

Detalii de implementare.
Declararea variabilelor,
transmiterea parametrilor
catre subprograme

2018

Declararea variabilelor

- variabile globale -declarate in afara oricarei functii
- variabile locale -declarate intr-un bloc de instructiuni (ex: in corpul unei functii)
- variabile locale statice -precedate de static

- **variabile globale**

 - vizibile in orice functie

 - initializate implicit cu 0

- **variabile locale**

 - vizibile doar in bloc(existente doar in bloc)

 - NU sunt initializate automat

- **variabile locale statice**

 - durata de existenta -tot programul

 - vizibile doar in apelul functiei

 - initializarea - implicit cu 0, la primul apel, pentru oricate apeluri ale functiei

 - la un nou apel al functiei, varibila are valoarea din ultimul apel al functiei

```
int vg;                //initializata cu 0-implicit

void f()
{int vl; cout<<vl;    //valoare nedeterminata
static int vs1;      //initializata cu 0-implicit
static int vs2=0;
vs1++;vs2++;
cout<<vs1<<vs2;
}

int main()
{cout<<vg;
f();                //vs1=1;vs2=1;
f();                //vs1=2;vs2=2;
}
```

Tipuri de date complexe

```
int i;  
int *p;          //adresa de intreg  
int &r=i; //alt nume pentru variabila i (initializare obligatorie)
```

```
int main()  
{ p=&i;          // adresa variabilei i  
  *p=1;         //zona adresata de p  
  r=1;          // r=i=1  
  p=&r;          // adresa zonei referite =adresa lui i  
}
```

Declaratori complecsi

```
prioritate()=prioritate[]>prioritate*
```

```
int v[10]; // vector de intregi
```

```
int **aa; // adresa unei adrese de intreg
```

```
int * va[10]; // vector de adrese de intregi
```

```
int * fp(); //functie care intoarce adresa unui intreg
```

```
int (*pf)(); //adresa unei functii ce intoarce un intreg
```

Alocare dinamica

```
int *p;
```

```
p=new int; /* o zona int e ocupata -intoarce adresa de  
          inceput a zonei*/
```

```
*p=1; // accesarea zonei ocupate
```

```
p=new int(1); // aloca si initializeaza zona
```

```
p=new int[2]; /* ocupa o zona de 2 intregi si intoarce adresa  
              de inceput a zonei */
```

Eliberare zona dinamica

delete p; /* zona ocupata e considerata libera ->poate fi
realocata altei variabile*/

delete [] p; /* zona continua de mai multe elemente va fi
considerata neocupata*/

Obs.1. pot elibera doar zone alocate dinamic

Obs.2. Orice zona alocata dinamic ramane alocata pana
este apelat operatorul delete

Pointeri si referinte care nu modifica zona

```
int a,b;
```

```
const int * p; //pointer care nu modifica zona
```

```
p=&a; //poate primi valori dupa declarare
```

```
a=0; // variabila se poate modifica
```

```
//*p=0; // nu poate modifica variabila adresata
```

```
int * const pc=&a; // pointer constant- trebuie initializat
```

```
*pc=0; // poate modifica zona
```

```
//pc=&b; // nu poate adresa ALTA zona
```

```
const int &rc=a;// trebuie initializata  
a=0; // variabila se poate modifica  
//rc=0; // rc nu poate modifica variabila referita
```

```
int &r=a; /*orice referinta este constanta -nu poate fi un alt  
nume pentru ALTA variabila*/  
r=b; // copiaza valoarea lui b in zona comuna a lui r si a
```

Transfer parametrii

- prin valoare -parametrul formal este o COPIE -pe stiva a parametrului actual (modificarile din functie NU modifica parametrul actual); pot transmite o adresa prin valoare
- prin referinta-parametrul formal este un alt nume al parametrului actual (modificarile din functie se fac in zona de memorie a parametrului actual) ; parametrul actual e obligatoriu o zona de memorie

```
void f(int i, int*p, int &r)
{ i++; // se modifica doar copia
  (*p)++; // se modifica variabila de la adresa p
          /* transmitere tot prin valoare -a adresei*/
  r++; // se modifica parametrul actual
}
```

```
int main()
{int x,y,z;
  x=y=z=0;
  f(x,&y,z);
}
```

```
int f1(){return 2;} // se intoarce o copie
```

```
int * f2(int i, int *p, int &r) //intoarce o adresa
```

```
{int l;
```

```
// return &l; return &i; /* eroare adresa zona de pe stiva*/
```

```
return p; return &r;
```

```
int *pl=new int; return pl;}
```

```
int & f2(int i, int *p, int &r) /*intoarce alt nume pt un int */
```

```
{int l;
```

```
//return l; return i; /* eroare alt nume pt zona de pe stiva*/
```

```
return *p; return r;
```

```
int *pl=new int; return *pl;}
```

Intoarcere rezultate prin parametrii

```
void fv(int copie){cin>>copie;} // citirea se face in copie  
void fr(int &rez){cin>>rez;}
```

```
int main()  
{int v=0;  
fv(v); // copie =0; citeste in copie; v ramane 0  
fr(v); /* rez va fi alt nume pentru v -citirea se va face in v*/  
}
```

```
void f1(int * & rezP)
{rezP=new int;}
```

```
int main()
{int *p=NULL;
f1(p); /* rezP e alt nume pt variabila p
adresa zonei noi se va pune in p */
}
```

```
void f2(int **adRezP)
{*adRezP=new int;}
```

```
int main()
{int *p=NULL;
int **aa;
aa=&p; // adresa variabilei p
f2(aa);/* in zona adresata adica in variabila p se pune
adresa zonei noi*/
}
```


Diferente alocare statica /dinamica

```
int v[10];
```

- v=adresa primului element din vector
- v=pointer constant (nu va adresa alta zona)
- are zona alocata de dimensiune fixa

```
int *p;
```

- trebuie alocata zona ex: p=new int[10];
- pointer variabil ex: p=v
- dimensiune variabila a zonei ex: p=new int[n];

Vectori

```
int v[10];  
// v e adresa primului element  
//v+1 e adresa urmatorului element (int)  
v[0] este *v;  
v[1] este *(v+1);  
v[i] este *(v+i);
```

Transmitere vectori

```
void f1(int v[10]) {v[0]=0;};  
void f2(int *v)    {v[0]=0;};  
void f3(int v[])  {v[0]=0;};  
// transmite (echivalent) adresa primului element  
  
int main()  
{int a[10];  
f1(a); f2(a); f3(a);  
}  
/* modifica valoare de la adresa v (elementele din a) */
```

Matrici

-vector cu 5 elemente de tip vector de 10 int
int m[5][10]; // zona continua de 50 int
m[i][j] este elementul de pe linia i coloana j
m+1 e adresa urmatorului vector de 10 elem
m[i][j] este $*(*(m+i)+j)$

-vector cu 5 elemente de tip pointer la int

```
int *m1[5]; /*zonele nu sunt continue-trebuie alocate*/
```

```
for(int i=0;i<lin;i++) m1[i]= new int [col];
```

m1+1 e adresa urmatorului pointer

m[i][j] este $*(*(m+i)+j)$

- adresa

```
int ** m2;// nu are spatiu alocat
```

```
m2=new int *[lin];
```

```
for (int i=0;i<lin;i++) m2[i]=new int[col];
```

m[i][j] este $*(*(m+i)+j)$

Transmitere matrici

- trebuie specificata ultima dimensiune

(m+1 adresa urmatorului vector de 10 int)

```
void f4(int m[5][10]){}  
void f5(int m[][10]){}  
  
-nu trebuie specificata ultima dimensiune  
(m+1 este adresa urmatorului pointer la int)  
void f6(int *m[5]){}  
void f6(int **m){}
```

Sir sau vector de caractere

```
char v[10];
```

-vector de caractere

```
for(int i=0;i<10;i++) cin>>v[i];
```

-sir de caractere (terminat cu caracterul '\0')

```
cin>>v;
```

```
//ocupa un octet in plus -pt '\0'
```

Citire siruri de caractere

```
cin>>v; // se opreste la primul caracter alb
```

```
cin.get(v,nr,c); /* citeste cel mult nr-1 caractere sau a fost  
                 intalnit delimitatorul c (implicit '\n') */
```

```
cin.get(); /* mai trebuie citit un caracter (delimitatorul) */
```


-biblioteca string -functii ce lucreaza cu siruri sfarsite cu delimitator :

```
strlen(sir), strcpy(dest,sursa), strcmp(sir1,sir2)
```

-copiere

```
char dest[10],sursa[10];
```

```
//dest=sursa; /* ar schimba zona adresata-nu face copierea*/
```

```
strcpy(dest,sursa);
```

-comparare:

```
char s1[10],s2[10];
```

```
s1<s2; // compara adresele de inceput ale vectorilor
```

```
strcmp(s1,s2); // compara lexicografic sirurile (<0),0,(>0)
```

Structuri

```
struct data
```

```
{int zi,luna,an;
```

```
};
```

```
data azi,ieri,*pd; //data este un tip de date
```

```
pd=&azi; // *pd este zona adresata adica azi
```

```
azi.zi=17;
```

```
(*pd).zi=17; /*echivalent cu*/
```

```
pd->zi=17;
```

```
azi=ieri; /* copiaza bit cu bit informatia
```

```
eroare in cazul campurilor alocate dinamic*/
```

Structuri recursive

```
struct nod  
{int inf;  
  nod * a_urm;  
};
```

```
nod el,*p;
```

```
p=&el;          // *p este zona adresata adica el
```

```
el.inf=0;
```

```
(*p).inf=0;    // echivalent cu
```

```
p->inf=0;
```

```
p= new nod ; /* aloca zona pentru un nod
                si intoarce adresa nodului*/
// *p este zona adresata adica nodul alocat

(*p).el =0; // echivalent cu
p->el=0;

delete p; // considera nodul ca fiind zona libera
```

Instructiuni

```
if (cond1) instr1;  
if (cond2) instr2;  
else instr2;
```

/* else se leaga de cel mai apropiat if
cod echivalent cu : */

```
if (cond1) instr1;  
{if (cond2) instr2;  
else instr2;}
```

```
int x=0;
cout<<x++; // se afiseaza 0 si x devine 1
int y=0;
cout << ++y; // y devine 1 si se afiseaza 1
```

evaluarea este DIFERITA de atribuire:

```
int i=0,j=0;
if(i=1) instr1; /* i primeste val 1 , rezultatul atribuirii este 1
                (adevarat) deci executa instr1*/
if(j==1)instr1; /* conditie falsa -nu se executa instr1*/
```

break- iese din switch sau o ciclare (for, while, do while)

```
/* daca switch nu are break -executa la rand instructiunile  
   pana la sfarsit/break;*/
```

```
int i = 1;
```

```
switch (i) { case 1:cout<<"1";  
            case 2: cout << "2";  
            case 3: cout<<"3"; break;  
            } // afiseaza 1 2 3
```

continue - sare restul instructiunii de ciclare

```
while (/* ... */)
```

```
{ // ...
```

```
  continue; // echivalent cu goto contin;
```

```
// ...
```

```
  contin::;
```

```
}
```



```
do { // ...  
continue; //echivalent cu goto contin;  
// ...  
contin;;  
} while (/* ... */);
```

```
for (/* ... */) {  
// ...  
continue; // echivalent cu goto contin;  
// ...  
contin;;  
}
```

Probleme propuse

1. Diferenta dintre doua date calendaristice (calculul nr de zile fata de inceputul anului)
2. Ordonarea unei matrici crescator pe fiecare linie si pe fiecare coloana (copierea matricii intr-un vector, ordonarea acestuia si redistribuirea liniilor in matrice)