CAPITOLUL I

1. SUBPROGRAME

1.1. Prezentare generală

Să ne imaginăm că dorim să scriem un program în care să citim o mulţime de numere întregi ale căror proprietăţi dorim sa le studiem şi eventual să efectuăm în funcţie de proprietăţile lor diferite operaţii cu numerele citite.
De exemplu presupunem că dorim să ştim dacă numerele citite sunt prime sau nu, dacă nu sunt prime dorim să le descompunem în factori primi, vrem să verificăm dacă au proprietea de palindrom sau să le convertim în alte baze şi altele.
Operaţiile pe care le-am înşirat mai sus sunt suficiente să ne dăm seama că programul pe care urmează să-l scriem este lung, greu de scris de urmărit şi de corectat în situaţia în care mai facem şi greşeli pe parcursul scrierii lui.

Ce este de făcut pentru a nu abandona înainte de a începe rezolvarea unei probleme complexe?

Limbajul C++ ne pune la dispoziţie instrumentele necesare pentru a ne uşura munca, pentru a nu fi nevoiţi să repetăm de multe ori aceleaşi secvenţe de program, pentru a fi uşor de gândit, de realizat, de corectat şi de înţeles şi de către o altă persoană decât cea care l-a scris.
Ne rămâne doar să mai învăţăm câte ceva despre subprograme.

Utilizarea subprogramelor ne permite accesul la ceea ce poate oferi programarea structurată în C++.

Beneficiem astfel de următoarele avantaje:

- Putem **reutiliza cu uşurinţă codul scris**.
- **Elaborarea algoritmilor devine mult mai uşoară** datorită descompunerii problemei în probleme mai simple, mai uşor de rezolvat.
- **Corectarea erorilor** se realizează **cu mai mare uşurinţă**.
- Descompunând problema şi rezolvând la un moment dat probleme mai simple, **reducem numărul erorilor** la elaborarea algoritmilor.

În C++ subprogramele sunt de tip funcţie.

Exemplu:
Se citesc două numere întregi. Să se calculeze suma lor.

```
#include <iostream.h>           //SUMA
#include <conio.h>
int suma(int a, int b);         //antetul funcţiei
void main()
{int x,y,z;
 cout<<"x="; cin>>x;
 cout<<"y="; cin>>y;
 z=suma(x,y);                   //apelul funcţiei suma
 cout<<"suma="<<z;
 getch();
}
int suma(int a, int b)          //funcţia suma de tip int, cu parametrii a şi b de tip int
{int c;
 c=a+b;
 return c;
}
```

În exemplul de mai sus am definit funcţia ***suma***, de tip întreg, cu parametrii **a** şi **b** de tip întreg. După cum se poate observa funcţia returnează o valoare întreagă, adică de acelaşi tip cu tipul funcţiei.

return c;

Funcţia se apelează prin numele ei, are două argumente, **x** şi **y** de acelaşi tip cu parametrii funcţiei.

La apelul funcţiei valoarea returnată de funcţie se depune în variabila **z**.

Este necesar să facem câteva precizări legate de variabile, înainte de a studia mai amănunţit funcţiile.

1.2. Variabile globale şi variabile locale
Domeniul de vizibilitate, durata de viaţă şi zona de memorie alocată

Variabilele pot fi definite în orice poziţie într-un program, astfel putem defini variabile înafara funcţiilor (chiar înafara funcţiei principale main), în acest caz poartă numele de ***variabile globale***, în interiorul unei funcţii caz în care le vom numi ***variabile locale***.

În realitate în C++ variabilele pot fi definite în orice punct al programului. Sigur nu este indiferentă poziţia în care se defineşte o variabilă, de aceasta depinde domeniul de vizibilitate al variabilei respective, adică zona de program din care poate fi accesată variabila.

Domeniul de vizibilitate al unei variabile începe în momentul declarării ei şi sfârşeşte în moentul în care se încheie structura în interiorul căreia a fost definită.

1. **Variabilele globale**, declarate la început de program (înaintea oricarei funcţii) sunt accesibile din orice punct al programului.

Dăcă există funcţii definite înaintea acestor variabile, pentru acele funcţii variabilele nu sunt vizibile (nu pot fi accesate din interiorul acestor funcţii).

Durata de viaţă a unei variabile globale, este atâta timp cât programul se execută.

Are alocat spaţiu de memorie pe tot parcursul execuţiei programului.

La declarare, variabilele globale se iniţializează în mod automat cu 0.

2. **Variabilele locale** definite în interiorul unei funcţii, sunt accesibile doar funcţiei, nu pot fi utilizate înafara ei.

Durata de viaţă a unei variabile locale este pe tot parcursul execuţiei funcţiei dacă a fost definită la începutul blocului de instrucţiuni.

Are alocată zonă de memorie numai pe parcursul execuţiei funcţiei respective.

Dacă o variabilă este definită în interiorul unui bloc de instrucţiuni, poate fi accesată doar în interiorul acelui bloc, nu poate fi utilizată înafara lui.

Durata de viaţă a acestei variabile este până la terminrea execuţiei blocului în care a fost definită.

Are alocată zonă de memorie numai pe parcursul executiei blocului respectiv.

Variabilele locale nu se iniţializează în mod automat. Dacă nu le iniţializează programatorul, ele reţin o aşa numită *valoare reziduală*, adică ceea ce se gaseste în memorie în acel moment.

```
#include <iostream.h>
float a;            -variabilă globală accesibilă pe tot parcursul
                     programului
void main()
{ char b;           - variabilă locală accesibilă doar în interiorul
                     funcţiei main
    {
     int c=5;       -variabilă locală accesibilă doar în interiorul
                     blocului în care a fost definită
     cout<<a<<b<<c;
    }
}
```

Observaţie:

În cazul în care într-un bloc sau într-o funcţie sunt vizibile mai multe variabile cu acelaşi nume, dar cu domenii de vizibilitate diferite, se accesează variabila cu domeniul cel mai mic de vizibilitate.

Exemplu:

În programul de mai jos sunt definite două variabile cu numele **a**. O variabilă globală şi o variabila locală definită în cadrul funcţiei **f**.

În timpul execuţiei, functia **f accesează variabila locală**.

```
#include <iostream.h>
#include <conio.h>
int a;   //a  este variabilă globală

int f()            //funcţia suma de tip int, cu parametrii a şi b de tip int
{int a=5;    //funcţia foloseste variabila locala a(definita in funcţie), nu pe cea globală
 a++;
 return a;
}

void main()
{
 a=f();                    //apelul funcţiei suma
 cout<<"a="<<a;
 getch();
}
```

Rezultatul rularii programului:

a=6

În ceea ce priveşte **zona de memorare a variabilelor**, putem spune că şi din acest punct de vedere sunt diferenţe între variabilele globale şi cele locale.

Sistemul de operare alocă fiecărui program trei zone distincte de memorie internă, pentru memorarea variabilelor programului.

Există şi posibilitatea ca o variabilă să fie memorată într-un registru al microprocesorului. Timpul de acces la variabilele memorate în regiştrii este foarte mic şi astfel poate fi îmbunatăţit timpul de execuţie al programului.

Zonele de memorie internă alocte:

Segmentul de date	-Variabile globale
Segmentul de stiva	-Variabile locale
Heap	-Date alocate dinamic

1. **Variabilele globale** se memorează în cadrul **segmentului de date**.
2. **Variabilele locale** se memorează pe **segmentul de stivă** pus la dispoziţia funcţiei în interiorul careia sunt definite.
3. Datele alocate dinamic nu constitue obiectul acestui manual.

1.3. Structura funcţiilor şi apelul lor

O funcţie se compune din **antet** şi **corpul funcţiei**.

- **Antetul** unei funcţii cuprinde tipul de dată returnat de funcţie, numele funcţiei şi lista parametrilor.
- **Corpul funcţiei** este format din una sau mai multe instrucţiuni cuprinse între acolade.

1.3.1. Formatul unei funcţii:

tip nume(parametru1, parametru2, …) { instrucţiuni }

- **tip** – reprezintă un tip de dată şi desemnează tipul datei returnate de funcţie (nu este obligatoriu).
- **nume** – este un identificator prin intermediul căruia se apelează funcţia.
- **parametrii** – permit transmiterea argumentelor către funcţie atunci când este apelată. Fiecare parametru constă dintr-un tip de dată urmat de un identificator, ca orice variabilă şi acţionează ca o variabilă locală.
- **Instrucţiuni** – reprezintă corpul funcţiei şi trebuie să fie cuprins între acolade {…} .

1.3.2. Declararea şi definirea unei funcţii

A declara o funcţie înseamnă a o "anunţa", adică a face cunoscut programului că această funcţie există şi caracteristicile ei, adică tipul, numele şi lista eventualilor parametrii. Declararea unei funcţii se face cu ajutorul antetului funcţiei.

tip nume(parametru1, parametru2, …);

A defini o funcţie înseamnă a o descrie înpreună cu corpul funcţiei.

tip nume(parametru1, parametru2, …) { instrucţiuni }

definiţia unei funcţii ţine loc si de declaraţie în cazul în care funcţia este plasată înaintea funcţiei principale *main().*

1.3.3. Structura unui program care conţine subprograme

Un program poate să conţină oricâte funcţii.

Aşa cum o variabilă nu poate fi accesată în limbajul de programare C înainte de a fi definită, nici o funcţie nu poate fi apelată fără ca programul să ştie caracteristicile acelei functii. Pentru a putea fi apeltă funcţia trebuie să fie cel putin declarată.

Există două modalităţi de a plasa funcţiile într-un program:

a. **Înaintea funcţiei principale main()**, ca în exemplul de mai jos:

```
#include <iostream.h>
int f()
{int a=3;
 return a;
}
void main()
{cout<<f;
}
```

b. **După funcţia principală main()**, caz în care funcţia trebuie declarată înainte,

```
#include <iostream.h>
int f();                    //declararea funcţiei f

void main()
{cout<<f;
}
int f()                     //definirea funcţiei f
{int a=3;
 return a;
}
```

1.3.4. Apelul unei funcţii

O funcţie poate fi apelată din funcţia principală *main*, dintr-o altă funţie sau chiar din funcţia însăşi. Acest din urmă caz îl vom trata într-un capitol separat.

După cum am spus în paragraful 1.3.1, antetul unei funcţii poate conţine tipul datei returnate de funcţie sau nu. Aceasta datorită faptului că o funcţie poate să returneze o valoare sau nu.

Din acest punct de vedere putem împarţi funcţiile în două categorii:

1. **Funcţii care returnează o valoare**, definite astfel:

tip nume(parametru1, parametru2, …) { instrucţiuni }

O funcţie care returnează o valoare trebuie să conţină linia de cod:

return expresie;

Rezultatul returnat de funcţie poate fi utilizat într-o expresie sau poate fi afişat.

Prin urmare apelul unei funcţii care returnează o valoare poate fi de forma:

variabila=nume(parametrul1,parametrul2,…);

sau

cout<<nume(listă parametrii);

Exemplu:

Să se însumeze primele n numere naturale.

```
#include <iostream.h>
#include <conio.h>
int n;                  //declararea variabilei globale
int suma();             //declararea functiei – funcţia nu are parametrii

void main()
{cout<<"n="; cin>>n;
 cout<<"suma primelor n numere naturale:"<<suma();     //apelul funcţiei
 getch();
}

int suma()              //definirea functiei
{int i,s=0;
 for(i=1;i<=n;i++)
   s+=i;
 return s;              //valoarea returnata de funcţie
}
```

2. **Funcţiile care nu returnează o valoare**, **în loc de *tip* vor conţine cuvântul *void***

void nume(parametru1, parametru2, …) { instrucţiuni }

Apelul unei astfel de funcţii:

nume(parametrul1,parametrul2,…);

```
#include <iostream.h>          //AFISARE
#include <conio.h>
int n;
int afisare();          //declararea functiei – funcţia nu are parametrii

void main()
{cout<<"n="; cin>>n;
 afisare();             //apelul funcţiei
 getch();
}

int afisare()           //definirea funcţiei
{int i,s=0;
 for(i=1;i<=n;i++)
   cout<<i<<" ";
}
```

Observaţie:

O funcţie poate să nu conţină parametrii.

În acest caz atât la declarare cât şi la definire şi apel numele funcţiei va fi urmat de paranteze rotunde, dar fără să conţină nici un parametru.

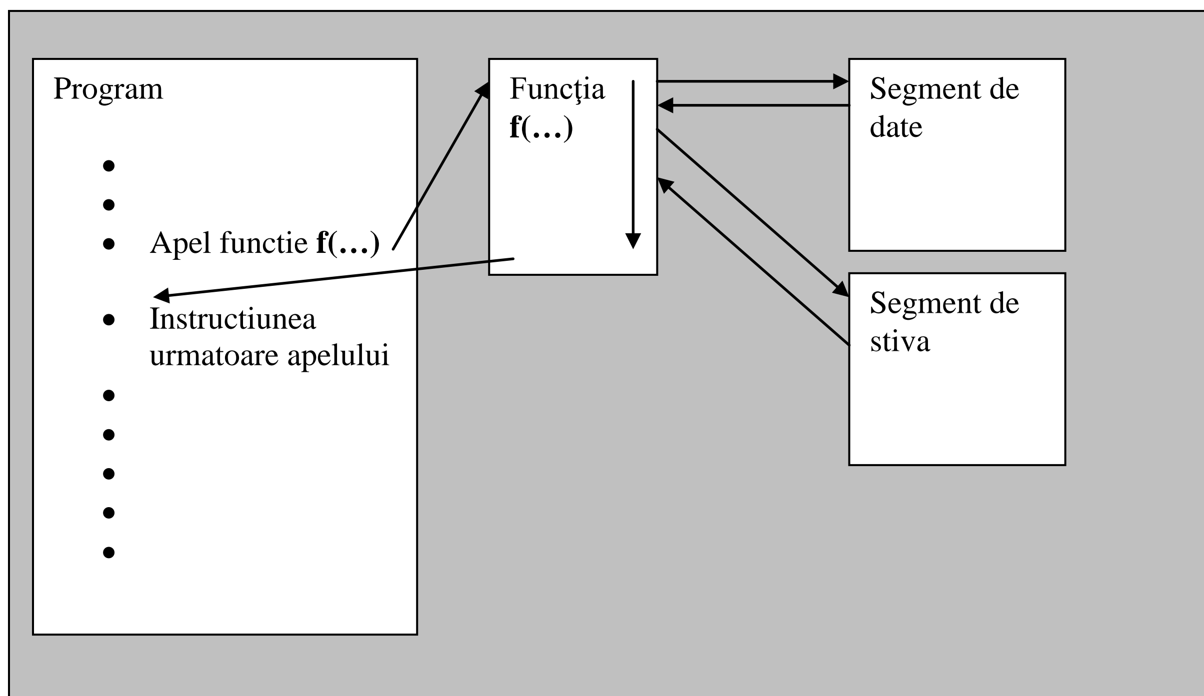
Parantezele sunt obligatorii pentru a informa compilatorul că este vorba despre o funcţie şi nu de o variabilă.

1.3.5. Modul de funcţionare al unui program care conţine una sau mai multe funcţii

În momentul în care se apelează o funcţie controlul programului se va da funcţiei respective pentru a fi executată.

În momentul apelării unei funcţii se întplă mai următoarele:

- **Se alocă spaţiu pe segmentul de stivă unde se memorează parametrii funcţiei** (acă funcţia are parametrii)**.**
- Se **salvează adresa instrucţiunii următoare apelului**, pentru ca la revenirea din funcţie programul să continue cu instrucţiunea următoare.
- Salt la prima instrucţiune a funcţiei
- **Se alocă spaţiu pe segmentul de stivă unde se memorează variabilele locale** (dacă au fost declarate variabile locale**).**
- Se **execută în segvenţă instrucţiunile funcţiei** apelate.
- Se **eliberează segmentul de stivă alocat funcţiei.**
- Se **revine în punctul în care s-a apelat funcţia** şi programul **continuă cu instrucţiunea următoare**



1.3.6. Transmiterea parametrilor

Parametrii unei funcţii au rolul de a transmite date funcţiei şi/sau de a transmite date din funcţie spre zona de program care a apelat funcţia. Parametrii se descriu prin tipul şi numele lor, în antetul funcţiei, despărţiţi între ei prin virgulă.

Parametrii care se află în antetul funcţiei se numesc **parametri formali** iar cei care se află în instrucţiunea de apel a funcţiei se numesc **parametri efectivi**.

Numele (identificatorii) parametrilor formali poate să fie diferit de numele parametrilor efectivi sau poate să coincidă.
Transmiterea parametrilor se face prin corespondenţă, din acest motiv între parametrii formali şi cei efectivi trebuie să existe o anumită concordanţă:

- **Numărul** parametrilor formali trebuie să coincidă cu numărul parametrilor reali. Există o singură excepţie şi anume parametrii cu valoare implicită, despre care vom vorbi mai târziu.
- **Ordinea** parametrilor formali trebuie să fie aceeaşi cu cea a parametrilor efectivi, deoarece după cum am spus transmiterea parametrilor se face prin corespondenţă.
- **Tipul** parametrilor efectivi trebuie să poată fi convertit implicit în tipul parametrilor formali sau să coincidă.

Exemplu:

```
int suma (int a, float b)
              ↑      ↑

x=suma(       2,     3.14   );
```

- Parametrii se memorează pe segmentul de stivă pus la dispoziţia funcţiei în momentul apelării ei. Aceştia se memorează în ordinea în care au fost trecuţi în antetul funcţiei.
- Parametrii transmişi şi memoraţi pe segmentul de stivă sunt variabile locale, iar numele lor este cel din lista de parametrii formali.

Din punctul de vedere al modului de transmitere a parametrilor putem împărţi parametrii în două categorii: **parametrii transmişi prin valoare** şi **parametrii transmişi prin referinţă.**

1. **Parametrii transmişi prin valoare**

Parametrii transmişi prin valoare au rolul de a transmite date din zona de program în care s-a apelat funcţia înspre funcţie.

Funcţia va memora aceste date pe segmentul de stivă pus la dispoziţie. După terminarea execuţiei funcţiei, segmentul de stivă se eliberează şi astfel programul nu mai are acces la variabilele memorate pe stivă. Din acest motiv parametrii transmişi prin valoare nu pot transmite datele modificate în cadrul funcţiei înspre zona de program de unde a fost apelată funcţia.

Prin valoare se pot transmite: **variabile**, **expresii** şi chiar **funcţii**.

Exemplu:

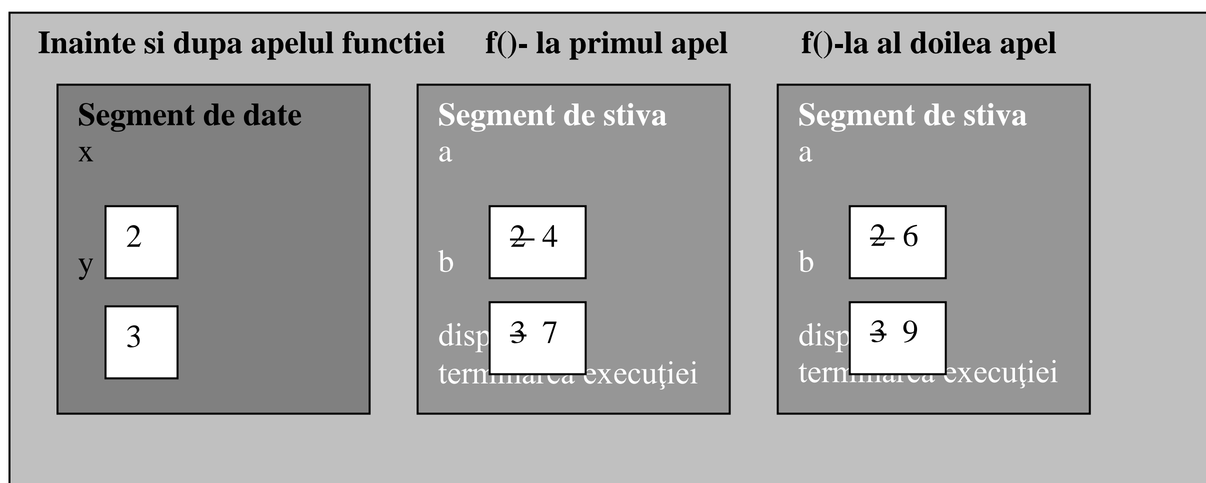
```
#include <iostream.h>          //PARAMETRII TRANSMISI PRIN VALOARE
#include <conio.h>             //parametrii transmisi prin valoare
void f(int a,int b);

void main()
{ int x=2,y=3;
  f(x,y);                      //parametrii efectivi sunt variabile
  cout<<"x="<<x<<" y="<<y<<endl;        //la revenire x si y sunt nemodificati
  f(2*x,2*y);                  //parametrii efectivi sunt expresii
  cout<<"x="<<x<<" y="<<y;              //la revenire x si y sunt nemodificati
  getch();
}
void f(int a,int b)      //functia modifica valorile parametrilor, dar nu le transmite inafara
{a=a+2;
 b=a+3;
 cout<<"a="<<a<<" b="<<b<<endl;
}
```

```
Rezultatul rulării programului:
a=4 b=7
x=2 y=3
a=6 b=9
x=2 y=3
```

Observaţi că în urma apelării funcţiei **f()**, deşi în interiorul funcţiei variabilele **a** şi **b** şi-au modificat valorile, noile valori nu s-au transmis variabilelor **x** şi **y**, acestea rămân nemodificate.

Acest lucru se întâmplă datorită faptului că pe segmentul de stivă **s-a creat o copie** a variabilelor **x** şi **y** cu numele **a** şi **b**, iar funcţia operează asupra acestora.

La terminarea execuţiei funcţiei, se eliberează segmentul de stivă iar valorile variabilelor **a** şi **b** se pierd.



La declararea unei funcţii putem specifica o valoare implicită pentru unii parametrii.

În cazul în care nu există parametrii efectivi corespunzători, vor fi folosite de către funcţie valorile implicite.

În cazul în care se transmit parametrii efectivi, se vor folosii valorile acestora.

Priviţi exemplul de mai jos:

```
#include <iostream.h>
#include <conio.h>
void afisare(int a,int b=1,int c=2);  //b si c sunt parametrii cu valori implicite

void main()
{//am specificat valori pentru toti parametrii, functia le va utiliza pe cele specificate ca
//parametrii efectivi
  afisare(3,4,5);
//am specificat valori doar pentru a si b, pentru c se va folosi valoarea implicita
  afisare(5,6);
//am specificat valori doar pentru a , pentru b si c se vor folosi valoarile implicite
  afisare(7);
 getch();
}

void afisare(int a,int b,int c)          //nu se specifică de două ori valoarea implicită
{cout<<"a="<<a<<" b="<<b<<" c="<<c<<endl;
}
```

În exemplul de mai sus antetul funcţiei conţine doi parametri cu valori implicite b=1 şi c=2.

La apelul **afisare(3,4,5);** s-au transmis toţi cei 3 parametrii iar funcţia a folosit valorile transmise.

La apelul **afisare(5,6);** s-au transmis 2 parametrii iar funcţia a folosit valorile transmise pentru a şi b şi valoarea implicită pentru c.

La apelul **afisare(7);** s-a transmis doar valoarea parametrului a iar pentru b şi c s-au folosit valorile implicite.

2. Parametrii transmişi prin referinţă

De multe ori este necesar să modificăm în interiorul unei funcţii valorile unei variabile externe funcţiei. În acest scop se folosesc parametrii transmişi prin referinţă.

Prin urmare rolul parametrilor transmişi prin referinţă este ca la revenirea din funcţie, variabila transmisă, să reţină valoarea modificată în timpul execuţiei funcţieie.

Pentru ca un parametru să fie transmis prin referinţă **numele lui trebuie să fie precedat de caracterul ampersand &**, la declararea parametrilor, în antetul funcţiei.

Caracterul ampersand specifică faptul că parametrul este transmis prin referinţă şi nu prin valoare.

Exemplu:

```
int  f(int &a, int &b);
            ↕         ↕
     f (    x   ,    y  );
```

Atunci când o variabilă este transmisă prin referinţă, **nu** se face o copie a ei pe segmentul de stivă ci **se reţine pe segmentul de stivă adresa variabilei însăşi**. Cu alte cuvinte modificările se efectuează la adresa reţinută, prin urmare chiar asupra variabilei transmise.

O funcţie poate avea unul sau mai mulţi parametrii transmişi prin referinţă.

Parametrii efectivi corespunzători parametrilor transmişi prin referinţă trebuie să fie nume de variabile.

Să analizăm următorul exemplu:

```
#include <iostream.h>       //PARAMETRII TRANSMISI PRIN REFERINTA
#include <conio.h>
void f(int &a,int b);  //parametrul a este transmis prin referinta
                       //parametrul b este transmis prin valoare
void main()
{ int x=1,y=2;
  cout<<"x="<<x<<" y="<<y<<endl;          //x si y inainte de apelatrea functiei
  f(x,y);
  cout<<"x="<<x<<" y="<<y<<endl;          //x si y dupa  apelatrea functiei
  getch();
}

void f(int &a,int b)
{a=2*a;                                   //modificarea valorilor prametrilor a si b
 b=2*b;
 cout<<"a="<<a<<" b="<<b<<endl;           // a si b in functie
}
```

```
Rezultatul rulării:

x=1 y=2
a=2 b=4
a=2 b=2
```

Parametrul **a este transmis prin referinţă** iar **parametrul b este transmis prin valoare.**

Iniţial **x=1 şi y=2**, în cadrul funcţiei **a devine 2** iar **b devine 4**.

Deoarece **a** este un parametru transmis prin referinţă valoarea lui este transmisă în **x**. Întrucât **b** este parametru transmis prin valoare, **y nu** preia valoarea modificată a lui **b**

1.4. Probleme rezolvate

1. Se citeşte o literă şi un număr întreg n. Să se calculeze:

-pentru litera a $S1=1^2 + 3^2 + 5^2 + ... + (2n-1)^2$

-pentru litera b   S2=1 + 1*2+1*2*3 + ... +1*2*3...n

```
#include <iostream.h>       //SUME
#include <conio.h>
#include <math.h>                //conţine funcţiile matematice
long s1(int n);
long s2(int n);

void main()
{ int n;  char lit;                   //n-nr. elemente, lit-litera citita
  cout<<"n="; cin>>n;
  cout<<"lit(a sau b)="; cin>>lit;
  if(lit=='a')
    cout<<"S1="<<s1(n);
  else
    cout<<"s2="<<s2(n);
  getch();
}
long s1(int n)            //funcţia s1 are un parametru transmis prin valoare
{int i; long s=0;
 for(i=1;i<=n;i++)
   s+=pow(2*i-1,2);
 return s;                //funcţia s1 returneaza  valoarea calculată
}
long s2(int n)            //funcţia s2 are un parametru transmis prin valoare
{int i;
 long s=0,p=1;
 for(i=1;i<=n;i++)
   {p*=i;
    s+=p;   }
 return s;                //funcţia s2 returnează valoarea calculată
}
```

2. Se citeşte un număr natural n. Să se numere câte cifre pare şi câte cifre impare are numărul.

Vom realiza o funcţie *cifre* pentru numărarea cifrelor pare şi impare, căreia îi transmitem 3 parametrii.
n – numărul citit, **parametru transmis prin valoare**
c_pare-numărul cifrelor pare, **parametru transmis prin referinţă**, care va transmite numarul cifrelor pare ale numarului, din funcţie înspre programul apelant.
c_impare-numărul cifrelor impare, **parametru transmis prin referinţă**, care va transmite numarul cifrelor impare ale numărului, din funcţie înspre programul apelant.
Funcţia *cifre* va returna programului principal numărul cifrelor pare şi numărul cifrelor impare, prin intermediul parametrilor transmişi prin referinţă.

```
#include <iostream.h>          //NR. CIFRE PARE SI IMPARE
#include <conio.h>
void cifre(int n, int &x, int &y);     //declararea functiei cifre

void main()
{long n;
 int c_pare=0,c_impare=0;     //c_pare-nr. cifrelor pare,c_impare-nr. Cifrelor impare
 cout<<"n="; cin>>n;
 cifre(n,c_pare,c_impare);
 cout<<"nr. cifre pare="<<c_pare<<endl;
 cout<<"nr. cifre impare="<<c_impare<<endl;
 getch();
}

void cifre(int n, int &x, int &y)     //definirea functiei cifre
{ while (n)                    //cat timp numarul mai are cifre
    {if ((n%10)%2==0)          //verifica daca ultima cifra a numarului este para
      x++;                     //incrementează numărul cifrelor pare
     else
      y++;                     //incrementeaza numarul cifrelor impare
     n/=10;                    //elimina ultima cifra
    }
}
```

3. Se citeşte un număr natural n. Să se verifice şi să se afişeze dacă este un număr prim, altfel să se afişeze descompunerea în factori primi.

Definim o funcţie *prim*, care primeşte ca parametru transmis prin valoare numărul n şi returnează 1 dacă numarul este prim şi 0 dacă numarul nu este prim.
Definim de asemenea o funcţie *factori*, funcţie fără tip, care primeşte ca parametru numărul n şi afişează factorii primi şi ordinul lor de multiplicitate.

```
#include <iostream.h>          //PRIM si FACTORI  PRIMI
#include <conio.h>
int prim(int n);         //declararea funcţiei prim, n parametru transmis prin valoare
void factori(int n);     //declararea funcţiei factori, n parametru transmis prin valoare

void main()
{long n;
 cout<<"n="; cin>>n;
 if(prim(n))             //apelul funcţiei prim ca expresie logică
    cout<<n<<" este numar prim";
 else
    {cout<<"descompunerea in factori primi: ";
     factori(n);         //apelul funcţiei factori
    }
 getch();
}

int prim(int n)          //definirea functiei prim
{int i;
 if (n==0 || n==0)
   return 0;
 for(i=2;i<=n/2;i++)
    if(n%i==0)
      return 0;
 return 1;
}

void factori(int n)      //definirea functiei factori
{int i=2,k;
 while (n!=1)
   {k=0;
    while(n%i==0)
      {k++;
       n/=i;
      }
    if (k)
      cout<<i<<"^"<<k<<" ";
    i++;
   }
}
```

4. Se citeşte un număr natural n. Să se descompună în sumă de numere fibonacci.

Definim funcţia *max_f*, care determină cel mai mare număr fibonacci mai mic sau egal cu un număr dat. Funcţia primeşte ca parametru transmis prin valoare numarul dat.

Definim funcţia *fibo*, care apelează funcţia *max_f* şi astfel determină *max*, cel mai mare număr fibonacci mai mic sau egal cu numărul dat, apoi scade din numărul dat, numarul *max*.

Se reia algoritmul atâta timp cât numărul dat este diferit de zero.

```
//DESCOMPUNEREA UNUI NUMAR IN SUMA DE NUMERE FIBONACCI
#include <conio.h>
#include <iostream.h>

void fibo(unsigned long m);                  // declararea funcţiei fibo
unsigned long max_f(unsigned long nr);  //declararea functiai max_f

void main()
{unsigned long n;
 clrscr();
 cout<<"n=";cin>>n;
 cout<<n<<" = ";
 fibo(n);                                //apelul funcţiei fibo
 getch();
}

unsigned long max_f(unsigned long nr)    //definirea functiei max_f
{unsigned long a,b,c;
 a=0;b=1;
 while(b<=nr)
  {c=a+b;
   a=b;
   b=c;
  }
 return a;                               //valoarea returnată de functia max_f
}

void fibo(unsigned long m)               //definirea functiei fibo
{int max;
 while (m)
  {max=max_f(m);                         //apelul functiei max_f din interiorul functiei fibo
   cout<<"+"<<max;
   m-=max;
  }
}
```

5. Se citeşte un număr natural n în baza 10 şi b un număr natural 2<=b<=9 reprezentând o bază de numeraţie. Să se transforme numărul n în baza b.

Vom scrie o funcţie *conv*, de tip void, cu doi parametrii transmişi prin valoare, respectiv n şi b.

Rezultatul conversiei îl vom memora într-un tablou unidimensional t.

În finalul funcţiei afişăm conţinutul tabloului în ordinea inversă construirii lui.

```
#include <conio.h>              //CONV 10->b
#define MAX 20
void conv(int n, int b);        //antetul funcţiei

void main()
{int n,b;
 cout<<"n="; cin>>n;            //numărul în baza 10
 cout<<"b="; cin>>b;            //baza de numeraţie
 conv(n,b);                     //apelul funcţiei de conversie
 getch();
}

void conv(int n, int b)         //definirea funcţiei conv
{int i=0,j,t[MAX];
 while (n)
  {t[i++]=n%b;                  //restul împărţirii lui n la b se depune în tabloul t
   n=n/b;                       //n devine câtul împărţirii lui n la b
  }
 for(j=i-1;j>=0;j--)            //afisarea numărului în baza b
   cout<<t[j];
}
```

6. Se citeşte un număr natural n şi apoi n cifre reprezentând un număr în baza b, 1<=b<=9. Să se transforme numărul dat din baza b în baza 10.

Vom realiza două funcţii. Funcţia citire, pentru citirea numărului în baza b şi a bazei. Funcţia va avea 3 parametrii:

- n, numărul de cifre, parametru transmis prin referinţă
- t- tablou unidimensional care va conţine cifrele numărului, acest parametru se transmite prin valoare deşi rezultatul citirii trebuie transmis şi celorlalte funcţii. Întrucât numele unui tablou este adresa primului octet din tablou. Fiind adresă, modificările se realizează în funcţie chiar asupra tabloului la adresa la care este memorat tabloul în segmentul de date.
- b-baza de numeraţie, parametru transmis prin referinţă

Funcţia conv, converteşte numărul din baza b în baza 10. Returnează numărul în baza 10. Funcţia are trei parametrii transmişi prin valoare: n, t, b, cu semnificaţia de mai sus.

```
#include <iostream.h>            //CONV b->10
#include <conio.h>
#include <math.h>
#define MAX 20
void citire(int &n, int t[MAX], int &b);
int conv(int n, int t[MAX], int b);

void main()
{int n,b,t[MAX],nr;    //n-nr.de cifre,b-baza,t-conţine numarul in baxa b
 citire(n,t,b);                              //apelul funcţiei citire
 nr=conv(n,t,b);                             //apelul funcţiei conv
 cout<<"nr. in baza 10= "<<nr;
 getch();
}

void citire(int &n, int t[MAX], int &b)     //citeste datele de intrare
{int i;
 cout<<"baza=";cin>>b;
 cout<<"n=";cin>>n;
 cout<<"introduceti "<<n<<" cifre in baza "<<b<<":"<<endl;
 for(i=n-1;i>=0;i--)
   {cin>>t[i];}
}

int conv(int n,int t[MAX],int b)             //converteste numarul in baza 10
{int i,nr=0;
 for(i=n-1;i>=0;i--)
   nr+=t[i]*pow(b,i);
 return nr;
}
```

1.5. Evaluare

TESTUL 1

1. Ce înţelegeţi prin *declararea unei funcţii*, dar prin *definirea unei funcţii*?
2. Descrieţi antetul unei funcţii.
3. Ce va afişa următorul program?

```
#include <conio.h>
#include <iostream.h>
int a=5,b=10;
void f(int &a, int b);
void main()
{cout<<a<<" "<<b<<endl;
 f(a,b);
 cout<<a<<" "<<b;
 getch();
}
void f(int &a, int b)
{a=a+10;
 b=b+10;
 cout<<a<<" "<<b<<endl;
}
```

a.)	b.)	c.)	d.)
5 10 15 20 5 10	5 10 15 20 5 20	5 10 15 20 15 10	5 10 15 20 15 20

4. Scrieţi o funcţie pentru calculul celui mai mare divizor comun a două numere naturale. Funcţia va avea ca parametri cele două numere şi va returna cel mai mare divizor comun.

TESTUL 2

1. Ce înţelegeţi prin *parametrii formali*, dar prin *parametrii efectivi*?
2. Ce înţelegeţi prin *parametrii transmişi prin valoare*, dar prin *parametrii transmişi prin referinţă*?
3. Ce va afişa următorul program?

```
#include <conio.h>
#include <iostream.h>
int a=5,b=10;
void f(int &a, int b);
void main()
{cout<<a<<" "<<b<<endl;
 f(a,b);
 cout<<a<<" "<<b;
 getch();
}
void f(int &b, int a)
{a=a+10;
 b=b+10;
 cout<<a<<" "<<b<<endl;
}
```

a.) 5 10 b.) 5 10 c.) 5 10 d.) 5 10

```
15 20          20 15          20 15          15 20
15 10          15 10           5 10          10 15
```

4. Scrieţi o funcţie care verifică dacă un număr este palindrom. Funcţia va primi ca parametru numărul şi va returna 1 dacă numărul este palindrom şi zero în caz contrar.
5. Să se afişeze numerele cuprinse între 100 şi 1000 care sunt pătrate perfecte şi sunt prime cu un număr k citit.

TESTUL 3

1. Care este domeniul de vizibilitate al unei variabile declarate la începutul funcţiei *main*, dar valoarea ei iniţială?
2. Care este domeniul de vizibilitate al unei variabile declarate la începutul programului, înaintea oricarei funcţii? Dar valoarea ei?
3. Ce va afişa următorul program?

```
#include <conio.h>
#include <iostream.h>
int a=2,b=4;
void f(int &a, int b);
void main()
{cout<<a<<" "<<b<<endl;
 f(a,b);
 cout<<a<<" "<<b<<endl;
 getch();
}
void f(int &a, int b)
{int aux;
 aux=a;
 a=b;
 b=aux;
 cout<<a<<" "<<b<<endl;
}
```

```
a.) 2 4     b.) 2 4     c.) 2 4
    4 2         4 2         2 4
    4 2         4 4         4 2
```

4. Se citesc n numere naturale. Se cere să se calculeze suma numerelor care au toate cifrele impare.

1.6. Probleme propuse

1. Să se afişeze numerele naturale mai mari decât 100 şi mai mici decât 500 care au toate cifrele distincte utilizând o funcţie care primeşte ca parametru un număr şi returnează 1 dacă are toate cifrele distincte şi 0 dacă nu sunt distincte.

2. Se citesc n numere. Să se afişeze numerele obţinute prin inversarea cifrelor.

3. Să se afişeze toate numerele până la 1000 al căror pătrat se termină cu cifra n.

4. Se citesc numere naturale până la întâlnirea lui 0. Să se afişeze inversele numerelor citite, pentru care media aritmetică a cifrelor este strict mai mică decât 5.

5. Se citesc n numere naturale. Să se calculeze suma numerelor care au toate cifrele pare. Dacă nu există nici un astfel de număr se va afişa un mesaj.

6. Calculaţi suma şi produsul divizorilor proprii ai unui număr citit.

7. Fiind dat un număr natural n, să se afişeze toţi divizorii săi şi media aritmetică a divizorilor săi cuprinşi între două valori citite a şi b sau un mesaj, dacă nu are divizori între aceste valori.

8. Fiind dat un număr natural n, afişaţi divizorii săi şi număraţi câţi dintre aceştia sunt numere prime.

9. Să se genereze toate numerele prime mai mici sau egale cu un număr n citit de la tastatură.

10. Scrieţi un program care calculează suma a două fracţii ordinale. Afişarea se va face după simplificarea fracţiilor.

11. Se citeşte un număr natural n. Să se stabilească dacă acesta este un termen al şirului lui Fibonacci.

12. Se citeşte un număr natural n diferit de 0. Să se scrie toate tripletele de numere pitagoreice ($a^2+b^2=c^2$), mai mici decât n, nenule.

13. Se citeşte un număr natural n, în baza 10. Să se transforme numărul în baza 16.

14. Se citesc n caractere reprezentând un număr în baza 16. Să se transforme în baza 10.

Răspuns la testele grilă:
TESTUL 1 : c
TESTUL 2 : b
TESTUL 3 : b