Laborator 10 – Funcţii

Funcţia reprezintă o secvenţă de declaraţii şi instrucţiuni care realizează o anumită sarcină.
Noţiunea de funcţie s-a dezvoltat din necesitatea de a utiliza în cadrul unui program aceeaşi secvenţă de calcul de mai multe ori. Astfel o funcţie creată poate fi apelată în diferite puncte ale programului de atâtea ori cât este necesar.

Funcţiile extind facilităţile limbajului pentru a se adapta unor aplicaţii dezvoltate de programator. De asemenea funcţiile se pretează unor tehnici de programare structurată, întrucât sunt componente independente ale programului. Funcţiile din limbajul C sunt similare subrutinelor din limbajul FORTRAN sau procedurilor din limbajul Pascal. Orice program trebuie să conţină cel puţin o funcţie: main().

Sintaxa definiţiei unei funcţii (dată de standardul ANSI C) este următoarea:

```
Antetul funcţiei
{
Corpul funcţiei
}
```

sau dezvoltat

```
tip returnat nume_functie(lista parametri formali)
{
secvenţa de declaraţii
secvenţa de instrucţiuni
}
```

tip returnat reprezintă tipul de date returnate de funcţie. O funcţie poate returna orice tip de date în afară de tablouri.

În funcţie de tipul valorii returnate se deosebesc două tipuri de funcţii în limbajul C:
-funcţii care nu returnează o valoare la revenirea din ele: tip_returnat este void;
-funcţii care returnează o valoare la revenirea din ele.

Observaţie:
Dacă în definiţia funcţiei nu este precizat tipul, acesta este implicit int.

Expresia lista parametri formali este o listă de parametri separaţi prin virgule, conţinând tipul şi numele fiecărui parametru.

Chiar şi în cazul în care funcţia nu are nici un parametru, parantezele sunt necesare sau apare explicit cuvântul void.

Spre deosebire de declaraţia variabilelor, toţi parametrii unei funcţii trebuie declaraţi explicit individual.
Lista de parametri şi de tipuri are următorul format:
tip1 parametru1, tip2 parametru2,…

Corpul funcţiei constă din declaraţii şi instrucţiuni şi este inclus între acolade. După acolada de închidere nu se pune punct şi virgulă.

În limbajul C nu este permis să se definească o funcţie în interiorul altei funcţii, de aceea toate funcţiile au acelaşi nivel al domeniului de acţiune.

Apelul funcţiilor

a. Apelul funcţiilor care nu returnează o valoare

În acest caz apelul este realizat printr-o instrucţiune de apel:
nume_functie(lista parametri efectivi);

Expresia lista parametri efectivi este o expresie sau mai multe separate prin virgulă sau vidă dacă funcţia nu are parametri. Parametrii efectivi corespund ca număr, ordine (preferabil şi de tip) cu parametrii formali. Parametrii formali şi cei efectivi sunt denumiţi şi argumente. În momentul apelului, parametrii formali sunt înlocuiţi cu valorile parametrilor efectivi, iar la revenirea din funcţie valorile parametrilor formali se pierd.

Compilatorul găseşte şi semnalează orice conversie de tip nepermisă între tipul argumentelor folosite la apel şi tipul parametrilor formali. Dacă tipul parametrului formal diferă de tipul parametrului efectiv, limbajul C permite conversia automată a valorii parametrului efectiv spre tipul parametrului formal corespunzător. Datorită faptului că in limbajul C++ regula apelului este mai complexă, se recomandă să nu se folosească nici în C regula de conversie automată.

De exemplu dacă funcţie are un parametru de tip float, iar n este de tip int, se recomandă a folosi apelul cu conversie explicită spre tipul float.
functie((float)n);

Dacă o funcţie nu are parametri, prototipul său foloseşte void în locul listei de parametri.
De exemplu
void g(void);
funcţia g nu are parametri şi nu returnează nici o valoare. Orice apel al funcţiei cu parametrii reprezintă o eroare.

b. Apelul funcţiilor care returnează o valoare
În acest caz apelul este realizat printr-o instrucţiune de apel:
nume_functie(lista parametri efectivi);
sau sub formă de operand al unei expresii.
În primul caz se pierde valoarea returnată iar în cel de-al doilea caz valoarea returnată se foloseşte pentru evaluarea expresiei respective.

Revenirea dintr-o funcţie

Revenirea dintr-o funcţie se poate realiza în două moduri:
1. după ce ultima instrucţiune din funcţie a fost executată şi a fost întâlnită acolada de închidere;
2. după ce a fost întâlnită instrucţiunea return.

Majoritatea funcţiilor folosesc instrucţiunea return pentru a stopa execuţia unei funcţii, fie deoarece trebuie returnată o valoare fie pentru a simplifica şi a mări eficienţa codurilor funcţiei. Dacă funcţia nu returnează o valoare se va folosi instrucţiunea
return;
fără a include nici o expresie după cuvântul cheie return.

Dacă funcţia returnează o valoare se va folosi instrucţiunea
return expresie;
Valoarea expresiei este chiar valoarea returnată de funcţie.

Dacă tipul expresiei diferă de tipul funcţiei dată în antetul funcţiei se va face conversia automat spre tipul din antet, chiar înainte de a se reveni din funcţie.

Instrucţiunea return poate să se găsească oriunde în corpul funcţiei, ceea ce măreşte flexibilitatea programelor scrise în C. Toate funcţiile exceptând cele de tip void, returnează valori care sunt indicate explicit de instrucţiunea return. Dacă nu există nici o instrucţiune return, valoarea returnată este nedefinită din punct de vedere tehnic.
O funcţie nu poate fi obiectul unei atribuiri. De exemplu următoarea instrucţiune este incorectă, în program semnalându-se eroare:
max(x,y)=200;

Apelul prin valoare

În limbajul C se foloseşte în majoritatea cazurilor apelul prin valoare pentru transmiterea argumentelor. La apelul unei funcţii, fiecărui parametru formal i se atribuie valoarea parametrului efectiv care-i corespunde. Parametrii efectivi sunt protejaţi, ei nu pot fi modificaţi în acest caz, funcţia apelată neavând acces la ei.

Transferul datelor este unidirecţional de la funcţia care face apelul spre cea care a fost apelată dar nu şi invers. Acest transfer prin valoare prezintă avantaje în cazul în care funcţia apelată nu trebuie să modifice parametrii efectivi.

Apelul prin referinţă

În cazul acestui tip de apel, adresa unui parametru efectiv este copiată în locaţia de memorie a parametrului formal. Astfel funcţia apelată va putea modifica valoarea parametrilor efectivi având acces la adresa acestora. Transferul datelor este bidirecţional între funcţia care face apelul şi funcţia de apelat.

Apelul este prin referinţă când parametrul efectiv este un pointer. Pointerii pot fi transferaţi la fel ca orice altă valoare. În foarte multe situaţii parametrul efectiv este numele unui tablou.

Funcţii recursive

Limbajul C acceptă utilizarea funcţiilor recursive. Funcţiile recursive sunt funcţiile care se autoapelează. Nu este necesară o sintaxă specială pentru a indica faptul că o funcţie este recursivă.

În cazul apelului recursiv al unei funcţii, aceasta este reapelată înainte de a se reveni din ea. La fiecare reapel parametrii şi variabilele locale se alocă pe stivă într-o zonă independentă.

Ele au valori distincte la fiecare reapelare. După execuţia fiecărui apel recursiv şi returnarea unei valori, variabilele locale şi parametrii vechi sunt eliminaţi din stivă şi execuţia programului continuă de la punctul interior funcţiei de unde a fost apelată. Astfel orice reapel al unui apel recursiv va conduce la curăţirea stivei şi la o revenire din funcţie, parametrii şi variabilele locale vor avea valorile dinaintea reapelului respectiv. Variabilele statice îşi păstrează însă valoarea la un reapel.
Observaţie:
In cazul oricărei funcţii recursive trebuie să existe o instrucţiune if pentru a determina funcţia să revină fără a executa apelul recursiv. În caz contrar, funcţia nu va reveni niciodată în urma unui apel. Pentru a nu comite astfel de erori este de recomandat utilizarea unor instrucţiuni printf( ) şi getchar( ) în timpul scrierii funcţiilor pentru a controla ce se întâmplă şi a opri execuţia în caz de eroare.

Recursivitatea nu conduce la economie de memorie şi nici la o execuţie mai rapidă a programelor decât echivalentele lor iterative. Ea oferă însă avantajul că permite o descriere mai compactă şi mai clară a funcţiilor care exprimă procese de calcul recursive.

Varianta recursivă se poate folosi cu succes pentru a crea versiuni mai clare şi mai simple ale unor algoritmi (de exemplu sortarea rapidă, algoritmi de inteligenţă artificială etc).

Exemple

1.

```
#include <stdio.h>
#include <conio.h>
double patrat(double);
void main(void)
{
double a,b;
printf("Introduceti valoarea lui a:");
scanf("%lf",&a);
printf("a=  %lf\na^2=%lf\n",a, patrat(a));
b=patrat(a)-3*a+2;
printf("b=   %lf\n",b);
printf("Apasati orice tasta pentru continuare\n");
getch();
}
double patrat(double x) /*functia ridicare la patrat*/
{
return x*x;
}
```

2. Să se calculeze valoarea combinărilor $C_n^k = \frac{n!}{k!(n-k)!}$ utilizând o funcţie pentru calculul factorialului. Programul va asigura citirea repetată a datelor de intrare n şi k.

```
#include<stdio.h>
#include<conio.h>
double fact(int);
```

```
void main(void)
{
int n,k;
double combin;
char var;
do{
        printf("\nIntroduceti numarul n:");
        scanf("%d", &n);
        while(n<0)
        {
                printf("Dimensiune eronata:\n");
                printf("Introduceti alt n:");
                scanf("%d", &n);
        }
        printf("Introduceti numarul k<n:");
        scanf("%d", &k);
        while(n<k)
        {
                printf("Dimensiune eronata:\n");
                printf("Introduceti alt k<n:");
                scanf("%d", &k);
        }
        combin=fact(n)/fact(k)/fact(n-k);

        printf("Combinari de %d luate cate d=%lf\n", n,k, combin);
        printf("Doriti sa continuati introducerea datelor? (D/N)");
}
while((var=getch())=='D' ||var=='d');
printf("\nProgramul s-a incheiat\n");
getch();
}
double fact(int n) /*calculeaza pe n!*/
{
double f;
int i;
 if(n==0)
          return 1.0;
 else
        for( i=2, f=1.0; i <= n; i++)
                f *= i;
 return f;
}
```

3. Să se scrie un program pentru calculul funcţiei sin(x) în intervalul [0,360] parcurs cu pasul h. Pentru calculul lui sin(x) se va utiliza următoarea dezvoltare în serie de puteri:

$$\sin(x) = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots + (-1)^i \frac{x^{2i+1}}{(2i+1)!} + \ldots$$

unde argumentul x este exprimat în radiani.

Din dezvoltarea în serie se vor considera toţi acei termeni care sunt mai mari în valoare absolută decât o eroare impusă ε. Să se tipărească valoarea argumentului x, valoarea funcţiei calculate prin dezvoltarea în serie precum si eroarea faţă de rezultatul obţinut utilizând funcţia de bibliotecă sin(x) din fişierul antet math.h.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#define PI 3.14159265
#define EPS 1e-10
double sinus( double);
void main(void)
{
int i,h;
double x;
printf("Introduceti pasul de variatie al argumentului x:");
scanf("%d",&h);
x=PI/180.;
printf("Argument x       sinus(x)       eroare\n");
for(i=0;i<=360;i=i+h)
printf("%10d %15.10lf %15.10lf\n",i,sinus(x*i),sinus(x*i)-sin(x*i));
printf("Apasati orice tasta pentru continuare\n");
getch();
}

double sinus(double x) /*functia de calcul al lui sin(x)*/
{
double s,f,t;
int n, semn,y;
if(x>0)
  semn=1;
else
  {
  semn=-1;
  x=-x;
  }
y=x/(2*PI);
x=x-y*2*PI;
if(x>=1.5*PI)
  {
  x=2*PI-x;
  semn=-semn;
  }
else
 if(x>=PI)
 {
    x=x-PI;
    semn=-semn;
 }
 else
```

```
   if(x>PI/2)
     x=PI-x;
s=x;
n=3;
f=-x*x;
t=x*f/6.0;
while(fabs(t)>=EPS){
          s=s+t;
          t=t*f;
          n++;
          t=t/n;
          n++;
          t=t/n;
                              }
return (s*semn);
}
```

4. Să se scrie un program care schimbă între ele valorile a două variabile var1 şi var2.

```
#include <stdio.h>
#include <conio.h>
void schimba(double *a,double *b);
void main(void)
{
double var1,var2;
printf("Introduceti valoarea lui var1:");
scanf("%lf",&var1);
printf("Introduceti valoarea lui var2:");
scanf("%lf",&var2);
printf("Valorile initiale:  %lf\t%lf\n",var1,var2);
schimba(&var1,&var2);
printf("Valorile schimbate: %lf\t%lf\n",var1,var2);
printf("Apasati orice tasta pentru continuare\n");
getch();
}

void schimba(double *a,double *b) /*functia schimbare a doua valori*/
{
int aux;
aux=*a;
*a=*b;
*b=aux;
}
```

5. Să se scrie un program care calculează produsul elementelor unui şir de numere întregi. Se va utiliza o funcţie pentru calculul produsului.

```
#include <stdio.h>
#include <conio.h>
double produs(int sir[20],int n);
```

```
void main(void)
{
int n,i;
int a[20];
printf("Valoarea lui n=");
scanf("%d",&n);
while(n<=0||n>20){
  printf("Dimensiune eronata:%d\n",n);
  printf("Introduceti alt n:");
  scanf("%d",&n);
                    }
for(i=0;i<n;i++){
        printf("Elementul a[%d]=",i);
        scanf("%d",&a[i]);
                  }
printf("Valoarea produsului: %lf\n",produs(a,n));
printf("Apasati orice tasta pentru continuare\n");
getch();
}

double produs(int sir[20], int n) /*functia produsul elementelor unui sir*/
{
double p;
for(p=1;n>0;n--)
        p*=sir[n-1];
return p;
}
```

6. Să se scrie un program care calculează, citeşte un număr întreg din fişierul de intrare, apoi calculează şi tipăreşte “n!”. Să se folosească o funcţie recursivă pentru calculul lui n!.

```
#include <stdio.h>
#include <conio.h>
double factorial(int);
void main(void)
{
int n,i;
double fact;
printf("Introduceti valoarea lui n:");
scanf("%d",&n);
printf("n=%d    n!=%g\n",n,factorial(n));
printf("Apasati orice tasta pentru continuare\n");
getch();
}
double factorial(int n)
{
double f;
if(n==1)
        return 1.;
else
```

```
        f=factorial(n-1)*n;
return f;
}
```

7. Să se scrie un program pentru calculul sumei unui şir de numere reale folosind o funcţie recursivă.

```
#include <stdio.h>
#include <conio.h>
double sumasir(int sir[],int n);
void main(void)
{
int n,i;
int a[20];
printf("Valoarea lui n=");
scanf("%d",&n);
while(n<=0||n>20){
  printf("Dimensiune eronata:%d\n",n);
  printf("Introduceti alt n:");
  scanf("%d",&n);
                        }
for(i=0;i<n;i++){
        printf("Elementul a[%d]=",i);
        scanf("%d",&a[i]);
                    }
printf("Valoarea sumei: %lf\n",sumasir(a,n));
printf("Apasati orice tasta pentru continuare\n");
getch();
}
double sumasir(int sir[],int n)
{
double s;
if(n==0)
        return 0.;
else
        s=sir[n-1]+sumasir(sir,n-1);
return s;
}
```

8. Să se scrie un program care calculează valoarea funcţiei f într-un interval dat [a,b] parcurs cu pasul p. Funcţia este definită de expresia:

$$f(x)=\begin{cases}\dfrac{x^2-5}{x+7} & x<-2\\ 3x^3-5x^2+2 & -2\le x<4\\ \dfrac{x+8}{(x-10)(x^2-7x+6)} & x\ge 4\end{cases}$$

Să se tipărească valoarea argumentului x şi valoarea funcţiei dacă funcţia se poate calcula sau numai valoarea argumentului dacă funcţia nu se poate calcula.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int test=1; /*declaratie ca variabila globala*/
double functie(double);
void main(void)
{
double x,a,b,p,f;
printf("Introduceti valoarea lui a:");
scanf("%lf",&a);
printf("Introduceti valoarea lui b:");
scanf("%lf",&b);
printf("Introduceti valoarea pasului p:");
scanf("%lf",&p);
for(x=a;x<=b;x+=p){
f=functie(x);
if(test==1)
        printf("x=%lf\tf(x)=%lf\n",x,f);
else    {
        printf("Functia nu se poate calcula in x=%lf\n",x);
        test=1;}
                }
printf("Apasati orice tasta pentru continuare\n");
getch();
}
double functie(double x) /*functia cu acolada*/
{
double f;
if(x<-2){
        if(x!=-7){
                f=(pow(x,2.)-5)/(x+7);
                return f;}
        else
                test=0;
                return 0;}
else
        if(x<4){
                f=3*pow(x,3.)-5*pow(x,2)+2;
                return f;}
else
        if(x>=4){
                if(x!=6&&x!=10){
                        f=(x+8)/(x-10)/(pow(x,2.)-7*x+6);
                        return f;}
                else
                        test=0;
                return 0;}
}
```