Laborator 8

Tema: Funcții și Clase Template </>

Poveste

O companie dezvoltă un sistem generic pentru înregistrarea și stocarea datelor (logging). Sistemul trebuie să poată gestiona **înregistrări** de diverse tipuri. Fiecare **înregistrare** este formată dintr-o **descriere** textuală și o **valoare** asociată. Această valoare poate fi un tip de date simplu (cum ar fi un cod de eroare de tip `int`) sau un obiect complex (cum ar fi un **profil de utilizator**).

Acest scenariu este perfect pentru a aplica conceptele de **clase template**. Vom crea o clasă generică `Inregistrare<T>` care poate stoca o valoare de orice tip `T`. Astfel, putem refolosi aceeași structură pentru a salva date despre utilizatori, senzori, erori etc., demonstrând puterea și flexibilitatea template-urilor în C++.

Scopul laboratorului

- înțelegerea conceptului de **programare generică**;
- definirea și utilizarea **funcțiilor template**;
- definirea și instanțierea **claselor template**;
- organizarea codului template în fișiere header.

Structura fișierelor

- **profilUtilizator.h** / **profilUtilizator.cpp** — Clasă non-template ce folosește `char*` pentru atribute.
- **inregistrare.h** — Clasa template `Inregistrare<T>` care va fi implementată direct în header.
- **main.cpp** — Testarea funcționalităților.

Fișierul `profilUtilizator.h`

```h
#pragma once
#include <iostream>

class ProfilUtilizator {
private:
    char* numeUtilizator;
    int nivel;

public:
    // Metode necesare:
    // - Constructor cu parametri
    // - Destructor
    // - Constructor de copiere
    // - Operator de atribuire
    // - Operator de afisare <<
};
```

Fișierul `inregistrare.h`

```h
#pragma once
#include <iostream>

template <class T>
class Inregistrare {
private:
    char* descriere;
    T valoare;

public:
    // Metodele clasei template (implementate direct in .h):
    // - Constructor cu parametri
    // - Destructor
    // - Constructor de copiere
    // - Operator de atribuire
    // - Operator de afisare <<
};
```

Explicații suplimentare

- **Programare Generică** — Permite scrierea de cod (funcții sau clase) care funcționează cu diverse tipuri de date. Acest lucru este realizat în C++ prin **template-uri**. Codul este scris o singură dată, iar compilatorul generează versiunile necesare pentru fiecare tip de date specific utilizat.
- **Clase Template** — O clasă prefixată cu `template <class T>` sau `template <typename T>` devine un șablon. `T` este un placeholder pentru un tip de date care va fi specificat la crearea unui obiect. De exemplu, `Inregistrare<ProfilUtilizator>` instruiește compilatorul să genereze o versiune a clasei în care `T` este înlocuit cu `ProfilUtilizator`.
- **Regula celor Trei** — Deoarece ambele clase, `ProfilUtilizator` și `Inregistrare`, gestionează memorie alocată dinamic (`char*`), este esențială implementarea corectă a destructorului, constructorului de copiere și operatorului de atribuire pentru a evita probleme precum memory leaks sau double free.

Cerințe (9p + 1p oficiu)

1. **Implementare Clasă Non-Template (3p)**
   - (2p) Implementarea corectă a Regulii celor Trei (destructor, cc, op=) pentru clasa `ProfilUtilizator` pentru a gestiona membrul `char* numeUtilizator`.
   - (1p) Implementarea constructorului și a operatorului de stream pentru `ProfilUtilizator`.
2. **Implementare Clasă Template (3p)**
   - (2p) Implementarea corectă a Regulii celor Trei pentru clasa template `Inregistrare<T>` (gestionând `char* descriere`), direct în fișierul `.h`.
   - (1p) Supraincărcarea corectă a operatorului de stream ca funcție template prietenă, implementată tot în `.h`.
3. **Testare în main.cpp (3p)**
   - (1p) Crearea și afișarea unor obiecte `Inregistrare<ProfilUtilizator>`.
   - (1p) Testarea riguroasă a constructorului de copiere și a operatorului de atribuire pentru ambele clase, pentru a valida managementul memoriei.
   - (1p) Instanțierea și utilizarea clasei `Inregistrare` cu un tip de date de bază (ex. `int`, `double`) pentru a demonstra generalitatea.
4. **(1p) Punct din oficiu.**

Compilare și verificare

```
g++ -g main.cpp profilUtilizator.cpp -o lab8
./lab8
valgrind ./lab8
```

Bonus GitHub Classroom

Studenții pot obține până la **3p bonus** dacă încarcă toate laboratoarele până la finalul semestrului, respectiv **1p bonus** dacă încarcă mai mult de jumătate dintre ele.

Pași pentru încărcarea laboratorului:

(a) Accesați cursul de pe OCW – Programare Orientată pe Obiecte, secțiunea **Resurse** → **Bonus GitHub laborator**.

(b) Acolo veți găsi linkul **GitHub Classroom** pentru Laboratorul 8:

`https://classroom.github.com/a/CeuVjhXY`

(c) După acceptare, veți primi propriul repository, ex. `laborator-8-NumePrenume`.

(d) Clonați repository-ul local:

```
git clone git@github.com:Laborator-POO-2025-2026/laborator-8-NumePrenume.git
cd laborator-8-NumePrenume
```

(e) După ce ați adăugat fișierele corespunzătoare (`profilUtilizator.h`, `profilUtilizator.cpp`, `inregistrare.h`, etc.), urmează comenzile:

```
git add .
git commit -m "Implementare laborator 8 - Functii si Clase Template"
git push origin main
```