Cursul 6
FUNDAMENTE LIMBAJULUI C++ (v)

OBIECTIVE

- Scopul, vizibilitatea si tipul identificatorilor,
- Includerea enumerarilor,
- Declaratii

PREZENTARE GENERALA

Asa cum am vazut, prin utilizarea instructiunilor compuse, putem obtine sectiuni de program in care identificatorii au asociate diferite atribute, accesibilitatea acestora fiind supusa acestor modificari. Acesti factori ce intervin in aceste procese sunt grupati in trei categorii : **scop, vizibilitate si tipul numelui.**

SCOPUL

Prin ***scopul*** unui identificator, intelegem acea portiune din program in care este valabila o legare a identificatorului de o entitate (obiect, variabila, constanta sau functie).

In C exista cinci categorii de scop ale unui identificator :

- bloc,
- functie,
- prototipul functiei,
- fisier,
- structura.

Scopul identificatorilor declarati in interiorul unui bloc se afla in acel bloc. Acest domeniu are drept punct de plecare momentul declaratiei identificatorului si se incheie odata cu inchiderea blocului. Atunci cand blocul contine subblocuri, scopul identificatorului nu contine si acele subblocuri.

Prin ***bloc*** intelegem o instructiune compusa. In limbajul **C** o declaratie poate fi in interiorul unui bloc sau in afara blocurilor. Domeniu scopului de tip bloc este aria cuprinsa intr-o pereche de acolade.

Majoritatea identificatorilor din cadrul unei functii au scopul de tip bloc. Lista parametrilor formali ai functiei intra in aceasta categorie. Domeniul scopului de tip ***functie*** se aplica numai etichetelor interioare unei functii.

Prototipul functiei reprezinta declararea functiei in cauza. Prin prototip al unei functii intelegem o declaratie a acesteia, in care se fac cunoscute numele, tipul returnat si lista parametrilor sai (ca numar, tip si ca identificatori). Domeniul scopului de tip prototip al functiei se refera la lista parametrilor formali ai acesteia.

Un nume declarat in afara oricarui bloc sau declaratie de clasa are un domeniu de tip ***fisier***. Acest domeniu incepe in punctul in care numele este definit si tine pana la sfarsitul fisierului, care contine definitia respectiva. El poate fi utilizat in domeniul respectiv fara nici o restrictie daca nu este redefinit in blocurile incluse in domeniul sau. Daca un nume care are un domeniu de tip fisier este redefinit intr-un bloc inclus in domeniul sau, atunci el poate fi folosit, in acel bloc, daca este precedat de operatorul de rezolutie. Deci, domeniu al scopului de tip fisier este domeniul exterior functiilor si declaratiilor de clasa.

Ultima categorie a scopului este cea in care scopul cuprinde declaratia unei ***clase***. Membrii unei clase au drept scop acea clasa, iar numele lor sunt ascunse restului programului. In cazul **structurilor** si **uniunilor**, care sunt forme speciale ale claselor, scopul membrilor acestor tipuri de date se inscrie in categoria clasei. Numele clasei nu are drept scop acea clasa, ci se stabileste in functie de locul declaratiei clasei respective.

Identificatorii care nu se afla in blocuri, functii sau clase, au ca domeniu al scopului intregul ***fisier***. Acesta este delimitat de declaratia identificatorului, pe de o parte, iar pe de cealalta parte, de sfarsitul fisierului. Daca identificatorul se afla intr-un fisier header, scopul acestuia va fi de tip fisier si va cuprinde orice fisier ce include headerul respectiv.

VIZIBILITATEA

Vizibilitatea unei variabile este o caracteristica ce defineste partile unui program, care vor putea recunoaste variabila respectiva. Astfel, o variabila poate fi recunoscuta in interiorul unui bloc, al unui fisier, al unui grup de fisiere sau in tot programul.

Un identificator este recunoscut in domeniul sau, daca nu este redefinit in blocuri incluse in domeniul respectiv. Un identificator redefinit in blocuri din domeniul sau, devine temporar ascuns. Un identificator cu domeniul de tip fisier poate fi facut vizibil in domeniul in care este redefinit, folosind ***operatorul de scop*** : " **::** " , iar daca identificatorul respectiv este numele unei clase, atunci el va fi precedat de cuvantul cheie corespunzator : **class, struct** sau **union.**

Prin urmare, domeniul de vizibilitate al unui identificator este acea parte a domeniului sau in care el poate fi utilizat.

Numele utilizate in **C** se impart in patru mari categorii:

- etichete de instructiuni,
- etichete de structuri, uniuni si enumerari,
- nume ale membrilor tipurilor agregate (structuri, uniuni),
- functii, variabile, nume introduse prin **typedef** si membri ai enumerarilor.

Doi identificatori pot avea acelasi nume si apartine aceleasi categorii a numelui, in cazul in care au scopuri diferite.
In exemplul urmator, sunt utilizate doua declaratii ale structurii **punct** in functii diferite, deci, cu scopuri distincte:

```
void func1 ( )
{
    struct punct { int x, y ; };
    ........
}

void func2 ( )
{
    enum punct { simplu, dublu };
    ........
}
```

Doi identificatori nu pot avea aceeasi denumire daca se afla in acelasi domeniu al scopului si apartin aceleiasi categorii a numelui. In exemplul de mai jos, sunt utilizate doua declaratii ale structurii **punct** in aceeasi functie, ceea ce este ilegal, avand un duplicat al numelui.

```
void func ( )
{
    struct punct { int x, y ;};
    enum punct { simplu, dublu };
}
```

In cazul in care numele a doi identificatori fac parte din categorii diferite ale numelui, nu conteaza ce domeniu al scopului au. Deasemenea, acestia pot avea aceeasi denumire, fara a exista riscul aparitiei de erori sau confuzii. Spre exemplu, in cadrul unei functii putem avea o variabila **x** si o eticheta cu acelasi nume, deci tot **x**.

```
void func (int x)
```

```
{
    if (x==5) goto x;
    cout << "NOT ";
x:
    cout << " EQUAL \n";
}
```

INCLUDEREA ENUMERARILOR

In C este posibila declararea unei enumerari si a unei variabile, avand acest tip in cadrul unei structuri.

```
struct luna {
    enum sapt {luni,marti,miercuri,joi,vineri,sambata,duminica} zile[7];
    int numar_zile
};
```

Astfel, am declarat **sapt** ca fiind o enumerare si **zile** un vector de elemente de tip **sapt**. Deorece in **C**, scopul numelui unei enumerari declarate in interiorul unei structuri depaseste granitele acelei structuri, **sapt** nu este ascuns exteriorului structurii, pe cand **zile** este. Altfel scris, exemplul de mai sus va arata:

```
enum sapt { luni, marti, miercuri, joi, vineri, sambata, duminica };

struct luna {
    enum sapt zile[7];
    int numar_zile;
};
```

Pentru a accesa un tip enumerat in interiorul unei structuri, va trebui sa utilizam operatorul de scop si sa completam numele tipului cu numele structurii:

```
enum luna::sapt x;
x=luna::marti;
```

In acest caz, operandul drept al operatorului de scop indica identificatorul sau numele tipului, iar operatorul stang desemneaza scopul, acesta putand fi numele oricarei clase, structuri sau uniuni. Aceasta metoda de accesare este disponibila numai in cazul in care tipul enumerat este declarat public si nu se afla in sectiunea privata a clasei. Asa cum se aplica asupra structurilor, si asupra claselor si uniunilor se poate aplica regula ascunderii enumeratelor.

DECLARATII IN C++

O ***declaratie*** specifica un tip si este urmata de o lista de una sau mai multe variabile de acel tip. Declaratiile pot aparea in unul din urmatoarele contexte:

- declaratii de variabile,
- declaratii de tipuri de date, sau
- prototipuri de functii.

Regulile de utilizare a variabilelor sunt:

- Putem declara variabilele acolo unde avem nevoie de ele, la inceputul buclei .
- Putem declara variabilele in interiorul unor blocuri.
- Nu avem voie sa folosim duble declaratii.
- O declarare o putem completa cu o initializare (int i=0;).
- Declararea se face in interiorul blocului ciclului.

```
int i; // Declaratie de variabila
nt i=5; // Declaratie de variabila si initializare

struct ceas { // Declaratie de tip data
    int sec ;
    ceas (int s);
    void oms( int &ore, int &minute, int &secunde );
};

int aduna(int a,int b ); // Prototip de functie
```

In cadrul celei de-a doua declaratie este permis ca pe langa specificarea tipului variabilei, sa se efectueze chiar si initializarea acesteia. Aceste declaratii seamana cu instructiunile de atribuire, dar nu sunt atribuiri. Diferenta intre atribuiri si initializari consta in aceea ca, in timp ce atribuirile efectueaza o simpla incarcare de memorie a unei valori, in cadrul initializarilor de acest tip este specificat si tipul variabilei.

Declaratiile pot fi plasate atat in interiorul blocurilor, cat si in exteriorul acestora, si au ca scop intreg fisierul. Acestea pot fi plasate oriunde in cadrul acestuia, chiar daca, de obicei, aceste declaratii se afla chiar la inceputul fisierului. Nu este permisa dublarea declaratiei unei variabile in cadrul aceluiasi domeniu al scopului. Prin plasarea declaratiilor in interiorul ciclului, scopul va fi local, deci in interiorul blocului ciclului.