# Laborator 5

Tema: Supraincărcarea Operatorilor (Clasa Fractie) ➕

---

## Poveste

Mihai, un elev pasionat de matematică, trebuie să implementeze o clasă pentru a lucra cu numere raționale (fracții). El vrea să poată aduna, scădea, compara și afișa fracții folosind operatorii standard din C++ (de exemplu: `f1 + f2`, `f1 == f2`, `cin » f1`). Acesta este un exercițiu perfect pentru a înțelege conceptul de **supraincărcare a operatorilor**.

---

## Scopul laboratorului

- înțelegerea conceptului de supraincărcare a operatorilor;
- implementarea operatorilor aritmetici (`+`, `*`, `+=`, `++`) și relaționali (`==`, `<`, `<=`, `!=`);
- implementarea operatorului unar (`-`);
- supraincărcarea operatorilor de stream (`«`, `»`) pentru citire și afișare;
- diferențierea dintre operatorii implementați ca funcții membre și cei implementați ca funcții `friend`;
- scrierea unei funcții helper private pentru simplificarea fracțiilor.

---

## Structura fișierelor

- **fractie.h** – declararea clasei `Fractie`;
- **fractie.cpp** – implementarea metodelor clasei;
- **main.cpp** – testarea funcționalităților.

# Fișierul `fractie.h`

```h
#ifndef FRACTIE_H
#define FRACTIE_H
#include <iostream> // Necesar pentru std::ostream si std::istream

class Fractie {
    int a; // numarator
    int b; // numitor

public:
    // Constructor cu parametri impliciti
    Fractie(int aa = 0, int bb = 1);

    // Metode specificate
    double getValoare() const; // cat face a/b
    Fractie getInv() const;    // b/a
    void setData(int aa, int bb); // modifica numaratorul si numitorul
    int getA() const;          // returneaza numaratorul
    int getB() const;          // returneaza numitorul

    // Operatori friend (aritmetici si stream)
    friend Fractie operator+(const Fractie& f1, const Fractie& f2);
    friend Fractie operator*(const Fractie& f1, const Fractie& f2);

    friend Fractie operator-(const Fractie& f);
    // transforma numerele in negatul lor, de ex 2/3 -> -2/3

    friend std::istream& operator>>(std::istream& in, Fractie& z);
    friend std::ostream& operator<<(std::ostream& out, const Fractie& z);

    // Operatori membri (aritmetici si relationali)
    Fractie& operator+=(const Fractie& other);

    bool operator==(const Fractie& other) const;
    bool operator!=(const Fractie& other) const;
    bool operator<(const Fractie& other) const;
    bool operator<=(const Fractie& other) const;

    // Operatori de incrementare (prefix si postfix)
    Fractie& operator++();      // forma prefixata (++f)
    Fractie operator++(int); // forma postfixata (f++)
};

#endif
```

# Fișierul `fractie.cpp`

```cpp
#include "fractie.h"
#include <iostream>

using namespace std;

// --- Constructor ---
Fractie::Fractie(int aa, int bb) {
    // TODO: Validati bb != 0. Daca bb == 0, aruncati o exceptie
    // sau setati fractia la 0/1 si afisati o eroare.
    // TODO: Initializati a = aa; b = bb;
}

// --- Metode ---
void Fractie::setData(int aa, int bb) {
    // TODO: Validati bb != 0.
    // TODO: Setati a = aa; b = bb;
}

double Fractie::getValoare() const {
    // TODO: Returnati (double)a / b;
}

Fractie Fractie::getInv() const {
    // TODO: Returnati Fractie(b, a); (constructorul va valida noul numitor 'a')
}

int Fractie::getA() const { /* TODO: return a; */ }
int Fractie::getB() const { /* TODO: return b; */ }

// --- Operatori friend ---
Fractie operator+(const Fractie& f1, const Fractie& f2) {
    // TODO: Calculati (f1.a * f2.b + f2.a * f1.b)
    // TODO: Numitorul comun (f1.b * f2.b)
    // TODO: Returnati Fractie(numarator_nou, numitor_nou);
}

Fractie operator*(const Fractie& f1, const Fractie& f2) {
    // TODO: Returnati Fractie(f1.a * f2.a, f1.b * f2.b);
}

Fractie operator-(const Fractie& f) {
    // TODO: Returnati Fractie(-f.a, f.b);
}
```

```cpp
std::istream& operator>>(std::istream& in, Fractie& z) {
    // TODO: Cititi numaratorul, un caracter (ex: '/') si numitorul
    // int aa, bb; char sep;
    // in >> aa >> sep >> bb;
    // TODO: Apelati z.setData(aa, bb) pentru validare sau verificati din nou.
    // TODO: Returnati 'in'.
}

std::ostream& operator<<(std::ostream& out, const Fractie& z) {
    // TODO: Afisati fractia (ex: "out << z.a << "/" << z.b;")
    // TODO: Returnati 'out'.
}

// --- Operatori membri ---
Fractie& Fractie::operator+=(const Fractie& other) {
    // TODO: *this = *this + other; (foloseste operatorul friend +)
    // TODO: Returnati *this;
}

bool Fractie::operator==(const Fractie& other) const {
    // TODO: Returnati (a == other.a && b == other.b);
}

bool Fractie::operator!=(const Fractie& other) const {
    // TODO: Returnati !(*this == other);
}

bool Fractie::operator<(const Fractie& other) const {
    // TODO: Aduceți la același numitor și comparați numărătorii
    // ex: return a * other.b < other.a * b;
}

bool Fractie::operator<=(const Fractie& other) const {
    // TODO: return (*this < other) || (*this == other);
}

Fractie& Fractie::operator++() {
    // TODO: (prefix) Aduna 1 la fractie (a/b + 1/1) => a = a + b;
    // TODO: Returnati *this;
}

Fractie Fractie::operator++(int) {
    // TODO: (postfix) Salvati o copie a *this (Fractie temp = *this;)
    // TODO: Apelati ++(*this); (operatorul prefix)
    // TODO: Returnati copia salvata (temp);
}
```

# Fișierul `main.cpp` (model)

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

int main() {

    // TODO 1: Creati doua obiecte f1(1, 2) si f2(3, 4) sau orice alte valori.

    // TODO 2: Testati operatorul << (afisare)
    // ex: cout << "f1 = " << f1 << endl;
    // ex: cout << "f2 = " << f2 << endl;

    // TODO 3: Testati operatorii aritmetici (+, *, - unar)
    // Fractie f_suma = f1 + f2;
    // Fractie f_prod = f1 * f2;
    // Fractie f_neg = -f1;
    // cout << "Suma: " << f_suma << endl;
    // cout << "Produs: " << f_prod << endl;
    // cout << "Negativ: " << f_neg << endl;

    // TODO 4: Testati operatorii relationali (==, !=, <, <=)
    // if (f1 < f2) { cout << "f1 este mai mica decat f2" << endl; }
    // if (f1 != f2) { ... }

    // TODO 5: Testati operatorii += si ++ (prefix si postfix)
    // f1 += f2;
    // cout << "f1 dupa += f2: " << f1 << endl;
    // cout << "Test prefix ++f1: " << ++f1 << endl;
    // cout << "Test postfix f1++: " << f1++ << endl;
    // cout << "f1 dupa postfix: " << f1 << endl;

    // TODO 6: Testati operatorul >> (citire)
    // Fractie f_citita;
    // cout << "Introduceti o fractie (format a/b): ";
    // cin >> f_citita;
    // cout << "Ati citit: " << f_citita << endl;

    // TODO 7: Testati metodele getInv() si getValoare()
    // cout << "Inversa lui f_citita: " << f_citita.getInv() << endl;
    // cout << "Valoarea lui f_citita: " << f_citita.getValoare() << endl;

    return 0;
}
```

# Explicații suplimentare

- **Supraincărcarea operatorilor** — Permite claselor definite de utilizator să se comporte similar cu tipurile de date primitive (de exemplu, să putem scrie `f1 + f2` în loc de `f1.adunare(f2)`).
- **Operator membru** — Se folosește când operatorul modifică starea obiectului curent (`this`) sau când operandul stâng *este* un obiect al clasei.

```
// f1 += f2; (f1 este 'this')
Fractie& operator+=(const Fractie& other);
```

- **Operator `friend`** — Se folosește pentru operatorii binari unde operandul stâng *nu* este un obiect al clasei (ca la `«` și `»`) sau pentru operatori simetrici (ca `+` și `*`) unde preferăm ca ambii operanzi să fie tratați la fel.

```
// cout << f1; (operandul stang este 'cout' de tip ostream)
friend std::ostream& operator<<(std::ostream& out, const Fractie& z);
// f1 + f2; (ambii sunt parametri)
friend Fractie operator+(const Fractie& f1, const Fractie& f2);
```

# Cerințe (9p + 1p oficiu)

1. **Implementare Clasa Fractie (5p)**
   - (2p) Constructor, `setData`, `getA`, `getB`, `getValoare`, `getInv`.
   - (1p) Operatori aritmetici: `+` (friend), `*` (friend), `-` (unar, friend), `+=` (membru).
   - (1p) Operatori relaționali: `==`, `!=`, `<`, `<=` (toți membri).
   - (1p) Operatori stream `«` (friend), `»` (friend) și operatorii `++` (prefix/postfix, membri).
2. **Testare în main.cpp (4p)**
   - (1p) Testarea constructorului și a operatorului `«`.
   - (1p) Testarea operatorilor aritmetici (`+`, `*`, `-`, `+=`).
   - (1p) Testarea operatorilor relaționali (ex: într-un `if`).
   - (1p) Testarea operatorilor `++` (prefix și postfix) și a operatorului `»`.
3. **(1p) Punct din oficiu.**

# Compilare și verificare

```
g++ -g main.cpp fractie.cpp -o lab5
./lab5
valgrind ./lab5
```

# 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 gasi linkul **GitHub Classroom** pentru Laboratorul 5:

   `https://classroom.github.com/a/3Zm1gDBB`

3. După acceptare, veți primi propriul repository, ex. `laborator-5-NumePrenume`.
4. Clonați repository-ul local:

   ```
   git clone
    ↪  git@github.com:Laborator-POO-2025-2026/laborator-5-NumePrenume.git
   cd laborator-5-NumePrenume
   ```

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

   ```
   git add .
   git commit -m "Implementare laborator 5 - Clasa Fractie"
   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.