ΣΕΤ ΑΣΚΗΣΕΩΝ 2

ΕΡΓΑΣΤΗΡΙΟ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ Ι, ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2018-2019

Προθεσμία: Τρίτη 18/12/2018, 23:59

Διαβάστε πριν ξεκινήσετε

Διαβάστε την εκφώνηση προσεκτικά και "σχεδιάστε" το πρόγραμμά σας στο χαρτί.

Για κάθε στάδιο, αποφασίστε τι μεταβλητές θα χρειαστείτε, τι ονόματα θα τους δώσετε, αν χρειάζονται σταθερές κι αν ναι για ποιες ποσότητες, τι δομές ελέγχου θα χρησιμοποιήσετε για κάθε λειτουργία και πώς θα κάνετε τους υπολογισμούς που χρειάζονται.

Μη διστάζετε να ζητήσετε βοήθεια! Χρησιμοποιήστε κατά προτίμηση την εφαρμογή συζητήσεων στο e-class και μόνο αν είναι απαραίτητο email στο `ce120lab@gmail.com` (π.χ. αν πραγματικά επιβάλλεται να στείλετε κάποιο κομμάτι κώδικα μαζί με το μήνυμά σας).

Η εργασία αυτή μπορεί να γίνει σε ομάδες μέχρι και 2 ατόμων. Δε χρειάζεται να είστε ομάδα με το ίδιο άτομο που είστε στο εργαστήριο ή το ίδιο άτομο με το οποίο κάνατε την 1η εργασία. Μπορείτε να συζητάτε τις ασκήσεις με συμφοιτητές σας αλλά δεν επιτρέπεται η ανταλλαγή κώδικα με οποιονδήποτε τρόπο.

Ξεκινήστε νωρίς! Ο προγραμματισμός είναι πάντα ΠΟΛΥ πιο χρονοβόρος από ότι περιμένετε.

Εκπρόθεσμες ασκήσεις δε γίνονται δεκτές.

Οι ασκήσεις σας θα βαθμολογηθούν στα παρακάτω (χωρίς ιδιαίτερη σειρά):

- Ορθότητα
- Γενική μορφοποίηση προγράμματος (στοίχιση, ονόματα μεταβλητών και σταθερών, κτλ.)
- Σχεδιασμός προγράμματος και αποτελεσματική χρήση κατάλληλων δομών, μεταβλητών, σταθερών κτλ.
- Σχεδιασμός και χρήση κατάλληλων συναρτήσεων. Θα σας προτείνουμε κάποιες, αλλά περιμένουμε να σκεφτείτε και δικές σας.
- Συμμόρφωση με τις προδιαγραφές
- Αποτελεσματικά σχόλια, σύμφωνα με τους κανόνες σχολιασμού του σχετικού φυλλαδίου.

Απαγορεύεται αυστηρά η χρήση goto και η χρήση καθολικών μεταβλητών εκτός από αυτές που επιτρέπουμε ρητά. Επίσης απαγορεύεται η αναδρομική κλήση της main.

ΔΙΑΒΑΣΤΕ ΟΛΗ ΤΗΝ ΕΚΦΩΝΗΣΗ ΚΑΘΕ ΑΣΚΗΣΗΣ ΠΡΙΝ ΞΕΚΙΝΗΣΕΤΕ!

Άσκηση 1 : Ναρκαλιευτής

Το πρόγραμμά σας πρέπει να αποθηκευτεί σε αρχείο με όνομα `hw2a.c`

Σε αυτή την άσκηση θα γράψετε ένα πρόγραμμα το οποίο υλοποιεί το παιχνίδι του ναρκαλιευτή σε τερματικό (κονσόλα), χωρίς γραφικά.

Αρχικοποίηση παιχνιδιού

Επιθυμούμε κάθε φορά που ξεκινά νέο παιχνίδι να αρχικοποιούνται με τυχαίο τρόπο οι θέσεις των ναρκών. Για να το επιτύχουμε αυτό χρησιμοποιούμε μια γεννήτρια ψευδοτυχαίων αριθμών:μια συνάρτηση η οποία κάθε φορά που καλείται σε ένα πρόγραμμα παράγει έναν καινούργιο ψευδοτυχαίο αριθμό. Η συνάρτηση αυτή στη C λέγεται `rand` και για να χρησιμοποιηθεί χρειάζεται `#include<stdlib.h>` .

Επιπλέον χρησιμοποιούμε τη συνάρτηση `srand` η οποία παίρνει ως παράμετρο μια τιμή που λέγεται seed, με βάση την οποία θα παραχθούν από τη `rand` οι ψευδοτυχαίοι αριθμοί. Αν το seed είναι πάντα ίδιο, τότε η ακολουθία τιμών που θα παράγει η `rand` κατά την εκτέλεση του προγράμματος θα είναι επίσης πάντα ίδια, με συνέπεια κάθε εκτέλεση του προγράμματος να βάζει τελικά τις νάρκες πάντα στα ίδια κελιά. Για να αποφύγουμε κάτι τέτοιο τυπικά θέτουμε ως seed την τρέχουσα ώρα μέσω της συνάρτησης `time`. Για τη συνάρτηση `time` χρειάζεται `#include<time.h>` . Όμως, δεν είναι πάντα επιθυμητό να παράγονται ψευδοτυχαίοι αριθμοί σε κάθε εκτέλεση: αν το πρόγραμμά μας έχει κάποιο λογικό λάθος, θέλουμε να παράγει πάντα τα ίδια αποτελέσματα ώστε να μπορέσουμε να εντοπίσουμε το λάθος. Γι αυτό το λόγο, στην άσκηση θα δοθεί η δυνατότητα να χρησιμοποιήσουμε είτε την ώρα είτε κάποια συγκεκριμένη τιμή ως seed.

Σημειώστε πως η συνάρτηση `srand` πρέπει να καλείται μία φορά μόνο, στην αρχή του προγράμματος, και ποτέ μέσα σε επανάληψη.

Στο παρακάτω πρόγραμμα μπορείτε να δείτε ένα παράδειγμα χρήσης αυτών των συναρτήσεων. Το πρόγραμμα παράγει μια σειρά 15 ψευδοτυχαίων αριθμών από το 0 έως και το 99. Επειδή η `rand` κανονικά παράγει ακεραίους από το 0 μέχρι και το RAND_MAX, χρησιμοποιούμε τον τελεστή % για να περιορίσουμε το εύρος αποτελεσμάτων στο επιθυμητό.

Δοκιμάστε να τρέξετε το πρόγραμμα μερικές φορές με χρήση της `time` μέσα στην `srand` και μερικές με χρήση κάποιου συγκεκριμένου θετικού ακεραίου για να δείτε τη διαφορά στα αποτελέσματα. Δοκιμάστε επίσης να τρέξετε το πρόγραμμα με κλήση της `srand` μέσα στην `for`, για να δείτε γιατί είναι λάθος αυτό.

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

int main(int argc, char *argv[]) {
    int i;
    srand(time(NULL));
    for (i=1; i<=15; i++) {
        printf("%d\n", rand() % 100 );
    }
    return 0;
}
```

Διαβάστε όλα τα βήματα καθώς και τις οδηγίες και διευκρινίσεις που ακολουθούν πριν ξεκινήσετε να σχεδιάζετε τη λύση σας. Θα σας επιτραπεί να χρησιμοποιήσετε δύο καθολικές μεταβλητές για να αποθηκεύσετε δύο πίνακες. Δεν επιτρέπονται άλλες καθολικές μεταβλητές στο πρόγραμμα.

Σε γενικές γραμμές, το πρόγραμμα θα πρέπει να αρχικοποιεί το παιχνίδι και σε κάθε γύρο να εκτυπώνει τον καμβά, να διαβάζει τις συντεταγμένες του κελιού το οποίο ο χρήστης θέλει να αποκαλύψει και να ανανεώνει την κατάσταση του παιχνιδιού. Στο τέλος ενός παιχνιδιού, αν ο χρήστης επιθυμεί, μπορεί να παίξει νέο παιχίδι (ακόμη και διαφορετικού βαθμού δυσκολίας).

Όπου σας λέμε να εκτυπώσετε το διαχωριστικό string, εισάγετε μια `printf("\n#\n");`

Βήμα 0

Εκτυπώστε χαρακτήρα αλλαγής γραμμής και το μήνυμα `Seed?` με ένα κενό μετά το ερωτηματικό. Ακολούθως διαβάστε έναν απρόσημο ακέραιο. Αν αυτός είναι μηδέν, θέστε το seed του προγράμματος όσο είναι η ώρα, διαφορετικά θέστε το όσο είναι ο ακέραιος που μόλις διαβάσατε.

Βήμα 1

Εκτυπώστε χαρακτήρα αλλαγής γραμμής και το μενού:

```
Choose difficulty level:
EASY (E/e)
MEDIUM (M/m)
EXPERT (X/x)
==>
```

Με ένα κενό μετά το ==>

Ο χρήστης επιλέγει το επίπεδο δυσκολίας πληκτρολογώντας **`E`** ή **`e`** για το επίπεδο `EASY`, **`M`** ή **`m`** για το επίπεδο `MEDIUM`, **`X`** ή **`x`** για το επίπεδο `EXPERT`. Αν πληκτρολογήσει οτιδήποτε άλλο, το βήμα 1 επαναλαμβάνεται έως ότου δοθεί έγκυρη επιλογή.

Τα επίπεδα διαφοροποιούνται ως προς το μέγεθος του καμβά του παιχνιδιού ως εξής:

- `EASY`: μέγεθος καμβά 6x6
- `MEDIUM`: μέγεθος καμβά 12x12
- `EXPERT`: μέγεθος καμβά 24x24

Σε όλα τα επίπεδα το ποσοστό των ναρκών ως προς το σύνολο των θέσεων του καμβά δεν πρέπει να υπερβαίνει το **20%**.

Κάθε θέση (τετραγωνάκι) του καμβά μπορεί να περιέχει τα εξής:

- Νάρκη, συμβολίζεται με τον χαρακτήρα `'*'`.
- Έναν αριθμό με τιμές **`0-8`** που δηλώνει πόσες νάρκες περιβάλλουν τη συγκεκριμένη θέση.

Ανάλογα με την επιλογή του χρήστη και τις παραπάνω προδιαγραφές, αρχικοποιήστε το παιχνίδι μέσω κατάλληλης συνάρτησης.

Εκτυπώστε το διαχωριστικό string.

Βήμα 2

Γράψτε μια συνάρτηση που σχεδιάζει τον καμβά. Ο καμβάς εκτυπώνεται με ένα χαρακτήρα αλλαγής γραμμής πριν και δύο μετά, και στο ενδιάμεσο ως εξής (για μέγεθος 6 - προσαρμόστε αναλόγως τα άλλα μεγέθη):

```
    1   2   3   4   5   6

1   -   -   -   -   -   -
2   -   -   -   -   -   -
3   -   -   -   -   -   -
4   -   -   -   -   -   -
5   -   -   -   -   -   -
6   -   -   -   -   -   -
```

Κάθε στήλη έχει πλάτος 4 χαρακτήρες και υπάρχει μια κενή γραμμή μετά τη γραμμή με την αρίθμηση (δηλαδή το παραπάνω σχήμα καταλαμβάνει 8 γραμμές). Στην αρχή, σε κάθε θέση εμφανίζονται παύλες γιατί δεν έχουν αποκαλυφθεί τα περιεχόμενα.

Βήμα 3

Αφού ζωγραφιστεί στην οθόνη ο καμβάς, ο χρήστης καλείται να επιλέξει μία θέση του καμβά την οποία επιθυμεί να αποκαλύψει. Αρχικά εκτυπώστε χαρακτήρα αλλαγής γραμμής, `Row>` κι ένα κενό και διαβάστε τη γραμμή κι εφόσον είναι σωστή, χαρακτήρα αλλαγής γραμμής, `Col>` κι ένα κενό και διαβάστε τη στήλη.

Μετά την ανάγνωση κάθε μίας τιμής γίνεται έλεγχος για το αν αυτή είναι θετική και μικρότερη ή ίση του αντίστοιχου μεγέθους του καμβά. Αν δεν είναι, το πρόγραμμα εκτυπώνει χαρακτήρα αλλαγής γραμμής, το μήνυμα λάθους `Invalid Z. Try again!`, το διαχωριστικό string και επαναλαμβάνει την εκτύπωση Row> ή Col> και την ανάγνωση αυτής της τιμής. Το Z είναι η λέξη `row` ή `column` ανάλογα με το ποια ποσότητα ήταν λάθος.

Αν όμως έχει δοθεί 0 για τη γραμμή, τότε εκτυπώνεται χαρακτήρας αλλαγής γραμμής, το μήνυμα `Game interrupted.` και χαρακτήρας αλλαγής γραμμής. Το πρόγραμμα διακόπτει το τρέχον παιχνίδι (χωρίς να διαβάσει τη στήλη) και συνεχίζει στο **βήμα 5.**

Εφόσον η θέση που επιλέχθηκε δεν περιέχει νάρκη, από εδώ και μέχρι το τέλος του παιχνιδιού θα πρέπει να εμφανίζονται τα περιεχόμενά της.

Βήμα 4

Ελέγχουμε εάν συντρέχουν λόγοι τερματισμού του παιχνιδιού. Το παιχνίδι τελειώνει στις παρακάτω περιπτώσεις:

1. **επιτυχώς** όταν ο χρήστης αποκαλύψει όλες τις θέσεις στις οποίες δεν υπάρχει νάρκη. Σε αυτή την περίπτωση εκτυπώνεται χαρακτήρας αλλαγής γραμμής, το μήνυμα `Congratulations!` και χαρακτήρας αλλαγής γραμμής.
2. **ανεπιτυχώς** εάν ο χρήστης επιλέξει ένα τετραγωνάκι στο οποίο υπάρχει μία νάρκη. Σε αυτή την περίπτωση εκτυπώνεται χαρακτήρας αλλαγής γραμμής, το μήνυμα `KABOOM!` και χαρακτήρας αλλαγής γραμμής.

Και στις δύο παραπάνω περιπτώσεις, μετά το μήνυμα εκτυπώνεται ο καμβάς με όλα τα κελιά αποκαλυμμένα (όπως στο παρακάτω παράδειγμα) και προχωρά στο βήμα 5.

```
     1   2   3   4   5   6

1    1   3   *   2   1   *
2    2   *   *   3   2   1
3    4   *   5   *   2   1
4    *   *   5   4   4   *
5    3   5   *   *   *   2
6    1   *   *   4   2   1
```

Εάν δεν συντρέχουν λόγοι τερματισμού του παιχνιδιού τότε το πρόγραμμα επαναλαμβάνει τα **βήματα 3 και 4**.

Βήμα 5

Σε αυτό το βήμα καταλήγουμε αν έχει τερματίσει το τρέχον παιχνίδι. Το πρόγραμμα εκτυπώνει το διαχωριστικό string και το μήνυμα `Play again? (y/n)` (υπάρχει ένα κενό μετά τη δεξιά παρένθεση). Ο χρήστης καλείται να εισάγει **Y** ή **y** για να ξαναπαίξει ή **N** ή **n** για να τερματίσει το πρόγραμμα. Σε κάθε εισαγωγή διαφορετικού χαρακτήρα το βήμα 5 επαναλαμβάνεται. Σε περίπτωση καταφατικής απάντησης ('**y**' ή '**Y**') το πρόγραμμα επαναλαμβάνει την εκτέλεση **των βημάτων 1-5.**

Οδηγίες και συμβουλές για την υλοποίηση του προγράμματος

Δεδομένα

Δημιουργήστε δύο **global** (καθολικούς) διδιάστατους πίνακες χαρακτήρων με μέγεθος το μέγιστο μέγεθος του καμβά (αντιστοιχεί στο επίπεδο δυσκολίας **EXPERT**). Ας τους ονομάσουμε (τα ονόματα είναι ενδεικτικά και δεν απαιτείται να τα χρησιμοποιήσετε στο πρόγραμμα σας):

- **game:** αποθηκεύει τις τιμές των θέσεων του καμβά και
- **displayed:** αποθηκεύει την πληροφορία εάν μία θέση του πίνακα **game** έχει αποκαλυφθεί στον χρήστη ή όχι.

Συναρτήσεις

1. Δημιουργήστε μία συνάρτηση αρχικοποίησης των πινάκων **game** και **displayed**
2. Δημιουργήστε μία συνάρτηση που υπολογίζει αν έχουν αποκαλυφθεί όλες οι θέσεις που δεν περιέχουν νάρκη.
3. Δημιουργήστε μία συνάρτηση η οποία εκτυπώνει τον καμβά
4. Δημιουργήστε μία συνάρτηση η οποία εκτυπώνει τη λύση, όπως αυτή αποκαλύπτεται στον χρήστη στο τέλος του παιχνιδιού.

Όλες οι παραπάνω συναρτήσεις λαμβάνουν ως όρισμα μία παράμετρο τύπου **int** που αντιστοιχεί στον αριθμό των χρησιμοποιούμενων γραμμών και στηλών των πινάκων **game** και **displayed**, ανάλογα με το επίπεδο δυσκολίας που έχει επιλέξει κάθε φορά ο χρήστης.

Προσπαθήστε να υλοποιήσετε το παιχνίδι με χρήση τουλάχιστον των παραπάνω συναρτήσεων. Καλό είναι να προσθέσετε και άλλες.

Υπολογισμός του αριθμού των ναρκών

Ο υπολογισμός του αριθμού των ναρκών προκύπτει εάν πολλαπλασιάσετε τον αριθμό των θέσεων του καμβά (διαφέρει για κάθε επίπεδο) με το μέγιστο ποσοστό πλήρωσης (**20%**). Αποκόπτοντας το δεκαδικό μέρος του αριθμού που προκύπτει (θα σας φανεί χρήσιμη η συνάρτηση `trunc` της μαθηματικής βιβλιοθήκης) λαμβάνετε τον αριθμό των ναρκών που θα πρέπει να καταχωρήσετε σε κάθε παιχνίδι.

Αρχικοποίηση των πινάκων `game` και `displayed`

Σε κάθε νέο παιχνίδι καλείστε να αρχικοποιήσετε τους πίνακες **`game`** και **`displayed`** ως εξής:

- **`displayed`:** κάθε θέση του πίνακα **`displayed`** μπορεί να πάρει μόνο 2 τιμές, μία που δηλώνει ότι η αντίστοιχη θέση του πίνακα **`game`** έχει αποκαλυφθεί στον χρήστη και μία που δηλώνει ότι δεν έχει αποκαλυφθεί. Επιλέξτε τις τιμές αυτές μόνοι σας, ορίζοντας τις αντίστοιχες σταθερές και αρχικοποιήστε τον πίνακα ώστε να μην έχει αποκαλυφθεί καμία θέση του πίνακα.
- **`game`:** Αρχικά αρχικοποιήστε τον πίνακα ώστε να μην περιέχει καμία πληροφορία. Στη συνέχεια προσδιορίστε τις θέσεις των ναρκών. Για κάθε μία νάρκη που θέλετε να τοποθετήσετε δημιουργήστε δύο τυχαίους αριθμούς, ένα που αφορά τη γραμμή στην οποία θα τοποθετηθεί η νάρκη και έναν που αφορά τη στήλη. Εάν σε μία θέση υπάρχει ήδη νάρκη, επαναλάβετε την διαδικασία μέχρι να βρείτε μία κενή θέση. Δώστε ιδιαίτερη προσοχή ώστε να παράγετε πιθανές θέσεις που θα περιέχονται στο χρησιμοποιούμενο μέρος του καμβά (πχ. αν το επίπεδο δυσκολίας είναι easy, δεν πρέπει να παράγετε θέσεις που θα μπουν σε γραμμή ή στήλη μεγαλύτερη της έκτης).

Άσκηση 2 : Έλεγχος ορθογραφίας.

Το πρόγραμμά σας πρέπει να αποθηκευτεί σε αρχείο με όνομα `hw2b.c` .

Θα γράψετε ένα πρόγραμμα το οποίο βρίσκει αν μια λέξη υπάρχει στο λεξικό, κι αν όχι, προτείνει τις δύο κοντινότερες σε αυτή λέξεις που βρίσκει στο λεξικό.

Το πρόγραμμα ξεκινά εκτυπώνοντας το μήνυμα `Word to check:` (με ένα κενό μετά το ':') και διαβάζει από το πληκτρολόγιο τη λέξη που ο χρήστης θέλει να ελέγξει. Ακολούθως εκτυπώνει το διαχωριστικό string και μετά διαβάζει μια σειρά λέξεων λεξικού μέχρις ότου:

(a) Να βρει στο λεξικό μια λέξη ίδια με αυτή που προσδιόρισε ο χρήστης, ή

(b) Να διαβάσει τη λέξη `EndOfDictionary` η οποία σηματοδοτεί το τέλος των λέξεων λεξικού.

Κατά τη διάρκεια ανάγνωσης των λέξεων λεξικού, για κάθε λέξη που διαβάζει εκτυπώνει χαρακτήρα αλλαγής γραμμής, το μήνυμα `Distance: dd, Word: w` ακολουθούμενο από χαρακτήρα αλλαγής γραμμής, όπου `dd` η απόσταση της λέξης-λεξικού από τη λέξη του χρήστη με πλάτος 2 και `w` η λέξη λεξικού που μόλις διαβάστηκε.

Παράλληλα, διατηρεί κάθε στιγμή τις δύο λέξεις λεξικού που έχουν τις δύο μικρότερες αποστάσεις από τη λέξη του χρήστη. Σε περιπτώσεις ισοψηφίας (πχ. υπάρχουν περισσότερες από δύο λέξεις με ίδια απόσταση), επιλέγετε όποια διαβάστηκε πρώτη (hint: χρησιμοποιήστε κατάλληλο τελεστή σύγκρισης).

Η απόσταση ανάμεσα σε δύο λέξεις ορίζεται ως το πλήθος των γραμμάτων που βρίσκονται σε αντίστοιχες θέσεις και είναι διαφορετικά (δεν αναγνωρίζουμε διαφορά ανάμεσα σε μικρά-κεφαλαία).

Παραδείγματα:

Λέξη 1	`baseball`	`baseball`	`baseball`	`BaseBall`
Λέξη 2	`soccer`	`basket`	`ballpark`	`basebaLLer`
Απόσταση	8	5	5	2

Αν η λέξη του χρήστη βρεθεί στο λεξικό, τότε η ανάγνωση λέξεων λεξικού σταματά άμεσα, το πρόγραμμα εκτυπώνει το διαχωριστικό string και το μήνυμα `"w" is in the dictionary.` ακολουθούμενο από χαρακτήρα αλλαγής γραμμής, όπου `w` η λέξη του χρήστη, και τερματίζει (hint: μπορείτε να χρησιμοποιήσετε εντολή `return` σε αυτό το σημείο για να τερματίσει η εκτέλεση της `main`.)

Αν η λέξη του χρήστη δε βρεθεί στο λεξικό, τότε το πρόγραμμα εκτυπώνει το διαχωριστικό string και το μήνυμα:

```
"w" is not in the dictionary.
Suggestions:
"w1", d1
"w2", d2
```

ακολουθούμενο από χαρακτήρα αλλαγής γραμμής, όπου `w` είναι η λέξη του χρήστη, `w1`, `w2` είναι οι λέξεις λεξικού που έχουν τις δύο μικρότερες αποστάσεις από τη λέξη του χρήστη, εμφανιζόμενες με αύξουσα λεξικογραφική σειρά και `d1`, `d2` είναι οι αντίστοιχες αποστάσεις. Εάν υπάρχουν λιγότερες από δύο προτεινόμενες λέξεις, εμφανίζονται όσες υπάρχουν.

Υποθέσεις, απαιτήσεις, συμβουλές:

- Θεωρήστε ότι μια λέξη έχει το πολύ 29 γράμματα, αλλά το πρόγραμμά σας πρέπει να είναι γραμμένο έτσι ώστε να μπορεί να αλλάξει εύκολα αυτό το μέγεθος.
- Όπου είναι δυνατό, να χρησιμοποιείτε συναρτήσεις συμβολοσειρών.
- Για τις συγκρίσεις που αγνοούν αν ένα γράμμα είναι κεφαλαίο ή μικρό, θα σας φανεί χρήσιμη η συνάρτηση `tolower` η οποία παίρνει ως παράμετρο ένα γράμμα κι επιστρέφει το ίδιο γράμμα σε μικρό (πχ. η `tolower('A')` επιστρέφει `'a'`). Χρειάζεται `#include<ctype.h>`

Πώς να παραδώσετε τη δουλειά σας

Πριν παραδώσετε το πρόγραμμά σας, προσθέστε σε σχόλια στην αρχή του αρχείου τα πλήρη ονόματα και ΑΕΜ των μελών της ομάδας. Παρακαλούμε να γράφετε τα σχόλια ΜΟΝΟ με λατινικούς χαρακτήρες.

1. **Κατασκευάστε ένα κατάλογο** με όνομα `hw2submit` και αντιγράψτε μέσα σε αυτόν το `hw2a.c` και το `hw2b.c`
2. Κατασκευάστε ένα αρχείο με όνομα team.txt και προσθέστε σε αυτό τα πλήρη ονόματα και ΑΕΜ των μελών της ομάδας, ακόμη κι αν η ομάδα αποτελείται από ένα άτομο.
3. **Κάντε δεξί κλικ** στον κατάλογο `hw2submit` και επιλέξτε `Compress here as tar.gz`
4. **Πηγαίνετε στο Autolab**,
5. **Επιλέξτε το hw2.**
   (a) Αν είστε ομάδα δύο ατόμων, **κατασκευάστε** μια ομάδα μέσω της επιλογής Group Options. Αυτή θα ισχύει μόνο για το hw2.
   (b) Κάντε Submit το hw2submit.tar.gz στο Autolab.