----------------------------

ЗВІТ З ЛАБОРАТОРНОЇ РОБОТИ

за курсом «Об’єктно-орієнтоване програмування»

студента групи ПЗ-22-3

Казанікова Іллі Івановича

кафедра математичного забезпечення ЕОМ, ДНУ

2023/2024 навч.р.

Загальні вимоги:

1. Реалізація методів ініціалізації (конструктор по замовчуванню та конструктор з параметрами), копіювання (конструктор копіювання).

конструктор по замовчуванню:



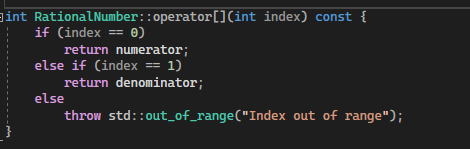
конструктор з параметрами:



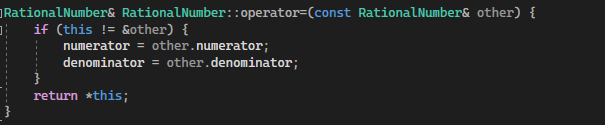
конструктор копіювання:



1. Індексації (перевантаження []):

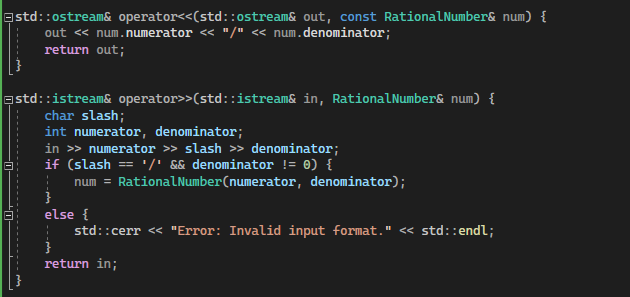


1. Присвоювання (перевантаження =):

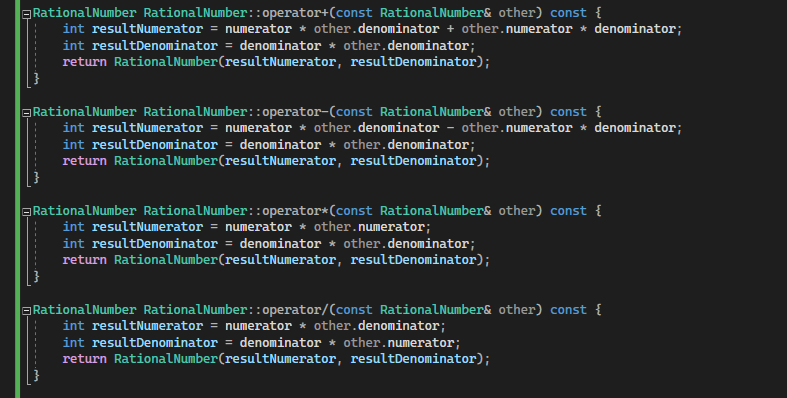


1. Перевантаження потокового введення/виведення таким чином, що повинно працювати введення як з файлу, так і з консолі, виведення в файл та консоль. Демонструвати, як стан об’єкту може бути збережений на диск та відновлений зі збереженого стану.

Перевантаження потокового введення/виведення:



Перевантаження арифметичних дій:



1. Застосування вказаної структури даних для розв’язання типової задачі.



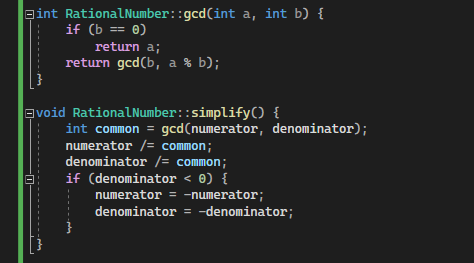
Індивідуальне завдання:

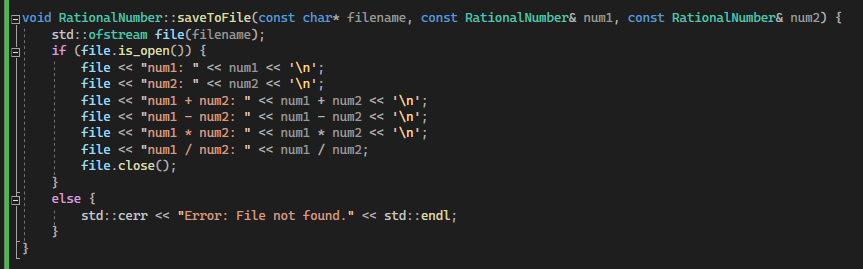
Варіант 9

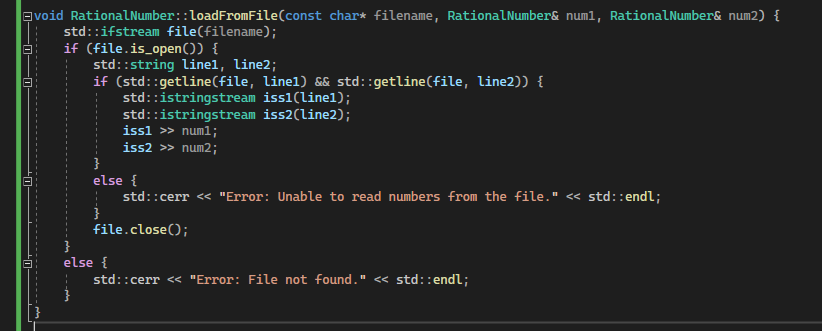
**Раціональні числа: різні форми зовнішнього представлення, точні обчислення. Реалізувати клас для збереження раціонального числа (чисельник, знаменник). Виконати перевантаження арифметичних операцій і операцій приведення типів. Передбачити автоматичне перетворення раціонального числа в нескорочуваний дріб.**

Всього є 3 файли: RationalNumber.h;Rational.cpp; main.cpp

Rational.cpp:

функція пошуку НСД: 

функція виведення результатів в файл:

функція читання чисел с файлу:

повний код файлів:

**RationalNumber.h**:

#include <iostream>

#include <fstream>

class RationalNumber {

private:

int numerator;

int denominator;

int gcd(int a, int b);

void simplify();

public:

RationalNumber();

RationalNumber(int num, int denom);

RationalNumber(const RationalNumber& other);

int operator[](int index) const;

RationalNumber& operator=(const RationalNumber& other);

RationalNumber operator+(const RationalNumber& other) const;

RationalNumber operator-(const RationalNumber& other) const;

RationalNumber operator\*(const RationalNumber& other) const;

RationalNumber operator/(const RationalNumber& other) const;

friend std::ostream& operator<<(std::ostream& out, const RationalNumber& num);

friend std::istream& operator>>(std::istream& in, RationalNumber& num);

void saveToFile(const char\* filename, const RationalNumber& num1, const RationalNumber& num2);

void loadFromFile(const char\* filename, RationalNumber& num1, RationalNumber& num2);

};

**RationalNumber.cpp:**

#include "RationalNumber.h"

#include <iostream>

#include <fstream>

#include <sstream>

RationalNumber::RationalNumber() : numerator(0), denominator(1) {}

RationalNumber::RationalNumber(int num, int denom) : numerator(num), denominator(denom) {

simplify();

}

RationalNumber::RationalNumber(const RationalNumber& other) : numerator(other.numerator), denominator(other.denominator) {}

int RationalNumber::gcd(int a, int b) {

if (b == 0)

return a;

return gcd(b, a % b);

}

void RationalNumber::simplify() {

int common = gcd(numerator, denominator);

numerator /= common;

denominator /= common;

if (denominator < 0) {

numerator = -numerator;

denominator = -denominator;

}

}

int RationalNumber::operator[](int index) const {

if (index == 0)

return numerator;

else if (index == 1)

return denominator;

else

throw std::out\_of\_range("Index out of range");

}

RationalNumber& RationalNumber::operator=(const RationalNumber& other) {

if (this != &other) {

numerator = other.numerator;

denominator = other.denominator;

}

return \*this;

}

RationalNumber RationalNumber::operator+(const RationalNumber& other) const {

int resultNumerator = numerator \* other.denominator + other.numerator \* denominator;

int resultDenominator = denominator \* other.denominator;

return RationalNumber(resultNumerator, resultDenominator);

}

RationalNumber RationalNumber::operator-(const RationalNumber& other) const {

int resultNumerator = numerator \* other.denominator - other.numerator \* denominator;

int resultDenominator = denominator \* other.denominator;

return RationalNumber(resultNumerator, resultDenominator);

}

RationalNumber RationalNumber::operator\*(const RationalNumber& other) const {

int resultNumerator = numerator \* other.numerator;

int resultDenominator = denominator \* other.denominator;

return RationalNumber(resultNumerator, resultDenominator);

}

RationalNumber RationalNumber::operator/(const RationalNumber& other) const {

int resultNumerator = numerator \* other.denominator;

int resultDenominator = denominator \* other.numerator;

return RationalNumber(resultNumerator, resultDenominator);

}

std::ostream& operator<<(std::ostream& out, const RationalNumber& num) {

out << num.numerator << "/" << num.denominator;

return out;

}

std::istream& operator>>(std::istream& in, RationalNumber& num) {

char slash;

int numerator, denominator;

in >> numerator >> slash >> denominator;

if (slash == '/' && denominator != 0) {

num = RationalNumber(numerator, denominator);

}

else {

std::cerr << "Error: Invalid input format." << std::endl;

}

return in;

}

void RationalNumber::saveToFile(const char\* filename, const RationalNumber& num1, const RationalNumber& num2) {

std::ofstream file(filename);

if (file.is\_open()) {

file << "num1: " << num1 << '\n';

file << "num2: " << num2 << '\n';

file << "num1 + num2: " << num1 + num2 << '\n';

file << "num1 - num2: " << num1 - num2 << '\n';

file << "num1 \* num2: " << num1 \* num2 << '\n';

file << "num1 / num2: " << num1 / num2;

file.close();

}

else {

std::cerr << "Error: File not found." << std::endl;

}

}

void RationalNumber::loadFromFile(const char\* filename, RationalNumber& num1, RationalNumber& num2) {

std::ifstream file(filename);

if (file.is\_open()) {

std::string line1, line2;

if (std::getline(file, line1) && std::getline(file, line2)) {

std::istringstream iss1(line1);

std::istringstream iss2(line2);

iss1 >> num1;

iss2 >> num2;

}

else {

std::cerr << "Error: Unable to read numbers from the file." << std::endl;

}

file.close();

}

else {

std::cerr << "Error: File not found." << std::endl;

}

}

**main.cpp:**

#include <iostream>

#include "RationalNumber.h"

int main() {

RationalNumber num1, num2;

RationalNumber loader;

int choice;

do {

std::cout << "Menu:" << std::endl;

std::cout << "1. Enter two rational numbers." << std::endl;

std::cout << "2. Perform addition." << std::endl;

std::cout << "3. Perform subtraction." << std::endl;

std::cout << "4. Perform multiplication." << std::endl;

std::cout << "5. Perform division." << std::endl;

std::cout << "6. Save to file." << std::endl;

std::cout << "7. Load from file." << std::endl;

std::cout << "8. Exit." << std::endl;

std::cout << "Enter your choice: ";

std::cin >> choice;

switch (choice) {

case 1:

std::cout << "Enter the first rational number (numerator/denominator): ";

std::cin >> num1;

std::cout << "Enter the second rational number (numerator/denominator): ";

std::cin >> num2;

break;

case 2:

std::cout << "Sum: " << num1 + num2 << std::endl;

break;

case 3:

std::cout << "Difference: " << num1 - num2 << std::endl;

break;

case 4:

std::cout << "Product: " << num1 \* num2 << std::endl;

break;

case 5:

std::cout << "Quotient: " << num1 / num2 << std::endl;

break;

case 6:

num1.saveToFile("rational\_number.txt", num1, num2);

std::cout << "Saved to file: rational\_number.txt" << std::endl;

break;

case 7:

loader.loadFromFile("input.txt", num1, num2);

std::cout << "Loaded numbers from file: num1 = " << num1 << ", num2 = " << num2 << std::endl;

break;

case 8:

std::cout << "Exiting the program." << std::endl;

break;

default:

std::cerr << "Error: Invalid choice. Please enter a valid option." << std::endl;

}

} while (choice != 8);

return 0;

}