POO – C++ - Laborator 8

Cuprins

1. Ordinea de apelare a constructorilor și a destructorilor

- *Ordinea de apelare a constructorilor pentru un obiect dintr-o clasă derivată:*
  - *Constructorul clasei de bază*
  - *Constructorul clasei derivate*

- *Ordinea de apelare a destructorilor pentru un obiect dintr-o clasă derivată:*
  - *Destructorul clasei derivate*
  - *Destructorul clasei de bază*

Prezentăm un exemplu complet de moștenire, cu clasele completate cu constructori și destructori. În cazul destructorilor, aceștia vor afisa un simplu mesaj, deoarece nu există dealocări de memorie necesare:

Persoana.h

```
class PersoanaAC
{
protected:
	string m_sCnp;
	string m_sNume;
	string m_sAdresa;
public:
	PersoanaAC();
	PersoanaAC(string cnp, string nume, string adresa);
	~PersoanaAC();
	void afisareProfil();
	void schimbareAdresa(string adresaNoua);
};
```

Student.h

```
class StudentAC : public PersoanaAC
{
	int m_ianStudiu;
	int m_inotaP2;
public:
	StudentAC();
	StudentAC(string cnp, string nume, string adresa, int
anStudiu, int notaP2);
	~StudentAC();
	void afisareProfil();
	void inscriereAnStudiu(int anStudiuNou);
};
```

Persoana.cpp

```
PersoanaAC::PersoanaAC()
{
	cout<<"constr. fara arg. PersoanaAC"<<endl;
	m_sCnp = string(13,'0');
	m_sNume = "";
	m_sAdresa = "";
}
PersoanaAC::PersoanaAC(string cnp, string nume, string adresa)
{
	cout<<"constr. cu arg. PersoanaAC"<<endl;
	m_sCnp = cnp;
	m_sNume = nume;
	m_sAdresa = adresa;
}
PersoanaAC::~PersoanaAC()
{
	cout<<"destructor PersoanaAC"<<endl;
}
```

Student.cpp

```
StudentAC::StudentAC()
{
	cout<<"constr. fara arg. StudentAC"<<endl;
	m_ianStudiu = 0;
	m_inotaP2 = 0;
}
StudentAC::StudentAC(string cnp, string nume, string adresa,
int anStudiu, int notaP2) :
	PersoanaAC(cnp, nume, adresa), m_ianStudiu(anStudiu),
	m_inotaP2(notaP2)
{
	cout<<"constr. cu arg. StudentAC"<<endl;
}
StudentAC::~StudentAC()
{
	cout<<"destructor StudentAC"<<endl;
}
```

Test.cpp

```
int main ()
{
  PersoanaAC p1("1234567890123" ,"Ana","Iasi");
  p1.afisareProfil();
  StudentAC s2;
  s2.afisareProfil();
  StudentAC s1("1234567890122", "Ion","Vaslui",2,10);
  s1.schimbareAdresa("Bucuresti");
  s1.inscriereAnStudiu(3);
  s1.afisareProfil();
  return 0;
}
```

La rulare, acest exemplu va furniza la iesire:

```
constr. cu arg. PersoanaAC
Nume: Ana CNP: 1234567890123 Adresa: Iasi
constr. fara arg. PersoanaAC
constr. fara arg. StudentAC
Nume:  CNP: 0000000000000 Adresa:
An studiu: 0 Nota P2: 0
constr. cu arg. PersoanaAC
constr. cu arg. StudentAC
Nume: Ion CNP: 1234567890122 Adresa: Bucuresti
An studiu: 3 Nota P2: 10
destructor StudentAC
destructor PersoanaAC
destructor StudentAC
destructor PersoanaAC
destructor PersoanaAC
```

Se observă apelarea constructorului fără argumente al clasei `PersoanaAC` pentru obiectul `s2` de tipul `StudentAC`. Acest constructor este apelat deoarece pentru un obiect al unei clase derivate, constructorul apelat implicit pentru clasa de baza este constructorul fără listă de argumente (dacă există constructori declarați în clasă).

Prin apeluri de forma

```
StudentAC::StudentAC(…) :
     PersoanaAC(cnp, nume, adresa), ...
```

se poate controla ce constructor din clasa de bază va fi folosit (se forţează compilatorul pentru a apela un anumit constructor).

2. Mostenirea multiplă

Mostenirea multiplă permite unei clase derivate să moștenească mai mult de un singur părinte. Considerăm exemplul clasei Profesor care poate extinde atât clasa Persoana, cât și clasa Angajat:

Persoana.h

```
class Persoana
{
private:
    string m_sNume;
    int m_iVarsta;

public:
    Persoana(string nume, int varsta)
        : m_sNume (nume), m_iVarsta(varsta)
    {}
    int getVarsta()
     {
          return m_iVarsta;
```

```
    }
};
```

Angajat.h

```
class Angajat
{
private:
    double m_dSalariu;

public:
    Angajat(double salariu)
        : m_dSalariu(salariu)
    {}
    double getSalariu()
     {
          return m_dSalariu;
     }
};
```

Profesor.h

```
class Profesor: public Persoana, public Angajat
{
private:
     int m_iGradDidactic;

public:
    Profesor(string nume, int varsta, double salariu, int
gradDidactic)
         : Persoana(nume, varsta),
          Angajat(salariu),
          m_iGradDidactic(gradDidactic)
    {
    }
};
```

Test.cpp

```
int main()
{
     Profesor p1("Ion",23,100.08,1);
     return 0;
}
```

In exemplul de mai sus, obiectul `p1` de tipul `Profesor` va avea variabila membra proprie clasei `Profesor` (`m_iGradDidactic`), dar va avea și variabilele membre din cele două clase de bază pe care le moștenește: `m_sNume` și `M_iVarsta` din clasa `Persoana`, respectiv `m_dSalariu` din clasa `Angajat`.

3. Exerciții

- Implementați cele două exemple furnizate în cadrul laboratorului.

- Verificați la adresa http://www.cplusplus.com/reference/string/string/string/ ce alte tipuri de constructori pot fi folosiți la inițializarea variabilelor de tip string. Pentru unul din exemplele implementate, utilizați cel putin alți doi constructori din lista de pe site.
- Extindeți primul exemplu cu implementarea următoarelor:
    - O funcție afisareProfil ce nu aparține nici unei clase și afișează datele corespunzătoare unui Student.
    - O funcție membră a clasei StudentAC ce compară notele a doi studenți (între nota studentului pentru care se apelează funcția și nota studentului primit ca parametru) și returnează un pointer la studentul cu nota cea mai mare.
    - O nouă clasă StudentMaster derivată din StudentAC cu următorii membri:
        - m_sNumeDizertatie de tip string ce va conține numele lucrării de dizertație
    - Pentru clasa StudentMaster să se implementeze constructorii necesari și un destructor și să se testeze ordinea de apelare a acestora.
    - Sa se creeze in main() un vector de obiecte de tip StudentMaster și să se determine studentul cu nota cea mai mare (m_inotap2).