Συναρτήσεις

Αρθρωτός Σχεδιασμός – Συναρτήσεις (modular design - functions)

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

Τμηματοποίηση

Τεμαχισμός μεγάλων και σύνθετων προβλημάτων σε μικρά, απλά τμήματα.

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

Μία συνάρτηση…

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

Ήδη γνωστές συναρτήσεις

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

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

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

Βασικά στοιχεία των συναρτήσεων

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

Βασικά στοιχεία των συναρτήσεων

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

1) *Πρόταση δήλωσης της συνάρτησης*

2) *Πρόταση ορισμού της συνάρτησης*

3) *Πρόταση κλήσης της συνάρτησης*

Παράδειγμα: Χωρίς συναρτήσεις

/* Μετατροπή βαθμών Φαρενάιτ σε βαθμούς Κελσίου */

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

θα ενταχθούν σε συνάρτηση

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

Παράδειγμα: Με συναρτήσεις

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

Δήλωση συνάρτησης

Κλήση συνάρτησης

Σώμα συνάρτησης

Όνομα συνάρτησης

Δήλωση συναρτήσεων

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

- **Παράδειγμα:**

```
int make_tea(float water, int teabags );
```

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

Σύνταξη συναρτήσεων

(μην ξεχνάτε το ;)

- **Σύνταξη της δήλωσης της συνάρτησης:**

  ***return_type* function_name(*argument_list*);**

- **Σύνταξη της συνάρτησης:**

***return_type* function_name(*argument_list*)**

{

προτάσεις;

}

(Σχεδόν τα ίδια)

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

Δήλωση και Ορισμός

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

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

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

όνομα συνάρτησης

λίστα ορισμάτων (ένα όρισμα)

Επιστρεφόμενη τιμή

ορισμός συνάρτησης

Γενική μορφή ενός C προγράμματος

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

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

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

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

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

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

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

```
#include <…h>
#define ......
```

→ ***include και define***

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

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

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

need_to_declare.cpp

Τοπικές μεταβλητές (local variables)

- **Έχουν νόημα <u>μόνο</u> μέσα στη συνάρτηση που δηλώνονται.**

- **Δύο διαφορετικές συναρτήσεις μπορούν να έχουν τοπική μεταβλητή με το ίδιο όνομα χωρίς να παρουσιάζεται πρόβλημα.**

Παράδειγμα με τοπικές μεταβλητές

```
#include<stdio.h>

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

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

Ίδια ονόματα τοπικών μεταβλητών

Αποτέλεσμα στην οθόνη out=16.0

out=24.5 μέσα στην square

Καθολικές μεταβλητές (global variables)

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

Παράδειγμα με καθολικές μεταβλητές

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

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

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

Η glob γίνεται 3.0

func_call_to_func.cpp

Εμβέλεια μεταβλητών (scope)

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

Εμβέλεια μεταβλητών (συνέχεια)

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

Παράδειγμα: Να προσδιορισθεί η εμβέλεια του ακόλουθου πηγαίου κώδικα:

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

Επεξηγήσεις στην επόμενη διαφάνεια

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

```
if (a>b) return a;
else return b;
```

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

//structured_prog_example_2.cpp

Επεξηγήσεις:

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

buildingArea.cpp

Διάρκεια μεταβλητών (duration)

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

Παράδειγμα:

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

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

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

Παράδειγμα:

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

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

Λύση:

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

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

Παράδειγμα στατικών μεταβλητών:

```
#include <stdio.h>

float get_average(float newdata);   // δήλωση συνάρτησης
main()
{
  float data=1.0;
  float average;
  while (data!=0)
  {
    printf( "\n Give a number or press 0 to finish: " );
    scanf( "%f",&data );
    average=get_average(data);
    printf( "The new average is %f\n",average );
  }
}  // τέλος της main, συνέχεια στην επόμενη διαφάνεια
```

//structured_prog_example_3.cpp

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

Εκτελούνται μόνο την πρώτη φορά. Τις επόμενες διατηρούν το αποτέλεσμα της προηγούμενης κλήσης και σε αυτό προστίθενται στη μεν total το newdata, στη δε count η μονάδα.

Αποτέλεσμα:

C:\temp\prog.exe

```
 Give a number or press 0 to finish: 10
The new average is 10.000000

 Give a number or press 0 to finish: 20
The new average is 15.000000

 Give a number or press 0 to finish: 30
The new average is 20.000000

 Give a number or press 0 to finish: 40
The new average is 25.000000

 Give a number or press 0 to finish: 280
The new average is 76.000000
```

10/1=10

(10+20)/2=15

(30+30)/3=20

Κλήση συναρτήσεων με πίνακες

function_call_array.cpp

Παράδειγμα με επιστροφή μεταβλητών δομής

```
#include <stdio.h>
struct Distance
{
  int feet;
  int inches;
};
Distance addeng1(Distance dd1, Distance dd2);  // δήλωση addeng1
void eng1disp(Distance dd);   // δήλωση eng1disp

main()
{
  Distance d1,d2,d3;
  printf("\n1st number of feet:");  scanf("%d",&d1.feet);
  printf("\n1st number of inches:");  scanf("%d",&d1.inches );
  printf("\n2nd second number of feet:"); scanf("%d",&d2.feet);
  printf("\n2nd second number of inches:");
  scanf("%d",&d2.inches);
```

Εισαγωγή δεδομένων

```
  d3=addeng1(d1,d2);     // d3 = d1 + d2
  eng1disp(d1);
  printf( " +" );
  eng1disp(d2);
  printf( " =" );
  eng1disp(d3);
  printf( "\n" );
} // τέλος της main

void eng1disp(Distance dd)
{
  printf( "%d'%d" ",dd.feet,dd.inches );
}
```

Δέχεται μία δομή τύπου Distance και εμφανίζει στην οθόνη το περιεχόμενό της.

```
Distance addeng1(Distance dd1, Distance dd2)
{
  Distance dd3;
  dd3.inches=dd1.inches+dd2.inches;
  dd3.feet=0;
  if (dd3.inches>=12)
  {
    dd3.inches=dd3.inches-12;
    dd3.feet++;
  }
  dd3.feet=dd3.feet+dd1.feet+dd2.feet;
  return(dd3);
}
```

*Αν ο αριθμός των ιντσών υπερβαίνει το 12 συμπληρώνεται ένα πόδι, επομένως πρέπει να προστεθεί η μονάδα στη μεταβλητή **feet** και να αφαιρεθούν 12 ίντσες από τη μεταβλητή **inches**.*

//structured_prog_example_4.cpp

Αποτέλεσμα:

```
C:\temp\prog.exe

Give the first number of feet:5

Give the first number of inches:4

Give the second number of feet:3

Give the second number of inches:9
5'4''   +3'9''   =9'1''
```

Παράδειγμα

```
struct addressT  {
  char street_name[40];
  int street_number;
  char city[40];  };
struct personT  {
   addressT addr;
   char name[50];  };
main()  {
  personT pers;
  addressT ad;
  . . . . . .
  ad=func(pers);
}
addressT func(personT prsn)  {
  . . . . . .
  return(prsn.addr);
}
```

Τιμές στη μεταβλητή *pers*

Τιμές <u>τουλάχιστον</u> στο μέλος *prsn.addr*

Αναδρομικότητα (recursion)

Μία συνάρτηση ονομάζεται αναδρομική όταν μία εντολή του σώματος της συνάρτησης καλεί τον ίδιο της τον εαυτό. Η αναδρομή είναι μία διαδικασία με την οποία ορίζουμε κάτι μέσω του ίδιου του οριζόμενου.

Παράδειγμα: Να ορισθεί συνάρτηση που υπολογίζει το άθροισμα των αριθμών από 1 έως n.

(επεξηγήσεις στην επόμενη διαφάνεια)

1ος τρόπος:

```
int sum(int n)
{
  int i, total=0;
  for (i=0;i<=n;i++)
    total+=i;
  return(total);
}
```

2ος τρόπος (με αναδρομή):

```
int sum(int n)
{
  if (n<=1) return(n);
  else return(sum(n-1)+n);
}
```

Μέσα στη *sum()* καλείται ο εαυτός της.

Επεξηγήσεις:

```
if (n<=1) return(n);
else return(sum(n-1)+n);
```

Εάν το **n** είναι ίσο με 1, τότε το άθροισμα ταυτίζεται με το **n** (οριακή περίπτωση). Στη γενική περίπτωση, θεωρούμε ότι ο υπολογισμός του αθροίσματος **n** μπορεί να θεωρηθεί ως υπολογισμός του αθροίσματος των αριθμών από το 1 έως το **n-1** συν το **n**. Αντίστοιχα, ο υπολογισμός του αθροίσματος **n-1** μπορεί να θεωρηθεί ως υπολογισμός του αθροίσματος των αριθμών από το 1 έως το **n-2** συν το **n-1**. Ακολουθώντας την παραπάνω διαδικασία, μπορούμε να ορίσουμε τα εξής:

$$1+\ldots+n = (1+\ldots+(n-1)) + n$$
$$(1+\ldots+(n-1)) = (1+\ldots+(n-2)) + (n-1)$$
$$(1+\ldots+(n-2)) = (1+\ldots+(n-3)) + (n-2)$$
$$(1+\ldots+(n-3)) = (1+\ldots+(n-4)) + (n-3)$$

κ.ο.κ.

Από τα παραπάνω προκύπτει ότι κάθε σχέση είναι ίδια με την προηγούμενη, με απλή αλλαγή των ορισμάτων.

Τι σημαίνει όμως αυτό;

Την πρώτη φορά καλείται η συνάρτηση ***sum()*** με όρισμα **n**. Με την πρόταση ***return(sum(n-1)+n)*** η ***sum()*** καλεί τον ίδιο της τον εαυτό με διαφορετικό όμως όρισμα (**n-1**). Η ενεργοποίηση αυτή θα προκαλέσει με τη σειρά της νέα ενεργοποίηση και αυτό θα συνεχισθεί έως ότου προκληθεί διακοπή. Η διακοπή είναι αποκλειστική ευθύνη του προγραμματιστή. Στο συγκεκριμένο παράδειγμα η διακοπή προκαλείται με την πρόταση ***if (n<=1) return(n)*** ,που σημαίνει ότι όταν το **n** φθάσει να γίνει 1 υπάρχει πλέον αποτέλεσμα. Έτσι οι διαδοχικές κλήσεις για **n=4** είναι:

sum(4) καλεί τη sum(3)

sum(3) καλεί τη sum(2)

sum(2) καλεί τη sum(1)

η sum(1) δίνει αποτέλεσμα 1 και το επιστρέφει στη sum(2)

η sum(2) δίνει αποτέλεσμα 1+2=3 και το επιστρέφει στη sum(3)

η sum(3) δίνει αποτέλεσμα 3+3=6 και το επιστρέφει στη sum(4)

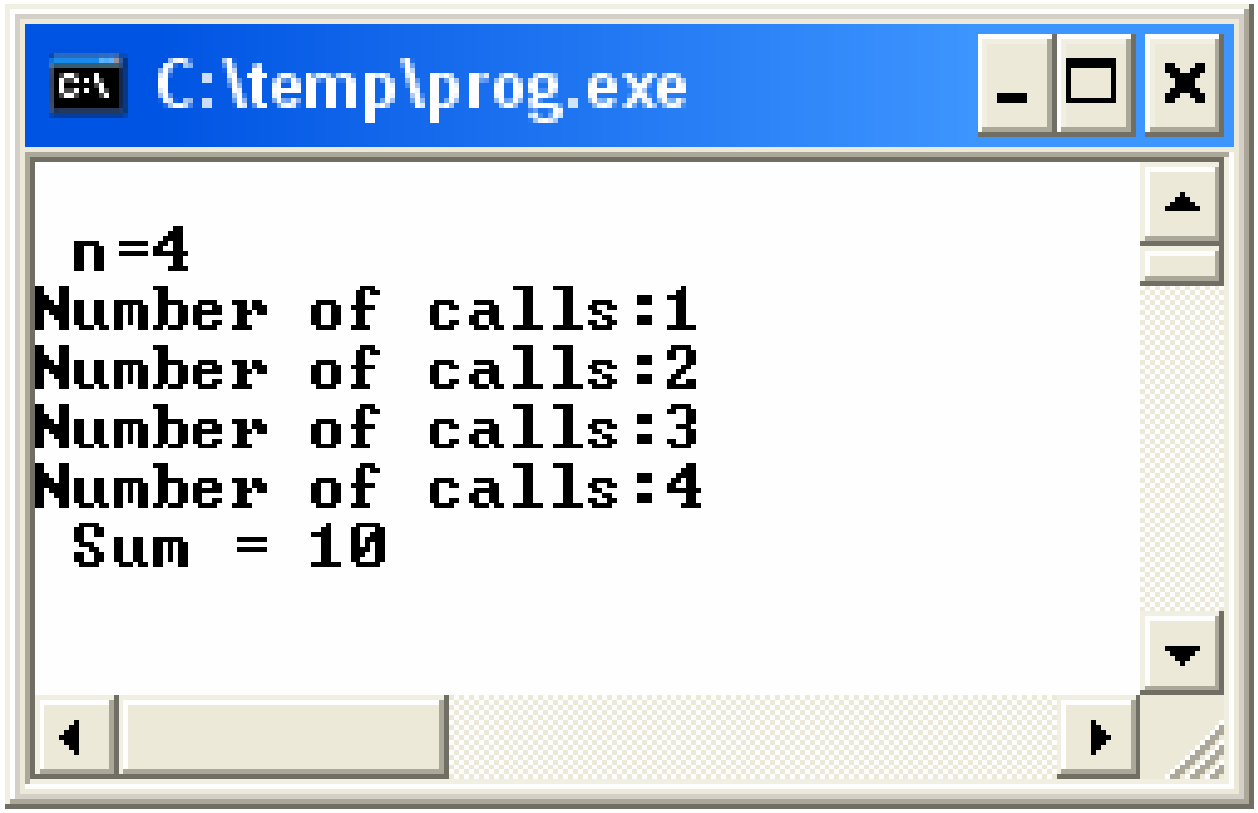
η sum(4) δίνει αποτέλεσμα 6+4=10, το οποίο είναι και το τελικό

Το πλήρες πρόγραμμα έχει την ακόλουθη μορφή: ***//structured_prog_example_5.cpp***

```
#include <stdio.h>

int sum(int n);              // δήλωση της συνάρτησης sum
int number_of_calls=0;
main(){
  int n=4;                   // ανάθεση n=4
  printf( "\n n=%d",n );
  printf( "\n Sum = %d", sum(n) );
  printf( "\n   Press any key to finish" ); getch();
}   // τέλος της main
int sum(int n){              // ορισμός της συνάρτησης sum
  if (n<=1){
    number_of_calls++;
    printf( "\nNumber of calls:%d",number_of_calls );
    return(n);
  }
  else{
    number_of_calls++;
    printf( "\nNumber of calls:%d",number_of_calls );
    return(sum(n-1)+n);
  }
}  // τέλος της sum
```

Αποτέλεσμα:


C:\temp\prog.exe
n=4
Number of calls:1
Number of calls:2
Number of calls:3
Number of calls:4
Sum = 10

Μειονεκτήματα αναδρομικών συναρτήσεων

- **Οι περισσότερες αναδρομικές συναρτήσεις δεν εξοικονομούν σημαντικό μέγεθος κώδικα ή μνήμης για τις μεταβλητές.**
- **Οι αναδρομικές εκδοχές των περισσότερων συναρτήσεων μπορεί να εκτελούνται κάπως πιο αργά από τα επαναληπτικά τους ισοδύναμα εξαιτίας των πρόσθετων κλήσεων σε συναρτήσεις. Η πιθανή μείωση όμως δεν είναι αξιοσημείωτη.**
- **Υπάρχει μικρή πιθανότητα οι πολλές αναδρομικές κλήσεις μίας συνάρτησης να προκαλέσουν *υπερχείλιση της στοίβας* (stack overflow), επειδή ο χώρος αποθήκευσης των παραμέτρων και των τοπικών μεταβλητών της συνάρτησης είναι στη στοίβα και κάθε νέα κλήση παράγει ένα νέο αντίγραφο αυτών των μεταβλητών. Ωστόσο, εφόσον διατηρείται ο έλεγχος της αναδρομικής συνάρτησης και υπάρχει συνθήκη διακοπής, το ζήτημα είναι ήσσονος σημασίας.**

Πλεονεκτήματα αναδρομικών συναρτήσεων

- **Το βασικότερο πλεονέκτημα των αναδρομικών συναρτήσεων είναι ότι μπορούν να χρησιμοποιηθούν για να δημιουργηθούν καθαρότερες και απλούστερες εκδοχές πολλών αλγορίθμων.**
- **Δημιουργείται συμπαγέστερος κώδικας και είναι ιδιαίτερα χρήσιμες σε αναδρομικώς οριζόμενα δεδομένα όπως οι λίστες και τα δένδρα.**

Άσκηση: ***Να καταστρωθεί πρόγραμμα, με χρήση αναδρομικής συνάρτησης, το οποίο δέχεται ως είσοδο από το πληκτρολογίο έναν ακέραιο αριθμό n και επιστρέφει στην οθόνη το παραγοντικό του.***

$$n! = 1 \bullet 2 \bullet \ldots \bullet n$$

Λύση: ***Το πρόβλημα είναι απολύτως αντίστοιχο με εκείνο του προηγούμενου παραδείγματος. Οι μόνες διαφορές είναι ότι πρέπει να διαβάζεται ο n και ότι πολλαπλασιάζονται ακέραιοι και δεν αθροίζονται. Κατά συνέπεια, μία πιθανή λύση είναι η εξής:***

```
#include <stdio.h>
                                                    //factorial.cpp

int fact(int n);
main(){
  int n;
  printf( "Give n:" );
  scanf( "%d",&n );
  printf( "\n The factorial of %d is %d",n,fact(n) );
  printf( "\n       Press any key to finish" );
  getch();
}  // τέλος της main

int fact(int n){
  if (n<=0)
  {
    printf( "\nERROR! n should be a positive integer\n");
    return(0);        //0 is returned in case of error
  }
  else if (n<=1) return(n);
  else return(fact(n-1)*n);
}  // τέλος της fact
```