Laborator 4

Tema: Studentul Gurmand

Poveste

Ioana, pasionată de gastronomie, a ajuns la partea a doua a proiectului său despre rețete. După ce a învățat cum să gestioneze o colecție de obiecte, descoperă acum nevoia de a copia și reasigna obiecte fără a produce erori de memorie. În același timp, observă că fiecare rețetă ar trebui să aibă un cod unic, care nu se repetă niciodată — chiar dacă obiectele se distrug.

Scopul laboratorului

- folosirea constructorului de copiere și a operatorului de atribuire;
- diferențierea dintre `const` și `static`;
- înțelegerea conceptului de **unicitate a obiectelor** prin atribute `const`;
- implementarea unui contor comun tuturor obiectelor (membru `static`);
- verificarea corectitudinii alocării și eliberării memoriei cu `valgrind`.

Structura fișierelor

- **reteta.h** – declararea clasei `Reteta`;
- **reteta.cpp** – implementarea metodelor clasei;
- **main.cpp** – testarea funcționalităților.

Fișierul `reteta.h`

```h
#ifndef RETETA_H
#define RETETA_H

class Reteta {
    char* nume;
    char* categorie;
    int timpPreparare;                  // in minute
    float calorii;                      // kcal / portie

    const int codUnic;                  // valoare unica per obiect
    static int numarRetete;             // contor comun tuturor obiectelor

public:
    // Constructori
    Reteta();
    Reteta(const char* nume, const char* categorie, const int timpPreparare,
    ↪  const float calorii);
    Reteta(const Reteta& other);        // constructor de copiere
    Reteta& operator=(const Reteta& other); // operator de atribuire

    // Destructor
    ~Reteta();

    // Getteri const
    char* getNume() const;
    char* getCategorie() const;
    int getTimpPreparare() const;
    float getCalorii() const;

    const int getCodUnic() const;

    // Setteri
    void setNume(const char* nume);
    void setCategorie(const char* categorie);
    void setTimpPreparare(const int& timp);
    void setCalorii(const float& calorii);

    // Alte metode
    void afisare() const;
    static int getNumarRetete(); // metoda statica
};

#endif
```

Fișierul `reteta.cpp`

```cpp
#include "reteta.h"
#include <iostream>
#include <cstring>
using namespace std;

// Initializarea membrului static
int Reteta::numarRetete = 0;

// Constructor default
Reteta::Reteta() : codUnic(++numarRetete) {
    // TODO: Initializati calorii cu 0.
    // TODO: Alocati memorie pentru nume si categorie (siruri goale) sau
    ↪ initializati cu nullptr.
    // TODO: Initializati timpul de preparare cu 0.
    // TODO: Afisati un mesaj (ex: "Reteta creata fara parametri.").
}

// Constructor cu parametri
Reteta::Reteta(const char* nume, const char* categorie, const int
↪ timpPreparare, const float calorii)
    : codUnic(++numarRetete) {
    // TODO: Alocati memorie pentru this->nume si copiati continutul lui 'nume'
    ↪ in el.
    // TODO: Alocati memorie pentru this->categorie si copiati continutul lui
    ↪ 'categorie'.
    // TODO: Initializati this->timpPreparare si this->calorii cu valorile
    ↪ primite.
    // TODO: Afisati un mesaj (ex: "Reteta creata: ...").
}

// Constructor de copiere (DEEP COPY)
Reteta::Reteta(const Reteta& other) : codUnic(++numarRetete) {
    // TODO: Copiati valorile din 'other' in obiectul curent.
    // TODO: Alocati memorie noua pentru sirurile de caractere (nume,
    ↪ categorie).
    // TODO: NU faceti this->nume = other.nume (shallow copy)!
    // TODO: Afisati un mesaj (ex: "Clona retetei: ...").
}

// Operator de atribuire (DEEP COPY)
Reteta& Reteta::operator=(const Reteta& other) {
    // TODO: Verificati auto-atribuirea (if (this == &other)).
    // TODO: Eliberati memoria veche alocata pentru nume si categorie.
    // TODO: Alocati memorie noua si copiati continutul din 'other' in 'this'.
    // TODO: Copiati valorile pentru timpPreparare si calorii.
    // TODO: Returnati *this pentru a permite atribuiri inlantuite.
    return *this;
}
```

```cpp
// Destructor
Reteta::~Reteta() {
    // TODO: Eliberati memoria alocata pentru nume si categorie (delete[]).
    // TODO: Afisati un mesaj (ex: "Reteta stearsa: ...").
}

// Metoda afisare
void Reteta::afisare() const {
    // TODO: Afisati numele, categoria, timpul, caloriile si codul unic.
}

// Metoda statica
int Reteta::getNumarRetete() {
    // TODO: Returnati numarul total de retete create.
    return numarRetete;
}
```

Fișierul `main.cpp` (model)

```cpp
#include "reteta.h"

int main() {

    // TODO 1: Creati un obiect r1 folosind constructorul cu parametri
    // si il afisati.

    // TODO 2: Creati un obiect r2 folosind constructorul de copiere.
    // Ex: Reteta r2(r1);

    // TODO 3: Testati operatorul de atribuire (=)
    // Ex: r3 = r1;

    // TODO 4: Cititi numarul de retete n si alocati un vector dinamic.
    // Ex: Reteta* v_retete = new Reteta[n]

    // TODO 5: Cititi datele fiecărei rețete (nume, categorie, timp, calorii)
    // si completati vectorul folosind metoda set sau constructorul
    // cu parametri.

    // TODO 6: Afisati toate rețetele citite si codurile lor unice.

    // TODO 7: Afișați numarul total de rețete create folosind metoda statica.

    // TODO 8: Eliberati memoria si verificati cu Valgrind.
    // g++ -g main.cpp reteta.cpp -o lab4
    // valgrind ./lab4

    return 0;
}
```

Explicații suplimentare

- **const** — marchează variabile care nu se pot modifica după inițializare.

```
const int codUnic; // se setează doar în lista de inițializare
```

- **static** — aparține clasei, nu obiectului; are o singură copie comună.

```
static int numarRetete; // comun tuturor obiectelor
```

- Membrii **static** se inițializează în fișierul `.cpp`:

```
int Reteta::numarRetete = 0;
```

Cerințe (10p)

1. (3p) Implementați clasa `Reteta` cu atributele: `nume`, `categorie`, `timpPreparare`, `calorii`.
2. (2p) Implementați constructorul de copiere și operatorul de atribuire.
3. (2p) Adăugați un atribut `const int codUnic` și un membru static `numarRetete`.
4. (1p) Afișați codul unic al fiecărei rețete.
5. (1p) Afișați numărul total de rețete create.
6. (1p) Testați totul cu `valgrind`.

Compilare și verificare

```
g++ -g main.cpp reteta.cpp -o lab4
./lab4
valgrind ./lab4
```

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:

1. Accesați cursul de pe OCW – Programare Orientată pe Obiecte, secțiunea **Resurse** → **Bonus GitHub laborator**.
2. Acolo veți găsi linkul **GitHub Classroom** pentru Laboratorul 4:

   `https://classroom.github.com/a/e5IrDPgO`

3. După acceptare, veți primi propriul repository, ex. `laborator-4-NumePrenume`.
4. Clonați repository-ul local:

   ```
   git clone
   ↪  git@github.com:Laborator-POO-2025-2026/laborator-4-NumePrenume.git
   cd laborator-4-NumePrenume
   ```

5. După ce ați adăugat fișierele `reteta.h`, `reteta.cpp` și `main.cpp`, urmează comenzile:

   ```
   git add .
   git commit -m "Mesaj de commit"
   git push origin main
   ```

Atenție: laboratoarele copiate între studenți sunt considerate nevalide, iar niciunul dintre cei doi participanți nu va primi bonusul. Repository-urile sunt monitorizate automat de sistemul GitHub Classroom.