SUBPROGRAME C++

1. Introducere

Subprogramele sunt părţi ale unui program, identificabile prin nume, care se pot activa la cerere prin intermediul acestor nume.

Prezenţa subprogramelor implică funcţionarea în strânsă legătură a două noţiuni: **definiţia** unui subprogram şi **apelul** unui subprogram.

Definiţia unui subprogram reprezintă de fapt descrierea unui proces de calcul cu ajutorul variabilelor virtuale (parametri formali) iar **apelul** unui subprogram nu este altceva decât execuţia procesului de calcul pentru cazuri concrete (cu ajutorul parametrilor reali, (efectivi, actuali)).

2. Structura unui subprogram C++

Un subprogram (funcţie) are *o **definiţie*** şi atâtea ***apeluri*** câte sunt necesare.

Definiţia unei funcţii are forma generală:

```
tip_returnat nume_funcţie (lista parametrilor formali)
      {
            instrucţiune; // corpul funcţiei
      }
```

Tip_returnat	Reprezintă tipul rezultatului calculat şi returnat de funcţie şi poate fi: int, char, char*, long, float, void, etc. În cazul în care tipul rezultatului este **diferit de void**, corpul funcţiei trebuie să conţină cel puţin o instrucţiune **return**. Înstrucţiunea return va specifica valoarea calculată şi returnată de funcţie care trebuie să fie de acelaşi tip ca şi **tip_returnat.**
Nume_funcţie	Reprezintă numele dat funcţiei de către cel ce o defineşte, pentru a o putea apela.
Lista_parametrilor_formali	Reprezintă o listă de declaraţii de variabile separate prin virgulă. Această listă poate să fie şi vidă.
Instrucţiune	Este o instrucţiune vidă sau o instrucţiune simplă sau o instrucţiune compusă.

3. Apelul unei funcţii . Revenirea dintr-o funcţie

3.1 Apelul **unei** funcţii care nu returnează o valoare are forma generală:

```
nume_funcţie (lista parametrilor efectivi);
```

parametru efectiv = parametru actual = parametru real = parametru de apel

lista parametrilor efectivi = fie vidă, fie o expresie sau mai multe despărţite prin virgulă

O funcţie care returnează o valoare poate fi apelată fie printr-o instrucţiune de apel simplă (cazul funcţiilor care nu returnează valori) şi în plus poate fi apelată ca operand al unei expresii. În cazul în care funcţia se apelază print-o instrucţiune de apel simplă, rezultatul funcţiei se pierde. Când funcţia se apelează ca operand, valoarea returnată va fi utilizată în expresie.

La apelul unei funcţii, valorile parametrilor efectivi se atribuie parametrilor formali corespunzători. În cazul în care unul din tipul unui paramatru efectiv diferă de tipul parametrului formal corespunzător, parametrul efectiv va fi convertit spre parametru formal (dacă este posibil, altfel compilatorul generează eroare).

În momentul în care se întâlneşte un apel de funcţie, controlul execuţiei programul este transferat primei instrucţiuni din funcţie, urmând a se executa secvenţial instrucţiunile funcţiei.

Revenirea dintr-o funcţie se face în unul din următoarele cazuri:

- după execuţia ultimei instrucţiuni din corpul funcţiei
- la întâlnirea unei instrucţiuni **return**

Instrucţiunea return are formatele:

- **return ;**
- **return expresie ;**

Exemplul 3.1	Exemplul 3.2

`# include <iostream.h>`
 	
`void f1 ()`
`{`
`  cout << "abc";`
`}`
 	
`void main ()`
`{`
`  clrscr();`
`  f1();`
`}`
	`# include <iostream.h>`
	
	`void f1 (int k)`
	`{`
	`  for (int i=1; i<=k ; i++)`
	`  cout << "abc"<< " ";`
	`}`
	`void main ()`
	`{`
	`  clrscr();`
	`  f1(3);`
	`}`

Se va afisa: Abc	Se va afişa: abc abc abc
Funcţia nu returnează o valoare Funcţia nu are parametri Apelul funcţiei este o instrucţiune de apel simplă	Funcţia nu returnează o valoare Funcţia are un parametru formal de tip int Apelul funcţiei este o instrucţiune de apel sinplă şi se face cu ajutorul unui parametru actual care este de acelaşi tip cu tipul parametrului formal corespunzător

Exemplul 3.3	Exemplul 3.4

`# include <iostream.h>`
`# include <math.h>`
 	
`int prim (int x)`
`{`
`int nr_div=0;`
`for (int i=2; i<=sqrt(x); i++)`
`if (x%i==0)`
`nr_div++;`
`if (nr_div==0)`
`return 1;`
`else`
`return 0;`
`}`
 	
`void main ()`
`{`
`int N;`
`cout << "N="; cin >> N;`
 	
`if (prim(N))`
`cout << "PRIM";`
`else`
`cout << "NU E PRIM";`
`}`
	`# include <iostream.h>`
	`# include <math.h>`
	
	`int prim (int x)`
	`{`
	`for (int i=2; i<=sqrt(x); i++)`
	`if (x%i==0)`
	`return 0;`
	`return(1);`
	`}`
	
	`void main ()`
	`{`
	`int N;`
	`cout << "N="; cin >> N;`
	
	`if (prim(N))`
	`cout << "PRIM";`
	`else`
	`cout << "NU E PRIM";`
	
	`}`

Funcţia returnează o valoare de tip int Funcţia are un parametru de tip int Rezultatul funcţiei este este utilizat în cadrul unei expresii.	În cazul în care se întâlneşte un divizor a lui x se execută instrucţiunea return 0. Astfel apelul funcţiei se încheie. Dacă x este număr prim, instrucţiunea return 0 nu se execută niciodată şi în acest caz, după terminarea execuţiei instrucţiunii for, se execută instrucţiunea return 1 (care determină încheierea execuţiei funcţiei).

În cazul în care **tipul returnat de funcţie lipseşte** din definiţia funcţiei, acesta este implicit **int** şi nu void.

Exemplul 3.5	Exemplul 3.6	Exemplul 3.7

`P( )`
`{`
`cout << " un int";`
`}`
 	
`void main ()`
`{`
`clrscr();`
`cout << p();`
`}`
	`p( )`
	`{`
	`return 25;`
	`}`
	
	`void main ()`
	`{`
	`clrscr();`
	`cout << p();`
	`}`
		`void p( )`
		`{`
		`cout << "void";`
		`}`
		
		`void main ()`
		`{`
		`clrscr();`
		`cout << p();`
		`}`

Compilatorul generează WARNING Se va afişa un număr întreg	Se afişează 25	Compilatorul generează eroare

4. Prototipul unei funcţii

Pentru a apela o funcţie, aceasta trebui mai întâi definită. Astfel apelul unei funcţii trebuie precedat de definiţia funcţiei respective.

O a doua posibilitate de apelare a funcţiei constă în scrierea prototipului funcţiei înainte ca acesta să fie apelată.

Prototipul funcţiei conţine informaţii asemănătoare cu cele din antetul funcţiei. Pot lipsi numele parametrilor formali (contează doar tipul şi ordinea acestora), în plus acesa este urmat de ";".

Exemplul 4.1.

```
# include <iostream.h>

int max (int, int);                    PROTOTIPUL FUNŢIEI
void main ()
{
   cout << max(10, 20);
}                                      APELUL FUNCŢIEI
int max (int a, int b)
{
   if (a>b)
         return a;                     DEFINIŢIA FUNCŢIEI
   else                                antetul funcţiei
         return b;                     si
}                                      - corpul functie
```

5. Variabile locale şi variabile globale

5.1. Funcţia main.

În C, numele de funţie main determină prima instrucţiune pe care o va executa programul. Acesta este unica diferenţă dintre main şi celelalte funcţii. Din acest motiv se poate spune că "orice se poate face în main se poate face şi în celelalte funcţii".

5.2. Variabile locale

La fel cum se declară variabilele în cadrul funcţiei main, la fel se pot declara varibile în cadrul celorlalte funcţii. Aceste variabile se numesc locale şi sunt accesibile doar de funcţia care le-a declarat. La fel în cadrul unei funcţii se pot apela şi alte funcţii, ca şi în main, dacă acestea au fost definite înaintea eventualului apel sau dacă este prezent un prototip de funcţie înaintea funcţiei apelante şi o definiţie de funcţie în cadrul programului respectiv sau în fişierele incluse în programului respectiv.

5.3. Variabile globale

Variabilele globale sunt declarate înafara oricărei funcţii şi pot sunt vizibile (pot fi utilizate) în tot programul (în programul principal şi în subprograme) din momentul declarării lor.

Exemplul 5.1

```
# include <iostream.h>
int N;
void f1()
{
   int x=5;
   N=10;
   cout << endl<<N;
   cout << endl << x;
}
```

Exemplul 5.2

```
# include <iostream.h>
int N;
void f1()
{
   int x=5;
   cout << endl << x;
   P=2 //eroare
}
int P=9;
```

Exemplul 5.3

```
# include
<iostream.h>
int N;
void f1(int p)
{
   int x=p;
   cout << x;
}
void main ()
```

`void main ()`
`{`
`N=4;`
`cout << N;`
`f1();`
`}`
	`void main ()`
	`{`
	`f1();`
	`cout << x;`
	`P=7;//corect`
	`}`
		`{`
		`f1(3);`
		`}`

N este variabilă globală. Poate fi accesată în cadrul oricărei funcţii. x este variabilă locală, vizibilă doar în cadrul funcţiei f1() Se va afişa: 4 10 5	Compilatorul generează eroare deoarece funcţia main încearcă să acceseze variabila x care este vizibilă doar în funcţia f1(). Compilatorul generează eroare deoarece P este accesată în f1() înainte de a fi declarată.	Se afişează 3 N este variabilă globală. Poate fi accesată în cadrul oricărei funcţii. x este variabilă locală. Poate fi accesată doar în cadrul funcţiei f1() p este parametru formal. Poate fi accesat doar în f1().

5.4. Regula de omonimie

În cazul în care există o variabilă locală care are acelaşi nume cu o variabilă globală, aceste două variabile se numesc **variabile omonime**.

Variabilele locale sunt prioritare (ascund) variabilele globale omonime.

Exemplul 5.4	

`Int N=10;`
`Void f1()`
`{`
`int N=2;`
`cout << N;`
`}`
`void main ()`
`{`
`f1();`
`cout << N;`
`}`
	Variabila N este definită atât ca variabilă globală cât şi ca variabilă locală în f1().
	Se va afisa:
	2 10
	Funcţia f1() acţionează asupra variabilei locale N.
	Funcţia main() acţionează supra variabilei globale N.

Întrebare. Cum gestionează compilatorul cele două variabile omonime ?

Răspuns:

Variabilelor **globale** li se rezervă spaţiu de memorare la începutul execuţiei programului, într-o zonă de memorie numită "zonă de date". Acest spaţiu va fi ocupat până la încheierea execuţiei programului.

Variabilelor **locale** li se rezervă spaţiu într-o zonă specială de memorie numită "stiva". La încheierea execuţiei subprogramului, conţinutul stivei este eliberat. Din acest motiv, variabilele globale sun vizibile doar în interiorul subprogramului în care au fost declarate.

6. Parametri formali şi parametri actuali

Parametri formali apar în antetul subprogramului şi sunt utilizaţi de subprogram pentru descrierea abstractă a unui proces de calcul .

Parametri actuali apar în instrucţiunea de apelare a uni subprogram şi sunt folosiţi la execuţia unui proces de calcul pentru valori concrete.

Parametrii formali nu sunt variabile. O variabilă este caracterizată de nume, tip, şi adresă. Legarea unui parametru formal la o **adresă** se realizează în timpul execuţiei instrucţiunii de apelare a subprogramului.

7. Apel prin valoare şi apel prin referinţă

Există două tipuri de apel al subprogramelor:

A. Apel prin valoare

B. Apel prin referinţă

7.1. Apel prin valoare – se transmite o copie a parametrului actual.

Valorile transmise la apelul unui subprogram sunt memorate în stivă. Datorită faptului că, după terminarea execuţiei unui subprogram, stiva este eliberată, în cazul apelului prin valoare parametrul actual **nu se modifică (se operează asupra unei copii a parametrului efectiv)**

7.2. Apel prin referinţă - se transmite adresa parametrului actual.

În cazul apelului prin referinţă, subprogramul, cunoscând adresa parametrului actual, **acţionează direct asupra locaţiei de memorie** indicată de aceasta, **modificând** valoarea parametrului actual.

În C, implicit apelul este prin valoare. Pentru a specifica un apel prin referinţă, în lista parametrilor formali, numele parametrului formal va trebui precedat de cuvântul simbolul **&.**

Exemplul 7.1

```
void schimba_valoare (int x, int y)                  APEL PRIN VALOARE
{
   int   z=x;
   x = y;
   y = z;
}
void schimba_referinta (int &a, int &b)              APEL PRIN REFERINŢĂ
{
   int aux=a;
   a=b;
   b=aux;
}
void main ()
{
   int M=1, N=5;
   schimba_valoare(M,N);                             Se va afişa:
   cout << "M="<<M<< "   " << "N="<<N<<endl;         M=1  N=5
   schimba_referinta(M,N);
   cout << "M="<<M<< "   " << "N="<<N<<endl;         M=5  N=1
}
```

8. Transmiterea tablourilor unei funcţii

În C numele unui tablou reprezintă adresa primului element din tablou, celelalte elemente fiind memorate la adresele următoare de memorie. Din acest motiv, în cazul transmiterii unui tablou unei funcţii se transmite de fapt o adresă, realizându-se un apel numit pointer care determină modificarea parametrului actual.

Exemplul 8.1

```
# include <iostream.h>

void Genereaza (int A[100], int &x)
{
    cout << "Nr. de elemente=";
    cin >> x;
    for (int i=0; i<x; i++)
         A[i]=random (20);
}
void main ()
{
    int T[100], N;
    Genereaza (T, N);
    for (int i=0; i<N; i++)
         cout << T[i]<< " ";
}
```

Funcţia modifică parametrul actual T prin intermediul parametrului formal A deoarece se relizează apelul prin pointer.

Funcţia modifică valoarea parametrului actual N prin intermediul parametrului x deoarece se realizează apelul prin referinţă.

Datorită faptului că funcţia foloseşte doar adresa primului element pentru a accesa celelalte elemente ale vectorului, în cadrul prototipului sau antetului funcţiei este suficient dacă se specifică faptul că parametrul este un vector, nefiind necesară precizarea numărului de elemente ale vectorului.

Exemplul 8.2.

```
# include <iostream.h>

void Genereaza (int A[], int &x)
{
    cout << "Nr. de elemente=";
    cin >> x;
    for (int i=0; i<x; i++)
         A[I]=random (20);
}

void main ()
{
    int T[100], N;
    Genereaza (T, N);
    For (int i=0; i<N; i++)
         Cout << T[i]<< " ";
}
```

Funcţia modifică parametrul actual T prin intermediul parametrului formal A deoarece se relizează apelul prin pointer.

Funcţia modifică valoarea parametrului actual N prin intermediul parametrului x deoarece se realizează apelul prin referinţă.

O matrice este gestionată în memoria internă ca o succesiune de elemente. Liniile sunt memorate succesiv. Astfel pentru a reţine o matrice este suficient dacă se cunosc: adresa de început a primului element din matrice şi lungimea unei linii (adică numărul de coloane). Astfel la transmiterea unei matrice într-o funcţie este suficient dacă se precizează numele matrice respective (adresa de primului element) şi dimensiunea unei linii (numărul de coloane).

Exemplul 8.3.

```
void Genereaza (int A[][10], int &x, int &y)
{
    cout << "Nr. de linii: ";
    cin >> x;
    cout << "Nr. de coloane: ";
    cin >> y;
    for (int i=0; i<x; i++)
          for (int j=0; j<y; j++)
                A[i][j]=random (20);
}

void Afiseaza (int A[][10], int x, int y)
{
    for (int i=0; i<x; i++)
    {
          for (int j=0; j<y; j++)
          {
                cout.width(5);
                cout << A[i][j];
          }
          cout<<endl;
    }
}
void main ()
{
    int T[10][10], N, M;
    Genereaza (T, N, M);
    Afiseaza (T, N, M);
}
```

9. TRANSMITEREA ŞIRURILOR DE CARACTERE UNEI FUNCŢII

Datorită faptului că pentru a memora un şir de caractere compilatorul păstrează doar adresa de început a şirului de caractere iar restul caracterelor sunt memorate folosind octeţii următori până la întâlnirea unui '\0', în cazul unui parametru de tip şir de caractere, funcţia primeşte adresa de început a şirului, modificând astfel parametrul actual.

Exemplul 9.1.

```
void modifica (char * p)
{
   p[2]='x';
}
void afiseaza (char * p)
{
   cout << endl << p;
}
void main ()
{
   char s[20]="abcd";
   modifica(s);
   afiseaza(s);
 }
```

10. TRANSMITEREA STRUCTURILOR UNEI FUNCŢII

Exemplul 10.1.

```
struct persoana
   {
        char  nume[20];
        int varsta;
   };

void Citeste (persoana &x)
{
   cout << "Numele:";
   gets(x.nume);
   cout << "Varsta:";
   cin >>x.varsta;
}

void Afiseaza (persoana x)
{
   cout << "NUMELE: " << x.nume;
   cout << endl;
   cout << "VARSTA: " << x.varsta;
}

void main ()
{
   clrscr();

   persoana P;

   Citeste (P);
   Afiseaza(P);
}
```

Modificarea membrilor structurii necesita apelul prin referinţă

Pentru a afişa o structură este suficient apelul prin valoare