ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

# Εισαγωγή στον Προγραμματισμό

## Εργασία #3

## Ιανουάριος 2024

Στόχος της τρίτης και τελευταίας εργασίας είναι να υλοποιήσουμε μερικά πιο πολύπλοκα προγράμματα και να ολοκληρώσουμε την εισαγωγή μας στην γλώσσα C. Συγκεκριμένα, στοχεύουμε σε εμπειρία με τα ακόλουθα:

1. Χρήση δομών στην C
2. Αναζήτηση σε χώρο καταστάσεων
3. Γραφή μεγαλύτερων προγραμμάτων

**Υποβολή Εργασίας**. Όλες οι υποβολές των εργασιών θα γίνουν μέσω GitHub και συγκεκριμένα στο github.com/progintro [2]. Προκειμένου να ξεκινήσεις, μπορείς να δεχτείς την άσκηση με αυτήν την: πρόσκληση [3]. Σε αντίθεση με τις προηγούμενες εργασίες, αυτή είναι **προαιρετικά ομαδική με ομάδες μέχρι 2 άτομα**. Εάν θέλετε να δουλέψετε ως ομάδα, πρέπει ένας από εσάς να αποδεχτεί την πρόσκληση και να δημιουργήσει την ομάδα σας **με συγκεκριμένο όνομα**. Αφού η ομάδα δημιουργηθεί ο/η συνεργάτης σας μπορεί να αποδεχτεί την πρόσκληση και να επιλέξει να προστεθεί στην ομάδα σας. **Προσοχή: μην επιλέξετε λάθος ομάδα, βεβαιωθείτε ότι κάνατε join την σωστή.** Αν επιλέξετε να εργαστείτε ως ομάδα, το repository για την εργασία θα είναι κοινό ανάμεσά σας και θα μπορέσετε να συνεργαστείτε σαν επαγγελματίες software engineers - προσοχή στα conflicts! Τέλος, για να είναι ξεκάθαρο ποια άτομα συνεργάστηκαν για την εργασία, **κάθε repository πρέπει να έχει ένα AUTHORS αρχείο** το οποίο θα περιέχει τα στοιχεία σας στην ακόλουθη μορφή:

```
$ cat AUTHORS
sdi2300998,barbouni-2005,ΘΑΝΟΣ ΜΠΑΡΜΠΟΥΝΗΣ
sdi2300999,kolios-2005,ΚΩΣΤΑΣ ΚΟΛΙΟΣ
```

δηλαδή μία γραμμή για κάθε άτομο, με πρώτο το sdi σας, μετά το github username και τέλος το όνομά σας. **Υποβολές χωρίς σωστό AUTHORS αρχείο δεν θα εξεταστούν**. Αυτό ισχύει και για ατομικές υποβολές.

Σχήμα 1: Μια σκούπα Roomba έχει κολλήσει στην γωνία ενός δωματίου προσπαθώντας να πάει στο άλλο δωμάτιο. Η Zoomba μας δεν θα έχει τέτοια προβλήματα.

# 1 Zoomba (100 Μονάδες)

Το να σκουπίζεις το δωμάτιο σου είναι μερικές φορές βαρετό. Τόσο βαρετό, που αναγκαστήκαμε να εφεύρουμε ρομποτικές σκούπες που κάνουν την δουλειά για εμάς. Οι πιο γνωστές από αυτές τις σκούπες λέγονται Roomba [9], έχουν πάνω από το 45% της αγοράς, και είναι πανάκριβες, κοστίζουν από εκατοντάδες μέχρι χιλιάδες ευρώ ανάλογα με το μοντέλο. Παρόλα αυτά, έχουν ένα σωρό προβλήματα. Για παράδειγμα, όταν πάνε να καθαρίσουν ένα δωμάτιο, αντί να πάνε κατευθείαν στο σημείο που χρειάζεται καθάρισμα, κάνουν τυχαίες βόλτες στον χώρο [10] προκειμένου να "καθαρίσουν παντού". Αυτές οι τυχαίες βόλτες μπορούν να οδηγήσουν την σκούπα στο να κολλήσει (Σχήμα 1) ή να καταναλώσει όλη την μπαταρία της πολύ γρήγορα. Οπότε αντί για λύση στο σκούπισμα, τελικά καταλήγεις να κυνηγάς την σκούπα από πίσω για να την ξεκολλάς κάθε τρεις και λίγο, να την επαναφορτίζεις και τελικά σκουπίζεις μόνος σου τα ψίχουλα τα οποία δεν έφτασε ποτέ.

Απελπισμένοι από τις υπάρχουσες λύσεις, αποφασίσαμε να χτυπήσουμε την αγορά των ρομποτικών σκουπών δημιουργώντας την δική μας ρομποτική σκούπα ονόματι Zoomba. Zoom επειδή είναι σούπερ γρήγορη (super fast - zoom!), αποδοτική σε ενέργεια (energy efficient - μπα;) και επειδή επιτέλους σου λύνει το πρόβλημα του σκουπίσματος αφήνοντάς σε να ασχοληθείς με άλλες δραστηριότητες (παραδείγματος χάρη με γυμναστική Zumba). Το όνομα και το υλικό για το marketing ίσως χρειάζεται λίγη δουλειά ακόμα αλλά αυτό είναι θετικό καθώς δεν έχουμε λύση για

το βασικό μας πρόβλημα: το λογισμικό που θα περάσουμε στην σκούπα μας που πρέπει να βρίσκει τον πιο αποδοτικό τρόπο να πηγαίνει στο σημείο του δωματίου που απαιτείται καθαρισμός δεν έχει υλοποιηθεί!

Ευτυχώς οι hardware-άδες της ομάδας μας κατάφεραν να φτιάξουν ένα ραντάρ το οποίο σκανάρει το δωμάτιο αυτόματα, δημιουργεί τον "χάρτη" του δωματίου 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.

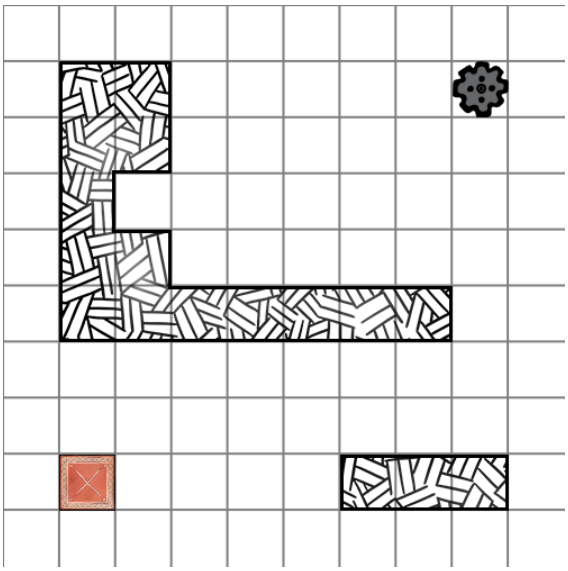
## Τεχνικές Προδιαγραφές

- 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:

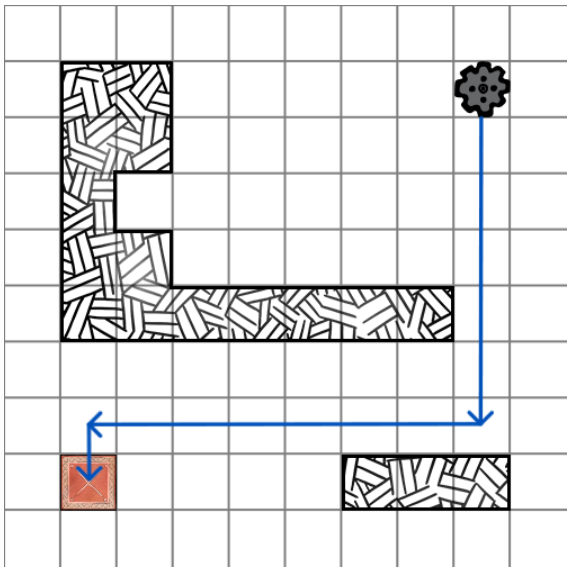
  ```
  ROOM_DIMENSION_N
  ZOOMBA_X ZOOMBA_Y ZOOMBA_TARGET_X ZOOMBA_TARGET_Y
  ROOM_ENCODED_AS_1s_AND_0s
  ```

  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
  ```



Αρχική θέση που δόθηκε στην Zoomba. Με το κόκκινο X δείχνουμε την περιοχή που χρειάζεται καθάρισμα ενώ με το στρογγυλό γρανάζι δείχνουμε την αρχική θέση της Zoomba.



Μια από τις λύσεις που θέλουμε να επιλέξει η Zoomba (αποτελείται από 14 βήματα: DDDDDDLLLLLLLD). Παρατηρήστε ότι υπάρχουν πάνω από μία βέλτιστες λύσεις.

Σχήμα 2: Ο αρχικός χάρτης που δόθηκε στην zoomba και η διαδρομή που επέλεξε να διατρέξει.

```
0110000000
0100000000
0110000000
0111111100
0000000000
0000000000
0000001110
0000000000
```

- 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:

  ```
  DDDDDDLLLLLLLD
  ```

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
  ```

- README Filepath: zoomba/README.md
- C Filepath: zoomba/src/zoomba.c
- **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 απότι 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
$ 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 [1] 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 και συγκεκριμένα: η πρώτη υποβολή θα λάβει +100%, η δεύτερη +70% και η τρίτη +40%. Σε περίπτωση ισοβαθμιών, θα ελέγξουμε την τεκμηρίωση των εργασιών. Αν δεν καταστεί δυνατό να λύσουμε τις ισοβαθμίες, μπορεί να επιλέξουμε περισσότερες υποβολές.

# 2 Η Newton-Raphson Ξαναχτυπά! (Bonus 50 Μονάδες)

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. Και αυτή είναι η πραγματικότητα που ξέραμε όλοι μέχρι που ... οι μαθηματικοί ανακάλυψαν τους φανταστικούς αριθμούς [6]! Οι φανταστικοί αριθμοί έχουν καταπληκτικές ιδιότητες, για παράδειγμα η φανταστική μονάδα $i$ όταν υψωθεί στο τετράγωνο μας κάνει $-1$ ($i^2 = -1$) και επομένως 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) [4]. Που χρειάζονται οι μιγαδικοί αριθμοί; Έχουν πάμπολλες εφαρμογές, από την ηλεκτρονική και την μηχανική, μέχρι τα σήματα και συστήματα και τις τηλεπικοινωνίες.

Προκειμένου να λύσουμε αυτήν την άσκηση θα χρειαστεί να μάθουμε τι είναι οι μιγαδικοί αριθμοί και στην συνέχεια θα δοκιμάσουμε να βρούμε λύσεις σε πολυώνυμα στο μιγαδικό επίπεδο, δηλαδή αντί τα πολυώνυμά μας να δέχονται πραγματικούς αριθμούς ως είσοδο, θα δέχονται μιγαδικούς. Η μέθοδος Newton παραμένει η ίδια [8], 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 [4]):

$$(a + bi) + (c + di) = (a + c) + (b + d)i$$
$$(a + bi) \cdot (c + di) = (ac - bd) + (ad + bc)i$$
$$\frac{a + bi}{c + di} = \frac{ac + bd}{c^2 + d^2} + \frac{bc - ad}{c^2 + d^2}i$$

$$|a + bi| = \sqrt{a^2 + b^2}$$

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 $z0$ 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 κάνετε την 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:

```
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
```

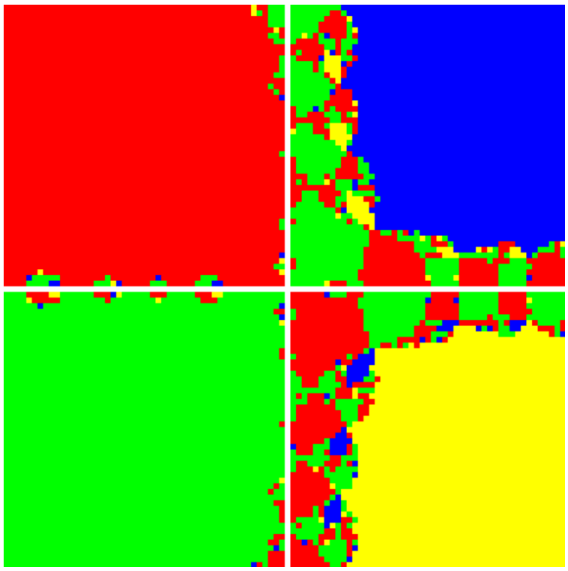
- README Filepath: fractal/README.md
- **Ένα αρχείο που να περιέχει στοιχεία εισόδου και ένα εξόδου διαφορετικά από αυτά της άσκησης. Συγκεκριμένα προτείνουμε να βάλετε έναν συνδυασμό που θεωρείται ότι είναι δύσκολος να γίνει σωστός**.
  - input Filepath: fractal/test/input
  - output Filepath: fractal/test/output

- 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., $1000 \times 1000$.
- **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.

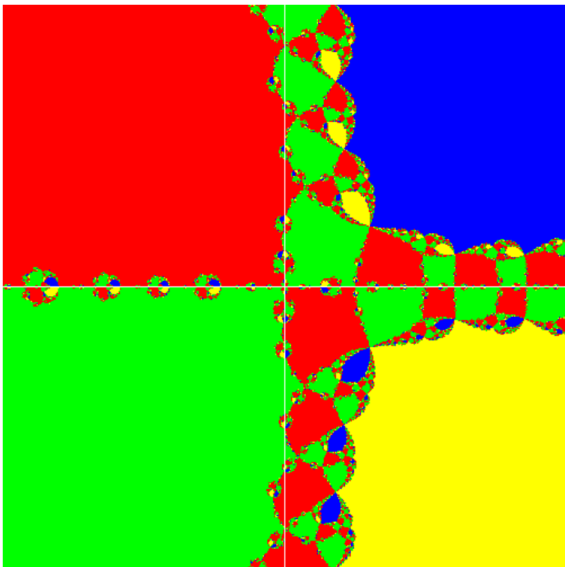
Έστω ότι θέλουμε να βρούμε τις λύσεις της Newton-Raphson για το πολυώνυμο $1 + z - z^3 + z^4$ για τα ακόλουθα $z_0$: $\{-1 - i, -1 + 0i, -1 + i, 0 - i, 0, 0 + i, 1 - i, 1, 1 + i\}$, δηλαδή θέλουμε να σαρώσουμε όλους τους μιγαδικούς από το $-1 - i$ μέχρι το $1 + i$ με βήμα 1. Παρακάτω παραθέτουμε την 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 στην οποίες αποτιμάται το πολυώνυμο για τα διάφορα $z_0$:



Newton fractal από το $-2-2i$ μέχρι το $+2+2i$.



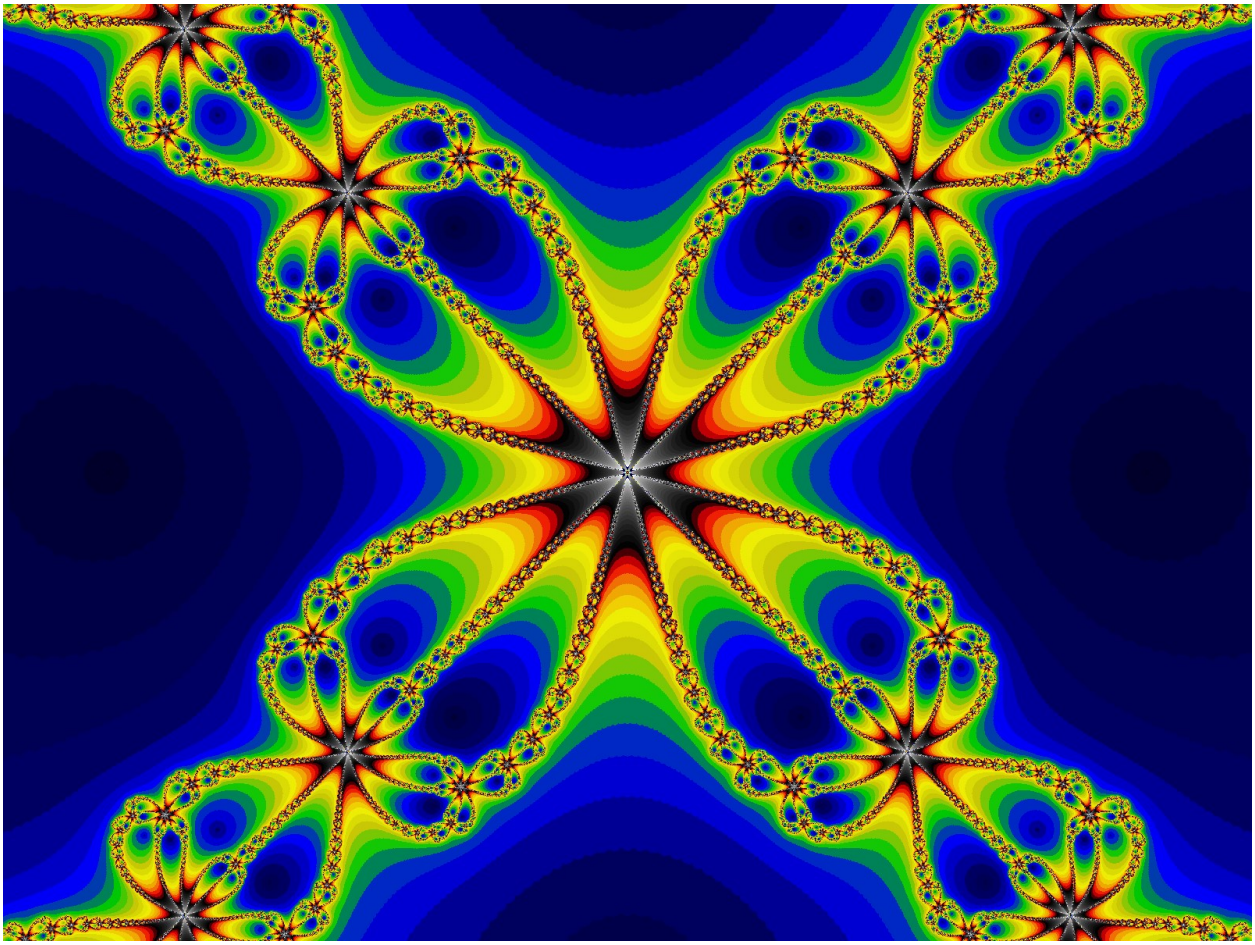
Το ίδιο fractal με βήμα $5\times$ μικρότερο.

Σχήμα 3: Newton fractals για το πολυώνυμο $1+z-z^3+z^4$ με διαφορετικό βήμα.

| 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 |

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
```



Σχήμα 4: Newton fractal όπου το βάθος των χρωμάτων υποδεικνύει πόσο γρήγορα συγκλίνει (αριθμός επαναλήψεων) το κάθε σημείο σε μια ρίζα του $z^4 - 1$.

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 διά-

σταση 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 [5]. 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 [7]. Στα σχήματα παρακάτω δίνουμε κάποια παραδείγματα για το πως μπορεί να μοιάζουν οι λύσεις για κάποια πολυώνυμα.

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.

Αν φτάσατε μέχρι εδώ, σας συγχαίρω για το κουράγιο σας! Ως συνήθως, στο αρχείο README.md πρέπει να προσθέσετε οποιεσδήποτε καινούριες παρατηρήσεις είχατε για αυτό το μέρος της άσκησης. Ο κώδικας απαιτείται να είναι καλά τεκμηριωμένος με σχόλια καθώς αυτό θα είναι μέρος της βαθμολόγησης. Καλή συνέχεια!

## Αναφορές


[1] Artificial Intelligence @ DIT . `https://cgi.di.uoa.gr/~ys02/siteAI2023/lectures.html`.

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

[3] Πρόσκληση για Εργασία 3 . `https://classroom.github.com/a/4dk33i1v`.

[4] Complex Numbers. `https://en.wikipedia.org/wiki/Complex_number`.

[5] Fractal. `https://en.wikipedia.org/wiki/Fractal`.

[6] Imaginary Numbers. https://en.wikipedia.org/wiki/Imaginary_number.

[7] Newton Fractal. https://en.wikipedia.org/wiki/Newton_fractal.

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

[9] Roomba. https://en.wikipedia.org/wiki/Roomba.

[10] Roomba Random Patterns. https://www.cnet.com/home/kitchen-and-household/this-is-why-your-roombas-random-patterns-actually-make-perfect-sense/.