ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Τμήμα Πληροφορικής και Τηλεπικοινωνιών • Εισαγωγή στον Προγραμματισμό

Περιεχόμενα

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.

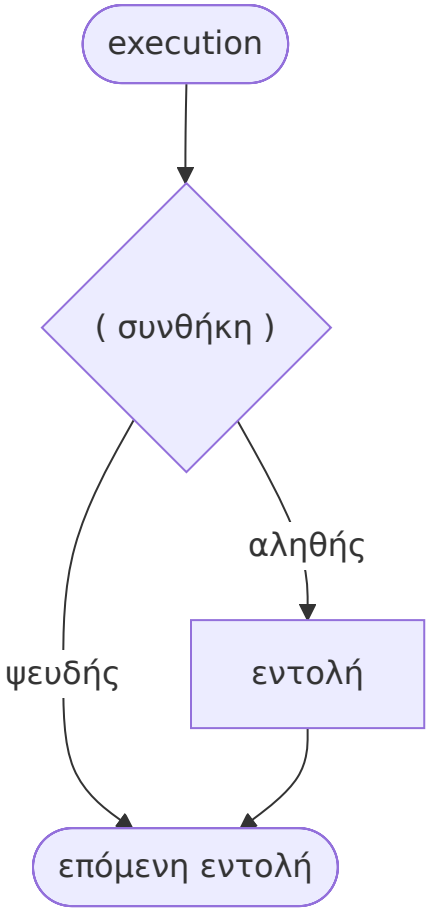
Μια συνθήκη στην C είναι απλώς μια έκφραση (Κεφάλαιο 5): τιμή **μηδέν** σημαίνει **ψευδής**, **οποιαδήποτε άλλη** τιμή σημαίνει **αληθής**. Γι' αυτό γράφουμε `while (1)` ή `if (n)`.

§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`.

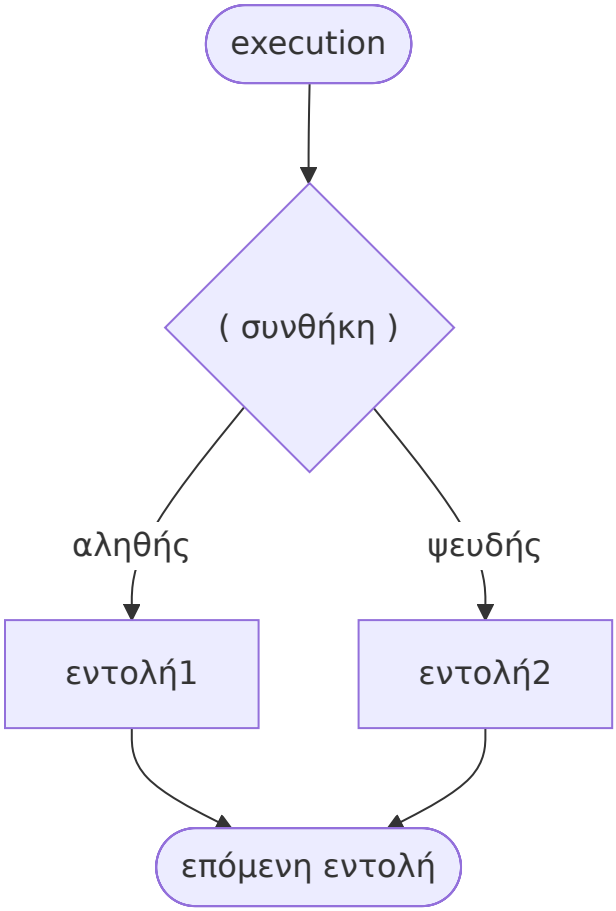


Σχήμα: διάγραμμα ροής της `if`.

§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`.

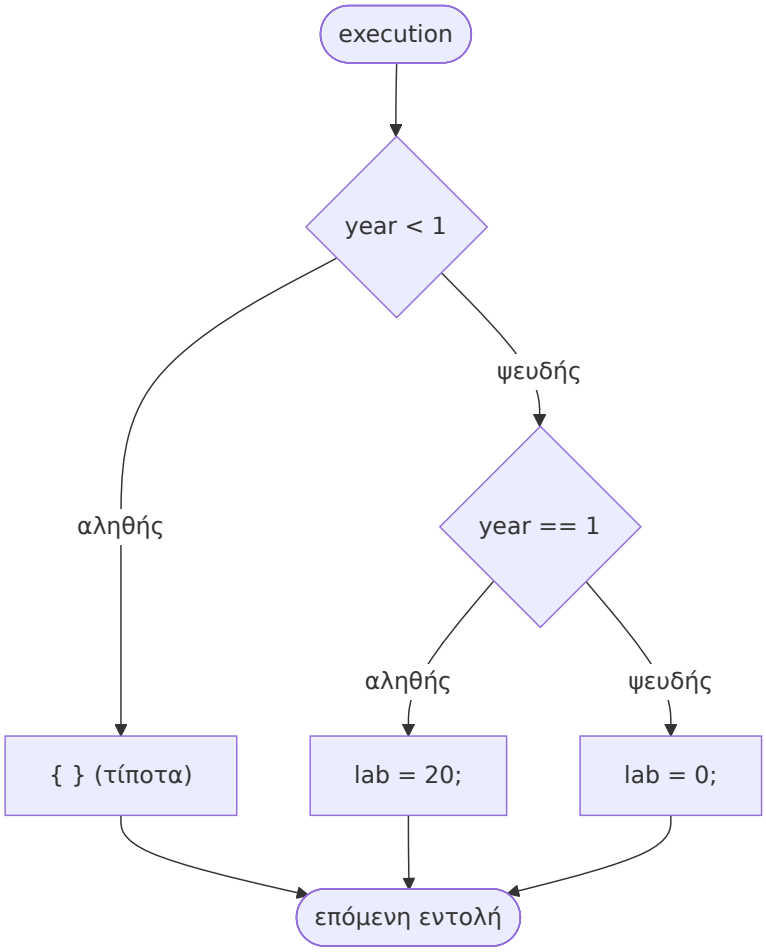
Το `else` δεν είναι απαραίτητο (το ίδιο αποτέλεσμα βγαίνει με μια αρχική ανάθεση και μια απλή `if`), αλλά δείχνει καθαρά ότι οι δύο περιπτώσεις αποκλείονται. Για απλές επιλογές τιμής υπάρχει και ο τελεστής συνθήκης `? :` του Κεφαλαίου 5. Βλ. το παράδειγμα Μέγιστο δύο αριθμών.

§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;
```

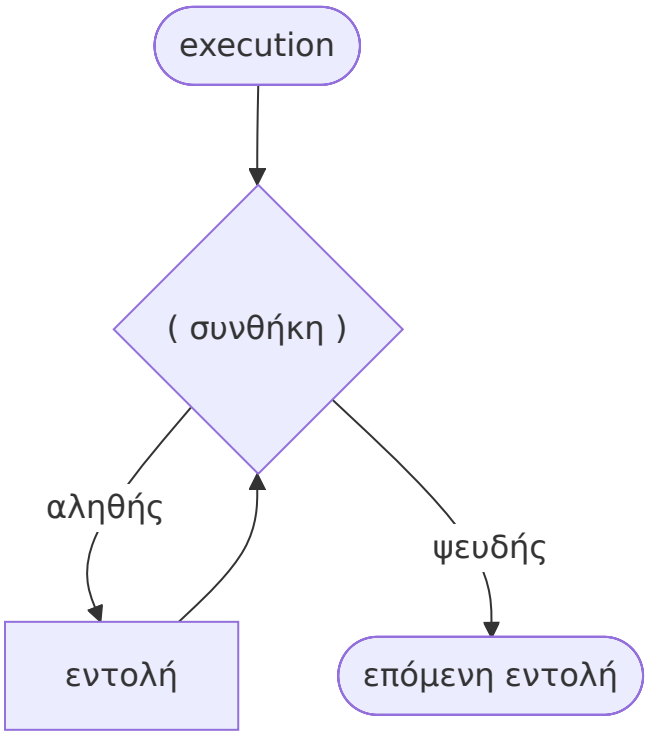
το `else` ανήκει στην εσωτερική `if (year == 1)`, παρόλο που η στοίχιση λέει το αντίθετο. Αυτό είναι το πρόβλημα του **dangling else**. Η λύση των διαφανειών: **αν δεν είμαστε σίγουροι, χρησιμοποιούμε αγκύλες `{ }` για να υποδείξουμε τα όρια των σύνθετων εντολών**. Ο gcc με `-Wall` προειδοποιεί: `suggest 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.*

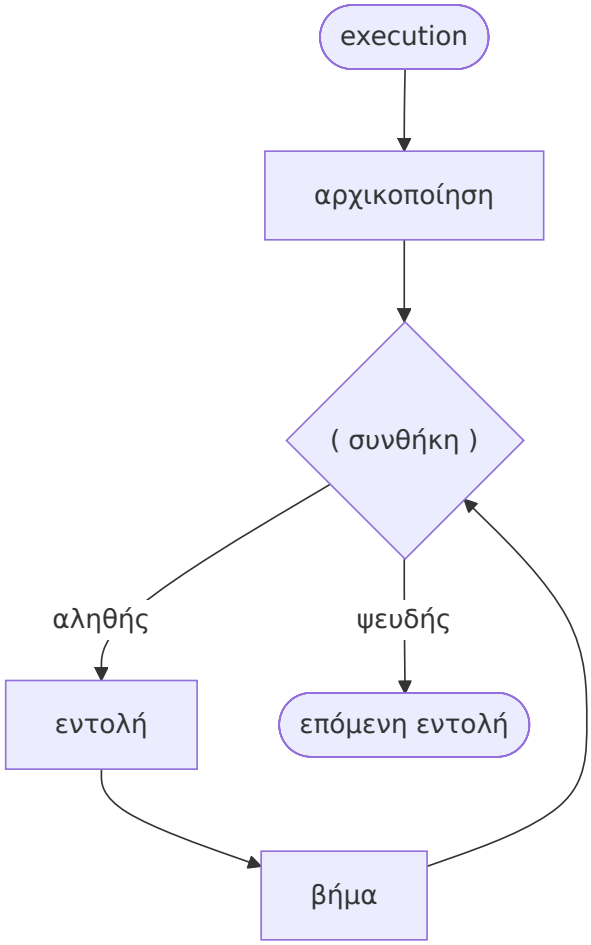
§6.9 Η εντολή for

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`.

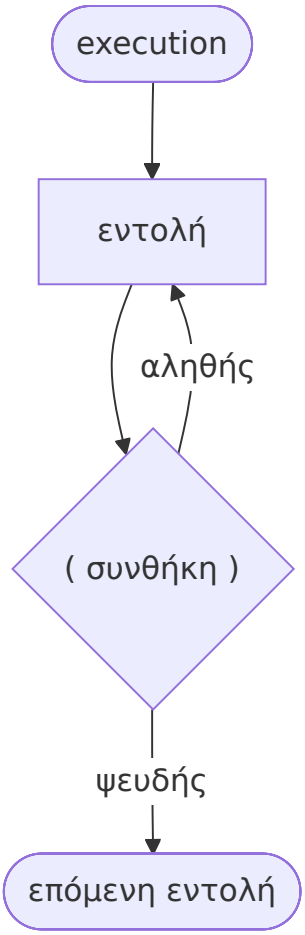
Και τα τρία μέρη είναι προαιρετικά, αλλά τα δύο `;` μένουν πάντα. Συνθήκη που λείπει θεωρείται πάντα αληθής, άρα το `for (;;)` είναι η κλασική γραφή του ατέρμονα βρόχου. Ο βρόχος `for (i = 0; i < N; i++)` εκτελείται ακριβώς `N` φορές, για `i = 0, 1, …, N-1`.

§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	χωριστά	στην επικεφαλίδα	χωριστά
`;` μετά τη συνθήκη	λάθος (κενό σώμα)	λάθος (κενό σώμα)	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;

```c
#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`.

```c
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 `;`

Ελληνικά	English	Σύντομος ορισμός
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;[1]
- E6.2 Γιατί ένα block `{ … }` μπορεί να σταθεί ως σώμα μιας `if`, ενώ δύο σκέτες εντολές όχι;[2]
- E6.3 Σε ποια `if` ανήκει ένα `else` όταν υπάρχουν δύο υποψήφιες και δεν υπάρχουν αγκύλες;[3]

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

[2] 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`.

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

- **E6.4** Ποια είναι η διαφορά ανάμεσα σε `while` και `do-while`; Δώστε μια περίπτωση όπου δίνουν διαφορετικό αποτέλεσμα.[4]
- **E6.5** Γράψτε έναν βρόχο `while` ισοδύναμο με το `for (i = 0; i < 100; i++) printf("Hello world\n");`.[5]
- **E6.6** Πόσες φορές εκτελείται το σώμα του `for (i = 0; i <= N; i++);`[6]

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

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`

[4] 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`.

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

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

- 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`