# МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Кемеровский государственный университет»

институт цифры кафедра цифровых технологий

# ОТЧЕТ

о выполнении

Лабораторной работы №4 **«ООП на С++: переопределение операторов»**

по курсу «Языки программирования»

студента 1 курса

Коник Ильи Николаевича

(ФИО полностью)

направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

направленность (профиль) подготовки «Интеллектуальные информационные системы и анализ данных»

Преподаватель:

Дуванов И.О.

Кемерово 2025

# Тема: «ООП на С++: переопределение операторов» *(9 баллов)*

## Задание

1. В соответствии с вариантами задания и приведённой ниже спецификацией реализуйте программу на языке C++.
2. Разработайте тестовые задания, и протестируйте программу одним из методов тестирования.

## Требования к оформлению программ:

* 1. **Содержание**. Программа должна делать только то, что предусмотрено заданием. Не нужно выполнять лишнюю работу.
  2. **Спецификация.** В преамбуле программы в комментариях указывать сведения:
     + Кто выполнил.
     + Что делает программа (кратко).
     + Что на входе, имена входных файлов указываются.
     + Что на выходе (что является результатов работы программы).

# Ввод и вывод

* + - Приглашение пользователю. (Например, сколько чисел, какого типа и через какой разделитель нужно вводить).
    - Контрольный вывод (все введенные данные выводить на экран, и только после этого выполнять необходимые вычисления.)
    - «Защита от дурака». Проверять вводимые данные на корректность. Например, если необходимо считать количество чего – то, то эта величина не может быть отрицательной и т.д.
  1. **Структура кода.** Набираемый код должен быть хорошо структурированным. Использовать:
     + Отступы.
     + Комментарии – поясняют решение программы.
     + Осмысленные названия переменных.
     + Пояснения о назначении переменных в комментариях (кроме счетчиков).

# Декомпозиция кода

* + - Функциональная
    - Объектно-ориентированная

# Многофайловые проекты

* + - Классы определять в отдельном h-файле, а все методы классов – в соответствующем cpp - файле. Созданный заголовочный файл подключать к проекту.
    - Методы класса реализовывать не в определении класса.
    - cpp-файл и соответствующий ему h-файл называть одинаково. В качестве

названия выбирать имя того класса, который определен в соответствующем модуле.

## Работа в классе:

Рассмотрите реализацию класса из прикрепленных файлов кода lab4. Проанализируйте работу программы.

## Задания для самостоятельного выполнения:

***Задание 1.***

Напишите класс комплексного числа Complex.

В классе должно быть два поля re и im, обозначающие действительную и мнимую части комплексного числа.

В классе должно быть два конструктора: конструктор по умолчанию Complex(), который присваивает полям re и im нули, и конструктор вида Complex(int x, int y), который присваивает полям re и im значения x и y соответственно.

1. Добавьте конструктор копирования и переопределите операцию присваивания "=".
2. Добавьте переопределение операций:
   1. Арифметические операции "+","\*", "-", "/"
   2. Арифметические операции декремента в префиксной и постфиксной формах "--" (вычитание 1 от полей)
   3. Арифметические операции инкремента в префиксной и постфиксной формах "++" (добавление 1 к полям)
   4. Логические операции ">", "<", "==", "!=" (Сравнение чисел осуществлять по модулю комплексного числа)
3. Переопределите операторы для ввода-вывода (>>, <<), вывод комплексного числа осуществлять в виде: x + y\* i
4. Продемонстрируйте и протестируйте работу конструктора копирования, всех переопределенных операторов и операторов для ввода-вывода.

Complex.cpp

#include "Complex.h"

Complex::Complex() {

    re = new double(0);

    im = new double(0);

}

Complex::Complex(double x, double y) {

    this->re = new double(x);

    this->im = new double(y);

}

Complex::Complex(const Complex& z) {

    this->re = new double(\*z.re);

    this->im = new double(\*z.im);

}

Complex::~Complex() {

    delete re;

    delete im;

}

// Установка действительной части

void Complex::SetRe(double x) {

    \*re = x;

}

// Установка мнимой части

void Complex::SetIm(double y) {

    \*im = y;

}

// Получение действительной части

double Complex::GetRe() const {

    return \*re;

}

// Получение мнимой части

double Complex::GetIm() const {

    return \*im;

}

// Вычисление модуля комплексного числа

double Complex::Abs() const {

    return std::sqrt((\*re) \* (\*re) + (\*im) \* (\*im));    // Формула: sqrt(re^2 + im^2)

}

// Вычисление аргумента комплексного числа

double Complex::Arg() const {

    return std::atan2(\*im, \*re);    // Формула: atan2(im, re)

}

// Вывод числа в алгебраической форме

void Complex::Print() const {

    std::cout << \*re << " + i\*" << \*im << std::endl;    // Формат: a + bi

}

// Вывод числа в тригонометрической форме

void Complex::TrigPrint() const {

    double rho = Abs();

    double phi = Arg();

    std::cout << rho << " \* (cos(" << phi << ") + i\*sin(" << phi << "))" << std::endl;  // Формат: r \* (cos(phi) + i\*sin(phi))

}

// Вывод числа в экспоненциальной форме

void Complex::ExpPrint() const {

    double rho = Abs();

    double phi = Arg();

    std::cout << rho << " \* e^(i\*" << phi << ")" << std::endl;  // Формат: r \* e^(i\*phi)

}

Complex Complex::operator+(const Complex& z) const {

    return Complex(\*re + \*z.re, \*im + \*z.im);   // Формула: (a + bi) + (c + di) = (a + c) + (b + d)i

}

Complex Complex::operator-(const Complex& z) const {

    return Complex(\*re - \*z.re, \*im - \*z.im);   // Формула: (a + bi) + (c + di) = (a + c) + (b + d)i

}

Complex Complex::operator\*(const Complex& z) const {

    return Complex(\*re \* \*z.re - \*im \* \*z.im, \*re \* \*z.im + \*im \* \*z.re);   // Формула: (a + bi) \* (c + di) = (a\*c - b\*d) + (a\*d + b\*c)i   // Формула: (a + bi) + (c + di) = (a + c) + (b + d)i

}

Complex Complex::operator/(const Complex& z) const {

    double denominator = \*z.re \* \*z.re + \*z.im \* \*z.im; // Знаменатель: c^2 + d^2

    return Complex((\*re \* \*z.re + \*im \* \*z.im) / denominator, (\*im \* \*z.re - \*re \* \*z.im) / denominator);  // Формула: (a + bi) + (c + di) = (a + c) + (b + d)i

}

Complex &Complex::operator++() {

    \*this->re += 1;

    \*this->im += 1;

    return \*this;

}

Complex &Complex::operator--() {

    \*this->re -= 1;

    \*this->im -= 1;

    return \*this;

}

Complex &Complex::operator++(int) {

    \*this->re += 1;

    \*this->im += 1;

    return \*this;

}

Complex &Complex::operator--(int) {

    \*this->re -= 1;

    \*this->im -= 1;

    return \*this;

}

bool Complex::operator==(const Complex &z) const {

    return \*re == \*z.re && \*im == \*z.im;

}

bool Complex::operator!=(const Complex &z) const {

    return !(\*this == z);

}

bool Complex::operator<(const Complex &z) const {

    return this->Abs() < z.Abs();

}

bool Complex::operator>(const Complex &z) const {

    return this->Abs() > z.Abs();

}

bool Complex::operator<=(const Complex &z) const {

    return this->Abs() <= z.Abs();

}

bool Complex::operator>=(const Complex &z) const {

    return this->Abs() >= z.Abs();

}

// Сложение двух комплексных чисел

Complex Complex::Add(const Complex& z) const {

    return Complex(\*re + \*z.re, \*im + \*z.im);   // Формула: (a + bi) + (c + di) = (a + c) + (b + d)i

}

// Вычитание комплексных чисел

Complex Complex::Sub(const Complex& z) const {

    return Complex(\*re - \*z.re, \*im - \*z.im);   // Формула: (a + bi) - (c + di) = (a - c) + (b - d)i

}

// Умножение комплексных чисел

Complex Complex::Mult(const Complex& z) const {

    return Complex(\*re \* \*z.re - \*im \* \*z.im, \*re \* \*z.im + \*im \* \*z.re);   // Формула: (a + bi) \* (c + di) = (a\*c - b\*d) + (a\*d + b\*c)i

}

// Деление комплексных чисел

Complex Complex::Div(const Complex& z) const {

    double denominator = \*z.re \* \*z.re + \*z.im \* \*z.im; // Знаменатель: c^2 + d^2

    return Complex((\*re \* \*z.re + \*im \* \*z.im) / denominator, (\*im \* \*z.re - \*re \* \*z.im) / denominator);   // Формула: (a + bi) / (c + di) = [(a\*c + b\*d) + (b\*c - a\*d)i] / (c^2 + d^2)

}

Complex operator+(int number, const Complex& z) {

    return Complex(z.GetRe() + number, z.GetIm());

}

Complex operator+(const Complex& z, int number) {

    return Complex(z.GetRe() + number, z.GetIm());

}

Complex.h

//

// Created by ivan on 4/4/25.

//

#ifndef COMPLEX\_H

#define COMPLEX\_H

#include <iostream>

#include <cmath>

class Complex {

    double \*re; // Указатель на действительную часть числа

    double \*im; // Указатель на мнимую часть числа

public:

    Complex(); // Конструктор по умолчанию (инициализирует число как 0 + 0i)

    Complex(const Complex& z);

    Complex(double x, double y); // Конструктор с параметрами (инициализирует число как x + yi)

    ~Complex();

    void SetRe(double x); // Устанавливает действительную часть числа

    void SetIm(double y); // Устанавливает мнимую часть числа

    double GetRe() const; // Возвращает действительную часть числа

    double GetIm() const; // Возвращает мнимую часть числа

    double Abs() const; // Возвращает модуль комплексного числа

    double Arg() const; // Возвращает аргумент комплексного числа

    void Print() const; // Выводит число в алгебраической форме (a + bi)

    void TrigPrint() const; // Выводит число в тригонометрической форме (r \* (cos(phi) + i\*sin(phi)))

    void ExpPrint() const; // Выводит число в экспоненциальной форме (r \* e^(i\*phi))

    Complex operator+(const Complex& z) const;

    Complex operator-(const Complex& z) const;

    Complex operator\*(const Complex& z) const;

    Complex operator/(const Complex& z) const;

    Complex& operator++();

    Complex& operator--();

    // постфиксная запись

    Complex& operator++(int);

    Complex& operator--(int);

    bool operator==(const Complex& z) const;

    bool operator!=(const Complex& z) const;

    bool operator<(const Complex& z) const;

    bool operator>(const Complex& z) const;

    bool operator<=(const Complex& z) const;

    bool operator>=(const Complex& z) const;

    Complex Add(const Complex &z) const; // Сложение двух комплексных чисел

    Complex Sub(const Complex &z) const; // Вычитание комплексных чисел

    Complex Mult(const Complex &z) const; // Умножение комплексных чисел

    Complex Div(const Complex &z) const; // Деление комплексных чисел

};

Complex operator+(int number, const Complex& z);

Complex operator+(const Complex& z, int number);

// Complex operator+(const Complex& z1, const Complex& z2);

#endif //COMPLEX\_H

Main.cpp

#include <iostream>

#include "Complex.h"

int main() {

    std::cout << "Enter real and imaginary parts for first complex number: ";

    double re1, im1;

    std::cin >> re1 >> im1;

    Complex z1(re1, im1);

    std::cout << "Enter real and imaginary parts for second complex number: ";

    double re2, im2;

    std::cin >> re2 >> im2;

    Complex z2(re2, im2);

    // Display numbers

    std::cout << "First number: ";

    z1.Print();

    std::cout << "Second number: ";

    z2.Print();

    // Arithmetic operations

    Complex sum = z1 + z2;

    std::cout << "Sum: ";

    sum.Print();

    Complex diff = z1 - z2;

    std::cout << "Difference: ";

    diff.Print();

    Complex prod = z1 \* z2;

    std::cout << "Product: ";

    prod.Print();

    Complex quot = z1 / z2;

    std::cout << "Quotient: ";

    quot.Print();

    // Increment/Decrement

    Complex incr1 = ++z1;

    std::cout << "Prefix increment: ";

    incr1.Print();

    Complex incr2 = z1++;

    std::cout << "Postfix increment: ";

    incr2.Print();

    // Comparison

    std::cout << "z1 == z2: " << (z1 == z2) << std::endl;

    std::cout << "z1 >= z2: " << (z1 >= z2) << std::endl;

    // Scalar addition

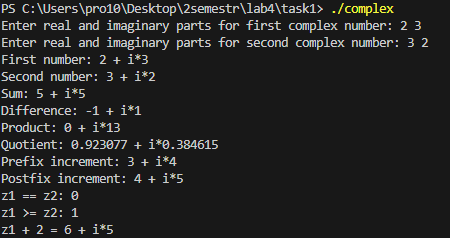
    std::cout << "z1 + 2 = ";

    Complex z3 = 2 + z1;

    z3.Print();

    return 0;

}



## Задание 2.

Напишите класс Rational (рациональное число). Конструктор класса должен получать на вход два числа типа int (числитель и знаменатель). Считайте, что по умолчанию числитель равен 0, а знаменатель — 1.

Переопределите бинарные операторы сложения, вычитания, умножения и деления (работающие в том числе и с аргументами типа int), унарные плюс и минус. Предусмотрите функции-члены Numerator и Denominator для получения числителя и знаменателя несократимого представления этой дроби (знаменатель должен быть положительным). Переопределите также операторы +=, -=, \*= и /=. Не забудьте определить операторы == и !=.

Для получения несократимого представления дроби используйте функцию std::gcd стандартной библиотеки.

Rational.cpp

#include "Rational.h"

void Rational::Normalize() {

    if (denominator == 0) {

        throw std::invalid\_argument("Denominator cannot be zero");

    }

    // Приводим знаменатель к положительному виду

    if (denominator < 0) {

        numerator = -numerator;

        denominator = -denominator;

    }

    // Сокращаем дробь

    int gcd = std::gcd(std::abs(numerator), std::abs(denominator));

    if (gcd != 0) {

        numerator /= gcd;

        denominator /= gcd;

    }

}

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

    Normalize();

}

int Rational::Numerator() const { return numerator; }

int Rational::Denominator() const { return denominator; }

Rational Rational::operator+(const Rational& rhs) const {

    return Rational(

        numerator \* rhs.denominator + rhs.numerator \* denominator,

        denominator \* rhs.denominator

    );

}

Rational Rational::operator-(const Rational& rhs) const {

    return Rational(

        numerator \* rhs.denominator - rhs.numerator \* denominator,

        denominator \* rhs.denominator

    );

}

Rational Rational::operator\*(const Rational& rhs) const {

    return Rational(

        numerator \* rhs.numerator,

        denominator \* rhs.denominator

    );

}

Rational Rational::operator/(const Rational& rhs) const {

    return Rational(

        numerator \* rhs.denominator,

        denominator \* rhs.numerator

    );

}

Rational Rational::operator+() const {

    return \*this;

}

Rational Rational::operator-() const {

    return Rational(-numerator, denominator);

}

Rational& Rational::operator+=(const Rational& rhs) {

    \*this = \*this + rhs;

    return \*this;

}

Rational& Rational::operator-=(const Rational& rhs) {

    \*this = \*this - rhs;

    return \*this;

}

Rational& Rational::operator\*=(const Rational& rhs) {

    \*this = \*this \* rhs;

    return \*this;

}

Rational& Rational::operator/=(const Rational& rhs) {

    \*this = \*this / rhs;

    return \*this;

}

bool Rational::operator==(const Rational& rhs) const {

    return numerator == rhs.numerator && denominator == rhs.denominator;

}

bool Rational::operator!=(const Rational& rhs) const {

    return !(\*this == rhs);

}

// Реализации дружественных операторов с int

Rational operator+(const Rational& r, int value) {

    return r + Rational(value);

}

Rational operator+(int value, const Rational& r) {

    return Rational(value) + r;

}

Rational operator-(const Rational& r, int value) {

    return r - Rational(value);

}

Rational operator-(int value, const Rational& r) {

    return Rational(value) - r;

}

Rational operator\*(const Rational& r, int value) {

    return r \* Rational(value);

}

Rational operator\*(int value, const Rational& r) {

    return Rational(value) \* r;

}

Rational operator/(const Rational& r, int value) {

    return r / Rational(value);

}

Rational operator/(int value, const Rational& r) {

    return Rational(value) / r;

}

std::ostream& operator<<(std::ostream& os, const Rational& r) {

    os << r.numerator << "/" << r.denominator;

    return os;

}

std::istream& operator>>(std::istream& is, Rational& r) {

    int num, denom = 1;

    char slash;

    if (is >> num) {

        if (is >> slash && slash == '/') {

            if (is >> denom) {

                r = Rational(num, denom);

                return is;

            }

        } else {

            // Если нет слэша, значит ввели целое число

            is.putback(slash);

            r = Rational(num);

            return is;

        }

    }

    is.setstate(std::ios::failbit);

    return is;

}

Rational.h

#ifndef RATIONAL\_H

#define RATIONAL\_H

#include <iostream>

#include <numeric> // Для std::gcd

#include <stdexcept> // Для std::invalid\_argument

class Rational {

private:

    int numerator;

    int denominator;

    void Normalize(); // Приведение к несократимому виду

public:

    // Конструкторы

    Rational(int num = 0, int denom = 1);

    // Получение числителя и знаменателя

    int Numerator() const;

    int Denominator() const;

    // Арифметические операторы

    Rational operator+(const Rational& rhs) const;

    Rational operator-(const Rational& rhs) const;

    Rational operator\*(const Rational& rhs) const;

    Rational operator/(const Rational& rhs) const;

    // Унарные операторы

    Rational operator+() const;

    Rational operator-() const;

    // Операторы присваивания

    Rational& operator+=(const Rational& rhs);

    Rational& operator-=(const Rational& rhs);

    Rational& operator\*=(const Rational& rhs);

    Rational& operator/=(const Rational& rhs);

    // Операторы сравнения

    bool operator==(const Rational& rhs) const;

    bool operator!=(const Rational& rhs) const;

    // Дружественные операторы для работы с int

    friend Rational operator+(const Rational& r, int value);

    friend Rational operator+(int value, const Rational& r);

    friend Rational operator-(const Rational& r, int value);

    friend Rational operator-(int value, const Rational& r);

    friend Rational operator\*(const Rational& r, int value);

    friend Rational operator\*(int value, const Rational& r);

    friend Rational operator/(const Rational& r, int value);

    friend Rational operator/(int value, const Rational& r);

    // Ввод/вывод

    friend std::ostream& operator<<(std::ostream& os, const Rational& r);

    friend std::istream& operator>>(std::istream& is, Rational& r);

};

#endif // RATIONAL\_H

Main.cpp

#include "Rational.h"

#include <iostream>

#include <cassert>

int main() {

    // Тестирование конструктора

    Rational r1(6, 8);

    assert(r1.Numerator() == 3 && r1.Denominator() == 4);

    Rational r2(-2, -4);

    assert(r2.Numerator() == 1 && r2.Denominator() == 2);

    Rational r3(3, -9);

    assert(r3.Numerator() == -1 && r3.Denominator() == 3);

    // Тестирование арифметических операций

    Rational sum = r1 + r2;

    assert(sum.Numerator() == 5 && sum.Denominator() == 4);

    Rational diff = r1 - r2;

    assert(diff.Numerator() == 1 && diff.Denominator() == 4);

    Rational prod = r1 \* r2;

    assert(prod.Numerator() == 3 && prod.Denominator() == 8);

    Rational quot = r1 / r2;

    assert(quot.Numerator() == 3 && quot.Denominator() == 2);

    // Тестирование операторов с int

    Rational r4 = r1 + 1;

    assert(r4.Numerator() == 7 && r4.Denominator() == 4);

    Rational r5 = 2 - r2;

    assert(r5.Numerator() == 3 && r5.Denominator() == 2);

    // Тестирование операторов присваивания

    r1 += r2;

    assert(r1.Numerator() == 5 && r1.Denominator() == 4);

    r1 \*= r3;

    assert(r1.Numerator() == -5 && r1.Denominator() == 12);

    // Тестирование ввода/вывода

    Rational r6;

    std::cout << "Enter fraction (format a/b): ";

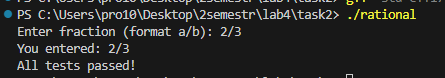
    std::cin >> r6;

    std::cout << "You entered: " << r6 << std::endl;

    std::cout << "All tests passed!" << std::endl;

    return 0;

}



# Критерии оценки:

Если программа компилируется и оформлена по требованиям, то за лабораторную работу начисляются баллы.

Максимально можно получить 9 баллов:

* Задание 1 – 4 балла
* Задание 2 – 5 баллов