Θεματική ενότητα 6:
Συναρτήσεις

Αρθρωτός Σχεδιασμός – Συναρτήσεις (modular design - functions)

- **Βασική ιδέα της επιστήμης των υπολογιστών:**

Τμηματοποίηση

Τεμαχισμός μεγάλων και σύνθετων προβλημάτων σε μικρά, απλά τμήματα.

- **Στη C αυτά τα τμήματα ονομάζονται συναρτήσεις (functions): Αποτελούν αυτόνομες, επώνυμες μονάδες κώδικα, σχεδιασμένες να επιτελούν συγκεκριμένο έργο. Μπορούν να κληθούν επανειλημμένα σε ένα πρόγραμμα, δεχόμενες κάθε φορά διαφορετικές τιμές στις εισόδους τους.**

Μία συνάρτηση…

- εκτελεί ένα σαφώς καθορισμένο έργο (π.χ. *printf*)
- μπορεί να χρησιμοποιηθεί από άλλα προγράμματα
- είναι ένα "μαύρο κουτί", το οποίο:
  - έχει ένα απλό σύνολο εισόδων (μεταβλητές)
  - μία απλή έξοδο
  - ένα κρυμμένο σώμα, αποτελούμενο από προτάσεις

Ήδη γνωστές συναρτήσεις

- ***main()*** **: Η πρώτη (και τελευταία) συνάρτηση που εκτελείται όταν εκτελείται ένα πρόγραμμα.**

- ***printf("%f \n",x); scanf("%d",&x);*** **…**

- **Επιτρέπεται (μάλιστα ενθαρρύνεται!) η ένθεση συναρτήσεων:**
  - **Μέσα από τη *main()* μπορεί να κληθεί οποιαδήποτε συνάρτηση.**
  - **Οι δικές μας συναρτήσεις μπορούν να καλέσουν άλλες, κ.ο.κ.**

Βασικά στοιχεία των συναρτήσεων

- Μία συνάρτηση έχει:
  - ένα **όνομα**, για να λειτουργήσει μία συνάρτηση πρέπει να κληθεί κατ' όνομα
  - ένα **σώμα**, ένα σύνολο προτάσεων και μεταβλητών
  - (προαιρετικά) **εισόδους**, μία λίστα ορισμάτων
  - μία (προαιρετικά) **έξοδο**, τερματίζοντας επιστρέφει μία τιμή
- Όπως οι μεταβλητές, έτσι και οι συναρτήσεις πρέπει να **«δηλώνονται»** προτού χρησιμοποιηθούν.
- Επίσης πρέπει να **«ορίζονται»** — να έχει γραφεί το σώμα τους.

Βασικά στοιχεία των συναρτήσεων

Το όνομα της συνάρτησης συναντάται σε ένα πρόγραμμα σε προτάσεις τριών διαφορετικών μορφών:

1) *Πρόταση δήλωσης της συνάρτησης*

2) *Πρόταση ορισμού της συνάρτησης*

3) *Πρόταση κλήσης της συνάρτησης*

Παράδειγμα: Χωρίς συναρτήσεις

/* Μετατροπή βαθμών Φαρενάιτ σε βαθμούς Κελσίου */

```
#include<stdio.h>
int main ()
{
  float degF,degC, ratio;
  printf( "Enter degrees F: " );
  scanf( "%f",&degF );
  ratio = (float) 5/9;
  degC = (degF-32)*ratio;
  printf( "%f degrees F are %f degrees C\n",degF,degC );
  return 0;  }
```

θα ενταχθούν σε συνάρτηση

Πώς θα γράφαμε τον κώδικα εάν αυτή η μετατροπή χρειαζόταν σε πολλά διαφορετικά σημεία του προγράμματος;

Παράδειγμα: Με συναρτήσεις

```
#include<stdio.h>
float F_to_C (float far);
int main () {
  float degF,degC;
  printf("Enter degrees F: ");
  scanf("%f", &degF);
  degC = F_to_C (degF);
  printf("%f degrees F are %f degrees C\n",degF,degC);
  return 0;
}
float F_to_C (float far) {
  float ratio = 5.0 / 9.0;
  return((far-32)*ratio);
}
```

Δήλωση συνάρτησης

Κλήση συνάρτησης

Σώμα συνάρτησης

Όνομα συνάρτησης

Δήλωση συναρτήσεων

- ***Πρότυπο*** **συνάρτησης, αποτελούμενο από τρία τμήματα, όπου ορίζονται:**
  - **Το όνομα: να είναι ενδεικτικό της λειτουργίας της**
  - **Είσοδοι (εφόσον υπάρχουν): μία *λίστα ορισμάτων*, αποτελούμενη από ονόματα μεταβλητών εισόδου και τύπων δεδομένων**
  - **Τύπος της εξόδου: τύπος δεδομένου της επιστρεφόμενης τιμής, εφόσον επιστρέφεται τιμή (*προκαθορισμένος τύπος, default: int*)**

- **Πού γίνεται η δήλωση; Τοποθετείστε τη μετά από τις προτάσεις #include, πριν όμως τη *main()*.**

Σύνταξη συναρτήσεων

(να μην ξεχνάτε το ;)

- Σύνταξη της δήλωσης της συνάρτησης:

return_type function_name(*argument_list*);

- Σύνταξη της συνάρτησης:

return_type function_name(*argument_list*)

{

προτάσεις;

}

(ίδιος αριθμός και τύποι δεδομένων, ελεύθερη επιλογή ονομάτων)

- Εάν η συνάρτηση δεν επιστρέφει τιμή, σημειώστε ως return_type τη λέξη **void**.
- Οι συναρτήσεις μπορούν να έχουν τις δικές τους εσωτερικές μεταβλητές, όπως ακριβώς έχει η *main()*.
- ***return()***: θέτει την έξοδο της συνάρτησης.

Δήλωση και Ορισμός

```
#include<stdio.h>
float F_to_C (float far);

int main () {
  float degF,degC;
  printf("Enter degrees F: ");
  scanf("%f", &degF);
  degC = F_to_C (degF);
  printf("%f degrees F are %f degrees C\n",degF,degC);
  return 0;
}
```

```
float F_to_C (float far) {
  float ratio = 5.0 / 9.0;
  return(far-32)*ratio);
}
```

όνομα συνάρτησης

λίστα ορισμάτων (ένα όρισμα)

Επιστρεφόμενη τιμή

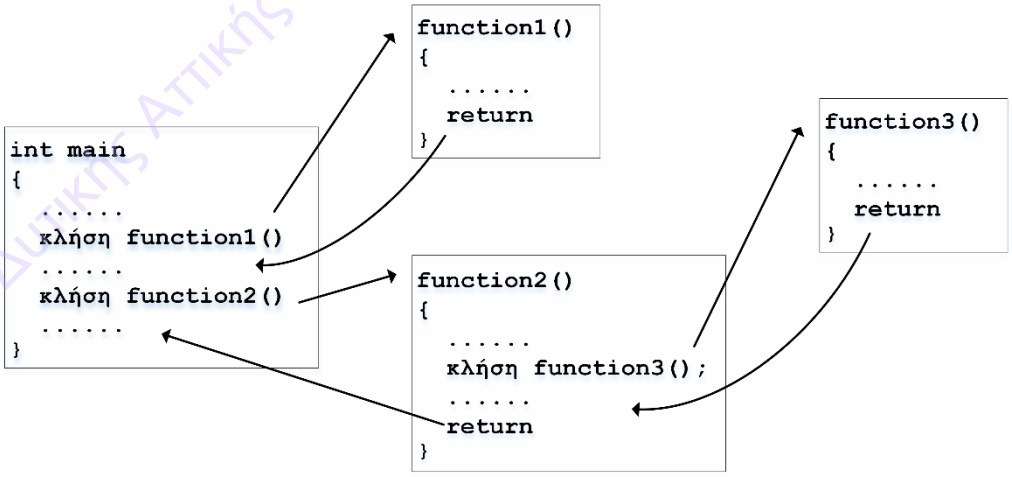
ορισμός συνάρτησης

Γενική μορφή ενός C προγράμματος

```
εντολές προεπεξεργαστή (#include, #define,…)
δηλώσεις συναρτήσεων
δηλώσεις μεταβλητών (εφόσον απαιτούνται)
int main()
{
    δηλώσεις μεταβλητών
    προτάσεις
}
func1()
{
    …
}
func2()
{
    …
}
```

Η λειτουργία κλήσης της συνάρτησης

Όταν καλείται μία συνάρτηση, π.χ. **y=maximumTwoIntegers(x,32);**, οι τιμές **x** (δηλαδή το **10**) και **32** αντιγράφονται μία προς μία στις μεταβλητές που συνιστούν τη λίστα παραμέτρων. Αυτό σημαίνει ότι η συνάρτηση δε διαχειρίζεται τα αυθεντικά δεδομένα αλλά αντίγραφα αυτών. Αυτός ο τρόπος κλήσης ονομάζεται **κλήση κατ' αξία** (call by value). Σε επόμενη ενότητα θα μελετηθεί ένας δεύτερος τρόπος κλήσης συναρτήσεων, η **κλήση κατ' αναφορά** (call by reference). Ο τρόπος επικοινωνίας ανάμεσα στα τμήματα ενός δομημένου προγράμματος απεικονίζεται στο σχήμα.



Η λειτουργία κλήσης της συνάρτησης

Όταν καλείται μία συνάρτηση (είτε από τη **main()** είτε από άλλη συνάρτηση), ο έλεγχος ροής του προγράμματος περνά σε αυτήν και αρχίζει η εκτέλεση των εντολών της. Ο έλεγχος του προγράμματος επιστρέφει στην καλούσα συνάρτηση είτε όταν εκτελεσθούν όλες οι εντολές της καλούμενης συνάρτησης είτε όταν εκτελεσθεί εντολή **return**. Η επιστροφή στην καλούσα συνάρτηση γίνεται στην εντολή που ακολουθεί την κλήση της καλούμενης συνάρτησης.

Κατά συνέπεια, στην πρόταση **y=maximumTwoIntegers(x,32);** πρώτα εκτελείται η εντολή κλήσης συνάρτησης και όταν ολοκληρωθεί η λειτουργία της **maximumTwoIntegers**, η επιστρεφόμενη τιμή ανατίθεται στη μεταβλητή **y** της **main()**.

Η λειτουργία κλήσης της συνάρτησης

Για τον χειρισμό των συναρτήσεων δημιουργείται στη μνήμη του υπολογιστή μία στοίβα κλήσεων (call stack). Κάθε φορά που καλείται μία συνάρτηση προστίθεται μία εγγραφή στη στοίβα κλήσεων, που περιέχει πληροφορίες σχετικά με την κατάσταση της συνάρτησης, δηλαδή πληροφορίες σχετικά με τις παραμέτρους της συνάρτησης, τις μεταβλητές της, τη διεύθυνση της επιστροφής κ.λ.π. Με το πέρας της συνάρτησης η στοίβα αδειάζει από την εγγραφή, δηλαδή οι παράμετροι και οι μεταβλητές της συνάρτησης παύουν να υφίστανται, υπό την έννοια ότι οι θέσεις μνήμης που καταλάμβαναν θεωρούνται πλέον μη σχετιζόμενες με τη λειτουργία του προγράμματος, άρα ελεύθερες προς διάθεση.

Θα πρέπει να σημειωθεί ότι μπορεί να παραληφθεί η δήλωση της συνάρτησης, εάν η συνάρτηση παρουσιάζεται μέσα στο πρόγραμμα πριν από την πρώτη κλήση της. Ωστόσο αυτή είναι μία ριψοκίνδυνη τακτική και θα πρέπει να αποφεύγεται!!!

```
#include <…h>
#define ……
float square(float y)
{
 return(y*y);
}
int main()
{
 float x=15.2;
 printf( “x^2=%f\n”,square(x) );
 return 0; }
```

Παραλήφθηκε η δήλωση της *square* γιατί αυτή ορίζεται πριν κληθεί. Εάν όμως γράφαμε τη *square* κάτω από τη *main* έπρεπε να τη δηλώσουμε.

Παράδειγμα: Να γραφεί πρόγραμμα, στο οποίο να καλούνται οι συναρτήσεις, τα πρωτότυπα των οποίων δίνονται ως ακολούθως:

```
int max(int a, int b);
double power(double x, int n);
```

Λύση: Πριν από κάθε κλήση συνάρτησης πρέπει να υπάρχει στον πηγαίο κώδικα το πρωτότυπό της, έτσι ώστε ο μεταγλωττιστής να ελέγξει αν κάθε πρόταση κλήσης είναι σύμφωνη ως προς τον αριθμό και τον τύπο των ορισμάτων, καθώς και τον τύπο της επιστρεφόμενης τιμής.

```
#include <…h>
#define ……
```

→ ***include και define***

```
int max(int a, int b);
double power(double x, int n);

int main()  {
  int num=5;
  printf( “%d\n”,max(12/2,num+3 );
  printf( “%f\n”,power(num,3) );
  return 0;  }
```

Στο παράδειγμα αυτό δε γράψαμε τις συναρτήσεις, παρά μόνο τον τρόπο δήλωσης και κλήσης.

need_to_declare.c

Τοπικές μεταβλητές

- *Έχουν νόημα <u>μόνο</u> μέσα στη συνάρτηση που δηλώνονται.*
- *Δύο διαφορετικές συναρτήσεις μπορούν να έχουν τοπική μεταβλητή με το ίδιο όνομα, χωρίς να παρουσιάζεται πρόβλημα.*

```
#include<stdio.h>

float square (float x);
int main () {
    float in,out;
     in = -4.0;
     out = square(in);
     printf("%f squared is %f\n",in,out);
    return 0; }

float  square (float x)
{
    float out;
     out = 24.5;
     return (x*x);
}
```

Ίδια ονόματα τοπικών μεταβλητών

Αποτέλεσμα στην οθόνη out=16.0

out=24.5 μέσα στην square

Καθολικές μεταβλητές (global variables)

- Δηλώνονται πριν τη *main()*.
- Εφαρμόζονται σε ΟΛΑ τα τμήματα ενός προγράμματος.
- Όταν μεταβάλλεται η τιμή μίας καθολικής μεταβλητής σε οποιοδήποτε σημείο του προγράμματος, η νέα τιμή μεταφέρεται σε όλο το υπόλοιπο πρόγραμμα.
- Οι καθολικές μεταβλητές είναι μία κακή ιδέα, καθώς αποτρέπουν τον ξεκάθαρο μερισμό του προβλήματος σε ανεξάρτητα τμήματα.
- Για μία τοπική μεταβλητή ο χώρος στη μνήμη δεσμεύεται μόλις ο έλεγχος περάσει στη συνάρτηση, αποδεσμεύεται δε με το τέλος αυτής, οπότε και η μεταβλητή δεν έχει πλέον νόημα.

Παράδειγμα με καθολικές μεταβλητές

```
#include<stdio.h>
float glob;      // καθολική μεταβλητή
float square (float x);
int main ()  {
  float in;
  glob = 2.0;
  in = square(glob);
  printf( "%f squared is %f\n",glob,in );
  in = square(glob);
  printf( "%f squared is %f\n",glob,in );
  return 0;  }
float  square (float x)  {
  glob=glob+1.0;
  return (x*x);  }
```

Ζητά από τη *square()* το τετράγωνο της glob, δηλαδή το τετράγωνο του 2.0

Τώρα ζητά από τη *square()* το τετράγωνο τη νέας τιμής της glob, δηλαδή το τετράγωνο του 3.0

Η glob γίνεται 3.0

func_call_to_func.c

Εμβέλεια μεταβλητών (scope)

- **Εμβέλεια προγράμματος:** μεταβλητές αυτής της εμβέλειας είναι οι καθολικές. Είναι ορατές από όλες τις συναρτήσεις του πρόγραμματος, έστω κι αν βρίσκονται σε διαφορετικά αρχεία πηγαίου κώδικα.
- **Εμβέλεια αρχείου:** μεταβλητές αυτής της εμβέλειας είναι ορατές μόνο στο αρχείο που δηλώνονται και μάλιστα από το σημείο της δήλωσής τους και κάτω. Μεταβλητή που δηλώνεται με τη λέξη κλειδί *static* πριν από τον τύπο, έχει εμβέλεια αρχείου, π.χ. static int velocity.
- **Εμβέλεια συνάρτησης:** Προσδιορίζει την ορατότητα του ονόματος από την αρχή της συνάρτησης έως το τέλος της. Εμβέλεια συνάρτησης έχουν μόνο οι goto ετικέτες.
- **Εμβέλεια μπλοκ:** Προσδιορίζει την ορατότητα από το σημείο δήλωσης έως το τέλος του μπλοκ στο οποίο δηλώνεται. Μπλοκ είναι ένα σύνολο από προτάσεις, οι οποίες περικλείονται σε άγκιστρα. Μπλοκ είναι η σύνθετη πρόταση αλλά και το σώμα συνάρτησης. Εμβέλεια μπλοκ έχουν και τα τυπικά ορίσματα των συναρτήσεων.

Εμβέλεια μεταβλητών (συνέχεια)

Η C επιτρέπει τη χρήση ενός ονόματος για την αναφορά σε διαφορετικά αντικείμενα, με την προϋπόθεση ότι αυτά έχουν διαφορετική εμβέλεια ώστε να αποφεύγεται η *σύγκρουση ονομάτων* (name conflict). Εάν οι περιοχές εμβέλειας έχουν επικάλυψη, τότε το όνομα με τη μικρότερη εμβέλεια αποκρύπτει (hides) το όνομα με τη μεγαλύτερη.

Παράδειγμα: Να προσδιορισθεί η εμβέλεια του ακόλουθου πηγαίου κώδικα:

```
1  #include <stdio.h>
2  int max(int a, int b);
3  void func(int x);
4  int a; static int b;
5  int main() {
6    b=12; a=b--;
7    printf( “a:%d\tb:%d\tmax(b+5,a):%d\n”,a,b,max(b+5,a) );
8    func(a+b);
9    Return 0;  }
10 int c=13;
11 int max(int a, int b){
12   return(a>b?a:b);
13 }
14 void func(int x){
15   int b=20;
16   printf( “a:%d\tb:%d\tc:%d\tx:%d\tmax(x,b):%d\n”,
17      a,b,c,x,max(x,b) );
18 }
```

Επεξηγήσεις στην επόμενη διαφάνεια

a:12 b:11 max(b+5,a):16

if (a>b) return a;

else return b;

a:12 b:20 c:13 x:23 max(x,b):23

Επεξηγήσεις:

- 4 int a; static int b; Η a είναι καθολική μεταβλητή με εμβέλεια προγράμματος. Η b έχει εμβέλεια αρχείου, όπως προσδιορίζει η λέξη *static*.
- 10 int c=13; Έχει εμβέλεια προγράμματος αλλά είναι ενεργή από το σημείο δήλωσής της και κάτω (γραμμή 10).
- 11 int max(int a, int b){ Οι a και b έχουν εμβέλεια μπλοκ και αποκρύπτουν για το σώμα της *max()* τις καθολικές μεταβλητές a και b.
- 6 b=12; a=b--; Αποδίδονται οι τιμές 12 και 11 στις a και b, αντίστοιχα.
- 7 printf("a:%d\tb:%d\tmax(b+5,a):%d\n",a,b,max(b+5,a)); Καλείται η συνάρτηση *max()* και αυτή δίνει στα τυπικά ορίσματα a και b τις τιμές 16 και 12, αντίστοιχα. Η *max()* επιστρέφει στην *printf* το 16.
- 8 func(a+b); Καλείται η συνάρτηση *func()* και αυτή δίνει στο τυπικό όρισμα x την τιμή 12+11=23. Το όρισμα x έχει εμβέλεια μπλοκ. Η τοπική μεταβλητή b=20, που δηλώνεται στη γραμμή 15, αποκρύπτει από το σώμα της *func()* την καθολική μεταβλητή b. Αντίθετα η καθολική μεταβλητή a είναι ορατή από το σώμα της *func()*.

building_area.c

Διάρκεια μεταβλητών (duration)

- Η διάρκεια ορίζει το χρόνο κατά τον οποίο το όνομα της μεταβλητής είναι συνδεδεμένο με τη θέση μνήμης που περιέχει την τιμή της μεταβλητής. Ορίζονται ως *χρόνοι δέσμευσης* και *αποδέσμευσης* οι χρόνοι που το όνομα συνδέεται με και αποσυνδέεται από τη μνήμη, αντίστοιχα.
- Για τις καθολικές μεταβλητές δεσμεύεται χώρος με την έναρξη εκτέλεσης του προγράμματος και η μεταβλητή συσχετίζεται με την ίδια θέση μνήμης έως το τέλος του προγράμματος. Είναι *πλήρους διάρκειας*.
- Οι τοπικές μεταβλητές είναι *περιορισμένης διάρκειας*. Η ανάθεση της μνήμης σε τοπική μεταβλητή γίνεται με τη είσοδο στο χώρο εμβέλειάς της και η αποδέσμευσή της με την έξοδο από αυτόν. Δηλαδή η τοπική μεταβλητή δε διατηρεί την τιμή της από τη μία κλήση της συνάρτησης στην επόμενη.
- Εάν όμως προστεθεί στη δήλωση μίας τοπικής μεταβλητής η λέξη *static*, διατηρεί την τιμή της και καθίσταται πλήρους διάρκειας.

Static μεταβλητές

Μία **static** μεταβλητή παρουσιάζει τα ακόλουθα χαρακτηριστικά:

- Κατά την πρώτη κλήση της συνάρτησης στην οποία βρίσκεται, μία **static** μεταβλητή δημιουργείται και – κατά την εκτέλεση της συνάρτησης – λαμβάνει τιμή. Όταν ολοκληρώνεται η πρώτη κλήση της συνάρτησης, η μεταβλητή και η τιμή που απέκτησε διατηρούνται εν ενεργεία και αναμένουν επόμενη κλήση για να χρησιμοποιηθούν.

- Σε κάθε επόμενη κλήση της συνάρτησης δεν εκτελείται η πρόταση δήλωσης της **static** μεταβλητής αλλά η μεταβλητή υφίσταται και μάλιστα με την τιμή που είχε όταν ολοκληρώθηκε η προηγούμενη κλήση της συνάρτησης. Μετά το πέρας της κλήσης της συνάρτησης, η μεταβλητή διατηρείται με τη νέα τιμή που –πιθανόν – έλαβε στην κλήση αυτή. Αυτή η διαδικασία συνεχίζεται καθόλη τη διάρκεια εκτέλεσης του προγράμματος.

Παράδειγμα:

```
func(int x);
{
  int temp;
  static int num;
  . . . . . ...
}
```

Η μεταβλητή ***num*** είναι τοπική αλλά έχει διάρκεια προγράμματος, σε αντίθεση με την ***temp***, η οποία έχει διάρκεια συνάρτησης.

Προσοχή πρέπει να δοθεί στην αρχικοποίηση των τοπικών μεταβλητών. Μία τοπική μεταβλητή περιορισμένης διάρκειας αρχικοποιείται, εφόσον βέβαια κάτι τέτοιο έχει ορισθεί, με κάθε είσοδο στο μπλοκ που αυτή ορίζεται. Αντίθετα, μία τοπική μεταβλητή πλήρους διάρκειας αρχικοποιείται μόνο με την ενεργοποίηση του προγράμματος.

Παράδειγμα:

α) Να περιγραφεί η επίδραση της λέξης κλειδί ***static*** στις δύο δηλώσεις του ακόλουθου πηγαίου κώδικα: ***β)*** Πότε αρχικοποιείται η **count** και πότε η **num**; **static int num;**

```
void func(void)  {
  static int count=0;
  int num=100;
  . . . . . .
}
```

Λύση:

α) **Η *static* στη δήλωση της καθολικής μεταβλητής num περιορίζει την ορατότητά της μόνο στο αρχείο που δηλώνεται. Αντίθετα η *static* στη δήλωση της τοπικής μεταβλητής count ορίζει γι' αυτήν διάρκεια προγράμματος.**

β) **Η count ως τοπική μεταβλητή αρχικοποείται μία φορά με την είσοδο στο πρόγραμμα. Αντίθετα η num ως τοπική μεταβλητή περιορισμένης διάρκειας αρχικοποιείται σε κάθε ενεργοποίηση της συνάρτησης *func()*.**

Παράδειγμα στατικών μεταβλητών

Ο κώδικας που ακολουθεί υλοποιεί τον υπολογισμό του κυλιόμενου μέσου όρου, σύμφωνα με τον οποίο προστίθενται διαδοχικά δεδομένα και σε κάθε προσθήκη δεδομένου υπολογίζεται ο νέος μέσος όρος. Με αυτόν τον τρόπο δημιουργείται μία ακολουθία μέσων όρων ως εξής:

- Η εισαγωγή του πρώτου δεδομένου x_1 οδηγεί στον μέσο όρο $MO_1 = \frac{x_1}{1}$
- Η εισαγωγή του δεύτερου δεδομένου x_2 οδηγεί στον μέσο όρο $MO_2 = \frac{x_1 + x_2}{2} = \frac{x_1 + x_2}{1+1}$
- Η εισαγωγή του τρίτου δεδομένου x_3 οδηγεί στον μέσο όρο $MO_3 = \frac{x_1 + x_2 + x_3}{3} = \frac{(x_1 + x_2) + x_3}{2+1}$
- Η εισαγωγή του k-στου δεδομένου x_k οδηγεί στον μέσο όρο

$$MO_k = \frac{x_1 + x_2 + \ldots + x_k}{k} = \frac{(x_1 + x_2 + \ldots + x_{k-1}) + x_k}{(k-1)+1}$$

Παράδειγμα στατικών μεταβλητών

Με βάση τα παραπάνω αν ορίσουμε $MO_k = \frac{A_k}{\Pi_k}$, ο υπολογισμός του μέσου όρου μετά την εισαγωγή του επόμενου δεδομένου προκύπτει από τη σχέση: $MO_{k+1} = \frac{A_k + x_{k+1}}{\Pi_k + 1}$, δηλαδή μπορεί να εξαχθεί βάσει των ήδη υπολογισθεισών τιμών των A_k και Π_k.

Είναι προφανές ότι η έννοια της **static** μεταβλητής είναι σαφέστατα πιο σύνθετη από εκείνη της απλής τοπικής μεταβλητής. Η υλοποίηση όμως ορισμένων προγραμμάτων με χρήση **static** μεταβλητών μπορεί να επιφέρει σημαντικές βελτιώσεις στις απαιτήσεις αποθηκευτικού χώρου και στον χρόνο εκτέλεσης ενός προγράμματος.

Παράδειγμα στατικών μεταβλητών

Εάν χρησιμοποιείτο προσέγγιση χωρίς **static** μεταβλητές, θα έπρεπε σε κάθε επανάληψη να αθροίζονται όλα τα δεδομένα που εισήχθησαν στις προηγούμενες επαναλήψεις, τα οποία μάλιστα θα έπρεπε να παραμείνουν αποθηκευμένα στη μνήμη. Αυτό σημαίνει ότι στην *k*-στη επανάληψη για τον υπολογισμό του αριθμητή του μέσου όρου θα έπρεπε να γίνουν (**k-1**) προσθέσεις. Κατά συνέπεια η διαδοχική εισαγωγή ***N*** δεδομένων θα απαιτούσε ***N*** θέσεις μνήμης για την αποθήκευση των δεδομένων και $1+2+\ldots+(N-1)=\frac{(N-1)\cdot(N-2)}{2}$ προσθέσεις, δηλαδή η υλοποίηση του κυλιόμενου μέσου όρου θα απαιτούσε αλγόριθμο τάξης N^2.

Αντίθετα, η χρήση **static** μεταβλητών απαιτεί μόνο 2 θέσεις μνήμης (μία για τον αριθμητή και μία για τον παρονομαστή). Επιπρόσθετα, σε κάθε νέα εισαγωγή δεδομένου γίνεται μία πρόσθεση για τον υπολογισμό του αριθμητή και μία ακόμη για τον υπολογισμό του παρονομαστή. Επομένως η διαδοχική εισαγωγή ***N*** δεδομένων θα απαιτούσε ***2N*** προσθέσεις, δηλαδή η υλοποίηση του κυλιόμενου μέσου όρου θα απαιτούσε αλγόριθμο τάξης ***N***.

```
#include <stdio.h>
float get_average(float newdata);   // δήλωση συνάρτησης
int main()
{
  float data=1.0;
  float average;
  while (data!=0)
  {
    printf( "\n\tGive a number or press 0 to finish: " );
    scanf( "%f",&data );
    average=get_average(data);
    if (fabs(data)>0.000001) printf( "\tThe new average is %.3f\n",average );
    else printf( "\tThe final average is %.3f\n",average );
  }
  return 0;
}   // τέλος της main, συνέχεια στην επόμενη διαφάνεια
```

```
float get_average(float newdata)
{
  static float total=0.0;
  static int count=0;
  if (fabs(newdata)>0.000001) count++;
  total=total+newdata;
  return(total/count);
}  //end of get_average
```

Εκτελούνται μόνο την πρώτη φορά. Τις επόμενες διατηρούν το αποτέλεσμα της προηγούμενης κλήσης και σε αυτό προστίθενται στη μεν total το newdata, στη δε count η μονάδα. Ο έλεγχος της τιμής του newdata γίνεται για να μην εξαχθεί μέσος όρος όταν δοθεί τιμή τερματισμού, αλλά να διατηρηθεί ο προηγούμενος μέσος όρος.

Αποτέλεσμα:

```
Give a number or press 0 to finish: 2
The new average is 2.000

Give a number or press 0 to finish: 4
The new average is 3.000

Give a number or press 0 to finish: 6
The new average is 4.000

Give a number or press 0 to finish: 0
The final average is 4.000
```

2/1=2

(2+4)/2=3

(2+4+6)/3=4

Διατηρείται το προηγούμενο πλήθος στοιχείων, καθώς και οι τιμές τους.