**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра «Информационная безопасность»

Лабораторная работа №3

по дисциплине «Разработка безопасного программного обеспечения»

Понятие класса. Члены класса. Друзья класса. Перегрузка операций

Вариант 25

Выполнили студенты группы: БАС 2101

Сотников Герман Константинович

Руководитель: Барков В.В.

Москва 2023

**Цель работы**

Научиться создавать простые классы, создавать конструкторы, перегружать операции, создавать объекты классов и передавать их в функции.

**Задание**

В ходе выполнения лабораторной работы требуется разработать статическую библиотеку для работы с комплексными числами и рациональными дробями и консольное приложение, демонстрирующее работу библиотеки.

#### **Задание 1**

Все общедоступные функции и классы должны быть определены в модуле Math в пространстве имен Math.

Класс комплексных чисел Complex должен содержать компоненты, указанные в таблицах ниже.

Таблица 1 – Данные-члены класса Complex (варианты 1-15)

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Уровень доступа | Дополнительная информация |
| Любое (например, m\_re) | double | Закрытый | Хранит реальную часть комплексного числа |
| Любое (например, m\_im) | double | Закрытый | Хранит мнимую часть комплексного числа |

Таблица 2 – Данные-члены класса Complex (варианты 15-30)

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Уровень доступа | Дополнительная информация |
| Любое (например, m\_mod) | double | Закрытый | Хранит модуль комплексного числа |
| Любое (например, m\_arg) | double | Закрытый | Хранит аргумент комплексного числа |

Таблица 3 – Конструкторы класса Complex (варианты 1-30)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Варианты 1-15 | Варианты 15-30 |
| Конструктор по умолчанию | Общедоступный | Инициализирует реальную и мнимую часть значениями 0 | Инициализирует модуль и аргумент значениями 0 |
| Конструктор инициализации | Общедоступный | Инициализирует реальную и мнимую часть переданными параметрами | Получает в качестве параметров реальную и мнимую часть, вычисляет модуль и аргумент и инициализирует соответствующие значения в классе. |
| Конструктор преобразования | Общедоступный | Инициализирует реальную часть переданным значением, мнимую часть значением 0 | Инициализирует модуль переданным значением, аргумент значением 0 |

Таблица 3 – Статические функции-члены класса Complex (варианты 1-30)

|  |  |  |
| --- | --- | --- |
| Название | Уровень доступа | Комментарий |
| FromExponentialForm | Общедоступный | Принимает на вход модуль и аргумент и создает новый экземпляр класса Complex |
| FromAlgebraicForm | Общедоступный | Принимает на вход реальную и мнимую часть и создает новый экземпляр класса Complex |

Таблица 4 – Функции-члены класса Complex (варианты 1-30)

|  |  |  |
| --- | --- | --- |
| Название | Уровень доступа | Комментарий |
| Re | Общедоступный | Возвращает реальную часть комплексного числа |
| Im | Общедоступный | Возвращает мнимую часть комплексного числа |
| Mod | Общедоступный | Возвращает модуль комплексного числа |
| Arg | Общедоступный | Возвращает аргумент комплексного числа |

Таблица 5 – Перегруженные операции класса Complex, являющиеся функциями-членами (варианты 1-30)

|  |  |  |
| --- | --- | --- |
| Название | Уровень доступа | Комментарий |
| operator double | Общедоступный | Операция **явного** преобразования в тип double. Возвращает реальную часть комплексного числа. |
| operator- | Общедоступный | Операция унарный минус. Возвращает копию объекта, значения реальной и мнимой части которого имеют противоположный знак (по сравнению с предыдущим значением) |
| operator++ | Общедоступный | Префиксная форма инкремента. Изменяет состояние текущего объекта путем добавления к реальной части текущего числа значения 1 и возвращает ссылку на текущий объект |
| operator++ | Общедоступный | Постфиксная форма инкремента. Изменяет состояние текущего объекта путем добавления к реальной части текущего числа значения 1 и возвращает копию прежнего объекта |
| operator-- | Общедоступный | Префиксная форма декремента. Изменяет состояние текущего объекта путем вычитания из реальной части текущего числа значение 1 и возвращает ссылку на текущий объект |
| operator-- | Общедоступный | Постфиксная форма декремента. Изменяет состояние текущего объекта путем вычитания из реальной части значение 1 и возвращает копию прежнего объекта |
| operator+= | Общедоступный | Составная операция присваивания со сложением. Изменяет состояние текущего объекта путем прибавления к текущему значению значения параметра (комплексное число) и возвращает ссылку на текущий объект |
| operator-= | Общедоступный | Составная операция присваивания с вычитанием. Изменяет состояние текущего объекта путем вычитания из текущего значения значения параметра (комплексное число) и возвращает ссылку на текущий объект |
| operator\*= | Общедоступный | Составная операция присваивания с умножением. Изменяет состояние текущего объекта путем умножения текущего значения на значение параметра (комплексное число) и возвращает ссылку на текущий объект |
| operator/= | Общедоступный | Составная операция присваивания с делением. Изменяет состояние текущего объекта путем деления текущего значения на значение параметра (комплексное число) и возвращает ссылку на текущий объект |

Таблица 6 – Перегруженные операции класса Complex, являющиеся глобальными (свободными) функциями (варианты 1-30)

|  |  |
| --- | --- |
| Функция | Комментарий |
| operator+ | Складывает два комплексных числа и возвращает копию вновь созданного объекта |
| operator- | Вычитает два комплексных числа и возвращает копию вновь созданного объекта |
| operator\* | Умножает два комплексных числа и возвращает копию вновь созданного объекта |
| operator/ | Делит два комплексных числа и возвращает копию вновь созданного объекта |
| operator ""i | Определяет пользовательский литерал вида 6.0i. Возвращает комплексное число с мнимой частью, переданной в качестве параметра |
| operator ""i | Определяет пользовательский литерал вида 6i. Возвращает комплексное число с мнимой частью, переданной в качестве параметра |
| operator<< | Определяет операцию вывода комплексного числа в поток в виде x + yi |

Перегруженные операции сложения, вычитания, умножения, деления и их краткие формы должны работать даже если один из аргументов является типом double (левый или правый). При этом дополнительно перегружать операции с параметром типа double не следует.

В пространстве имен Math разработать математические функции, указанные в таблице 7.

Таблица 7 – Разрабатываемые математические функции

|  |  |
| --- | --- |
| Функция | Комментарий |
| FindGreatestCommonDivisor | Возвращает наибольший общий делитель двух чисел.  Для вычисления используйте алгоритм Эвклида:  Пусть a и b целые положительные числа  Пусть b <= a и r остаток от деления a на b.  Тогда:  1. Если b = 0, тогда НОД = a.  2. Иначе a = b, b = r. Перейти к 1. |
| FindLeastCommonMultiple | Возвращает наименьшее общее кратное двух чисел.  Для вычисления можно использовать следующую формулу: ,  где lcm – наименьшее общее кратное,  gcd(x,y) – наибольший общий делитель чисел x и y. |

Класс Rational, представляющий дробь, должен содержать следующие компоненты, представленные в таблицах 8-12.

Конструктор, а также все модифицирующие состояния функции-члены должны нормализовать дробь, после завершения внесения изменений:

1. Дробь должна быть сокращена
2. Знак дроби должен храниться в числителе
3. Знаменатель должен быть положительным

Таблица 8 – Данные-члены класса Rational (варианты 1-30)

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Уровень доступа | Дополнительная информация |
| Любое (например, m\_nominator) | int | Закрытый | Хранит числитель дроби |
| Любое (например, m\_denominator) | int | Закрытый | Хранит знаменатель дроби |

Таблица 9 – Конструкторы класса Rational (варианты 1-30)

|  |  |  |
| --- | --- | --- |
|  |  | Варианты 1-30 |
| Конструктор по умолчанию | Общедоступный | Инициализирует числитель значением 0, знаменатель значением 1. |
| Конструктор инициализации | Общедоступный | Инициализирует числитель и знаменатель переданными параметрами. Нормализует представления дроби: производит сокращение дроби и делает знаменатель положительным (не меняя значения дроби) |
| Конструктор преобразования | Общедоступный | Инициализирует числитель переданным значением, знаменатель значением |

Таблица 10 – Функции-члены класса Rational (варианты 1-30)

|  |  |  |
| --- | --- | --- |
| Название | Уровень доступа | Комментарий |
| Nominator | Общедоступный | Возвращает числитель рациональной дроби |
| Denominator | Общедоступный | Возвращает знаменатель рациональной дроби |

Таблица 11 – Перегруженные операции класса Rational, являющиеся функциями-членами (варианты 1-30)

|  |  |  |
| --- | --- | --- |
| Название | Уровень доступа | Комментарий |
| operator double | Общедоступный | Операция **явного** преобразования в тип double. Возвращает результат деления числителя на знаменатель. |
| operator- | Общедоступный | Операция унарный минус. Возвращает копию объекта, значение числителя которого имеет противоположный знак (по сравнению с предыдущим значением) |
| operator++ | Общедоступный | Префиксная форма инкремента. Изменяет состояние текущего объекта путем добавления к дроби значения 1 и возвращает ссылку на текущий объект |
| operator++ | Общедоступный | Постфиксная форма инкремента. Изменяет состояние текущего объекта путем добавления к дроби значения 1 и возвращает копию прежнего объекта |
| operator-- | Общедоступный | Префиксная форма декремента. Изменяет состояние текущего объекта путем вычитания из дроби значения 1 и возвращает ссылку на текущий объект |
| operator-- | Общедоступный | Постфиксная форма декремента. Изменяет состояние текущего объекта путем вычитания из дроби значения 1 и возвращает копию прежнего объекта |
| operator+= | Общедоступный | Составная операция присваивания со сложением. Изменяет состояние текущего объекта путем прибавления к текущему значению значения параметра (рациональная дробь) и возвращает ссылку на текущий объект |
| operator-= | Общедоступный | Составная операция присваивания с вычитанием. Изменяет состояние текущего объекта путем вычитания из текущего значения значения параметра (рациональная дробь) и возвращает ссылку на текущий объект |
| operator\*= | Общедоступный | Составная операция присваивания с умножением. Изменяет состояние текущего объекта путем умножения текущего значения на значение параметра (рациональная дробь) и возвращает ссылку на текущий объект |
| operator/= | Общедоступный | Составная операция присваивания с делением. Изменяет состояние текущего объекта путем деления текущего значения на значение параметра (рациональная дробь) и возвращает ссылку на текущий объект |

Таблица 12 – Перегруженные операции класса Rational, являющиеся глобальными (свободными) функциями (варианты 1-30)

|  |  |
| --- | --- |
| Функция | Комментарий |
| operator+ | Складывает две рациональные дроби и возвращает копию вновь созданного объекта |
| operator- | Вычитает две рациональные дроби и возвращает копию вновь созданного объекта |
| operator\* | Умножает две рациональные дроби и возвращает копию вновь созданного объекта |
| operator/ | Делит две рациональные дроби и возвращает копию вновь созданного объекта |
| operator == | Сравнивает две рациональные дроби. Если они равны, вовращает true. В противном случае возвращает false |
| operator <=>  или  operator >  operator <  operator <=  operator >= | Функции сравнения рациональных дробей |
| operator<< | Определяет операцию вывода рациональной дроби в поток в виде x/ y |

Перегруженные операции сложения, вычитания, умножения, деления и их краткие формы должны работать даже если один из аргументов является типом int (левый или правый). При этом дополнительно перегружать операции с параметром типа int не следует.

За основу следует взять проект <https://github.com/v-v-barkov-mtuci/CppLab5>

Разработать необходимые классы таким образом, чтобы тесты скомпилировались и выполнились без ошибок.

#### **Задание 2**

Разработайте для объектов вашего класса предложенные в каждом варианте пользовательские функции.

Прототипы функций:

Complex f(const Complex &z);

Rational f(const Rational &r);

double f(double x);

Разработайте функцию main, организующую ввод данных и демонстрацию работы разработанных функций

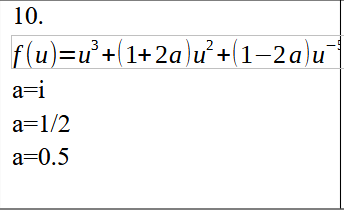


Рисунок 1. Индивидуальное задание для варианта 10.

**Ход работы**

**Задание №1**

**Код программы**

**Math.ixx:**

#include <iostream>

#include <math.h>

export module Math;

export namespace Math

//123

{

export class Complex

{

private:

double m\_mod;

double m\_arg;

public:

Complex()

{

m\_mod = 0;

m\_arg = 0;

}

Complex(double real, double image)

{

m\_mod = sqrt(real \* real + image \* image);

m\_arg = atan2(image, real);

if (m\_mod < 6.6613381477509392e-15) {

m\_mod = 0;

}

if ((m\_arg > 3.1415926535897) && (m\_arg < 3.1415926535898)) {

m\_arg = 0;

}

}

Complex(double a)

{

m\_mod = a;

m\_arg = 0;

}

static Complex FromExponentialForm(double mod, double arg)

{

return Complex(mod \* cos(arg), mod \* sin(arg));

}

static Complex FromAlgebraicForm(double real, double img)

{

Complex a;

a.m\_mod = sqrt(real \* real + img \* img);

a.m\_arg = atan2(img, real);

return a;

}

double Re() const

{

return m\_mod \* cos(m\_arg);

}

double Im() const

{

return m\_mod \* sin(m\_arg);

}

double Mod() const

{

return m\_mod;

}

double Arg() const

{

return m\_arg;

}

explicit operator double() const {

return this->Re();

}

Complex operator-() const

{

return Complex(-1 \* this->Re(), -1 \* this->Im());

}

Complex& operator++()

{

double real = this->Re() + 1;

double img = this->Im();

this->m\_mod = sqrt(real \* real + img \* img);

this->m\_arg = atan2(img, real);

return \*this;

}

Complex operator++ (int) {

Complex copy(\*this);

double real = this->Re() + 1;

double img = this->Im();

this->m\_mod = sqrt(real \* real + img \* img);

this->m\_arg = atan2(img, real);

return copy;

}

Complex& operator--()

{

double real = this->Re() - 1;

double img = this->Im();

this->m\_mod = sqrt(real \* real + img \* img);

this->m\_arg = atan2(img, real);

return \*this;

}

Complex operator-- (int) {

Complex copy(\*this);

double real = this->Re() - 1;

double img = this->Im();

this->m\_mod = sqrt(real \* real + img \* img);

this->m\_arg = atan2(img, real);

return copy;

}

Complex& operator+=(Complex a)

{

double real = Re();

double img = Im();

real += a.Re();

img += a.Im();

m\_mod = sqrt(real \* real + img \* img);

m\_arg = atan2(img, real);

return \*this;

}

Complex& operator-=(Complex a)

{

double real = Re();

double img = Im();

real -= a.Re();

img -= a.Im();

m\_mod = sqrt(real \* real + img \* img);

m\_arg = atan2(img, real);

return \*this;

}

Complex& operator\*=(Complex a)

{

m\_mod \*= a.m\_mod;

m\_arg += a.m\_arg;

return \*this;

}

Complex& operator/=(Complex a)

{

m\_mod /= a.m\_mod;

m\_arg -= a.m\_arg;

return \*this;

}

friend Complex operator+ (const Complex& left, const Complex& right);

friend Complex operator- (const Complex& left, const Complex& right);

friend Complex operator\* (const Complex& left, const Complex& right);

friend Complex operator/ (const Complex& left, const Complex& right);

friend Complex operator ""i(long double im);

friend Complex operator ""i(unsigned long long im);

friend std::ostream& operator<<(std::ostream& stream, const Complex& a);

friend Complex Poww(const Complex& r, double B);

};

export Complex operator+(const Complex& left, const Complex& right)

{

double real = left.Re() + right.Re();

double img = left.Im() + right.Im();

return Complex(real, img);

}

export Complex operator-(const Complex& left, const Complex& right)

{

double real = left.Re() - right.Re();

double img = left.Im() - right.Im();

return Complex(real, img);

}

export Complex operator\*(const Complex& left, const Complex& right)

{

double real = left.Mod() \* right.Mod();

double img = left.Arg() + right.Arg();

double real1 = real \* cos(img);

double img1 = real \* sin(img);

return Complex(real1, img1);

}

export Complex operator/(const Complex& left, const Complex& right)

{

double real = left.Mod() / right.Mod();

double img = left.Arg() - right.Arg();

double real1 = real \* cos(img);

double img1 = real \* sin(img);

return Complex(real1, img1);

}

export Complex operator ""i(unsigned long long im) {

return Complex(0.0, static\_cast<double>(im));

}

export Complex operator ""i(long double im) {

return Complex(0.0, static\_cast<double>(im));

}

export std::ostream& operator<<(std::ostream& stream, const Complex& a) {

if (a.Im() < 0) {

stream << a.Re() << " " << a.Im() << "i";

}

else {

stream << a.Re() << " + " << a.Im() << "i";

}

return stream;

}

export int FindGreatestCommonDivisor(int a, int b) {

if (a < 0) {

a \*= -1;

}

if (b < 0) {

b \*= -1;

}

while (a != 0 && b != 0) {

if (a > b) {

a %= b;

}

else {

b %= a;

}

}

return a + b;

}

export int FindLeastCommonMultiple(int a, int b) {

return abs(a \* b) / FindGreatestCommonDivisor(a, b);

}

export Complex Sin(const Complex& z) {

Complex a = 2 \* z;

return Complex(sin((double)a));

}

export class Rational {

private:

int m\_nominator;

int m\_denominator;

void normalizator() {

if (m\_denominator < 0) {

m\_denominator \*= -1;

m\_nominator \*= -1;

}

int nod = FindGreatestCommonDivisor(m\_nominator, m\_denominator);

m\_nominator /= nod;

m\_denominator /= nod;

}

public:

Rational() {

m\_nominator = 0;

m\_denominator = 1;

}

Rational(int x, int y) {

m\_nominator = x;

m\_denominator = y;

this->normalizator();

}

Rational(int x) {

m\_nominator = x;

m\_denominator = 1;

}

int Nominator() const {

return m\_nominator;

}

int Denominator() const {

return m\_denominator;

}

explicit operator double() const {

return static\_cast<double>(m\_nominator) / m\_denominator;

}

Rational operator-() {

Rational a(\*this);

a.m\_nominator \*= -1;

return a;

}

Rational& operator++ () {

Rational a(\*this);

m\_nominator += a.m\_denominator;

normalizator();

return \*this;

}

Rational operator++ (int) {

Rational a(\*this);

m\_nominator += a.m\_denominator;

normalizator();

return a;

}

Rational& operator-- () {

Rational a(\*this);

m\_nominator -= a.m\_denominator;

normalizator();

return \*this;

}

Rational operator-- (int) {

Rational a(\*this);

m\_nominator -= a.m\_denominator;

normalizator();

return a;

}

Rational& operator+=(Rational a) {

int Nominator = m\_nominator \* a.m\_denominator + m\_denominator \* a.m\_nominator;

int Denominator = m\_denominator \* a.m\_denominator;

m\_nominator = Nominator;

m\_denominator = Denominator;

normalizator();

return \*this;

}

Rational& operator-=(Rational a) {

int Nominator = m\_nominator \* a.m\_denominator - m\_denominator \* a.m\_nominator;

int Denominator = m\_denominator \* a.m\_denominator;

m\_nominator = Nominator;

m\_denominator = Denominator;

normalizator();

return \*this;

}

Rational& operator\*=(Rational a) {

m\_denominator \*= a.m\_denominator;

m\_nominator \*= a.m\_nominator;

normalizator();

return \*this;

}

Rational& operator/=(Rational a) {

m\_denominator \*= a.m\_nominator;

m\_nominator \*= a.m\_denominator;

normalizator();

return \*this;

}

friend Rational operator+ (const Rational& left, const Rational& right);

friend Rational operator- (const Rational& left, const Rational& right);

friend Rational operator\* (const Rational& left, const Rational& right);

friend Rational operator/(const Rational& left, const Rational& right);

friend bool operator==(const Rational& left, const Rational& right);

friend bool operator>(const Rational& left, const Rational& right);

friend bool operator<(const Rational& left, const Rational& right);

friend bool operator>=(const Rational& left, const Rational& right);

friend bool operator<=(const Rational& left, const Rational& right);

friend std::ostream& operator<< (std::ostream& stream, const Rational& rational);

friend Rational Poww(const Rational& r, double B);

};

export Rational operator+ (const Rational& left, const Rational& right) {

int denominator = FindLeastCommonMultiple(left.m\_denominator, right.m\_denominator);

int nominator = denominator / left.m\_denominator \* left.m\_nominator;

nominator += denominator / right.m\_denominator \* right.m\_nominator;

return Rational(nominator, denominator);

}

export Rational operator-(const Rational& left, const Rational& right)

{

int denominator = FindLeastCommonMultiple(left.m\_denominator, right.m\_denominator);

int nominator = denominator / left.m\_denominator \* left.m\_nominator;

nominator -= denominator / right.m\_denominator \* right.m\_nominator;

return Rational{ nominator, denominator };

}

export Rational operator\*(const Rational& left, const Rational& right)

{

return Rational(left.m\_nominator \* right.m\_nominator, right.m\_denominator \* left.m\_denominator);

}

export Rational operator/(const Rational& left, const Rational& right)

{

return Rational(left.m\_nominator \* right.m\_denominator, left.m\_denominator \* right.m\_nominator);

}

export bool operator==(const Rational& left, const Rational& right)

{

return left.m\_nominator == right.m\_nominator && left.m\_denominator == right.m\_denominator;

}

export bool operator>(const Rational& left, const Rational& right)

{

int lcm = FindLeastCommonMultiple(left.m\_denominator, right.m\_denominator);

return lcm / left.m\_denominator \* left.m\_nominator > lcm / right.m\_denominator \* right.m\_nominator;

}

export bool operator<(const Rational& left, const Rational& right)

{

int lcm = FindLeastCommonMultiple(left.m\_denominator, right.m\_denominator);

return lcm / left.m\_denominator \* left.m\_nominator < lcm / right.m\_denominator \* right.m\_nominator;

}

export bool operator>=(const Rational& left, const Rational& right)

{

int lcm = FindLeastCommonMultiple(left.m\_denominator, right.m\_denominator);

return lcm / left.m\_denominator \* left.m\_nominator >= lcm / right.m\_denominator \* right.m\_nominator;

}

export bool operator<=(const Rational& left, const Rational& right)

{

int lcm = FindLeastCommonMultiple(left.m\_denominator, right.m\_denominator);

return lcm / left.m\_denominator \* left.m\_nominator <= lcm / right.m\_denominator \* right.m\_nominator;

}

export std::ostream& operator<<(std::ostream& stream, const Rational& rational) {

stream << rational.m\_nominator << "/" << rational.m\_denominator;

return stream;

}

export Rational Poww(const Rational& r, double B) {

Rational res = r;

if (B == 0) {

return 1;

}

else {

if (B < 0) {

B = B \* (-1);

while (B - 1 > 0) {

res = res \* r;

B--;

}

res = 1 / res;

}

else {

while (B - 1 > 0) {

res = res \* r;

B--;

}

}

}

return res;

}

export Complex Poww(const Complex& r, double B) {

Complex res = r;

if (B == 0) {

return 1;

}

else {

if (B < 0) {

B = B \* (-1);

while (B - 1 > 0) {

res = res \* r;

B--;

}

res = 1 / res;

}

else {

while (B - 1 > 0) {

res = res \* r;

B--;

}

}

}

return res;

}

export Complex f(const Complex& c) {

Complex a = Complex(0.0, 1.0);

Complex result = Poww(c, 3) + (1 + 2 \* a) \* Poww(c, 2) + (1 - 2 \* a) \* Poww(c, -5 \* (double)a);

return result;

}

export Rational f(const Rational& r) {

Rational a(1, 2);

Rational result = Poww(r, 3) + (1 + 2 \* a) \* Poww(r, 2) + (1 - 2 \* a) \* Poww(r, -5 \* (double)a);

return result;

}

export double f(const double& b) {

double a = 0.5;

double result = pow(b, 3) + (1 + 2 \* a) \* pow(b, 2) + (1 - 2 \* a) \* pow(b, -5 \* a);

return result;

}

}

**Задание №2**

**Код Main.cpp:**

#include <iostream>

#include <complex>

#define \_SILENCE\_NONFLOATING\_COMPLEX\_DEPRECATION\_WARNING

import Math;

using namespace std;

int main()

{

cout << "Vvedite deistvitelnuy chast complexnogo chisla:";

double re;

cin >> re;

cout << "Vvedite mnimuy chast complexnogo chisla:";

double im;

cin >> im;

Math::Complex z(re, im);

cout << "Resultat funkcii na complexnih chislah:" << Math::f(z) << std::endl;

cout << "Vvedite chislitel racionalnogo chisla:";

double nom;

cin >> nom;

cout << "Vvedite znamenatel racionalnogo chisla:";

double denom;

cin >> denom;

Math::Rational r(nom, denom);

cout << "Resultat funkcii na racionalnih chislah:" << Math::f(r) << endl;

cout << "Enter the number: ";

double u;

cin >> u;

cout << "The result of function:" << Math::f(u) << endl;

return 0;

}

Реализация индивидуальных функций и вспомогательных для них функций:

export Rational Poww(const Rational& r, double B) {

Rational res = r;

if (B == 0) {

return 1;

}

else {

if (B < 0) {

B = B \* (-1);

while (B - 1 > 0) {

res = res \* r;

B--;

}

res = 1 / res;

}

else {

while (B - 1 > 0) {

res = res \* r;

B--;

}

}

}

return res;

}

export Complex Poww(const Complex& r, double B) {

Complex res = r;

if (B == 0) {

return 1;

}

else {

if (B < 0) {

B = B \* (-1);

while (B - 1 > 0) {

res = res \* r;

B--;

}

res = 1 / res;

}

else {

while (B - 1 > 0) {

res = res \* r;

B--;

}

}

}

return res;

}

export Complex f(const Complex& c) {

Complex a = Complex(0.0, 1.0);

Complex result = Poww(c, 3) + (1 + 2 \* a) \* Poww(c, 2) + (1 - 2 \* a) \* Poww(c, -5 \* (double)a);

return result;

}

export Rational f(const Rational& r) {

Rational a(1, 2);

Rational result = Poww(r, 3) + (1 + 2 \* a) \* Poww(r, 2) + (1 - 2 \* a) \* Poww(r, -5 \* (double)a);

return result;

}

export double f(const double& b) {

double a = 0.5;

double result = pow(b, 3) + (1 + 2 \* a) \* pow(b, 2) + (1 - 2 \* a) \* pow(b, -5 \* a);

return result;

}

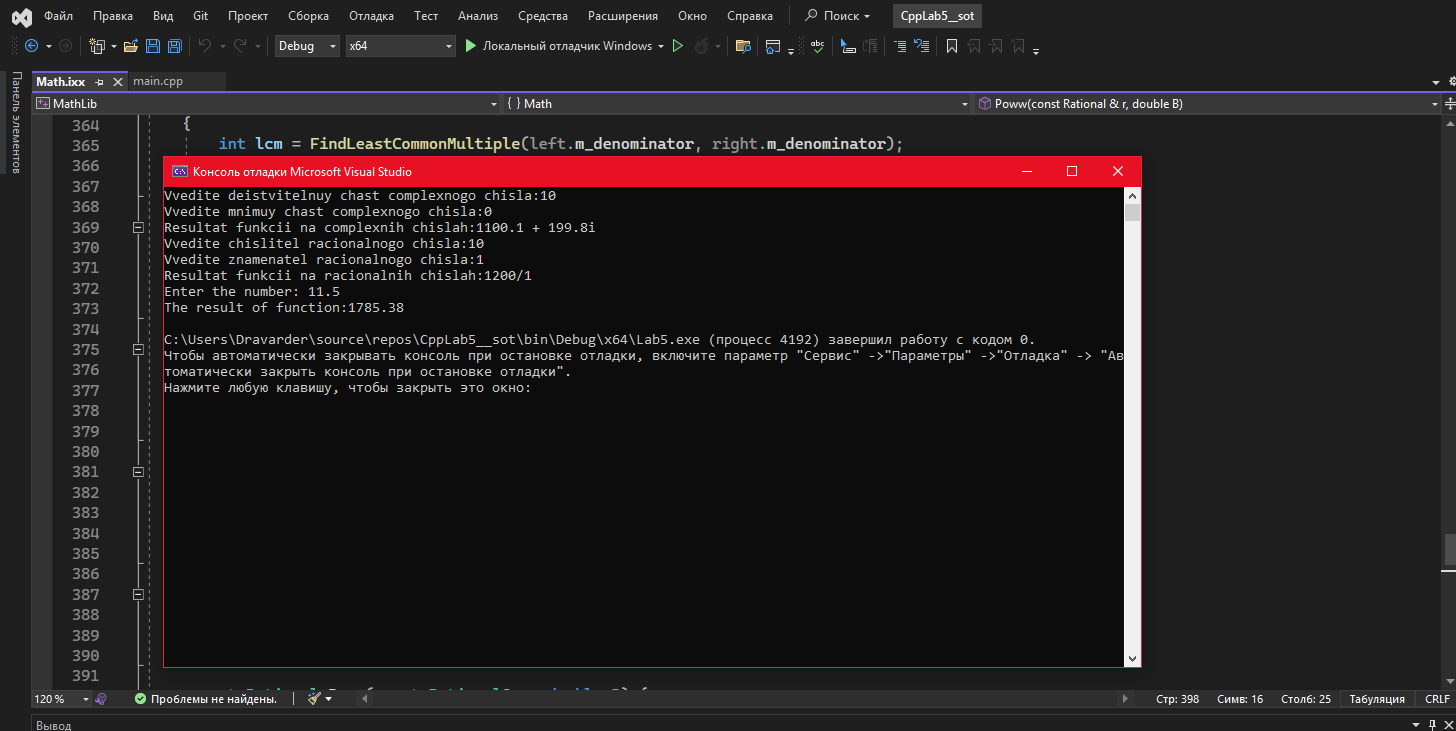


Рисунок 2. Результат работы программы

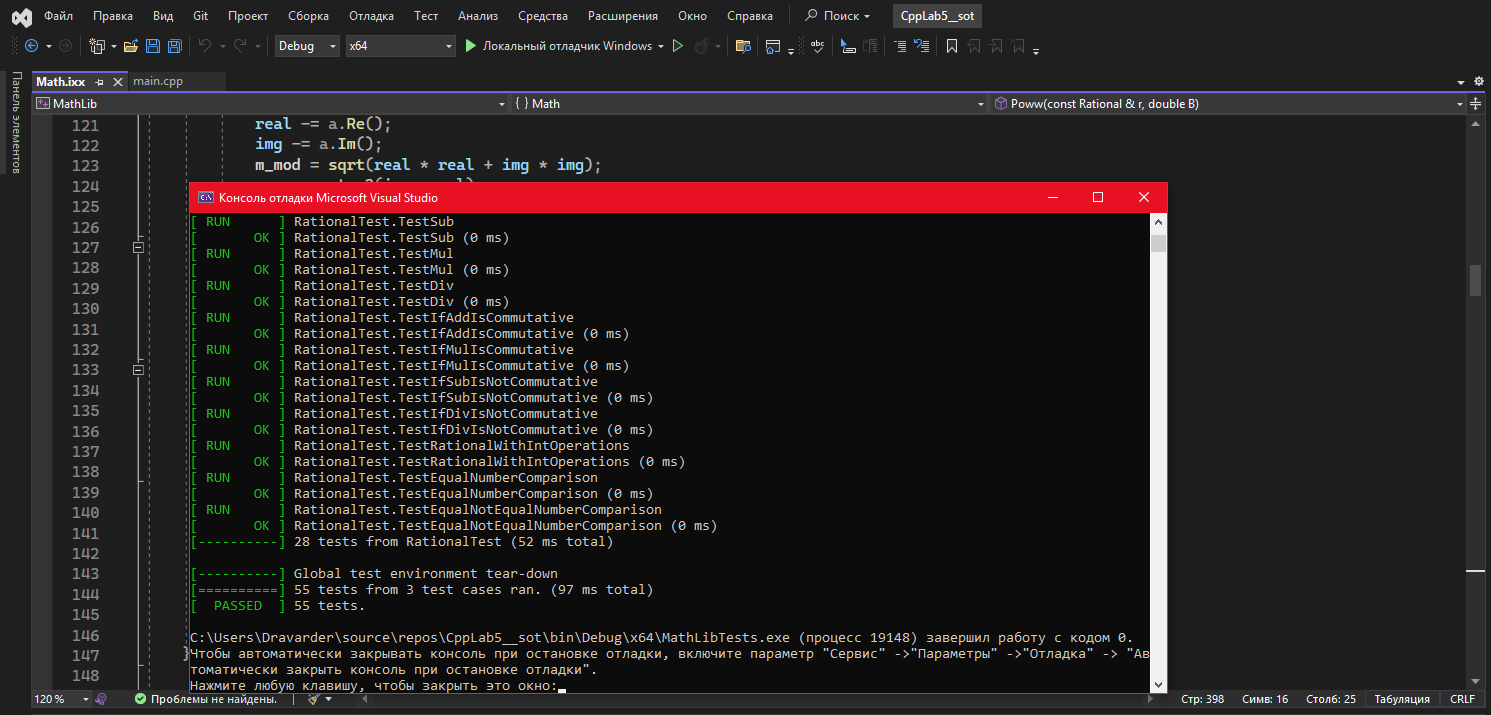


Рисунок 3. Результат прохождения тестов

**Вывод**

В процессе выполнения лабораторной работы, мне удалось освоить создание простых классов, конструкторов, перегрузку операций, создание экземпляров классов и передачу их в функции.

**Github:** https://github.com/Dravarder/CppLab5\_\_sot