Τμήμα Εφαρμοσμένης Πληροφορικής
ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ
Εξάμηνο Β'
Φύλλο Ασκήσεων 2: ΣΤΟΙΒΕΣ (υλοποίηση με πίνακες)
Μάγια Σατρατζέμη, Γεωργία Κολωνιάρη

Παρατηρήσεις:

1. *Τα δεδομένα εισόδου διαβάζονται πάντα με ξεχωριστές εντολές scanf( ) το καθένα και με τη σειρά που δηλώνονται στις εκφωνήσεις.*
2. *Αντίστοιχα για τα δεδομένα εξόδου και όπου δεν υπάρχουν περαιτέρω διευκρινήσεις για τη μορφή τους, αυτά θα εμφανίζονται με ξεχωριστές εντολές printf(" ..\n") το καθένα και με τη σειρά που δηλώνονται στις εκφωνήσεις.*
   - *i) Τα στοιχεία των κόμβων της στοίβας θα εμφανίζονται σε μια γραμμή με ένα κενό χαρακτήρα μεταξύ τους. Σε περίπτωση που οι κόμβοι της στοίβας περιέχουν περισσότερα από ένα στοιχεία, τότε τα στοιχεία κάθε κόμβου θα εμφανίζονται σε μια γραμμή με ένα κενό χαρακτήρα μεταξύ τους, ενώ κάθε κόμβος θα εμφανίζεται σε διαφορετική γραμμή.*
   - *ii) Αν κατά τη διάσχιση της λίστας διαπιστώσετε ότι η στοίβα είναι κενή, τότε να εμφανίζετε αντίστοιχα το μήνυμα 'EMPTY STACK'.*
3. *Σε όσες από τις ασκήσεις θεωρείται δεδομένη η ύπαρξη στοίβας θα πρέπει προηγουμένως να τη δημιουργήσετε.*

1. Θεωρείστε ότι A = 7.0, B = 4.0, C = 3.0, D = -2.0. Υπολογίστε τις ακόλουθες RPN εκφράσεις:

(a) *A B + C / D **
(b) *A B C + / D **
(c) *A B C D + / **
(d) *A B + C + D +*
(e) *A B + C D + +*
(f) *A B C + + D +*
(g) *A B C D + + +*
(h) *A B – C – D –*
(ie) *A B – C D - -*
(j) *A B C - - D -*
(k) *A B C D - - -*

2. Για κάθε μία από τις ακόλουθες RPN εκφράσεις ιχνηλατήστε τον αλγόριθμο υπολογισμού RPN εκφράσεων και καθορίστε ποια είναι τα περιεχόμενα της στοίβας ακριβώς πριν από το διάβασμα του χαρακτήρα που σημειώνεται με ☊. Επίσης, υπολογίστε κάθε RPN έκφραση.

(a) *32 5 3 + / 5 **
☊ ☊
(b) *2 17 – 5 / 3 **
☊ ☊
(c) *19 7 15 5 - - -*
☊ ☊ ☊

3. Μετατρέψτε τις ακόλουθες ενδοθεματικές εκφράσεις σε RPN:

(a) *A * B + C - D*
(b) *A + B / C + D*
(c) *(A + B) / C + D*
(d) *A + B / (C + D)*
(e) *(A + B) / (C + D)*
(f) *(A – B) * (C – (D + E))*
(g) *(((A – B) – C) – D) - E*
(h) *A – (B – (C – (D – E)))*

4. Για κάθε μία από τις ακόλουθες ενδοθεματικές εκφράσεις ιχνηλατήστε τον αλγόριθμο μετατροπής ενδοθεματικής έκφρασης σε RPN εκφράσεων και καθορίστε ποια είναι τα περιεχόμενα της στοίβας και το αποτέλεσμα (RPN έκφραση μέχρι τη δεδομένη στιγμή) ακριβώς πριν από το διάβασμα του χαρακτήρα που σημειώνεται με ☊. Επίσης, προσδιορίστε την τελική RPN έκφραση.

(a) *A + B / C - D*
☊ ☊ ☊
(b) *(A + B) / C – D + E*
☊ ☊

(c) *A + B / (C – D) - E*
∩ ∩∩

(d) *A + B / (C – D) * E*
∩ ∩∩

5. Χρησιμοποιήστε τις λειτουργίες που έχουν οριστεί στον ΑΤΔ στοίβα με πίνακα και γράψτε στο κυρίως πρόγραμμα κώδικα για κάθε μία από τις παρακάτω λειτουργίες:

- **(a)** Θέστε στη μεταβλητή *x* την τιμή του δεύτερου στοιχείου από την κορυφή της στοίβας, αφήνοντας τη στοίβα χωρίς τα δύο πρώτα στοιχεία της κορυφής.
- **(b)** Θέστε στη μεταβλητή *x* την τιμή του δεύτερου στοιχείου από την κορυφή της στοίβας, αφήνοντας τη στοίβα αμετάβλητη (δεν θα διαγραφεί κανένα στοιχείο).
- **(c)** Θέστε στη μεταβλητή *x* την τιμή του *n*-οστού στοιχείου από την κορυφή της στοίβας, αφήνοντας τη στοίβα χωρίς τα *n* πρώτα στοιχεία της κορυφής.
- **(d)** Θέστε στη μεταβλητή *x* την τιμή του *n*-οστού στοιχείου από την κορυφή της στοίβας, αφήνοντας τη στοίβα αμετάβλητη. (*Υπόδειξη*: Χρησιμοποιείστε μία άλλη, βοηθητική στοίβα.)
- **(e)** Θέστε στη μεταβλητή *x* την τιμή του τελευταίου στοιχείου της στοίβας, αφήνοντας τη στοίβα αμετάβλητη.
- **(f)** Θέστε στη μεταβλητή *x* την τιμή του τρίτου στοιχείου από τη βάση της στοίβας, αφήνοντας τη στοίβα αμετάβλητη.
- **(g)** Θέστε στη μεταβλητή *x* την τιμή του τελευταίου στοιχείου της στοίβας, αφήνοντας τη στοίβα κενή.

Στο κυρίως πρόγραμμα θα δημιουργείται πρώτα η στοίβα και θα προστίθενται σ' αυτή 20 αριθμοί. Για λόγους απλότητας μπορεί να χρησιμοποιηθεί ένας βρόχος *for*, σε κάθε επανάληψη του οποίου θα προστίθεται η τιμή της μεταβλητής ελέγχου στη στοίβα. Στη συνέχεια εμφανίστε το περιεχόμενο της στοίβας (καλέστε τη βοηθητική συνάρτηση TraverseStack. Επίσης, πριν την εκτέλεση των παραπάνω λειτουργιών θα διαβάζεται ο ακέραιος αριθμός *n* (*n<=Stack.Top*) που χρειάζεται στις λειτουργίες **(c)** και **(d)**. Μετά από την εκτέλεση κάθε λειτουργίας θα εμφανίζεται η τιμή της μεταβλητής *x*. Η έξοδος του προγράμματος φαίνεται στη παρακάτω εικόνα.

```
plithos sto stack 20
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
Give n (n<=19) 4
a->18
b->16
c->14
d->10
e->0
f->2
g->0
Πιέστε ένα πλήκτρο για συνέχεια. . .
```

6. Γράψτε μία συνάρτηση που επιστρέφει το στοιχείο της κορυφής μιας στοίβας τύπου *StackType* την οποία δέχεται μέσω παραμέτρου. Το στοιχείο της κορυφής που επιστρέφει η συνάρτηση **δεν θα διαγράφεται** από τη στοίβα. Υλοποιήστε δύο εκδόσεις της συνάρτησης:

- **(a)** υλοποιήστε τη συνάρτηση *GetTopElementA* σε επίπεδο εφαρμογής: μόνο οι λειτουργίες *Push*, *Pop*, *CreateStack* και *EmptyStack* μπορούν να χρησιμοποιηθούν.
- **(b)** υλοποιήστε τη συνάρτηση *GetTopElementB* σε επίπεδο υλοποίησης: μπορείτε να προσπελάσετε άμεσα τη δομή που χρησιμοποιείται για την αποθήκευση της στοίβας.

Οι δύο διαδικασίες θα καλούνται από το κυρίως πρόγραμμα όπου και θα εμφανίζεται η τιμή που επιστρέφουν. Για να ελέγξετε την ορθότητα των δύο διαδικασιών δημιουργήστε προηγουμένως στο κυρίως πρόγραμμα μια στοίβα που περιλαμβάνει όλους τους περιττούς αριθμούς στο διάστημα [1..100].

7. Γράψτε μία συνάρτηση *GetNthElement* που δέχεται μια στοίβα τύπου *StackType* και έναν ακέραιο αριθμό *n* και επιστρέφει το *n*-οστό στοιχείο της στοίβας από την κορυφή της στοίβας ($1^{ο}$ στοιχείο είναι αυτό που βρίσκεται στη θέση Stack.Top, $2^{ο}$ στη θέση Stack.Top-1, κτλ). Το στοιχείο που επιστρέφει η συνάρτηση δεν θα διαγράφεται από τη στοίβα και η στοίβα θα παραμένει αναλλοίωτη. Υλοποιήστε δύο εκδόσεις της συνάρτησης:

- **(a)** υλοποιήστε τη συνάρτηση *GetNthElementA* σε επίπεδο εφαρμογής: μόνο οι λειτουργίες *Push*, *Pop* μπορούν να χρησιμοποιηθούν.
- **(b)** υλοποιήστε τη συνάρτηση *GetNthElementB* σε επίπεδο υλοποίησης: μπορείτε να προσπελάσετε άμεσα τη δομή που χρησιμοποιείται για την αποθήκευση της στοίβας χωρίς να κάνετε χρήση των διαδικασιών *Push* και *Pop* .

Στο κυρίως πρόγραμμα θα διαβάζεται το *n*, θα καλούνται οι δύο διαδικασίες και θα εμφανίζεται η τιμή που επιστρέφουν. Για να ελέγξετε την ορθότητα των δύο διαδικασιών δημιουργήστε προηγουμένως στο κυρίως πρόγραμμα μια στοίβα που περιλαμβάνει όλους τους άρτιους αριθμούς στο διάστημα [1..100].

8. Γράψτε ένα πρόγραμμα που θα δέχεται ένα αλφαριθμητικό, διαβάζοντας το χαρακτήρα προς χαρακτήρα (μέχρι ο χρήστης να δώσει ως χαρακτήρα τον χαρακτήρα αλλαγής γραμμής, ASCII code 10) και θα ελέγχει αν το αλφαριθμητικό που έχει σχηματιστεί, έχει τη μορφή

x C y

όπου *x* είναι ένα αλφαριθμητικό και *y* είναι το αντίστροφο του *x*. Για παράδειγμα, αν *x = 'ABABBA'* τότε *y = 'ABBABA'*. Ο έλεγχος θα γίνεται κατά το διάβασμα του κάθε χαρακτήρα. Το πρόγραμμα θα εμφανίζει το μήνυμα *TRUE* ή *FALSE* αντίστοιχα αν το αλφαριθμητικό έχει ή όχι αυτή τη μορφή.
Υπόδειξη: οι χαρακτήρες που θα διαβαστούν μέχρι να δοθεί ο χαρακτήρας 'C' εισάγονται σε μια στοίβα και κάθε χαρακτήρας που διαβάζεται μετά τον 'C' συγκρίνεται με το στοιχείο που βρίσκεται στην κορυφή της στοίβας. Για το διάβασμα κάθε χαρακτήρα χρησιμοποιείστε : `scanf("%c", &ch); fflush(stdin);`
Διαφορετικά στιγμιότυπα εκτέλεσης δίνονται στη συνέχεια:

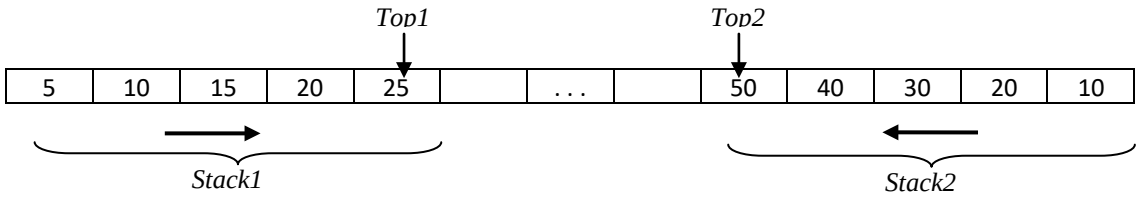
```
Enter string:
m↵
a↵
y↵
a↵
C↵
a↵
y↵
a↵
m↵
↵
TRUE
```

```
Enter string:
m↵
a↵
y↵
a↵
C↵
n↵
FALSE
```

```
Enter string:
m↵
a↵
y↵
a↵
↵
FALSE
```

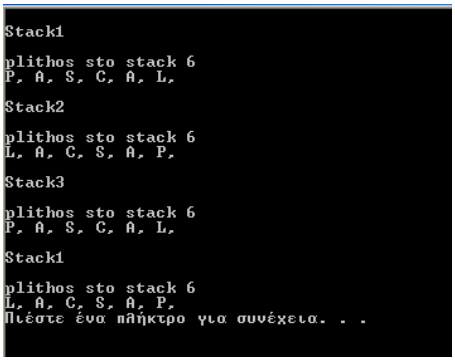
```
Enter string:
m↵
a↵
y↵
a↵
C↵
a↵
y↵
a↵
m↵
a↵
FALSE
```

9. Να υλοποιήσετε μία δομή δεδομένων που θα αποθηκεύει δύο στοίβες σε ένα πίνακα. Οι δύο στοίβες θα αναπτύσσονται όπως φαίνεται στο παρακάτω σχήμα:



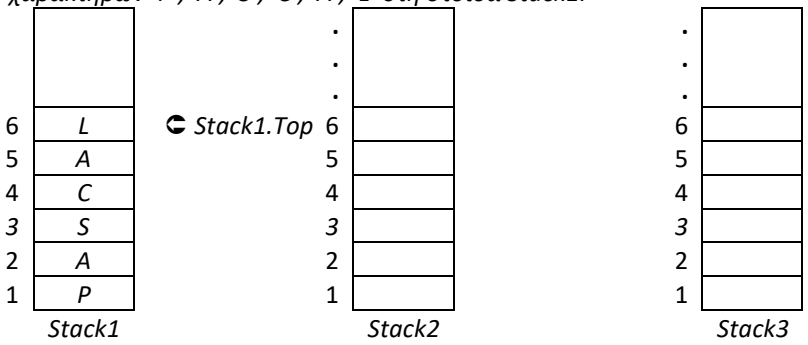
Πρέπει να κάνετε τις απαραίτητες διορθώσεις στην υλοποίηση του ΑΤΔ στοίβα με πίνακα και να γράψετε συναρτήσεις και διαδικασίες για όλες τις βασικές λειτουργίες της στοίβας. Τα υποπρογράμματα CreateStack, *Push*, *Pop*, *EmptyStack* θα πρέπει να δέχονται επιπλέον τον αριθμό της στοίβας, 1 ή 2, που θα επεξεργαστούν. Το υποπρόγραμμα που υλοποιεί τη λειτουργία *FullStack* θα αποτυγχάνει (και οι 2 στοίβες γεμάτες) αν έχουν χρησιμοποιηθεί όλες οι θέσεις του πίνακα. Να ελέγξετε το πρόγραμμα σας για μέγεθος πίνακα 10 θέσεων και δεδομένα του παραπάνω σχήματος για κάθε στοίβα. Καλέστε κατά σειρά τις παρακάτω λειτουργίες: δημιουργήστε τις 2 κενές στοίβες, διαπιστώστε ότι οι στοίβες είναι κενές καλώντας το αντίστοιχο υποπρόγραμμα κάθε φορά, προσθέστε στις 2 στοίβες τα παραπάνω στοιχεία, προσθέστε ένα νέο στοιχείο στη $1^{η}$ στοίβα, θα πρέπει να εμφανίσει μήνυμα ότι η στοίβα είναι γεμάτη, προσθέστε ένα νέο στοιχείο στη $2^{η}$ στοίβα, θα πρέπει να εμφανίσει μήνυμα ότι η στοίβα είναι γεμάτη, αφαιρέστε το κορυφαίο στοιχείο και από τις 2 στοίβες και εμφανίστε τα.

10. Να γράψετε ένα πρόγραμμα που θα αντιστρέφει τη σειρά των στοιχείων μιας στοίβας *Stack1*. Για την αντιστροφή των στοιχείων μέσα στην ίδια τη στοίβα μπορείτε να χρησιμοποιήσετε δύο βοηθητικές στοίβες (*Stack2* και *Stack3*) και τις λειτουργίες που έχουν υλοποιηθεί στον ΑΤΔ στοίβα με πίνακα, αλλά όχι επιπλέον μεταβλητές. Η έξοδος του προγράμματος φαίνεται στη παρακάτω εικόνα (εμφανίζετε το περιεχόμενο των 3 στοιβών με τη βοήθεια της βοηθητικής συνάρτησης TraverseStack μετά από κάθε βήμα του αλγόριθμου που σας προτείνετε στη συνέχεια).

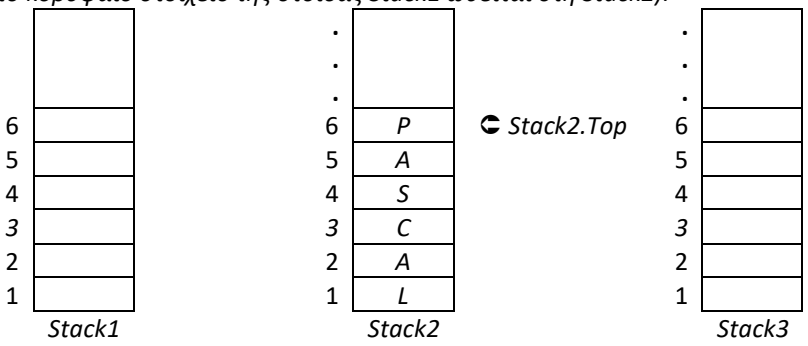


Τα βήματα του αλγορίθμου που πρέπει να υλοποιήσετε φαίνονται στο παρακάτω σχήμα:

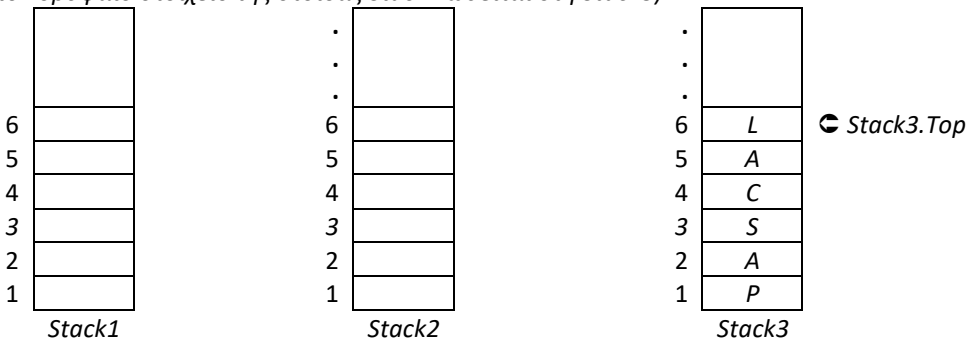
1. *Εισαγωγή των χαρακτήρων 'P', 'A', 'S', 'C', 'A', 'L' στη στοίβα Stack1:*



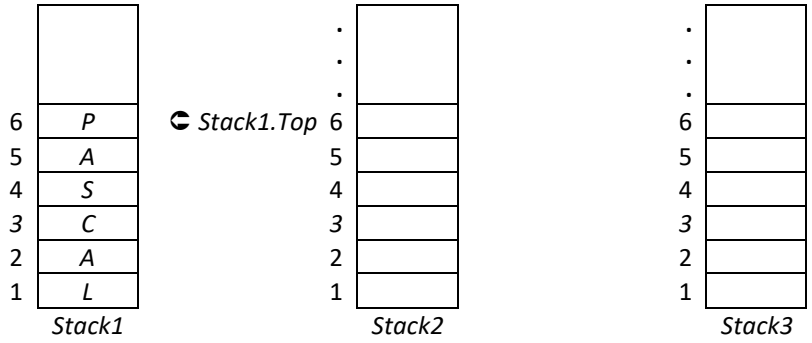
2. *Διαγραφή των στοιχείων της στοίβας Stack1 και εισαγωγή τους στη στοίβα Stack2 (κάθε φορά που διαγράφεται το κορυφαίο στοιχείο της στοίβας Stack1 ωθείται στη Stack2):*



3. *Διαγραφή των στοιχείων της στοίβας Stack2 και εισαγωγή τους στη στοίβα Stack3 (κάθε φορά που διαγράφεται το κορυφαίο στοιχείο της στοίβας Stack2 ωθείται στη Stack3):*



4. *Διαγραφή των στοιχείων της στοίβας Stack3 και εισαγωγή τους στη στοίβα Stack1 (κάθε φορά που διαγράφεται το κορυφαίο στοιχείο της στοίβας Stack3 ωθείται στη Stack1):*



11. Να γράψετε τη συνάρτηση FilterStack που θα δέχεται μια στοίβα Stack που περιέχει αριθμούς και ένα στοιχείο item και θα διαγράφει το στοιχείο item από τη στοίβα. Για παράδειγμα αν η Stack περιέχει τους αριθμούς [2, 13, 10, 5, 7, 1] (κορυφή της στοίβας το 2) και δίνεται το στοιχείο 5, τότε από τη Stack θα διαγράφεται το 5 και θα περιέχει τα στοιχεία [2, 13, 10, 7, 1]. Το πρόγραμμα θα δημιουργεί τη Stack, θα εμφανίζει τα στοιχεία της στοίβας με τη βοηθητική συνάρτηση TraverseStack, στη συνέχεια θα καλεί τη συνάρτηση FilterStack και μετά τη βοηθητική συνάρτηση TraverseStack (ώστε να εμφανιστούν τα στοιχεία της στοίβας πριν και μετά τη διαγραφή του στοιχείου item). *Υπόδειξη: χρησιμοποιείστε μια βοηθητική στοίβα έστω την TempStack, για να απωθήσετε τα στοιχεία της Stack και να τα ωθήσετε στην TempStack μέχρι να εντοπίσετε το στοιχείο item και στη συνέχεια απωθήσετε τα στοιχεία της TempStack και ωθήσετε τα στην Stack.*

12. Στο δοσμένο αρχείο κεφαλίδας StackADT.h, η στοίβα υλοποιείται με έναν πίνακα σταθερού μεγέθους StackLimit = 50. Τροποποιήστε τον ορισμό της στοίβας για να μπορεί να χρησιμοποιεί πίνακες μεταβλητού μεγέθους. Όταν ζητηθεί η προσθήκη ενός στοιχείου και η στοίβα είναι γεμάτη, δε θα εμφανίζεται πια το μήνυμα «Full Stack…». Αντίθετα, ο πίνακας των στοιχείων της στοίβας θα αντικαθίσταται από έναν πίνακα διπλάσιου μεγέθους.
<u>Υπόδειξη:</u> Χρησιμοποιήστε δείκτη αντί για πίνακες σταθερού μεγέθους. Τροποποιήστε τη συνάρτηση Push έτσι ώστε όταν φτάσει στο όριο StackLimit να δεσμεύει μνήμη με τη συνάρτηση realloc της βιβλιοθήκης <stdlib.h>.

13. Σε αυτή την άσκηση θέλουμε να υλοποιήσουμε ένα κομπιουτεράκι (calculator) που υποστηρίζει την αναίρεση (undo) μιας πράξης. Το κομπιουτεράκι θα εκτελείται μέσω της γραμμής εντολών και θα περιλαμβάνει τις ακόλουθες λειτουργίες:

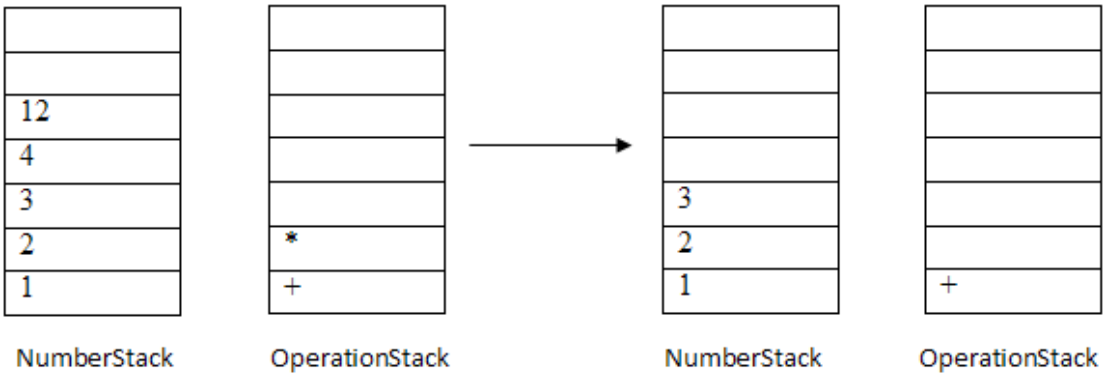
```
DIATHESIMES LEITOURGIES: +,-,*,/,c,s,q,u
 c = clear, s = show result, q = quit, u = undo
Operation:
```

Ο χρήστης θα μπορεί να επιλέξει μία από αυτές τις λειτουργίες. Όταν για πρώτη φορά πληκτρολογεί μια πράξη (+,-,*,/) το πρόγραμμα θα του ζητά 2 αριθμούς:

```
DIATHESIMES LEITOURGIES: +,-,*,/,c,s,q,u
 c = clear, s = show result, q = quit, u = undo
Operation: +
Number1: 1
Number2: 2_
```

Οι πράξεις που θα ακολουθήσουν εφαρμόζονται στο αποτέλεσμα της προηγούμενης. Για παράδειγμα αν η πρώτη πράξη είναι η 1+2=3 (όπως φαίνεται στο παραπάνω σχήμα), στη συνέχεια ο χρήστης θα εισάγει μόνο μια πράξη (π.χ *) και έναν αριθμό (π.χ 4) και το αποτέλεσμα θα γίνεται: 3*4 = 12. Αν θέλει να ξεκινήσει μια καινούρια πράξη θα εισάγει ως λειτουργία (operation) τον καθαρισμό (clear) πατώντας το «c». Αν θέλει να ακυρώσει την προηγούμενη πράξη θα ζητά την αναίρεση της, πατώντας το «u». Στο συγκεκριμένο παράδειγμα η αναίρεση του πολλαπλασιασμού 3*4 οδηγεί και πάλι στο προηγούμενο αποτέλεσμα (1+2=3). Η αναίρεση είναι μια λειτουργία που σε παρόμοιες περιπτώσεις (π.χ στους επεξεργαστές κειμένου) μπορεί να υλοποιηθεί με τη χρήση της δομής δεδομένων **στοίβα**. Στην προκειμένη περίπτωση μπορούμε για παράδειγμα να διατηρούμε 2 στοίβες:
Α) Τη στοίβα *NumberStack* που θα αποθηκεύει τα δεδομένα εισόδου και εξόδου μιας πράξης. Έτσι, η πρόσθεση 1+2=3 προκαλεί την εισαγωγή (**Push**) 3 αριθμών (1,2,3) στη στοίβα *NumberStack*.

Β) Τη στοίβα *OperationStack* που θα αποθηκεύει τις πράξεις που εφαρμόζονται στα δεδομένα εισόδου. Έτσι, η πρόσθεση 1+2=3 προκαλεί την εισαγωγή (Push) του «+» στη στοίβα *OperationStack*. Αντί για το + θα μπορούσαμε να εισάγουμε στη στοίβα αριθμούς αντιστοιχίζοντας έναν αριθμό (0,1,2,3) σε κάθε μία από τις πράξεις (+,-,*,/).
Η αναίρεση μιας πράξης γίνεται με την εξαγωγή (**Pop**) δύο αριθμών από τη στοίβα *NumberStack* και ενός συμβόλου από τη στοίβα *OperationStack*. Στο παρακάτω σχήμα απεικονίζονται τα περιεχόμενα των στοιβών *NumberStack* και *OperationStack* για την ακολουθία πράξεων 1+2=3, 3*4=12 πριν και μετά την αναίρεση (undo) της τελευταίας πράξης (3*4=12):



Γράψτε ένα πρόγραμμα που θα υλοποιεί το κομπιουτεράκι, το οποίο θα υποστηρίζει ένα μικρό και σταθερό αριθμό αναιρέσεων λόγω της στατικής υλοποίησης της στοίβας με πίνακα σταθερού μεγέθους.
Σημείωση 1: Όταν ο χρήστης πληκτρολογεί το c (clear) οι στοίβες *NumberStack* και *OperationStack* θα αδειάζουν.
Σημείωση 2: Όταν ο χρήστης πληκτρολογεί το q (quit) για να τερματίσει την εκτέλεση θα εμφανίζονται με τη βοηθητική συνάρτηση *TraverseStack* τα περιεχόμενα των *NumberStack* και *OperationStack*.

```
DIATHESIMES LEITOURGIES: +,-,*,/,c,s,q,u
 c = clear, s = show result, q = quit, u = undo
Operation: +
Number1: 1
Number2: 2

DIATHESIMES LEITOURGIES: +,-,*,/,c,s,q,u
 c = clear, s = show result, q = quit, u = undo
Operation: *
Number: 4

DIATHESIMES LEITOURGIES: +,-,*,/,c,s,q,u
 c = clear, s = show result, q = quit, u = undo
Operation: s
Result: 12

DIATHESIMES LEITOURGIES: +,-,*,/,c,s,q,u
 c = clear, s = show result, q = quit, u = undo
Operation: u

DIATHESIMES LEITOURGIES: +,-,*,/,c,s,q,u
 c = clear, s = show result, q = quit, u = undo
Operation: s
Result: 3

DIATHESIMES LEITOURGIES: +,-,*,/,c,s,q,u
 c = clear, s = show result, q = quit, u = undo
Operation: q

plithos sto stack 3
3, 2, 1,

plithos sto stack 1
0,
Πιέστε ένα πλήκτρο για συνέχεια. . .
```

14. Σε ένα λειτουργικό σύστημα διατηρείται μία στοίβα κλήσεων (call stack), στην οποία αποθηκεύονται διευθύνσεις μνήμης. Κάθε φορά που καλείται μία υπο-ρουτίνα (έστω GetAverage()), προστίθεται (push) στην στοίβα η διεύθυνση μνήμης της αμέσως επόμενης εντολής της ρουτίνας που την κάλεσε (έστω η CalculateStandardDeviation()). Εάν η CalculateStandardDeviation() αποτελείται από τις παρακάτω εντολές:

ΔΙΕΥΘΥΝΣΗ ΜΝΗΜΗΣ	ΕΝΤΟΛΗ
0	float CalculateStandardDeviation(int X[]){
1	int i,n=sizeof(X);
2	float SD=0;
3	for(i=0;i<n;i++){
4	Avg=GetAverage();
5	SD+=(X[i]-Avg)^2;
	}
6	SD/=n;
7	SD=sqrt(SD);
8	return SD;}

Όταν κληθεί η GetAverage() στην στοίβα προστίθεται η τιμή 5. Όταν τελειώσει την εκτέλεσή της, η τιμή 5 αφαιρείται από τη στοίβα (pop), και συνεχίζει η εκτέλεση των εντολών από τη διεύθυνση 4. Δεδομένου ότι για λόγους ασφαλείας δεν μπορεί να έχει πρόσβαση κάποια υπο-ρουτίνα σε όλες τις θέσεις μνήμης, όταν πραγματοποιείται η pop ελέγχεται εάν η διεύθυνση βρίσκεται εντός των ορίων που έχει θέσει το λειτουργικό σύστημα (με συγκεκριμένο μηχανισμό, ο οποίος δεν αφορά όμως το αντικείμενο της άσκησης). Παράλληλα, αντί για την αποθήκευση της απόλυτης τιμής της διεύθυνσης, μπορεί να αποθηκεύεται η σχετική θέση της διεύθυνσης. Για χάρη απλότητας, θεωρήστε ότι αποθηκεύεται η διαφορά της τελευταίας εντολής της GetAverage() και της επόμενης εντολής της CalculateStandardDeviation(). Εάν θεωρήσουμε ότι η GetAverage() αποτελείται από τις παρακάτω εντολές:

ΔΙΕΥΘΥΝΣΗ ΜΝΗΜΗΣ	ΕΝΤΟΛΗ
9	float GetAverage(int X[]){
10	float Avg=0;
11	int n= sizeof(X);
12	for(i=0;i<n;i++){
13	Avg +=X[I;
	}
14	Avg /=n;
15	return Avg;}

Θα προστεθεί στην στοίβα η τιμή -10, ώστε (θέση μνήμης όταν τελειώσει η εκτέλεση)+(διαφορά)=(νέα θέση μνήμης), δηλαδή 15-10=5.
Γράψτε ένα πρόγραμμα, χρησιμοποιώντας την ΑΔΤ στοίβα, το οποίο προσομοιώνει τη λειτουργία αυτή ως εξής:

- Το πρόγραμμα δέχεται τη μέγιστη διεύθυνση μνήμης (έστω M = 100)
- Στη συνέχεια δέχεται σχετικές διευθύνσεις μνήμης, τις οποίες αποθηκεύει σε μία στοίβα και σταματά όταν δοθεί η τιμή 0 (καμία μετακίνηση).
- Ζητείται η τρέχουσα διεύθυνση μνήμης.
- Εκτελεί τις εντολές μία-μία, με τη σειρά που εξέρχονται από τη στοίβα, υπολογίζοντας την νέα διεύθυνση μνήμης και εκτυπώνοντας το μήνυμα (“Executing instruction: διεύθυνση μνήμης ”).
- Σε περίπτωση που η σχετική διεύθυνση οδηγεί σε μη επιτρεπτή θέση μνήμης (<0, ή >M) εκτυπώνεται το μήνυμα (“Access Violation Exception at address: διεύθυνση μνήμης ”) και σταματά η εκτέλεση.

```
Please enter maximum memory address: 100
Please enter the next relative memory address: -10
Please enter the next relative memory address: -20
Please enter the next relative memory address: 30
Please enter the next relative memory address: 0
Please enter the current memory address: 50
Executing instruction: 80
Executing instruction: 60
Executing instruction: 50
```

```
Please enter maximum memory address: 100
Please enter the next relative memory address: 120
Please enter the next relative memory address: -10
Please enter the next relative memory address: 30
Please enter the next relative memory address: 0
Please enter the current memory address: 50
Executing instruction: 80
Executing instruction: 70
Access Violation Exception at address: 190
```

```
Please enter maximum memory address: 100
Please enter the next relative memory address: -100
Please enter the next relative memory address: -20
Please enter the next relative memory address: 30
Please enter the next relative memory address: 0
Please enter the current memory address: 50
Executing instruction: 80
Executing instruction: 60
Access Violation Exception at address: -40
```

15. Η πώληση μετοχών δημιουργεί εν δυνάμει κέρδος, το οποίο φορολογείται ανάλογα με την χώρα όπου διεκπεραιώνεται η συναλλαγή. Για τον υπολογισμό αυτού του κέρδους, λαμβάνονται υπόψη οι τιμές αγοράς και πώλησης των μετοχών. Στις Η.Π.Α., σε περίπτωση που οι μετοχές που πωλούνται έχουν αγοραστεί με διαφορετικές τιμές πώλησης η κάθε μία, τότε θεωρείται ότι πωλούνται πρώτα οι μετοχές που αγοράστηκαν πιο πρόσφατα και στη συνέχεια οι υπόλοιπες (προτεραιότητα Last In First Out – LIFO).
Έστω ότι ένας αγοραστής (Α) αγοράζει από έναν πωλητή (Π) 100 μετοχές της εταιρείας (Ε) στις 1/1/2016, έναντι 10€/μετοχή και ο Π κατέχει 150 μετοχές της Ε. Από τις 150 οι 80 αγοράστηκαν στις 1/12/2015 έναντι 8€/μετοχή ενώ οι υπόλοιπες 70 αγοράστηκαν στις 1/11/2015 έναντι 10€/μετοχή. Τότε, θεωρείται ότι πωλούνται και οι 80 μετοχές (1/12/2015) συν τις άλλες 20 που είχαν αγοραστεί παλαιότερα (1/11/2015). Επομένως, το κέρδος του Π θα είναι
100*10-(80*8+20*10)=160€.
Παρατηρείστε ότι η ακριβής χρονική στιγμή που αγοράστηκαν οι μετοχές δεν έχει σημασία παρά μόνο η σειρά. Αυτή η σειρά μπορεί να μοντελοποιηθεί με τη χρήση στοιβών. Συγκεκριμένα, κάθε φορά που πραγματοποιείται αγορά μετοχών, η ποσότητα και η τιμή τους μπορεί να προστίθεται σε δύο στοίβες (μία για την ποσότητα και μία για την τιμή). Χρησιμοποιώντας την ΑΔΤ Στοίβα, γράψτε πρόγραμμα το οποίο:

a. Θα διαβάζει 3 ακεραίους χωρισμένους με κόμμα την φορά και θα σταματάει εάν οποιοσδήποτε είναι αρνητικός. Η σημασία του κάθε ακεραίου φαίνεται στον παρακάτω πίνακα:

Επιτρεπτές τιμές	0,1	1-1000	1-1000
Σημασία	0-Αγορά 1-Πώληση	Ποσότητα	Τιμή

b. Εάν πρόκειται για αγορά θα εισάγει την τιμή και την ποσότητα στις στοίβες.
c. Εάν πρόκειται για πώληση θα αφαιρεί την σωστή ποσότητα από τις στοίβες και θα υπολογίζει και θα εμφανίζει το κέρδος της συναλλαγής.
d. Στο τέλος (εάν οποιοσδήποτε ακέραιος είναι αρνητικός) θα εμφανίζει το συνολικό κέρδος.

```
Enter the orderType,volume,price: 0,10,15
Transaction profit: 0
Enter the orderType,volume,price: 0,15,30
Transaction profit: 0
Enter the orderType,volume,price: 1,5,5
Transaction profit: -125
Enter the orderType,volume,price: 1,20,25
Transaction profit: 50
Enter the orderType,volume,price: -1
Transaction profit: 0
Total profit: -75
Press any key to continue . . .
```