθμούς. Περιμένουμε ότι ο ορισμός του τύπου θα έχει αυτήν την μορφή:

```c
typedef struct {
  double real;
  double imag;
} complex;
```

- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με τις ακόλουθες εντολές σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr):

  ```
  gcc -Ofast -Wall -Wextra -Werror -pedantic -c -o complexlib.o complexlib.c
  gcc -Ofast -Wall -Wextra -Werror -pedantic -c -o fractal.o fractal.c
  gcc -o fractal complexlib.o fractal.o -lm
  ```

- README Filepath: `fractal/README.md`
- **Ένα αρχείο που να περιέχει στοιχεία εισόδου και ένα εξόδου διαφορετικά από αυτά της άσκησης. Συγκεκριμένα προτείνουμε να βάλετε έναν συνδυασμό που θεωρείται ότι είναι δύσκολος να γίνει σωστός.**
  - input Filepath: `fractal/test/input`
  - output Filepath: `fractal/test/output`
- Όλες οι μιγαδικές λύσεις θα πρέπει να είναι δεκαδικοί με δύο ψηφία ακριβείας και πρόσημο.
- Συνθήκες τερματισμού: ο αλγόριθμος πρέπει να τερματίσει *όταν*:
  1. Ο αλγόριθμος έχει συγκλίνει και ισχύει: $|z_{n+1} - z_n| < 10^{-6}$. Σε αυτήν την περίπτωση τυπώνουμε το αποτέλεσμα z_{n+1} με ακρίβεια δύο δεκαδικών ψηφίων και πρόσημο για κάθε μέρος του μιγαδικού.
  2. Ο αλγόριθμος αποκλίνει (π.χ., διαίρεση με 0). Σε αυτήν την περίπτωση τυπώνουμε το αποτέλεσμα `nan`.
  3. Ο αλγόριθμος δεν τερματίζει μετά από 1000 επαναλήψεις. Σε αυτήν την περίπτωση τυπώνουμε το αποτέλεσμα `incomplete`.
- Πρέπει να ολοκληρώνει την εκτέλεση μέσα σε: 60 δευτερόλεπτα.
- Μέγιστος βαθμός πολυωνύμου: 10.
- Μέγιστος αριθμός στοιχείων του πίνακα που ζητάμε να εκτυπωθεί (για πόσα διαφορετικά z_0 να τρέξουμε την Newton Raphson): 10^6, π.χ., 1000 × 1000.
- **Δεν επιτρέπεται** η χρήση των built in complex αριθμών, π.χ., όπως ορίζονται στο `complex.h`, η χρήση του τύπου `_Complex` κτλ. Θέλουμε όλες οι πράξεις να οριστούν από εσάς και να βασίζονται στον τύπο `complex` που ορίσαμε παραπάνω.

Έστω ότι θέλουμε να βρούμε τις λύσεις της Newton-Raphson για το πολυώνυμο $1 + z - z^3 + z^4$ για τα ακόλουθα z_0: $\{-1 - i, -1 + 0i, -1 + i, 0 - i, 0, 0 + i, 1 - i, 1, 1 + i\}$, δηλαδή θέλουμε

να σαρώσουμε όλους τους μιγαδικούς από το $-1 - i$ μέχρι το $1 + i$ με βήμα 1. Παρακάτω παραθέτουμε την αλληλεπίδραση με μια ενδεικτική λύση:

```
$ cat input
4
1.0 1.0 0.0 -1.0 1.0
-1.0 -1.0 1.01 1.01 1.0
$ ./fractal input
-0.57-0.46i incomplete -0.57+0.46i
+1.07-0.86i incomplete +1.07+0.86i
+1.07-0.86i incomplete +1.07+0.86i
```

όπου βλέπουμε ότι για $z_0 = -1 - i$ (το πρώτο στοιχείο της πρώτης σειράς) η Newton συγκλίνει στην ρίζα $-0.57 - 0.46i$, ενώ για $z_0 = 0$ το αποτέλεσμα είναι incomplete (το 2ο στοιχείο της 2ης σειράς). Επομένως το αποτέλεσμα είναι ένας δισδιάστατος πίνακας με τις ρίζες στις οποίες αποτιμάται το πολυώνυμο για τα διάφορα z_0:

real/imag	-i	0i	1i
-1	-0.57-0.46i	incomplete	-0.57+0.46i
+0	+1.07-0.86i	incomplete	+1.07+0.86i
+1	+1.07-0.86i	incomplete	+1.07+0.86i

Το Σχήμα 3 της εκφώνησης δείχνει τα Newton fractals για το πολυώνυμο $1 + z - z^3 + z^4$ από το $-2 - 2i$ μέχρι το $+2 + 2i$, και το ίδιο fractal με βήμα 5× μικρότερο: κάθε ρίζα έχει δικό της χρώμα.

Αντίστοιχα μπορούμε να αυξήσουμε την πυκνότητα των αποτελεσμάτων που ελέγχουμε σε ένα παράθυρο, μειώνοντας το βήμα (τους μιγαδικούς στο $[-0.2 + 0.2i, 0.21 + 0.51i]$ με βήμα 0.05). Για παράδειγμα:

```
$ cat input
4
1.0 1.0 0.0 -1.0 1.0
-0.2 0.2 0.21 0.51 0.05
$ ./fractal input
-0.57+0.46i -0.57+0.46i -0.57+0.46i -0.57+0.46i -0.57+0.46i -0.57+0.46i
 ↪  -0.57+0.46i
-0.57+0.46i -0.57+0.46i -0.57+0.46i -0.57+0.46i -0.57+0.46i -0.57+0.46i
 ↪  -0.57+0.46i
-0.57+0.46i -0.57+0.46i -0.57+0.46i -0.57+0.46i -0.57+0.46i -0.57+0.46i
 ↪  -0.57+0.46i
+1.07-0.86i -0.57+0.46i -0.57+0.46i -0.57+0.46i +1.07+0.86i -0.57-0.46i
 ↪  -0.57-0.46i
```

```
-0.57-0.46i +1.07-0.86i -0.57-0.46i -0.57-0.46i -0.57-0.46i -0.57-0.46i
 ↪  +1.07+0.86i
-0.57+0.46i -0.57-0.46i -0.57-0.46i -0.57-0.46i -0.57-0.46i -0.57-0.46i
 ↪  -0.57-0.46i
-0.57-0.46i -0.57-0.46i -0.57-0.46i -0.57-0.46i -0.57-0.46i -0.57-0.46i
 ↪  -0.57-0.46i
-0.57-0.46i -0.57-0.46i -0.57-0.46i -0.57-0.46i -0.57-0.46i -0.57-0.46i
 ↪  -0.57-0.46i
-0.57-0.46i -0.57-0.46i -0.57-0.46i -0.57-0.46i -0.57-0.46i -0.57-0.46i
 ↪  -0.57-0.46i
```

Αν κοιτάξουμε προσεκτικά το παραπάνω βλέπουμε 4 διαφορετικές λύσεις, τις: $-0.57 - 0.46i$, $-0.57 + 0.46i$, $+1.07 + 0.86i$ και $+1.07 - 0.86i$ (πράγματι παρατηρούμε ότι $f(-0.57 - 0.46i) = -0.00 - 0.00i$) και επομένως καταφέραμε να βρούμε και τις τέσσερις ρίζες αυτού του πολυωνύμου! Η μόνη δυσκολία ίσως βρίσκεται στο να βρούμε τις περιοχές των μιγαδικών που πρέπει να σκανάρουμε.

Στο αρχείο `README.md` πρέπει να προσθέσετε οποιεσδήποτε παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.

2.1 Newton Fractals (Bonus 50 Μονάδες) Αν συνεχίσουμε να μειώνουμε το βήμα από την είσοδο του fractal παραπάνω οι πίνακες γίνονται μεγάλοι γρήγορα και αυξάνοντας την ανάλυση (resolution) των αποτελεσμάτων βλέπουμε πως μια μικρή αλλαγή της τάξης του 0.01 σε μια διάσταση μπορεί να κάνει την μέθοδο Newton να συγκλίνει σε διαφορετική ρίζα. Αν πειραματιστούμε αρκετά και έχουμε υπομονή μπορεί να παρατηρήσουμε ότι οι ρίζες προκύπτουν επαναλαμβανόμενα με μια κανονικότητα που θυμίζει fractal. Ένας τρόπος να οπτικοποιήσουμε αυτήν την κανονικότητα είναι να δώσουμε ένα χρώμα στην κάθε διαφορετική λύση στην οποία συγκλίνει η Newton-Raphson και να βάλουμε όλα αυτά τα χρώματα σε ένα δισδιάστατο επίπεδο που αναπαριστά τις αρχικές τιμές του z_0 στο πεδίο των μιγαδικών. Τα fractal που δημιουργούνται με αυτήν την μέθοδο λέγονται Newton Fractals. (Το Σχήμα 4 της εκφώνησης δείχνει ένα Newton fractal για το $z^4 - 1$, όπου το βάθος των χρωμάτων υποδεικνύει πόσο γρήγορα συγκλίνει, σε αριθμό επαναλήψεων, το κάθε σημείο σε μια ρίζα.)

Σε αυτήν την επέκταση της προηγούμενης άσκησης, ο στόχος είναι να προσθέσετε την δυνατότητα στο πρόγραμμά σας να δημιουργεί fractal visualizations.

Τεχνικές Προδιαγραφές (Επέκταση της λύσης fractal)

- Εκτός από το πρώτο και κύριο όρισμα (το αρχείο με την είσοδο για το πολυώνυμο), το πρόγραμμά σας θα πρέπει να μπορεί να αναγνωρίζει δύο ακόμα προαιρετικά ορίσματα `"-g output.bmp"`, δηλαδή το όρισμα `"-g"` που ορίζει ότι θέλουμε graphical output και το όρισμα που το ακολουθεί - π.χ., `"output.bmp"` - το οποίο είναι το αρχείο στο οποίο θέλουμε να σώσουμε την οπτικοποίηση του fractal μας.

- Filepath: `fractal/data/input`. Ένα αρχείο με πολυώνυμο που δοκιμάσατε να οπτικοποιήσετε.
- Filepath: `fractal/data/output.bmp`. Το αρχείο από Newton Fractal που οπτικοποίησε το πρόγραμμά σας για το παραπάνω input.

Υπόδειξη Ξεκινήστε από τη βιβλιοθήκη: ορίστε τον `complex` στο `complexlib.h` (με include guard) μαζί με τα πρωτότυπα, υλοποιήστε και ελέγξτε κάθε πράξη χωριστά, και μόνο μετά γράψτε τη Newton στο `fractal.c`. Υπολογίστε το πολυώνυμο και την παράγωγό του με τον κανόνα του Horner πάνω σε μιγαδικούς, και προσέξτε τη σειρά σάρωσης: κάθε γραμμή της εξόδου αντιστοιχεί σε ένα πραγματικό μέρος και κάθε στήλη σε ένα φανταστικό. Προσέξτε επίσης το «μείον μηδέν» στην εκτύπωση και τη συσσώρευση σφάλματος όταν προσθέτετε το βήμα επανειλημμένα· για το bonus, ξαναχρησιμοποιήστε ό,τι μάθατε για τη μορφή BMP στο fauxtoshop.

Κεφάλαιο 24: Προχωρημένα Θέματα

Ζέσταμα: από τις διαφάνειες (A24.1)

A24.1 · Ο τυχερός αριθμός σε δυαδικό

A24.1 · *Διάλεξη 24: Προχωρημένα Θέματα, διαφάνεια 32* · ★☆☆ · trace · `slides-lec24-binary-literal`

Τι θα τυπώσει αυτό το πρόγραμμα;

```c
#include <stdio.h>

int main() {
  int i = 0b101010;
  printf("My lucky number is: %d\n", i);
  return 0;
}
```

Υπόδειξη Το πρόθεμα `0b` (C2X) σημαίνει ότι τα ψηφία που ακολουθούν είναι δυαδικά. Γράψτε τη θέση κάθε bit από δεξιά (θέση 0) και αθροίστε τις δυνάμεις του 2 για όσα bit είναι 1.

Κεφάλαιο 25: Επίλυση Προβλημάτων #3

Εργαστήριο (A25.1)

A25.1 · Καθαρή διαχείριση μνήμης

A25.1 · *Εργαστήριο 9, Άσκηση 5* · ★★☆ · programming · `lab-lab09-grades-tree`

Η άσκηση βασίζεται στο παράρτημα «Αποσφαλμάτωση προγραμμάτων (Πράξη 5η)» του εργαστηρίου, που δείχνει πώς τρέχουμε ένα πρόγραμμα κάτω από το valgrind:

```
gcc -g3 -o grades grades.c
valgrind --leak-check=full ./grades
```

και πώς διαβάζουμε τις κατηγορίες διαρροών (`definitely lost`, `indirectly lost`, `possibly lost`, `still reachable`). Η μόνη έξοδος που θεωρούμε καθαρή είναι:

```
==2578== HEAP SUMMARY:
==2578==     in use at exit: 0 bytes in 0 blocks
==2578==   total heap usage: 5 allocs, 5 frees, 4,160 bytes allocated
==2578==
==2578== All heap blocks were freed -- no leaks are possible
==2578==
==2578== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

5.1 Καθαρίστε τις δομές σας.

Επιστρέψτε στις Ασκήσεις 3 και 4 αυτού του εργαστηρίου. Καμία από τις δύο δεν αποδεσμεύει τη μνήμη που δεσμεύει.

5.1.1 Γράψτε τη συνάρτηση `void free_list(Listptr ptr)` και καλέστε την από τη `main` του `grades.c`. Επιβεβαιώστε με valgrind ότι βλέπετε το μήνυμα `All heap blocks were freed`.

5.1.2 Γράψτε την **αναδρομική** συνάρτηση `void free_tree(Treeptr p)` για το `tree.c`. Προσοχή στη σειρά: πρέπει να αποδεσμεύσετε πρώτα τα υποδένδρα και μετά τον κόμβο. Τι θα δείξει το valgrind αν το κάνετε ανάποδα;

5.2 Ξαναδείτε ένα παλιό σφάλμα.

Στο εργαστήριο 8 αποσφαλματώσαμε το `wages.c` με τη βοήθεια ενός debugger. Τρέξτε τώρα την **αρχική, λανθασμένη** έκδοση του με valgrind. Ποιο ακριβώς μήνυμα σας δίνει, και σε ποια γραμμή; Σας οδηγεί πιο γρήγορα στο πρόβλημα από ό,τι ο `gdb`;

Υπόδειξη Στη λίστα, κρατήστε τον δείκτη στον επόμενο κόμβο πριν αποδεσμεύσετε τον τρέχοντα, αλλιώς διαβάζετε μνήμη που μόλις ελευθερώσατε. Στο δένδρο, η σειρά είναι μια post-order διάσχιση: αν ελευθερώσετε πρώτα τον κόμβο, τα `left` και `right` του διαβάζονται από αποδεσμευμένη μνήμη.

Εργασίες (A25.2–A25.4)

A25.2 · DNA Matching

A25.2 · *Εργασία 2 (2023-24), Άσκηση 2* · ★★★ · programming · `hw-2023-hw2-dna`

Ένα εργαστήριο βιοπληροφορικής ψάχνει λύσεις για να βρίσκει *αποδοτικά* κοινές αλληλουχίες DNA ανάμεσα σε διαφορετικά άτομα ή ακόμα και οργανισμούς, γνωστό και ως DNA matching.

Το DNA συνήθως το αναπαριστούμε με την μορφή μιας αλυσίδας (συμβολοσειράς) από τέσσερα διαφορετικά στοιχεία που λέγονται *βάσεις*: A (Αδενίνη), G (Γουανίνη), T (Θυμίνη) και C (Κυτοσίνη) τα οποία μπορούν να συνδυαστούν με οποιαδήποτε σειρά. Προκειμένου να κάνουμε DNA matching θέλουμε να δούμε αν υπάρχουν κοινά τμήματα σε αυτές τις αλυσίδες DNA και να δούμε πόσο μεγάλες είναι. Το Σχήμα 4 της εκφώνησης δείχνει ένα παράδειγμα δύο αλυσίδων με κοινό τμήμα: οι αλυσίδες `CATTAGATATAGACG` και `CTATAGATATAGGGC` έχουν κοινό το τμήμα `TAGATATAG`.

Για το ζητούμενο αυτής της άσκησης, καλείστε να γράψετε ένα πρόγραμμα που διαβάζει δύο αλυσίδες DNA, βρίσκει την μέγιστη κοινή αλυσίδα ανάμεσα στις δύο και την τυπώνει.

Τεχνικές Προδιαγραφές

- Repository Name: `progintro/hw2-<YourUsername>`
- C Filepath: `dna/src/dna.c`
- Το πρόγραμμά σας θα πρέπει να παίρνει δύο ορίσματα από την γραμμή εντολών. Και τα δύο θα είναι ονόματα αρχείων που περιέχουν τις εισόδους για το πρόγραμμα, δηλαδή τις δύο αλυσίδες DNA - στην μορφή `./dna filename1 filename2`. Για το περιεχόμενο των αρχείων δείτε παρακάτω. Αν δεν περαστούν δύο ορίσματα στην γραμμή εντολών ή οποιοδήποτε από τα αρχεία δεν μπορεί να ανοιχτεί το πρόγραμμα πρέπει να επιστρέφει με κωδικό εξόδου (exit code) 1.
- Η βασική είσοδος για το πρόγραμμα θα είναι μέσω των περιεχομένων των αρχείων. Τα αρχεία θα περιέχουν τις αλυσίδες DNA οι οποίες αποτελούνται από τους κεφαλαίους χαρακτήρες A, G, T και C. Όλοι οι χαρακτήρες θα είναι σε μορφή ASCII, και το αρχείο θα διαβάζεται και ως κείμενο. Οποιοσδήποτε χαρακτήρας εκτός από τους A, G, T, C πρέπει να αγνοηθεί καθώς θεωρείται σφάλμα που δημιουργήθηκε κατά το sequencing της αλυσίδας.
- Η έξοδος του προγράμματος θα πρέπει να είναι η μέγιστη κοινή αλυσίδα των δύο αλυσίδων που δώσαμε ως είσοδο, ακολουθούμενη από μια καινούρια γραμμή. Όλα τα στοιχεία της αλυσίδας εξόδου θα πρέπει να είναι βάσεις DNA - δεν επιτρέπονται άλλοι χαρακτήρες. Εάν υπάρχουν πάνω από μία μέγιστη κοινή αλυσίδα, αρκεί να επιστραφεί μία από αυτές.
- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας

πρέπει να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr): `gcc -Ofast -m32 -Wall -Wextra -Werror -pedantic -o dna dna.c -lm`
- README Filepath: `dna/README.md`
- **Ένα αρχείο που να περιέχει στοιχεία εισόδου και ένα εξόδου διαφορετικά από αυτά της άσκησης. Συγκεκριμένα προτείνουμε να βάλετε έναν συνδυασμό που θεωρείται ότι είναι δύσκολος να γίνει σωστός**.
  - input1 Filepath: `dna/test/input1.dna`
  - input2 Filepath: `dna/test/input2.dna`
  - output Filepath: `dna/test/output.dna`
- Πρέπει να ολοκληρώνει την εκτέλεση μέσα σε: 90 δευτερόλεπτα.
- Το μέγιστο μέγεθος επιτρεπτής αλυσίδας: 100.000 βάσεις.

Παρακάτω παραθέτουμε την αλληλεπίδραση με μια ενδεικτική λύση με μερικά δείγματα DNA που κατεβάσαμε:

```
$ gcc -Ofast -Wall -Wextra -Werror -pedantic -o dna dna.c -lm
$ wc -c *.dna
 62487 alien.dna
  8193 carsonella-ruddii.dna
 47811 claviceps-purpurea.dna
 21992 escherichia-coli.dna
    16 sample1.dna
    16 sample2.dna
 99492 theobroma-cacao.dna
240007 total
$ $./dna
Error: arguments missing. Usage: ./dna dnafile1 dnafile2
$ echo $?
1
$ ./dna3 foo.bar bar.zonk
Error: cannot open file foo.bar
$ echo $?
1
$ cat sample1.dna
CATTAGATATAGACG
$ cat sample2.dna
CTATAGATATAGGGC
$ ./dna sample1.dna sample2.dna
TAGATATAG
$ echo -e "CTATAGAT\nHello WORLDATAGGG" > split.dna
$ ./dna split.dna split.dna
CTATAGATATAGGG
$ ./dna carsonella-ruddii.dna sample1.dna
```

```
ATATAGACG
$ ./dna escherichia-coli.dna carsonella-ruddii.dna
AATTAAAATTTTATT
$ ./dna claviceps-purpurea.dna carsonella-ruddii.dna
GTTTTTTTTTTCT
$ time ./dna theobroma-cacao.dna escherichia-coli.dna
GGTTTGCTTTTATG

real 0m4.550s
user 0m4.549s
sys 0m0.001s
$ time ./dna theobroma-cacao.dna alien.dna
AAAAAAAAAAAAAAAAACC

real 0m2.828s
user 0m2.827s
sys 0m0.001s
$ time ./dna theobroma-cacao.dna theobroma-cacao.dna > shared.dna

real 0m21.817s
user 0m21.815s
sys 0m0.001s
$ wc -c shared.dna
98165 shared.dna
$ md5sum shared.dna
5ca8c9e6fd76d46221d3beadd610b165  shared.dna
$ time ./dna alien.dna alien.dna > shared.dna

real 0m1.917s
user 0m1.915s
sys 0m0.001s
$ wc -c shared.dna
62488 shared.dna
$ md5sum shared.dna
d4544ab6971c6a54bb375159adc5852b  shared.dna
```

Τα παραπάνω dna αρχεία είναι στο: https://github.com/progintro/data/tree/main/dna άλλα μπορείτε να δοκιμάσετε και άλλα παραδείγματα δικά σας. Στο αρχείο `README.md` πρέπει να προσθέσετε οποιεσδήποτε παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.

Υπόδειξη Πρόκειται για το πρόβλημα της μέγιστης κοινής *υποσυμβολοσειράς* (διαδοχικοί χαρακτήρες, όχι υποακολουθία). Φιλτράρετε πρώτα κάθε αρχείο σε μια καθαρή συμβολοσειρά μόνο με A, G, T, C, σε δυναμικά δεσμευμένη μνήμη. Μια σχέση του τύπου «μήκος κοινού τμήματος που τελειώνει στη θέση i της πρώτης και j της δεύτερης» δίνει λύση δυναμικού προγραμματισμού· ένας πλήρης πίνακας 100.000 × 100.000 όμως δεν χωράει στη μνήμη, οπότε σκεφτείτε ποιες γραμμές του χρειάζεστε πραγματικά κάθε στιγμή.

Δείτε επίσης

- Υποδειγματική υποβολή φοιτητή (περιέχει λύση: δείτε την αφού λύσετε την άσκηση)

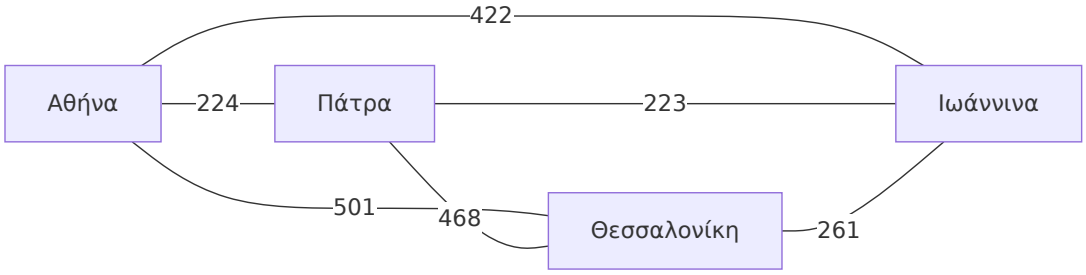
A25.3 · Το Καλύτερο GPS (jabbamaps)

A25.3 · *Εργασία 2 (2024-25), Άσκηση 2* · ★★★ · programming · `hw-2024-hw2-jabbamaps`

Το Καλύτερο GPS (jabbamaps - 50 Μονάδες)

Πολλές εφαρμογές (όπως το Google Maps) σου επιτρέπουν να προσθέσεις πολλές στάσεις στη διαδρομή σου, αλλά δεν σου προτείνουν να τις αναδιατάξεις ακόμη και αν αυτό σου γλύτωνε χρόνο. Έχει σημασία με ποια σειρά θα καλύψουμε αυτές τις στάσεις; Αυτό θα είναι το πρόβλημα με το οποίο θα ασχοληθούμε σε αυτήν την άσκηση.

Ας δούμε μια απλοποιημένη εκδοχή του προβλήματός μας για 4 στάσεις. Έστω ότι θέλουμε να κάνουμε τον γύρο τις Ελλάδας και συγκεκριμένα να καλύψουμε τέσσερις πόλεις: Αθήνα, Θεσσαλονίκη, Γιάννενα και Πάτρα. Από κάθε πόλη μπορούμε να κινηθούμε προς οποιαδήποτε άλλη, αλλά κάθε κίνησή μας έχει ένα κόστος (έστω χιλιομετρικό). Για παράδειγμα, για να πάμε από την Αθήνα στην Θεσσαλονίκη έχουμε κόστος 501, ενώ για να πάμε στην Πάτρα έχουμε κόστος 224. Θεωρούμε ότι η κατεύθυνση δεν έχει σημασία και το χιλιομετρικό κόστος είναι το ίδιο ανεξαρτήτως κατεύθυνσης.



Σχήμα: Από κάθε πόλη μπορείς να αποφασίσεις να κινηθείς σε οποιαδήποτε άλλη πόλη με κάποιο χιλιομετρικό κόστος. Ποια είναι η πιο αποδοτική (από απόψεως κόστους) σειρά με την οποία μπορείς να τις επισκεφτείς;

Έστω ότι ξεκινάμε από την Αθήνα (η πόλη από την οποία ξεκινάμε δεν έχει σημασία, καθώς

πρέπει να επισκεφτούμε όλες τις πόλεις), έχουμε 3 διαφορετικές επιλογές για την πόλη που θα επισκεφτούμε στην συνέχεια. Προσέξτε ότι οποιαδήποτε πόλη επιλέξουμε έχει συνέπειες και για τις επόμενες επιλογές μας. Ποιο είναι το μονοπάτι με το ελάχιστο κόστος; Αν επιλέξω να κάνω το Αθήνα ⟶ Θεσσαλονίκη ⟶ Ιωάννινα ⟶ Πάτρα θα χρειαστώ 501 + 261 + 223 = 985 χιλιόμετρα. Αντίθετα, αν πάω Αθήνα ⟶ Πάτρα ⟶ Ιωάννινα ⟶ Θεσσαλονίκη χρειάζομαι 224 + 223 + 261 = 708 χιλιόμετρα. Με μια αλλαγή στην σειρά επίσκεψης γλύτωσα 277 χιλιόμετρα! Είναι όμως αυτή η βέλτιστη επιλογή; Πόσες δυνατές διαφορετικές διατάξεις υπάρχουν στην σειρά με την οποία μπορώ να επισκεφτώ τις πόλεις;

Το πρόβλημα στο οποίο πρέπει να αποφασίσουμε με ποια σειρά θα επισκεφτούμε μια σειρά πόλεων λέγεται *Travelling Salesman Problem*, ελληνιστί το πρόβλημα του πλανόδιου πωλητή. Φτάσαμε επομένως στο ζητούμενο αυτής της άσκησης: να γράψετε ένα πρόγραμμα το οποίο θα διαβάζει έναν χάρτη με τις αποστάσεις όλων των ζευγών πόλεων που θέλουμε να επισκεφτούμε και θα μας υπολογίζει το μονοπάτι ελαχίστου κόστους, εύκολα, γρήγορα και πάνω απ'όλα τζάμπα!

Τεχνικές Προδιαγραφές

- Repository Name: progintro/hw2-<YourUsername>
- C Filepath: jabbamaps/src/jabbamaps.c
- Το πρόγραμμά θα πρέπει να παίρνει ακριβώς ένα όρισμα, το όνομα του αρχείου που περιέχει τα δεδομένα του χάρτη. Αν το πρόγραμμα εκτελεστεί με ορίσματα που δεν ακολουθούν τις παραπάνω προδιαγραφές, πρέπει να εκτυπώσει αντίστοιχο μήνυμα όπως στα παρακάτω παραδείγματα και να επιστρέφει με κωδικό εξόδου (exit code) 1.
- Το αρχείο με τα δεδομένα του χάρτη θα έχει σε κάθε γραμμή του ένα ζεύγος πόλεων και στην συνέχεια την απόστασή τους. Συγκεκριμένα, κάθε γραμμή θα είναι της μορφής `city1-city2: distance`, όπου city1 και city2 είναι το ζεύγος των πόλεων και distance είναι η απόστασή τους. Τα ονόματα των πόλεων δεν θα έχουν τους χαρακτήρες '-' ή ':' και η απόστασή τους θα είναι πάντα ένας ακέραιος αριθμός.
- Η πρώτη πόλη που θα διαβάσουμε στον χάρτη θέλουμε να είναι η πόλη από την οποία θα ξεκινήσουμε την διαδρομή μας. Το συνολικό κόστος δεν θα πρέπει να περιλαμβάνει το κόστος επιστροφής στην αρχική μας πόλη.
- Οι αποστάσεις των πόλεων δεν θα υπερβαίνουν το 2^{31}.
- Οι πόλεις που θα πρέπει να επισκεφτούμε δεν θα είναι πάνω από 64.
- Το μόνοπάτι σας πρέπει να επισκεφτεί *κάθε* πόλη *ακριβώς* μία φορά.
- Το αρχείο C που θα υποβληθεί πρέπει να μεταγλωττίζεται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το αρχείο σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με την ακόλουθη εντολή σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr): `gcc -m32 -Ofast -Wall -Wextra -Werror -pedantic -o jabbamaps jabbamaps.c`
- README Filepath: jabbamaps/README.md
- Πρέπει να ολοκληρώνει την εκτέλεση μέσα σε: 30 δευτερόλεπτα.

Ενδεικτικές εκτελέσεις ακολουθούν σε κάποιους από τους χάρτες που υπάρχουν στο https://gi

thub.com/progintro/data:

```
$ cat map4.txt
Athens-Thessaloniki: 501
Athens-Ioannina: 422
Athens-Patras: 224
Patras-Thessaloniki: 468
Patras-Ioannina: 223
Thessaloniki-Ioannina: 261
$ ./jabbamaps map4.txt
We will visit the cities in the following order:
Athens -(224)-> Patras -(223)-> Ioannina -(261)-> Thessaloniki
Total cost: 708
$ ./jabbamaps map7.txt
We will visit the cities in the following order:
Athens -(211)-> Amfissa -(122)-> Patras -(223)-> Ioannina -(128)->
  Trikala -(123)-> Volos -(211)-> Thessaloniki
Total cost: 1018
```

Ιδανικά, επιθυμούμε το πρόγραμμά μας να δουλεύει και για μεγαλύτερους χάρτες, όσο ασυνήθιστα και να είναι τα ονόματα των πόλεων ή οι αποστάσεις μεταξύ τους. Για παράδειγμα:

```
$ ./jabbamaps tatooine.txt
We will visit the cities in the following order:
Republic City -(65)-> Aldera -(124)-> Anchorhead -(44)-> Lessu -(45)->
  Mos Pelgo -(32)-> Canto Bight -(65)-> Mos Espa -(80)-> Coronet City -(53)->
  Hanna City -(50)-> Sern Prime -(62)-> NiJedha -(20)-> Kachirho -(66)->
  Tipoca City -(141)-> Sundari -(51)-> Galactic City -(29)->
  Capital City (Lothal City) -(157)-> Mos Eisley -(317)-> Stalgasin Hive -(26)->
  Coral City -(25)-> Otoh Gunga -(13)-> Theed -(18)-> Cloud City -(136)-> Eriadu
↪  City
Total cost: 1619
```

Όσο περισσότερες οι πόλεις, τόσοι περισσότεροι οι συνδυασμοί που πρέπει να εξετάσουμε και το πρόβλημα πλέον δεν είναι εύκολο να επιβεβαιωθεί ”με το μάτι”. Είναι η λύση που βρήκαμε η βέλτιστη ή μήπως υπάρχει καλύτερη;

Ως συνήθως, υπενθυμίζουμε πως στο αρχείο README.md πρέπει να προσθέσετε οποιεσδήποτε παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.

Υπόδειξη Πρώτα μετατρέψτε το αρχείο σε πίνακα αποστάσεων: κάθε νέο όνομα πόλης παίρνει έναν αριθμό (προσοχή, τα ονόματα έχουν κενά και παρενθέσεις), και η απόσταση μπαίνει στο `dist[i][j]` και στο `dist[j][i]`. Η δοκιμή όλων των $(n-1)!$ διατάξεων σβήνει ήδη γύρω στις 12 πόλεις· για περισσότερες σκεφτείτε δυναμικό προγραμματισμό πάνω σε

υποσύνολα (bitmask, αλγόριθμος Held-Karp), που όμως για 64 πόλεις δεν χωρά ούτε σε μνήμη ούτε σε χρόνο, οπότε εκεί χρειάζεστε ευριστικές μεθόδους (nearest neighbour, 2-opt). Με `-m32`, κρατήστε το συνολικό κόστος σε τύπο 64 bits.

A25.4 · Ανελκυστήρες για Ανυπόμονους και Ανυπόμονες (elevate)

A25.4 · *Εργασία 2 (2025-26), Άσκηση 1* · ★★★ · programming · `hw-2025-hw2-elevate`

Έχετε βρεθεί ποτέ να περιμένετε τον ανελκυστήρα να έρθει στον όροφό σας και να παίρνει υπερβολικά πολύ χρόνο; Σε αυτήν την άσκηση θα γράψουμε ένα πρόγραμμα-οδηγό (driver) που βελτιστοποιεί τις αποφάσεις ενός ανελκυστήρα, ώστε οι επιβάτες να περπατούν όσο το δυνατόν λιγότερους ορόφους.

Μοντελοποίηση του προβλήματος. Έστω ότι ο ανελκυστήρας μας λειτουργεί σε ένα κτήριο με άπειρους ορόφους (όροφοι 1, 2, 3, ...). Στο ισόγειο (όροφος 0) βρίσκονται $numPeople$ άνθρωποι, καθένας από τους οποίους θέλει να πάει στον όροφο $dests_i$ ($1 \leq i \leq numPeople$). Για κάποιους (τεχνικούς και άλλους λόγους—ο κατασκευαστής ήταν ελαφρώς σφιχτοχέρης) ο ανελκυστήρας θα ξεκινήσει από το ισόγειο και θα κάνει *το πολύ* $numStops$ στάσεις. Κάθε επιβάτης θα βγει στον όροφο (κάποια από τις στάσεις του ανελκυστήρα) που είναι πιο κοντά στον όροφο που είναι ο προορισμός του και θα πάει από τις σκάλες στον προορισμό του. Κάθε μετάβαση επιβάτη από τις σκάλες από όροφο σε γειτονικό του όροφο (είτε ανεβαίνοντας, είτε κατεβαίνοντας) έχει κόστος 1. Ποια είναι η βέλτιστη λύση (συνολικό ελάχιστο κόστος), δηλαδή ποιες στάσεις πρέπει να κάνει ο ανελκυστήρας, ώστε οι επιβάτες του να κάνουν τις λιγότερες δυνατές μετακινήσεις με τα πόδια; Μεταξύ λύσεων με το ίδιο κόστος, προτιμώνται εκείνες που ο ανελκυστήρας σταματά σε χαμηλότερους ορόφους. Να σημειωθεί ότι δεν είναι απαραίτητο όλοι οι επιβάτες να μπουν στον ανελκυστήρα. Κάποιοι μπορεί να πάνε με τα πόδια από το ισόγειο στον προορισμό τους (επειδή αυτός θα είναι πιο κοντά στο ισόγειο απ' ότι στην πρώτη στάση του ανελκυστήρα). Επίσης, υπάρχει ενδεχόμενο να υπάρχει λύση στο πρόβλημα με λιγότερες από $numStops$ στάσεις, επειδή ο ανελκυστήρας θα σταματήσει σε όλους τους προορισμούς των επιβατών του και το κόστος θα ισούται με 0.

Ας δούμε λοιπόν ένα παράδειγμα. Έστω $numPeople = 5$, $numStops = 2$ και $dests = [11\ 2\ 7\ 13\ 7]$. Αν ο ανελκυστήρας κάνει τις δύο στάσεις του στους ορόφους $stops = [5\ 12]$, τότε ο πρώτος επιβάτης με προορισμό τον όροφο 11 θα βγει στον όροφο 12 (πλησιέστερη στάση στον προορισμό του) και θα κατέβει με τις σκάλες 1 όροφο (κόστος 1). Ο δεύτερος επιβάτης με προορισμό τον όροφο 2 δεν θα μπει στον ανελκυστήρα και θα ανέβει δύο ορόφους με τις σκάλες (κόστος 2). Για τον τρίτο επιβάτη, έχουμε κόστος ίσο με 2 (προορισμός ο όροφος 7 και στάση ο όροφος 5). Με όμοιο τρόπο, βρίσκουμε ότι το κόστος για τον τέταρτο επιβάτη είναι 1 και για τον πέμπτο είναι 2. Το συνολικό κόστος ισούται με $1 + 2 + 2 + 1 + 2 = 8$. Όμως, η λύση $stops = [7\ 11]$ έχει κόστος ίσο με 4 (γιατί;) και μπορεί να αποδειχθεί ότι είναι η βέλτιστη λύση.

Σκέψεις για την επίλυση του προβλήματος. Δεν υπάρχει περίπτωση η βέλτιστη λύση να περιέχει στάση σε όροφο μεγαλύτερο από τον υψηλότερο προορισμό των επιβατών (αν υπήρχε

τέτοια στάση, θα μπορούσαμε να την μεταφέρουμε σε αυτόν τον όροφο και η νέα λύση σίγουρα θα έχει μικρότερο κόστος). Για τη συνέχεια, ας συμβολίσουμε τον υψηλότερο προορισμό των επιβατών ως $numFloors$.

Η διαχείριση του ενδεχομένου να αρκούν λιγότερες στάσεις από $numStops$, οπότε έχουμε λύση ελαχίστου κόστους ίσου με 0, μπορεί να γίνει με απλό τρόπο. Σκεφτείτε την περίπτωση $numPeople = 5$, $numStops = 3$ και $dests = [6\ 6\ 6\ 6\ 6]$. Αρκεί ο ανελκυστήρας να κάνει μόνο μία στάση, στον όροφο 6 (κόστος 0). Την λύση αυτή θα μπορούσαμε να την αναπαραστήσουμε σαν $stops = [0\ 0\ 6]$. Δηλαδή, θεωρούμε ότι υπάρχουν και επιπλέον, πρακτικά ανύπαρκτες, στάσεις στο ισόγειο.

Αν a και b είναι δύο διαδοχικές στάσεις του ανελκυστήρα, ας θεωρήσουμε ότι με $fw(a, b)$ (τα αρχικά για τις λέξεις Floors Walked) συμβολίζουμε το συνολικό πλήθος των ορόφων που θα χρειαστεί να περπατήσουν όλοι οι επιβάτες που έχουν προορισμό όροφο d πιο ψηλά από τον όροφο a ($a < d$) και το πολύ μέχρι τον όροφο b ($d \leq b$) για να πάνε στον όροφο d από τις σκάλες, βγαίνοντας από τον ανελκυστήρα στον όροφο a ή στον όροφο b, ανάλογα με το ποιος είναι κοντινότερος στον d. Το b μπορεί να ισούται με ∞, στην περίπτωση που ο όροφος a είναι ο τελευταίος που θα σταματήσει ο ανελκυστήρας. Επίσης, το a μπορεί να ισούται με 0, στην περίπτωση που το b είναι η πρώτη στάση του ανελκυστήρα.

Έστω $M_{i,j}$ το ελάχιστο κόστος για να εξυπηρετηθούν όλοι οι επιβάτες με i ακριβώς στάσεις του ανελκυστήρα, η υψηλότερη από τις οποίες είναι στον όροφο j. Εκφράζεται από την ακόλουθη αναδρομική σχέση:

$$
\begin{aligned}
M_{0,j} &= fw(0, \infty), \quad 0 \leq j \leq numFloors \\
M_{i,j} &= \min_{k=0}^{j} \{M_{i-1,k} - fw(k, \infty) + fw(k, j) + fw(j, \infty)\}, \\
&\qquad \text{where } 1 \leq i \leq numStops,\ 0 \leq j \leq numFloors
\end{aligned}
$$

Αν ο ανελκυστήρας δεν κινηθεί καθόλου ($i = 0$ στάσεις), όλοι οι επιβάτες θα πάνε στους προορισμούς τους από τις σκάλες. Όταν προστεθεί μία επιπλέον στάση στον όροφο j μετά από κάποια στον όροφο k, αλλάζει μόνο το κόστος των επιβατών με προορισμούς ψηλότερα από τον όροφο k: από το $M_{i-1,k}$ αφαιρούμε το κόστος τους με τις σκάλες από τον k και το αντικαθιστούμε με το κόστος όσων έχουν προορισμό μέχρι τον j (ανάμεσα σε k και j) συν το κόστος όσων έχουν προορισμό πάνω από τον j. Το ελάχιστο από αυτά τα κόστη, για όλα τα πιθανά k, είναι το $M_{i,j}$. Τελικά, το ελάχιστο κόστος για το πλήρες πρόβλημα ισούται με

$$
MinCost = \min_{j=0}^{numFloors} \{M_{numStops,j}\}
$$

και η τελευταία στάση του ανελκυστήρα είναι εκείνο το j για το οποίο έχουμε το ελάχιστο στον προηγούμενο τύπο.

Στην εργασία αυτή λοιπόν, καλείστε να υλοποιήσετε ένα πρόγραμμα σε γλώσσα C το οποίο να βρίσκει αποδοτικά την βέλτιστη από πλευράς κόστους λειτουργίας του ανελκυστήρα. Καθώς δεν γνωρίζουμε εξ αρχής ποια λύση θα έχει καλύτερη απόδοση, θέλουμε να δοκιμάσετε τέσσερις (4) εναλλακτικές μεθόδους έτσι ώστε να μπορέσουμε να συγκρίνουμε τις επιδόσεις τους (benchmarking) και να επιλέξουμε την καλύτερη για χρήση σε συστήματα παραγωγής.

Τεχνικές Προδιαγραφές

- Repository Name: `progintro/hw2-<YourUsername>`
- README Filepath: `elevate/README.md`
- Αρχεία C και headerfiles (Filepaths):
  - `elevate/src/elevate.c` : το αρχείο που θα περιέχει την `main` σας.
  - `elevate/src/recurse.c` : το αρχείο με την αναδρομική υλοποίηση
  - `elevate/src/brute.c` : το αρχείο με την υλοποίηση ωμής βίας
  - `elevate/src/memoize.c` : το αρχείο με την υλοποίηση που χρησιμοποιεί memoization
  - `elevate/src/dp.c` : το αρχείο με την υλοποίηση που χρησιμοποιεί δυναμικό προγραμματισμό
  - `elevate/src/elevate.h` : το αρχείο κεφαλίδας με τους ορισμούς προτύπων συναρτήσεων όπου απαιτείται.
- Τα αρχεία C που θα υποβληθούν πρέπει να μεταγλωττίζονται χωρίς ειδοποιήσεις για λάθη και με κωδικό επιστροφής (exit code) που να είναι 0. Συγκεκριμένα, το πρόγραμμά σας **πρέπει** να μπορεί να μεταγλωττιστεί επιτυχώς με τις ακόλουθες εντολές σε ένα από τα μηχανήματα του εργαστηρίου (linuxXY.di.uoa.gr):

```
gcc -Os -c -Wall -Wextra -Werror -pedantic recurse.c
gcc -Os -c -Wall -Wextra -Werror -pedantic brute.c
gcc -Os -c -Wall -Wextra -Werror -pedantic memoize.c
gcc -Os -c -Wall -Wextra -Werror -pedantic dp.c
gcc -Os -c -Wall -Wextra -Werror -pedantic elevate.c
gcc -Os -o elevate recurse.o brute.o memoize.o dp.o elevate.o
```

- Το πρόγραμμά σας θα πρέπει να παίρνει δύο ορίσματα από την γραμμή εντολών:
  1. Το πρώτο υποχρεωτικό όρισμα θα είναι το όνομα του αρχείου εισόδου. Το αρχείο, θα περιέχει τα δεδομένα του προβλήματος, δηλαδή τον αριθμό των ατόμων (numPeople), τον αριθμό των στάσεων (numStops) καθώς και τις στάσεις που θέλουν να σταματήσουν τα άτομα ($dests_i$) όπου $1 \leq i \leq numPeople$.
  2. Το δεύτερο όρισμα θα έχει την μορφή `--mode=MODE` όπου ο χρήστης θα επιλέγει ποια εναλλακτική υλοποίηση (MODE) επιθυμεί ανάμεσα στις ακόλουθες επιλογές: (1) `recurse`, (2) `brute`, (3) `memoize`, (4) `dp`. Αφού ολοκληρώσετε την υλοποίησή σας, μπορείτε να μετατρέψετε αυτό το όρισμα σε προαιρετικό έτσι ώστε να χρησιμοποιεί

την βέλτιστη λύση όταν ο χρήστης δεν το προσδιορίσει (default behavior).

- Το πρόγραμμά σας πρέπει να χρησιμοποιεί την πρότυπη έξοδο για να τυπώσει τα αποτελέσματά του: την τελευταία στάση που θα κάνει ο ανελκυστήρας καθώς και το ελάχιστο κόστος σε αυτήν την περίπτωση.
- Το πρόγραμμά σας πρέπει να επιστρέφει κωδικό εξόδου (exit code) 0 όταν ολοκληρώσει τον υπολογισμό του επιτυχώς - διαφορετικά πρέπει να επιστρέφει με κωδικό εξόδου 1.

####### Αναδρομική μέθοδος `--mode=recurse` (10%)

Στο αρχείο `recurse.c` θέλουμε να γράψετε μια λύση στο πρόβλημα με αναδρομικό τρόπο βασισμένη στην Σχέση που δόθηκε στην περιγραφή του προβλήματος. Η διεπαφή (interface) της λύσης είναι πάνω σε εσάς, αλλά θέλουμε υποχρεωτικά η υλοποίησή σας στο `recurse.c` να καλείται από την `main` που θα βρίσκεται στο `elevate.c`. Στο πρόγραμμά σας δεν επιτρέπεται να ορίσετε άλλον πίνακα εκτός από αυτόν που χρειάζεται για τη φύλαξη των προορισμών των επιβατών. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ cat input1.txt
5 2
11 2 7 13 7
$ ./elevate input1.txt --mode=recurse
Last stop at floor: 11
The minimum cost is: 4
$ cat input2.txt
7 3
8 10 3 8 12 6 5
$ ./elevate input2.txt --mode=recurse
Last stop at floor: 10
The minimum cost is: 5
$ cat input3.txt
6 4
5 3 3 5 5 3
$ ./elevate input3.txt --mode=recurse
Last stop at floor: 5
The minimum cost is: 0
$ cat input4.txt
5 0
1 2 3 4 5
$ ./elevate input4.txt --mode=recurse
No lift stops
The minimum cost is: 15
$ cat input5.txt
20 4
8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44
```

```
$ time ./elevate input5.txt --mode=recurse
Last stop at floor: 42
The minimum cost is: 30
0.515u 0.003s 0:13.29 3.8%      0+0k 0+0io 0pf+0w
$ cat input6.txt
20 5
8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44
$ time ./elevate input6.txt --mode=recurse
Last stop at floor: 42
The minimum cost is: 20
4.318u 0.000s 0:14.74 29.2%     0+0k 0+0io 0pf+0w
$ cat input7.txt
20 6
8 32 14 14 6 7 25 43 12 9 1 28 27 25 33 38 42 27 14 44
$ time ./elevate input7.txt --mode=recurse
Last stop at floor: 43
The minimum cost is: 15
31.997u 0.000s 0:38.59 82.8%    0+0k 0+0io 0pf+0w
$ cat input8.txt
25 7
1 2 2 3 5 6 6 8 10 13 15 15 16 17 18 18 18 20 22 25 30 38 49 55 62
$ time ./elevate input8.txt --mode=recurse
Last stop at floor: 62
The minimum cost is: 35
3253.247u 0.003s 54:26.76 99.5% 0+0k 0+0io 0pf+0w
```

####### Μέθοδος ”ωμής βίας” `--mode=brute` (30%)

Παρατηρήστε ότι στις τελευταίες ενδεικτικές εκτελέσεις της αναδρομικής μεθόδου, ο χρόνος αυξάνει δραματικά. Μία εντελώς διαφορετική μέθοδος για να προσεγγιστεί το πρόβλημα, η οποία δεν βασίζεται στην αναδρομική σχέση που δόθηκε, είναι να κατασκευάζονται συστηματικά όλες οι πιθανές λύσεις, δηλαδή οι δυνατοί συνδυασμοί στάσεων του ανελκυστήρα (brute force), για κάθε μία από αυτές να υπολογίζεται το κόστος της και, τελικά, σαν λύση να δοθεί αυτή που έχει το ελάχιστο κόστος. Αν ο μέγιστος αριθμός στάσεων είναι $numStops$, θα πρέπει να εξετασθούν όλες οι πιθανές λύσεις με 1 στάση, 2 στάσεις, κοκ., $numStops$ στάσεις. Υπενθυμίζεται ότι έχει νόημα να συζητάμε για στάσεις από 1 μέχρι $numFloors$. Η υλοποίησή σας πρέπει να γίνει εξ ολοκλήρου στο `brute.c` και να καλείται και πάλι από την `main` εντός του `elevate.c`. Ενδεικτικές εκτελέσεις ακολουθούν:

```
$ ./elevate input1.txt --mode=brute
Lift stops are: 7 11
The minimum cost is: 4
$
$ ./elevate input2.txt --mode=brute
```

```
Lift stops are: 5 8 10
The minimum cost is: 5
$
$ ./elevate input3.txt --mode=brute
Lift stops are: 3 5
The minimum cost is: 0
$
$ ./elevate input4.txt --mode=brute
No lift stops
The minimum cost is: 15
$
$ time ./elevate input5.txt --mode=brute
Lift stops are: 7 14 27 42
The minimum cost is: 30
0.035u 0.003s 0:02.39 1.2%      0+0k 0+0io 0pf+0w
$
$ time ./elevate input6.txt --mode=brute
Lift stops are: 7 14 27 32 42
The minimum cost is: 20
0.240u 0.003s 0:02.87 8.3%      0+0k 0+0io 0pf+0w
$
$ time ./elevate input7.txt --mode=brute
Lift stops are: 7 14 27 32 38 43
The minimum cost is: 15
1.732u 0.003s 0:04.15 41.6%     0+0k 0+0io 0pf+0w
$
$ time ./elevate input8.txt --mode=brute
Lift stops are: 6 15 18 25 38 49 62
The minimum cost is: 35
118.186u 0.000s 2:00.45 98.1%   0+0k 0+0io 0pf+0w
```

####### Αναδρομή με απομνημόνευση `--mode=memoize` (20%)

Παρότι με την προηγούμενη εκδοχή βελτιώθηκαν σημαντικά οι χρόνοι εκτέλεσης, όταν το μέγεθος της εισόδου μεγαλώνει αρκετά, η χρονική απόδοση του προγράμματος δεν είναι καλή, αφού ελέγχουμε εξαντλητικά όλες τις πιθανές λύσεις, το πλήθος των οποίων αυξάνει εκθετικά με το πλήθος των ορόφων στους οποίους είναι πιθανό να γίνουν στάσεις.

Οπότε, ας επιστρέψουμε στην αναδρομική μέθοδο. Γιατί όμως η αναδρομική μέθοδος έχει ιδιαίτερα μεγάλους χρόνους εκτέλεσης για μεγάλες εισόδους; Μπορούμε να δούμε ότι εφαρμόζοντας την αναδρομική σχέση που δόθηκε, τα περισσότερα M_{ij} υπολογίζονται περισσότερες από μία φορά το καθένα, αρκετά από αυτά υπερβολικά μεγάλο αριθμό φορών. Πώς θα μπορούσαμε να το διορθώσουμε; Αρκεί να χρησιμοποιήσουμε ένα δισδιάστατο πίνακα στον οποίο να αποθηκεύεται κάθε M_{ij} την πρώτη φορά που υπολογίζεται και όταν χρειάζεται

πάλι, να μην επαναϋπολογίζεται, αλλά να λαμβάνεται η τιμή του από τον πίνακα. Η λογική της βελτιωμένης μεθόδου εξακολουθεί να είναι αναδρομική, απλώς κάθε M_{ij} υπολογίζεται ακριβώς μία φορά. Αυτή η τεχνική λέγεται απομνημόνευση—*memoization* στα αγγλικά (προσοχή *όχι* memorization). Υλοποιήστε αυτήν την μέθοδο στο αρχείο `memoize.c` και ως συνήθως καλέστε την από την `main` όταν δοθεί το κατάλληλο όρισμα από την γραμμή εντολών. Ενδεικτικές εκτελέσεις ακολουθούν:

```
$ ./elevate input1.txt --mode=memoize
Last stop at floor: 11
The minimum cost is: 4
$
$ ./elevate input2.txt --mode=memoize
Last stop at floor: 10
The minimum cost is: 5
$
$ ./elevate input3.txt --mode=memoize
Last stop at floor: 5
The minimum cost is: 0
$
$ ./elevate input4.txt --mode=memoize
No lift stops
The minimum cost is: 15
$
$ time ./elevate input5.txt --mode=memoize
Last stop at floor: 42
The minimum cost is: 30
0.006u 0.000s 0:02.37 0.0%      0+0k 0+0io 0pf+0w
$
$ time ./elevate input6.txt --mode=memoize
Last stop at floor: 42
The minimum cost is: 20
0.006u 0.000s 0:01.88 0.0%      0+0k 0+0io 0pf+0w
$
$ time ./elevate input7.txt --mode=memoize
Last stop at floor: 43
The minimum cost is: 15
0.007u 0.000s 0:02.05 0.0%      0+0k 0+0io 0pf+0w
$
$ time ./elevate input8.txt --mode=memoize
Last stop at floor: 62
The minimum cost is: 35
0.007u 0.003s 0:02.32 0.0%      0+0k 0+0io 0pf+0w
```

####### Δυναμικός Προγραμματισμός `--mode=dp` (40%)

Στην αναδρομική μέθοδο με απομνημόνευση, η λογική του υπολογισμού των M_{ij} είναι με σειρά *από-επάνω-προς-τα-κάτω* (top-down), δηλαδή αρχίζουμε από τα μεγάλα i, οπότε απαιτείται ο υπολογισμός των τιμών για μικρότερα i, κοκ., μέχρι να φτάσουμε στο $i = 0$. Μία αντίστροφη λογική είναι να συμπληρωθεί ο πίνακας των M_{ij} με σειρά *από-κάτω-προς-τα-επάνω* (bottom-up): αρχίζουμε τους υπολογισμούς για $i = 0$, συνεχίζουμε με $i = 1$ και τελικά καταλήγουμε στους υπολογισμούς για $i = numStops$. Η προσέγγιση αυτή χαρακτηρίζεται στη βιβλιογραφία ως *δυναμικός προγραμματισμός* (dynamic programming) και είναι από τις πιο συχνά χρησιμοποιούμενες τεχνικές βελτιστοποίησης. Χρησιμοποιώντας δυναμικό προγραμματισμό, υλοποιήστε στο αρχείο `dp.c` την εύρεση της βέλτιστης λύσης. Όταν δίνεται το επιπλέον όρισμα `--debug`, θέλουμε η λύση σας να εκτυπώνει και τις τιμές M_{ij}, μία γραμμή για κάθε i (από 0 έως $numStops$) καθώς και όλες τις στάσεις του για την βέλτιστη λύση! Σκεφτείτε πώς θα μπορούσε να γίνει αυτό. Αρκούν οι τιμές M_{ij}; Μάλλον όχι. Μία ιδέα, όχι όμως δεσμευτική, είναι να χρησιμοποιηθεί και ένας δεύτερος δισδιάστατος πίνακας, στον οποίο να φυλάσσεται για κάθε (i, j) η τιμή του k για την οποία στη στάση $i - 1$ βρέθηκε η ελάχιστη τιμή για το $M(i, j)$, σύμφωνα με την αναδρομική σχέση που δόθηκε αρχικά. Θα πρέπει να σκεφτείτε, βέβαια, πώς θα εκμεταλλευτείτε τα περιεχόμενα αυτού του επιπλέον πίνακα για να βρείτε τις στάσεις του ανελκυστήρα. Ενδεικτικές εκτελέσεις ακολουθούν (στην εκφώνηση οι μεγάλες γραμμές αναδιπλώνονται· εδώ κάθε γραμμή $M_{i,\cdot}$ είναι σε μία γραμμή):

```
$ ./elevate input1.txt --mode=dp --debug
40 40 40 40 40 40 40 40 40 40 40 40 40 40
40 35 30 27 24 20 16 12 12 12 12 12 14 16
40 35 30 26 22 18 14 10 10  8  6  4  4  4
Lift stops are: 7 11
The minimum cost is: 4
$
$ ./elevate input2.txt --mode=dp --debug
52 52 52 52 52 52 52 52 52 52 52 52 52
52 45 38 31 26 21 18 16 14 16 18 21 24
52 45 38 31 25 19 15 13  9  9  9 10 10
52 45 38 31 25 19 14 11  7  7  5  5  5
Lift stops are: 5 8 10
The minimum cost is: 5
$
$ ./elevate input3.txt --mode=dp --debug
24 24 24 24 24 24
24 18 12  6  6  6
24 18 12  6  3  0
24 18 12  6  3  0
24 18 12  6  3  0
Lift stops are: 3 5
The minimum cost is: 0
$
```

```
$ ./elevate input4.txt --mode=dp --debug
15 15 15 15 15 15
No lift stops
The minimum cost is: 15
$
$ time ./elevate input5.txt --mode=dp --debug
449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449
 ↪  449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449
 ↪  449 449 449 449 449 449
449 429 411 392 373 354 335 318 303 290 279 268 257 247 237 232 227 221 215 208
 ↪  201 194 187 180 173 165 161 157 157 156 155 154 153 154 157 160 163 166 169
 ↪  174 179 184 189 196 205
449 429 410 391 372 353 334 317 301 286 272 258 243 230 217 210 203 195 187 179
 ↪  169 158 147 136 125 114 107 100  97  96  95  94  93  94  97 100 103 106 107 109
 ↪  108 107 105 105 107
449 429 410 391 372 353 334 316 300 285 271 256 241 228 215 206 195 184 173 162
 ↪  151 140 129 118 107  96  89  82  79  78  77  76  70  66  64  62  60  58  55  54
 ↪  52  50  48  48  50
449 429 410 391 372 353 334 316 299 284 270 255 240 227 213 204 193 182 171 160
 ↪  149 138 127 116 105  94  87  78  73  70  64  58  52  48  46  44  42  40  37  36
 ↪  34  32  30  30  32
Lift stops are: 7 14 27 42
The minimum cost is: 30
0.003u 0.003s 0:02.28 0.0%      0+0k 0+0io 0pf+0w
$
$ time ./elevate input6.txt --mode=dp --debug
449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449
 ↪  449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449
 ↪  449 449 449 449 449 449
449 429 411 392 373 354 335 318 303 290 279 268 257 247 237 232 227 221 215 208
 ↪  201 194 187 180 173 165 161 157 157 156 155 154 153 154 157 160 163 166 169
 ↪  174 179 184 189 196 205
449 429 410 391 372 353 334 317 301 286 272 258 243 230 217 210 203 195 187 179
 ↪  169 158 147 136 125 114 107 100  97  96  95  94  93  94  97 100 103 106 107 109
 ↪  108 107 105 105 107
449 429 410 391 372 353 334 316 300 285 271 256 241 228 215 206 195 184 173 162
 ↪  151 140 129 118 107  96  89  82  79  78  77  76  70  66  64  62  60  58  55  54
 ↪  52  50  48  48  50
449 429 410 391 372 353 334 316 299 284 270 255 240 227 213 204 193 182 171 160
 ↪  149 138 127 116 105  94  87  78  73  70  64  58  52  48  46  44  42  40  37  36
 ↪  34  32  30  30  32
```

```
449 429 410 391 372 353 334 316 299 283 269 254 239 226 212 202 191 180 169 158
 ↪  147 136 125 114 103  92  85  76  71  66  60  54  48  44  42  40  36  32  28  26
 ↪  24  22  20  20  21
Lift stops are: 7 14 27 32 42
The minimum cost is: 20
0.006u 0.000s 0:02.01 0.0%      0+0k 0+0io 0pf+0w
$
$ time ./elevate input7.txt --mode=dp --debug
449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449
 ↪  449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449 449
 ↪  449 449 449 449 449 449
449 429 411 392 373 354 335 318 303 290 279 268 257 247 237 232 227 221 215 208
 ↪  201 194 187 180 173 165 161 157 157 156 155 154 153 154 157 160 163 166 169
 ↪  174 179 184 189 196 205
449 429 410 391 372 353 334 317 301 286 272 258 243 230 217 210 203 195 187 179
 ↪  169 158 147 136 125 114 107 100  97  96  95  94  93  94  97 100 103 106 107 109
 ↪  108 107 105 105 107
449 429 410 391 372 353 334 316 300 285 271 256 241 228 215 206 195 184 173 162
 ↪  151 140 129 118 107  96  89  82  79  78  77  76  70  66  64  62  60  58  55  54
 ↪  52  50  48  48  50
449 429 410 391 372 353 334 316 299 284 270 255 240 227 213 204 193 182 171 160
 ↪  149 138 127 116 105  94  87  78  73  70  64  58  52  48  46  44  42  40  37  36
 ↪  34  32  30  30  32
449 429 410 391 372 353 334 316 299 283 269 254 239 226 212 202 191 180 169 158
 ↪  147 136 125 114 103  92  85  76  71  66  60  54  48  44  42  40  36  32  28  26
 ↪  24  22  20  20  21
449 429 410 391 372 353 334 316 299 283 268 253 238 225 211 201 190 179 168 157
 ↪  146 135 124 113 102  91  83  74  69  64  58  52  46  42  40  36  32  28  24  22
 ↪  20  18  16  15  16
Lift stops are: 7 14 27 32 38 43
The minimum cost is: 15
0.007u 0.000s 0:02.02 0.0%      0+0k 0+0io 0pf+0w
$
$ time ./elevate input8.txt --mode=dp --debug
474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474
 ↪  474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474
 ↪  474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474 474
 ↪  474 474 474 474 474
474 449 426 406 388 368 350 335 320 307 294 282 270 256 244 232 224 217 212 213
 ↪  214 216 218 222 226 230 236 241 246 251 256 261 266 270 274 277 280 280 280
 ↪  282 284 285 286 287 288 288 288 288 288 288 290 291 292 293 294 295 298 301
 ↪  304 307 310 312 314
```

```
474 449 425 404 384 363 344 329 314 298 282 267 252 237 224 210 200 192 186 186
 ↪  186 186 180 176 172 167 164 161 157 153 149 147 145 143 140 137 134 131 128
 ↪  127 126 125 124 122 120 118 116 114 112 110 110 110 110 110 110 110 112 114
 ↪  116 117 118 119 120
474 449 425 403 383 362 343 326 308 292 276 261 246 231 218 204 194 186 176 172
 ↪  167 162 156 152 147 142 139 136 132 128 124 122 120 117 114 111 108 105 102
 ↪  101 100  99  98  96  94  92  90  88  86  84  84  84  84  84  84  84  86  88  89
 ↪  90  91  92  93
474 449 425 403 382 361 342 325 307 291 274 259 244 228 214 200 190 180 170 166
 ↪  160 154 148 143 138 133 130 126 122 118 114 112 110 107 104 101  98  94  90
 ↪  88  86  84  82  80  78  76  74  72  70  67  66  65  64  63  62  61  62  63  64
 ↪  65  66  67  68
474 449 425 403 382 361 341 324 306 290 273 258 242 226 212 198 187 176 166 162
 ↪  154 148 142 137 132 127 124 120 116 112 108 105 102  99  96  93  89  85  81  79
 ↪  77  75  73  71  69  67  65  63  60  57  56  55  54  53  52  51  52  53  54  55
 ↪  55  55  54
474 449 425 403 382 361 341 323 305 289 272 257 241 225 211 196 185 174 164 158
 ↪  150 144 138 133 128 123 120 116 112 107 102  99  96  93  90  87  83  79  75  73
 ↪  71  69  67  65  63  60  57  54  51  48  47  46  45  44  43  42  43  44  45  45
 ↪  45  45  44
474 449 425 403 382 361 341 323 305 288 271 256 240 224 210 195 183 172 162 156
 ↪  148 142 136 131 126 120 116 112 108 103  98  95  92  89  86  82  78  74  70  68
 ↪  66  64  62  60  57  54  51  48  45  42  41  40  39  38  37  36  36  36  36  36
 ↪  36  36  35
Lift stops are: 6 15 18 25 38 49 62
The minimum cost is: 35
0.010u 0.000s 0:02.29 0.4%      0+0k 0+0io 0pf+0w
```

Παρατηρήστε ότι ο χρόνος επίλυσης παραμένει μικρός! Ποια μέθοδο θα επιλέγατε για χρήση στην παραγωγή και γιατί; Δικαιολογήστε την επιλογή σας στο README και φροντίστε η επιλογή αυτή να είναι και η προεπιλεγμένη (default) αν ο χρήστης δεν επιλέξει κάποια από τα γνωστά modes.

####### Model Context Protocol (MCP) Server — Προαιρετικό Bonus 50%

Μέχρι στιγμής στο μάθημα είδαμε τρεις βασικούς τρόπους εισόδου για προγράμματα: (1) ορίσματα στην γραμμή εντολών, (2) πρότυπη είσοδο και (3) αρχεία. Υπάρχει κάποια καλύτερη λύση όμως αν ο χρήστης είναι ένα σύστημα τεχνητής νοημοσύνης; Τον Νοέμβριο του 2024 η εταιρεία Anthropic πρότεινε το Model Context Protocol (MCP), που θεωρείται σήμερα *το* standard για την υλοποίηση διεπαφών για συστήματα τεχνητής νοημοσύνης. Παρόλο που περιέχει φοβιστικούς όρους όπως ”Server” και ”Protocol”, στην πιο απλή μορφή του είναι απλά ένα πρόγραμμα που διαβάζει από το stdin και τυπώνει στο stdout.

Η αποστολή σας λοιπόν στα πλαίσια αυτού του bonus—εφόσον την αποδεχτείτε—θα είναι η υλοποίηση ενός MCP server για τον elevate λύτη σας. Πιο συγκεκριμένα, θα χρειαστεί να

υλοποιήσετε μια νέα συνάρτηση `main` σε ένα αρχείο `elevate/src/mcp.c` το οποίο θα πρέπει να μεταγλωττίζεται με αντίστοιχο τρόπο όπως παραπάνω και θα προσφέρει τις ακόλουθες δυνατότητες:

- **Initialize (Αρχικοποίηση)**. Όταν ο server σας ξεκινά, περιμένει να του στείλει κάποιος χρήστης ένα μήνυμα που να ζητήσει πληροφορίες για τις δυνατότητές του και να απαντήσει κατάλληλα.
- **List Tools (Λίστα Δυνατοτήτων)**. Μετά την χειραψία το μοντέλο μπορεί να ζητήσει από τον server σας να τυπώσει τις δυνατότητές του στην πρότυπη έξοδο, έτσι ώστε το σύστημα τεχνητής νοημοσύνης να γνωρίζει πως να το αξιοποιήσει.
- **Tool Call Response (Απαντήσεις σε Ερωτήσεις)**. Για κάθε ερώτηση που ταιριάζει στις δυνατότητες που διαφήμισε ο server σας παραπάνω, το πρόγραμμά σας πρέπει να μπορεί να προσφέρει απάντηση.

Η συνολική διάδραση είναι: ο client στέλνει `initialize` (stdin) και ο server απαντά με το αποτέλεσμα (stdout)· ο client στέλνει `tools/list` και ο server απαντά με τη λίστα των εργαλείων· ο client στέλνει `tools/call` για το `elevate` με ορίσματα, ο server τρέχει `elevate(numPeople, numStops, dests, mode)` και απαντά με `structuredContent`. Ας ξεκινήσουμε το πρόγραμμά μας:

```
$ gcc -Os -c -Wall -Wextra -Werror -pedantic recurse.c
$ gcc -Os -c -Wall -Wextra -Werror -pedantic brute.c
$ gcc -Os -c -Wall -Wextra -Werror -pedantic memoize.c
$ gcc -Os -c -Wall -Wextra -Werror -pedantic dp.c
$ gcc -Os -c -Wall -Wextra -Werror -pedantic mcp.c
$ gcc -Os -o mcp recurse.o brute.o memoize.o dp.o mcp.o
$ ./mcp
```

Initialize. Ο server μας περιμένει είσοδο από τον χρήστη. Ας ξεκινήσουμε την χειραψία στέλνοντας το αντίστοιχο μήνυμα αρχικοποίησης:

```
{"jsonrpc":"2.0","id":1,"method":"initialize","params":
{"protocolVersion":"2025-11-25","capabilities":{"tools":{}},"clientInfo":
{"name":"example-mcp-client","version":"1.0.0","title":"Example MCP Client"}}}
```

Το παραπάνω μήνυμα είναι σε μορφή JSON, ενώ το όλο πρωτόκολλο βασίζεται σε έναν τρόπο κωδικοποίησης που λέγεται JSON-RPC. Για λόγους αποδοτικότητας αποφεύγουμε τις καινούριες γραμμές (εδώ χρησιμοποιούνται μόνο για να φαίνεται το μήνυμα) καθώς και τα κενά (για να διαβάσετε ένα JSON αρχείο με καλύτερη αναγνωσιμότητα μπορείτε να τρέξετε `python3 -m json.tool < file.json`). Σε ένα τέτοιο μήνυμα αρχικοποίησης ο server μας πρέπει να απαντήσει με το όνομά του:

```
{"jsonrpc":"2.0","id":1,"result":{"protocolVersion":"2025-06-18","capabilities":
{"experimental":{},"prompts":{"listChanged":false},"resources":
↪ {"subscribe":false,"listChanged":false},"tools":{"listChanged":true}},"serverInfo":
```

```
{"name":"elevate-mcp-server","version":"0.0.1"}}}
```

List Tools. Τώρα ο χρήστης μπορεί να μάθει περισσότερα για τον server στέλνοντας το μήνυμα `tools/list`:

```
{"jsonrpc":"2.0","id":3,"method":"tools/list","params":{}}
```

Στο οποίο ο server μας απαντάει ως εξής:

```
{"jsonrpc":"2.0","id":3,"result":{"tools":[{"name":"elevate","description":
"Compute the optimal elevator final stop and cost.","inputSchema":
{"properties":{"numPeople":{
"description":"Number of people that want to use the elevator.","type":"integer"},
"numStops":{"description":"Maximum number of stops the elevator can make.",
"type":"integer"},"dests":{"description":
"Destination stop of each passenger.","items":{"type":"integer"},"type":"array"},
"mode":{"default":"dp","description":
"Solving mode to be used: recurse, brute, memoize, or dp.",
"enum":["recurse","brute","memoize","dp"],"type":"string"}},"required":
["numPeople","numStops","dests"],"type":"object"},"outputSchema":
{"description":"Result of the elevator optimization.","properties":{"finalStop":
{"description":"Chosen final elevator stop (highest floor the elevator visits).",
"type":"integer"},
"minCost":{"description":"Total minimal walking cost for all passengers.",
"type":"integer"},"status":{"description":"0 on success, non-zero on error.",
"type":"integer"},"message":{"description":
"Human-readable status or error message.","type":"string"},"time":
{"description":"Elapsed time for the computation in microseconds.",
"type":"integer"}},"required":["finalStop","minCost","status","message","time"],
"type":"object"},"_meta":{"_fastmcp":{"tags":[]}}}]}}
```

Ο server μας τύπωσε τις δυνατότητές του (όλες τις παραμέτρους που δέχεται, περιγραφές τους καθώς και τύπους) και περιμένει εντολές από το σύστημα τεχνητής νοημοσύνης.

Tool Call Response. Ας του πούμε τι θέλουμε, κάνοντας copy-paste στο stdin:

```
{"jsonrpc":"2.0","id":2,"method":"tools/call","params":
{"name":"elevate","arguments":{"numPeople":5,"numStops":2,"dests":
[11,2,7,13,7],"mode":"dp"}}}
```

και στο stdout παρατηρούμε την απάντηση:

```
{"jsonrpc":"2.0","id":2,"result":{"content":
[{"type":"text","text":"{\"finalStop\":11,\"minCost\":4,\"status\":0,\"message\":
\"OK\",\"time\":15}"}],"structuredContent":
⌋
↪  {"finalStop":11,"minCost":4,"status":0,"message":"OK","time":15},"isError":false}}
```

Αυτό ήταν! Ο server μας πλέον είναι συμβατός με οποιοδήποτε σύστημα τεχνητής νοημοσύνης υποστηρίζει MCP, όπως το Claude Code, VS Code, Cursor, κοκ. Η εκφώνηση δείχνει μια ενδεικτική διάδραση όπου ο server προστίθεται με `claude mcp add --transport stdio elevate -- ./mcp` και το μοντέλο τον καλεί για το παράδειγμα με `dests = [11,2,7,13,7]` και για benchmarking όλων των modes. Αν κάποιο παιδί καταφέρει να κάνει live demo (όχι βίντεο, πραγματική διάδραση) της MCP σύνδεσης με κάποιο πραγματικό μοντέλο κατά την προφορική εξέταση, η βαθμολογία για το bonus κομμάτι διπλασιάζεται.

Επίλογος Πριν την υποβολή της εργασίας, μην ξεχάσετε το `elevate/README.md`, μέσα στο οποίο πρέπει να συμπεριλάβετε τις όποιες παρατηρήσεις σας κατά την διεκπεραίωση της άσκησης! Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης.

Υπόδειξη Ξεκίνα από μια συνάρτηση `fw(a, b)` που χειρίζεται σωστά το $b = \infty$ και έλεγξέ την με το χέρι στο πρώτο παράδειγμα (π.χ. $M_{0,j} = 40$). Για το `brute` σκέψου πώς απαριθμούνται όλοι οι αύξοντες συνδυασμοί στάσεων (με αναδρομή ή με έναν «μετρητή» σαν οδόμετρο). Τα memoize και dp χρειάζονται πίνακα $(numStops + 1) \times (numFloors + 1)$ που δεσμεύεται δυναμικά, με μια ειδική τιμή για «δεν υπολογίστηκε ακόμα»· για να ανακατασκευάσεις τις στάσεις κράτα σε δεύτερο πίνακα ποιο k έδωσε το ελάχιστο και ακολούθησέ το προς τα πίσω. Πρόσεξε τις ισοπαλίες (προτιμώνται χαμηλότεροι όροφοι) και την περίπτωση $numStops = 0$.

Θέματα εξετάσεων (A25.5–A25.17)

A25.5 · Σκαλί-Σκαλί

A25.5 · *Κατατακτήριες Δεκεμβρίου 2023, Θέμα 3* · ★★☆ · programming · `exam-2023-dec-q3`

Ένα παιδί ανεβαίνει μια σκάλα και σε κάθε βήμα του ανεβαίνει 1, 2 ή 3 σκαλιά. Έστω ότι η σκάλα έχει 5 σκαλιά. Ένας τρόπος να τα ανέβει είναι 1-1-1-1-1, δηλαδή ένα σκαλί την φορά. Ένας άλλος είναι ο 2-2-1 (δύο σκαλιά στα πρώτα δύο βήματα και μετά ένα σκαλί). Άλλοι πιθανοί τρόποι είναι οι 3-2, 2-3, 3-1-1, κ.ο.κ - σε κάθε περίπτωση ο συνολικός αριθμός των σκαλιών που ανέβηκε πρέπει να ισούται με τον αριθμό των σκαλιών της σκάλας. Γράψτε ένα πρόγραμμα C το οποίο διαβάζει τον συνολικό αριθμό των σκαλοπατιών και επιστρέφει τον αριθμό των διαφορετικών τρόπων με τους οποίους το παιδί μπορεί να ανέβει την σκάλα. Το πρόγραμμά σας θέλουμε να βγάζει σωστά αποτελέσματα για σκάλες μέχρι 50 σκαλιά. Λάβετε υπόψη σας ότι στην C δεν μπορούν να αναπαρασταθούν ακέραιοι μεγαλύτεροι από το ^{64}-1, $δεδομνουτιοευρτεροστποσακεραουπουμποροµενα χουμεεναιο`unsignedlonglong`(των8bytes).Οαλγρ$ όπου $ ο αριθμός των σκαλιών. Ακολουθούν ενδεικτικές εκτελέσεις:

```
$ ./step
Please provide the number of steps: 4
There are 7 different ways to climb the ladder
```

```
$ ./step
Please provide the number of steps: 5
There are 13 different ways to climb the ladder
$ ./step
Please provide the number of steps: 6
There are 24 different ways to climb the ladder
$ ./step
Please provide the number of steps: 50
There are 10562230626642 different ways to climb the ladder
```

Υπόδειξη Σκεφτείτε το τελευταίο βήμα του παιδιού: ήταν 1, 2 ή 3 σκαλιά, οπότε οι τρόποι για $ σκαλιά εκφράζονται μέσω των τρόπων για λιγότερα σκαλιά. Η απλή αναδρομή επαναϋπολογίζει τα ίδια υποπροβλήματα εκθετικά πολλές φορές· υπολογίστε τις τιμές από κάτω προς τα πάνω (ή απομνημονεύστε τις) ώστε να πετύχετε (n)$. Προσέξτε τις αρχικές περιπτώσεις και χρησιμοποιήστε `unsigned long long` με `%llu`.

A25.6 · Μένοντας στις Σωστές Θερμίδες

A25.6 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #14, Θέμα 2* · ★★☆ · programming · `exam-2023-fall-ex14-q2`

Πρόγραμμα: `right.c`

Γράψτε ένα πρόγραμμα το οποίο δέχεται ως πρώτο όρισμα ένα σύνολο θερμίδων το οποίο δεν πρέπει να υπερβούμε και τα υπόλοιπα ορίσματα είναι οι θερμίδες που περιέχει κάθε πιάτο μπροστά μας και τυπώνει τον μέγιστο αριθμό πιάτων που μπορούμε να φάμε χωρίς να υπερβούμε το όριο θερμίδων. Οι θερμίδες μπορούν να είναι μόνο θετικοί ακέραιοι και κάθε όρισμα αντιστοιχεί σε ακριβώς ένα πιάτο μπροστά μας (δεν μπορούμε να φάμε το ίδιο πιάτο δύο φορές!). Παραδείγματα εκτέλεσης ακολουθούν:

```
$ gcc -o right right.c
$ ./right
Need a target set of calories
$ ./right 100 101
Within 100 calories we can fit 0 different plates
$ ./right 100 30 70 20 85 15 9 50
Within 100 calories we can fit 4 different plates
$ ./right 1000 29 37 982 3 38 3899 22 737 89 478 27892 787 2897
Within 1000 calories we can fit 7 different plates
```

Υπόδειξη Σκεφτείτε ποια πιάτα συμφέρει να διαλέξετε πρώτα αν θέλετε όσο το δυνατόν περισσότερα· η ταξινόμηση βοηθά. Ελέγξτε ότι όλα τα ορίσματα είναι θετικοί ακέραιοι.

A25.7 · Ανεβαίνοντας Επίπεδο

A25.7 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #2 (Pokémon Themed), Θέμα 4* · ★★☆ · programming · `exam-2023-fall-ex2-q4`

Πρόγραμμα: `levelup.c` (25 μονάδες)

Ένα pokémon μπορεί να ανέβει 1, 2 ή 3 επίπεδα σε κάθε αγώνα ανάλογα με το πόσο δύσκολος είναι ο αντίπαλος (εύκολος, μεσαίος, δύσκολος). Γράψτε ένα πρόγραμμα που παίρνει ένα επίπεδο "στόχο" ως όρισμα και τυπώνει στην πρότυπη έξοδο με πόσους διαφορετικούς τρόπους μπορεί να φτάσει σε αυτό. Έστω ότι θέλουμε να φτάσουμε στο επίπεδο 5. Ένας τρόπος να φτάσουμε είναι 1-1-1-1-1, δηλαδή να κερδίσουμε 5 εύκολους αντιπάλους. Ένας άλλος είναι ο 2-2-1 (δύο μεσαίους και μετά έναν εύκολο). Άλλοι πιθανοί τρόποι είναι οι 3-2, 2-3, 3-1-1, κ.ο.κ - σε κάθε περίπτωση θέλουμε να φτάσουμε *ακριβώς* στο επίπεδο στόχο. Παραδείγματα εκτελέσεων ακολουθούν:

```
$ gcc -o levelup.c levelup
$ ./levelup 3
There are 4 different ways to reach level 3
$ ./levelup 5
There are 13 different ways to reach level 5
$ ./levelup 6
There are 24 different ways to reach level 6
$ ./levelup 50
There are 10562230626642 different ways to reach level 50
```

Υπόδειξη Σκεφτείτε αναδρομικά: με πόσους τρόπους φτάνω στο `n` αν ο τελευταίος αγώνας μού έδωσε 1, 2 ή 3 επίπεδα; Η απλή αναδρομή όμως ξαναϋπολογίζει τα ίδια υποπροβλήματα εκθετικά πολλές φορές και δεν θα τελειώσει για `n = 50`· αποθηκεύστε τα ενδιάμεσα αποτελέσματα ή υπολογίστε από κάτω προς τα πάνω. Το αποτέλεσμα ξεπερνά τον `int`, και οι αρχικές τιμές (τα `n` 1, 2, 3 ή 0) θέλουν προσοχή.

A25.8 · Επενδύσεις στο Χρηματιστήριο

A25.8 · *Εξέταση Ιουνίου 2026, Θέμα 3* · ★★☆ · programming · `exam-2026-jun-q3`

Επενδύσεις στο Χρηματιστήριο (45 Μονάδες)

Σε μια στιγμή αυθορμητισμού, αποφασίσαμε να μπούμε ενεργά στον χώρο του χρηματιστηρίου και των μετοχών. Κάνοντας αυτήν την κίνηση, μάθαμε πως το πρώτο πρόβλημα που πρέπει να λύσουμε είναι το πότε πρέπει να πουλήσεις και πότε να αγοράσεις μια μετοχή. Αυτό θα είναι και το πρόβλημα που θα μας απασχολήσει σε αυτό το θέμα.

Γράψτε ένα πρόγραμμα το οποίο θα διαβάζει από την πρότυπη είσοδο τις τιμές που λαμβάνει μια μετοχή σε διαδοχικές μέρες και τυπώνει: (1) την ελάχιστη τιμή της μετοχής και (2) την μέγιστη τιμή της μετοχής. Επιπλέον, όταν δοθεί το όρισμα `--optimal` θέλουμε να τυπώνει στην πρότυπη έξοδο και τον βέλτιστο συνδυασμό πότε έπρεπε να αγοράσει κανείς την μετοχή, πότε να την πουλήσει καθώς και το αναμενόμενο κέρδος (17/45 της βαθμολογίας).

Περιγράψτε την χρονική και χωρική πολυπλοκότητα του αλγορίθμου σας ως προς τον αριθμό N των ακεραίων και εξηγήστε αν είναι η βέλτιστη (8/45 της βαθμολογίας).

Παραδείγματα εκτέλεσης ακολουθούν:

```
$ ./stonks
Give me the number of days: 10
Give me 10 stock values for each day: 7 1 5 3 6 4 9 2 8 3
Minimum: 1
Maximum: 9
$ ./stonks --optimal
Give me the number of days: 8
Give me 8 stock values for each day: 10 9 8 7 1 3 5 6
Minimum: 1
Maximum: 10
Optimal strategy to buy on day 5 and sell on day 8 for profit 5.
```

Υπόδειξη Παρατηρήστε στο δεύτερο παράδειγμα ότι η αγορά πρέπει να γίνει *πριν* την πώληση, άρα το κέρδος δεν είναι απλώς μέγιστο μείον ελάχιστο. Η απλή λύση δοκιμάζει κάθε ζεύγος ημερών σε $O(N^2)$· σκεφτείτε όμως τι αρκεί να θυμάστε από τις προηγούμενες μέρες όταν εξετάζετε την πώληση σε μια συγκεκριμένη μέρα, ώστε να αρκεί ένα πέρασμα. Προσέξτε ότι οι μέρες μετρούν από το 1 και ελέγξτε το `argv` με `strcmp`.

A25.9 · Το Νερό Νεράκι

A25.9 · *Κατατακτήριες Δεκεμβρίου 2023, Θέμα 4* · ★★★ · programming · `exam-2023-dec-q4`

Το νερό είναι πολύτιμο και είναι σημαντικό να γνωρίζουμε ποια είναι τα διαθέσιμα αποθεματά μας. Δοθέντος του υψομετρικού χάρτη μιας περιοχής, θέλουμε να μπορούμε να υπολογίζουμε την μέγιστη ποσότητα νερού της βροχής που μπορεί να διατηρηθεί μέσα στις φυσικές δεξαμενές που σχηματίζονται από την διαμόρφωση του εδάφους. Για απλοποίηση, θεωρούμε ότι ο υψομετρικός χάρτης είναι μονοδιάστατος και μας δίνεται ως μια σειρά θετικών ακεραίων (μονάδες ύψους με πλάτος 1) και η ποσότητα νερού μετριέται επίσης στις ίδιες μονάδες ύψους. Το Σχήμα 1 δείχνει ένα παράδειγμα.

Θέση	1	2	3	4	5	6	7	8	9	10	11	12
Γη (ύψος)	0	1	0	2	1	0	1	3	2	1	2	1
Νερό	0	0	1	0	1	2	1	0	0	1	0	0

Σχήμα 1: Οπτικοποίηση της ποσότητας νερού που θα "παγιδευτεί" μετά την βροχή για έναν ενδεικτικό υψομετρικό χάρτη. Στο παράδειγμα, μέχρι 6 μονάδες νερού μπορούν να συγκρατηθούν από αυτόν τον χάρτη (1 μονάδα στην 3η θέση, 1 στην 5η θέση, 2 στην 6η θέση, 1 στην 7η θέση και 1 στην 10η θέση).

> Σημείωση: στο πρωτότυπο το Σχήμα 1 είναι ραβδόγραμμα (καφέ η γη, μπλε το νερό)· εδώ δίνεται ως πίνακας με τα ίδια δεδομένα.

Γράψτε ένα πρόγραμμα C που διαβάζει τον πίνακα των υψών του υψομετρικού χάρτη και επιστρέφει την μέγιστη ποσότητα του νερού της βροχής που μπορεί να συγκρατηθεί από αυτόν. Το πρόγραμμά σας πρέπει να έχει χρονική πολυπλοκότητα καλύτερη από (n^2)$, όπου $ ο αριθμός των δοθέντων υψών και την ελάχιστη δυνατή χρήση μνήμης. Αφού γράψετε το πρόγραμμά σας, καταγράψτε και εξηγήστε την πολυπλοκότητά του ως προς τον χρόνο και τον χώρο μνήμης που απαιτεί. Ακολουθούν ενδεικτικές εκτελέσεις:

```
$ ./water
Provide the number of heights: 12
Provide the heights: 0 1 0 2 1 0 1 3 2 1 2 1
Up to 6 units of water can be trapped
$ ./water
Provide the number of heights: 12
Provide the heights: 4 1 0 2 1 0 1 3 2 3 2 1
Up to 14 units of water can be trapped
./water
Provide the number of heights: 20
Provide the heights: 0 2 3 1 4 4 4 4 3 8 9 3 9 0 6 4 6 0 9 5
Up to 38 units of water can be trapped
```

Υπόδειξη Το νερό πάνω από τη θέση $ ορίζεται από το μικρότερο από τα δύο ψηλότερα «τοιχώματα» αριστερά και δεξιά της, μείον το ύψος της θέσης. Με δύο προ-υπολογισμένους πίνακες μεγίστων παίρνετε (n)$ χρόνο αλλά (n)$ επιπλέον μνήμη· για ελάχιστη μνήμη σκεφτείτε δύο δείκτες που κινούνται από τα άκρα προς το κέντρο κρατώντας μόνο τα τρέχοντα μέγιστα. Δεσμεύστε τον πίνακα των υψών δυναμικά αφού διαβάσετε το $.

A25.10 · Τρόποι να Φάμε Παϊδάκια

A25.10 · *Online τελική εξέταση Δεκεμβρίου 2023, Εξέταση #14, Θέμα 4* · ★★★ · programming · `exam-2023-fall-ex14-q4`

Πρόγραμμα: `chops.c`

Ο Θάνος μόλις παρήγγειλε N παϊδάκια (τουλάχιστον 3) και αναρωτιέται με πόσους διαφορετικούς τρόπους μπορεί να τα φάει. Ξέρει ότι με κάθε του μπουκιά μπορεί να φάει 1 ή 2 παϊδάκια. Επειδή πεινάει πολύ σκοπεύει να φάει 3 παϊδάκια με ακριβώς μία μπουκιά. Για παράδειγμα, αν του φέρουν 5 παϊδάκια, μπορεί να τα φάει με 5 τρόπους: 1-1-3 (1 με μια μπουκιά, 1 με μια μπουκιά, 3 με μια μπουκιά), 1-3-1, 3-1-1, 2-3 και 3-2. Αν του φέρουν 3 παϊδάκια υπάρχει ακριβώς ένας τρόπος (3 με την μία). Αν του φέρουν 4 υπάρχουν 2 τρόποι (1-3, 3-1). Γράψτε ένα πρόγραμμα το οποίο παίρνει τον αριθμό από τα παϊδάκια ως ακέραιο όρισμα και τυπώνει τον αριθμό των διαφορετικών τρόπων με τους οποίους ο Θάνος μπορεί να τα φάει όλα. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ ./chops 2
I would never order less than 3 chops.
$ ./chops 3
There are 1 different ways to eat 3 chops.
$ ./chops 4
There are 2 different ways to eat 4 chops.
$ ./chops 5
There are 5 different ways to eat 5 chops.
$ ./chops 6
There are 10 different ways to eat 6 chops.
$ ./chops 50
There are 168903452400 different ways to eat 50 chops.
```

Υπόδειξη Βρείτε πρώτα με πόσους τρόπους τρώγονται k παϊδάκια μόνο με μπουκιές 1 ή 2 (αναδρομική σχέση). Μετά σκεφτείτε πού μπορεί να μπει η μοναδική μπουκιά των 3 και τι μένει αριστερά και δεξιά της. Για N = 50 χρειάζεστε `long long` και αποφυγή της εκθετικής αναδρομής (υπολογισμός από κάτω προς τα πάνω).

A25.11 · Η Τριπλέτα Στόχος

A25.11 · *Κατατακτήριες Δεκεμβρίου 2024, Θέμα 3* · ★★★ · programming · `exam-2024-dec-q3`

Γράψτε ένα πρόγραμμα το οποίο παίρνει ως όρισμα από την γραμμή εντολών έναν ακέραιο-στόχο (goal) και από την πρότυπη είσοδο (stdin) ένα σύνολο διαφορετικών υποψηφίων ακεραίων (candidates) και τυπώνει όλες τις πιθανές τριπλέτες υποψηφίων ακεραίων των οποίων το άθροισμα ισούται με τον στόχο. Ο κάθε υποψήφιος ακέραιος μπορεί να χρησιμοποιηθεί μέχρι μία φορά σε κάθε τριπλέτα. Για παράδειγμα αν δοθεί ο στόχος 10 και οι ακέραιοι 6, 1, 7, 2, 3 ως υποψήφιοι για τις τριπλέτες υπάρχουν ακριβώς δύο συνδυασμοί που οδηγούν στον στόχο: $1+3+6=10$ και $1+2+7=10$ (ο συνδυασμός $6+2+2=10$ *δεν* είναι δεκτός καθώς ο ακέραιος 2 επαναλαμβάνεται).

Ο αλγόριθμός σας πρέπει να έχει χρονική πολυπλοκότητα γρηγορότερη από $O(n^3)$, όπου n είναι ο αριθμός των ακεραίων που σας δίνονται. Παραδείγματα εκτέλεσης ακολουθούν:

```
$ ./triplet 10
Provide the numbers: 6 1 7 2 3
Triplet found: 1 + 2 + 7 = 10
Triplet found: 1 + 3 + 6 = 10
$ ./triplet 42
Provide the numbers: 1 2 3
No triplet leads to 42
$ ./triplet 42
Provide the numbers: 19 21 3 5 12 11
Triplet found: 11 + 12 + 19 = 42
$ ./triplet 42
Provide the numbers: 42 27 25 12 31 5 26 40 34 3 18
Triplet found: 3 + 5 + 34 = 42
Triplet found: 3 + 12 + 27 = 42
Triplet found: 5 + 12 + 25 = 42
```

Υπόδειξη Το πλήθος των αριθμών δεν είναι γνωστό από πριν, οπότε διαβάστε τους μέχρι το EOF σε πίνακα που μεγαλώνει με `realloc`. Ταξινομήστε τον πρώτα: αυτό κάνει τις τριπλέτες να βγαίνουν σε αύξουσα σειρά, όπως στα παραδείγματα, και σας επιτρέπει, για σταθερό μικρότερο στοιχείο, να ψάξετε τα άλλα δύο με δύο δείκτες που κινούνται ο ένας προς τον άλλον, αντί για τρίτο εμφωλευμένο βρόχο.

A25.12 · Λύσε τον Λαβύρινθο

A25.12 · *Κατατακτήριες Δεκεμβρίου 2024, Θέμα 4* · ★★★ · programming · `exam-2024-dec-q4`

> Σημείωση: το πρωτότυπο ξεκινά με το Σχήμα 1, δύο εικόνες του λαβυρίνθου 7x7 του αρχείου `maze.txt` παρακάτω: (i) ο αρχικός λαβύρινθος, όπου ξεκινάμε στο (0, 0) και θέλουμε να φτάσουμε στο (6, 6), και (ii) μια ενδεικτική λύση, όπου επιτρέπονται κινήσεις μόνο προς τα κάτω (D) και δεξιά (R). Οι εικόνες υπάρχουν στο PDF της εξέτασης.

Σχήμα 1: Οπτικοποίηση ενός λαβυρίνθου και της λύσης του. Παρατηρήστε ότι το ρομπότ μας ξεκινάει πάντα την διαδρομή του στο (0,0) και ολοκληρώνει την διαδρομή του στο (N-1, N-1) όπου N η διάσταση του λαβυρίνθου.

Σε αυτό το πρόβλημα θα προγραμματίσουμε ένα ρομπότ το οποίο μπορεί να βρίσκει την έξοδο σε μια κατηγορία τετραγωνικών λαβυρίνθων, όπου η είσοδος είναι πάνω αριστερά και η έξοδος κάτω δεξιά. Για λόγους οικονομίας, το ρομπότ μας μπορεί να κινηθεί μόνο προς τα κάτω (D - Down) ή προς τα δεξιά (R - Right) σε κάθε βήμα. Το Σχήμα 1 δείχνει ένα παράδειγμα με το ρομπότ μας να βρίσκει το μονοπάτι DDDDRRRRDDRR και να φτάνει στην έξοδο.

Γράψτε ένα πρόγραμμα C που διαβάζει τον πίνακα με τα περιεχόμενα του λαβυρίνθου και τυπώνει (εφόσον υπάρχει) ένα βέλτιστο μονοπάτι για να φτάσει το ρομπότ μας στην έξοδο. Το πρόγραμμά σας πρέπει να διαβάζει τον λαβύρινθο από την πρότυπη είσοδο όπου θα δίνονται: (1) στην πρώτη γραμμή η διάσταση του τετραγωνικού πλέγματος, και (2) στην συνέχεια τα περιεχόμενα του λαβυρίνθου όπου ”1” θα συμβολίζει ένα κελί με τοίχο και ”0” θα συμβολίζει το κενό. Το πρόγραμμά σας πρέπει να είναι αποδοτικό σε χρονική και χωρική πολυπλοκότητα. Αφού ολοκληρώσετε το πρόγραμμά σας, καταγράψτε και εξηγήστε την πολυπλοκότητά του ως προς τον χρόνο και τον χώρο μνήμης που απαιτεί. Ακολουθούν ενδεικτικές εκτελέσεις:

```
$ cat maze.txt
7
0001000
0111010
0100010
0101110
0000010
0111010
0100000
$ ./robot < maze.txt
Path: DDDDRRRRDDRR
$ cat nopath.txt
7
0001000
0111010
0100010
0101110
0000010
0111110
0100000
$ ./robot < nopath.txt
No path found
```

Υπόδειξη Αφού κινούμαστε μόνο κάτω και δεξιά, κάθε μονοπάτι έχει ακριβώς $2N - 2$ βήματα, άρα το ερώτημα είναι μόνο αν υπάρχει. Μια αναδρομική δοκιμή όλων των διαδρομών είναι εκθετική· αντί γι' αυτό, συμπληρώστε έναν δισδιάστατο πίνακα που λέει για κάθε κελί αν από εκεί φτάνουμε στην έξοδο (ή αποθηκεύστε τα αποτελέσματα της αναδρομής), ώστε κάθε κελί να εξετάζεται μία φορά. Προσέξτε τις περιπτώσεις όπου η αρχή ή η έξοδος είναι τοίχος και τον τρόπο που διαβάζετε γραμμές ψηφίων χωρίς κενά.

A25.13 · Βέλτιστη Μοιρασιά Πίτσας

A25.13 · *Εξέταση Ιουλίου 2024, Θέμα 3* · ★★★ · programming · `exam-2024-jul-q3`

Παραγγείλατε πίτσες για να δείτε τον τελικό του Euro και διαπιστώσατε τελευταία στιγμή πως ήρθαν υπερβολικά πολλοί καλεσμένοι (ως συνήθως κάποιοι αυτοπροσκλήθηκαν). Παρόλα αυτά κάνατε την καρδιά σας πέτρα και αποφασίσατε να ταΐσετε όσους περισσότερους γίνεται. Και τι καλύτερος τρόπος για να γίνει αυτό παρά μέσα από ένα πρόγραμμα - περνάς ευχάριστα τον χρόνο σου όσο παίζουν οι διαφημίσεις.

Γράψτε ένα πρόγραμμα το οποίο διαβάζει από την πρότυπη είσοδο: (1) τον αριθμό από πίτσες που έχουμε, (2) τον αριθμό κομματιών κάθε πίτσας, (3) τον αριθμό των ατόμων που έχουμε, (4) τον αριθμό των κομματιών που επιθυμεί το κάθε άτομο και τυπώνει στην πρότυπη έξοδο τον μέγιστο αριθμό ατόμων που μπορούμε να ικανοποιήσουμε **πλήρως** (αν κάποιο άτομο επιθυμεί 5 κομμάτια, πρέπει να φάει 5, αν απλά φάει 1 δεν αρκεί). Περιγράψτε την χρονική και χωρική πολυπλοκότητα του αλγορίθμου σας και εξηγήστε αν είναι η βέλτιστη (8/25 της βαθμολογίας). Παράδειγμα εκτέλεσης ακολουθεί:

```
$ ./pizza
Number of pizzas available: 3
Enter the number of slices for each pizza: 8 10 5
Number of people: 4
Number of slices desired by each person: 6 9 7 5
Max number of people that can be satisfied: 3
```

Υπόδειξη Για να ικανοποιήσετε όσο το δυνατόν περισσότερα άτομα, ποιους συμφέρει να εξυπηρετήσετε πρώτους; Σκεφτείτε μια άπληστη (greedy) στρατηγική που βασίζεται σε ταξινόμηση των επιθυμιών, και αιτιολογήστε γιατί καμία άλλη επιλογή δεν δίνει περισσότερα άτομα. Οι πίνακες έχουν μέγεθος που δίνεται στην είσοδο, και η πολυπλοκότητα καθορίζεται κυρίως από τον αλγόριθμο ταξινόμησης που θα διαλέξετε.

A25.14 · Γεμίζοντας με χρώμα

A25.14 · *Εξέταση Σεπτεμβρίου 2024, Θέμα 6* · ★★★ · programming · `exam-2024-sep-q6`

Γεμίζοντας με Χρώμα [25 Μονάδες]

Αν έχετε χρησιμοποιήσει κάποιο πρόγραμμα ζωγραφικής στον υπολογιστή (π.χ., Paint), ίσως έχετε κάνει χρήση του ”κουβά”, του εργαλείου δηλαδή που μας βοηθάει να γεμίζουμε γρήγορα κλειστές επιφάνειες χωρίς εμπόδια. Το Σχήμα 1 δείχνει ένα παράδειγμα:

```
 πριν          μετά
......        ......
..###.        ..###.
.#..#.  -->   .####.
.#X.#.        .####.
.###..        .###..
......        ......
```

Σχήμα 1: Οπτικοποίηση της διαδικασίας γεμίσματος με χρώμα. Στο σχήμα στα αριστερά επιλέξαμε την θέση (3, 2) - 3η γραμμή, 2η στήλη - για να την γεμίσουμε με χρώμα. Στο σχήμα στα δεξιά παρατηρούμε το αποτέλεσμα του γεμίσματός μας.

> Σημείωση: το Σχήμα 1 του πρωτοτύπου είναι πλέγμα 6×6 με χρωματιστά κελιά· εδώ αποδίδεται με # για χρωματισμένο κελί και X για το σημείο εκκίνησης.

Για τις ανάγκες της εφαρμογής μας κάνουμε τις εξής παραδοχές: (1) έχουμε μόνο δύο χρώματα που αναπαρίστανται με 0 και 1, (2) το πρόγραμμά μας γεμίζει την εικόνα μόνο με το χρώμα 1, (3) οι εικόνες που χειριζόμαστε είναι τετραγωνικές και (4) το χρώμα μπορεί μόνο να γεμίζει επιφάνειες που επικοινωνούν ελεύθερα με το αρχικό σημείο κάνοντας μόνο κινήσεις πάνω, κάτω, αριστερά, δεξιά (όχι διαγώνια). Εάν ένα κελί περιέχει ήδη το χρώμα 1, το κελί δεν θεωρείται ελεύθερο και δρα ως ”τοίχος” για το γέμισμά μας.

Γράψτε ένα πρόγραμμα το οποίο παίρνει ως όρισμα το όνομα του αρχείου που περιέχει την εικόνα και το αρχικό σημείο για γέμισμα και τυπώνει στην έξοδο την τελική εικόνα. Η πρώτη σειρά του αρχείου περιέχει την διάσταση της εικόνας, η δεύτερη σειρά τις συντεταγμένες του αρχικού σημείου και τέλος ακολουθεί η δισδιάστατη εικόνα. Παράδειγμα εκτέλεσης ακολουθεί:

```
$ cat picture.txt
6
3 2
000000
001110
010010
010010
011100
000000
$ ./fill picture.txt
000000
001110
011110
011110
011100
000000
```

Υπόδειξη Το γέμισμα είναι φυσικά αναδρομικό: χρωματίζετε ένα ελεύθερο κελί και ζητάτε το ίδιο για τους τέσσερις γείτονές του. Η βασική περίπτωση είναι κελί εκτός ορίων ή κελί που έχει ήδη 1, κάτι που εξασφαλίζει και τον τερματισμό. Προσέξτε ότι οι γραμμές του αρχείου είναι ψηφία χωρίς κενά και ότι οι συντεταγμένες του παραδείγματος (3, 2) πρέπει να αντιστοιχιστούν σωστά σε δείκτες πίνακα (δείτε το σχήμα).

A25.15 · Το Καλό το Μονοπάτι - path

A25.15 · *Εξέταση Σεπτεμβρίου 2025, Θέμα 5* · ★★★ · programming · `exam-2025-sep-q5`

Το Καλό το Μονοπάτι - path [25 Μονάδες]

Στην προσπάθειά του να τονώσει την τουριστική κίνηση, ο Ελληνικός Ορειβατικός Σύλλογος (ΕΟΣ) αποφάσισε να χρηματοδοτήσει ένα καινούριο έργο για την δημιουργία νέων, ήσσονος προσπαθείας μονοπατιών που να απευθύνονται σε αρχαρίους. Ο ΕΟΣ έχει ήδη πρόσβαση σε αναλυτικούς χάρτες με την δυσκολία της κάθε υποδιαδρομής. Το μόνο που λείπει για να προχωρήσει στην χάραξη των μονοπατιών, είναι ένα σύστημα το οποίο δοθέντος ενός χάρτη να υπολογίζει εύκολα και γρήγορα το μονοπάτι ελαχίστου κόστους. Ως συνήθως, το τμήμα μας προσφέρθηκε να βοηθήσει—αμισθί εννοείται—για την ταχεία διεκπεραίωση του έργου. Το Σχήμα 1 δείχνει έναν χάρτη-παράδειγμα από το φαράγγι του Βίκου:

[10]	13	10	11	10
[10]	10	10	18	10
[15]	42	42	42	42
[17]	**[5]**	**[5]**	**[5]**	14
10	10	10	**[10]**	**[10]**

Σχήμα 1: Ένας 5x5 χάρτης του ΕΟΣ με τα κόστη κάθε υποδιαδρομής να φαίνονται σε κάθε κελί του πλέγματος. Η διαδρομή ελαχίστου κόστους εικονίζεται σκιαγραφημένη. Παρατηρήστε πως η διαδρομή αποφεύγει τα κελιά υψηλού κόστους (τα 42 αναπαριστούν τον Βοϊδομάτη).

> Σημείωση: στο πρωτότυπο η διαδρομή είναι χρωματισμένα κελιά· εδώ σημειώνεται με `[ ]`.

Θεωρούμε πως όλοι οι χάρτες του ΕΟΣ είναι ένα τετραγωνικό πλέγμα και πως σε κάθε κελί του υπάρχει ένας ακέραιος αριθμός που αναπαριστά το κόστος της κάθε υποδιαδρομής. Για λόγους απλοποίησης, θεωρούμε πως σε κάθε χάρτη ο στόχος είναι να χαράξουμε ένα μονοπάτι από την άνω αριστερή γωνία του χάρτη μέχρι την κάτω δεξιά. Επίσης, κατά την διάρκεια της χάραξης επιτρέπονται μόνο μονοπάτια τα οποία κινούνται προς τα κάτω ή προς τα δεξιά (όχι προς τα πάνω / αριστερά / διαγώνια).

Γράψτε ένα πρόγραμμα το οποίο παίρνει ως όρισμα το όνομα του αρχείου που περιέχει τον χάρτη (διάσταση καθώς και τα κόστη κάθε υποδιαδρομής) και υπολογίζει το μονοπάτι ελαχίστου κόστους. Αν η διάσταση του πλέγματος είναι N, ποια η χρονική και χωρική πολυπλοκότητα της λύσης σας (8/25 της βαθμολογίας); Παράδειγμα εκτέλεσης ακολουθεί:

```
$ cat map.txt
5
10 13 10 11 10
10 10 10 18 10
```

```
15 42 42 42 42
17  5  5  5 14
10 10 10 10 10
$ ./path map.txt
Minimum Cost: 87
Minimum Cost Path: 10 -> 10 -> 15 -> 17 -> 5 -> 5 -> 5 -> 10 -> 10
```

Υπόδειξη Η εξαντλητική αναδρομή δοκιμάζει εκθετικά πολλά μονοπάτια· παρατηρήστε ότι το ελάχιστο κόστος για να φτάσετε σε ένα κελί εξαρτάται μόνο από το κελί από πάνω και το κελί από αριστερά, οπότε μπορείτε να γεμίσετε έναν πίνακα `N×N` γραμμή-γραμμή (δυναμικός προγραμματισμός). Για να τυπώσετε και το μονοπάτι, ξεκινήστε από την κάτω δεξιά γωνία και ακολουθήστε προς τα πίσω από πού ήρθε κάθε ελάχιστο. Το `N` διαβάζεται από το αρχείο, άρα ο πίνακας χρειάζεται δυναμική δέσμευση· προσέξτε και την πρώτη γραμμή/στήλη.

A25.16 · Περικύκλωση - encirclement

A25.16 · *Εξέταση Ιανουαρίου 2026, Θέμα 5* · ★★★ · programming · `exam-2026-jan-q5`

Περικύκλωση - encirclement [25 Μονάδες]

Σε πολλά παιχνίδια στρατηγικής η περικύκλωση του αντιπάλου αφαιρεί τις δυνατότητες κίνησής του και πολύ συχνά οδηγεί σε γρήγορη νίκη. Πότε είναι όμως περικυκλωμένο ένα πιόνι του αντιπάλου; Το Σχήμα 1 δείχνει ένα παράδειγμα:

	A	B	C	D	E	F	G	H	I	J
10		O	O	O						
9	O				O					
8	O		X		O					
7	O					O	O	O		
6		O	O	O			X		O	
5					O		X		O	
4			O		X	O	O	O	O	
3		O	X	O						
2			O			O				
1					O	X	O			

Σχήμα 1: Τα πιόνια του μαύρου (X) στις θέσεις C3, C8, G5, G6 είναι περικυκλωμένα, ενώ στις θέσεις F1, E4 είναι ελεύθερα καθώς έχουν διέξοδο προς τα άκρα του πλέγματος.

Θεωρούμε πως ο χάρτης του παιχνιδιού είναι ένα τετραγωνικό πλέγμα και πως κάθε κελί: (1) είτε περιέχει μαύρα πιόνια (X), (2) είτε περιέχει λευκά πιόνια (O), (3) είτε είναι κενό (.). Ένα πιόνι θεωρείται περικυκλωμένο όταν δεν υπάρχει σειρά κινήσεων πάνω, κάτω, αριστερά, δεξιά (δεν

επιτρέπονται διαγώνιες κινήσεις) που να επιτρέπει στο πιόνι να φτάσει στο άκρο του πλέγματος κινούμενο μόνο σε άδεια κελιά ή κελιά που περιέχουν το ίδιο χρώμα.

Γράψτε ένα πρόγραμμα το οποίο διαβάζει από την πρότυπη είσοδο (stdin) την διάσταση του πλέγματος και τα περιεχόμενα του χάρτη και τυπώνει στην πρότυπη έξοδο (stdout) για το κάθε μαύρο πιόνι αν είναι ελεύθερο ή όχι. Το πρόγραμμά σας πρέπει να είναι όσο πιο αποδοτικό γίνεται και να μπορεί να διαχειριστεί περιπτώσεις σφάλματος.

Ποια είναι η χρονική και η χωρική πολυπλοκότητα του αλγορίθμου σας; (8/25)

Παράδειγμα εκτέλεσης ακολουθεί:

```
$ cat map.txt
10
. . . . O X O . . .
. . O . . O . . . .
. O X O . . . . . .
. . O . X O O O O .
. . . . O . X . O .
. O O O . . X . O .
O . . . . O O O . .
O . X . O . . . . .
O . . . O . . . . .
. O O O . . . . . .
$ ./encirclement < map.txt
The following pawns are free: F1, E4
The following pawns are encircled: C3, G5, G6, C8
```

Υπόδειξη Αντί να ψάχνετε ξεχωριστά έξοδο για κάθε πιόνι, αντιστρέψτε το πρόβλημα: ξεκινήστε από όλα τα διαβατά κελιά (κενά ή X) του περιγράμματος και σημαδέψτε με flood fill (DFS ή BFS, με σημάδι ”επισκέφθηκα”) ό,τι φτάνετε κινούμενοι μόνο οριζόντια και κάθετα· όσα X δεν σημαδεύτηκαν είναι περικυκλωμένα. Συγκρίνετε το αρχείο με το Σχήμα 1: η πρώτη γραμμή του χάρτη στο αρχείο είναι η γραμμή 1 του σχήματος, και η έξοδος παρατάσσει τα πιόνια με αυτή τη σειρά. Ελέγξτε μη έγκυρη διάσταση, λάθος χαρακτήρες και λιγότερα κελιά από όσα περιμένετε.

A25.17 · Η Μεγαλύτερη Χωρητικότητα - capacity

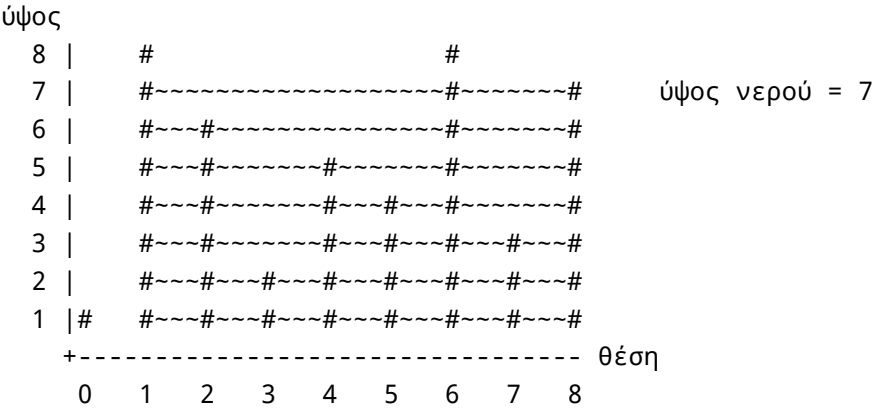
A25.17 · *Εξέταση Σεπτεμβρίου 2026, Θέμα 5* · ★★★ · programming · `exam-2026-sep-q5`

Η Μεγαλύτερη Χωρητικότητα - capacity [25 Μονάδες]

Οι πολιτικοί μηχανικοί του ΕΜΠ χρειάζονται τη βοήθειά μας! Θέλουν να κατασκευάσουν μια σειρά από φράγματα και, προτού ξεκινήσουν τις εργασίες, πρέπει να υπολογίσουν την *μέγιστη*

δυνατή ποσότητα νερού που μπορεί να αποθηκευτεί ανάμεσα στις διάφορες βουνοκορφές. Για λόγους απλοποίησης, θεωρούμε ότι έχουμε N κορυφές τοποθετημένες σε ίσες αποστάσεις μεταξύ τους. Η κάθε κορυφή i βρίσκεται στη θέση i και έχει ύψος h_i. Επιλέγοντας δύο κορυφές i και j, με $i < j$, σχηματίζεται μαζί με το έδαφος μια ορθογώνια δεξαμενή, όπως φαίνεται στο Σχήμα 1 (δεν μας πειράζει αν παρεμβάλλονται άλλες βουνοκορφές ενδιάμεσα).

Η στάθμη του νερού δεν μπορεί φυσικά να ξεπεράσει την χαμηλότερη από τις κορυφές που σχηματίζουν την ορθογώνια δεξαμενή. Για τις ανάγκες του προβλήματος θεωρούμε τη χωρητικότητα ως το εμβαδόν του ορθογωνίου που σχηματίζεται από την δεξαμενή.



Σχήμα 1: Παράδειγμα για μια σειρά από κορυφές: $h = [1, 8, 6, 2, 5, 4, 8, 3, 7]$. Οι κορυφές στις θέσεις 1 και 8 σχηματίζουν τη δεξαμενή μέγιστης χωρητικότητας. Η απόστασή τους είναι $8 - 1 = 7$ και το ύψος του νερού είναι 7, επομένως η χωρητικότητα είναι $7 \cdot 7 = 49$.

> Σημείωση: στο πρωτότυπο το Σχήμα 1 είναι γράφημα με ράβδους και γραμμοσκιασμένη δεξαμενή· εδώ οι κορυφές σημειώνονται με # και το νερό με ~.

Γράψτε ένα πρόγραμμα το οποίο παίρνει από την πρότυπη είσοδο στην πρώτη γραμμή το πλήθος N των κορυφών και στην δεύτερη γραμμή τα N ακέραια ύψη των κορυφών $h_0, h_1, \ldots, h_{N-1}$ και υπολογίζει τη **μέγιστη δυνατή χωρητικότητα** που μπορεί να σχηματιστεί από μία δεξαμενή καθώς και τις κορυφές ανάμεσα στις οποίες θα εμφανιστεί αυτή η χωρητικότητα. Ο αλγόριθμός σας θέλουμε να είναι σωστός αλλά και αποδοτικός.

Ποια είναι η χρονική και η χωρική πολυπλοκότητα της λύσης σας; Αιτιολογήστε σύντομα την απάντησή σας (8/25 της βαθμολογίας).

Παράδειγμα εκτέλεσης ακολουθεί:

```
$ cat mountains.txt
9
1 8 6 2 5 4 8 3 7

$ ./capacity < mountains.txt
Maximum Capacity between mountains at positions 1 and 8: (8-1) * 7 = 49
```

Υπόδειξη Η χωρητικότητα του ζεύγους (i, j) είναι $(j - i) \cdot \min(h_i, h_j)$· ο έλεγχος όλων των ζευγών είναι σωστός αλλά $O(N^2)$, οπότε ξεκινήστε από αυτόν και μετά ψάξτε κάτι γρηγορότερο. Σκεφτείτε δύο δείκτες θέσης, έναν σε κάθε άκρο, που κινούνται ο ένας προς τον άλλον: ποιον από τους δύο δεν έχει νόημα να κρατήσετε, αφού κάθε μετακίνηση μικραίνει το πλάτος; Θυμηθείτε να κρατάτε και τις θέσεις του καλύτερου ζεύγους, να ελέγχετε την τιμή επιστροφής της `scanf` και να δεσμεύσετε τον πίνακα ανάλογα με το N.

Παράρτημα Α: How to Make? (προσκεκλημένη διάλεξη)

Ζέσταμα: από τις διαφάνειες (A26.1–A26.5)

A26.1 · Τα στάδια του C build process

A26.1 · *How to Make? (προσκεκλημένη διάλεξη), διαφάνεια 5* · ★☆☆ · short-answer · `slides-lecmake-build-pipeline`

«Say 'Aye' if you know what this is!» Η διαφάνεια δείχνει το παρακάτω διάγραμμα:

```
Translation Unit (.c) -> Preprocessor (cpp) -> Preprocessed Translation Unit (.c)
  -> Compiler (gcc) -> Object File (.o) -> Linker (ld) -> Executable
                          Object File (.o) -----^
                    Dynamic Library (.so) -----^
```

Εξηγήστε τι κάνει το καθένα από τα τρία στάδια (preprocessor, compiler, linker) και τι παράγει. Γιατί ο linker δέχεται και άλλα object files και δυναμικές βιβλιοθήκες;

Υπόδειξη Σκεφτείτε τι απογίνονται οι οδηγίες `#include` και `#define`, σε ποιο στάδιο ο κώδικας γίνεται γλώσσα μηχανής, και πού βρίσκεται η υλοποίηση της `printf` που καλεί το πρόγραμμά σας.

A26.2 · Script ή recompile με το χέρι;

A26.2 · *How to Make? (προσκεκλημένη διάλεξη), διαφάνειες 17-21* · ★☆☆ · short-answer · `slides-lecmake-forgot-recompile`

Χτίζουμε ένα project με ένα bash script, `build.sh`, που τρέχει ένα `gcc -c` για κάθε αρχείο `.c` και ένα `gcc -o` για τη σύνδεση.

1. Κάθε φορά που τρέχουμε το script, ξαναμεταγλωττίζεται όλο το πρόγραμμα από την αρχή, ανεξάρτητα από το αν άλλαξε κάποιο αρχείο. «Και;» Σκεφτείτε την GNU C Library: 1.524.568 γραμμές κώδικα σε πάνω από 14.000 αρχεία. Αν θέλαμε απλά να βάλουμε σχόλια σε ένα αρχείο, ακούγεται για καλή χρήση του χρόνου μας;

2. Εναλλακτική: «Θα κάνω recompile με το χέρι τα αρχεία που αλλάζω, θα κάνω link με ένα bash script, κι αν θέλω να κάνω compile όλο το project, θα έχω ένα άλλο script για αυτό.» Κι αν ξεχάσεις να κάνεις κάτι recompile; Τι θα δείτε όταν τρέξετε το πρόγραμμα;
3. Τι πληροφορία χρειάζεται ένα εργαλείο για να αποφασίσει μόνο του ποια αρχεία να ξαναμεταγλωττίσει;

Υπόδειξη Για το 2, θυμηθείτε την ιστορία του Steve Johnson που αναφέρει ο Stuart Feldman. Για το 3, σκεφτείτε τι εξαρτάται από τι (ένα `.o` από ποια `.c` και `.h`) και πώς καταλαβαίνουμε ότι ένα αρχείο άλλαξε μετά από ένα άλλο.

A26.3 · Διαφορετικά flags ανά αρχείο

A26.3 · *How to Make? (προσκεκλημένη διάλεξη), διαφάνειες 10-14* · ★☆☆ · tooling · `slides-lecmake-per-file-flags`

Ένα project έχει δύο αρχεία, το `main.c` και το `primes.c`, και μέχρι τώρα χτίζεται με

```
$ gcc -Wall -Wextra -Werror -std=c99 -pedantic -o main main.c primes.c
```

Θέλω το `primes.c` translation unit να μην έχει το standard enforcement (`-std=c99`), επειδή θέλω να χρησιμοποιήσω την `reallocarray`, που είναι non-standard. Το `main.c` πρέπει να κρατήσει όλα τα flags. Πώς θα το κάνω αυτό; Γράψτε τις εντολές.

Υπόδειξη Με μία εντολή `gcc` όλα τα αρχεία παίρνουν τα ίδια flags. Σπάστε το compilation σε βήματα: θυμηθείτε ποιο flag του `gcc` σταματά πριν από τη σύνδεση και τι αρχείο παράγει.

A26.4 · Compiler error ή linking error; (math.h και libm.so)

A26.4 · *How to Make? (προσκεκλημένη διάλεξη), διαφάνειες 6-8* · ★☆☆ · debug · `slides-lecmake-sqrt-errors`

Το πρόγραμμα:

```c
#include <stdio.h>
// does-not-compile
int main(void) {
    printf("%.2f\n", sqrt(3));
    return 0;
}
```

```
$ gcc -o main main.c
main.c: In function 'main':
main.c:4:20: error: implicit declaration of function 'sqrt'
```

```
[-Wimplicit-function-declaration]
main.c:2:1: note: include '<math.h>' or provide a declaration of 'sqrt'
```

1. Τι έγινε εδώ; Τι είδους error είναι αυτό;
2. Προσθέτουμε `#include <math.h>` και ξανατρέχουμε:

   ```
   $ gcc -o main main.c
   /usr/bin/ld: /tmp/ccaw9D8e.o: in function `main':
   main.c:(.text+0x1f): undefined reference to `sqrt'
   collect2: error: ld returned 1 exit status
   ```

   Τι είδους error είναι αυτό τώρα; Παρατηρήστε ότι, σε αντίθεση με τον compiler, το μήνυμα δεν είναι και πολύ βοηθητικό. Έχει κανείς ιδέα γιατί;
3. Πώς το διορθώνουμε ώστε το `./main` να τυπώσει `1.73`;

Υπόδειξη Ποιο πρόγραμμα βγάζει κάθε μήνυμα (δείτε τη λέξη `ld`); Ένα αρχείο `.h` δίνει δηλώσεις, όχι υλοποιήσεις: σκεφτείτε σε ποιο object file ή βιβλιοθήκη βρίσκεται η `sqrt` και πώς το λέτε στον linker.

A26.5 · Ένα Makefile 2-3 γραμμών

A26.5 · *How to Make? (προσκεκλημένη διάλεξη), διαφάνειες 31-39* · ★★☆ · tooling · `slides-lecmake-short-makefile`

Το παρακάτω Makefile χτίζει το project `main` (αρχεία `main.c`, `primes.c`, `primes.h`):

```
main: main.o primes.o
    gcc -o main main.o primes.o

main.o: main.c primes.h
    gcc -Wall -Wextra -Werror -pedantic -std=c99 -c main.c

primes.o: primes.c primes.h
    gcc -Wall -Wextra -Werror -pedantic -c primes.c
```

Λειτουργεί; Ναι. Είναι καλό; Όχι.

1. Ξαναγράψτε το με macros (`CC`, `CFLAGS`), ώστε κάποιος που χρησιμοποιεί `clang` να αλλάζει μία μόνο γραμμή.
2. Χρησιμοποιήστε automatic variables και ένα wildcard rule, ώστε να μην επαναλαμβάνονται τα rules των object files.
3. Χρησιμοποιώντας τα built-in rules του Make, κάντε το Makefile 2-3 γραμμές.

Ελέγξτε ότι μετά από `touch main.c` το `make` ξαναφτιάχνει μόνο το `main.o` και το `main`.

Υπόδειξη Τα `$@`, `$^` και `$<` είναι το target, όλα τα prerequisites και το πρώτο prerequisite. Τα built-in rules χρησιμοποιούν τα macros `CC` και `CFLAGS`· ένα rule χωρίς recipe απλώς προσθέτει prerequisites. Το `-std=c99` μόνο για το `main.o` χρειάζεται ανάθεση μέσα σε rule.
