



Προγραμματισμός Υπολογιστών

**Καθολικές – Τοπικές – Στατικές Μεταβλητές
Χρήση #define για συναρτήσεις**

Νικόλαος Ζ. Ζάχαρης

Καθηγητής

Πανεπιστήμιο Δυτικής Αττικής

Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Τοπικές μεταβλητές

```
Project Classes Debug
Project1

main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int prod2Numbers(int a, int b){
5      int apot;
6      apot = a * b;
7      return apot;
8  }
9
10 int sum2Numbers(int a, int b){
11     int apot;
12     apot = a + b;
13     return apot;
14 }
15
16
17 int main(int argc, char *argv[]) {
18     int sum = sum2Numbers(4,3);
19     printf("sum = %d  apot = %d\n", sum , apot);
20
21 }
22
```

Τα ίδια ονόματα μεταβλητών μπορούν να χρησιμοποιηθούν όσες φορές θέλουμε σε διαφορετικές συναρτήσεις και λειτουργίες.

Οι τοπικές μεταβλητές έχουν εμβέλεια μόνο μέσα στο σώμα της συνάρτησης ή λειτουργίας και όχι έξω από αυτές.

Line	Col	File	Message
		D:\Temp\static\main.c	In function 'main':
19	41	D:\Temp\static\main.c	[Error] 'apot' undeclared (first use in this function)
19	41	D:\Temp\static\main.c	[Note] each undeclared identifier is reported only once for each function it appears in
28		D:\Temp\static\Makefile.win	recipe for target 'main.o' failed

Καθολικές μεταβλητές

```
main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int sum = 0;
5
6  void testA() {
7      printf("TestA sum = %d\n", sum);
8      sum += 5;
9  }
10
11 void testB() {
12     printf("TestB sum = %d\n", sum);
13     sum += 10;
14 }
15
16 void testC() {
17     int sum = 7;
18     printf("TestC sum = %d\n", sum);
19     sum += 5;
20 }
21
22 int main(int argc, char *argv[]) {
23     sum = 5;
24     testA();
25     testB();
26     testC();
27     printf("sum = %d\n", sum);
28     return 0;
29 }
```

Όλες οι μεταβλητές που ορίζονται εκτός του σώματος οποιασδήποτε συνάρτησης ή λειτουργίας ονομάζονται καθολικές μεταβλητές και έχουν εμβέλεια σε όλο το πρόγραμμα.

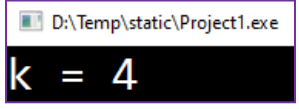
Επιτρέπεται μέσα στο σώμα μιας συνάρτησης ή λειτουργίας, μία τοπική μεταβλητή να έχει το ίδιο όνομα με μία καθολική μεταβλητή. Τότε στην περίπτωση που εκτελείται αυτή η συνάρτηση ή λειτουργία, θεωρούμε ότι όλες οι αλλαγές στη τιμή συμβαίνουν στην τοπική μεταβλητή.

D:\Temp\static\Project1.exe

```
TestA sum = 5
TestB sum = 10
TestC sum = 7
sum = 20
```

Στατικές μεταβλητές

```
[*] main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void DemoStatic(int x) {
5      static int k = 0;
6
7      if(x == 0){
8          printf("k = %d\n", k);
9      }
10     k++;
11 }
12
13 int main(int argc, char *argv[]) {
14     DemoStatic(1);
15     DemoStatic(1);
16     DemoStatic(1);
17     DemoStatic(1);
18     DemoStatic(0);
19     return 0;
20 }
```



Οι μεταβλητές που δηλώνονται μέσα στο σώμα μιας λειτουργίας ή συνάρτησης σαν static διατηρούν τη τιμή ανάμεσα στις διαδοχικές κλήσεις τους

Κατά αυτό τον τρόπο μπορούμε να γνωρίζουμε πόσες φορές έγινε κλήση της συνάρτησης.

Χρήση του #define για συναρτήσεις

```
main.c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define Min(a, b) (a < b) ? a : b
5  #define Sum(a, b) a + b
6  #define Pwr2(a)  a * a
7
8  int main(int argc, char *argv[]) {
9
10     printf("Min = %d\n", Min(2,3));
11     printf("Sum = %d\n", Sum(2,3));
12     printf("Pwr = %d\n", Pwr2(3));
13     printf("Pwr = %d\n", Pwr2(3+2));
14
15     return 0;
16 }
```

```
Min = 2
Sum = 5
Pwr = 9
Pwr = 11
```

Μπορούμε να χρησιμοποιήσουμε τη δήλωση #define για τον ορισμό μιας συνάρτησης

Προσοχή όμως στα ορίσματα γιατί το #define αντικαθιστά το σημείο της κλήσης με τις εντολές που έχουμε ορίσει και μπορεί να προκύψουν απρόσμενα αποτελέσματα.

Για παράδειγμα στη γραμμή 12 το αποτέλεσμα είναι 9 και είναι σωστό. Όμως στην επόμενη γραμμή το αποτέλεσμα είναι 11 αλλά δεν είναι το αναμενόμενο. Ο λόγος είναι ότι το σημείο κλήσης `Pwr2(3+2)` έγινε αντικατάσταση με τις εντολές :

$$3+2 * 3 + 2 = 3 + 6 + 2 = 11$$

Για να πάρουμε σαν αποτέλεσμα το 25 θα πρέπει να ξαναγράψουμε τη Pwr2 όπως παρακάτω :

```
#define Pwr2(a) (a) * (a)
```