Programare Orientată Obiect

Laborator 5

Despre ce vorbim azi

- Pointeri la obiecte și masive de obiecte
- Includerea (compunerea claselor)
- Tipuri de membri în clasă
- Clase și funcții prietene

Pointeri la obiecte

- Se definește la fel ca exemplul din al doilea laborator:

  Student *ps;

- Putem accesa membri (cei publici) folosind operatorul -> ;
- Declararea unui astfel de pointer nu declanșează apelul constructorului ci doar aloca 2 sau 4 bytes pentru variabila de tip pointer;
- Pentru alocarea spațiului de memorie aferent trebuie să utilizăm operatorul **new**;

Pointeri la obiecte

- Acesta determină apelul constructorului
- Ex:

```
Student *ps = new Student;
Student *ps2 = new Student("Nume", 1, 10);
```

- Pentru dezalocarea spațiului de memorie se folosește operatorul **delete**, care apelează și destructorul clasei;
- Funcțiile *malloc* și *free* nu pot fi folosite în lucrul cu obiecte deoarece ele nu generează apelul constructorului, respectiv al destructorului

Masive de obiecte

- Deoarece clasele reprezintă un tip abstract și încapsulat, putem declara masive de obiecte
- Ex:

```
Studente serie[] = {
                    Student(“Nume1”, 1, 9.5),
                    Student(“Nume2”, 2, 10),
                    Student(“Nume3”, 3, 8)
                    };
```

Elementele unui program C

- Partea de declarare a bibliotecilor folosite (sau de includere a lor) - **1**
- Partea corespunzătoare variabilelor globale și a funcțiilor - **2**
- Programul principal - **3**
- Ex:

```
#include <iostream.h> 1
int i; 2
long suma(int x, int y) 2
{
        return x+y;
}
void main() 3
{ … }
```

Elementele unui program C++

- Locul claselor este în partea corespunzătoare variabilelor globale (ba chiar în fișiere header separate ce vor fi incluse în partea de includere);
- Ele sunt globale (pot fi văzute de programul principal, de alte funcții globale, de alte clase)
- Definirea lor se termină cu }; dar cu mici excepții (funcțiile definite ulterior tot în zona globală dar cu rezoluție de clasă fac parte tot din clasa respectivă)
- Clasele pot conține la rândul lor funcții (așa cum am văzut până acum)
- Dacă omitem rezoluția de clasă vom scrie doar o funcție globală și atât (ea nu va face parte din clasa, nu va primi pointerul this ca parametru, nu va avea acces la membri privați sau protejați)

Compunerea claselor

- O clasă poate avea ca membru o altă clasă;
- Acest lucru se întâmplă (se cere a fi modelat în acest fel) atunci când între cele doua clase există o relație de incluziune (prima clasa răspunde la întrebarea “has a”)
- Ex de clasa ce poate fi inclusă în clasa Student:

```
class Catalog
{
public:
        int* note;
…
} situatie_scolara;
```

- Accesul membrilor clasei catalog se va face folosind sintaxa Student::Catalog

Tipuri de membri în clasă

- Clase cu membri constanți – odată definiți (de obicei prin constructor), nu mai pot fi modificați
- Clase cu membri statici – în contextul claselor identificatorul **static** capătă un alt înțeles față de cel de la variabile. El se referă la atribute și funcții ce țin de clasă în general și nu de instanțe ale ei (ex: numărul total de studenți). Accesul la aceștia se face cu ajutorul operatorului de rezoluție (ex: Student::numar_studenti=0)
- Clase cu membri pointeri de membri

Clase și funcții prietene

- Accesul la membrii unei clase privați sau protejați poate fi acordat unor funcții externe clasei sau din alte clase;
- Acest lucru se realizează prin declararea funcțiilor respective ca prietene (folosind cuvântul cheie **friend**)
- Pentru a avea acces la membrii respectivi funcția va trebui să primească drept parametru o referință de tipul clasei
- Continuăm cu un exemplu concludent

Funcții prietene

```
class Student;
class Profesor
{
        public: int vezi_medie(Student &);
};
class Student
{
        …
        private: friend int Profesor::vezi_medie(Student &);
        …
};
int Profesor::vezi_medie(Student &s) { return s.medie; }
```

Precizări

- Declarația friend trebuie făcută, așa cum se observă, în clasa care acordă drepturi de acces
- Nu puteam omite declarația formală de pe primul rând;
- Putem declara o întreaga clasă drept prieten, însă în acest caz avem probleme grave de securitate (clasa prieten poate accesa toate datele membre ale celeilalte clase, indiferent că sunt publice, private sau protejate)
- Dezavantajul funcțiilor prieten e că pot și modifica membri, însă fără ajutorul lor nu putem modela comunicarea între anumite obiecte