Οι συναρτήσεις στη γλώσσα C

Οι συναρτήσεις αποτελούν τα βασικότερα στοιχεία της γλώσσας προγραμματισμού C.

Με τη βοήθεια των συναρτήσεων αναπτύσσεται όλη η δραστηριότητα των προγραμμάτων

Ο γενικός τύπος μιας συνάρτησης μπορεί να γίνει:

```
όνομα_ συνάρτησης ()
/* ταυτόχρονη δήλωση του τύπου και του ονόματος των παραμέτρων */
{
    σώμα της συνάρτησης (διάφορες εντολές) ;
 }
```

- Ο αριθμός των παραμέτρων μπορεί να είναι από 0 μέχρι 255.
- Αν δεν υπάρχουν καθόλου παράμετροι δεν χρειάζεται η δήλωσή τους.

```
π.χ.   max_number( a, b, c)
         int a, b, c;
ή
       max_number( int a, int b, int c)
```

Τιμές επιστροφής

- '0λες οι συναρτήσεις επιστρέφουν μία τιμή.
- Αυτή η τιμή μπορεί είτε να προσδιορισθεί απόλυτα από την εντολή return, είτε να είναι 0, αν δεν προσδιορίζεται καμιά άλλη τιμή.
- Ελλείψει προσδιορισμού του τύπου της επιστροφής οι συναρτήσεις θα επιστρέψουν ακέραιες τιμές.

Συναρτήσεις Επιστροφής Μη-Ακεραίων Τιμών

Ο γενικός τύπος δήλωσης μιας συνάρτησης είναι:

```
τύπος   όνομα_συνάρτησης (λίστα παραμέτρων)
δήλωση παραμέτρων;
{
  εντολές της συνάρτησης;
 }
```

Παράδειγμα

```
float fsum(float x, float y)
{
  return(x+y);
  }
```

Ορισμός και χρήση μιας συνάρτησης

```
#include <stdio.h>
 float sum( float a, float b ){
      return (a + b);
  }
  int main( ) {
       float x, y, z ;
       x = 10.5 ;
       y = 20.436 ;
       z = sum (x, y);
   }
```

Χρήση των τιμών επιστροφής των συναρτήσεων

```
#include <stdio.h>
int  mul (int a, int b){
    return (a*b) ;
}

 int main (int x, int y, int z) {
    x=10;
    y=20;
    z = mul (x, y);
    printf("%d\n", mul (x, y) );
    return(0);
 }
```

Παράδειγμα πρωθύστερης δήλωσης

```
#include <stdio.h>
float sum(float a, float b);
int main( ){
    float first, second;
    first=123.23;
    second=99.09;
    printf("%f\n", sum(first, second)) ;
}
float sum(float a, float b){
     return(a+b);
}
```

Συνάρτηση υπολογισμού τετραγώνου αριθμού

```
1 #include <stdio.h>
2 int square(int y); // function prototype
3 int main(void)
4 {
5     // loop 10 times and calculate and output square of x each time
6     for (int x = 1; x <= 10; ++x) {
7         printf("%d ", square(x)); // function call
8     }
9     puts("");
10 }
11 // square function definition returns the square of its parameter
12 int square(int y) // y is a copy of the argument to the function
13 {
14     return y * y; // returns the square of y as an int
15 }
```

Αποτέλεσμα

1 4 9 16 25 36 49 64 81 100

Συνάρτηση υπολογισμού μέγιστης τιμής

```
1 #include <stdio.h>
2 int maximum(int x, int y, int z); // function prototype
3 int main(void)
4 {
5     int number1; //first integer entered by the user
6     int number2; //second integer entered by the user
7     int number3; //third integer entered by the user
8     printf("%s", "Enter three integers: ");
9     scanf("%d%d%d", &number1, &number2, &number3);
10    // number1, number2 and number3 are arguments
11    // to the maximum function call
12    printf("Maximum is: %d\n", maximum(number1, number2,
13           number3));
14 }
```

Συνάρτηση υπολογισμού μέγιστης τιμής

```
1 int maximum(int x, int y, int z)
2 {
3 int main(void)
4 {
5     int max = x; // assume x is largest
6     if (y > max){ // if y is larger than max
7        max = y; //assign y to max
8     }
9     if (z > max){ // if z is larger than max;
10        max = y; //assign y to max
11    }
12    return(max); // max is largest value
13
14 }
```

Αποτέλεσμα

Enter three integers: 22 85 17

Maximum is: 85

Enter three integers: 47 32 14

Maximum is: 47

Enter three integers: 35 8 79

Maximum is: 79

Προδιαγραφές μετατροπής δεκαδικών

Data type	printf conversion specification	scanf conversion specification
Floating-point types		
long double	%Lf	%Lf
double	%f	%lf
float	%f	%f

Προδιαγραφές μετατροπής ακεραίων

Data type	printf conversion specification	scanf conversion specification
Integer types		
unsigned long long int	%llu	%llu
long long int	%lld	%lld
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
unsigned short	%hu	%hu
short	%hd	%hd
char	%c	%c

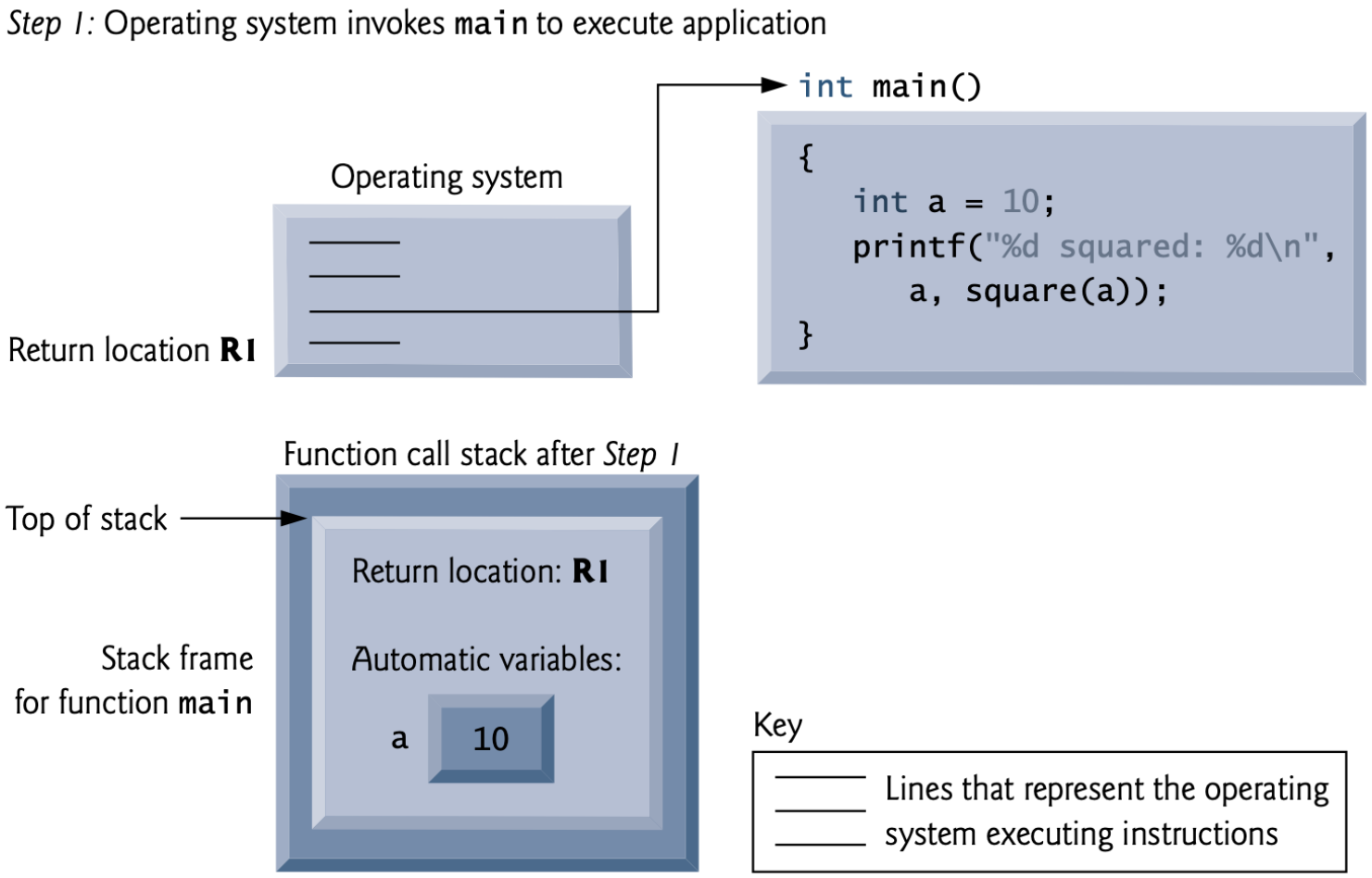
Επίδειξη χρήσης στοίβας

```
1 #include <stdio.h>
2 int square(int); // function prototype
3 int main()
4 {
5      int a = 10; // τιμή για τη συνάρτηση square
6     // τοπική αυτόματη μεταβλητή της main
7     printf("%d squared: %d\n", a, square(a));
8     //  εκτύπωση τετραγώνου
9     puts("");
10 }
11 // επιστρέφει το τετράγωνο ακεραίου
12 int square(int y) // y is a copy of the argument to the function
13 {
14     return y * y; // returns the square of y as an int
15 }
```

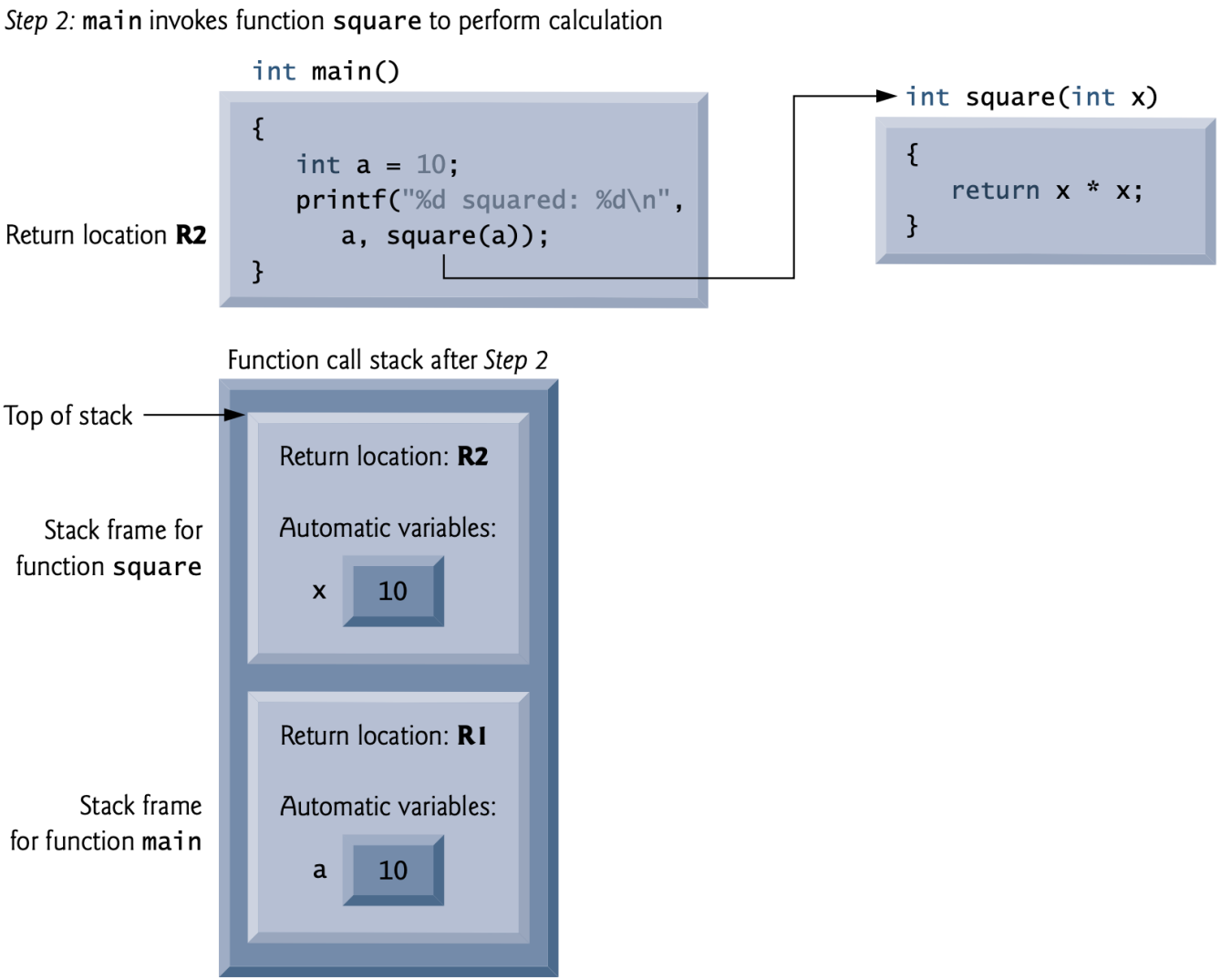
Αποτέλεσμα

10 squared 100

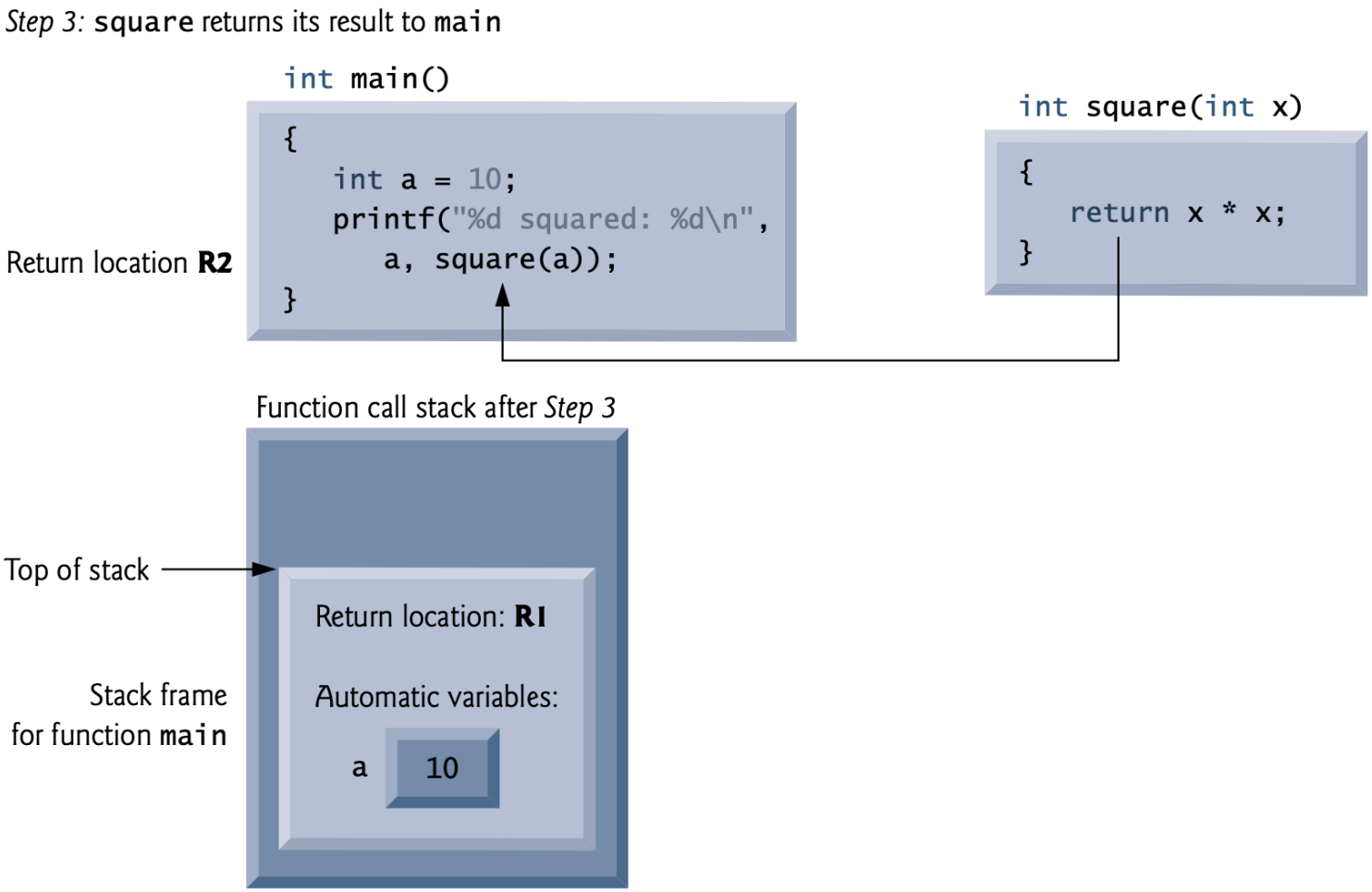
Στοίβα κλήσης συνάρτησης 1



Στοίβα κλήσης συνάρτησης 2



Στοίβα κλήσης συνάρτησης 3



Το πεδίο εμβέλειας των μεταβλητών

- Μία τοπική μεταβλητή είναι δυναμική, δημιουργείται όταν εκτελείται η συνάρτηση και καταστρέφεται όταν τελειώνει η συνάρτηση. Δηλαδή, μία τοπική μεταβλητή είναι γνωστή μόνο στη συνάρτηση στην οποία δηλώνεται.
- Μία γενική μεταβλητή δηλώνεται έξω από οποιαδήποτε συνάρτηση και είναι γνωστή σε όλες τις συναρτήσεις του προγράμματος. Οι γενικές μεταβλητές παραμένουν καθ' όλη τη διάρκεια του προγράμματος.
- Μία στατική μεταβλητή κρατάει την τιμή της μέσα σε μια συνάρτηση από κλήση σε κλήση. Είναι γνωστή μόνο στη συνάρτησή της ή στο αρχείο της και παραμένει κατά τη διάρκεια όλου του προγράμματος.

δήλωση μιας μεταβλητής ως static

```
int number( ) {
    static int new_num ;
    new_num = new_num +25;
    return(new_num);
 }
```

```c
// Fig. 5.16: fig05_16.c
// Scoping.
#include <stdio.h>

void useLocal(void); // function prototype
void useStaticLocal(void); // function prototype
void useGlobal(void); // function prototype

int x = 1; // global variable

int main(void)
{
   int x = 5; // local variable to main

   printf("local x in outer scope of main is %d\n", x);

   { // start new scope
      int x = 7; // local variable to new scope

      printf("local x in inner scope of main is %d\n", x);
   } // end new scope

   printf("local x in outer scope of main is %d\n", x);

   useLocal(); // useLocal has automatic local x
   useStaticLocal(); // useStaticLocal has static local x
   useGlobal(); // useGlobal uses global x
   useLocal(); // useLocal reinitializes automatic local x
   useStaticLocal(); // static local x retains its prior value
   useGlobal(); // global x also retains its value

   printf("\nlocal x in main is %d\n", x);
}

// useLocal reinitializes local variable x during each call
void useLocal(void)
{
   int x = 25; // initialized each time useLocal is called

   printf("\nlocal x in useLocal is %d after entering useLocal\n", x);
   ++x;
   printf("local x in useLocal is %d before exiting useLocal\n", x);
}
```

```c
// Fig. 5.16: fig05_16.c
// Scoping.
#include <stdio.h>

void useLocal(void); // function prototype
void useStaticLocal(void); // function prototype
void useGlobal(void); // function prototype

int x = 1; // global variable

int main(void)
{
   int x = 5; // local variable to main

   printf("local x in outer scope of main is %d\n", x);

   { // start new scope
      int x = 7; // local variable to new scope

      printf("local x in inner scope of main is %d\n", x);
   } // end new scope

   printf("local x in outer scope of main is %d\n", x);

   useLocal(); // useLocal has automatic local x
   useStaticLocal(); // useStaticLocal has static local x
   useGlobal(); // useGlobal uses global x
   useLocal(); // useLocal reinitializes automatic local x
   useStaticLocal(); // static local x retains its prior value
   useGlobal(); // global x also retains its value

   printf("\nlocal x in main is %d\n", x);
}

// useLocal reinitializes local variable x during each call
void useLocal(void)
{
   int x = 25; // initialized each time useLocal is called

   printf("\nlocal x in useLocal is %d after entering useLocal\n", x);
   ++x;
   printf("local x in useLocal is %d before exiting useLocal\n", x);
}
```

Αποτελέσματα

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5

local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal

local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal

global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal

local x in useLocal is 25 after entering useLocal
local x in useLocal is 26 before exiting useLocal

local static x is 51 on entering useStaticLocal
local static x is 52 on exiting useStaticLocal

global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal

local x in main is 5
```

Κλήση με Τιμή και Κλήση με Αναφορά

- Κλήση με τιμή. Αυτή η μέθοδος αντιγράφει την τιμή κάθε μιας από τις παραμέτρους στις τυπικές παραμέτρους της συνάρτησης. Με αυτή τη μέθοδο οι αλλαγές στις τιμές των παραμέτρων της συνάρτησης δεν έχουν κανένα αποτέλεσμα στις μεταβλητές οι οποίες χρησιμοποιούνται για την κλήση της συνάρτησης.
- Κλήση με αναφορά είναι ο δεύτερος τρόπος, με τον οποίο περνούν οι παράμετροι σε μία συνάρτηση. Με αυτή τη μέθοδο, η διεύθυνση της κάθε μίας παραμέτρου αντιγράφεται στις παραμέτρους της συνάρτησης. Αυτό σημαίνει ότι οι αλλαγές οι οποίες γίνονται στις παραμέτρους θα επηρεάσουν τις μεταβλητές οι οποίες χρησιμοποιούνται για την κλήση της συνάρτησης.

Παραδείγματα

```
sqr(int x)
{
    x = x*x;
    return(x);
 }
```

```
swap (int *x, int *y)
{
    int tmp;
    tmp = *x;
/* Αντικαθίσταται η τιμή της διεύθυνσης της μεταβλητής x */
    *x=*y;
    *y=tmp;
}
```

Ο σωστός τρόπος κλίσης της συνάρτησης swap( )

```
main( )
{
 int x, y;
 x=10;
 y=20;
 swap(&x, &y);
 printf("%d  %d", x, y);
 }
```

Κλήση συναρτήσεων με πίνακες

```
#include <stdio.h>
display(int num);

main ()
{
    int t[10], i;
    for(i=0; i<10; ++i)
        scanf("%d", t[i]);
    display(t);
 }

 display(int num)
 {
    int i;
    for(i=0; i<l0; ++i)
    printf("%d ",  num[i]);
 }
```

```
#include <stdio.h>
display(int num);

main( )
{
   int t [10], i;
   for(i=0; i<10; ++i)
        scanf("%d", t[i]);
   for(i=0; i<10; ++i)
        display(t[i]);
}

 display( int num)
 {
      printf("%d ",num)
  }
```

ΑΣΚΗΣΗ

Να γραφτεί ένα πρόγραμμα το οποίο θα διαβάζει μία σειρά χαρακτήρων από το πληκτρολόγιο και θα την εμφανίζει ανάποδα:

π.χ. το hello να γράφεται olleh

Λύση

```
#include <stdio.h>
int main( )
{
 int t;
 char s[80];
 gets(s);
 for ( t=strlen(s); t; t - - )
 putchar(s[t-1])
}
```

Επιδόσεις των συναρτήσεων

- Η συγγραφή και η χρήση συναρτήσεων κατά την ανάπτυξη ενός προγράμματος βελτιώνει πολύ την αναγνωσιμότητα και την αποτελεσματικότητα του προγράμματος και βοηθά σημαντικά στην πρόληψη σφαλμάτων τα οποία οφείλονται σε δευτερεύουσες επιδράσεις καθώς και στον τμηματικό έλεγχο της λειτουργίας του συνολικού τελικού προγράμματος.

Επιδόσεις των συναρτήσεων

- Υπάρχουν δύο σοβαροί λόγοι για τους οποίους ένα πρόγραμμα το οποίο δεν κάνει χρήση συναρτήσεων είναι ταχύτερο από την κλήση μιας ή περισσοτέρων συνάρτησης.
- Ο πρώτος λόγος είναι ότι μία εντολή κλήσης μιας συνάρτησης χρειάζεται κάποιο (ελάχιστο) χρόνο για να εκτελεστεί και
- Ο δεύτερος λόγος είναι ότι οι παράμετροι της συνάρτησης πρέπει να τοποθετηθούν στο σωρό (stack) της μνήμης, γεγονός το οποίο απαιτεί επίσης πρόσθετο χρόνο.

Βιβλιοθήκες (Header Files) της γλώσσας C

Όνομα	Περιγραφή	Δήλωση
<stdio.h>	**Βιβλιοθήκη συναρτήσεων Εισόδου/Εξόδου**	
printf	Γράφει μορφοποιημένα δεδομένα στη stdout	`int printf(const char *format, ...);`
<stdlib.h>	**Βιβλιοθήκη βασικών συναρτήσεων (γενικής χρήσης)**	
abs	Απόλυτη τιμή	`int abs(int x);`
<string.h>	**Βιβλιοθήκη επεξεργασίας χαρακτήρων**	
strlen	Προσδιορίζει το πλήθος των χαρακτήρων της μεταβλητής *str*	`size_t strlen(const char *str);`

Βιβλιοθήκες (Header Files) της γλώσσας C

Όνομα	Περιγραφή	
<math.h>	**Βιβλιοθήκη Μαθηματικών συναρτήσεων**	
<time.h>	**Βιβλιοθήκη συναρτήσεων Ημερομηνίας και ώρας**	
<ctype.h >	**Βιβλιοθήκη συναρτήσεων ελέγχου και μετατροπής χαρακτήρων**	
<limits.h>	**Βιβλιοθήκη για τον ορισμό των ιδιοτήτων των διαφόρων τύπων των μεταβλητών.** Περισσότερα στη διεύθυνση : **http://en.wikipedia.org/wiki/Limits.h**	
<complex.h>	**Βιβλιοθήκη συναρτήσεων για επεξεργασία μιγαδικών αριθμών** Περισσότερες πληροφορίες: ***http://www.opengroup.org/onlinepubs/009695399/basedefs/complex.h.html***	

Δημιουργία τυχαίων αριθμός

- Ο όρος **τυχαίος** αριθμός, στην πληροφορική, δεν σημαίνει ότι οι παραγόμενοι αριθμοί είναι απόλυτα τυχαίοι αλλά εξαρτώνται από τον αλγόριθμο δημιουργίας τους γι αυτό και λέγονται ψευδο-τυχαίοι (pseudo-random)
- Ένας τυχαίος αριθμός παράγεται από τη συνάρτηση rand( ), η οποία βρίσκεται στο επικεφαλής αρχείο stdlib.h
- Για τη δημιουργία διαφορετικού αρχικού σημείου εκκίνησης για την παραγωγή τυχαίων αριθμών πρέπει να κληθεί η συνάρτηση srand( )

Παραγωγή και εμφάνιση τυχαίων αριθμών με εύρος 1-100

```
void main(){
    int k, t;
    printf("Πρόγραμμα το οποίο παράγει 10 τυχαίους \n");
    for(t=0; t<10; ++t) {
        k = tux_arith(t) ;  /* δημιουργούμε ένα τυχαίο αριθμό *
        printf("%d", "Αριθμός = ", k\n");
    }
 }
```

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

/*Συνάρτηση για την παραγωγή τυχαίων αριθμών*/
 int tux_arith(int m) {
       int x,y,zz;
       time_t z ;
       (void ) time(&z);
       (void) srand(m*z) ;  // δημιουργία διαφορετικού αρχικού σημείου
        x=rand() ;    /* δημιουργία τυχαίου αριθμού */
        y=(x%100)+1 ;  // μετατροπή του σε εύρος 1-100
        return(y) ;
  }
```

Ασκήσεις

- **Να γράψετε ένα πρόγραμμα προσομοίωσης μιας ζαριάς δηλαδή, τυχαία εμφάνιση δύο αριθμών με τιμές 1 μέχρι 6.**

Ειδική εκτέλεση προγραμμάτων

- Πολλές φορές απαιτείται να δώσουμε αρχικές τιμές ή ειδικές τιμές σε ένα πρόγραμμα άμεσα δηλαδή, με την έναρξη της εκτέλεσης των πράξεων από τον υπολογιστή
- Ο ειδικός τρόπος με τον οποίο αντιμετωπίζει η γλώσσα C αυτή την περίπτωση δηλαδή, την άμεση εισαγωγή πληροφοριών στη συνάρτηση main( ) είναι η χρήση δύο ειδικών ενσωματωμένων παραμέτρων
- Αυτές είναι η argc και η argv οι οποίες είναι οι μοναδικές παράμετροι τις οποίες μπορεί να έχει η συνάρτηση main( )

Ειδική εκτέλεση προγραμμάτων

- Για να μεταφερθούμε στη μορφή εντολών του λειτουργικού συστήματος των Windows πρέπει να εκτελέσουμε την εντολή command.com ή την εντολή cmd.
- Για να επανέλθουμε στη γραφική μορφή του περιβάλλοντος των Windows πρέπει να γράψουμε την εντολή exit

Οι παράμετροι argc και argv

Εμφάνιση χαιρετισμού προς το χρήστη

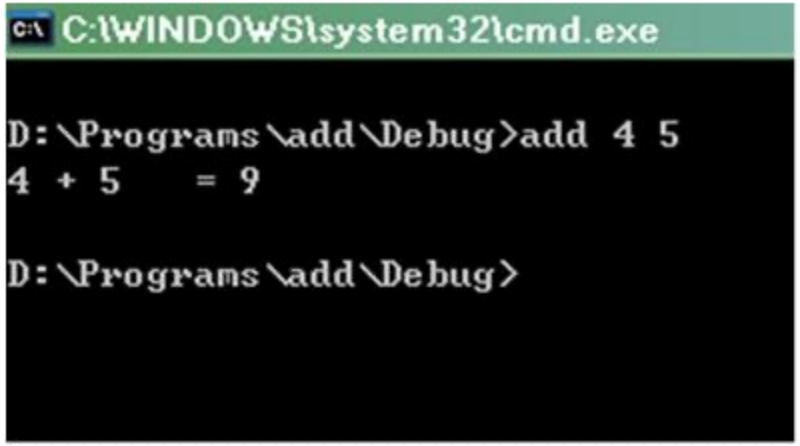
```
main(argc, argv)
        int argc;
    char *argv [ ];
{
     if( argc != 2)
    {
     printf("Ξεχάσατε να γράψετε το όνομα\n");
      exit(0);
    }
     printf("Hello %s", argv[ 1 ]);
 }
```

Ένα πρόγραμμα που εμφανίζει τις παραμέτρους που χρησιμοποιούνται στην κλήση του

```
main (argc, argv)
  int argc;
  char *argv [ ];
     {   int t;
         for(t=0; t<argc; ++t)
             {  i=0;
                while(argv [t] [i] )
                 { putchar(argv [t] [i] );
                   ++i;   }
             }
        }
```

Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
      int a = atoi(argv[1]);
      int b = atoi(argv[2]);
      int sum = a + b;
      printf("%s + %s  = %d\n",argv[1],argv[2],sum);
}
```



argc = 3

argv[0] = “add”

argv[1] = “4”

argv[2] = “5”

Οργάνωση των συναρτήσεων

Όταν πρόκειται να γράψουμε μία συνάρτηση, υπάρχουν τρεις διαφορετικές περιπτώσεις διαχείρισης του τελικού κειμένου της συνάρτησης:

- να αφήσουμε τη συνάρτηση στο ίδιο αρχείο μαζί με τη συνάρτηση main( )
- να τοποθετήσουμε τη συνάρτηση σε ένα άλλο ξεχωριστό αρχείο, μαζί με τις υπόλοιπες συναρτήσεις τις οποίες ήδη έχουμε γράψει
- να τοποθετήσουμε τη συνάρτηση σε μία βιβλιοθήκη συναρτήσεων για μελλοντική χρήση

Βιβλιοθήκες συναρτήσεων

- Στον προγραμματισμό, λέμε *βιβλιοθήκη* (library) μία συλλογή χρήσιμων συναρτήσεων και βοηθητικών δηλώσεων
- Μία βιβλιοθήκη χρήσιμων συναρτήσεων διαφέρει από ένα ξεχωριστά μεταγλωττισμένο αρχείο συναρτήσεων
- Όταν κάνουμε χρήση μίας βιβλιοθήκης, μόνο οι συναρτήσεις οι οποίες καλούνται για χρήση από το πρόγραμμά μας φορτώνονται στη μνήμη και συνδέονται με αυτό. Οι υπόλοιπες συναρτήσεις παραμένουν στη βιβλιοθήκη και δεν επιβαρύνουν τη μνήμη του υπολογιστή

Πρότυπη βιβλιοθήκη (standard library)

- **Η πρότυπη βιβλιοθήκη είναι μια συλλογή από αρχεία επικεφαλίδων και συναρτήσεις βιβλιοθήκης οι οποίες υλοποιούν τις βασικές λειτουργίες εισόδου/εξόδου, χειρίζονται αλφαριθμητικά δεδομένα, υπολογίζουν απλούς μαθηματικούς τύπους και πολλές άλλες κλασικές λειτουργίες οι οποίες διευκολύνουν τους προγραμματιστές**
- **Τα βασικά χαρακτηριστικά κάθε συνάρτησης όπως είναι το όνομα της συνάρτησης, ο τύπος των παραμέτρων και ο τύπος της τιμής επιστροφής, περιλαμβάνονται σε ένα αρχείο το οποίο καλείται *επικεφαλής αρχείο* (header file).**

Επικεφαλής αρχείο

Λέμε *επικεφαλής αρχείο* (header file) ένα αρχείο κειμένου το οποίο περιέχει ένα πρόγραμμα (κώδικα) και το οποίο μπορεί να χρησιμοποιηθεί με ειδικό τρόπο

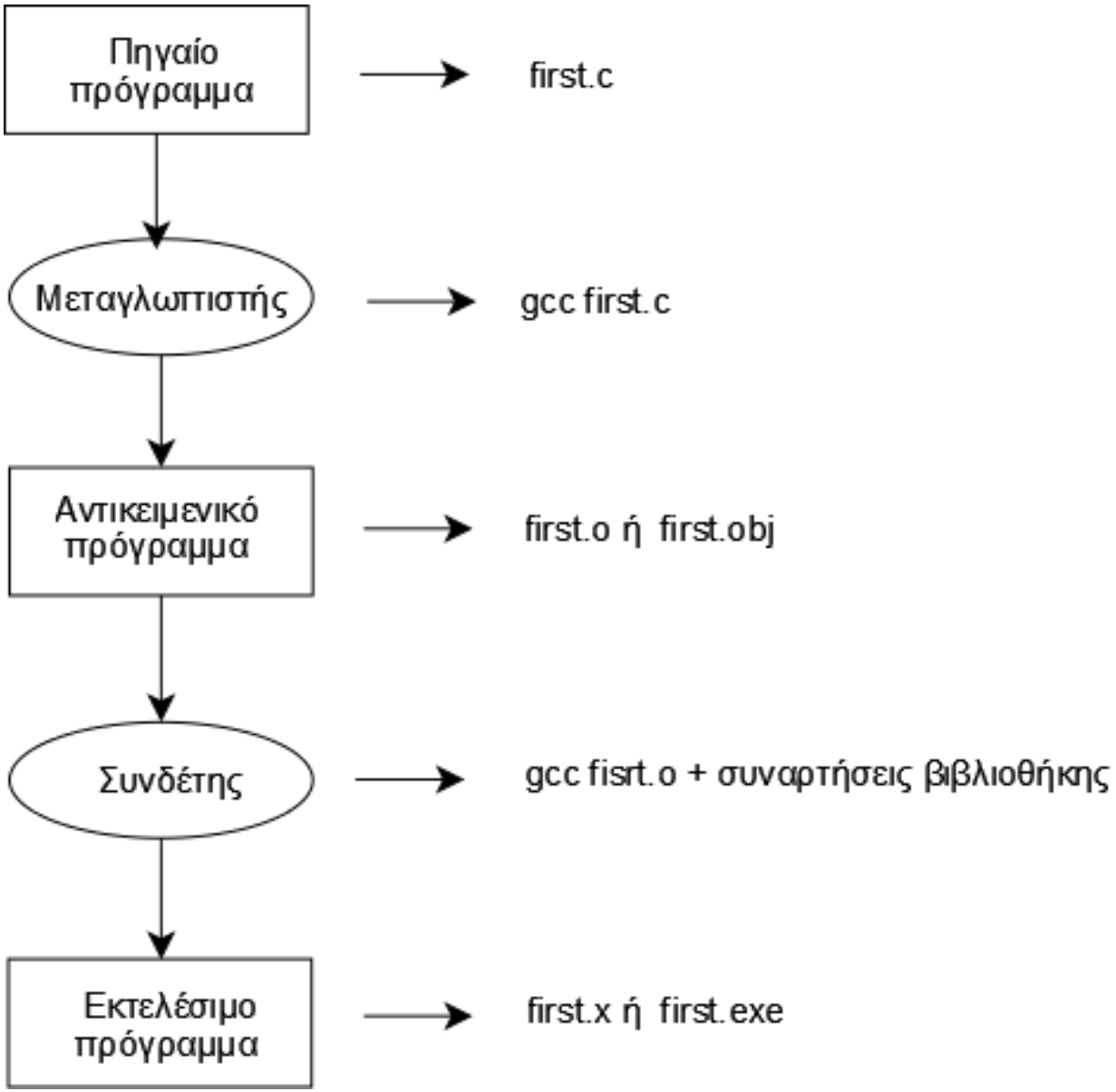
Στη γλώσσα C, το όνομα ενός επικεφαλής αρχείου εξ ορισμού τελειώνει με την επέκταση .h και χρησιμοποιείται με την επίκληση της οδηγίας (directive) # include

Π.χ.

#include <όνομα επικεφαλής αρχείου Βιβλιοθήκης>

ή

#include "όνομα επικεφαλής αρχείου ορισμένου από το χρήστη"


Πηγαίο πρόγραμμα
first.c
Μεταγλωττιστής
gcc first.c
Αντικειμενικό πρόγραμμα
first.o ή first.obj
Συνδέτης
gcc fisrt.o + συναρτήσεις βιβλιοθήκης
Εκτελέσιμο πρόγραμμα
first.x ή first.exe

Τα επικεφαλής αρχεία τα ορισμένα από το χρήστη είναι προγράμματα τα οποία γράφονται προκειμένου να χωριστούν οι δηλώσεις των συναρτήσεων καθώς και οι δηλώσεις των γενικών, στατικών και εξωτερικών μεταβλητών από τον υπόλοιπο πηγαίο κώδικα.

Αυτός ο διαχωρισμός παρέχει τρία σημαντικά πλεονεκτήματα:

- 1. Οργανώνει τον πηγαίο κώδικα του προγράμματος σε καλά καθορισμένες και οργανωμένες οντότητες
- 2. Χωρίζει τις διεπαφές του προγράμματος (interfaces) από την κύρια εφαρμογή του αλγορίθμου επίλυσης του προβλήματος
- 3. Παρέχει ευελιξία και ευκολότερη επαναχρησιμοποίηση στοιχείων του προγράμματος από τους προγραμματιστές

Παράδειγμα

Έστω ότι θέλουμε να γράψουμε ένα πρόγραμμα για να μετατρέπουμε τα μίλια σε χιλιόμετρα. Μπορούμε λοιπόν, να γράψουμε το ακόλουθο πρόγραμμα :

ProgramA.c:

```
const double FACTOR = 1.609344;
double cvtMileToKilom( double miles );
   int main()
{  double miles = 0;
  printf("Πληκτρολογήστε τη μέτρηση σε μίλια:");
  scanf("%lf", &miles);
  printf("%f μίλια είναι %f χιλιόμετρα.\n", miles, cvtMileToKilom( miles ) );
  return 0;    }
        double cvtMileToKilom( double miles )
          {  return FACTOR * miles;  }
```

Αν θελήσουμε να γράψουμε ένα νέο πρόγραμμα το οποίο θα μετατρέπει τα μίλια σε μέτρα μπορούμε να επαναχρησιμοποιήσουμε αυτό το πρόγραμμα για να δημιουργήσουμε γρήγορα και εύκολα το νέο πρόγραμμα προσθέτοντας μια νέα συνάρτηση.

ProgramB.c

```
const double FACTOR = 1.609344;
 double cvtMileToKilom( double miles );
double cvtMileToMeter( double miles );
 int main(void)
{   double miles = 0;
  printf("Πληκτρολογήστε τη μέτρηση σε μίλια:");
  scanf("%lf", &miles);
  printf("%f μίλια είναι %f μέτρα.\n", miles, cvtMileToMeter(miles) );
  return 0;
}
```

```
double cvtMileToKilom( double miles )
{
  return FACTOR * miles;
}

double cvtMileToMeter( double miles )
{
  return cvtMileToKilom( miles ) * 1000;
}
```

Υπάρχει όμως μια άλλη, ίσως καλύτερη, επιλογή για την επίλυση του ιδίου προβλήματος.

Με την επιλογή αυτή χωρίζουμε τη δήλωση της συνάρτησης cvtMileToKilom από την εφαρμογή δηλαδή, το υπόλοιπο πρόγραμμα. Έτσι, μπορούμε να αποθηκεύσουμε τη δήλωση σ' ένα επικεφαλής αρχείο επέκταση .h και το υπόλοιπο πρόγραμμα σε ένα αρχείο με επέκταση .c. Π,χ,

programC.h

```
const static double FACTOR = 1.609344;
 double cvtMileToKilom( double miles );
```

programC.c

```
#include "programA.h"
double cvtMileToKilom( double miles )
{   return FACTOR * miles; }
```

Αυτό που πρέπει να γίνει στο νέο πρόγραμμα είναι να αναφέρουμε το επικεφαλής αρχείο.

programD.c

```
#include "programC.h"
#include <stdio.h>
double cvtMileToMeter( double miles );
        int main()
{
  double miles = 0;
  printf("Πληκτρολογήστε τη μέτρηση σε μίλια:");
  scanf("%lf", &miles);
  printf("%f μίλια είναι %f μέτρα.\n", miles,
cvtMileToMeter(miles) );
  return 0;
}
      double cvtMileToMeter( double miles )
{
  return cvtMileToKilom( miles ) * 1000;
}
```

Δημιουργία του εκτελέσιμου αρχείου,

Ανάλογα με το λειτουργικό σύστημα, ακολουθούμε τα επόμενα:

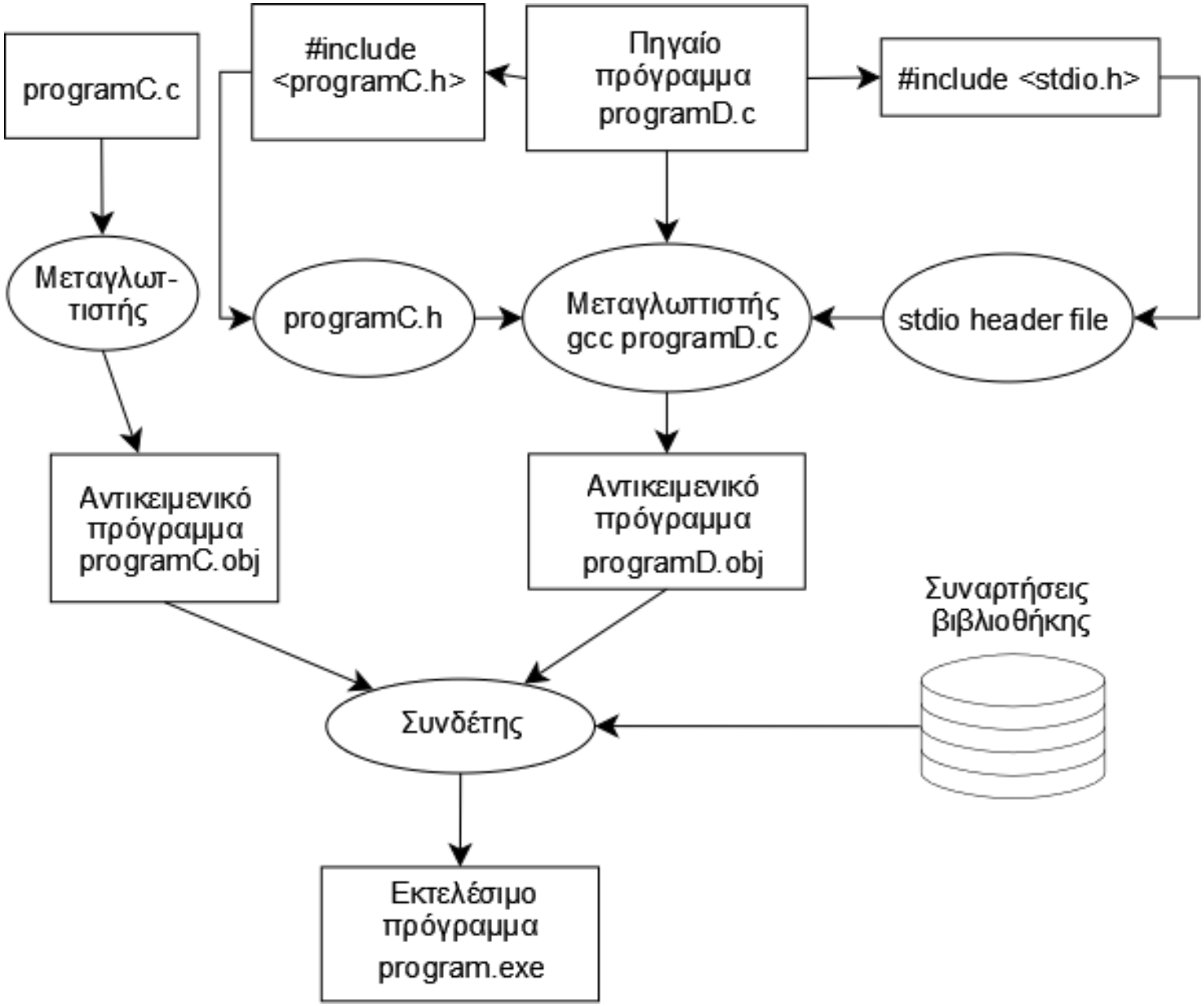
1. Σε περιβάλλον Unix/Linux γράφουμε :

gcc programD.c programC.c -o program

και για να τρέξουμε το πρόγραμμα γράφουμε :

./ program

2. Σε περιβάλλον Windows δημιουργούμε ένα σχέδιο (project) όπου και συμπεριλαμβάνουμε τα δύο αρχεία (programD.c και programC.c).


programC.c
#include <programC.h>
Πηγαίο πρόγραμμα programD.c
#include <stdio.h>
Μεταγλωτ-τιστής
programC.h
Μεταγλωττιστής gcc programD.c
stdio header file
Αντικειμενικό πρόγραμμα programC.obj
Αντικειμενικό πρόγραμμα programD.obj
Συναρτήσεις βιβλιοθήκης
Συνδέτης
Εκτελέσιμο πρόγραμμα program.exe

ΑΣΚΗΣΗ

- Ένας ακέραιος αριθμός λέγεται τέλειος αριθμός όταν οι παράγοντές του συμπεριλαμβανομένης και της μονάδας (αλλά όχι και του ίδιου του αριθμού), δίνουν ως άθροισμα τον ίδιο αριθμό.
- Για παράδειγμα, το 6 είναι ένας τέλειος αριθμός, επειδή 6=1+2+3.
- Να γράψετε ένα πρόγραμμα το οποίο να εμφανίζει στην οθόνη όλους τους τέλειους αριθμούς οι οποίοι είναι μικρότεροι του αριθμού ο οποίος θα ορίζεται από τη γραμμή εντολών με την έναρξη εκτέλεσης του προγράμματος

- Λέμε *μορφή εντολών* (Command mode) εκείνη τη μορφή με την οποία επικοινωνούμε άμεσα με το λειτουργικό σύστημα του υπολογιστή μέσω του πληκτρολογίου.
- Όταν μεταφερθούμε στη *μορφή εντολών* τότε, σε μια γραμμή της οθόνης γράφουμε τη σχετική εντολή προς το λειτουργικό σύστημα δηλαδή, την εντολή της έναρξης εκτέλεσης ενός προγράμματος.
- Συνήθως, γράφουμε τη δήλωση της συνάρτησης main( ):

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

- Στην προκειμένη περίπτωση θα πρέπει να ελέγξουμε αν το πλήθος των παραμέτρων οι οποίες έχουν πληκτρολογηθεί είναι ακριβώς δύο (2) οπότε η δεύτερη παράμετρος θα είναι ο αριθμός για τον οποίο θέλουμε να υπολογίσουμε τους πρώτους αριθμούς οι οποίοι θα είναι μικρότεροι από αυτόν.

- *Γράφουμε λοιπόν:*

```
if( argc!=2)
{
    printf("Ξεχάσατε να γράψετε τον αριθμό ");
    printf("στη γραμμή εντολής. \n Ξαναδοκιμάστε. \n");
     exit(0);
 }
```

```
#include <stdio.h>
#include <stdlib.h>
      int perfect( int value ); /* αρχέτυπο συνάρτησης */
      int main(int argc, char *argv[])
{
  int number, max_number; /* βοηθητικές μεταβλητές */
    if( argc!=2)   //  Έλεγχος αν έχει συμπληρωθεί ο αριθμός
     {
       printf("Ξεχάσατε να γράψετε τον αριθμό ");
       printf("στη γραμμή εντολής. \n Ξαναδοκιμάστε. \n");
        exit(0);
     }
     max_number= atoi(argv[1]);  // εισαγόμενος αριθμός
printf( "Εύρεση των πρώτων αριθμών μέχρι %d\n\n", max_number );
```

```
 /* ανακύκλωση από το 2 έως το max_number */
  for (number = 2; number <= max_number; number++ )
       {
/* εάν ο τρέχων αριθμός είναι πρώτος αριθμός   */
      if ( perfect( number ) )
        {
         printf( "Ο αριθμός %d είναι πρώτος \n", number );
        }
      }
   printf( "\n" );

   system("PAUSE");
   return 0; /* δηλώνει την επιτυχή ολοκλήρωση του
     προγράμματος  */
}
```

```
int perfect( int value )
{
  int factorSum = 1; /* απόδοση αρχικής τιμής στο άθροισμα  */
  int i;             /* μετρητής ανακύκλωσης */
  /* ανακύκλωση πιθανών τιμών παραγόντων */
  for ( i = 2; i <= value / 2; i++ )
  {      /* αν ο i είναι παράγοντας   */
    if ( value % i == 0 )
    {   factorSum += i; /* πρόσθεση στο άθροισμα  */   }
  }
/* επιστέφει true (1) αν η μεταβλητή value ισούται με το
    άθροισμα των παραγόντων  */
  if ( factorSum == value )
     {    return 1;    }
  else
     {     return 0;     }
}
```