Lucrarea nr.6

MOŞTENIREA

La definirea claselor, ar putea exista situații în care o clasă foloseşte multe sau chiar toate caracteristicile unei clase existente, adăugînd apoi unul sau mai mulți membrii de date sau funcții. În astfel de cazuri, C++ vă oferă posibilitatea de a crea noul obiect pe baza caracteristicilor obiectului existent. Deci, noul obiect va *moşteni* membrii clasei existente (numită clasă de bază). Atunci când o nouă clasă este creată pe baza uneia deja existente, noua clasă este o *clasă derivată*.

- Atunci când utilizează, programele folosesc o *clasă de bază* pentru a deriva o nouă clasă. Această nouă clasă (derivată) moşteneşte membrii clasei de bază.
- În scopul inițializării membrilor unei clase derivate, programul apelează funcțiile constructor ale clasei de bază şi clasei derivate.
- Programele pot accesa în mod simplu membrii clasei de bază şi ai clasei derivate prin intermediul operatorului punct.
- Pe lângă membrii *publici* (accesibili oricui) şi cei *privați* (accesibili numai metodelor clasei), C++ permite declararea membrilor *protejați,* accesibili de către membrii clasei de bază şi ai celei derivate.
- Pentru eliminarea conflictelor de nume dintre membrii clasei de bază şi cei ai clasei derivate, programele pot utiliza operatorul de rezoluție globală (::), precedat de numele clasei de bază sau al celei derivate.

Următorul program creează o clasă de bază numită carte şi apoi derivează din această clasă o clasă denumită FisaBiblioteca:

```
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
class Carte
{
 public:
      Carte(char *titlu) {strcpy(Carte::titlu,titlu);};
      void arata_titlu(void) {cout<<titlu;};
 private:
      char titlu[64];
 };
class FisaBiblioteca: public Carte
 {
 public:
    FisaBiblioteca(char *titlu, char *autor, char *editura): Carte(titlu)
     {
    strcpy(FisaBiblioteca::autor,autor);
       strcpy(FisaBiblioteca::editura,editura);
     };
    void arata_Biblioteca(void)
       {
         arata_titlu();
         cout<<' '<<autor<<' '<<endl<<editura<<endl;
        };
  private:
    char autor[64];
    char editura[64];
  };
void main(void)
```

```
 {
  FisaBiblioteca fisa("Insula cu elice","Jules Vernes","Teora");
  fisa.arata_Biblioteca();
 system("pause");
}
```

Următorul program creează o clasă de bază numită *angajat* şi apoi derivează din această clasă o clasă denumită *manager*.

```
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
class angajat
{
 public:
    angajat(char *,long, float);
    void arata_angajat(void);
 private:
       char nume[64];
   long angajat_id;
   float salariu;

};

angajat::angajat(char *nume,long angajat_id,float salariu)
{
strcpy(angajat::nume,nume);
angajat::angajat_id=angajat_id;
angajat::salariu = salariu;
}
void angajat::arata_angajat(void)
{
cout<<"Angajat:"<<nume<<endl;
cout<<"Id:"<<angajat_id<<endl;
cout<<"Salariu:"<<salariu<<endl;
}
class manager: public angajat
    {
    public:
    manager(char *,long, float,float);
    void arata_manager(void);
 private:
       float prima;

};
manager::manager(char *nume,long angajat_id,float salariu,float prima) : angajat(nume,
angajat_id, salariu)
{
manager::prima=prima;
}
void manager::arata_manager(void)
{
arata_angajat();
cout<<"Prima:"<<prima<<endl;
}
void main(void)
{
 angajat muncitor ("Popescu Ion",11,6000);
```

```
manager sef("Ionescu Ion",1,10000,2000);
muncitor.arata_angajat();
sef.arata_manager();
system("pause");
}
```

La derivarea unei clase dintr-o clasă de bază, funcția constructor a clasei derivate trebuie să apeleze constructorul clasei de bază. Pentru a apela constructorul clasei de bază, se plasează semnul (:) imediat după funcția constructor a clasi derivate, cu specificarea în continuare a numelui constructorului clasei de bază :

```
manager::manager(char *nume,long angajat_id,float salariu,float prima) : angajat(nume,
angajat_id, salariu)
{
manager::prima=prima;
}
void manager::arata_manager(void)
{
arata_angajat();
cout<<"Prima:"<<prima<<endl;
}
```

Atunci când o clasă moşteneşte caracteristicile mai multor clase, acea nouă clasă se obține printr-o *moştenire multiplă* .

Când se derivează o clasă prin intermediul moştenirii multiple, constructorul clasei derivate trebuie să apeleze funcțiile constructor ale fiecăreia din clasele de bază.

În acest caz, numele claselor de bază este separat prin virgulă. Exemplul următor prezintă o astfel de moştenire:

```
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
class Coperta {
   public:
     Coperta(char *titlu)  {strcpy(Coperta::titlu,titlu);};
   protected:
    char titlu[64]; };
class Pagina
 {
 public:
  Pagina(int linii=55) {Pagina::linii=linii;};
 protected:
  int linii;
  char *text;
  };
class Carte:public Coperta, public Pagina
  {    public:
       Carte(char *autor, char *titlu, float cost): Coperta(titlu),Pagina(60)
            {
            strcpy(Carte::autor,autor);
             strcpy(Carte::titlu,titlu);
             Carte::cost=cost;
            };
          void arata_carte(void)
           {
             cout<<titlu<<endl;
             cout<<autor<<'\n'<<cost<<endl;
             cout<<linii;
```

```
        };
    private:
            char autor[64];
            long cost;

};
  void main(void)
      {
       Carte roman("Arhanghelii","Ion Agarbiceanu",200000);
       roman.arata_carte();
system ("pause");
      }
```

În acest exemplu clasa *Carte* moşteneşte clasele *Coperta* şi *Pagina*.

Exemplu 2 :

```
#include <iostream.h>
#include <string.h>
#include <stdlib.h>

class Prima{
          public:
     Prima(float suma)
     {
     Prima::suma = suma;
     };
 protected:
          float suma;
   };
class masina{
          public:
          masina(char *marca)
      {
      strcpy(masina::marca,marca);
      };
 protected:
          char marca[64];
 };

class angajat:public Prima,public masina
{
 public:
    angajat(char *nume, float suma, char *marca):Prima(suma),masina(marca)
    {
    strcpy(angajat::nume,nume);
    strcpy(angajat::marca,marca);
    angajat::suma = suma;
    };
          void arata_angajat(void)
          {
```

```
                cout<<"Angajat:"<<nume<<endl;
                cout<<"Suma:"<<suma<<endl;
                cout<<"Masina:"<<marca<<endl;
        }
 private:
     char nume[64];
     float suma;
     char marca[64];
};
void main(void)
{
 angajat muncitor("Popescu Ion",2000,"Volvo");
 muncitor.arata_angajat();
 system("pause");
 }
```

Se poate de asemenea observa, că datele declarate ***protected*** în clasele de bază pot fi accesate de obiectele clasei derivate ca şi cum ar fi publice. Aceste variabile nu sunt accesibile direct obiectelor din clasa de bază. În cazul moştenirii private, în fața clasei de bază se utilizează cuvântul cheie ***private***. În acest tip de moştenire toți membrii clasei de bază devin de tip privat pentru clasa derivată, aşa cum arată următorul exemplu:

```
#include <iostream.h>
#include <string.h>
class Baza {
  public:
    Baza()  {cout<<"Constructorul clasei Baza\n";};
  int data;
};
class Derivata:private Baza
 {
  public:
   Derivata():Baza()
    {cout<<"Constructorul clasei Derivata\n";};

};
  void main(void)
     {
      Derivata obiect;
      obiect.data=5;
       }
```

La rularea acestui program compilatorul va genera o eroare atunci când se ajunge la linia de program în care se încearcă accesarea elemntului *data*.

Temă: Să se modifice programul anterior astfel încât elementul *data* să poată lua valoarea 5.

În mod asemănător, o clasă poate moşteni o clasă de bază în mod protejat prin utilizarea cuvântului cheie ***protected*** în fața clasei de bază. În acest caz, toți membrii clasei de bază devin membrii protejați pentru clasa derivată.

Atunci când se utilizează moştenirea multiplă pentru a deriva o clasă, clasa derivată trebuie să apeleze funcțiile constructor ale fiecăreia din clasele de bază. Ordinea apelării constructorilor depinde de ordinea în care clasa derivata a specificat clasele de bază. Funcțiile destructor se vor apela în ordine inversă față de funcțiile constructor, aşa cum arată programul următor:

```
#include <iostream.h>
#include <string.h>
#include <stdlib.h>

class Baza1 {
  public:
    Baza1()  {cout<<"Constructorul clasei Baza1\n";};
    ~Baza1()  {cout<<"Destructorul clasei Baza1\n";};
  int unu;
};
class Baza2 {
  public:
    Baza2()  {cout<<"Constructorul clasei Baza2\n";};
    ~Baza2()  {cout<<"Destructorul clasei Baza2\n";};
  int doi;
};
class Baza3 {
  public:
    Baza3()  {cout<<"Constructorul clasei Baza3\n";};
    ~Baza3()  {cout<<"Destructorul clasei Baza3\n";};
  int trei;
};
class Derivata:public Baza1,public Baza2,public Baza3
 {
  public:
   Derivata():Baza1(),Baza2(),Baza3()
     {cout<<"Constructorul clasei Derivata\n";};
   ~Derivata() {cout<<"Destructorul clasei Derivata\n";};

};
  void main(void)
     {
      Derivata obiect;
 system ("pause");
 }
```

Teme:

1. Să se implementeze o clasa **Data_de_Nastere** care sa aibă ca membrii datele de tip întreg *an, luna, zi* şi o metodă *arata_data* care sa afiseze aceasta data, o clasa **Student** care sa aiba ca membrii *nume, prenume si varsta* **,** o clasă **Note** care să aibă ca membru un vector de cinci elemente de tip real ce cuprinde cinci note ale studentului şi o clasă **Grupa** derivată din cele trei clase anterioare care sa aiba ca membru propriu o variabilă întreagă *nr_grupa*. Să se creeze 5 obiecte de tip **Grupa** şi să se afişeze informaţiile despre fiecare obiect sub forma: nr_grupa, nume, prenume, varsta, data nasterii si media notelor. Să se afişeze un mesaj la apelul funcţiilor constructor şi destructor pentru fiecare obiect.

2. Să se implementeze o clasa **Jucători** care să aibă ca membrii următoarele tablouri: *nume, prenume* şi *varsta* şi o clasă **Echipa**, derivată din clasa **Jucători** care să aibă ca membrii datele *nume_echipă, puncte, golaveraj* şi o metodă *arata_date* care să afişeze datele despre această echipă şi jucătorii ei. Să se creeze 5 obiecte de tip **Echipa** şi să se afişeze clasamentul acestor 5 echipe.