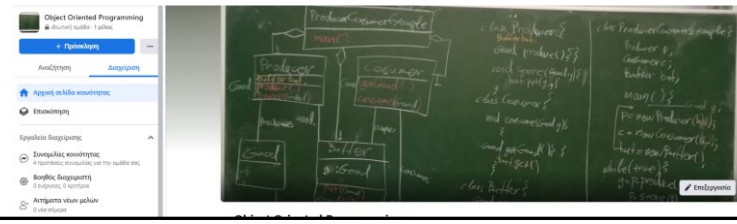
Αντικειμενοστρεφής Προγραμματισμός (Object-Oriented Programming)

(CEID_NNY106)

Προαπαιτούμενα: Functions and Program Structure (Δραστηριότητα No 1)

FB Group
https://www.facebook.com/groups/5894865063882886/



Kleanthis Thramboulidis
Prof. of Software and System Engineering
University of Patras
https://sites.google.com/site/thramboulidiskleanthis/

Εργαστηριακή Άσκηση No 1 – Στόχος

- Να θυμηθούμε την C.
- Να δούμε την διαδικασία ανάπτυξης λογισμικού.
- Να δούμε τους μηχανισμούς της C για χειρισμό πολυπλοκότητας.
- Να δούμε το πέρασμα στην ΟΟ προσέγγιση.

Object Oriented Programming Course (CEID_NNY106)
Εργαστηριακή Άσκηση
RPN Calculator σε C - Incremental Development

1. Στόχος

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

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

Reverse Polish Notation Calculator

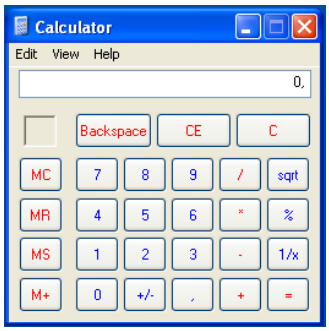
Η Εργαστηριακή Άσκηση βασίζεται στο παράδειγμα Reverse Polish Notation calculator που χρησιμοποιείται στο κεφάλαιο 8 "Οργάνωση προγράμματος" του βιβλίου «Διαδικαστικός προγραμματισμός – C». Την άσκηση μπορείτε να βρείτε στις παρακάτω πηγές:
1. Διαδικαστικός προγραμματισμός – C, Κ. Θραμπουλίδης, ΕΚΔΟΣΕΙΣ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

Εργαστηριακή Άσκηση No 1 – RPN Calculator

- Develop a program for the system to accept expressions based on the **Reverse Polish Notation** (RPN) and calculate their values.



free online calculator
http://www.math.com/students/calculators/source/scientific.htm

- Design Constraint
  - Stack has to be used to store operands.

Reverse Polish Notation (RPN)

Reverse Polish notation (**RPN**), also known as (…,) **Polish postfix notation** or simply **postfix notation**, is a mathematical notation in which operators *follow* their operands, in contrast to Polish notation (PN), in which operators *precede* their operands. It does not need any parentheses as long as each operator has a fixed number of operands.

3 4 +
Postfix notation
("Reverse Polish")

In reverse Polish notation, the operators follow their operands; for instance, to add 3 and 4 together, one would write 3 4 + rather than 3 + 4. If there are multiple operations, operators are given immediately after their final operands (often an operator takes two operands, in which case the operator is written after the second operand); so the expression written 3 − 4 + 5 in conventional notation would be written 3 4 − 5 + in reverse Polish notation: 4 is first subtracted from 3, then 5 is added to it.
The concept of a stack, a last-in/first-out construct, is integral to these actions.

Source: Wikipedia

(10+6) * (18 - 16) ⟺ **10 6 + 18 16 - ***

Στοίβα (Stack) – Last In First Out (LIFO)

Stack of dishes
adding or removing is only possible at the top.

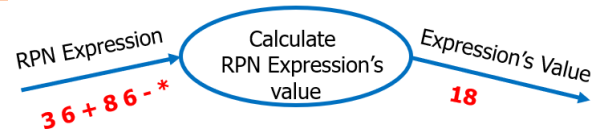
Stack (abstract data type)
In computer science, a **stack** is an abstract data type that serves as a collection of elements, with two main operations:

- **Push**, which adds an element to the collection, and
- **Pop**, which removes the most recently added element that was not yet removed.

Source: Wikipedia

Development process

- Develop a program for the system **to accept expressions** based on the Reverse Polish Notation (RPN) and **calculate their values.**



Να γράψω την main()?

Procedural Abstraction
Identify basic processes

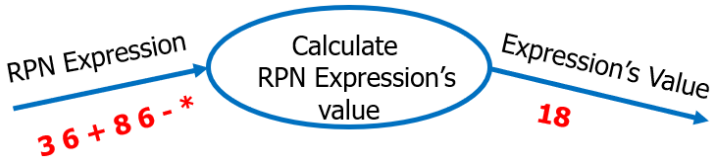
?

Data Abstraction
Data representation

Structured English of basic processes

```
main(){
....
...
}
```

Identify basic processes



1. Με ποια σειρά εκτελούνται οι συναρτήσεις;
2. Πως επικοινωνούν μεταξύ τους;

Βασικές διεργασίες

- Πάρε επόμενη συνιστώσα εισόδου (τελεστέο, τελεστή ή τέλος προγράμματος)
- Βάλε τελεστέο στην στοίβα
- Πάρε τελεστέο από την στοίβα
- Κάνε πρόσθεση
- Κάνε αφαίρεση
- Κάνε πολλαπλασιασμό
- Κάνε διαίρεση
- Δώσε το αποτέλεσμαστον χρήστη

Functions

- getOp()
- putInStack()
- getFromStack()
- add()
- sub()
- mul()
- div()
- presentResult()

main()

```
main(){
....
...
}
```


Structure English of program functions

main()

While the **next input** is not an indication of program termination
  In case it is a number
    put it in stack
  in case of +
    add
  in case of -
    sub
  ..
  in case of =
    present Result
  otherwise
    display error message.

add()
Get operands from stack
Apply the + operator
Put the result in stack

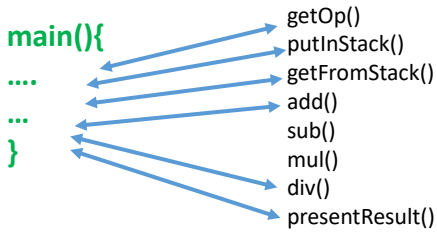
sub()
Get operands from stack
Apply the - operator
Put the result in stack

presentResult()
Get operand from stack
Display it

Get next input

getOp() function

- **What?**
- **How?**
- Consider its **communication** with other functions of the program.



What? Δήλωση Συνάρτησης

- It has to be defined in the function declaration (Function prototype)

```
int getOp(char input[]); ή
int getOp(char *input);
```

How? Ορισμός Συνάρτησης

- It is defined in function definition

Structure of main( )

1

```
inputType = getop();
while( inputType != EOF) {
   switch(inputType) {
         :
         :
         }
   inputType = getop();
   }
```

Does not exploit the flexibility of C

2

```
inputType = getop();
while( inputType != EOF) {
   switch(inputType) {
         :
         :
         }
   inputType = getop();
   }
```

3

```
while( (inputType = getop()) != EOF) {
   switch(inputType) {
         .....
         }
   }
```

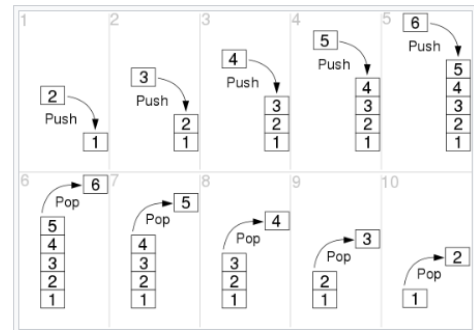
compact C code

Structure of main( )

```
while( (inputType = getop()) != EOF) {
   switch(inputType) {
        case NUMBER :
                putInStack();
                break;
        case '+':
                add();
                break;
        case '-':
                sub();
                break;
        .
        .
        default :
                printf("Λάθος : άγνωστη εντολή\n");
   }
}
```


Στοίβα (Stack) – Last In First Out (LIFO)

Stack of dishes
adding or removing is only possible at the top.



Stack (abstract data type)
In computer science, a **stack** is an abstract data type that serves as a collection of elements, with two main operations:

- **Push**, which adds an element to the collection, and
- **Pop**, which removes the most recently added element that was not yet removed.

Source: Wikipedia

Stack

```
10  int stack[50];
11  int sp=0;
```

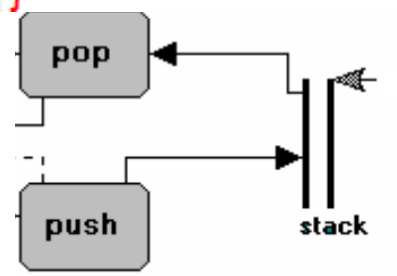
Κώδικας για τοποθέτηση στοιχείου στην στοίβα

```
stack[sp++]=op;
```

Κώδικας για αφαίρεση στοιχείου από την στοίβα

```
stack[--sp];
```

```
63  int pop(void){
64      return stack[--sp];
65  }
```




```
59  void push(int op){
60      stack[sp++]=op;
61  }
```

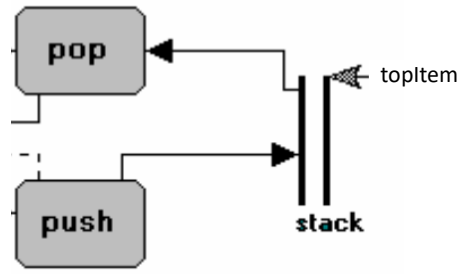

Using stack

```
add()
{
  int op1, op2, res;
  op1 = pop();
  op2 = pop();
  res = op1 + op2;
  push(res);
}
```

```
add()
{
  int op1, op2;
  op1 = pop();
  op2 = pop();
  push(op1+op2);
}
```

```
add()
{
  push(pop()+pop());
}
```

compact C code



getOp( )

```
int getOp(char input[] )
{
int i,c;

while((input[0]=c=myGetch())==' '|| c=='\t');
input[1]='\0';
if (!isdigit(c) && c!='.')
    return c;
i=0;
if (isdigit(c))
    while (isdigit(input[++i]=c= myGetch()));
if (c=='.')
    while (isdigit(input[++i]=c= myGetch()));
input[i]='\0';
if (c!=EOF)
   myUnGetch(c);
return NUMBER;
}
```

Why isn't getchar() used instead?

Χρησιμοποίησε την getchar() και δες την απόκριση του προγράμματος στην είσοδο
12 21+=

Program Structure - 1

```
17  //char inputType;
18  //char operand[20];
19  int stack[50];
20  int sp=0;
21  char buf[20];
22  int bufp=0;
23
24  int main(int argc, char *argv[]) {
48
49  char getOp(char in[]){
61
62  void putInStack(char *in){
67
68  void add(void){
72
73  void displayResult(void){
```

```
77  void push(int i){
78      stack[sp++]=i;
79  }
80
81  int pop(void){
82      return stack[--sp];
83  }
84
85  char myGetch(void){
88
89  void myUnGetch(char c){
```

Εντοπίστε προβλήματα

Program Structure - 2

```
17  //char inputType;
18  //char operand[20];
19
20  int main(int argc, char *argv[]) {
44
45  char getOp(char in[]){
57
58  void putInStack(char *in){
63
64  void add(void){
68
69  void displayResult(void){
72
73  int stack[50];
74  int sp=0;
75
76  void push(int n){
```

Εντοπίστε προβλήματα

```
73  int stack[50];
74  int sp=0;
75
76  void push(int n){
77      stack[sp++]=n;
78  }
79
80  int pop(void){
81      return stack[--sp];
82  }
83
84  char buf[20];
85  int bufp=0;
86
87  char myGetch(void){
90
91  void myUnGetch(char c){
```


Program Structure - 3

```
17  //char inputType;
18  //char operand[20];
19
20
21  int main(int argc, char *argv[]) {
45
46  char getOp(char in[]){
58
59  void putInStack(char *in){
64
65  void add(void){
69
70  void displayResult(void){
73
74  char buf[20];
75  int bufp=0;
76
77  char myGetch(void){
80
```

Εντοπίστε προβλήματα

```
70  void displayResult(void){
73
74  char buf[20];
75  int bufp=0;
76
77  char myGetch(void){
80
81  void myUnGetch(char c){
84
85  int stack[50];
86  int sp=0;
87
88  void push(int n){
91
92  int pop(void){
95
```

Program Structure – 4

RPNCalculator.c

```
 7  char getOp(char in[]);
 8  void putInStack(char *in);
 9  void add(void);
10  void displayResult(void);
11  void push(int n);
12  int pop(void);
13  char myGetch(void);
14  void myUnGetch(char c);
15
16  //char inputType;
17  //char operand[20];
18
19  int main(int argc, char *argv[]) {
43
44  char getOp(char in[]){
62
63  void putInStack(char *in){
68
69  void add(void){
73
74  void displayResult(void){
```

stack.c

```
1   static int stack[50];
2   static int sp=0;
3
4   void push(int n){
7
8   int pop(void){
```

getOp.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   static char buf[20];
5   static int bufp=0;
6
7   char myGetch(void){
11
12  void myUnGetch(char c){
```

Δώστε πρόταση βελτίωσης

Program Structure – 5

RPNCalculator.c

```
 7  char getOp(char op[]);
 8  void putInStack(char *in);
 9  void add(void);
10  void displayResult(void);
11  void push(int n);
12  int pop(void);
13  //char myGetch(void);
14  //void myUnGetch(char c);
15
16  //char inputType;
17  //char operand[20];
18
19  int main(int argc, char *argv[]) {
43
44  void putInStack(char *in){
50
51  void add(void){
55
56  void displayResult(void){
```

stack.c

```
1   static int stack[50];
2   static int sp=0;
3
4   void push(int n){
7
8   int pop(void){
```

[*] getOp.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   #define OPERAND 1
5
6   static char myGetch(void);
7   static void myUnGetch(char c);
8
9   static char buf[20];
10  static int bufp=0;
11
12  char getOp(char op[]){
30
31  static char myGetch(void){
35
36  static void myUnGetch(char c){
```

Δώστε πρόταση βελτίωσης

Program Structure – 6

RPNCalculator.c | stack.c | getOp.c

```
 7   char getOp(char op[]);
 8   void putInStack(char *in);
 9   void add(void);
10   void displayResult(void);
11   void push(int n);
12   int pop(void);
13   //char myGetch(void);
14   //void myUnGetch(char c);
15
16   //char inputType;
17   //char operand[20];
18
19 ⊞ int main(int argc, char *argv[]) {
43
44 ⊞ void putInStack(char *in){
50
51 ⊞ void add(void){
55
56 ⊞ void displayResult(void){
```

RPNCalculator.c | stack.c | getOp.c

```
1   static int stack[50];
2   static int sp=0;
3
4 ⊞ void push(int n){
7
8 ⊞ int pop(void){
```

RPNCalculator.c | stack.c | getOp.c

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   #define OPERAND 1
5
6   static char myGetch(void);
7   static void myUnGetch(char c);
8
9 ⊞ char getOp(char op[]){
27
28  static char buf[20];
29  static int bufp=0;
30
31 ⊞ static char myGetch(void){
35
36 ⊞ static void myUnGetch(char c){
```

Δώστε πρόταση βελτίωσης με χρήση header files

Handling Complexity

- Visibility of variables (**Scope of Variables**)
  - global, local
- Visibility of functions (function scope)
- Storage classes in C
  - auto, register, static and extern

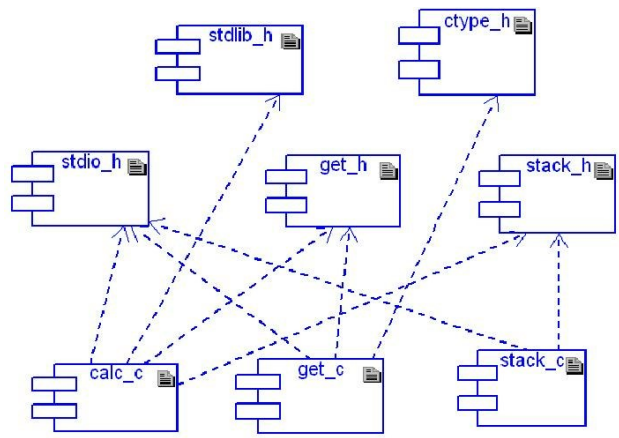
Storage Classes in C

- **auto**
  - is the default storage class for all local variables.
- **register**
  - is used to define local variables that should be stored in a register instead of RAM.
- **static**
  - is the default storage class for global variables.
  - can be 'seen' in the source file.
  - At link time, will not be seen by the other object modules.
- **extern**
  - is used to declare the use of a global variable defined in **another file.**

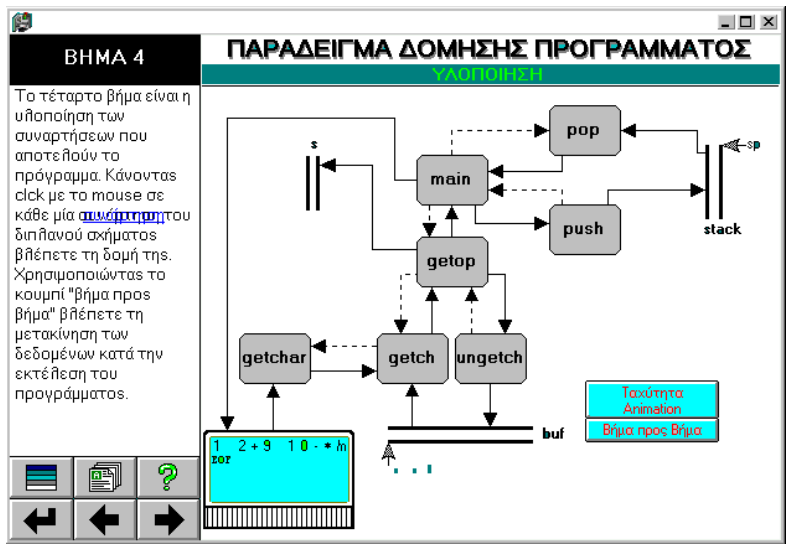
Program structure

- **File** is the basic mechanism of C to organize the program in a higher level of abstraction than the one provided by C constructs.

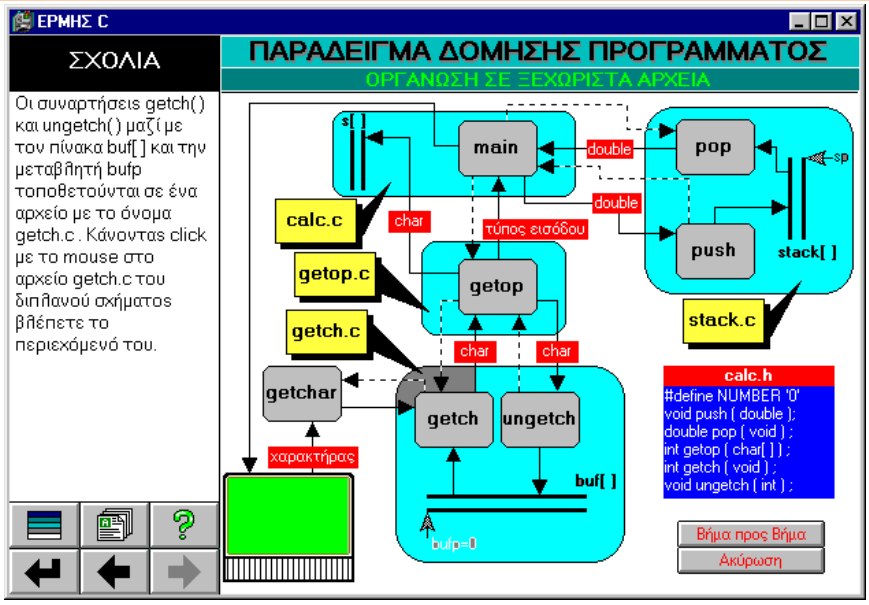
UML
component diagram



Program representation



Example of program structure



RPN Calculator in Java (student's implementation)