Οι συναρτήσεις στη γλώσσα C

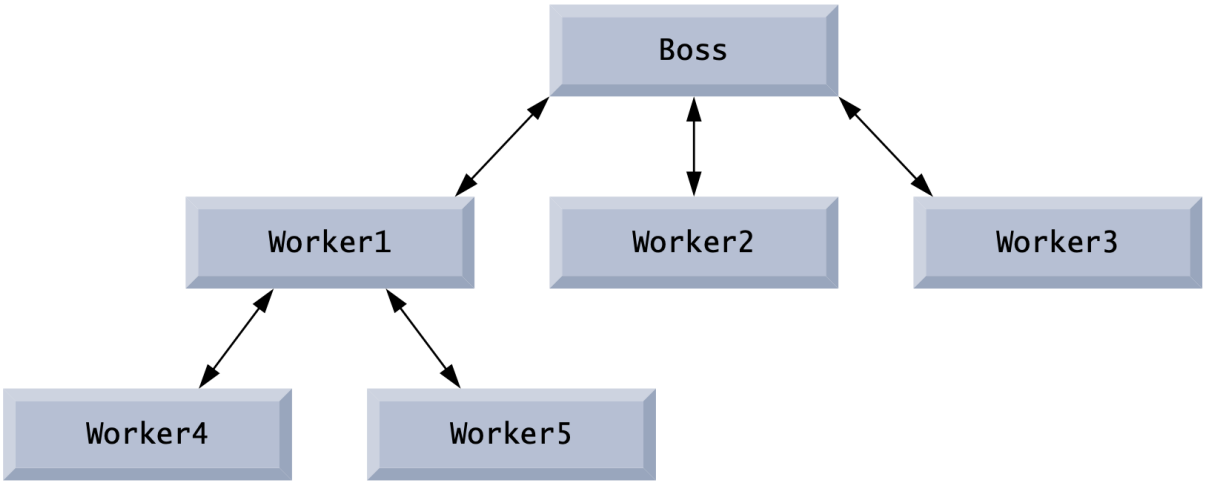
Οι συναρτήσεις αποτελούν τα βασικότερα στοιχεία της γλώσσας προγραμματισμού C.

Με τη βοήθεια των συναρτήσεων αναπτύσσεται όλη η δραστηριότητα των προγραμμάτων

Γενικά για τις συναρτήσεις

- Διαίρει και βασίλευε
- Αρθρωτότητα συστήματος
- Καθοριζόμενες από τον προγραμματιστή
- Προκατασκευασμένες
- C standard library + γλώσσα C -> πρόγραμμα !
  - Μαθηματικοι υπολογισμοί,
  - διαχείριση αλφαριθμητικών,
  - διαχείριση χαρακτήρων,
  - διαχείριση εισοδου εξόδου

Ιεραρχική σχέση συναρτήσεων



- Οι συναρτήσεις καλούνται από κλήσεις συναρτήσεων (function calls)
- Ορίζει τη συνάρτηση και την πληροφορία (παραμέτρους) που χρειαζεται η συνάρτηση-εργάτης για να εκτελέσει την εργασία (task) που της έχει ανατεθεί από τη συνάρτηση-προιστάμενος
- Αναλογία με το management.

Κύριες μαθηματικές συναρτήσεις

με τη βιβλιοθήκη math.h

Function	Description	Example
`sqrt(x)`	square root of x	`sqrt(900.0)` is 30.0 `sqrt(9.0)` is 3.0
`cbrt(x)`	cube root of x (C99 and C11 only)	`cbrt(27.0)` is 3.0 `cbrt(-8.0)` is -2.0
`exp(x)`	exponential function e^x	`exp(1.0)` is 2.718282 `exp(2.0)` is 7.389056
`log(x)`	natural logarithm of x (base e)	`log(2.718282)` is 1.0 `log(7.389056)` is 2.0
`log10(x)`	logarithm of x (base 10)	`log10(1.0)` is 0.0 `log10(10.0)` is 1.0 `log10(100.0)` is 2.0
`fabs(x)`	absolute value of x as a floating-point number	`fabs(13.5)` is 13.5 `fabs(0.0)` is 0.0 `fabs(-13.5)` is 13.5
`ceil(x)`	rounds x to the smallest integer not less than x	`ceil(9.2)` is 10.0 `ceil(-9.8)` is -9.0
`floor(x)`	rounds x to the largest integer not greater than x	`floor(9.2)` is 9.0 `floor(-9.8)` is -10.0
`pow(x, y)`	x raised to power y (x^y)	`pow(2, 7)` is 128.0 `pow(9, .5)` is 3.0

Παράδειγμα

- Οι παράμετροι μιας συνάρτησης μπορεί να είναι σταθερές μεταβλητές ή παραστάσεις
- Εάν c1 = 13.0, d = 3.0 και f = 4.0, τότε η εντολή:
- **printf ("%.2f", sqrt (c1 + d * f));**
- υπολογίζει και τυπώνει την τετραγωνική ρίζα του 13.0 + 3.0 * 4.0 = 25.0, δηλαδή 5.00.

Τιμές επιστροφής

- Υπαρχουν 3 περιπτώσεις επιστροφής του ελέγχου σπό τη συνάρτηση
- return;
- τίποτα
- return παράσταση

Συναρτήσεις Επιστροφής Μη-Ακεραίων Τιμών

Ο γενικός τύπος δήλωσης μιας συνάρτησης είναι:

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

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

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 ;
      printf(“The sum of x and y is .3f%.\n”, z);
   }
```

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 max = x; // assume x is largest
4      if (y > max){ // if y is larger than max
5         max = y; //assign y to max
6      }
7      if (z > max){ // if z is larger than max;
8         max = z; //assign y to max
9     }
10     return(max); // max is largest value
11
12 }
```

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

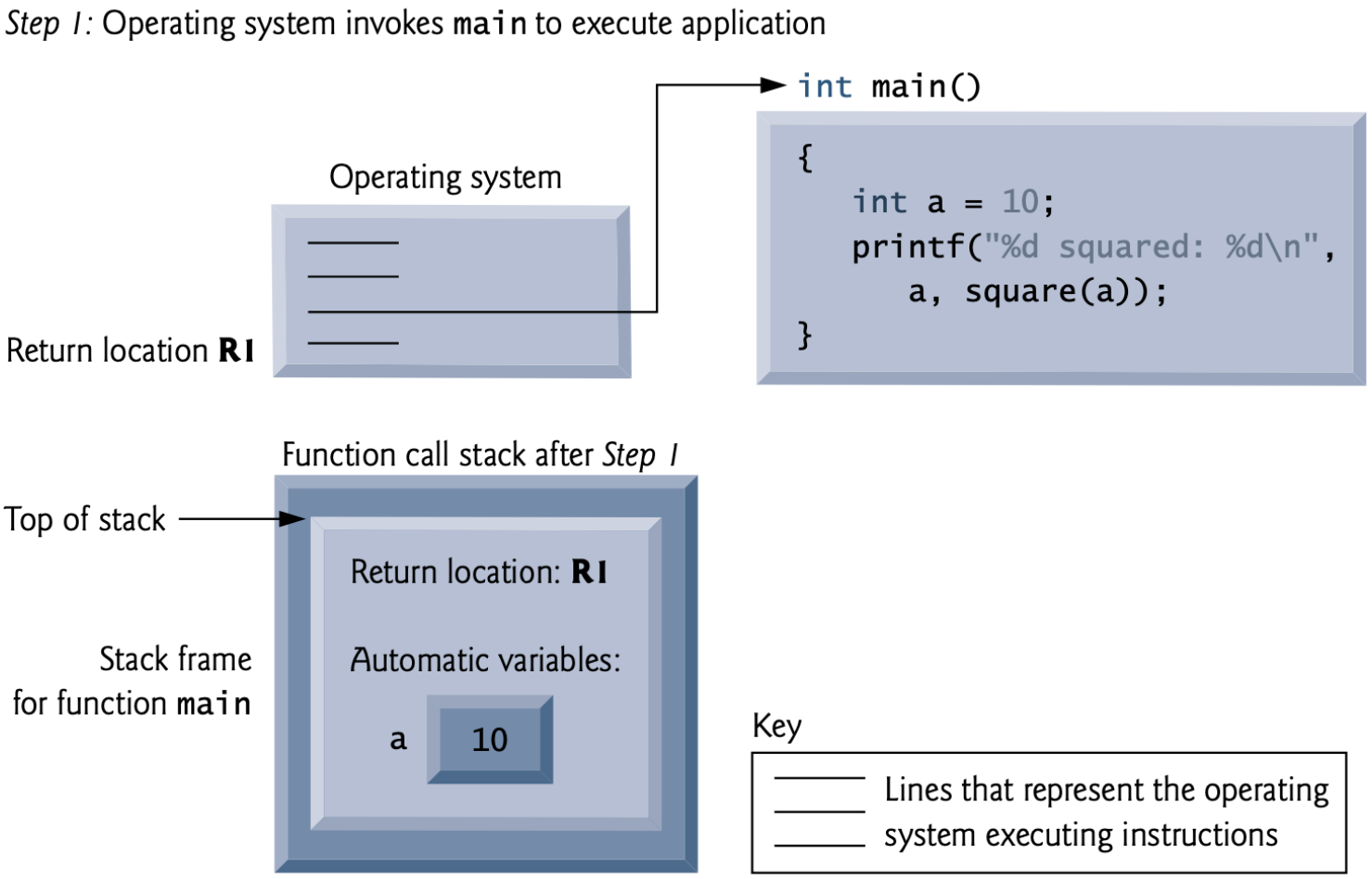
Επίδειξη χρήσης στοίβας

```
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 }
```

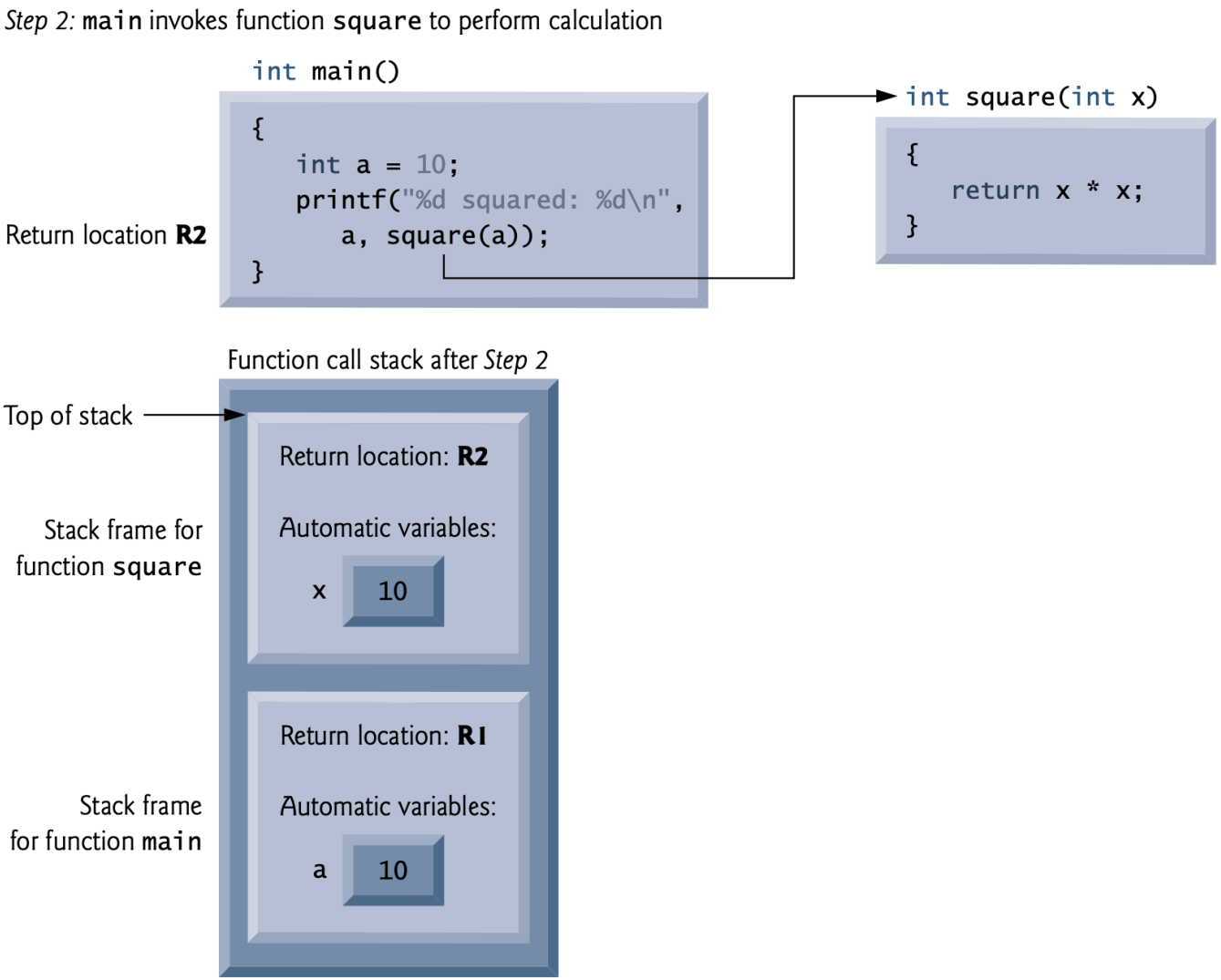
Αποτέλεσμα

10 squared 100

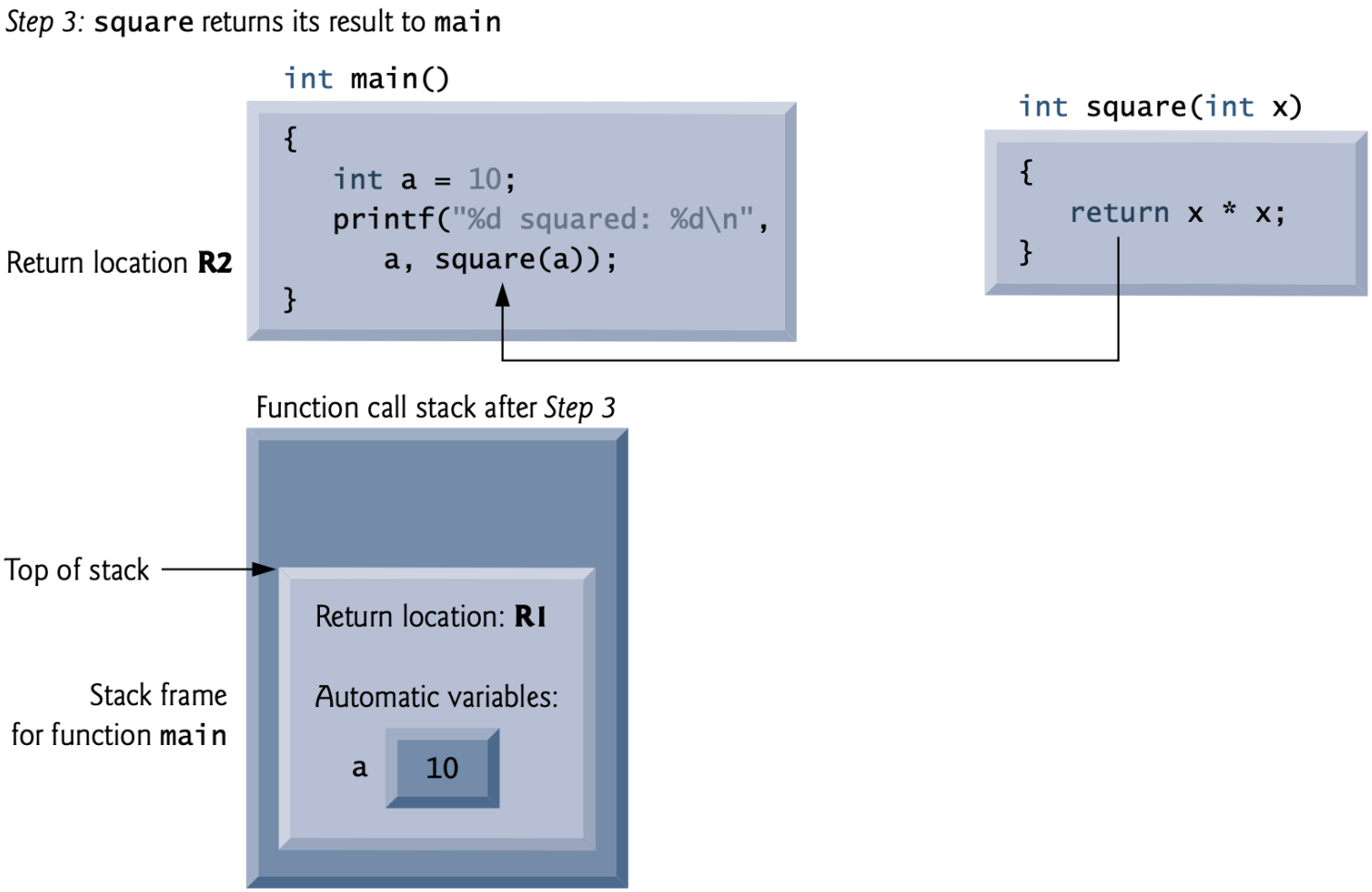
Στοίβα κλήσης συνάρτησης 1



Στοίβα κλήσης συνάρτησης 2



Στοίβα κλήσης συνάρτησης 3



Βιβλιοθήκες (Header Files) της γλώσσας C

Header	Explanation
`<assert.h>`	Contains information for adding diagnostics that aid program debugging.
`<ctype.h>`	Contains function prototypes for functions that test characters for certain properties, and function prototypes for functions that can be used to convert lowercase letters to uppercase letters and vice versa.
`<errno.h>`	Defines macros that are useful for reporting error conditions.
`<float.h>`	Contains the floating-point size limits of the system.
`<limits.h>`	Contains the integral size limits of the system.
`<locale.h>`	Contains function prototypes and other information that enables a program to be modified for the current locale on which it's running. The notion of locale enables the computer system to handle different conventions for expressing data such as dates, times, currency amounts and large numbers throughout the world.

Βιβλιοθήκες (Header Files) της γλώσσας C

Header	Explanation
`<math.h>`	Contains function prototypes for math library functions.
`<setjmp.h>`	Contains function prototypes for functions that allow bypassing of the usual function call and return sequence.
`<signal.h>`	Contains function prototypes and macros to handle various conditions that may arise during program execution.
`<stdarg.h>`	Defines macros for dealing with a list of arguments to a function whose number and types are unknown.
`<stddef.h>`	Contains common type definitions used by C for performing calculations.
`<stdio.h>`	Contains function prototypes for the standard input/output library functions, and information used by them.
`<stdlib.h>`	Contains function prototypes for conversions of numbers to text and text to numbers, memory allocation, random numbers and other utility functions.
`<string.h>`	Contains function prototypes for string-processing functions.
`<time.h>`	Contains function prototypes and types for manipulating the time and date.

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)**	
<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***	

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);
 }
```

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( )

Ρίξιμο ζαριού

```
1  // Fig. 5.11: fig05_11.c
2  // Shifted, scaled random integers produced by 1 + rand() % 6.
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main(void)
7  {
8     // loop 20 times
9     for (unsigned int i = 1; i <= 20; ++i) {
10
11       // pick random number from 1 to 6 and output it
12       printf("%10d", 1 + (rand() % 6));
13
14       // if counter is divisible by 5, begin new line of output
15       if (i % 5 == 0) {
16          puts("");
17       }
18    }
19 }
```

Αποτέλεσμα

6	6	5	5	6
5	1	1	5	3
6	6	2	4	2
6	2	3	4	1

Επαναληψιμότητα συνάρτησης!

Randomising

```
1  // Fig. 5.13: fig05_13.c
2  // Randomizing the die-rolling program.
3  #include <stdlib.h>
4  #include <stdio.h>
5
6  int main(void)
7  {
8     unsigned int seed; // number used to seed the random number generator
9
10    printf("%s", "Enter seed: ");
11    scanf("%u", &seed); // note %u for unsigned int
12
13    srand(seed); // seed the random number generator
14
15    // loop 10 times
16    for (unsigned int i = 1; i <= 10; ++i) {
17
18       // pick a random number from 1 to 6 and output it
19       printf("%10d", 1 + (rand() % 6));
20
21       // if counter is divisible by 5, begin a new line of output
22       if (i % 5 == 0) {
23          puts("");
24       }
25    }
26 }
```

Αποτέλεσμα

```
Enter seed: 67
        6       1       4       6       2
        1       6       1       6       4
```

```
Enter seed: 867
        2       4       6       1       6
        1       1       3       6       2
```

```
Enter seed: 67
        6       1       4       6       2
        1       6       1       6       4
```

```
#include <time.h> ;
srand( time (NULL ) ) ;
```

- Αναγκάζει τον υπολογιστή να διαβάσει το ρολόι του και να πάρει την τιμή για τη συναρτηση seed αυτόματα
- Επιστρέφει τον αριθμό το δευτερολέπτων που έχουν περάσει από τα μεσάνυχτα της 01/01/1970.
- Η τιμή αυτή μετατρέπεται σε μη προσημασμένο ακέραιο και χρησιμοποιείται για την διασπορά της γεννήτριας τυχαίων αριθμών

Κλίμακωση και μετατόπιση

```
0 ≤ rand() ≤ RAND_MAX
```

```
n = a + rand() % b;
```

a: shifting value / μετατόπιση : αρχικός αριθμός

b: scaling factor / συντελεστής κλιμάκωσης : εύρος αριθμών

Αναγνωριστικά / Identifiers

- Είναι τα ονόματα των μεταβλητών και των συναρτήσεων
- Είπαμε ήδη ότι τα ονόματα των μεταβλητών έχουν:
  - όνομα,
  - μέγεθος,
  - τύπο,
  - τιμή.
- έχουν επίσης
  - κλάση αποθήκευσης,
  - διάρκεια αποθήκευσης,
  - εμβέλεια
  - διασύνδεση.

κλάση αποθήκευσης

- Η κλάση αποθήκευσης ενός αναγνωριστικού ορίζει
  - τη διάρκεια αποθήκευσης την εμβέλεια και τη διασύνδεση.
- Η διάρκεια αποθήκευσης είναι η περίοδος κατά την οποία το αναγνωριστικό υπάρχει στη μνήμη
  - κάποια αναγνωριστικά υπάρχουν για ένα σύντομο διάστημα κάποια δημιουργούνται και καταστρέφονται επαναλαμβανόμενα και κάποια υπάρχουν για όλη τη διάρκεια του προγράμματος.
- Η εμβέλεια είναι που μπορεί να αναφερθεί ένα αναγνωριστικό κατά τη διάρκεια του προγράμματος
- Η διασύνδεση είναι για προγράμματα που χρησιμοποιούν πολλαπλά αρχεία πηγαίου κώδικα εάν το αναγνωριστικό είναι γνωστό μόνο στο τρέχον αρχείο ή και σε άλλα.

κλάση αποθήκευσης

- Είναι είτε αυτόματη ή στατική
- Αυτόματη αποθήκευση εχουν
  - οι τοπικές μεταβλητές
  - Καμία συνάρτηση
- Στατική αποθήκευση εχουν
  - Τοπικές μεταβλητές που έχουν δηλωθεί ως static και extern
  - Καθολικές μεταβλητές (by default extern)
  - Συναρτήσεις

Στατική αποθήκευση

- Τα αναγνωριστικά που έχουν στατική αποθήκευση υπάρχουν από την αρχή που δημιουργείται το πρόγραμμα μέχρις ότου το πρόγραμμα τερματιστεί.
- Όσον αφορά στις μεταβλητές:
  - Εκχωρείται και αρχικοποιείται αποθηκευτικός χώρος μια φορά στην αρχή προτού το πρόγραμμα αρχίσει να εκτελείται
- Όσον αφορά στις συναρτήσεις
  - δεν εκχωρείται χώρος αλλά το όνομα της συνάρτησης είναι γνωστό μόλις αρχίσει το πρόγραμμα να εκτελείται

Αυτόματη αποθήκευση

- Τα αναγνωριστικά που έχουν αυτόματη αποθήκευση
  - δημιουργούνται όταν ο έλεγχος μπει στο μπλοκ μέσα στο οποίο δηλώνονται
  - υπάρχουν όσο το μπλοκ είναι ενεργό και
  - καταστρέφονται όταν το πρόγραμμα ο έλεγχος φεύγει από το μπλοκ
- Οι τοπικές μεταβλητές έχουν αυτόματη αποθήκευση
  - Πχ υπάρχουν μόνο στο πλαίσιο στοίβας που ανήκει στην προσωρινή μνήμη της διεργασίας

εμβέλεια

- Παρόλα αυτά δεν σημαίνει ότι αυτές οι συναρτήσεις και οι μεταβλητές μπορούν να προσπεραστούν από οποιοδήποτε σημείο του προγράμματος:
  - αυτό έχει σχέση με την εμβέλεια.
- **Καθολικές μεταβλητές** δημιουργούνται βάζοντας τη δήλωσή τους έξω από κάθε συνάρτηση του προγράμματος έτσι ώστε να μπορούν να αναφερθούν από κάθε συνάρτηση που ακολουθεί τη δήλωσή τους
- κρατούν την τιμή τους καθ όλη τη διάρκεια της ζωής του προγράμματος
- για τον ίδιο λόγο χρησιμοποιούμε **πρωτότυπα συναρτήσεων** που τα τοποθετούμε πάνω πάνω στο πρόγραμμα μέσα στις αρχεία επικεφαλίδων

εμβέλεια

- Ονομάζουμε εμβέλεια ή πεδίο εμβέλειας μιας μεταβλητής το κομμάτι του προγράμματος μέσα στο οποίο μπορεί να γίνει αναφορά στη μεταβλητή αυτή.
- Π.χ., εάν έχουμε μια μεταβλητή που είναι τοπική μέσα σε ένα μπλοκ μπορεί να αναφερθεί μόνο, αφού έχει δηλωθεί, μέσα στο ίδιο μπλοκ καθώς και σε άλλα μπλοκ που είναι εμφωλευμένα.
- Υπάρχουν 4 πεδία εμβέλειας:
  - **εμβέλεια συνάρτησης** (function scope),
  - **εμβέλεια αρχείου** (file scope),
  - **εμβέλεια μπλοκ** (block scope)
  - **εμβέλεια πρωτοτύπου συνάρτησης** (function prototype scope).

Εμβέλεια συνάρτησης

- Εμβέλεια συναρτήσhς έχουν μόνο οι ετικέτες
- αναγνωριστικά που ακολουθούνται από :
- μπορούν να χρησιμοποιηθούν οπουδήποτε μέσα στο σώμα μιας συνάρτησης και είναι ορατές μόνο μέσα στη συνάρτηση αυτή.
- ετικέτες χρησιμοποιούνται μόνο στην εντολή switch και στην εντολή goto
- π.χ. start:

Εμβέλεια αρχείου

- Ένα αναγνωριστικό που δηλώνεται έξω από όλες τις συναρτήσεις έχει εμβέλεια αρχείου.
- Ένα τέτοιο αναγνωριστικό είναι γνωστό δηλαδή προσβάσιμο σε όλες τις συναρτήσεις από το σημείο από το οποίο δηλώνεται μέχρι το τέλος του αρχείου.
- Σε αυτή την κατηγορία ανήκουν
  - οι καθολικές μεταβλητές
  - οι δηλώσεις συναρτήσεων / τα πρωτότυπα συναρτήσεων τα οποία βρίσκονται έξω από κάθε συνάρτηση

Εμβέλεια μπλοκ

- Ένα αναγνωριστικό που ορίζεται μέσα σε ένα μπλοκ εντολών έχει εμβέλεια μπλοκ ; είναι γνωστό μέχρι το δεξί άγκιστρο που κλείνει το μπλοκ.
- Σε αυτή την κατηγορία ανήκουν
  - οι τοπικές μεταβλητές μιας συνάρτησης που δηλώνονται στην αρχή της συνάρτησης
  - οι τυπικές παράμετροι μιας συνάρτησης που δηλώνονται μέσα στην παρένθεση που ακολουθεί το όνομα της συνάρτησης επειδή αντιγράφονται αυτόματα σε τοπικές μεταβλητές με το ίδιο ονομα
- Όταν έχουμε εμφωλευμένα οκ μπλοκ εντολών, και κάποιο αναγνωριστικό στο εξωτερικό μπλοκ έχει το ίδιο όνομα με ένα αναγνωριστικό στο εσωτερικό μπλοκ τότε το αναγνωριστικό στο εξωτερικό μπλοκ είναι κρυμμένο μέχρι να τερματιστεί το εσωτερικό μπλοκ.

Εμβέλεια πρωτοτύπου συναρτήσης

- Εμβέλεια πρωτοτύπου συναρτήσης έχουν μόνο οι τυπικές παράμετροι μιας συνάρτησης.
- Ο μεταγλωττιστής αγνοεί τα ονόματα των τυπικών παραμέτρων και διαβάζει μόνο τους τύπους τους
- Αρα το ίδιο όνομα μπορεί να χρησιμοποιηθεί οπουδήποτε στο πρόγραμμα χωρίς να υπάρχει ασάφεια.

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 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])
}
```

Αναδρομή (recursion)

- Μία συνάρτηση είναι αναδρομική όταν μία εντολή στο σώμα της συνάρτησης καλεί την ίδια τη συνάρτηση. Η αναδρομή μερικές φορές λέγεται και *κυκλικός ορισμός*.

- Ένα πολύ χαρακτηριστικό παράδειγμα αναδρομής είναι η συνάρτηση η οποία υπολογίζει το παραγοντικό ενός ακεραίου αριθμού.

Συνάρτηση του παραγοντικού

- Ορισμός του n!
  n! = n x (n - 1) x … x 2 x 1
- Ο παραπάνω ορισμός μπορεί να γραφεί ως

$$n! = \begin{cases} 1 & \text{αν } n = 0 \\ n \times (n-1)! & \text{αλλιώς} \end{cases}$$

- Στον εναλλακτικό ορισμό, ο υπολογισμός του n! (σύνθετο πρόβλημα) ανάγεται στον υπολογισμό του (n-1)! (απλούστερο πρόβλημα).

Βασική δομή

```
συνάρτηση()
{

    if (έλεγχος απλής περίπτωσης) {
        return (απλή λύση χωρίς τη χρήση αναδρομής);
    } else {
        return (αναδρομική κλήση που απαιτεί την κλήση της
        ίδιας συνάρτησης);
}
```

Κατά τη χρήση της αναδρομής

- Βρίσκουμε τη λύση για την απλή περίπτωση.
- «Φανταζόμαστε» ότι έχουμε βρει τη λύση για μια (ή περισσότερες) απλούστερες περιπτώσεις
- Με βάση τις παραπάνω απλούστερες περιπτώσεις, συνθέτουμε τη λύση για τη γενικότερη περίπτωση

Συνάρτηση ύψωσης σε δύναμη

```
static int RaiseIntToPower(int n, int k)
{
   /* Έλεγχος απλής περίπτωσης */
   if (k ==0) {
      return (1);
   } else {
   /* Αναδρομική κλήση της ίδιας συνάρτησης */
      return (n * RaiseIntToPower(n, k-1));
   }
}
```

ΠΑΡΑΤΗΡΗΣΗ

- Όταν γράφουμε αναδρομικές συναρτήσεις, θα πρέπει να προβλέπουμε μία εντολή ελέγχου η οποία θα προκαλέσει το τέλος των πράξεων της συνάρτησης δηλαδή, να μην εκτελεστεί μια νέα επί πλέον αναδρομική κλήση.
- Αν δεν το κάνουμε και καλέσουμε μια αναδρομική συνάρτηση χωρίς έλεγχο τότε, αυτή η κλήση δεν θα επιστρέψει ποτέ και θα επαναλαμβάνεται συνεχώς η αναδρομή.
- Αυτό το σφάλμα εμφανίζεται συχνά, όταν γράφουμε αναδρομικές συναρτήσεις και για το λόγο αυτό χρειάζεται ιδιαίτερη προσοχή.

Τι θα συμβεί αν λείπει ο έλεγχος της απλής περίπτωσης;

- Τι κάνει το παρακάτω πρόγραμμα;

```
main() { inc(1);}

int inc(int k)
{
  k++;
  inc(k);
}
```

Ατέρμονος αναμονή

- Θεωρητικά, η προηγούμενη συνάρτηση, inc, όταν κληθεί, καλεί επ' άπειρον τον εαυτό της
  - Το πρόγραμμα μπαίνει σε μια ατέρμονα επανάληψη («κολλάει»)
- Πρακτικά, μετά από κάποιο χρόνο το πρόγραμμα τερματίζει εμφανίζοντας ένα μήνυμα σφάλματος.
  - π.χ. `Stack overflow` ή
  - `Segmentation Fault`

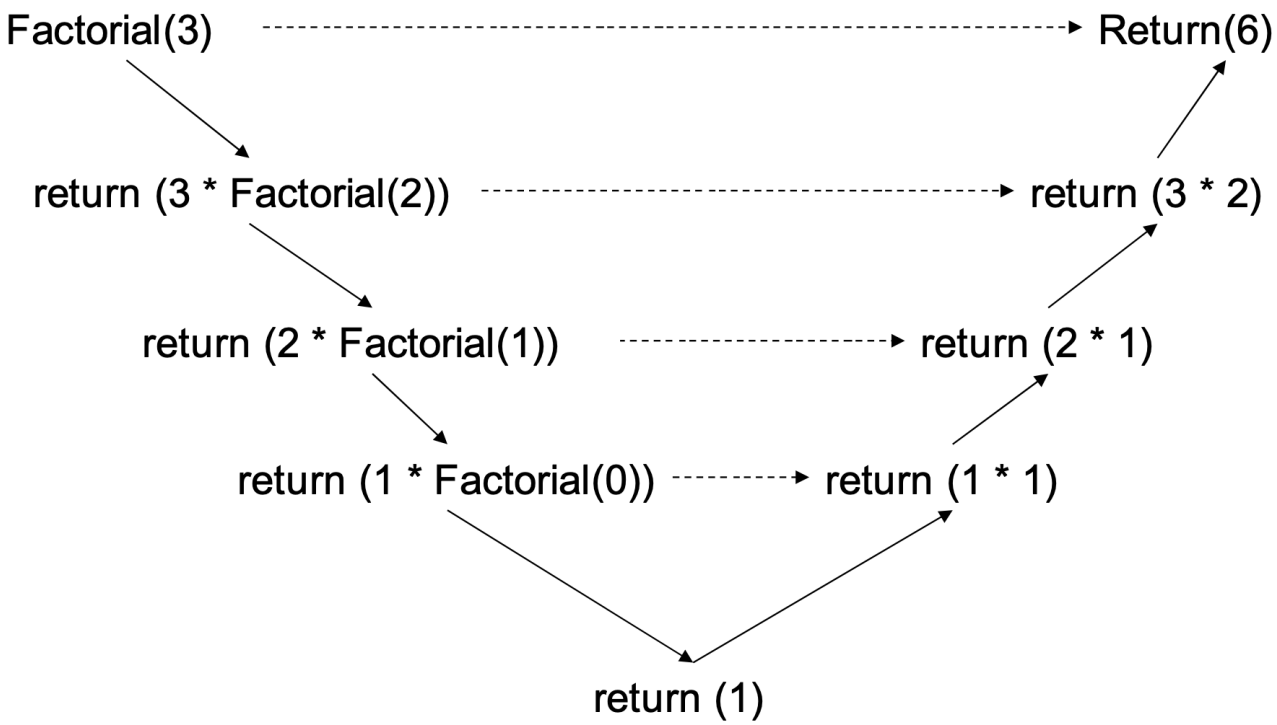
Γιατί συμβαίνει αυτό;

- Σε κάθε κλήση συνάρτησης (αναδρομική ή όχι) δεδομένα εγγράφονται σε μια περιοχή της μνήμης που ονομάζεται Στοίβα (Stack).
- Τέτοια δεδομένα είναι:
  - Οι τιμές των πραγματικών παραμέτρων
  - Η διεύθυνση επιστροφής
  - Τοπικές μεταβλητές
- Μετά από ένα μεγάλο αριθμό αναδρομικών κλήσεων η στοίβα υπερχειλίζει (overflow) και το πρόγραμμα τερματίζει με την εμφάνιση κατάλληλου μηνύματος.

Παράδειγμα

```
int Factorial(int n)
{
   if (n == 0)
      return 1;
   return (n * Factorial(n-1));
}
```

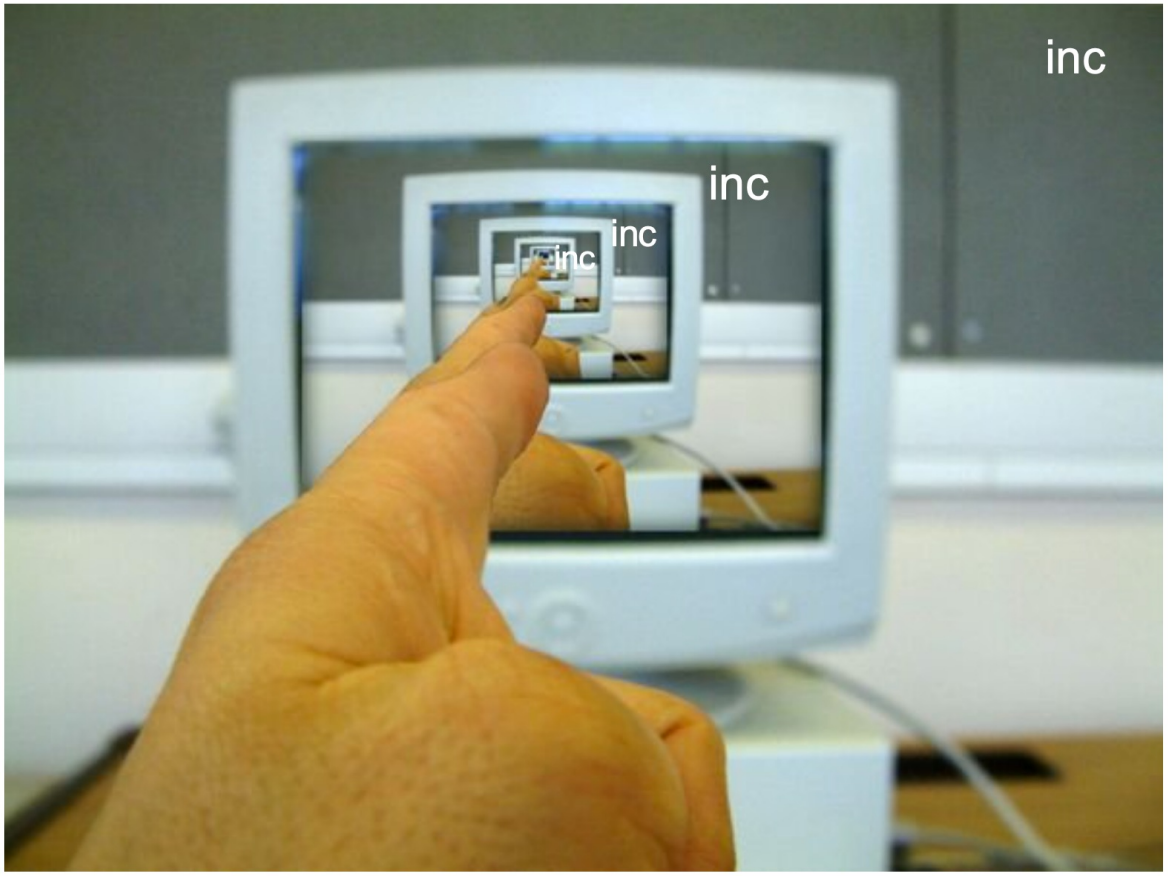
Παράδειγμα κλήσης factorial



Επαναληπτική υλοποίηση

```
int Factorial(int n)
{
   int i, result;
   result = 1;
   for (i = 2; i<= n; i++)
   {
     result = result * i;
   }
   return result;
}
```

Περίπτωση άπειρων αναφορών μιας οντότητας στον εαυτό της



Αναδρομική έκδοση υπολογισμού του παραγοντικού

```
int fact_r(int n) {
    int answer;
    if( n==1 ) return(1);
// Αναδρομική κλήση της συνάρτησης fact_r( )
    answer=fact_r(n-1)*n;
    return(answer);
}
```

Αναδρομική έκδοση υπολογισμού του παραγοντικού

```c
// Fig. 5.18: fig05_18.c
// Recursive factorial function.
#include <stdio.h>

unsigned long long int factorial(unsigned int number);

int main(void)
{
   // during each iteration, calculate
   // factorial(i) and display result
   for (unsigned int i = 0; i <= 21; ++i) {
      printf("%u! = %llu\n", i, factorial(i));
   }
}

// recursive definition of function factorial
unsigned long long int factorial(unsigned int number)
{
   // base case
   if (number <= 1) {
      return 1;
   }
   else { // recursive step
      return (number * factorial(number - 1));
   }
}
```

Μη αναδρομική έκδοση υπολογισμού του παραγοντικού

```
int fact(int n){
    int t, answer;
    answer = 1;          // Αρχική τιμή του παραγοντικού
    for( t = 1; t <= n ; t++)
  // Υπολογισμός των διαδοχικών τιμών
     answer = answer * t;
     return(answer);
}
```

ΑΣΚΗΣΗ

- Να γράψετε ένα πρόγραμμα το οποίο να ζητά να πληκτρολογήσει ο χρήστης ένα αριθμό μεταξύ των αριθμών -25 και 25.
- Να γράψετε και μία αναδρομική συνάρτηση η οποία:
  - να δέχεται μια ακέραια παράμετρο και
  - να εμφανίζει τους αριθμούς από την τιμή της παραμέτρου της μέχρι τον αριθμό επτά (7) στην οθόνη.

```
#include<stdio.h>
int fact2(int);
int fact3(int);  // Δήλωση των συναρτήσεων
void main(){
    int k;
    printf("Πρόγραμμα το οποίο ζητά ένα αριθμό μεταξύ των  -25 και 25
            (έστω m) \n");
    printf(" και εμφανίζει όλους τους αριθμούς από το m μέχρι τον αριθμό
             7  \n");
    do  {
        printf("\nΔώστε ένα αριθμό > -25 και < 25 : ");
        scanf("%d",&k);
        fflush(stdin);
     } while ( ! ( k < 25 && k > -25) );  // έλεγχος της τιμής εισόδου
```

```
    printf("Εμφάνιση των αριθμών από %d μέχρι τον αριθμό 7
    \n",k);
    /* Κλήση της κατάλληλης συνάρτησης  */
    if( k >= 7)
       fact2(k);
    else
       fact3(k);
}
```

```
int fact2(int n) {
/* Αναδρομική συνάρτηση όταν η παράμετρος >= 7  */
    if( n == 7 )
        return (printf("%d \n", n));
    else {
        printf("%d ", n);
        return( fact2( n - 1) );
    }
    printf("\n");
}
```

```
int fact3(int n) {
/* Αναδρομική συνάρτηση όταν η παράμετρος < 7  */
    if( n == 7)
       return( printf(” %d \n", n ));
     else {
        printf("%d ", n);
        return( fact3(n + 1));
     }
     printf( "\n") ;
}
```

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.

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

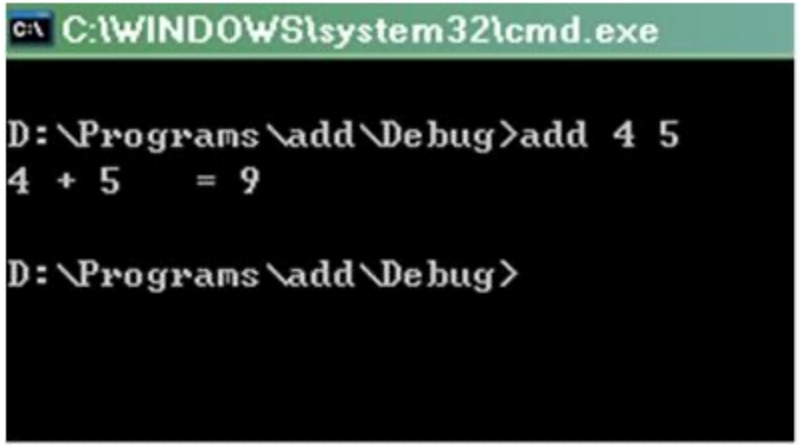
- **Να γράψετε ένα πρόγραμμα προσομοίωσης μιας ζαριάς δηλαδή, τυχαία εμφάνιση δύο αριθμών με τιμές 1 μέχρι 6.**

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

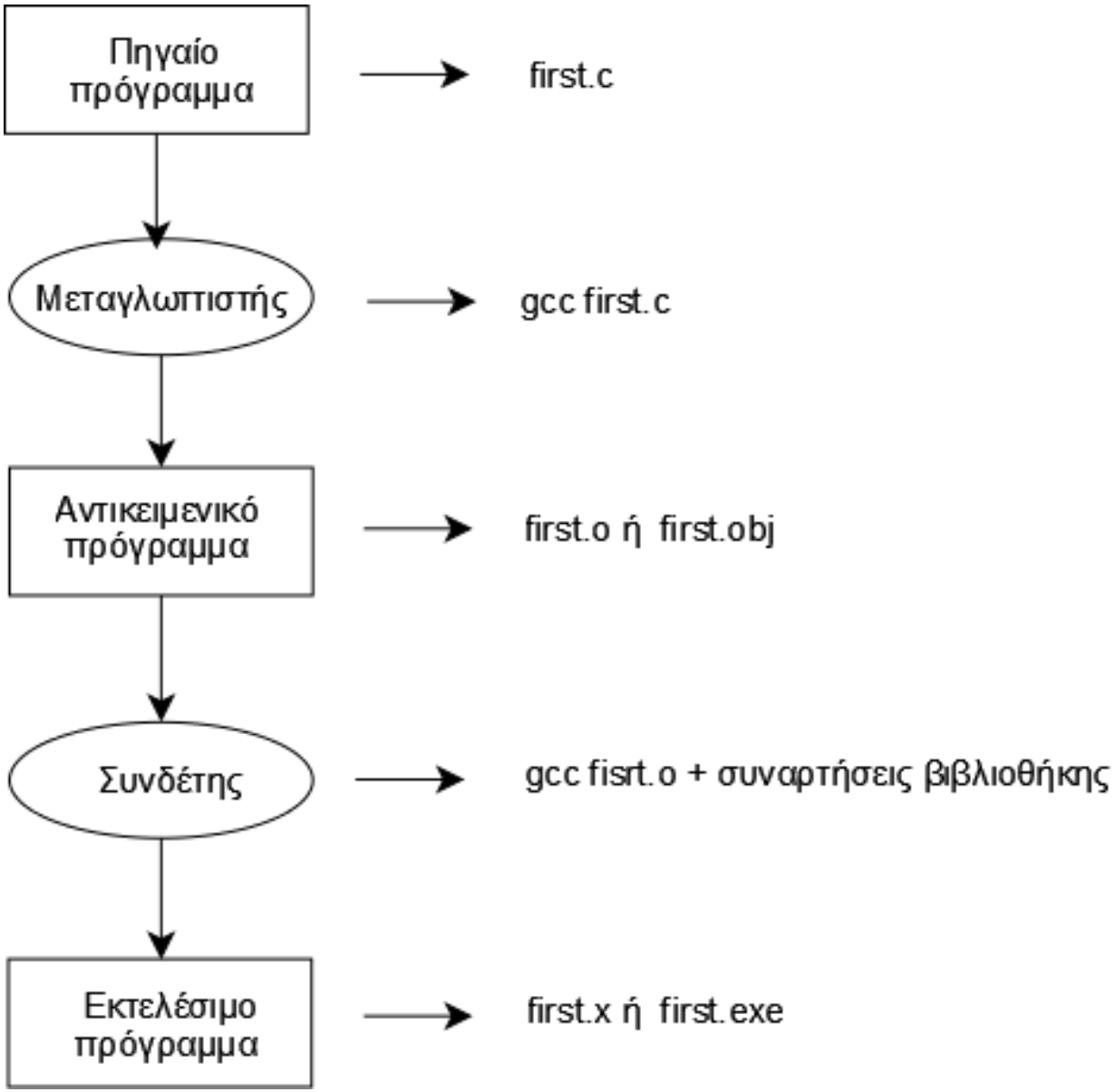
Στη γλώσσα C, το όνομα ενός επικεφαλής αρχείου εξ ορισμού τελειώνει με την επέκταση .h και χρησιμοποιείται με την επίκληση της οδηγίας (directive) # include

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:

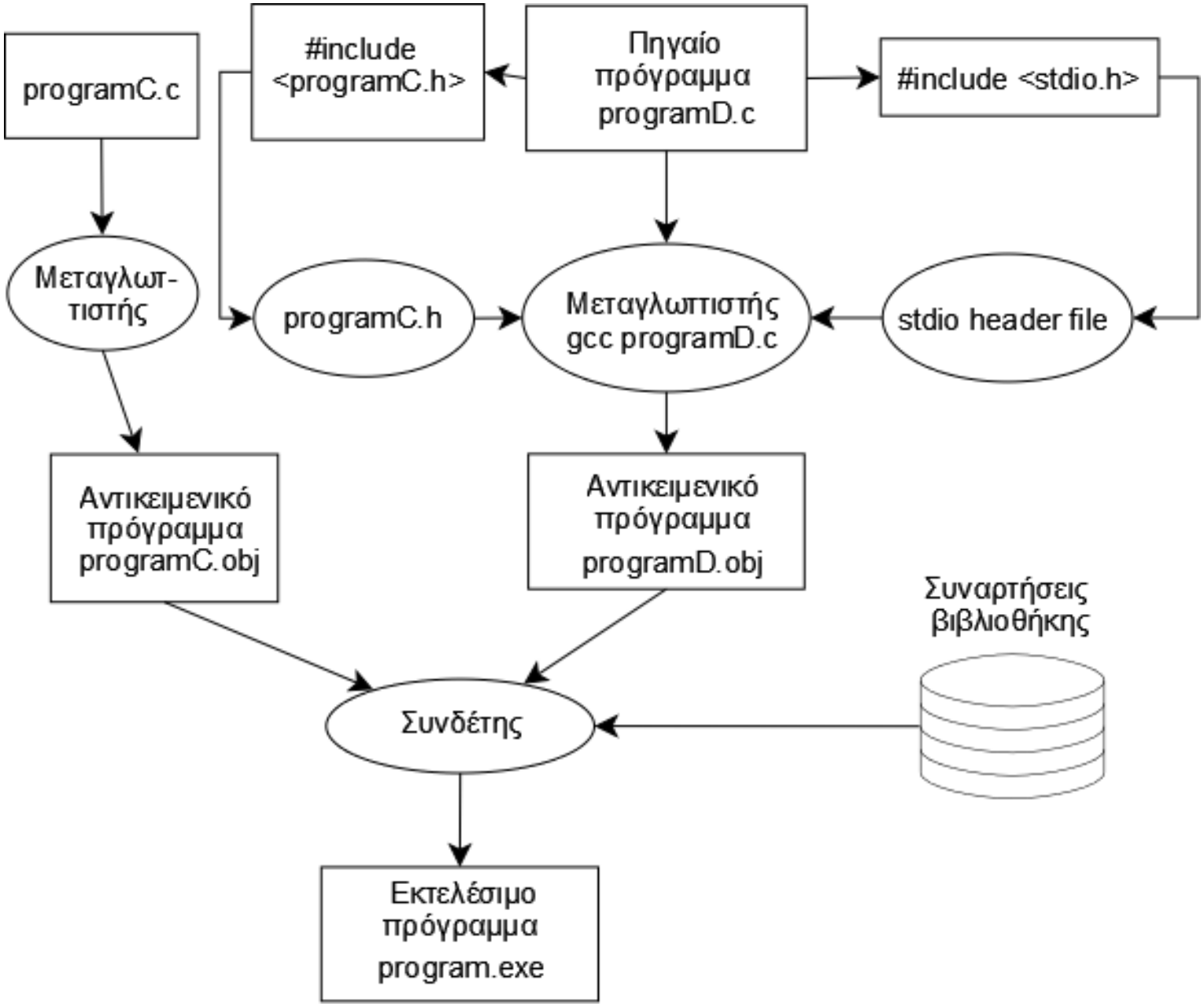
1. Σε περιβάλλον Unix/Linux γράφουμε :

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

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

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.
- Στο πρόγραμμα 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 τέλειους 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 τέλειων  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 τέλειος αριθμός   */
       if ( perfect( number ) )
       {
            printf( "Ο αριθμός %d είναι τέλειος \n", number );
        }
    }
   printf( "\n" );
   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;
}
```