## ΣΥΝΑΡΤΗΣΕΙΣ

- Η συνάρτηση είναι μια αυτοδύναμη μονάδα του κώδικα, που είναι σχεδιασμένη για την επίτευξη κάποιου συγκεκριμένου σκοπού.
- Μερικές συναρτήσεις απλώς εκτελούν κάποιες ενέργειες χωρίς να επιστρέφουν κάποια τιμή στο πρόγραμμα. Για παράδειγμα, η

  `void myputchar(char ch) { putchar(ch); }`

  απλώς εμφανίζει δεδομένα στην οθόνη του υπολογιστή.
- Άλλες συναρτήσεις υπολογίζουν κάποια τιμή που θα χρησιμοποιηθεί από το πρόγραμμα που τις κάλεσε. Π.χ. η `cos()` υπολογίζει, για λογαριασμό του προγράμματος που την κάλεσε, το συνημίτονο μιας γωνίας.
- Γενικά, μια συνάρτηση μπορεί, ταυτόχρονα, να εκτελεί κάποιες ενέργειες και να υπολογίζει τιμές, τις οποίες επιστρέφει στο πρόγραμμα που την κάλεσε.

## ΣΥΝΑΡΤΗΣΕΙΣ

- Γενική μορφή μιας C-συνάρτησης:

```
τύπος όνομα-συνάρτησης (λίστα με διακηρύξεις ορισμάτων)
{
  σώμα συνάρτησης
}
```

- Οι συναρτήσεις μας γλυτώνουν από επαναλήψεις. Για παράδειγμα, αν πρέπει να επαναλαμβάνουμε συχνά κάποια διαδικασία, είναι καλύτερο να γράψουμε μια κατάλληλη συνάρτηση γι'αυτό το σκοπό και να έχουμε το πρόγραμμα ή τα προγράμματα να καλούν αυτή τη συνάρτηση όποτε τη χρειάζονται.
- Ακόμη και αν δεν πρόκειται να χρησιμοποιηθεί πολλές φορές μια συνάρτηση, αξίζει να δομούμε τα προγράμματά μας με συναρτήσεις επειδή αυτά γίνονται πιο κατανοητά και εύκολα σε αλλαγές ή βελτιώσεις.

# ΣΥΝΑΡΤΗΣΕΙΣ

- **Για παράδειγμα, ας υποθέσουμε ότι θέλουμε να γράψουμε ένα πρόγραμμα που:**

  **Να διαβάζει μια λίστα αριθμών**
  **Να τους τοποθετεί σε αύξουσα διάταξη**
  **Να βρίσκει τη μέση τιμή τους**
  **Να εμφανίζει ένα γράφημα με μπάρες**

  **Θα μπορούσαμε να χρησιμοποιήσουμε το παρακάτω πρόγραμμα:**

```
int main()
{
  float list[50];

  readlist(list);
  sort(list);
  average(list);
  bargraph(list);
  return 0;
}
```

## ΔΗΜΙΟΥΡΓΙΑ ΜΙΑΣ ΑΠΛΗΣ ΣΥΝΑΡΤΗΣΗΣ ΧΩΡΙΣ ΟΡΙΣΜΑΤΑ

```
#include <stdio.h>
#define IDRYMA "PANEPISTHMIO DYTIKHS ATTIKHS"
#define CAMPUS "PANEPISTHMIOYPOLH I"
#define DIEYTHYNSH "AGIOU SPYRIDONOS ST., 122 10 EGALEO"

void starbar(); /* Αρχέτυπο συνάρτησης που θα χρησιμοποιηθεί
                   αργότερα από το πρόγραμμά μας */
int main()
{
  starbar();
  printf("%s\n%s\n%s\n",IDRYMA,CAMPUS,DIEYTHYNSH);
  starbar();
  return 0;
}

#define LIMIT 50  /* ΑΥΤΗ Η #define ΙΣΧΥΕΙ ΑΠΟ ΕΔΩ ΚΑΙ ΜΕΤΑ … */
void starbar()
{
  int i;
  for(i=0; i<LIMIT; i++)
    putchar('*');
  putchar('\n');
}
```

## ΔΗΜΙΟΥΡΓΙΑ ΜΙΑΣ ΑΠΛΗΣ ΣΥΝΑΡΤΗΣΗΣ ΧΩΡΙΣ ΟΡΙΣΜΑΤΑ

Έξοδος προγράμματος:

```
************************************************
PANEPISTHMIO DYTIKHS ATTIKHS
PANEPISTHMIOYPOLH I
AGIOU SPYRIDONOS ST., 122 10 EGALEO
************************************************
```

## ΠΑΡΑΔΕΙΓΜΑ ΣΥΝΑΡΤΗΣΗΣ ΜΕ ΟΡΙΣΜΑΤΑ

```
#include <stdio.h>
float square(float x);   /* Αρχέτυπο συνάρτησης square() */
float cube(float x);     /* Αρχέτυπο συνάρτησης cube() */

int main()
{
  float num, num2, num3;

  printf("Pliktrologhse enan arithmo: ");
  scanf("%f",&num);

  num2 = square(num);
  num3 = cube(num);
  printf("%.1f^2 = %.3f, %.1f^3 = %.3f\n",num,num2,num,num3);
  return 0;
}

float square(float x)  /* Υπολογίζει το τετράγωνο μιας τιμής */
{
  float res;

  res = x * x;
  return res;
}
```

## ΠΑΡΑΔΕΙΓΜΑ ΣΥΝΑΡΤΗΣΗΣ ΜΕ ΟΡΙΣΜΑΤΑ

```
float cube(float x)  /* Υπολογίζει τον κύβο μιας τιμής */
{
  return (x * x * x);
}
```

Παράδειγμα εκτέλεσης προγράμματος:

```
Pliktrologhse enan arithmo: 2.5 <enter>
2.5^2 = 6.250, 2.5^3 = 15.625
```

## ΕΜΦΩΛΕΥΜΕΝΕΣ ΚΛΗΣΕΙΣ ΣΥΝΑΡΤΗΣΕΩΝ

Τα ορίσματα μιας συνάρτησης μπορούν να είναι οποιεσδήποτε έγκυρες εκφράσεις της C (τιμές, μεταβλητές, αριθμητικές ή λογικές παραστάσεις ή μια άλλη συνάρτηση με τιμή επιστροφής). Οι παρακάτω εμφωλευμένες κλήσεις συναρτήσεων είναι απολύτως νόμιμες:

```
x = half(third(square(half(y))));
```

⟺

```
a = half(y);
b = square(a);
c = third(b);
x = half(c);
```

## ΧΡΗΣΗ ΣΥΝΑΡΤΗΣΕΩΝ ΣΕ ΕΚΦΡΑΣΕΙΣ

```
/* Ελεγχος αν ένα τρίγωνο είναι ορθογώνιο. Τα ορίσματα της
   συνάρτησης είναι τύπου int και αντιστοιχούν στις πλευρές
   του τριγώνου. Η συνάρτηση επιστρέφει TRUE/FALSE. */
int orthogonal_triangle(int a, int b, int c)
{
  if (square(a) == square(b) + square(c) ||
      square(b) == square(c) + square(a) ||
      square(c) == square(a) + square(b))
     return 1;
  else
     return 0;
}
```

Παράδειγμα χρήσης της συνάρτησης:

*Έστω ότι έχουμε δώσει ακέραιες τιμές στις μεταβλητές x, y, z*

```
if (orthogonal_triangle(x, y, z))
   printf("H triada (%d %d %d) ikanopoiei to Pythagoreio
           theorhma\n", x, y, z);
```

## ΠΙΝΑΚΑΣ ΩΣ ΟΡΙΣΜΑ ΣΥΝΑΡΤΗΣΗΣ

```
#include <stdio.h>
#define  DIM  5
float average(float pin[], int dim);
int main()
{
  int i;
  float a[DIM];

  for(i=0; i<DIM; i++)
  {  printf("\nPlease input a float for a[%d]: ",i);
     scanf("%f",&a[i]); }
  printf("Average of the %d numbers: %f\n",i,average(a,DIM));
  return 0;
}
float average(float pin[], int dim)
{
  int i;
  float sum;

  for(i=0, sum=0.0; i<dim; i++)
      sum += pin[i];
  return (sum/dim);
}
```

## Σχολιασμός προγράμματος

Η `average()` έχει δύο ορίσματα (παραμέτρους). Από την διακήρυξη των ορισμάτων βλέπουμε ότι το πρώτο (δηλαδή, το pin) είναι ένας μονοδιάστατος, τύπου float, πίνακας ενώ το άλλο (δηλαδή, το dim) είναι τύπου ακεραίου. Όταν το κυρίως πρόγραμμα καλεί τη συνάρτηση, εκχωρούνται τιμές στις παραμέτρους της. Δηλαδή, η κλήση `average(a,DIM)` εκχωρεί τη **διεύθυνση** του πρώτου στοιχείου του πίνακα a στη μεταβλητή pin (*αυτό θα εξηγηθεί αναλυτικά στο κεφάλαιο των δεικτών*) και, επίσης, εκχωρεί την τιμή 5 στην dim.

Τα ονόματα των ορισμάτων είναι στην ουσία νέες μεταβλητές. Ο υπολογιστής πρέπει να τους εξασφαλίσει θέσεις μνήμης όπως ακριβώς γίνεται με όλες τις μεταβλητές. Όλες οι μεταβλητές μιας συνάρτησης (ορίσματα και μεταβλητές που δηλώνονται στο σώμα της συνάρτησης) είναι **τοπικής εμβέλειας**, δηλαδή δημιουργούνται με την κλήση της συνάρτησης, υπάρχουν όσο ο έλεγχος του προγράμματος είναι στη συνάρτηση και καταργούνται (εξαφανίζονται) με το τέλος της συνάρτησης. Γι'αυτόν τον λόγο, θα ονομάζονται **τοπικές μεταβλητές**.

Οι τοπικές μεταβλητές μιας συνάρτησης μπορούν να έχουν τα ίδια ονόματα με αυτά των μεταβλητών άλλων συναρτήσεων, συμπεριλαμβανομένης και της main(), χωρίς να υπάρχει κίνδυνος σύγκρουσης. Για παράδειγμα, δεν θα υπήρχε κανένα πρόβλημα αν χρησιμοποιούσαμε το `a` αντί για `pin` ως πρώτο όρισμα της `average()`.

## ΚΛΑΣΕΙΣ ΚΑΙ ΕΜΒΕΛΕΙΑ ΜΕΤΑΒΛΗΤΩΝ (Storage Classes)

Όλες οι μεταβλητές που ορίζονται εντός μιας συνάρτησης (της main() συμπεριλαμβανομένης) είναι τοπικές και αντιστοιχούν στην κλάση **auto** της C.
Η κλάση **auto** είναι η **default κλάση** και δεν χρειάζεται να τη δηλώνουμε κατά τη διακήρυξη των μεταβλητών. Αν για κάποιο λόγο θα θέλαμε να υπενθυμίσουμε ότι μια μεταβλητή είναι σκοπίμως **αυτόματη** τότε μπορούμε να είμαστε αναλυτικοί κατά τη δήλωση των μεταβλητών και αντί, για παράδειγμα, της διακήρυξης

```
int  sum;
float a=10.3, b=-13.9;
```

θα μπορούσαμε να έχουμε την ισοδύναμη διακήρυξη

```
auto int  sum;
auto float a=10.3, b=-13.9;
```

Οι μεταβλητές κλάσης auto δημιουργούνται κάθε φορά που καλείται η συνάρτηση και καταργούνται με το τέλος της συνάρτησης. Συνεπώς, η εμβέλειά τους είναι στο αυστηρό πλαίσιο της συνάρτησης. Αντίθετα, μια μεταβλητή **στατικής κλάσης** (**static class**) δεν "χάνει" την τιμή της κάθε φορά που εκτελείται η συνάρτηση. Οι στατικές μεταβλητές έχουν την ίδια εμβέλεια με τις αυτόματες μεταβλητές.

## Παράδειγμα

```
#include <stdio.h>

void auto_static();

int main()
{
  int i;

  for(i=0; i<3; i++)
    auto_static();
  return 0;
}

void auto_static() /* Αυτόματες/στατικές μεταβλητές */
{
  int auto_var = 1;
  static int static_var = 1;

  printf("auto=%d, static=%d\n",auto_var++,static_var++);
}
```

**Εξοδος προγράμματος:**

```
auto=1, static=1
auto=1, static=2
auto=1, static=3
```

## ΕΞΩΤΕΡΙΚΕΣ – ΚΑΘΟΛΙΚΕΣ ΜΕΤΑΒΛΗΤΕΣ

Οι μεταβλητές που ορίζονται έξω από συναρτήσεις είναι **καθολικής εμβέλειας** και ονομάζονται **εξωτερικές** (**external**) ή καθολικές μεταβλητές. Μια καθολική μεταβλητή πρέπει να δηλώνεται μέσα στη συνάρτηση που τη χρησιμοποιεί με τη λέξη-κλειδί **extern**. Παράδειγμα:

```
int  a,b;      /* Δήλωση 4 καθολικών μεταβλητών */
char gl_ch;
float fl;

int main()
{
  extern int a; /* Δήλωση ότι η a είναι καθολική */
  auto int b;   /* Σκόπιμη δήλωση νέας τοπικής μεταβλητής που
                   έχει το ίδιο όνομα με μια καθολική μεταβλητή */
  extern char gl_ch;
  ...
}
void func1()
{
  extern int a;   /* Δήλωση ότι η a είναι καθολική */
  int  i,j;       /* Δήλωση νέων αυτόματων τοπικών μεταβλητών */
  extern float fl;
  ...
}
```

## Σχολιασμός προγράμματος

Οι μεταβλητές **a** και **gl_ch** έχουν δηλωθεί ως καθολικές μέσα στην **main()**. Όταν οι δηλώσεις των καθολικών μεταβλητών είναι στο ίδιο αρχείο και προηγούνται κάποιας συνάρτησης, τότε δεν είναι απαραίτητο να δηλωθούν ως καθολικές και μέσα στη συνάρτηση. Έτσι, η **fl** μπορεί επίσης να χρησιμοποιηθεί από τη **main()** ως καθολική μεταβλητή. Όμως, επειδή η **b** έχει δηλωθεί μέσα στη **main()** ως auto μεταβλητή, η μεταβλητή αυτή θα είναι διαφορετική από την εξωτερική. Η **main()** δεν θα έχει δυνατότητα προσπέλασης της καθολικής μεταβλητής b.

Αντίστοιχα, η **func1()** μπορεί να προσπελάσει όλες τις καθολικές μεταβλητές (δηλωμένες και αδήλωτες).

# ΑΝΑΔΡΟΜΙΚΕΣ ΣΥΝΑΡΤΗΣΕΙΣ

```
/* Εμφάνιση χαρακτήρων κατ'αντίστροφη σειρά */
#include <stdio.h>
int main()
{
  char ch;

  if((ch = getchar()) != '\n')
    main();       /* Αναδρομική κλήση συνάρτησης */
  putchar(ch);
  return 0;
}
```

**Έξοδος προγράμματος:**

```
recursive program[enter]

margorp evisrucer
```

## Σχολιασμός προγράμματος

Μία συνάρτηση που καλεί τον εαυτό της ονομάζεται **αναδρομική** (**recursive**). Όταν η main() εκτελείται για πρώτη φορά, ένας χαρακτήρας διαβάζεται από το πληκτρολόγιο και εκχωρείται στην τοπική μεταβλητή ch. Στη συνέχεια, αν δεν πληκτρολογήσαμε [enter], η main() καλείται εκ νέου (2η κλήση). Παρατηρήστε ότι η `putchar(ch)` δεν έχει ακόμη εκτελεστεί.

Με τη δεύτερη κλήση της main(), δημιουργείται μια **νέα** τοπική μεταβλητή ch, ένας νέος χαρακτήρας διαβάζεται και εκχωρείται στην ch και ξανακαλείται η main() (3η κλήση) χωρίς ακόμη να έχει εκτελεστεί η putchar(ch) είτε της 1ης είτε της 2ης κλήσης της main(). Η ίδια διαδικασία συνεχίζεται έως ότου πληκτρολογήσουμε [enter]. Τότε, η τελευταία κλήση της main() αποφεύγει την αναδρομή καθώς `ch == '\n'` και συνεχίζει εκτελώντας την `putchar(ch)`. Αυτό έχει ως αποτέλεσμα να τοποθετηθεί ο δρομέας στην αρχή της επόμενης γραμμής και να επιστραφεί ο έλεγχος στην αμέσως προηγούμενη κλήση της main() οπότε εμφανίζεται στην οθόνη ο τελευταίος χαρακτήρας που πληκτρολογήσαμε, ολοκληρώνεται η αναδρομική κλήση και επιστρέφεται ο έλεγχος στην αμέσως προηγούμενη κλήση. Αυτό συνεχίζεται μέχρι να επιστραφεί τελικά ο έλεγχος στην πρώτη κλήση της main() και να εμφανισθεί ο πρώτος χαρακτήρας.

## Οι αναδρομικές κλήσεις της main()

**Παράδειγμα, για είσοδο:** `Char[enter]`

```
1η κλήση: ch=getchar(); /* ch == 'C' */
   2η κλήση: ch=getchar(); /* ch == 'h' */
      3η κλήση: ch=getchar(); /* ch == 'a' */
         4η κλήση: ch=getchar(); /* ch == 'r' */
            5η κλήση: ch=getchar(); /* ch == '\n' */
            5η κλήση: putchar(ch); /* Εμφάνιση '\n' */
         4η κλήση: putchar(ch); /* Εμφάνιση 'r' */
      3η κλήση: putchar(ch); /* Εμφάνιση 'a' */
   2η κλήση: putchar(ch); /* Εμφάνιση 'h' */
1η κλήση: putchar(ch); /* Εμφάνιση 'C' */
```

## Άλλο παράδειγμα αναδρομής

```
/* Εύρεση στοιχείων της ακολουθίας Fibonacci */
#include <stdio.h>

int Fibonacci(int element);

int main()
{
  int stoixeio;

  printf("Poio Fibonacci stoixeio thelete?\n");
  scanf("%d",&stoixeio);
  printf("Fibo[%d] = %d\n",stoixeio,Fibonacci(stoixeio));
  return 0;
}

int Fibonacci(int element)  /* Αναδρομική συνάρτηση */
{
  if(element == 1 || element == 2)
    return(1);
  else
    return(Fibonacci(element-1) + Fibonacci(element-2));
}
```