7.2. TRANSFERUL PARAMETRILOR UNEI FUNCŢII

Funcţiile comunică între ele prin argumente (parametrii).

Există următoarele moduri de transfer (transmitere) a parametrilor către funcţiile apelate:
- Transfer prin valoare;
- Transfer prin pointeri;
- Transfer prin referinţă.

7.2.1. TRANFERUL PARAMETRILOR PRIN VALOARE

În exemplele anterioare, parametrii de la funcţia apelantă la funcţia apelată au fost transmişi ***prin valoare***. De la programul apelant către funcţia apelată, prin apel, se transmit valorile partametrilor efectivi, reali. Aceste valori vor fi atribuite, la apel, parametrilor formali. Deci procedeul de transmitere a parametrilor prin valoare constă *în încărcarea valorii parametrilor efectivi în zona de memorie a parametrilor formali (în stivă)*. La apelul unei funcţii, parametrii reali trebuie să corespundă - ca ordine şi tip - cu cei formali.

Exerciţiu: Să se scrie următorul program (care ilustrează legătura dintre pointeri şi vectori) şi să se urmărească rezultatele execuţiei acestuia.

```
void f1(float intr,int nr)// intr, nr - parametri formali
{
for (int i=0; i<nr;i++) intr *= 2.0;
cout<<"Val. Param. intr="<<intr<<'\n';// intr=12
}
void main()
{
float data=1.5; f1(data,3);
// apelul funcţiei f1; data, 3 sunt parametri efectivi
cout<<"data="<<data<<'\n';
// data=1.5 (nemodificat)
}
```

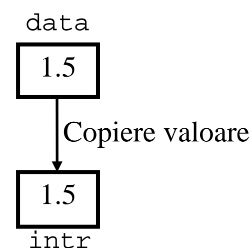


Figura 7.1. Transmiterea prin valoare

Fiecare argument efectiv utilizat la apelul funcţiei este evaluat, iar valoarea este atribuită parametrului formal corespunzător. În interiorul funcţiei, o copie locală a acestei valori va fi memorată în parametrul formal. O modificare a valorii parametrului formal în interiorul funcţiei (printr-o operaţie din corpul funcţiei), nu va modifica valoarea parametrului efectiv, ci doar valoarea parametrului formal, deci a copiei locale a parametrului efectiv (figura 6.1.). Faptul că variabila din programul apelant (parametrul efectiv) şi parametrul formal sunt obiecte distincte, poate constitui un mijloc util de protecţie. Astfel, în funcţia `f1`, valoarea parametrului formal intr este modificată (alterată) prin instrucţiunea ciclică for. În schimb, valoarea parametrului efectiv (data) din funcţia apelantă, rămâne nemodificată.

În cazul transmiterii parametrilor prin valoare, parametrii efectivi pot fi chiar expresii. Acestea sunt evaluate, iar valoarea lor va iniţializa, la apel, parametrii formali.

Exemplu:

```
double psi(int a, double b)
{
if (a > 0)  return a*b*2;
else  return -a+3*b; }
void main()
{ int x=4; double y=12.6, z; z=psi ( 3*x+9, y-5) + 28;
cout<<"z="<<z<<'\n';    }
```

Transferul valorii este însoţit de eventuale conversii de tip. Aceste conversii sunt realizate automat de compilator, în urma verificării apelului de funcţie, pe baza informaţiilor despre funcţie, sau sunt conversii explicite, specificate de programator, prin operatorul "cast".

Exemplu:

```
float f1(double, int);
void main()
{
int a, b; float g=f1(a, b);     // conversie automată: int a -> double a
float h=f1( (double) a, b);     // conversie explicită
}
```

Limbajul ***C*** este numit ***limbajul apelului prin valoare***, deoarece, de fiecare dată când o funcţie transmite argumente unei funcţii apelate, este transmisă, de fapt, o copie a parametrilor efectivi. În acest mod, dacă valoarea parametrilor formali (iniţializaţi cu valorile parametrilor efectivi) se modifică în interiorul funcţiei apelate, valorile parametrilor efectivi din funcţia apelantă nu vor fi afectate.

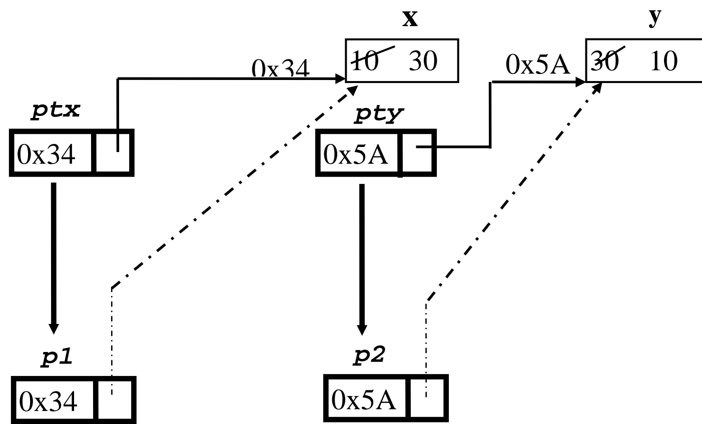
7.2.2. TRANSFERUL PARAMETRILOR PRIN POINTERI

În unele cazuri, parametrii transmişi unei funcţii pot fi pointeri (variabile care conţin adrese). În aceste cazuri, parametrii formali ai funcţiei apelate vor fi iniţializaţi cu valorile parametrilor efectivi, deci cu valorile unor adrese. Astfel, *funcţia apelată* ***poate modifica*** *conţinutul locaţiilor spre care pointează argumentele (pointerii).*

Exerciţiu: Să se citească 2 valori întregi şi să se interschimbe cele două valori. Se va folosi o funcţie de interschimbare.

```
#include <iostream.h>
void schimbă(int *, int *);            //prototipul functiei schimba
void main()
{
int x, y, *ptx, *pty;   ptx=&x;     pty=&y;
cout<<"x=";cin>>x;cout<<"y=";cin>>y;cout<<"x="<<x;cout<<"y="<<y<<'\n';
cout<<"Adr. lui x:"<<&x<<" Val lui x:"<<x<<'\n';
cout<<"Adr.lui y:"<<&y<<" Val y:"<<y<<'\n'; cout<<"Val. lui ptx:"<<ptx;
cout<<" Cont. locaţiei spre care pointează ptx:"<<*ptx<<'\n';
cout<<"Val. lui pty:"<<pty;
cout<<"Cont. locaţiei spre care pointează pty:"<<*pty;
schimba(ptx, pty);
// SAU: schimba(&x, &y);
cout<<"Adr. lui x:"<<&x<<" %x Val lui x: %d\n", &x, x);
cout<<"Adr. y:"<<&y<<" Val lui y:"<<y<<'\n';cout<<"Val. lui ptx:"<<ptx;
cout<<" Cont. locaţiei spre care pointează ptx:"<<*ptx<<'\n';
cout<<"Val. lui pty:"<<pty;
cout<<" Cont. locaţiei spre care pointează pty:"<<*pty<<'\n';
}
void schimbă( int *p1, int *p2)
{
cout<<"Val. lui p1:"<<p1;
cout<<" Cont. locaţiei spre care pointează p1:"<<*p1<<'\n';
cout<<"Val. lui p2:"<<p2;
cout<<" Cont. locaţiei spre care pointează p2:"<<*p2<<'\n';
int t = *p1;        // int *t ; t=p1;
*p2=*p1;    *p1=t;
cout<<"Val. lui p1:"<<p1;
cout<<" Cont. locaţiei spre care pointează p1:"<<*p1<<'\n';
cout<<"Val. lui p2:"<<p2;
cout<<" Cont. locaţiei spre care pointează p2:"<<*p2<<'\n';
}
```

Dacă parametrii funcţiei `schimbă` ar fi fost transmişi prin valoare, această funcţie ar fi interschimbat copiile parametrilor formali, iar în funcţia `main` modificările asupra parametrilor transmişi nu s-ar fi păstrat. În figura 6.2. sunt prezentate mecanismul de transmitere a parametrilor (prin pointeri) şi modificările efectuate asupra lor de către funcţia `schimbă`.



Parametrii formali `p1` şi `p2`, la apelul funcţiei `schimbă`, primesc valorile parametrilor efectivi `ptx` şi `pty`, care reprezintă adresele variabilelor `x` şi `y`. Astfel, variabilele pointer `p1` şi `ptx`, respectiv `p2` şi `pty` pointează către `x` şi `y`. Modificările asupra valorilor variabilelor `x` şi `y` realizate în corpul funcţiei `schimbă`, se păstrează şi în funcţia `main`.

Figura 7.2. Transmiterea parametrilor unei funcţii prin pointeri

Exerciţiu: Să se scrie următorul program şi să se urmărească rezultatele execuţiei acestuia.

```
#include <iostream.h>
double omega (long *k)
{
cout<<"k=", k);
// k conţine adr. lui i
cout<<"*k=";
cout<<k<<'\n';
// *k = 35001
double s=2+(*k)-3;
// s = 35000
cout<<"s="<<s<<'\n';
*k+=17; // *k = 35017
cout<<"*k="<<*k;
cout<<'\n';
return s;   }

void main()
{
long i = 35001; double w;
cout<<"i="<<i;cout<<"Adr.lui i:"<<&i<<'\n';
w=omega(&i); cout<<"i="<<i<< w="<<w<<'\n';  // i = 350017 w = 35000
}
```

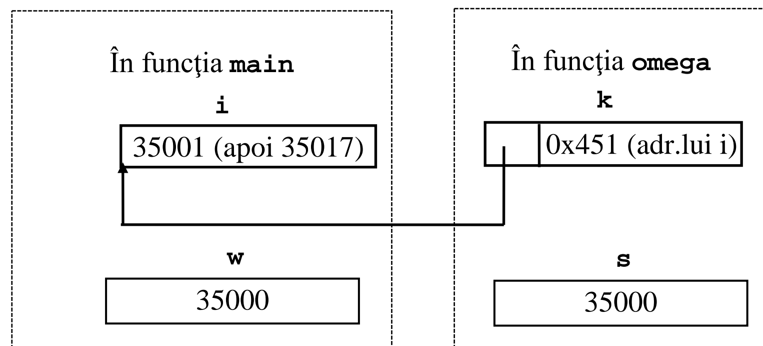


Figura 7.3. Transferul parametrilor prin pointeri

7.2.2.1. Funcţii care returnează pointeri

Valoarea returnată de o funcţie poate fi pointer, aşa cum se observă în exemplul următor:

Exemplu:

```
#include <iostream.h>
double *f (double *w, int k)
{   // w conţine adr. de început a vectorului a
cout<<"w="<<w<<" *w="<<*w<<'\n';       // w= adr. lui a ;*w = a[0]=10
return w+=k;
/*incrementeaza pointerului w cu 2(val. lui k); deci w pointează către elementul de indice 2 din vectorul a*/
}
void main()
{double a[10]={10,1,2,3,4,5,6,7,8,9}; int i=2;
cout<<"Adr. lui a este:"<<a;
double *pa=a;        // double *pa; pa=a;
cout<<"pa="<<pa<<'\n'     // pointerul pa conţine adresa de început a tabloului a
```

```
// a[i] = * (a + i)
// &a[i] = a + i
pa=f(a,i); cout<<"pa="<<pa<<" *pa="<<*pa<<'\n';
// pa conţine adr. lui a[2], adica adr. a + 2 * sizeof(double);
*pa=2;
}
```

7.2.3. TRANSFERUL PARAMETRILOR PRIN REFERINŢĂ

În acest mod de transmitere a parametrilor, unui parametru formal i se poate asocia (atribui) chiar obiectul parametrului efectiv. Astfel, parametrul efectiv poate fi modificat direct prin operaţiile din corpul funcţiei apelate.

În exemplul următor definim variabila `br`, ***variabilă referinţă*** către variabila `b`. Variabilele *b* şi *br* se găsesc, în memorie, la *aceeaşi* adresă şi sunt variabile sinonime.

Exemplu:

```
#include <stdio.h>
#include <iostream.h>
void main()
{
int b,c;
int &br=b;   //br referinţă la altă variabilă (b)
br=7;
cout<<"b="<<b<<'\n';      //b=7
cout<<"br="<<br<<'\n';    //br=7
cout<<"Adr. br este:"<<&br;          //Adr. br este:0xfff4
printf("Adr. b este:"<<&b<<'\n';     //Adr. b este:0xfff4
b=12; cout<<"br="<<br<<'\n';         //br=12
cout<<"b="<<b<<'\n';                 //b=12
c=br; cout<<"c="<<c<<'\n';           //c=12
}
```

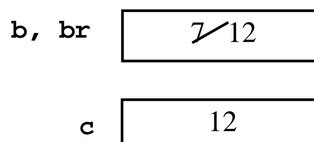


Figura 7.4. Variabilele referinţă `b, br`

Exemplul devenit clasic pentru explicarea apelului prin referinţă este cel al funcţiei de permutare (interschimbare) a două variabile.

Fie funcţia *schimb* definită astfel:

```
void schimb (double x, double y)
{ double t=x; x=y; y=t; }

void main()
{ double a=4.7, b=9.7;
. . . . . . . . . . .
schimb(a, b);            // apel funcţie
. . . . . . . . . . .  }
```

Parametri funcţiei `schimb` sunt transmişi prin valoare: parametrilor formali `x`, `y` li se atribuie (la apel) valorile parametrilor efectivi `a`, `b`. Funcţia `schimb` permută valorile parametrilor formali `x` şi `y`, dar permutarea nu are efect asupra parametrilor efectivi `a` şi `b`.

Pentru ca funcţia de interschimbare să poată permuta valorile parametrilor efectivi, în limbajul C/C++ parametrii formali trebuie să fie *pointeri către valorile care trebuie inter*schimbate:

```
void pschimb(double *x, double *y)
{ int t=*x; *x=*y; *y=t; }
void main()

{ double a=4.7, b=9.7;
. . . . . . . . . . .
pschimb(&a, &b);         // apel funcţie
/* SAU:
```

Se atribuie pointerilor `x` şi `y` valorile pointerilor `pa`, `pb`, deci adresele variabilelor `a` şi `b`. Funcţia `pschimb` permută valorile spre care pointează pointerii `x` şi `y`, deci valorile lui `a` şi `b` (figura 6.5).

```
double *pa, *pb;
pa=&a; pb=&b;
pschimb(pa, pb);*/
. . . . . . . . . . .   }
```

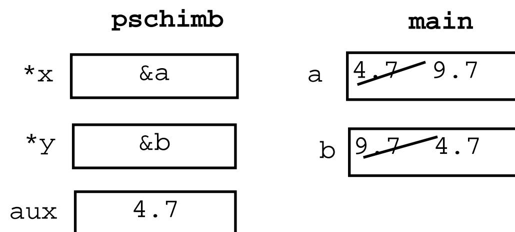


Figura 7.5. Transferul parametrilor prin pointeri

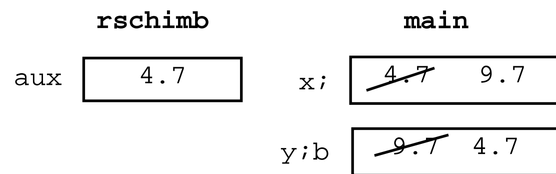


Figura 7.6. Transferul parametrilor prin referinţă

În limbajul C++ acceaşi funcţie de permutare se poate defini cu parametri formali de *tip referinţă*.

```
void rschimb(double &x, double &y)
{ int t=x; x=y; y=t; }
void main()
{ double a=4.7, b=9.7;
. . . . . . . . . . . . . . .
rschimb(a, b);            // apel funcţie
. . . . . . . . . . . . . . .   }
```

În acest caz, x şi y sunt sinonime cu a şi b (nume diferite pentru aceleaşi grupuri de locaţii de memorie). Interschimbarea valorilor variabilelor de x şi y înseamnă interschimbarea valorilor variabilelor a şi b (fig. 6.6.).

Comparând funcţiile `pschimb` şi `rschimb`, se observă că diferenţa dintre ele constă în modul de declarare a parametrilor formali. În cazul funcţiei `pschimb` parametrii formali sunt *pointeri* (de tip *double), în cazul funcţiei `rschimb`, parametrii formali sunt *referinţe* către date de tip double. În cazul transferului parametrilor prin referinţă, parametrii formali ai funcţiei referă aceleaşi locaţii de memorie (sunt sinonime pentru) parametrii efectivi.

Comparând cele trei moduri de transmitere a parametrilor către o funcţie, se poate observa:
1. La apelul *prin valoare* transferul datelor este *unidirecţional*, adică valorile se transferă numai de la funcţia apelantă către cea apelată. La apelul *prin referinţă* transferul datelor este *bidirecţional*, deoarece o modificare a parametrilor formali determină modificarea parametrilor efectivi, care sunt sinonime (au nume diferite, dar referă aceleaşi locaţii de memorie).
2. La transmiterea parametrilor *prin valoare*, ca parametrii efectivi pot apare *expresii* sau *nume de variabile*. La transmiterea parametrilor *prin referinţă*, ca parametri efectivi nu pot apare expresii, ci *doar nume de variabile*. La transmiterea parametrilor *prin pointeri*, ca parametri efectivi pot apare expresii de pointeri.
3. Transmiterea parametrilor unei funcţii prin referinţă este specifică limbajului C++.
4. Limbajul C este numit limbajul apelului prin valoare. Apelul poate deveni, însă, *apel prin referinţă* în cazul variabilelor simple, folosind pointeri, sau aşa cum vom vedea în paragraful 6.4., în cazul în care parametru efectiv este un tablou.
5. În limbajul C++ se poate alege, pentru fiecare parametru, tipul de apel: prin valoare sau prin referinţă, aşa cum ilustrează exemplele următoare:

Exerciţiu: Să se scrie următorul program şi să se urmărească rezultatele execuţiei acestuia.

```
#include <iostream.h>
#include <stdio.h>

double func(int a, double b, double *c, double &d)
{cout<<"********************** func ****************\n";
```

```
cout<<"a="<<a<<" b="<<b;                 //a=7 (a=t prin val); b=21 (b=u prin val)
cout<<" c="<<c<<" *c="<<*c<<'\n';        // c=ffe(c=w=&u) *c=21
cout<<" d="<<d;                          //d=17
cout<<"Adr d="<<&d<<'\n';                //Adr d=ffe6 (adr d=adr v)
a+=2; cout<<"a="<<a<<'\n';                   //a=9
d=2*a+b; cout<<"d="<<d<<'\n';                   //d=39
/*c=500;
cout<<" c="<<c<<" *c="<<*c<<'\n';     // c=ffe(c=w=&u) *c=21*/
cout<<"********************** func ****************\n";
return b+(*c);
}

void main()
{cout<<"\n\n \n MAIN MAIN";
int t=7; double u=12, v=17, *w, z; cout<<"u="<<u<<'\n';      //u=12
w=&u; *w=21;
cout<<"t="<<t<<" u="<<u<<" v="<<v;   //t=7 u=12 v=17 *w=21
cout<<" *w="<<*w<<" u="<<u<<'\n';    //*w=21 u=21
printf("w=%x Adr. u=%x\n",w,&u);     //w=ffee Adr. u=ffee
printf("v=%f Adr v=%x\n",v,&v);      //v=17.000 Adr v=ffe6
z=func(t,u,w, v);
cout<<"t="<<t<<"u="<<u<<"v="<<v;     //t=7 u=21 (NESCHIMBATI) v=39 (v=d)
cout<<" *w="<<*w<<" z="<<z<<'\n';    //*w=21 w=ffee z=42
printf(" w=%x\n",w);
}
```

Exemplul ilustrează următoarele probleme:
La apelul funcţiei `func`, parametrii `t` şi `u` sunt transmişi prin valoare, deci valorile lor vor fi atribuite parametrilor formali `a` şi `b`. Orice modificare a parametrilor formali `a` şi `b`, în funcţia `func`, nu va avea efect asupra parametrilor efectivi `t` şi `u`. Al treilea parametru formal al funcţiei `func` este transmis prin pointeri, deci `c` este de tip `double *` (pointer către un real), sau `*c` este de tip `double`.

La apelul funcţiei, valoarea pointerului `w` (adresa lui u : `w=&u`) este atribuită pointerului `c`. Deci pointerii `w` şi `c` conţin aceeaşi adresă, pointând către un real. Dacă s-ar modifica valoarea spre care pointează `c` în `func` (vezi instrucţiunile din comentariu `*c=500`), această modificare ar fi reflectată şi în funcţia apelantă, deoarece pointerul `w` are acelaşi conţinut ca şi pointerul `c`, deci pointează către aceeaşi locaţie de memorie. Parametrul formal `d` se transmite prin referinţă, deci, în momentul apelului, `d` şi `v` devin similare (`d` şi `v` sunt memorate la aceeaşi adresă). Modificarea valorii variabilei `d` în `func` se reflectă, deci, şi asupra parametrului efectiv din funcţia `main`.

Exerciţiu: Să se scrie următorul program (care ilustrează legătura dintre pointeri şi vectori) şi să se urmărească rezultatele execuţiei acestuia.

```
#include <iostream.h>
#include <stdio.h>
double omega(long &k)
{printf("Adr k=%x Val k=%ld\n",&k,k);          //Adr k=fff2 Val k=200001
double s=2+k-3;cout<<"s="<<s<<'\n';            //s=200000
k+=17;printf("Adr k=%x Val k=%ld\n",&k,k);     //Adr k=fff2 Val k=200018
return s;
}
void main()
{long a=200001;
printf("Adr a=%x Val a=%ld\n",&a,a);        //Adr a=fff2 Val a=200001
double w=omega(a); cout<<"w="<<w<<'\n';     //s=200000
}
```

Aşa cum s-a prezentat în paragrafele 2.5.3.2. şi 5.6., modificatorii sunt cuvinte cheie utilizaţi în declaraţii sau definiţii de variabile sau funcţii. Modificatorul de acces **const** poate apare în:

- Declaraţia unei variabile (precede tipul variabilei) restricţionând modificarea valorii datei;
- La declararea variabilelor pointeri definind pointeri constanţi către date neconstante, pointeri neconstanţi către date constante şi pointeri constanţi către date constante.
- În *lista declaraţiilor parametrilor formali ai unei funcţii*, conducând la imposibilitatea de a modifica valoarea parametrului respectiv în corpul funcţiei, ca în exemplul următor:

Exemplu:

```
#include <iostream.h>
#include <stdio.h>
int func(const int &a)
{printf("Adr a=%x Val a=%d\n",&a,a);int b=2*a+1;
//modificarea valorii parametrului a nu este permisă
cout<<"b="<<b<<'\n';return b;}
void main()
{const int c=33;int u;printf("Adr c=%x Val c=%d\n",&c,c);
u=func(c);cout<<"u="<<u<<'\n'; }
```

7.2.4. TRANSFERUL PARAMETRILOR CĂTRE FUNCŢIA main

În situaţiile în care se doreşte transmiterea a unor informaţii (opţiuni, date iniţiale, etc) către un program, *la lansarea în execuţie* a acestuia, este necesară definirea unor parametri către funcţia `main`. Se utilizează trei parametrii speciali: `argc`, `argv` şi `env`. Trebuie inclus headerul **stdarg.h**.
Prototipul funcţiei main cu parametri în linia de comandă este:

main (int argc, char *argv[ ], char *env[ ])

Dacă ***nu*** *se lucrează cu un mediu de programare integrat*, argumentele transmise către funcţia main trebuie editate (specificate) în linia de comandă prin care se lansează în execuţie programul respectiv. Linia de comandă tastată la lansarea în execuţie a programului este formată din grupuri de caractere delimitate de spaţiu sau tab. Fiecare grup este memorat într-un şir de caractere. Dacă se lucrează cu un mediu integrat (de exemplu, BorlandC), selecţia comanzii `Arguments`… din meniul `Run` determină afişarea unei casete de dialog în care utilizatorul poate introduce argumentele funcţiei `main`.

- Adresele de început ale acestor şiruri sunt memorate în tabloul de pointeri `argv[]`, în ordinea în care apar în linia de comandă (`argv[0]` memorează adresa şirului care constituie numele programului, `argv[1]` - adresa primului argument, etc.).
- Parametrul întreg `argc` memorează numărul de elemente din tabloul argv (`argc>=1`).
- Parametrul `env[]` este un tablou de pointeri către şiruri de caractere care pot specifica parametri ai sistemului de operare.

Funcţia `main` poate returna o valoare întreagă. În acest caz în antetul funcţiei se specifică la tipul valorii returnate `int`, sau nu se specifică nimic (implicit, tipul este int), iar în corpul funcţiei apare instrucţiunea *return valoare_intreagă*;. Numărul returnat este transferat sistemului de operare (programul apelant) şi poate fi tratat ca un cod de eroare sau de succes al încheierii execuţiei programului.

Exerciţiu: Să se implementeze un program care afişează argumentele transmise către funcţia main.

```
#include <iostream.h>
#include <stdarg.h>
void main(int argc, char *argv[], char *env[])
{cout<<"Nume program:"<<argv[0]<<'\n';//argv[0] contine numele programului
if(argc==1)
      cout<<"Lipsa argumente!\n";
else
      for (int i=1; i<argc; i++){
            cout<<"Argumentul "<<i<<":"<<argv[i]<<'\n';
}
```