

Tipuri de variabile

1. Noțiuni generale

Până în prezent am declarat variabile doar în corpul funcțiilor, inclusiv în cel al funcției **main()**. Variabilele astfel declarate se numesc **locale**. Sistemul de operare alocă fiecărui program trei zone distincte în memoria internă a calculatorului, în care se găsesc memorate variabilele programului. Harta simplificată a memoriei interne este prezentată în continuare:

segment de date
segment de stivă
heap

Segmentul de date este folosit pentru stocarea codului programului aflat în execuție și pentru variabilele care sunt folosite de către acesta pe toată durata execuției. Codul unui program se încarcă în segmentul de date numai în momentul în care acesta este lansat în execuție, această operație realizându-se de către o componentă a sistemului de operare numită *încărcător (loader)*. Reamintim că una din principalele sarcini ale sistemului de operare este gestionarea resurselor fizice și logice ale unui sistem de calcul, cele mai importante resurse fiind memoria RAM și timpul de lucru al procesorului. Evident, în momentul în care execuția programului se încheie, sistemul dealocă memoria din segmentul de date, programul și toate variabilele acestuia devenind inaccesibile.

Segmentul de stivă este utilizat pentru a realiza transferul parametrilor la apelul subprogramelor și pentru a controla execuția programului în cazul lucrului cu subprograme. În momentul în care este apelată o funcție (un subprogram), controlul programului este cedat acesteia. Funcția se execută instrucțiune cu instrucțiune până la încheierea normală sau forțată a acesteia. În acest moment controlul programului revine modulului care a făcut apelul (funcția **main()** sau orice altă funcție). Pentru ca această

revenire să se facă corect, pe stivă se salvează, în momentul apelului, adresa de revenire (spre exemplu, adresa instrucțiunii imediat următoare instrucțiunii de apel) și contextul din care s-a făcut apelul, adică valorile tuturor variabilelor locale modulului apelator. În momentul în care execuția funcției se încheie, se recuperează de pe stivă aceste informații în ordine inversă, adică se reface contextul din care s-a făcut apelul și se află adresa de revenire. În acest fel, execuția modulului apelator va continua corect după execuția apelului funcției. În absența segmentului de stivă nu ar fi posibilă implementarea lucrului cu subprograme. În continuare vom vedea că stiva are și alte destinații extrem de importante.

Zona *heap* se numește și *zonă de acumulare a variabilelor dinamice* și se folosește pentru alocarea variabilelor dinamice, create la cererea expresă a programatorului, în timpul execuției unui program prin apelul funcției **malloc()**, de exemplu, sau prin utilizarea operatorului **new**.

De asemenea, în limbajul C++, există posibilitatea ca variabilele să fie memorate într-un anumit registru al microprocesorului. În acest caz, timpul de acces la astfel de variabile este foarte mic, deci se pot obține programe optimizate. Numărul variabilelor care pot fi memorate în regiștrii interni este însă redus.

În general o variabilă se caracterizează prin patru attribute și anume:

- clasa de memorare
- vizibilitatea
- durata de viață
- tipul variabilei, singurul studiat până în acest moment

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.

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

- *vizibilitate la nivel de bloc* (instrucțiune compusă)
- *vizibilitate la nivel de fișier*, în cazul în care programul ocupă doar un singur fișier sursă (singurul caz tratat momentan)

- *vizibilitate la nivel de clasă*, legată de programarea orientată pe obiecte, facilitare specifică în C++

Durata de viață reprezintă timpul în care variabila respectivă are alocat spațiu în memoria internă (deci există efectiv). Astfel avem:

- *durată statică*, conform căreia variabila are alocat spațiu în tot timpul execuției programului
- *durată locală*, conform căreia variabila are alocat spațiu în timpul în care se execută instrucțiunile blocului respectiv (blocul care conține declarația variabilei)
- *durată dinamică*, caz în care alocarea și dealocarea spațiului necesar variabilei respective se face explicit de către programator, în timpul execuției programului, prin folosirea unor operatori și funcții speciale (de exemplu, **new**, **malloc()** etc.)

În limbajul C/C++ variabilele pot fi împărțite în trei mari categorii: *locale*, *globale* și *dinamice*.

2. Variabile globale

Aceste variabile se declară în afara corpului oricărei funcții, ca în exemplul următor:

```
#include<iostream>

int a;

void t()
{
    a=3;
    cout<<a;
}

int b;

int main()
{
    b=4;
    cout<<b;
    t();
}
```

Variabilele **a** și **b** sunt globale. Ele *pot fi utilizate de toate funcțiile care urmează în textul sursă declarației variabilei respective*. Deoarece pot fi utilizate, deci implicit și

modificate, de toate funcțiile care succed momentului declarației, aceste variabile se numesc **globale**.

Observație: La declarare, variabilele globale sunt inițializate de către compilator cu 0.

Atributele variabilelor globale sunt:

- *clasa de memorare* este segmentul de date
- *vizibilitatea*: sunt vizibile în toate funcțiile care le succed; în exemplul anterior, **a** poate fi accesată din corpul oricărei funcții, în timp ce variabila **b** poate fi accesată numai din funcția *main*
- *durata de viață* este statică, adică au spațiu de memorie rezervat în tot timpul execuției programului

3. Variabile locale

Aceste variabile sunt declarate în corpul funcțiilor, mai precis, pot fi declarate în orice bloc (instrucțiune compusă) al acestora.

Observație: Variabilele declarate în funcția **main()** sunt tot variabile locale, deci sunt vizibile numai în această funcție.

În exemplul următor, variabila **a** este declarată în corpul funcției **t()**, iar variabila **b** este declarată în corpul funcției **main()**:

```
void t()
{
    int a=3;
    cout<<a+2;
}
int main()
{
    int b=4;
    cout<<b--; t();
}
```

Atributele variabilelor locale sunt:

- *clasa de memorare* este, implicit, segmentul de stivă.

- *vizibilitatea* variabilelor locale este la nivelul blocului în care au fost declarate.
- *durata de viață* a variabilelor locale este atât timp cât durează execuția blocului respectiv, adică este *locală*

Observație: Variabilele locale nu sunt inițializate implicit de către compilator cu 0, această sarcină revenind programatorului. În cazul în care inițializarea este omisă, ele conțin o valoare oarecare numită valoare **reziduală** care poate duce la rezultate eronate.

Exemplul 1: În funcția următoare am declarat două variabile de tip **int**, numite **b** și **c**. Variabila **b** este vizibilă la nivelul funcției, dar variabila **c** este vizibilă doar la nivelul blocului în care a fost declarată. Ambele sunt variabile locale alocate pe segmentul de stivă.

```
void f()
{
    int b=7; // vizibilă în toată funcția
    {
        int c=9; // vizibilă numai în acest bloc (instr.compusă)
        cout << b << " " << c;
    }
}
```

Exemplul 2: În programul următor am declarat trei variabile, toate numite **a**. Una este globală, iar două sunt locale, dar declarate în blocuri diferite.

```
#include<iostream>

int a; // variabilă globală, vizibilă în orice punct din program

void f()
{
    int a=4; // variabilă locală, vizibilă în toată funcția
    {
        int a=3; // variabilă locală, vizibilă numai în acest bloc
        cout << a; // tipărește a=3
        a+=2;
    }
    cout << a; // tipărește a=4
    a++;
}
```

```

int main()
{
    a=5; // referă variabila globală a
    f();
    cout<<a; ; // tipărește a=5
}

```

Aplicatii:

<pre> #include <iostream> using namespace std; int N; void f1() { int x=5; N=10; cout << endl<<N; cout << endl << x; } int main () { N=4; cout << N; f1(); } </pre>	<pre> #include <iostream> using namespace std; int N; void f1() { int x=5; cout << endl << x; P=2; } int P=9; int main () {f1(); cout << x; P=7; } </pre>	<pre> #include <iostream> using namespace std; int N; void f1(int p) { int x=p; cout << x; } int main () { f1(77); } </pre>
---	--	---