ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν
Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Εισαγωγή στον Προγραμματισμό

Εργασία #1

Νοέμβριος 2023

Στόχος της εργασίας είναι να γράψουμε μερικά ακόμα προγράμματα σε C με χρήση βασικών τύπων και συναρτήσεων. Συγκεκριμένα, στοχεύουμε σε εμπειρία με τα ακόλουθα:

1. Χρήση αριθμών κινητής υποδιαστολής στην C
2. Χρήση συναρτήσεων
3. Χρήση αναδρομής
4. Χρήση δομών επανάληψης με διαφορετικά κριτήρια τερματισμού
5. Χρήση αλγοριθμικών απλοποιήσεων

Υποβολή Εργασίας. Όλες οι υποβολές των εργασιών θα γίνουν μέσω GitHub και συγκεκριμένα στο github.com/progintro [1]. Προκειμένου να ξεκινήσεις, μπορείς να δεχτείς την άσκηση με αυτήν την: πρόσκληση [2].

1 Η Μέθοδος Newton-Raphson (50 Μονάδες)

Η εύρεση ριζών σε πολυώνυμα 2ου βαθμού είναι κάτι που όλοι λίγο-πολύ περάσαμε στο σχολείο συνοδευόμενο από ευχάριστες ή τραυματικές αναμνήσεις [10]. Ο υπολογισμός της διακρίνουσας ($\Delta = \beta^2 - 4\alpha\gamma$) μας εντυπώθηκε στην μνήμη (για μερικούς από εμάς μέχρι τις εξετάσεις και μετά διεγράφη) και η χρησιμότητά της ήταν ότι μας επέτρεπε να υπολογίσουμε τις ρίζες σε κλειστή μορφή για οποιοδήποτε πολυώνυμο 2ου βαθμού. Παρόλα αυτά, το να βρίσκεις ρίζες σε πολυώνυμα είναι δύσκολη υπόθεση. Η κλειστή μορφή για πολυώνυμα 3ου βαθμού είναι σαφώς πιο δύσκολη (για μια γρήγορη αναδρομή που περιέχει προδοσία, εγωισμούς, ζήλια, γεωμετρία και φανταστικούς αριθμούς δείτε εδώ [5]). Για πολυώνυμα 5ου βαθμού και πάνω, είναι ακόμα χειρότερα - καθώς δεν υπάρχει κλειστή μορφή υπολογισμού [3]!

Δυστυχώς (ή ευτυχώς ανάλογα με την οπτική), πολυώνυμα 5ου βαθμού και πάνω υπάρχουν στην φυσική, την μηχανική, και άλλα πεδία και πολύ συχνά πρέπει να υπολογίσουμε τις ρίζες τους. Η εύρεση ριζών σε αυτά τα πολυώνυμα ήταν κάτι άπιαστο για την ανθρωπότητα για το μεγαλύτερο μέρος της ιστορίας μας. Σήμερα, μπορούμε να κάνουμε κάτι για να λύσουμε τέτοια προβλήματα; Ευτυχώς ναι! Η απάντηση είναι στις προσεγγιστικές μεθόδους που προσφέρει η αριθμητική ανάλυση.

Η μέθοδος Newton-Raphson, γνωστή και πιο απλά ως μέθοδος Newton, προς τιμήν του γνωστού μας Νεύτωνα [6] 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:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

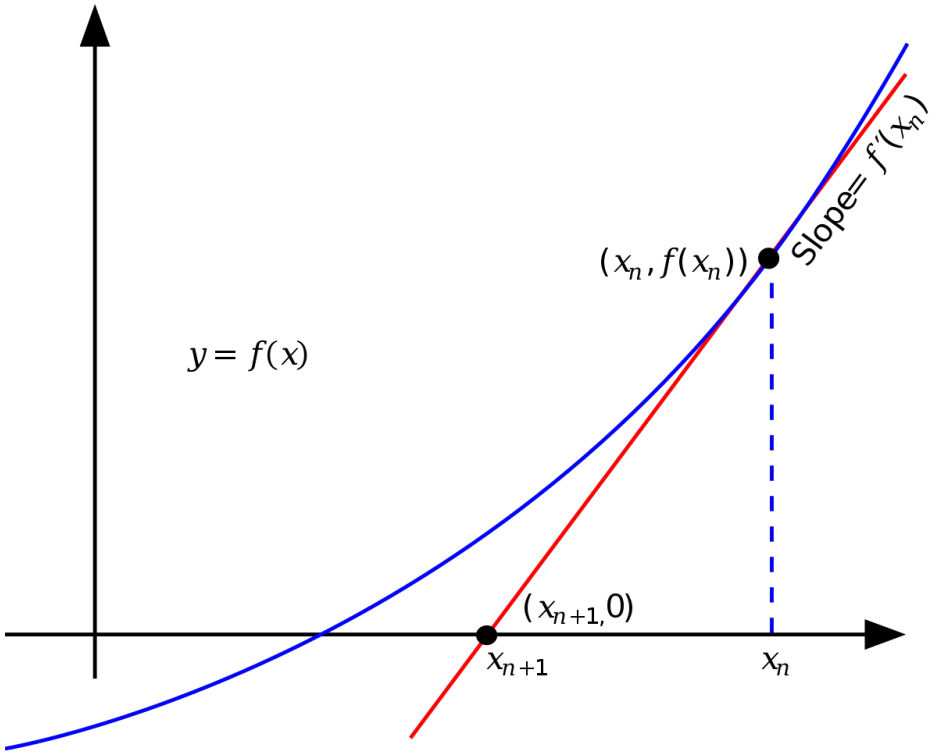
όπου x_{n+1} είναι η προσεγγιστική τιμή μιας ρίζας της συνάρτησης $f(x)$ μετά από $n+1$ επαναλήψεις. Το Σχήμα 1 παρουσιάζει ένα παράδειγμα το πως η μέθοδος συγκλίνει προς ρίζες της συνάρτησης με κάθε επανάληψη.

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. Σκεφτείτε πόσο χρόνο θα είχαμε γλυτώσει στο σχολείο αν είχαμε πρόσβαση σε ένα τέτοιο πρόγραμμα!

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
```



Σχήμα 1: Η μέθοδος Newton-Raphson ξεκινάει από ένα σημείο της καμπύλης x_n και χρησιμοποιεί την κλίση της καμπύλης προκειμένου στην επόμενη επανάληψη το x_{n+1} 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;

- 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.

2 Κατοπτρικά Πρώτα Τετράγωνα (50 Μονάδες)

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) [8], 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. Ενας φυσικός αριθμός μεγαλύτερος του 1 ονομάζεται *πρώτος (prime)* [9] 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:

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97$$

Οι πρώτοι αριθμοί είναι ιδιαίτερα χρήσιμοι στην θεωρία αριθμών και την κρυπτογραφία (χάρη σε αυτούς διατηρούμε τις επικοινωνίες μας ασφαλείς [12]). Μαθηματικοί στο παρελθόν έχουν ασχοληθεί με πολλά γνωστά προβλήματα όπως η κατανομή τους [11]. Μέχρι σήμερα, δεν έχει βρεθεί μια συνάρτηση που να παράγει τους πρώτους αριθμούς αποδοτικά [4].

Ορισμός 3. Ένας φυσικός αριθμός λέγεται *παλινδρομικός (palindromic) [7]* 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 ακέραοι 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.

3 Άψογα Τετράγωνα (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. 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 ακέραοι 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.

Αναφορές

[1] Οργανισμός για το μάθημα (GitHub progintro) . https://github.com/progintro.

[2] Πρόσκληση για Εργασία 1 . https://classroom.github.com/a/yAIvSiOZ.

[3] Abel-Ruffini Theorem. https://en.wikipedia.org/wiki/Abel%E2%80%93Ruffini_theorem.

[4] Formulas for Primes. https://en.wikipedia.org/wiki/Formula_for_primes.

[5] How Imaginary Numbers Were Invented - Depressed Cubic (αν ξεκινήσατε την άσκηση μια μέρα (ή ώρα) πριν την υποβολή συνιστώ να μην πατήσετε το λινκ). https://www.youtube.com/watch?v=cUzklzVXJwo.

[6] Newton's Method. https://en.wikipedia.org/wiki/Newton%27s_method.

[7] Palindromic Number. https://en.wikipedia.org/wiki/Palindromic_number.

[8] Perfect Square. https://en.wikipedia.org/wiki/Square_number.

[9] Prime Numbers. https://en.wikipedia.org/wiki/Prime_number.

[10] Quadratic Equation. https://en.wikipedia.org/wiki/Quadratic_equation.

[11] Riemann Hypothesis. https://en.wikipedia.org/wiki/Riemann_hypothesis.

[12] Rivest–Shamir–Adleman (RSA). https://en.wikipedia.org/wiki/RSA_(cryptosystem).