Object Oriented Programming Course (CEID_NNY106)

Εργαστηριακή Άσκηση

RPN Calculator σε C - Incremental Development

1. Στόχος

A) Εξοικείωση με:
- την Incremental Development τεχνική στην ανάπτυξη λογισμικού,
- την εφαρμογή RPNCalculator την οποία θα αναπτύξουμε στη συνέχεια με βάση την αντικειμενοστρεφή προσέγγιση.

B) Επανάληψη στη C η οποία είναι απαραίτητη για την μετάβαση στην Java.

Reverse Polish Notation Calculator

Η Εργαστηριακή Άσκηση βασίζεται στο παράδειγμα Reverse Polish Notation calculator που χρησιμοποιείται στο κεφάλαιο 8 "Οργάνωση προγράμματος" του βιβλίου «Διαδικαστικός προγραμματισμός – C». Την άσκηση μπορείτε να βρείτε στις παρακάτω πηγές:

1. Διαδικαστικός προγραμματισμός – C, Κ. Θραμπουλίδης, ΕΚΔΟΣΕΙΣ Α. ΤΖΙΟΛΑ & ΥΙΟΙ Α.Ε. (Κεφάλαιο 8 – Οργάνωση Προγράμματος)
2. Η Γλώσσα Προγραμματισμού C, Brian W. Kernighan, Dennis M. Ritchie, 2η/2008, Εκδόσεις ΚΛΕΙΔΑΡΙΘΜΟΣ ΕΠΕ (κεφάλαιο 4)
3. Από τις Υπολογιστικές στις Κυβερνο-Φυσικές Διεργασίες και το IoT: Αντικείμενα και Υπηρεσίες, Κ. Θραμπουλίδης, ISBN 978-960-418-961-8, 2022, ΕΚΔΟΣΕΙΣ Α. ΤΖΙΟΛΑ & ΥΙΟΙ Α.Ε. https://sites.google.com/view/fromcomputationaltocyber-physi/home
4. στην ιστοσελίδα https://sites.google.com/view/objecttechnologycourse/courses-activities/activity-no-0

2. Οδηγίες Εκτέλεσης

Αν δεν είστε ήδη εξοικειωμένοι με την τεχνική της αυξητικής ανάπτυξης (incremental development) προγράμματος **θα πρέπει να το κάνετε το συντομότερο δυνατό**. Η Εργαστηριακή Άσκηση έχει δομηθεί για να σας παρακινήσει να δουλέψετε υιοθετώντας την τεχνική αυτή.

Προσέξτε, η διαδικασία της αυξητικής ανάπτυξης του κώδικα σας δεν εφαρμόζεται μόνο σε επίπεδο εκδόσεων αλλά και στα πλαίσια της ανάπτυξης της κάθε έκδοσης, αν η λειτουργικότητα της έκδοσης είναι σύνθετη.

Με βάση τα παραπάνω σας συνιστώ να **εφαρμόσετε incremental development** για την ανάπτυξη μιας λειτουργούσας έκδοσης του προγράμματος RPNCalculator.

Την Εργαστηριακή Άσκηση θα δουλέψουμε βήμα προς βήμα στην 1η διάλεξη του μαθήματος. Εσείς θα συνεχίσετε την ανάπτυξη της, αξιοποιώντας την incremental development τεχνική, για να έχετε την δική σας λειτουργούσα έκδοση της εφαρμογής.

Προχωρήστε στην ανάπτυξη διαδοχικών εκδόσεων της εφαρμογής όπου η κάθε επόμενη θα προσθέτει επιπλέον λειτουργικότητα σε αυτήν της προηγούμενης έκδοσης.

3. Έκδοση No 1 (V1.0)

Ξεκινήστε με στόχο μια απλή έκδοση που θα υπολογίζει την τιμή πολύ απλών εκφράσεων με μονοψήφιους τελεστέους και με την υπόθεση πως ο χρήστης παρεμβάλει πάντα ένα κενό μεταξύ τελεστών και τελεστέων όπως στην παρακάτω απλή έκφραση

Απλή έκφραση: 3 6 + =

Α) Αξιοποιήστε το procedural abstraction και δουλέψτε σε πρώτη φάση μόνο με την main με βάση τον σκελετό που δώσαμε στην διάλεξη.

Ελέγξτε την σωστή λειτουργία της main πριν προχωρήσετε στην συγγραφή των άλλων συναρτήσεων (functions) της εφαρμογής.

Β) Δώστε την πιο απλή μορφή της getOp() η οποία θα σας επιτρέπει να ελέγξετε την σωστή λειτουργία της main().

Η εκτέλεση μια τέτοιας έκδοσης της εφαρμογής για είσοδο 3 6 + = θα μπορεί για παράδειγμα να έχει ως έξοδο την παρακάτω

```
RPNCalc started
putInStack
putInStack
add
presentResult
```

4. Έκδοση No 2 (V2.0)

Α) Αφού επιβεβαιώσετε τη σωστή λειτουργία της main(), προχωρήστε στην συγγραφή μιας getOp() που θα είναι ικανή να δίνει στην main, εκτός από τον τύπο της επόμενης συνιστώσας της έκφρασης, και τον τελεστέο. Έτσι η main θα έχει στην διάθεση της τους τελεστέους και θα μπορεί να κάνει την πράξη.

Ελέγξτε τη λειτουργία της καθώς και την ορθή επικοινωνία της με την main().

Β) Προχωρήστε αναπτύσσοντας την πιο απλή στοίβα που θα σας δώσει τη δυνατότητα να έχετε μια λειτουργούσα έκδοση που θα υπολογίζει την τιμή της απλής έκφρασης μας.

Η εκτέλεση μια τέτοιας έκδοσης της εφαρμογής για είσοδο 3 6 + = θα μπορεί για παράδειγμα να έχει ως έξοδο την παρακάτω

```
RPNCalc started
3 6 + =
putInStack 3
putInStack 6
adding
```

```
res= 9
```

Η παρακάτω ενδεικτική έξοδος δίνει περισσότερη πληροφορία για τον τρόπο εκτέλεσης της εφαρμογής

```
Rpn Calculator V1
getOp() running
3 6 + =
input =3        51      inputType = 1
putInStack() executed
getOp() running
input =         32      inputType = 32
getOp() running
input =6        54      inputType = 1
putInStack() executed
getOp() running
input =         32      inputType = 32
getOp() running
input =+        43      inputType = 43
add() executed
6       3
getOp() running
input =         32      inputType = 32
getOp() running
input ==        61      inputType = 61
getOp() running
input =
        10      inputType = 10
getOp() running
```

Δώστε την εξήγηση σας για την μορφή της παραπάνω εξόδου.

Στην συνέχεια έχετε δύο επιλογές.
Η πρώτη είναι να υλοποιήσετε τις εκδόσεις 2.1 και 2.2 και στην συνέχεια να προχωρήσετε στην έκδοση 3.
Εναλλακτικά αν δεν θέλετε να δώσετε πολύ χρόνο στην Εργαστηριακή Άσκηση προχωρήστε στην έκδοση 3.0. Στην περίπτωση αυτή θα σας είναι πιο δύσκολο να κατανοήσετε την έκδοση της getOp των K&R.

5. Έκδοση No 2.1 (V2.1)

A) Ορίστε ως μέγεθος στοίβας το 2 και δώστε την έκφραση 3 4 5 ++=. Παρατηρήστε την συμπεριφορά του προγράμματος σας. Δοκιμάστε την έκφραση 1 2 3 4 5 ++++=.

B) Δώστε την έκφραση 3 += και παρατηρήστε την συμπεριφορά του προγράμματος σας.

C) Με βάση τις παραπάνω παρατηρήσεις βελτιώστε τον κώδικα των push και pop ώστε να μην δουλεύουν έξω από τα όρια της στοίβα σας.

D) Τροποποιήστε την getOp ώστε να δέχεται τελεστέους με περισσότερα από ένα ψηφία.

6. Έκδοση No 2.2 (V2.2)

A) Στην βελτιωμένη έκδοση δώστε τώρα την έκφραση 31 421+=. Η έκφραση παραβιάζει την υπόθεση που είχαμε κάνει στην αρχή, δηλαδή πως θα πρέπει να υπάρχει κενό μεταξύ τελεστέου και τελεστή. Παρατηρήστε την έξοδο του προγράμματος σας. Το αποτέλεσμα λογικά θα είναι 421, το οποίο φυσικά είναι λάθος. Αιτιολογήστε το αποτέλεσμα αυτό με βάση τον κώδικα σας. Στο τέλος της άσκησης θα βρείτε την αιτιολόγηση.

B) Προχωρήστε στην τροποποίηση της getOp ώστε να μπορεί να υπολογίζει σωστά την τιμή της έκφρασης και στην περίπτωση που δεν υπάρχει κενό μεταξύ τελεστέου και τελεστή. Στο τέλος της άσκησης θα βρείτε την πρόταση μας.

7. Έκδοση No 2.3 (V2.3)

Δουλέψτε πάνω σε ένα αντίγραφο μιας από τις εκδόσεις V2.0 – V2.2 για να τροποποιήσετε τον κώδικα σας ώστε να επιτρέπεται στον προγραμματιστή να δημιουργεί και να χρησιμοποιεί περισσότερες από μία στοίβες. Εξετάσετε εναλλακτικές υλοποιήσεις. Δοκιμάστε οπωσδήποτε την χρήση της κατασκευής struct της C.

Στην ενότητα ενδεικτικές απαντήσεις θα βρείτε ορισμένες προτάσεις υλοποίησης. Για κάθε μία από αυτές ελέγξτε αν είναι αποδεκτή αιτιολογώντας ταυτόχρονα την απάντηση σας, είτε είναι αυτή θετική, είτε αρνητική.

8. Έκδοση No 3 (V3.0)

A) Για την έκδοση 3 χρησιμοποιήστε ως βάση την τελευταία από τις εκδόσεις V2.0 – V2.2 που έχετε υλοποιήσει.

B) Αντικαταστήστε την δική σας getOp() με την K&R getOp την οποία θα βρείτε στις διαφάνειες ή στις πηγές που σας δόθηκαν στο πλαίσιο και προσπαθήστε να κατανοήσετε τον κώδικα και την λειτουργικότητα της. Αν έχετε υλοποιήσει τις εκδόσεις 2.1 και 2.2 αυτό θα σας είναι εύκολο.

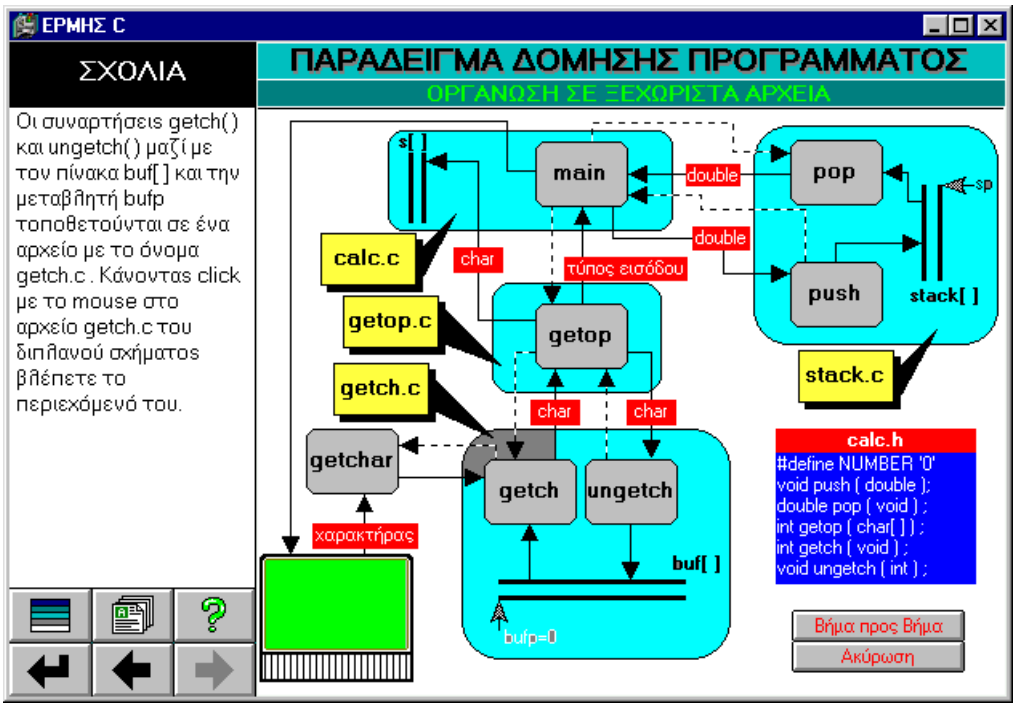
Γ) Τροποποιήστε την getOp() ώστε να χρησιμοποιεί την getchar() αντί της getch(). Δώστε την έκφραση 12 10+= (δεν υπάρχει κενό μετά το 10) και παρατηρήστε την λειτουργικότητα της getOp(). Αυτό θα σας βοηθήσει να κατανοήσετε το λόγο χρήσης των getch() και ungetcth() από τους K&R (αυτό για την περίπτωση που δεν υλοποιήσατε τις εκδόσεις 2.1 και 2.2).

9. Έκδοση No 4 (V4.0)

Κάντε μια προσπάθεια να δομήσετε το πρόγραμμα σας σε περισσότερα του ενός αρχεία.

Η παρακάτω εικόνα από το κεφάλαιο 8 του βιβλίου «Διαδικαστικός προγραμματισμός – C» δίνει ένα σενάριο δόμησης του κώδικα σε αρχεία. Είναι μια καλή άσκηση για οργάνωση

προγράμματος. Το πρόγραμμα ΕΡΜΗΣ C μπορείτε να βρείτε στο συνοδευτικό CD του βιβλίου.



10. Έκδοση No 5 (V5.0)

Τροποποιήστε την εφαρμογή σας ώστε να υπολογίζει την τιμή εκφράσεων με τελεστέους δεκαεξαδικούς αριθμούς.

Ίσως στο τέλος νοιώσετε όπως ο ναυαγός που ανάβει φωτιά στο νησί.

11. Ενδεικτικές Απαντήσεις

9.1 Έκδοση No 1

A) Ενδεικτική δόμηση της main αξιοποιώντας procedural abstraction
Προσέξτε πως αποφύγαμε την πρόταση κλήσης της getOp() δύο φορές, μία πριν την πρόταση while (γρ. 19) και μία μέσα στο σώμα της (γρ. 41) τροποποιώντας την αρχική while (γρ. 20) σε αυτή της γραμμής 21.

```
17  int main(int argc, char *argv[]) {
18      printf("RPN Calculator\n");
19  //  inputType=getOp();
20  //  while(inputType != EOF){
21      while((inputType=getOp())!=EOF){
22  //      printf("inputType= %c(as char)\t %d(as decimal)\n",
23  //              inputType,inputType);
24          switch(inputType){
25              case NUMBER:
26                  putInStack();
27                  break;
28              case '+':
29                  add();
30                  break;
31              case '*':
32                  mul();
33                  break;
34              case '=':
35                  presentResult();
36                  break;
37              default:
38                  printf("operation not suported\n");
39
40          }
41  //      inputType=getOp();
42      }
```

Την printf (γρ. 22-23) μπορείτε να προσθέσετε για να παρατηρήσετε την επιστρεφόμενη τιμή της getOp. Αυτό θα σας βοηθήσει να την βελτιώσετε στη συνέχεια ώστε να επιστρέφει στην main μόνο χρήσιμη πληροφορία και όχι τα κενά και τους χαρακτήρες \n.

B) Απλή μορφή της getOp() η οποία θα σας επιτρέπει να ελέγξετε την σωστή λειτουργία της main().

```
45  int getOp(void){
46      char ch;
47      printf("getOp\n");
48      ch=getchar();
49      if(isdigit(ch))
50          return NUMBER;
51      else
52          return ch;
53  }
```

9.2 Έκδοση No 2

A) Δηλώνουμε σε πρώτη φάση μια γενική μεταβλητή operand τύπου αλφαριθμητικού ώστε η getOp να βάζει μέσα σε αυτή τα ψηφία του τελεστέου (ένα μόνο για την απλή μας έκφραση, περισσότερα στη συνέχεια). Έτσι καθώς η operand είναι γενική μεταβλητή έχουν πρόσβαση σε αυτή και η getOp και η main.
Τροποποιούμε την getOp ώστε να αποθηκεύει το ψηφίο του τελεστέου στην πρώτη θέση του πίνακα χαρακτήρων operand.
Η παρακάτω έκδοση της getOp καταναλώνει τα κενά και τους χαρακτήρες \n καθώς αυτοί δεν ενδιαφέρουν την main() στην λειτουργία της. Αντικαταστήστε τις προτάσεις 59-61 με μία πρόταση while για να δώσετε συμπαγή κώδικα.

```
56 int getOp(void){
57     char ch;
58     printf("getOp\n");
59     ch=getchar();
60     while(ch==' '|| ch=='\n')
61         ch=getchar();
62     if(isdigit(ch)){
63         operand[0]=ch;
64         operand[1]='\0';
65         return OPERAND;
66     }
67
68     else
69         return ch;
70 }
```

B) Παρακάτω δίνεται μια ενδεικτική πρώτη υλοποίηση της στοίβας η οποία ικανοποιεί τις απαιτήσεις της έκδοσης αυτής.

```
19 int stack[50];
20 int sp=0;
```

```
77 void push(int n){
78     stack[sp]=n;
79     sp++;
80 }
81
82 int pop(void){
83     sp--;
84     return stack[sp];
85 }
```

Μπορείτε να αξιοποιήσετε πιο αποτελεσματικά τους τελεστές ++ και -- και να διαμορφώσετε το σώμα των push και pop να αποτελείται από μία μόνο πρόταση.

9.3 Έκδοση No 2.1

A) Το πρόγραμμα σας με την έκφραση 3 4 5 ++=ως είσοδο αποθηκεύει στην στοίβα τους τρεις τελεστέους δηλαδή το 3 το 4 και το 5 και στην συνέχεια εκτελεί 2 φορές την πράξη της πρόσθεσης. Τέλος δίνει το αποτέλεσμα. Προσέξτε όμως, η στοίβα σας έχει μέγεθος 2, άρα έχει δεσμευθεί χώρος για 2 στοιχεία. Εξηγήστε την συμπεριφορά του προγράμματος σας. Δοκιμάστε την έκφραση 1 2 3 4 5 ++++=.

B) Για είσοδο την 3 += ο τελεστής + έχει ως αποτέλεσμα να κληθεί η pop() δύο φορές. Τι σας επιστρέφει την 2η φορά καθώς η στοίβα σας έχει μόνο ένα στοιχείο, το 3; Παρατηρήστε το αποτέλεσμα που σας δίνεται για να αιτιολογήσετε.

C) Ενδεικτική υλοποίηση της στοίβας δίνεται παρακάτω

```
103 void push(int n){
104     if(sp>=STACK_SIZE)
105         printf("Stack full\n");
106     else
107         stack[sp++]=n;
108 //  sp++;
109 }
110
111 int pop(void){
112 //  sp--;
113 if(sp==0)
114     printf("stack empty\n");
115 else
116     return stack[--sp];
117 }
```

D) Παρακάτω δίνεται ενδεικτική υλοποίηση της getOp για τελεστέους με περισσότερα από ένα ψηφία.

```
59 int getOp(char op[]){
60     char ch;
61     int i;
62 //  printf("Enter RPN expression\n");
63 //  ch=getchar();
64 //  while(ch==' '|| ch=='\n')
65 //      ch=getchar();
66     while((ch=getchar())==' '|| ch=='\n');
67     if(isdigit(ch)){
68         for(i=0;isdigit(ch);i++){
69             op[i]=ch;
70             ch=getchar();
71         }
72         op[i]='\0';
73         return OPERAND;
74     }
75     else
76         return ch;
77 }
```

9.4 Έκδοση No 2.2

A) Η getOp καταναλώνει τον χαρακτήρα + για να προσδιορίσει το τέλος του τελεστέου. Αυτό έχει ως αποτέλεσμα όταν ξανακληθεί από την main να επιστρέψει τον επόμενο χαρακτήρα που είναι το =. Αυτό έχει ως αποτέλεσμα να έχουμε ως έξοδο την τιμή του τελευταίου τελεστέου που μπήκε στην στοίβα.

B) Τροποποιούμε την getOp ώστε να αποθηκεύει τον χαρακτήρα που χρησιμοποίησε για να αναγνωρίσει το τέλος του τελεστέου. Αυτό μπορούμε να το κάνουμε με μία συνάρτηση myUngetch() η οποία θα καλείται από την getOp ώστε να αποθηκεύσει αυτή τον χαρακτήρα σε ένα buffer. Ως αποτέλεσμα τώρα η getOp δεν θα πρέπει να καλεί την getchar αλλά μία άλλη συνάρτηση που την ονομάζουμε myGetch(). Αυτή αν υπάρχει από την προηγούμενη κλήση της getOp χαρακτήρας τον επιστρέφει αλλιώς καλεί την getchar και επιστρέφει την τιμή που αυτή επιστρέφει. Ενδεικτικός κώδικας δίνεται στη συνέχεια για τις myGetch() και myUngetch() και το πως η χρήση τους τροποποιεί την getOp().

```
61  int getOp(char op[]){
62      char ch;
63      int i;
64
65  //  while((ch=getchar())==' '|| ch=='\n');
66      while((ch=myGetch())==' '|| ch=='\n');
67      if(isdigit(ch)){
68          for(i=0;isdigit(ch);i++){
69              op[i]=ch;
70  //          ch=getchar();
71              ch=myGetch();
72          }
73          myUngetch(ch);
74          op[i]='\0';
75          return OPERAND;
76      }
77      else
78          return ch;
79  }
```

```
84  char buf[10];
85  int bufp=0;
86
87  void myUngetch(char ch){
88      buf[bufp++]=ch;
89  }
90
91  char myGetch(void){
92      if(!bufp)
93          return getchar();
94      else
95          return buf[--bufp];
96  }
```

9.5 Έκδοση No 2.3

Περιγράφονται στη συνέχεια τέσσερεις προτάσεις υλοποίησης της στοίβας για την υποστήριξη δημιουργίας περισσοτέρων από μίας στοιβών στο ίδιο πρόγραμμα. Για κάθε μία από αυτές ελέγξτε αν είναι αποδεκτή αιτιολογώντας ταυτόχρονα την απάντηση σας, είτε είναι αυτή θετική, είτε αρνητική.

Πρόταση 1

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  #define  STACK_SIZE 20
 5
 6  void push(int stack[],int sp,int n);
 7  int pop(int stack[],int sp);
 8
 9  int stack[STACK_SIZE];
10  int sp=0;
11
12  int stack1[STACK_SIZE];
13  int sp1=0;
14
15 ⊞ int main(int argc, char *argv[]) {
27
28 ⊞ void push(int stack[],int sp,int n){
36
37 ⊞ int pop(int stack[],int sp){
```

Πρόταση 2

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  #define  STACK_SIZE 20
 5
 6  void push(int stack[],int *spPtr,int n);
 7  int pop(int stack[],int *spPtr);
 8
 9  int stack[STACK_SIZE];
10  int sp=0;
11
12  int stack1[STACK_SIZE];
13  int sp1=0;
14
15  int main(int argc, char *argv[]) {
37
38  void push(int stack[],int *spPtr,int n){
47
48  int pop(int stack[],int *spPtr){
```

Πρόταση 3

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  #define  STACK_SIZE 20
 5
 6  typedef struct stack{
 7      int item[50];
 8      int sp;
 9  } Stack;
10
11  void push(Stack st,int n);
12  int pop(Stack st);
13
14  Stack st1, st2;
15
16  int main(int argc, char *argv[]) {
33
34  void push(Stack st,int n){
42
43  int pop(Stack st){
```

Πρόταση 4

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   #define  STACK_SIZE 20
5
6
7   typedef struct stack {
8       int item[50];
9       int sp;
10  } Stack;
11
12  void push(Stack *stPtr,int n);
13  int pop(Stack *stPtr);
14
15  Stack st1, st2;
16
17  int main(int argc, char *argv[]) {
45
46  void push(Stack *stPtr,int n){
54
55  int pop(Stack *stPtr){
```