Laboratorul 10. FUNCȚII ÎN C/C++

Introducere

În limbajul C++ (similar cu limbajul C standard) programul este o colecție de module distincte numite funcții, structura generală a programului fiind:

<directive preprocesor>
<declarații globale>
funcții

Un program C++ conține obligatoriu o funcție numită main() şi aceasta este apelată la lansarea în execuție a programului.

Programul sursă poate fi partiționat în mai multe fişiere grupate într-un proiect (se utilizează meniul Project din mediul de programare utilizat). Fiecare fişier conține declarații globale şi un set de funcții, dar numai unul conține funcția main().

În C++, ca şi în C, se utilizează declarații (prototipuri) şi definiții de funcții.

De regulă, se declară prototipurile de funcții în zona declarațiilor globale, urmând ca apoi să se facă definițiile funcțiilor. Definițiile se pot regăsi în diferite fişiere sursă.

Apelul unei funcții produce lansarea în execuție a funcției respective şi se poate face doar dacă, cel puțin prototipul funcției a fost deja stabilit.

O funcție reprezintă un bloc de instrucțiuni care este desemnat printr-un nume, care poate prelua date de intrare prin parametrii şi poate returna un rezultat.

10.1. Definiții de funcții

Sintaxa definiției unei funcții este:

```
tip_rez nume_funcție (<lista_parametri>)
{
        <declarații locale>
        secvență de instrucțiuni
}
```

unde:

- **tip_rez** - este un tip oarecare de dată şi reprezintă tipul rezultatului returnat de funcție. Dacă nu este specificat, implicit tipul rezultatului returnat este int. Pentru funcțiile care nu returnează rezultat trebuie să se specifice tipul void.
- **nume_functie -** este un identificator.
- **lista_parametri** - reprezintă enumerarea declarațiilor parametrilor sub forma:

tip_parametru nume_parametru, <tip_parametru nume_parametru>

Tipul parametrului poate fi orice tip valid de date.

Nu este admisă definirea unei funcții în blocul altei funcții şi nu sunt permise salturi cu instrucțiunea goto în afara funcției.

Apelul funcției constă din numele funcției urmat de lista de constante, variabile sau expresii asociate parametrilor încadrată între paranteze ().

Atât la definirea, cât şi la apelul funcţiilor, parantezele () urmează întotdeauna numele funcţiilor, chiar dacă acestea nu au parametri.

Se foloseşte denumirea de **parametri formali** pentru identificatorii din lista de argumente din definiţia funcţiei şi **parametri efectivi** (constantele, variabilele, expresiile) din lista unui apel al funcţiei.

Parametrii formali reprezintă variabile locale care au domeniu funcţia şi timpul de viaţă corespunde duratei de execuţie a funcţiei, deci valorile lor se memorează în stivă sau în registrele procesorului.

10.2. Declaraţii de funcţii. Prototipuri.

Apelul unei funcţii nu se poate face înainte de definiţia ei. Sunt situaţii în care nu este posibil acest lucru (în cazul funcţiilor care se apelează unele pe altele, sau al funcţiilor definite în alte fişiere sursă).

Pentru a oferi compilatorului posibilitatea de a verifica corectitudinea apelului unei funcţii (numărul şi tipurile parametrilor, tipul rezultatului, eventualele conversii care apar), se pot folosi declaraţii fără definire, numite prototipuri. Sintaxa generală a unui prototip este:

<tip> nume_functie (<lista_parametri>);

Lipsa lista_parametri este interpretată în C++ ca funcţie fără parametri, deci cuvântul void nu este necesar, spre deosebire de C standard care nu consideră funcţia fără parametri, ci cu orice listă de parametri şi, ca urmare, nu verifică parametrii efectivi la apelare.

În declaraţia funcţiei este suficient ca în lista_parametri să se specifice tipurile parametrilor, fără identificatorii lor. Dacă însă se specifică identificatorii, aceştia trebuie să fie identici cu cei folosiţi în antetul definiţiei funcţiei.

Prototipul funcţiei trebuie să fie plasat înainte de orice apel al funcţiei.

10.3. Transferul parametrilor

10.3.1. Transferul prin valoare. Conversii de tip.

La apelul unei funcţii, valoarea parametrilor efectivi este încărcată în zona de memorie corespunzătoare parametrilor formali. Acest procedeu se numeşte transfer prin valoare. Dacă parametrul efectiv este o variabilă, ea nu este afectată de nici o operaţie asupra parametrului formal, ceea ce poate constitui o protecţie utilă.

Transferul de valoare este însoţit de eventuale conversii de tip realizate de compilator, implicite sau explicite dacă se foloseşte operatorul cast. Conversia se realizează de la tipul parametrului efectiv către tipul parametrului formal.

Se foloseşte ca exemplu definirea şi apelul unei funcţii care preia două variabile şi realizează inversarea valorilor acestora între ele.

```
#include <stdio.h>

void schimba(int , int );                    // lista de parametri este formată din variabile int

void main()
```

```
{   int i, j;
    float p, q;
    scanf(”%d%d”, &i, &j);
    schimba(i, j);                  // apelul funcției include ca parametri efectivi variabilele i, j de tip int,
                                    // deci nu este necesară conversia lor; pe parcursul execuției funcției
                                    // schimba() se alocă variabilele locale a, respectiv b, care sunt
                                    // inițializate cu valorile i, respectiv j. Inversarea valorilor se produce
                                    // asupra variabilelor locale a şi b, variabilele i şi j ne fiind afectate
     printf(”\ni=%d, j=%d”, i, j);
    scanf(“%f%f”, &p, &q);
    schimba(p, q);                  // apelul funcției include ca parametri efectivi variabile de tip float,
                                    // deci va avea loc o conversie degradantă, implicită, float->int,
                                    // care însă nu afectează variabilele p şi q, ci conversia se face la
                                    // inițializarea variabilelor a şi b; la finalul execuției funcției se
                                    // constată că p şi q rămân nemodificate
    printf(”\ni=%f, j=%f”, p, q);
}

void schimba(int a, int b)
{   int temp ;                      // inversarea valorilor a două variabile se face prin
    temp=a;                         // atribuiri succesive, folosind o variabilă auxiliară, temp
    a=b;
    b=temp;
}
```

10.3.2. Transferul prin adresă

Pentru ca o funcție să poată modifica valoarea unei variabile folosită ca parametru efectiv, trebuie folosit un parametru formal de tip pointer, iar la apelul funcției să se folosească ca parametru efectiv adresa variabilei.

Se foloseşte pentru exemplificare tot definirea şi apelul unei funcții care preia două variabile şi realizează inversarea valorilor acestora, dar parametrii funcției vor fi pointeri.

```
#include <stdio.h>

void schimba(int *, int *);     // lista de parametri este formată din pointeri la int

void main()
{   int i, j;
    scanf(”%d%d”, &i, &j);
    schimba(&i, &j);            // la apelul funcției, parametrii efectivi sunt adrese de variabile
    printf(”\ni=%d, j=%d”, i, j);
}

void schimba(int *a, int *b)
{   int temp ;
    temp=*a;
    *a=*b;
    *b=temp;
}
```

Parametri tablou constituie o excepție de la transferul parametrilor prin valoare, deoarece funcția primeşte ca parametru adresa tabloului. Acest lucru este motivat de faptul că, în general,

tablourile conţin o cantitate mare de date al căror transfer prin valoare ar avea ca efect o scădere a vitezei de execuţie şi creştere a memoriei necesare prin copierea valorilor într-o variabilă locală. De altfel, numele tabloului este echivalent cu adresa sa. În prototip şi antetul funcţiei, parametrul tablou se specifică în lista de parametri sub forma:

tip nume_tablou[ ]

sau

tip *nume_tablou

10.3.3. Transferul prin variabile referinţă

Folosirea parametrilor formali variabile referinţă este similară folosirii parametrilor formali pointeri, asigurând posibilitatea de a modifica valorile parametrilor efectivi.

Utilizarea referinţelor prezintă avantajul că procesul de transfer prin referinţă este efectuat de compilator în mod transparent, scrierea funcţiei şi apelul ei fiind simplificate.

Dacă tipul parametrului efectiv nu coincide cu cel al parametrului formal referinţă, compilatorul efectuează o conversie, ca în cazul transferului prin valoare. Pentru realizarea conversiei se creează un obiect temporar de dimensiunea tipului referinţă, în care se înscrie valoarea convertită a parametrului efectiv, parametrul formal referinţă fiind asociat obiectului temporar. Se pierd astfel avantajele folosirii referinţelor ca parametri formali.

```
#include <stdio.h>

void schimba(int &, int &);              // lista de parametri este formată din referinţe de variabile int

void main()
{   int i, j;
   float p, q;
   scanf("%d%d", &i, &j);
   schimba(i, j);                  // apelul funcţiei include ca parametri efectivi variabilele i, j de tip int,
                                   // deci nu este necesară conversia lor; pe parcursul execuţiei funcţiei
                                   // schimba() ele sunt redenumite, a, respectiv b, inversarea valorilor
                                   // având efect asupra variabilelor din apel, i şi j
   printf("\ni=%d, j=%d", i, j);
   scanf("%f%f", &p, &q);
   schimba(p, q);                  // apelul funcţiei include ca parametri efectivi variabile de tip float,
                                   // deci va avea loc o conversie degradantă, implicită, float->int, care
                                   // are ca urmare crearea de variabile temporare; referinţele care se
                                   // creează sunt pentru aceste variabile temporare, şi nu pentru p şi q;
                                   // inversarea valorilor are efect asupra variabilelor temporare, astfel
                                   // că,la ieşirea din funcţie, se va constata faptul că valorile variabilelor
                                   // p şi q au rămas nemodificate
    printf("\ni=%f, j=%f", p, q);
}

void schimba(int &a, int &b)
{   int temp ;
   temp=a;
   a=b;
   b=temp;
}
```

Este utilă folosirea parametrilor formali referință şi în situația în care parametrul are dimensiune mare şi crearea în stivă a unei copii a valorii are ca efect reducerea vitezei de execuție şi creşterea semnificativă a stivei. În această situație, dacă nu se doreşte modificarea parametrului efectiv, acesta poate fi protejat prin folosirea modificatorului const la declararea parametrului formal referință.

10.4. Rezultatul unei funcții. Instrucțiunea return.

Instrucțiunea **return** determină încheierea execuției unei funcții şi revenirea în funcția apelantă.

Valoarea expresiei reprezintă rezultatul întors de funcție, deci trebuie să fie compatibil cu tipul indicat în prototip şi definiție.

Dacă tipul funcției este void, instrucțiunea return este necesară doar dacă se doreşte revenirea din funcție înainte de execuția întregii secvențe de instrucțiuni care alcătuieşte funcția.

Transferul rezultatului se poate face utilizând toate cele trei metode: prin valoare, pointer sau referință.

În cazul transferului prin pointer sau referință, obiectul a cărui adresă se întoarce nu trebuie să fie un obiect automatic, deoarece acesta dispare odată cu terminarea execuției funcției. În cazul returnării prin referință, o asemenea situație e semnalată de compilator ca eroare, în schimb, în cazul pointerilor nu este semnalată eroarea. Adresa de memorie returnată va corespunde unei zone de memorie care nu mai este alocată şi pot apărea efecte neaşteptate.

10.5. Funcții recursive

O funcție este numită funcție recursivă dacă ea se autoapelează, fie direct (în definiția ei se face apel la ea însăşi), fie indirect (prin apelul altor funcții). Limbajele C/C++ dispun de mecanisme speciale care permit suspendarea execuției unei funcții, salvarea datelor şi reactivarea execuției la momentul potrivit. Pentru fiecare apel al funcției, parametrii şi variabilele automatice se memorează pe stivă, având valori distincte. Variabilele statice ocupă tot timpul aceeaşi zonă de memorie (figurează într-un singur exemplar) şi îşi păstrează valoarea de la un apel la altul. Orice apel al unei funcții conduce la o revenire în funcția respectivă, în punctul următor instrucțiunii de apel. La revenirea dintr-o funcție, stiva este curățată (stiva revine la starea dinaintea apelului).

Un exemplu de funcție recursivă este funcția de calcul a factorialului (n!), definită astfel:

fact(n)=1, dacă n=0;

fact(n)=n*fact(n-1), dacă n>0;

Este prezentat în continuare programul care implementează această funcție.

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

double fact(int n)
{
  if (n<0)                  // valorile negative sunt invalide
```

```
  {       printf("\nParametru negativ !");
          exit(2);
  }

  if (n==0)
    return 1;
  else
    return n*fact(n-1);
}

void main()
{
  int nr;
  double f;
  printf("\nIntrodu un numar: ");
  scanf("%d",&nr);
  f=fact(nr);
  printf("\n %d ! = %.0lf",nr,f);
  getch();
}
```

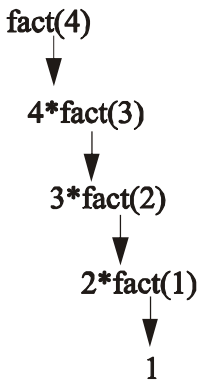

Fig. 10.1. Exemplificarea apelului funcțiilor recursive

Se observă că în corpul funcției fact se apelează însăşi funcția fact.

Spre exemplu, dacă se doreşte calcularea valorii 4!, se apelează funcția: fact(4). În interiorul său se calculează expresia 4*fact(3). Apelul funcției fact(3) va calcula la rândul său 3*fact(2), fact(2) calculează 2*fact(1) şi în fine, fact(1) calculează 1*fact(0). Valoarea 0 a parametrului funcției are ca efect întreruperea recursivității.

O funcție recursivă poate fi realizată şi iterativ. Modul de implementare trebuie ales în funcție de problemă. Deşi implementarea recursivă a unui algoritm permite o descriere clară şi compactă, recursivitatea nu conduce la economie de memorie şi nici la execuția mai rapidă a programelor. În general, se recomandă utilizarea funcțiilor recursive în anumite tehnici de programare, cum ar fi unele metode de căutare (backtracking).

Exemple

```
/*************************************************************************
```

Exemplul 10.1. Să se întocmească un program în care se definesc următoarele funcții:

- o functie de citire a elementelor unui tablou de întregi;
- o functie de afisare a elementelor unui tablou de întregi;
- o functie care returneaza maximul dintre elementele unui tablou de întregi;
- o functie de ordonare a elementelor unui tablou de întregi.

In functia main se declara un tablou de întregi si se exemplifica folosirea funcțiilor definite.

Tablourile au dimensiunea stabilită cu ajutorul unei constante definită cu directiva #define.

```
*************************************************************************/
#include<stdio.h>
#include<conio.h>
```

```
#define N 5                     // constanta folosita pentru a stabili dimensiunea tablourilor

void citire(int t[]);
void afisare(int t[]);
int maxim(int t[]);
void ordonare(int t[]);

void main()
{       int tab[N];
        citire(tab);
        printf("\nSe afiseaza valorile elementelor de tablou");
        afisare(tab);
        printf("\nValoarea maxima este: %d", maxim(tab));
        ordonare(tab);
        printf("\nSe afiseaza valorile elementelor de tablou");
        afisare(tab);
        getch();
}
void citire(int t[])
{       int i;
        for (i=0;i<N;i++)
        {       printf("\nelementul [%d]=", i+1);
                scanf("%d",&t[i]);
        }
}
void afisare(int t[])
{       int i;
        for (i=0;i<N;i++)
                printf("\nelementul [%d]= %d", i+1, t[i]);
}
int maxim(int t[])
{       int i, max=t[0];
        for (i=1;i<N;i++)
        if(max<t[i])
                max=t[i];
        return max;
}
void ordonare(int t[])                  // se foloseşte algoritmul "bubble sort"
{       int i, k, aux;
        do
        {
                k=0;
                for(i=0;i<N-1;i++)
                        if(t[i]>t[i+1])
                        {
                                aux=t[i];
                                t[i]=t[i+1];
                                t[i+1]=aux;
                                k=1;
                        }
        }
        while(k!=0);
}
```

Întrebări. Exerciții. Probleme.

1. Sa se scrie un program in care se defineste o functie care preia ca parametru un întreg. Functia returneaza 1 daca numarul este prim, sau 0, daca numarul nu este prim.

int prim(unsigned);

In functia main() se citesc doua valori pentru a şi b care reprezintă capetele unui interval [a , b]. Să se afişeze toate valorile prime din intervalul dat.

2. Să se întocmească un program în care se definesc următoarele funcții:

a. O funcție care copiază conținutul unui şir de caractere în alt şir;
b. O funcție care compară două şiruri de caractere;
c. O funcție care concatenează două şiruri.

În funcția main() se exemplifică folosirea acestor funcții.

Notă:

Nu se folosesc funcțiile din fişierul string.h !

3. Sa se rescrie programul din exemplul 10.1., funcțiile fiind definite pentru tablouri alocate dinamic, dimensiunea acestora fiind citită de la tastatură la execuția programului.