Συναρτήσεις (functions)

Δομή προγράμματος & Εμβέλεια μεταβλητών

Μάθημα 6

Βασικό λεξιλόγιο της C και συναρτήσεις

- Λέξεις-κλειδιά (keywords) της γλώσσας

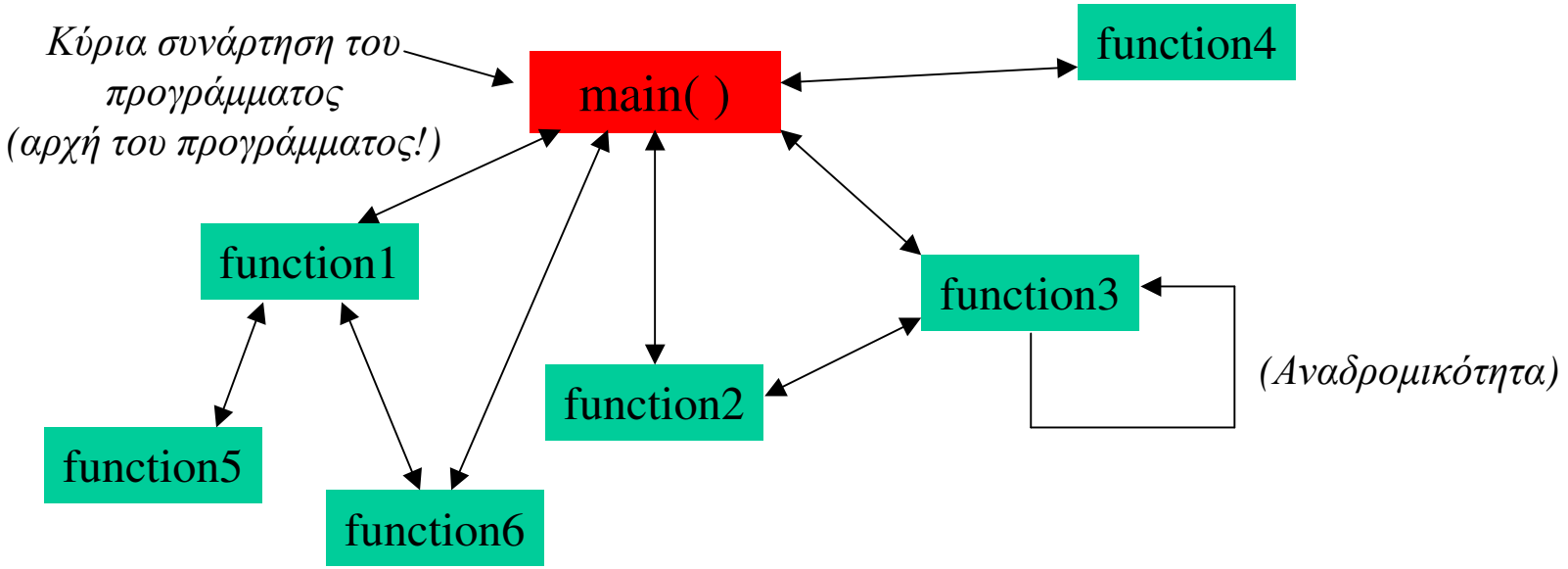
Αποτελούν δεσμευμένες λέξεις της γλώσσας και δεν επιτρέπεται να τις δηλώσει διαφορετικά ο προγραμματιστής

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Οι υπόλοιπες λέξεις-εντολές που εμφανίζονται στο πρόγραμμα καλούν **συναρτήσεις**, δηλαδή «*μικρά προγράμματα*» τα οποία έχουν γραφεί με τις παραπάνω λέξεις κλειδιά είτε από το χρήστη είτε από τους κατασκευαστές της C.

Συναρτήσεις και δομή προγράμματος

Οι **συναρτήσεις** αποτελούν μικρά τμήματα ενός εκτεταμένου προγράμματος στη C. Σε άλλες γλώσσες ονομάζονται και «υποπρογράμματα» ή «υπορουτίνες»



▪Κάθε συνάρτηση μπορεί να καλεί οποιαδήποτε άλλη συνάρτηση, ακόμα και τον εαυτό της. Όμως σε έναν σωστό σχεδιασμό προγράμματος οι συναρτήσεις ίσως θα πρέπει να ιεραρχούνται κατάλληλα.

Γενικά Χαρακτηριστικά

- Οι εντολές ενός προγράμματος (εκτός των λέξεων κλειδιών) αποτελούν συναρτήσεις που είτε μας παρέχονται έτοιμες από την βιβλιοθήκη της C (πχ. printf, getch, sqrt, strcpy κλπ) είτε από βιβλιοθήκη κάποιων τρίτων ή από συναρτήσεις του χρήστη.

- Γενικά ένα πρόγραμμα στη C πρέπει να αποτελείται από πολλές μικρές συναρτήσεις (οι οποίες εύκολα ελέγχονται ότι είναι σωστές) και όχι λίγες και μεγάλες.

- Οι συναρτήσεις μπορεί να είναι γραμμένες σε διαφορετικά αρχεία ".c". Το κάθε αρχείο μεταγλωττίζεται ξεχωριστά και στο τέλος ο διασυνδετής (Linker) συναρμολογεί όλους τους κώδικές σε έναν.

- Μια συνάρτηση πρέπει να είναι (όταν είναι δυνατό) αυτόνομη, δηλαδή να μην εξαρτάται από άλλα στοιχεία του προγράμματος. Έτσι μπορούμε να την χρησιμοποιήσουμε και σε άλλα προγράμματα (όπως είναι) χωρίς καμία μετατροπή και βέβαιοι πως δεν έχει λάθη αφού έχει ελεγχθεί επιμελώς.

Ορισμός Συνάρτησης

```
Επιστρεφόμενος_τύπος  όνομα_Συνάρτησης  (  δηλώσεις ορισμάτων )
{
        …. σώμα συνάρτησης ….
        return τιμή;
}
```

return : επιστρέφει μια τιμή (του τύπου της συνάρτησης)
Όνομα : Κάθε συνάρτηση έχει ένα όνομα που ακολουθεί τους κανόνες ονομάτων των μεταβλητών
Ορίσματα : Μεταβλητές που μεταφέρουν τιμές μέσα στην συνάρτηση για τους υπολογισμούς της συνάρτησης (δεν είναι απαραίτητες).

Επιστρεφόμενος τύπος **`void`** : Η συνάρτηση δεν επιστρέφει τιμή (δεν απαιτείται η εντολή `return`)

* Η εντολή `return` στην `main` προκαλεί έξοδο από το πρόγραμμα. **Έξοδος** από το πρόγραμμα μέσα από οποιαδήποτε συνάρτηση γίνεται με την χρήση της συνάρτησης **`exit(n)`** , όπου *n* ακέραιος, της καθιερωμένης βιβλιοθήκης

Ορισμός, Δήλωση και Κάλεσμα συνάρτησης

```c
/* prog61example.c */
#include <stdio.h>

int myAverage(double a, double b); /* declaration */

void wait_enter_to_exit()  /* definition */
{
      fflush(stdin);
      printf("Press Enter to exit");
      getchar();
      exit(1);
}

int main()
{
      double x,y;
      int n,k;
      printf("--- An example: how to use simple functions ---
      n=myAverage(15,27);
      printf("The average of 15 and 27 is %d\n",n);
      printf("input two numbers : "); scanf("%lf %lf",&x,&y);
      k=myAverage(x,y);
      printf("My average of x=%f and y=%f is k=%d\n",x,y,k);
      wait_enter_to_exit();
      return 0;
}

int myAverage(double a, double b) /*definition*/
{
        return (int)(0.5*(a+b+1));
}
```

Δήλωση Συνάρτησης 3

Ορισμός Συνάρτησης 1

Ορισμός Συνάρτησης 2

Ορισμός Συνάρτησης 3

Καλούσα : Η συνάρτηση που καλεί μια συνάρτηση
Καλούμενη : Η συνάρτηση που καλείται από μια συνάρτηση

Τις συναρτήσεις του προγράμματός μας τις γράφουμε με όποια σειρά επιθυμούμε

Ο μεταγλωττιστής αναγνωρίζει το όνομα μιας καλούμενης συνάρτησης αν αυτή έχει οριστεί ή δηλωθεί πριν την καλούσα συνάρτηση.

Στη δήλωση μιας συνάρτησης μπορούμε να παραλείψουμε τα ονόματα των μεταβλητών πχ γράφουμε
`int myAverage(double,double);`

Συναρτήσεις και μνήμη στοίβας (stack memory)

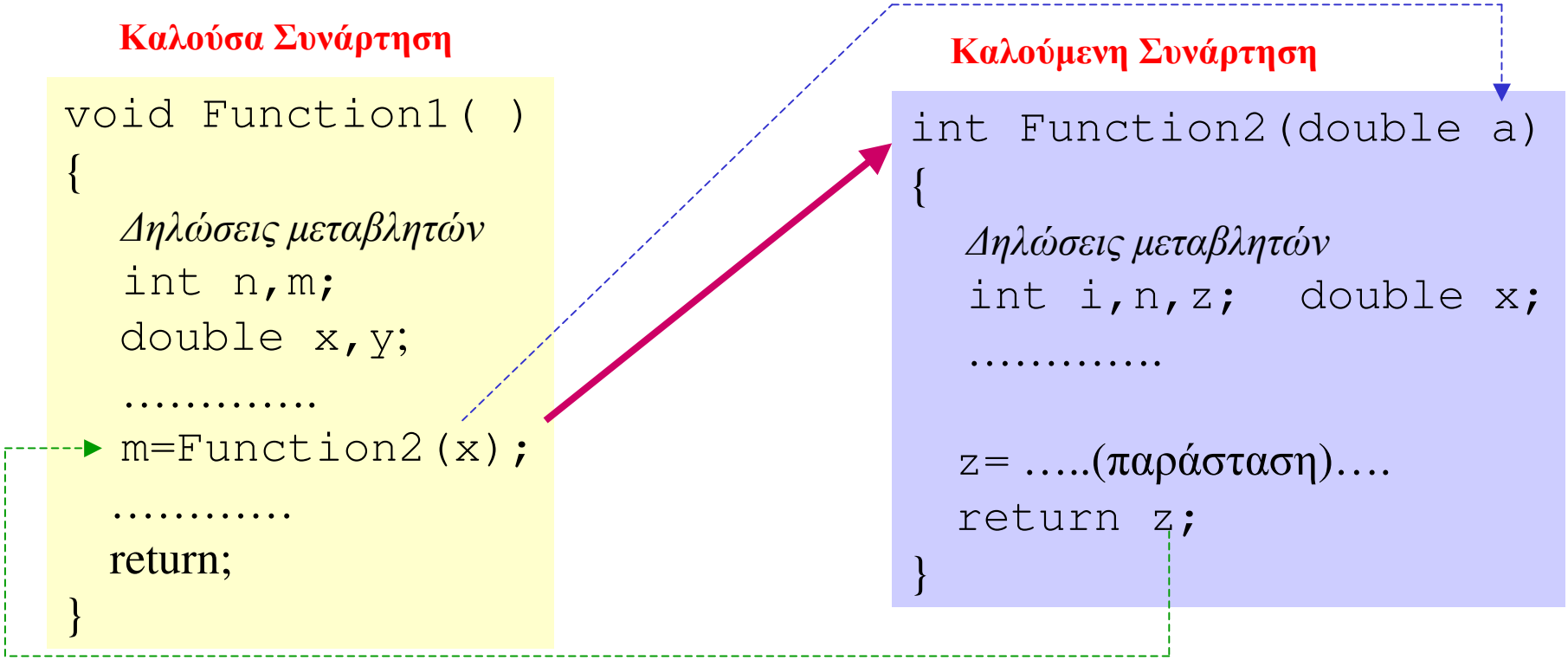
Όταν καλείται μια συνάρτηση δημιουργείται ένα **αντίγραφο** του κώδικά της στη μνήμη στοίβας. Εκεί εκτελείται και αποθηκεύονται τα δεδομένα της. Όταν εκτελεστεί η συνάρτηση αυτό το αντίγραφο σβήνει και ενημερώνεται με την επιστρεφόμενη τιμή η καλούσα συνάρτηση.

•Η συνάρτηση **main( )** καλείται πρώτη από τον κώδικα της C και παραμένει στη μνήμη στοίβας μέχρι τον τερματισμό του προγράμματος. Μπορεί να είναι τύπου **void** ή τύπου **integer** (εξαρτάται από το μεταγλωττιστή)

•Αν δεν οριστεί τύπος για μια συνάρτηση τότε θεωρείται ότι είναι τύπου integer (αυτό ίσως να μην είναι δεκτό απ' όλους τους μεταγλωττιστές)

•Αν μια συνάρτηση καλέσει τον εαυτό της, και πάλι δημιουργείται ένα δεύτερο αντίγραφο.

Συναρτήσεις και Τοπικές μεταβλητές



- Σε κάθε συνάρτηση μπορούν να ορίζονται μεταβλητές οι οποίες ισχύουν μόνο μέσα στη συνάρτηση και ονομάζονται **τοπικές μεταβλητές**.. Δύο μεταβλητές με το ίδιο όνομα που δηλώνονται σε διαφορετικές συναρτήσεις είναι διαφορετικές μεταβλητές.
- Οι παράμετροι αποτελούν τοπικές μεταβλητές για την καλούμενη συνάρτηση
- Το κάλεσμα μιας συνάρτησης (πχ η Function2 της καλούσας συνάρτησης) θεωρείται ως μια «τιμή» για την καλούσα συνάρτηση που είναι η επιστρεφόμενη τιμή της καλούμενης συνάρτησης
- Η τιμή μιας καλούμενης συνάρτησης αποδίδεται ή όχι σε μια μεταβλητή της καλούσας συνάρτησης

Εμβέλεια τοπικών μεταβλητών -παράδειγμα

```c
/* prog62example.c, test local variables */

#include <stdio.h>

double myTestFunction(int n, double x, double y)
{
    int k;
    double z;
    printf(" --- MyFuction Variables -----\n");
    printf("n=%d  x=%f   y=%f\n",n,x,y);
    k=n*100;
    z=y=x;
    printf("k=%d  x=%f   y=%f   z=%f\n",k,x,y);
    return z;
}

main()
{
      int k;
      double x=2.0, y=3.0, z=10.0;
      printf(" ---  variables in Main (1) -----\n");
      printf("x=%f   y=%f   z=%f\n",x,y,z);
      z=myTestFunction(2,y,x);
      printf(" ---  variables in Main (2) -----\
      printf("x=%f   y=%f   z=%f\n",x,y,z);
      printf("k=%d\n",k);
      getchar();
}
```

Τοπικές μεταβλητές
n,x,y,k,z

Ίδια ονόματα αλλά διαφορετικές μεταβλητές

Τοπικές μεταβλητές
k,x,y,z

Από τις μεταβλητές της main( ) μόνο η *z* επηρεάζεται από τον κώδικα της myTestFunction

```
C:\RunCourses\Cprog\Mathima6_functions\prog62localv...
 ---  variables in Main (1) -----
x=2.000000   y=3.000000   z=10.000000

--- MyFuction Variables -----
n=2  x=3.000000   y=2.000000
k=200  x=3.000000   y=3.000000   z=3.000000

 ---  variables in Main (2) -----
x=2.000000   y=3.000000   z=3.000000
k=2
```

Μεταβλητές και εμβέλεια στον κώδικα

```
{
        int a;
        double b;
        ..............
        ..............
}
```

Κάθε μεταβλητή που ορίζεται σε ένα **μπλοκ** κώδικα είναι μια **τοπική** μεταβλητή που έχει κύρος και αναγνωρίζεται μόνο μέσα στο μπλοκ .

πχ αν μια μεταβλητή δηλωθεί στο μπλοκ μιας συνάρτησης τότε είναι μια τοπική μεταβλητή για αυτή τη συνάρτηση και αναγνωρίζεται μόνο από αυτή

```
int X;

Function1()
{
        ..............
}

Function2()
{
        ..............
}
```

Κάθε μεταβλητή που ορίζεται έξω από τα μπλοκ των συναρτήσεων έχει κύρος και αναγνωρίζεται σε όλες τις συναρτήσεις του κώδικα που ακολουθούν. Μια τέτοια μεταβλητή ονομάζεται καθολική.

!Αν σε ένα μπλοκ ορίσουμε μια τοπική μεταβλητή που έχει το ίδιο όνομα με κάποια καθολική μεταβλητή του κώδικα, τότε μέσα στο μπλοκ είναι έγκυρη η τοπική μεταβλητή.

Βλ. κώδικα prog62variables.c

Παραδείγματα

```
int IsPrime(int n)
{
    int k;
    for(k=2; k<=n/2; k++)
                if(!(n%k)) return 0;
    return 1;
}
```

Η `IsPrime` δέχεται ένα ακέραιο και ελέγχει αν είναι πρώτος αριθμός. Επιστρέφει TRUE (1) ή FALSE (2)

```
long unsigned int Factorial(int n)
{
     int i;
     if(n<=0||n>12) return 0;
     long unsigned int f=1;
     for(i=2;i<=n;i++) f*=i;
     return f;
}
```

Η `Factorial` δέχεται ένα ακέραιο και επιστρέφει το παραγοντικό του.

Η `myPow` δέχεται έναν πραγματικό αριθμό `x` και έναν ακέραιο `n` και επιστρέφει την τιμή της δύναμης x^n.

```
double myPow(double x, int n)
{
        double y; int i,nn;
        if(n==0) return 1.0;
        nn=abs(n);
        y=x;
        for(i=2;i<=nn;i++) y*=x;
        if(n>0) return y; else return 1.0/y;
}
```

Δες κώδικες
prog63example1.c
prog63example2.c

Ασκήσεις

Άσκηση 6.10. Κάντε μια συνάρτηση που να δέχεται δύο ακέραιους και να επιστρέφει 1 αν είναι πρώτοι μεταξύ τους ή 0 αν δεν είναι.

Άσκηση 6.20. Μετατρέψτε κατάλληλα την συνάρτηση `Factorial` και φτιάξτε την `Factorialf`, η οποία να επιστέφει το παραγοντικό ενός ακεραίου n ως αριθμό double. Έτσι μπορούμε να υπολογίσουμε το παραγοντικό για ακεραίους $n>12$ κατά προσέγγιση.

Άσκηση 6.22. Χρησιμοποιήστε τις συναρτήσεις `myPow` και `Factorialf` και υπολογίστε για δεδομένες τιμές x και N (που δίνει ο χρήστης) το άθροισμα

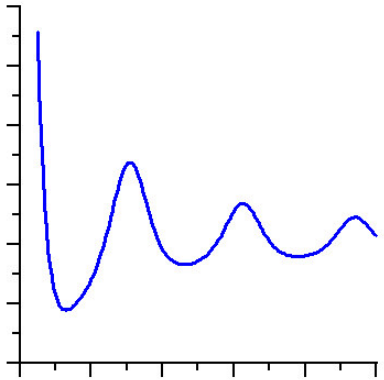
$$A = \sum_{k=1}^{N} \frac{x^k}{k!}$$

Διαπιστώστε ότι πάντα η σειρά συγκλίνει καθώς μεγαλώνει το N.

Άσκηση 6.23. Κατασκευάστε την συνάρτηση

$$A(x) = \frac{1}{x} \sum_{k=1}^{\infty} \frac{(\cos x)^k}{k!}$$

Για κάθε τιμή του x στο διάστημα [1,20], με βήμα Δx=0.1, υπολογίστε την $y=A(x)$. Αποθηκεύστε τις τιμές (x,y) σε ένα αρχείο και σχεδιάστε (με ένα πρόγραμμα σχεδίασης) την γραφική παράσταση.



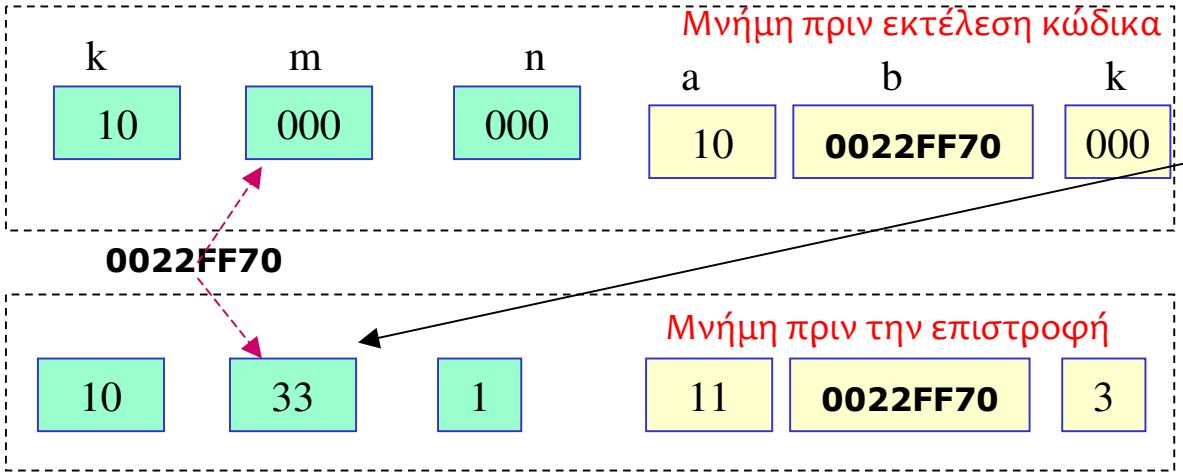
Δείκτες ως παράμετροι συνάρτησης

```
main()
{
        int k=10, m, n;
        printf("before:  k=%d   m=%d   (at memory address %p)\n",k,m,&m);
        n=MyFunction(k,&m);
        printf("After :  k=%d   m=%d   (returned value n=%d)\n",k,m,n);
        getchar();
}
```

```
int MyFunction(int a, int *b)
{
     int k=3;
     a++;
     *b=k*a;
     return 1;
}
```

10 0022FF70

Τοπικές μεταβλητές: **main( )** **MyFunction**

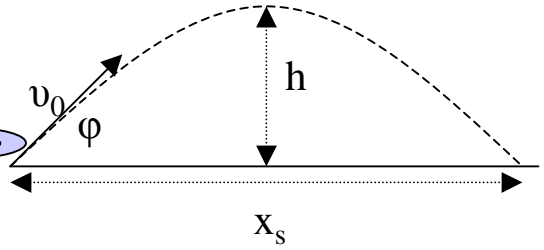


κώδικας
prog65pointer.c

*Με τη χρήση δεικτών ως παραμέτρους συνάρτησης μπορούμε να αλλάζουμε **οποιεσδήποτε τιμές** μιας συνάρτησης από μια **άλλη οποιαδήποτε** συνάρτηση.

Παράδειγμα

Δίνουμε αρχική ταχύτητα και γωνία βολής και υπολογίζουμε βεληνεκές και μέγιστο ύψος




```c
/* example660shooting.c */

#include <stdio.h>
#include <math.h>

#define pi  3.14159
#define g   9.81

int shootingrange(double v0, double phi, double *zmax, double *xmax);

main()
{
      double speed, angle, h, xs;
      int error;
      printf("--- Cannon Shooting -----\n");
      do{
         printf("Initial speed = "); scanf("%lf",&speed);
         printf("Initial angle (deg) = "); scanf("%lf",&angle);
         error=shootingrange(speed,angle,&h,&xs);
         if(error) printf("Data Error\n");
         else {
           printf("maximum height  =%f\n",h);
           printf("maximum distance=%f\n",xs);
         }
         printf("\n\n");
      }while(speed!=0&&angle!=0);
}
```

```c
double shootingtime(double vv)
/*vv: vertical velocity component*/
{
     return 2*vv/g;
}
```

```c
int shootingrange(double v0, double ph
{
     double vx, vy, tv;
     vx=v0*cos(phi*pi/180);
     vy=v0*sin(phi*pi/180);
     if(vy<0) return -1; /*error*/
     tv=shootingtime(vy);
     *zmax=vy*vy/(2*g);
     *xmax=vx*tv;
     return 0; /* no error */
}
```

Πίνακες ως παράμετροι συνάρτησης

- Η μεταβλητή ενός πίνακα είναι δείκτης. Συνεπώς το πέρασμα ενός **πίνακα** σε μια καλούμενη συνάρτηση είναι **ισοδύναμο** με τη χρήση μιας παραμέτρου **δείκτη**.
- Η καλούμενη συνάρτηση δεν γνωρίζει τον αριθμό των στοιχείων του πίνακα. Μπορούμε να τον δίνουμε με μια επιπλέον ακέραια παράμετρο.

```
void Function1( )
{
   double x[10],y;
   ..............
   y=Function2(10,x);
   ..............
   return;
}
```

```
double Function2(int n, double *A)
{
   double z;
   ........................
   z= …παράσταση, πχ A[0]+A[1]+ ….
   return z;
}
```

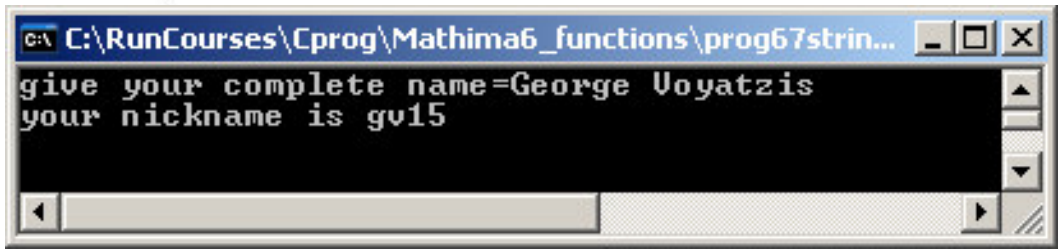
- Η `Function2` θα μπορούσε να οριστεί και ως `Function2(int n, double A[])`
- Τα στοιχεία `A[i]`, μέσα στην `Function2` είναι τα ίδια με τα στοιχεία `x[i]` της `Function1`

Βλ. κώδικα prog67arraypass.c

Αλφαριθμητικά ως παράμετροι συνάρτησης

- Ένα αλφαριθμητικό (string) είναι πίνακας χαρακτήρων. Συνεπώς το πέρασμα ενός **αλφαριθμητικού** σε μια καλούμενη συνάρτηση είναι **ισοδύναμο** με το πέρασμα ενός **πίνακα**

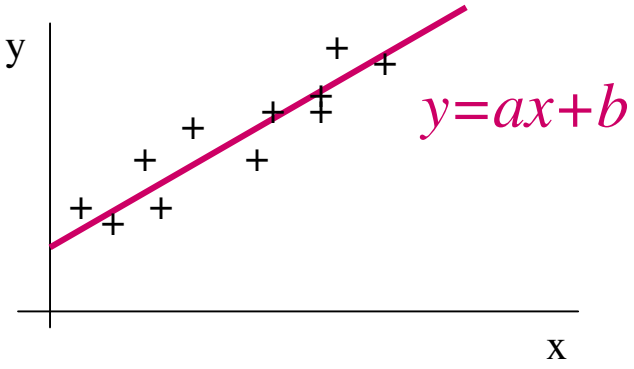
```
int GetNickName(char name[], char nick[])
/* char name[] is the complete name of a person (input)
   char nick[] is the computed nickname (output)
   return 0 when is Ok or a negative integer on error */
{
    int i,n;
    char ss[3], *p;
    n=strlen(name); if(n<3) return -1;
    nick[0]=tolower(name[0]);
    p=strchr(name,' ');
    if(p==NULL) return -2;
    nick[1]=tolower((++p)[0]);
    if(nick[1]==' '||nick[1]=='\0') return -3;
    nick[2]='\0';
    itoa(n,ss,10); strcat(nick,ss);
    return 0;
}
```



Βλ. κώδικα prog67string.c

Εφαρμογή: Μέθοδος Ελαχίστων Τετραγώνων

x_1	y_1
x_2	y_2
x_3	y_3
….	…
x_N	y_N



$$a = \frac{N\sum_{i=1}^{N} x_i y_i - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} y_i}{N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2}$$

$$b = \frac{\sum_{i=1}^{N} x_i^2 \sum_{i=1}^{N} y_i - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} x_i y_i}{N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2}$$

Ζητούμενα

*Διαβάζουμε τα δεδομένα από ένα αρχείο κειμένου το οποίο έχει αποθηκευμένα τα x_i και y_i σε δύο στήλες.

* Κατασκευάζουμε μια συνάρτηση που μας δίνει τους συντελεστές a και b.

* Εκτυπώνουμε το αποτέλεσμα στην οθόνη

Εφαρμογή: Μέθοδος Ελαχίστων Τετραγώνων

Σχεδίαση : Κύριο μέρος

Μεταβλητές

`x[]`, `y[]` **για την αποθήκευση δεδομένων.**
`N`: **αριθμός δεδομένων**
`filename` : **όνομα αρχείου**
`a` , `b` : **ζητούμενοι συντελεστές**

Συναρτήσεις

`ReadData`. Δίνουμε το όνομα αρχείου και τους δείκτες των πινάκων `x` και `y`. Η συνάρτηση επιστρέφει των αριθμό των δεδομένων στο αρχείο ή αριθμό <=0 σε περίπτωση λάθους.

`MyLSQfunction`: Δίνουμε ως παραμέτρους τα δεδομένα μας καθώς και τους δείκτες των μεταβλητών `a` και `b`. Οι μεταβλητές αυτές θα ενημερωθούν από την συνάρτηση.

```
int ReadData(char *, double [ ], double [ ]);
void MyLSQfunction(int, double [ ], double [ ], double *, double *);
```

```c
/*example67lsq.c*/
#include <stdio.h>
#define NMAX 1000

int ReadData(char*, double x[], double y[]);
void myLSQfunction(int,double [],double [],double*,double*);

main()
{
        char filename[132]; /* ονομα αρχείου δεδομενων */
        double x[NMAX], y[NMAX], a, b;
        int N; /* σύνολο δεδομένων */

        printf("Dwse onoma arxeiou  : "); gets(filename);
        N=ReadData(filename, x, y);
        if(N<=1) {
          printf("Error  (%d) \n",N); getchar(); exit(1);
        }
        printf("Number of Data = %d\n",N);
        myLSQfunction(N,x,y,&a,&b);
        printf("LSQ coefficients   a=%f    b=%f\n",a,b);
        getchar();
}
```

Εφαρμογή: Μέθοδος Ελαχίστων Τετραγώνων

Συνάρτηση ReadData

```
int ReadData(char *name, double x[], double y[])
{
    FILE *fdata;
    int counter=0;
    double xt,yt;

    fdata=fopen(name,"rt");
    if(fdata==NULL) return -1;  /* error */
    while(!feof(fdata)){
        int n;
        n=fscanf(fdata,"%lf  %lf\n",&xt,&yt);
        if(n!=2) {fclose(fdata); return counter;}
        x[counter]=xt; y[counter]=yt; counter++;
    }
    fclose(fdata);
    return counter;
}
```

Μετρητής δεδομένων του αρχείου

Τοπικές μεταβλητές για την αποθήκευση των δεδομένων μιας γραμμής

Επανέλαβε μέχρι το τέλος αρχείου

Διάβασμα γραμμής αρχείου

Λάθος,, τερματισμός

Ok, αποθήκευση στο πίνακα και αύξηση μετρητή

- Άνοιξε αρχείο
- Διάβαζε μέχρι τέλος αρχείου
- Διάβασε μια γραμμή με *x,y*
  και βάλ' τα στις τοπικές μεταβλητές. `xt, yt`)
- Αν **δεν** διαβάστηκαν δυο στοιχεία, τερμάτισε : κλείσε αρχείο και επέστρεψε τον αριθμό των δεδομένων που έχουν διαβαστεί σωστά.
- Αν **Ok**, ενημέρωσε τους πίνακες `x, y` με τις τιμές `xt, yt` και αύξησε τον μετρητή

Εφαρμογή: Μέθοδος Ελαχίστων Τετραγώνων

```
void myLSQfunction(int n, double X[], double Y[], double *a, double *b)
{
      double sx, sy, sxy, sx2, denom;
      sx=sumarray1(n,X);
      sy=sumarray1(n,Y);
      sxy=sumarray2(n,X,Y);
      sx2=sumarray2(n,X,X);
      denom=n*sx2-sx*sx;
      *a=(n*sxy-sx*sy)/denom;
      *b=(sy*sx2-sx*sxy)/denom;
}
```

Συνάρτηση myLSQfunction

- Υπολογίζονται τα απαραίτητα αθροίσματα των πινάκων δεδομένων X και Y
- Υπολογίζεται ο παρανομαστής των τύπων των συντελεστών (είναι κοινός για τους δύο τύπους)
- Υπολογίζονται, βάση των τύπων, οι συντελεστές και ενημερώνονται με αυτούς οι θέσεις μνήμης όπου δείχνουν οι δείκτες a και b

```
double sumarray1(int n, double A[])
{
      double sum=0.0; int i;
      for(i=0;i<n;i++) sum+=A[i];
      return sum;
}

double sumarray2(int n, double A[], double B[])
{
      double sum=0.0; int i;
      for(i=0;i<n;i++) sum+=A[i]*B[i];
      return sum;
}
```

*Ο υπολογισμός των αθροισμάτων γίνεται με τις βοηθητικές συναρτήσεις

sumarray1: επιστρέφει το άθροισμα των *n* (αρχικών) στοιχείων ενός πίνακα *A*

sumarray2: επιστρέφει το άθροισμα των γινομένων των *n* (αρχικών) στοιχείων δύο πινάκων A και B.

Βλ. κώδικα example67lsq.c

Δισδιάστατοι Πίνακες ως παράμετροι συνάρτησης

Έστω πίνακας **`int A[N][M]`** , (*N* γραμμές, *M* στήλες)

Παράμετρος στο κάλεσμα της συνάρτησης **`Function(A);`**

Παράμετρος στον ορισμό της συνάρτησης **`int Function(int A[][M])`**

```
int A[3][4]={{3,2,7,1},{1,4,8,9},{0,1,0,2}};
int B[4][3];

printMyarray4col(3,A);

CalcTranspose34(A,B);
```

Πχ Εκτύπωση πίνακα τεσσάρων στηλών

```
void printMyarray4col(int rows, int x[][4])
{
      int i,j;
      const int M=4;
      for(i=0;i<rows;i++) {
            for(j=0;j<M;j++) printf("%4d",x[i][j]);
            printf("\n");
      }
}
```

```
void CalcTranspose34(int X[][4], int Y[][3])
{
      const int N=3; /*rows of x, columns of y*/
      const int M=4; /*columns of x, rows of y*/
      int i,j;
      for(i=0;i<N;i++) for(j=0;j<M;j++) Y[j][i]=X[i][j];
}
```

Πχ Υπολογισμός του ανάστροφου πίνακα ενός πίνακα X (3x4), *παίρνουμε τον* $Y=X^T$ (4x3)

Βλ. κώδικα prog68array2d.c

Ασκήσεις

6.40. Κάντε μια συνάρτηση που να δέχεται ως ορίσματα το μήκος της υποτείνουσας και μια οξεία γωνία ενός ορθογωνίου τριγώνου. Το πρόγραμμα να επιστρέφει το μήκος της προσκείμενης και της απέναντι πλευράς.

6.50. Να κατασκευαστεί μια συνάρτηση η οποία να δέχεται έναν ακέραιο Κ και έναν πίνακα ακεραίων Α, να εξετάζει αν ο αριθμός Κ είναι στοιχείο του πίνακα Α και να επιστρέφει TRUE ή FALSE (υπάρχει ή δεν υπάρχει, αντίστοιχα)

6.51. Να κατασκευαστεί μια συνάρτηση η οποία να δέχεται δύο πίνακες ακεραίων και να επιστρέφει τον αριθμό *n* των στοιχείων τους που είναι κοινά.

6.52. Σε συνέχεια της άσκησης 6.51 κάντε ένα παιχνίδι Lotto : Να δίνει ο χρήστης *N* αριθμούς από το 1 έως το 49, το πρόγραμμα να βρίσκει 6 τυχαίους αριθμούς στο ίδιο διάστημα και στη συνέχεια να ελέγχει τον αριθμό των επιτυχιών.

Ασκήσεις

6.60. Να κατασκευαστεί μια συνάρτηση η οποία να υπολογίζει και να επιστρέφει την ορίζουσα ενός πίνακα NxN (όπου N: σταθερά του προγράμματος).

6.61. Να κατασκευαστεί μια συνάρτηση η οποία να δέχεται ένα πίνακα A[N][M] και δύο ακεραίους *K*,*L*. Η συνάρτηση να μεταθέτει την στήλη *K* του πίνακα στη στήλη *L*, και το αντίστροφο. Η συνάρτηση να επιστρέφει 1 (Ok) ή 0 (λάθος, πχ γιατί K>M)

*6.70. Να κατασκευαστεί μια συνάρτηση η οποία να εκτυπώνει τα στοιχεία ενός δισδιάστατου πίνακα (για οποιαδήποτε διάσταση)

Αναδρομικότητα

Μια συνάρτηση μπορεί να καλεί τον εαυτό της.

- Κάθε φορά δημιουργείτε και ένα αντίγραφο της συνάρτησης στη μνήμη stack. Αν δημιουργηθούν πολλά αντίγραφα (ο αριθμός εξαρτάται από το σύστημα) παίρνουμε το μήνυμα λάθους "stack overflow".
- Πρέπει η σχεδίαση μια τέτοιας συνάρτησης να είναι σωστή ώστε κάποια στιγμή να ξεκινήσει η διαδικασία των "return" σε κάθε αντίγραφο που έχει δημιουργηθεί.

```c
/*prog69recur.c*/
#include <stdio.h>

long unsigned int factorial(int n)
{
        if(n==1) return 1;
        else return n*factorial(n-1);
}

int main()
{
     int N=13; /*N must be less than 14*/
     printf("%lu\n",factorial(N));
     getchar();
}
```

Δηλώσεις Μεταβλητών “extern” και “static”

```
/* myFile1.c */

double g=9.81;

main( )
{
   ……..
    …..g….
    ……..
}
```

```
/* myFile2.c */

extern double g;

myFunction1( )
{
  …..g….
 }

myFunction2( )
{
  …..g….
 }
```

Έστω ότι ένας κώδικας αποτελείται από συναρτήσεις που βρίσκονται σε διαφορετικά αρχεία c και το κάθε αρχείο έχει δηλωμένες κάποιες καθολικές μεταβλητές.
ΤΟΤΕ

- Κάθε καθολική μεταβλητή πρέπει να είναι δηλωμένη μόνο σε ένα αρχείο.
- Οι συναρτήσεις ενός αρχείου του κώδικα μπορούν να χρησιμοποιήσουν μια καθολική μεταβλητή, που δηλώθηκε σε άλλο αρχείο, με την χρήση της δήλωσης **extern**.

Αν μια μεταβλητή δηλωθεί μέσα σε μια συνάρτηση με τον χαρακτηρισμό **static**, τότε είναι μια τοπική μεταβλητή της συνάρτησης που διατηρείται στη μνήμη όταν τερματιστεί η συνάρτηση. Όταν ξανακαλέσουμε την συνάρτηση η *static μεταβλητή* έχει την τιμή από το προηγούμενο κάλεσμα της συνάρτησης. Αν σε μια *μεταβλητή static* δώσουμε τιμή κατά την δήλωση, αυτή αποθηκεύεται στη μεταβλητή μόνο κατά το πρώτο κάλεσμα της συνάρτησης

```
int myFunction( )
{
    static int k=1;
  …. …. ….
  … k= ……
}
```

Η συνάρτηση main( )

Η συνάρτηση main μπορεί να οριστεί με δύο ορίσματα

```
int  main(int argc, char *argv[])
```

Αριθμός αλφαριθμητικών ορισμάτων

Πίνακας με τα αλφαριθμητικά ορίσματα

Το πρώτο όρισμα **argv[0]** περιέχει το όνομα του προγράμματος τα υπόλοιπα τα δίνει ο χρήστης όταν καλεί το πρόγραμμα, είτε από την κονσόλα είτε μέσω ενός "batch" αρχείου. Το όρισμα **argc** παίρνει αυτόματα τιμή.

Ένα πρόγραμμα που εκτυπώνει τα ορίσματά του

```c
/*prog69main.c*/
#include <stdio.h>

int main(int argc, char *argv[])
{
    int i;
    if(argc==1) printf("There are no arguments except program name\n");
    for(i=0;i<argc;i++) printf("%Argument %d = %s\n",i,argv[i]);
    getchar();
}
```