Lecţia 1. Introducere

1. Noţiunea de subprogram

Definiţie. *Un subprogram (funcţie) reprezintă un ansamblu de instrucţiuni (de declarare, executabile) scrise în vederea executării unei anumite prelucrări, ansamblu identificat printr-un nume şi implementat separat (în exteriorul oricărei alte funcţii).*

Subprogramele sunt unităţi de program care:

- au un algoritm propriu
- pot fi proiectate independent
- pot fi scrise independent
- pot fi compilate independent
- nu se pot executa independent ci numai în cadrul unui program (apel)

Un program scris în limbajul C/C++ este *un ansamblu de funcţii*, fiecare dintre acestea efectuând o activitate bine definită. Doar una dintre aceste funcţii este rădăcină sau funcţie principală - adică nu poate lipsi şi executarea programului începe automat cu ea. Aceasta se numeşte *main*.

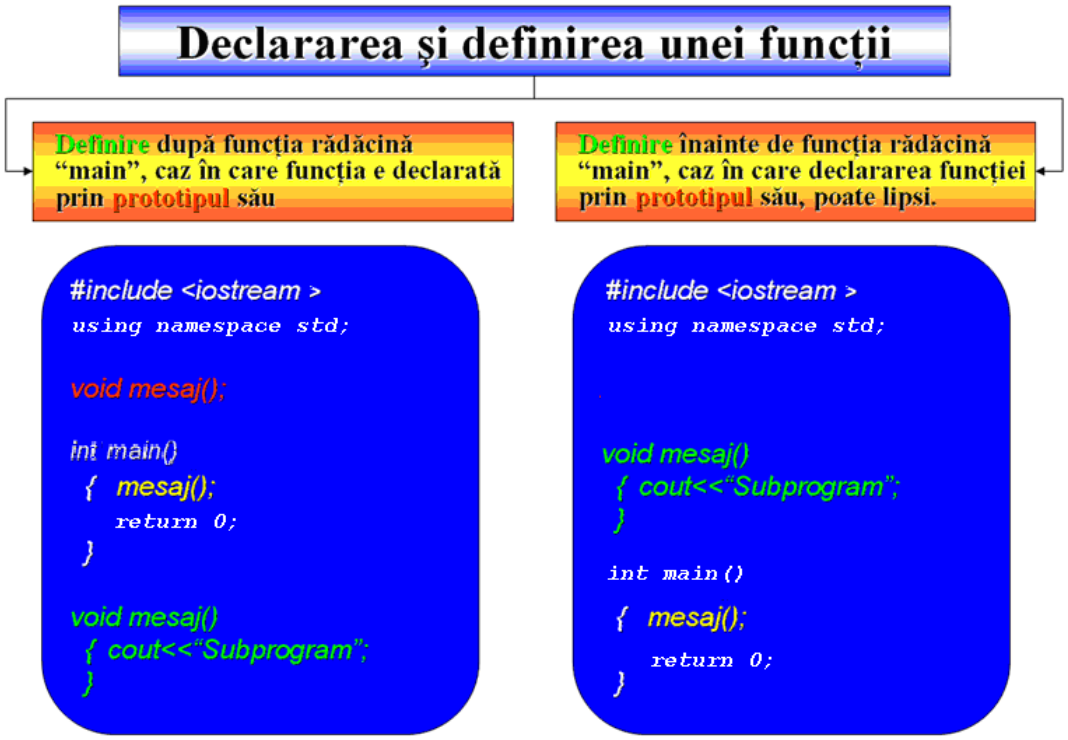
2. Avantajele utilizării subprogramelor

- elaborarea algoritmilor prin descompunerea problemei ce trebuie a fi rezolvată în mai multe probleme mai simple – în acest caz vom rezolva probleme mai simple
- modularizarea problemei (descompunerea in subprobleme)
- reutilizarea codului - subprogramul odată scris poate fi utilizat şi în alte programe
- reducerea numărului de erori care apar scrierea programului
- depistarea şi corectarea cu uşurinţă a erorilor
- realizarea unor programe uşor de urmărit.

3. Clasificarea subprogramelor

4. Structura unui subprogram C++

În limbajul C/C++ se utilizează **declaraţii şi definiţii** de funcţii.



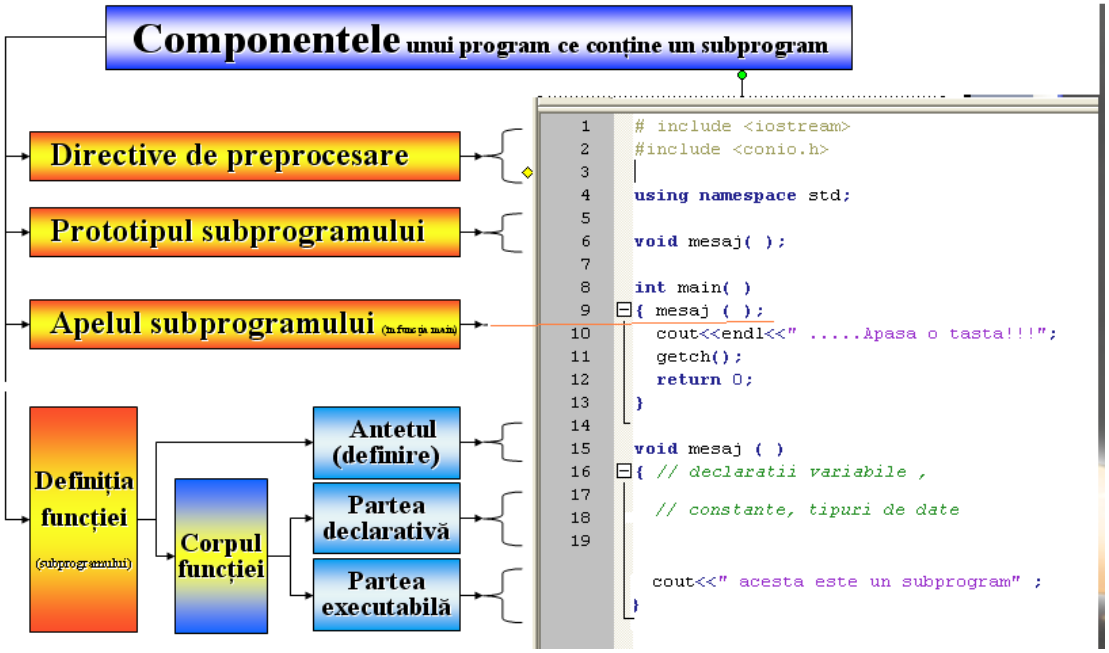
Declaraţia conţine antetul funcţiei şi informează compilatorul asupra tipului, numelui funcţiei şi a listei parametrilor formali (în care se poate indica tipul parametrilor formali şi numele acestora). Declaraţiile de funcţii se numesc ***prototipuri***, şi sunt constituite din antetul funcţiei.

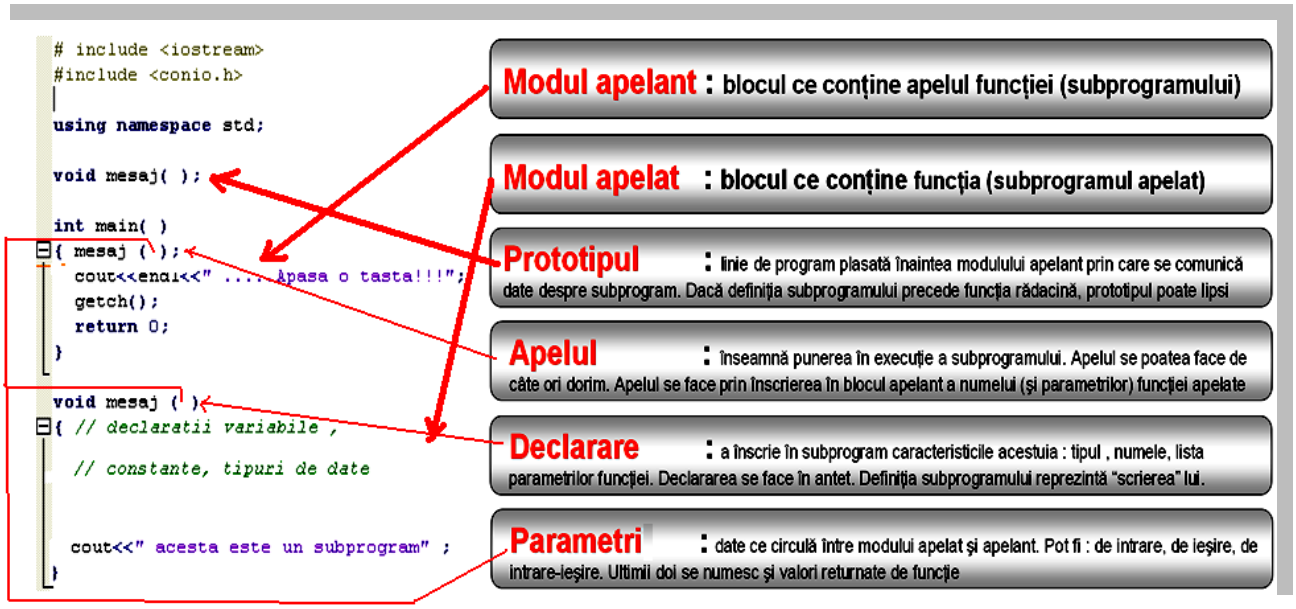
tip_returnat *nume_funcţie* (lista tipuri parametri formali); //prototip

Definiţia conţine antetul funcţiei şi corpul acesteia. Nu este admisă definirea unei funcţii în corpul altei funcţii.

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

Tip_returnat	☺ Reprezintă tipul rezultatului calculat şi returnat de funcţie sau tipul **void** (niciun rezultat!!) . O funcţie returnează cel mult un rezultat. ☺ Tipul rezultatului returnat poate fi orice tip de date cu excepţia masivelor (vectori, matrice, etc...). În cazul acesta se va utiliza tipul struct pentru a crea un nou tip de date ce include masivele. Noul tip poate fi un tip_returnat. ☺ Dacă dorim ca funcţia să returneze mai multe rezultate, putem utiliza variabile globale, tipul **struct** sau parametri transmişi prin referinţă (adresă) care să memoreze aceste rezultate şi să le furnizeze modulului apelant ☺ Dacă tipul rezultatului este **diferit de void**, corpul funcţiei trebuie să conţină cel puţin o instrucţiune **return**. **return expresie;** ☺ Tipul expresiei calculată şi returnată de funcţie trebuie să fie de acelaşi tip cu **tip_returnat** sau poate fi un tip convertibil implicit la tipul rezultatului.
Nume_funcţie	Un identificator ce reprezintă numele dat funcţiei de către cel ce o defineşte, pentru a o putea apela.
Lista parametrilor formali	Reprezintă o listă de declaraţii de parametri **formali**, separaţi prin virgulă: tip_parametru1 nume_parametru1,..., tip_parametruN nume_parametruN Această listă poate să fie şi vidă.
Instrucţiune compusă	Poate conţine declaraţii de variabile (variabile locale) şi instrucţiuni propriu-zise.




#include <iostream>
#include <conio.h>
using namespace std;
void mesaj( );
int main( )
{ mesaj ( );
cout<<endl<<" .... Apasa o tasta!!!";
getch();
return 0;
}
void mesaj ( )
{ // declaratii variabile ,
// constante, tipuri de date
cout<<" acesta este un subprogram" ;
}
Modul apelant : blocul ce conține apelul funcției (subprogramului)
Modul apelat : blocul ce conține funcția (subprogramul apelat)
Prototipul : linie de program plasată înaintea modulului apelant prin care se comunică date despre subprogram. Dacă definiția subprogramului precede funcția rădacină, prototipul poate lipsi
Apelul : înseamnă punerea în execuție a subprogramului. Apelul se poatea face de câte ori dorim. Apelul se face prin înscrierea în blocul apelant a numelui (și parametrilor) funcției apelate
Declarare : a înscrie în subprogram caracteristicile acestuia : tipul , numele, lista parametrilor funcției. Declararea se face în antet. Definiția subprogramului reprezintă "scrierea" lui.
Parametri : date ce circulă între modului apelat și apelant. Pot fi : de intrare, de ieșire, de intrare-ieșire. Ultimii doi se numesc și valori returnate de funcție

Lecţia 2.

Variabile globale şi variabile locale

Sistemul de operare alocă fiecărui program trei zone distincte în memoria internă în care se găsesc memorate variabilele programului.

segment de date / zona statică
segment stivă / zona stivă
zona heap

O variabilă se caracterizează prin 4 atribute:

1. **Clasa de memorare;**

   Precizează locul unde este memorată variabila respectivă. O variabilă poate fi memorată în segmentul de date, în cel de stivă, în heap sau într-un registru al microprocesorului.

2. **Vizibilitate**

   Precizează liniile textului sursă din care variabila respectivă poate fi accesată. Astfel avem:

   - Vizibilitate la nivel de bloc (instrucţiune compusă);
   - Vizibilitate la nivel de fişier – în cazul în care programul ocupă un singur fişier sursă;
   - Vizibilitate la nivel de clasă – este în legătură cu programarea pe obiecte.

Vizibilitatea unei variabile

Variabilele pot fi definite în C++ în orice poziție a programului. Poziția (locul) unde a fost definită o variabilă determină ***domeniul de vizibilitate a acesteia****. Acest domeniu începe în locul unde variabila este definită şi se sfârşeşte în locul de încheiere a blocului ce o conține.*

3. **Durata de viaţă**

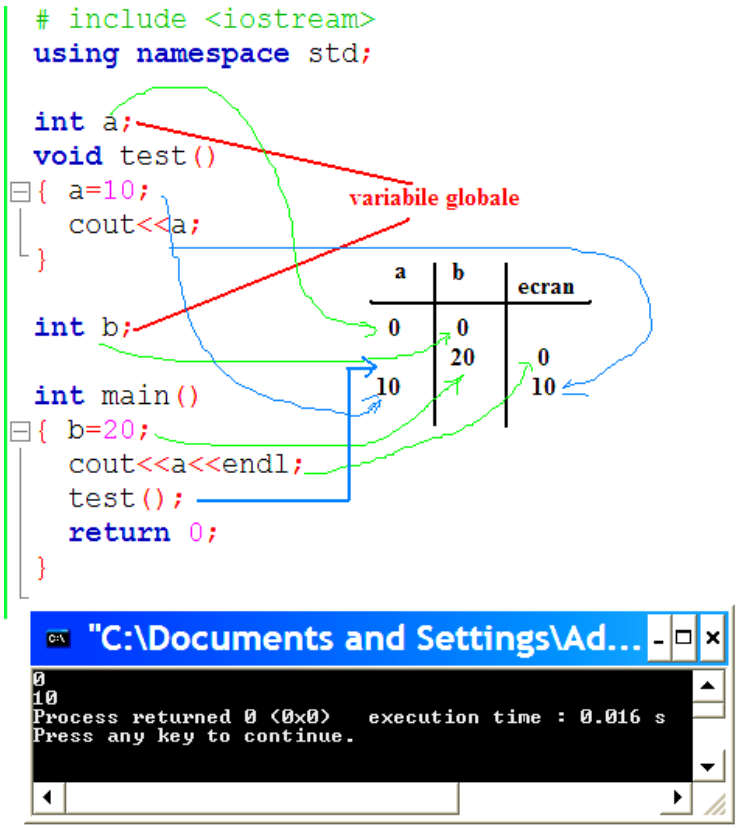
   Reprezintă timpul în care variabila respectivă arealocat spaţiul în memoria internă. Astfel avem:

   - Durata statică – variabila are alocat spaţiu în tot timpul execuţiei programului (Globală)
   - Durata locală – variabila are alocat spaţiu în timpul în care se execută instrucţiunile blocului respectiv (Locală)
   - Durată dinamică – alocarea şi dezalocarea spaţiului necesar variabilei respective se face de către programator prin operatori sau funcţii speciale.

4. **Tipul variabilei**

Variabile globale:

- Se declară în afara corpului oricărei funcţii.
- Pot fi utilizate de toate funcţiile care urmează în textul sursă declaraţiei variabilei respective.
- La declarare, variabilele globale sunt iniţializate implicit cu 0.
- **Atributele variabilelor globale sunt:**
  1. Clasa de memorare – segmentul de date.
  2. Durata de viaţă a variabilelor globale este statică. Ele au spaţiul de memorie rezervat în tot timpul execuţiei programului.
  3. Vizibilitatea – În cazul în care declaraţiile acestora sunt înaintea tuturor funcţiilor, ele sunt vizibile la nivelul întregului program (fişier). Dacă anumite funcţii se află plasate înaintea declaraţiilor acestor variabile, atunci ele sunt vizibile doar pentru funcţiile care sunt plasate după aceste declaraţii.



Variabile locale:

- Sunt declarate în corpul funcţiilor.
- Variabilele declarate în corpul funcţiei main() sunt locale
- **Atributele variabilelor locale sunt:**

1. **Clasa de memorare** a variabilelor locale este, implicit, segmentul de stivă.

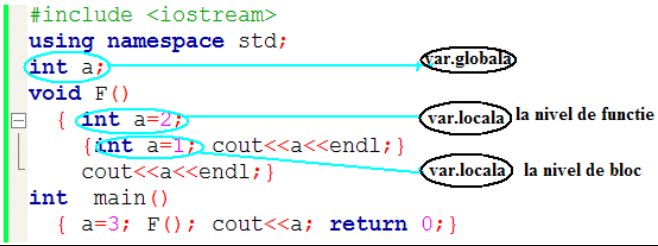
Există posibilitatea ca acestea să fie alocate în registrele microprocesorului, caz în care declaraţia lor trebuie precedată de cuvântul cheie register:	`void test ()`
	`{ int a=1;}`
	`int main()`
	`{ register int b=4; return 0;}`

Variabilele locale nu sunt iniţializate implicit cu 0. Dacă nu sunt iniţializate explicit de programator, ele reţin o valoare oarecare, numită valoare reziduală.

2. Durata de viaţă a variabilelor locale este locală. Trăiesc atâta timp cât durează executarea funcţiei sau blocului în care au fost definite.

3. **Vizibilitatea.**

- Vizibilitatea variabilelor locale este la nivelul blocului în care au fost declarate.




```
#include <iostream>
using namespace std;
int a;
void F()
{ int a=2;
   {int a=1; cout<<a<<endl;}
    cout<<a<<endl;}
int main()
   {a=3; F();cout<<a; return 0;}
```

- În cazul în care, într-un anumit bloc sunt vizibile mai multe variabile, toate cu acelaşi nume, dar au domenii de vizibilitate diferite, se accesează variabila cu vizibilitatea cea mai mică.

Var globala	Var.locale		Ecran
a_global (=0)	a_F	a_F_Bloc	
3	2	1	1 (a_F_Bloc) 2 (a_F) 3 (a_global)

```
#include <iostream>
using namespace std;
int a;
void F()
{ int a=2;
  {int a=1; cout<<a<<endl;}
    cout<<a<<endl;}
int  main()
 { a=3; F(); cout<<a; return 0;}
```

- Dacă se declară o variabilă locală în ciclul for ea este vizibilă doar în blocul for

```
int n=4, s=0;
for (int i=1; i<=n; i++) s+=i;
cout<<s;
```

Variabile globale şi variabile locale

"a" este variabila globală (este vizibilă pe tot parcursul programului)

"b" este variabila locală (este vizibilă doar în blocul de program curent)

"c" este variabila locală (este vizibilă doar în funcţia "mesaj()")

```
#include <iostream>
using namespace std;
int a;
void mesaj();

int main()
 {  mesaj();
    { //... instrucţiuni
      float b;
      //... instrucţiuni
    }
    return 0;
 }
void mesaj()
 { int c;
   cout<<"N = "; cin>>c;
   cout<<"Dublul = "<<c*2;
 }
```

Recomandări variabile globale şi locale

- Atenţie, variabilele globale pot fi modificate în interiorul subprogramelor. Se va acorda atenţie mare prelucrării acestora
- ***Utilizarea*** variabilelor globale să fie limitată deoarece creează dependenţe între subprograme
- Utilizarea variabilelor locale asigură portabilitatea programelor.
- Utilizarea constantelor globale permit modificarea la nevoie într-un singur loc a unei valori ce afectează întregul program (şi toate subprogramele sale)

Lecţia 3. Apelul subprogramelor.

Construirea apelului subprogramului

- O funcţie poate fi ***apelată*** printr-o ***instrucţiune de apel***, de forma:

nume_funcţie (lista_parametrilor_efectivi);

- Parametrii efectivi trebuie să corespundă cu cei formali ca ordine şi tip.

*Dacă subprogramul este definit de utilizator, el trebuie declarat prin **prototip** sau prin definirea subprogramului înainte de blocul apelant*

Apelul este modul prin care subprogramul e pus în execuție. Apelul poate fi făcut ori de câte ori este nevoie.

Apelul poate fi făcut din funcția rădăcină main(), dintr-o altă funcție sau din ea însăşi prin autoapelare (recursivitate)

*Dacă subprogramul este standard (de sistem), trebuie inclus fişierul ce conține subprogramul utilizat. Pentru a utiliza funcția radical (sqrt)Ex. **#include <cmath>***

Exemplu de apel al unei funcții ce nu returnează o valoare

p1.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int n;
5  void suma();///prototip functie
6
7  int main()
8  {
9   cout<<"n="; cin>>n;
10  suma(); ///apel
11  return 0;
12 }
13 void suma()///definirea functiei
14 { int i,s=0;
15   for(i=1;i<=n;i++)s+=i;
16   cout<<s;
17 }
18
```

Exemplu de apel al unei funcții ce returnează o valoare

p1.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int n;
5  int suma();///prototip functie
6
7  int main()
8  {
9   cout<<"n="; cin>>n;
10  cout<<"suma="<<suma(); ///apel
11  return 0;
12 }
13 int suma()///definirea functiei
14 { int i,s=0;
15   for(i=1;i<=n;i++)s+=i;
16   return s;
17 }
18
```

- La apelul unei funcţii, valorile parametrilor efectivi se atribuie parametrilor formali corespunzători. În cazul în care tipul unui parametru efectiv diferă de tipul parametrului formal corespunzător, acesta (tipul) va fi convertit spre tipul parametrului 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**
- La revenirea din funcţie, controlul este redat funcţiei apelante, şi execuţia continuă cu instrucţiunea următoare instrucţiunii de apel, din funcţia apelantă.

- O altă posibilitate de a apela o funcţie este aceea în care apelul funcţiei constituie operandul unei expresii (apelul functiei intervine intr-o expresie). Acest lucru este posibil doar în cazul în care funcţia returnează o valoare, folosită în calculul expresiei (doar pentru functiile cu tip).

Transmiterea parametrilor

Datele ce "circulă" între modulul apelant şi apelat se introduc între paranteze, după identificatorul (numele) subprogramului

- *În **antetul funcției** parametrii se înscriu prin tip şi nume, separați prin virgulă, fără a fi grupați. Aceştia se numesc **parametri formali***
- *În **apelul funcției** parametrii se înscriu separați prin virgulă, în aceeaşi ordine ca în antet (regula de corespondență), prin valori concrete. Ei se numesc **parametri efectivi (actuali)***
- *Regula de corespondență notifică o anumită concordanță între numărul, ordinea şi tipul parametrilor formali şi a parametrilor efectivi*

- Pentru memorarea parametrilor, subprogramele folosesc segmentul de stivă.
- Memorarea parametrilor transmişi se face în ordinea în care aceştia figurează în antet: de la stânga la dreapta.
- În cadrul subprogramului, parametrii transmişi şi memoraţi în stivă sunt variabile. Numele lor este cel din lista parametrilor formali.
- Variabilele obţinute în urma memorării parametrilor transmişi sunt variabile locale

Observație

- *Numărul parametrilor formali poate să difere de cel al parametrilor efectivi în cazul funcțiilor cu **număr de parametri variabili** şi în cazul **supraîncărcării funcțiilor***
- *Tipul parametrilor formali poate să difere de cel al parametrilor efectivi în cazul **conversiei implicite** a parametrilor efectivi în tipul parametrilor formali (ca o operație de atribuire)*
- *Tipul parametrilor formali poate să difere de tipul parametrilor efectivi şi în cazul **supraîncărcării funcțiilor***
- *Numele parametrilor formali poate să difere de numele parametrilor efectivi*

*Parametrii pot fi transmişi **prin valoare** sau pot fi transmişi **prin referință***

Transmiterea parametrilor prin valoare

Se utilizează atunci când suntem interesaţi ca subprogramul să lucreze cu acea valoare, dar, în prelucrare, nu ne interesează ca parametrul efectiv (cel din blocul apelant) să reţină valoarea modificată în subprogram.

```
#include <iostream>
using namespace std;
void test(int n)
{ n++; cout<<n<<endl;}
```

Var.locale		Ecran
n_main	n_test	
1	2	2 (n_F)

```
int main()
{int n=1;
 test(n); cout<<n<<endl;
 return 0;}
```

		1 (n_main)

adrese	Valoare
&n_test	1 → 2
& revenire din apelul test(1)	
&n - main	1

```
#include<iostream>
using namespace std;
void test(int n)
 {cout<<n<<endl;}
int main()
 { test(3); test(3+4*5);
   return 0;}
```

Var.locale		Ecran
apel_main	n_test	
test(3);	3	3 (n_test)
test(3+4*5);	23	23 (n_test)

```
#include <iostream>
using namespace std;

int suma(int a, int b)
   { return a+b;}

int main()
  { int c=4, d=3;
    cout<<suma(2,3)<<endl;
    cout<<suma(2+7,3-1*2)<<endl;
    cout<<suma(c,d)<<endl;
    cout<<suma(1.9,3.3)<<endl;
    return 0;
  }
```

apel	Variabila	valoare	ecran
	c_main	4	
	d_main	3	
suma(2,3)	a_suma	2	
	b_suma	3	
revenire			5
suma(2+7,3-1*2)	a_suma	9	
	b_suma	1	
revenire			10
suma(4,3)	a_suma	4	
	b_suma	3	
revenire			7
suma(1.9,3.3)	a_suma	1	
	b_suma	3	
revenire			4

PRECIZĂRI.

Se pot transmite prin valoare prin intermediul parametrilor efectivi:

- valori reţinute de variabile
- valori constante
- expresii (acestea pot conţine şi apeluri de funcţii) caz în care mai întâi se evaluează expresia şi apoi se transmite valoarea rezultată

Transmiterea ***„prin valoare”*** a tablourilor (MASIVELOR) permite ca funcţiile să întoarcă noile valori ale acestora (care au fost atribuite în funcţii). Numele masivelor sunt de fapt pointeri către componentele lor. La transmiterea numelui unui tablou ca parametru efectiv, se transmite de fapt adresa de început a tabloului

```
#include <iostream>
using namespace std;
int n;

void citeste (int x[10])
  {   cin>>n;
      for (int i=1; i<=n;i++)     cin>>x[i];
  }

int main()
  { int a[10];
    citeste(a);
    for(int i=1; i<=n;i++)    cout<<a[i]<<" ";
    return 0;
  }
```

Transmiterea parametrilor prin referinţă

- Se utilizează atunci când ne interesează ca la revenirea din subprogram variabila transmisă să reţină valoarea stabilită în timpul execuţiei programului şi nu valoarea de la apel.
- Subprogramul reţine în stivă adresele acestor variabilei.

Parametrii transmişi prin referință vor fi precedați de caracterul ampersand "&" atât la declararea cât şi la definirea funcției

În apel, parametrii efectivi corespunzători parametrilor formali transmişi prin referință trebuie **să fie variabile** de memorie.

EXEMPLU

```
#include<iostream.h>
#include<conio.h>
void f(int &a);   // prototipul funcției
int main ()
{ int x;
  cout<<"\nX = "; cin>>x;
  cout<<"\nInainte de functie X  = "<<x;
  f(x);    // apel pentru prelucrare
  cout<<"\nDupa functie     X  = "<<x;
        // x a modificat la revenire
  return 0; }
  void f(int &a)
  { a=a*a;
    cout<<"\nI functie        A  = "<<a; }
```

```
//Citirea si afisarea unui vector
#include <iostream>
using namespace std;

void citeste (int x[], int &n)
  {   cin>>n;
      for (int i=1; i<=n;i++)     cin>>x[i];
  }

int main()
  { int a[10],n;
    citeste(a,n);
    for(int i=1; i<=n;i++)    cout<<a[i]<<" ";
    return 0;
  }
```

Exemple de aplicații cu subprograme

EXEMPLU 1

```
#include <iostream>
using namespace std;

float med(int a, int b, int c)
{ float med=(float(a)+b+c)/3;
  return med; }


int main()
{ int x, y, z;
  cout<<"\nNota 1 = "; cin>>x;
  cout<<"\nNota 2 = "; cin>>y;
  cout<<"\nNota 3 = "; cin>>z;
  cout<<"\nMedia  = "
      <<med(x,y,z);
 return 0;
}
```

Detalii

Programul conține o funcție cu trei parametri de intrare, acestea fiind variabilele locale x, y, z (notele ca numere intregi). Funcția returnează valoarea calculată prin funcția insăşi (apelată ca operand). Parametrii sunt transmişi prin valoare.

EXEMPLU 2

```
#include<iostream>
 void med();

int main ()
{ cout<<"Calculam media a trei
          numere\n";
  med(); // apel procedural
  return 0;
}

void med()
{ int x, y, z;
  cout<<"\nNota 1 = "; cin>>x;
  cout<<"\nNota 2 = "; cin>>y;
  cout<<"\nNota 3 = "; cin>>z;
  float media=(float(x)+y+z)/3;
  cout<<"\nMedia  = "<<media;
}
```

Detalii

Programul contine o funcție fără parametri ce nu returnează nici o valoare. Funcția conține variabilele locale x, y, z (notele, numere intregi). Funcția este apelată ca un bloc de instrucțiuni procedurale.

EXEMPLUL 3

Să se scrie o funcţie care calculează cel mai mare divizor comun pentru două numere naturale nenule (utilizând algoritmul lui Euclid).

here | p1.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int cmmdc(int a, int b)
5  {   int r,d=a,i=b;
6      if(d<i){d=b; i=a;}
7      do
8        { r=d%i;  d=i; i=r;}
9      while(r!=0);
10     return d;
11 }
12 int main()
13 { int n1,n2;
14   cin>>n1>>n2;
15   if (n1&&n2) cout<<endl<<"cmmdc="<<cmmdc(n1,n2);
16   else cout<<"Numerele nu sunt nenule!";
17   return 0;
18 }
19
```