

An Universitar 2023-2024, Semestrul 2, Grupa 133

Laborant: Daniel Wagner

Laboratorul 2 POO - Ghid introductiv POO în C++

Conținutul laboratorului:

- I. [Recapitulare C - Introducere în Pointeri](#)
- II. [Introducere în C++](#)
 - 1. [Constructori](#)
 - 2. [Getteri și Setteri](#)
 - 3. [Constructor de Copiere](#)
 - 4. [Operatorul de Atribuire \(operator=\)](#)
 - 5. [Destructor](#)
- III. [Exerciții individuale](#)

I. Recapitulare C - Introducere în Pointeri

În C/C++, pointerii sunt variabile care stochează adresele de memorie a altor variabile. Aceștia sunt importanți în C/C++, permițând accesarea și modificarea directă a memoriei (conținutului RAM-ului) și facilitând crearea de structuri de date complexe (Vectori, Liste, etc.), gestionarea resurselor alocate dinamic, modificarea valorii unei variabile locale în interiorul unei funcții apelate care primește drept parametru adresa acelei variabile, și multe alte utilități despre care vom afla în cadrul acestui semestru.

Următorul exemplu ilustrează toate operațiile care pot fi făcute asupra unui pointer: declararea lui, inițializarea cu adresa unei variabile (sau începutul unui bloc de memorie alocat dinamic – vedem ulterior), și dereferențierea acelui pointer și atribuirea unei valori noi, prin care obținem schimbarea valorii variabilei a cărei adrese o reținem cu ajutorul pointer-ului:

```
// declar o variabila de un tip de date oarecare, spre exemplu int
int variabila;
// si o initializez cu o valoare, spre exemplu 10
variabila = 10;
// declar o variabila de tip pointer, in cazul actual, pointer catre int
// tipul de date al pointer-ului este Tip* pentru a retine adresa de
// memorie a unei variabile de tipul Tip
int *pVariabila;
// atribui variabilei de tip pointer catre int adresa variabilei de tip int
pVariabila = &variabila;
// acum, pot modifica valoarea variabilei indirect prin pointer
// cu ajutorul operatorului de dereferentiere `*`
*pVariabila = 20;
// valoarea variabilei initiale a fost schimbata indirect la 20
// deci, daca o afisam, sa ne asteptam sa vedem 20
cout << variabila << endl;
```

II. Introducere în C++

Pe parcursul acestui laborator, vom lucra cu clasa **Student**, în cadrul careia vom implementa funcțiile din subcapitolele de mai jos pentru început, și ulterior veți lucra exercițiile de la final individual.

Cod initial *ex1_Introducere_in_POO_in_C++_si_pointeri.cpp*:

```
#include <iostream>
#include <cstring>
using namespace std; // incepand cu laboratorul urmator, nu o sa mai folosim

class Student {
private:
    char *nume;
public:
    // aici vom adauga functiile (metodele) din clasa
    // si ulterior vom imparti in fisiere .h si .cpp
};

int main()
{
    // aici vom testa functionalitatea clasei Student
    return 0;
}
```

1. Constructori

Constructorii inițializează obiectele, care sunt instanțele clasei. Un constructor fără parametri cu corpul vid există în fiecare clasă (constructor implicit). Un constructor cu parametri permite inițializarea câmpurilor cu valorile trimise. Dacă implementați propriul constructor, constructorul implicit nu va mai exista, și trebuie implementat explicit. Pot exista mai mulți constructori, cu parametri diferiți. Acesta este un caz de supraîncărcare de funcții, polimorfism la compilare.

Exemplu: Inițializarea numelui studentului cu **new**:

```
Student(const char* nume)
{
    this->nume = new char[strlen(nume) + 1];
    strcpy(this->nume, nume);
}
```

2. Getteri și Setteri

Getterii și setterii controlează accesul la atributele unei clase (pe care întotdeauna le vom declara private/protected în cadrul acestui curs), astfel obținem încapsularea datelor, un concept important în POO, și protejăm integritatea datelor, să nu fie modificate/citite decât dacă permitem explicit. Vom vedea exemple de setteri/getteri mai complecși care ilustrează și constrangeri asupra datelor în laboratoarele ulterioare.

Exemplu: Accesarea numelui studentului (get și set):

```
char* getNume() const
{
    return this->nume;
}

void setNume(const char* nume)
{
    delete[] this->nume;
    this->nume = new char[strlen(nume) + 1];
    strcpy(this->nume, nume);
}
```

3. Constructor de Copiere

Pentru clase cu date membre alocate dinamic (pointeri), constructorul de copiere asigură crearea unei noi zone de memorie pentru variabila nouă, astfel facem o copie a conținutului memoriei în zona nou alocată, nu o copie a adresei de memorie (vedeți [deep-copy vs. shallow-copy](#)), evitând problemele rezultate din partajarea memoriei folosind doar shallow copy.

Exemplu: Copierea conținutului câmpurilor clasei Student, adică a numelui studentului, care este un câmp alocat dinamic:

```
Student(const Student& other)
{
    this->nume = new char[strlen(other.nume) + 1];
    strcpy(nume, other.nume);
}
```

4. Operatorul de Atribuire (operator=)

Suprascrierea operatorului = permite asignarea corectă între obiecte, gestionând alocarea și eliberarea memoriei pentru attributele dinamice. Precum constructorul de copiere, operator= trebuie scris explicit doar în cazul în care avem date membre alocate dinamic. Altfel, funcțiile implicite realizează copierea tuturor datelor membre, fiecare câmp în parte.

Exemplu: Supraîncărcarea operator= în care copiem numele alocat dinamic (și eventual alte câmpuri membre ale clasei, dacă există):

```
Student& operator=(const Student& other)
{
    if(this != &other)
    {
        delete[] this->nume;
        this->nume = new char[strlen(other.nume) + 1];
        strcpy(this->nume, other.nume);
    }
    return *this;
}
```

5. Destructor

Destructorul eliberează resursele alocate dinamic de obiect, prevenind scurgerile de memorie (memory leaks). Destructorul este necesar să îl implementăm de asemenea doar dacă avem date membre alocate dinamic (ceea ce vom avea!).

Exemplu: Eliberarea memoriei pentru **nume** (și eventual a altor resurse alocate dinamic în clasă):

```
~Student()
{
    delete[] this->nume;
}
```

III. Exerciții individuale

1) Pe baza codului lucrat la laborator, realizați următoarele:

- a) Actualizați clasa Student, astfel încât să conțină și o variabilă membră care memorează anul nașterii studentului.
- b) Actualizați toate metodele implementate (dacă este nevoie), să țină cont și de această variabilă. Ce metode ați actualizat și de ce?
- c) Creați getteri și setteri pentru variabila adăugată.
- d) Ilustrați aceste noi funcționalități în main(), spre exemplu să setați vârsta studentului, și după să afișați vârsta modificată a studentului.

2) Pe baza codului lucrat la laborator, realizați următoarele:

- a) Actualizați clasa Student, astfel încât să conțină și o variabilă membră care memorează notele obținute alocată dinamic, precum și o variabila care retine câte note are studentul.
- b) Actualizați toate metodele implementate (dacă este nevoie), să țină cont și de variabilele adăugate la punctul a). Ce metode ați actualizat și de ce?
- c) Creați getteri și setteri pentru variabilele adăugate la punctul a).
- d) Ilustrați aceste noi funcționalități în main(), spre exemplu să afișați notele unui student.

3) Pe baza codului lucrat la laborator, și a exercițiilor anterioare rezolvate, realizați următoarele:

- a) Adăugați o metodă care calculează media generală a studentului și o returnează.
- b) Actualizați setter-ul asigurându-vă că anul nașterii este plauzibil pentru un student (spre exemplu, după 1990 dar până în 2008). Dacă nu întâlnește această condiție, anunțați utilizatorul printr-un mesaj personalizat că anul nașterii trimis de acesta nu este plauzibil, și lăsați-l nemodificat. Observație: ulterior, veți învăța despre excepții, un mecanism mai bun de abordare a acestei probleme decât afișarea unui mesaj către utilizator.
- c) Apelați metoda creată la a) în main() și afișați media rezultată.
- d) Tot în main(), testați cazul în care încercați să setați vârsta studentului la o valoare neplauzibilă și observați că se afișează mesajul vostru, și totodată că vârsta a rămas neschimbată.