**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ**

**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»

Практикум №5

«Понятие класса. Члены класса. Друзья класса. Перегрузка операций»

Вариант 2

Выполнила: студентка группы БАС2101

Блинкова А. А.

Проверил: Барков В.В.

Москва

2023

Содержание

[Цель работы. 3](#_Toc148097382)

[Задание 4](#_Toc148097383)

[Ход работы 1](#_Toc148097384)3

[Задание №1 1](#_Toc148097385)3

[Задание №2 2](#_Toc148097386)0

[Вывод 2](#_Toc148097391)2

# 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.

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 |
| Конструктор по умолчанию | Общедоступный | Инициализирует реальную и мнимую часть значениями 0 |
| Конструктор инициализации | Общедоступный | Инициализирует реальную и мнимую часть переданными параметрами |
| Конструктор преобразования | Общедоступный | Инициализирует реальную часть переданным значением, мнимую часть 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.

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.

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

**Индивидуальные варианты заданий**

Составить пользовательскую перегруженную функцию f, указанную в индивидуальном задании (и необходимые вспомогательные функции)

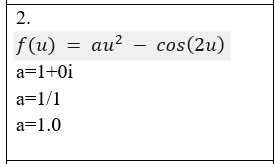
Прототипы функций:

*Complex f(const Complex &z);*

*Rational f(const Rational &r);*

*double f(double x);*

Индивидуальные варианты заданий:



# Ход работы

## Задание №1

Код для задания №1. Файл Math.ixx:

|  |
| --- |
| module;  #include <iostream>  export module Math;  export namespace Math  {  export class Complex  {  private:  double m\_re;  double m\_im;  public:  Complex() { m\_re = 0; m\_im = 0; }  Complex(double x, double y) { m\_re = x; m\_im = y; }  Complex(double x) { m\_re = x; m\_im = 0; }  static Complex FromExponentialForm(double x, double y)  {  double re = x \* cos(y);  double im = x \* sin(y);  Complex now(re, im);  return now;  }  static Complex FromAlgebraicForm(double re, double im)  {  Complex now(re, im);  return now;  }  double Re() const { return m\_re; }  double Im() const { return m\_im; }  double Mod() const { return sqrt(m\_re \* m\_re + m\_im \* m\_im); }  double Arg() const { return atan2(m\_im, m\_re); }  explicit operator double() const { return m\_re; }  Complex operator- () { return Complex(-m\_re, -m\_im); }  Complex& operator++ () { m\_re = m\_re + 1; return \*this; }  Complex operator++ (int x)  {  Complex star = \*this; m\_re = m\_re + 1; return star;  }  Complex& operator-- () { m\_re = m\_re - 1; return \*this; }  Complex operator-- (int x)  {  Complex star = \*this; this->m\_re = this->m\_re - 1; return star;  }  Complex& operator+= (Complex x) { m\_re = m\_re + x.m\_re; m\_im = m\_im + x.m\_im; return \*this; }  Complex& operator-= (Complex x) { m\_re = m\_re - x.m\_re; m\_im = m\_im - x.m\_im; return \*this; }  Complex& operator\*= (Complex x)  {  double real = m\_re;  double im = m\_im;  m\_re = real \* x.m\_re - im \* x.m\_im;  m\_im = real \* x.m\_im + im \* x.m\_re;  return \*this;  }  Complex& operator/= (Complex x)  {  double real1 = m\_re, real2 = x.m\_re;  double im1 = m\_im, im2 = x.m\_im;  m\_re = (real1 \* real2 + im1 \* im2) / (real2 \* real2 + im2 \* im2);  m\_im = (real2 \* im1 - real1 \* im2) / (real2 \* real2 + im2 \* im2);  return \*this;  }  friend Complex operator+(Complex lhs, Complex rhs);  friend Complex operator-(Complex lhs, Complex rhs);  friend Complex operator\*(Complex lhs, Complex rhs);  friend Complex operator/(Complex lhs, Complex rhs);  friend Complex operator""i(long double x);  friend Complex operator""i(unsigned long long x);  friend std::ostream& operator<< (std::ostream& os, const Complex complex);  };  export Complex operator+ (Complex lhs, Complex rhs)  {  double re = lhs.m\_re + rhs.m\_re;  double im = lhs.m\_im + rhs.m\_im;  return Complex(re, im);  }  export Complex operator- (Complex lhs, Complex rhs)  {  double re = lhs.m\_re - rhs.m\_re;  double im = lhs.m\_im - rhs.m\_im;  return Complex(re, im);  }  export Complex operator\*(Complex lhs, Complex rhs)  {  double re = lhs.m\_re \* rhs.m\_re - lhs.m\_im \* rhs.m\_im;  double im = lhs.m\_re \* rhs.m\_im + lhs.m\_im \* rhs.m\_re;  return Complex(re, im);  }  export Complex operator/ (Complex lhs, Complex rhs)  {  double re = (lhs.m\_re \* rhs.m\_re + lhs.m\_im \* rhs.m\_im) / (rhs.m\_re \* rhs.m\_re + rhs.m\_im \* rhs.m\_im);  double im = (lhs.m\_im \* rhs.m\_re - lhs.m\_re \* rhs.m\_im) / (rhs.m\_re \* rhs.m\_re + rhs.m\_im \* rhs.m\_im);  return Complex(re, im);  }  export Complex operator""i(long double x)  {  return Complex(0, static\_cast<double>(x));  }  export Complex operator""i(unsigned long long x)  {  return Complex(0, double(x));  }  export int FindGreatestCommonDivisor(int a, int b)  {  if (a < b)  {  int temp = a;  a = b;  b = temp;  }  int r;  while (b != 0)  {  r = a % b;  a = b;  b = r;  }  return abs(a);  }  export int FindLeastCommonMultiple(int x, int y)  {  return abs(x \* y) / FindGreatestCommonDivisor(x, y);  }  export std::ostream& operator<< (std::ostream& os, const Complex complex)  {  if (complex.m\_im < 0)  return os << complex.m\_re << complex.m\_im << "i";  else  return os << complex.m\_re << "+" << complex.m\_im << "i";  }  export class Rational  {  private:  int m\_nominator;  int m\_denominator;  public:  void socr()  {  int nod = FindGreatestCommonDivisor(m\_nominator, m\_denominator);  m\_nominator = m\_nominator / nod;  m\_denominator = m\_denominator / nod;  if (m\_nominator < 0 && m\_denominator < 0)  {  m\_nominator = -m\_nominator;  m\_denominator = -m\_denominator;  }  else  if (m\_nominator > 0 && m\_denominator < 0)  {  m\_nominator = -m\_nominator;  m\_denominator = -m\_denominator;  }  }  Rational() { m\_nominator = 0; m\_denominator = 1; }  Rational(int x, int y)  {  m\_nominator = x;  m\_denominator = y;  this->socr();  }  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 double(m\_nominator) / m\_denominator; }  Rational operator- () { return Rational(-this->m\_nominator, this->m\_denominator); }  Rational& operator++ () { m\_nominator += m\_denominator; socr(); return \*this; }  Rational operator++ (int x) { Rational star = \*this; (\*this).m\_nominator += m\_denominator; socr(); return star; }  Rational& operator-- () { m\_nominator -= m\_denominator; socr(); return \*this; }  Rational operator-- (int x) { Rational star = \*this; (\*this).m\_nominator -= m\_denominator; socr(); return star; }  Rational& operator+= (Rational x)  {  int no = m\_nominator \* x.m\_denominator + m\_denominator \* x.m\_nominator;  int de = m\_denominator \* x.m\_denominator;  m\_nominator = no;  m\_denominator = de;  socr();  return \*this;  }  Rational& operator-= (Rational x)  {  int no = m\_nominator \* x.m\_denominator - m\_denominator \* x.m\_nominator;  int de = m\_denominator \* x.m\_denominator;  m\_nominator = no;  m\_denominator = de;  socr();  return \*this;  }  Rational& operator\*= (Rational x)  {  int no = m\_nominator \* x.m\_nominator;  int de = m\_denominator \* x.m\_denominator;  m\_nominator = no;  m\_denominator = de;  socr();  return \*this;  }  Rational& operator/= (Rational x)  {  int no = m\_nominator \* x.m\_denominator;  int de = m\_denominator \* x.m\_nominator;  m\_nominator = no;  m\_denominator = de;  socr();  return \*this;  }  friend Rational operator+ (Rational lhs, Rational rhs);  friend Rational operator- (Rational lhs, Rational rhs);  friend Rational operator\* (Rational lhs, Rational rhs);  friend Rational operator/ (Rational lhs, Rational rhs);  friend bool operator== (Rational lhs, Rational rhs);  friend int operator<=> (Rational lhs, Rational rhs);  friend bool operator< (Rational lhs, Rational rhs);  friend bool operator<= (Rational lhs, Rational rhs);  friend bool operator> (Rational lhs, Rational rhs);  friend bool operator>= (Rational lhs, Rational rhs);  friend std::ostream& operator<< (std::ostream& os, const Rational& rational);  };  export Rational operator+ (Rational lhs, Rational rhs)  {  int no = lhs.m\_nominator \* rhs.m\_denominator + lhs.m\_denominator \* rhs.m\_nominator;  int de = lhs.m\_denominator \* rhs.m\_denominator;  Rational now = Rational(no, de);  now.socr();  return now;  }  export Rational operator- (Rational lhs, Rational rhs)  {  int no = lhs.m\_nominator \* rhs.m\_denominator - lhs.m\_denominator \* rhs.m\_nominator;  int de = lhs.m\_denominator \* rhs.m\_denominator;  Rational now = Rational(no, de);  now.socr();  return now;  }  export Rational operator\* (Rational lhs, Rational rhs)  {  int no = lhs.m\_nominator \* rhs.m\_nominator;  int de = lhs.m\_denominator \* rhs.m\_denominator;  Rational now = Rational(no, de);  now.socr();  return now;  }  export Rational operator/ (Rational lhs, Rational rhs)  {  int no = lhs.m\_nominator \* rhs.m\_denominator;  int de = lhs.m\_denominator \* rhs.m\_nominator;  Rational now = Rational(no, de);  now.socr();  return now;  }  export bool operator== (Rational lhs, Rational rhs)  {  if (lhs.m\_nominator == rhs.m\_nominator && lhs.m\_denominator == rhs.m\_denominator)  return true;  else  return false;  }  export int operator<=> (Rational lhs, Rational rhs)  {  if (lhs == rhs)  return 0;  Rational rez = lhs - rhs;  if (rez.Nominator() > 0)  return 1;  else  return -1;  }  export bool operator< (Rational lhs, Rational rhs)  {  if ((lhs <=> rhs) == -1)  return true;  else  return false;  }  export bool operator<= (Rational lhs, Rational rhs)  {  if (lhs <=> rhs == -1)  return true;  if (lhs <=> rhs == 0)  return true;  else  return false;  }  export bool operator> (Rational lhs, Rational rhs)  {  if (lhs <=> rhs == 1)  return true;  else  return false;  }  export bool operator>= (Rational lhs, Rational rhs)  {  {  if (lhs <=> rhs == 1)  return true;  if (lhs <=> rhs == 0)  return true;  else  return false;  }  }  export std::ostream& operator<< (std::ostream& os, const Rational& rational)  {  return os << rational.m\_nominator << "/" << rational.m\_denominator;  }  } |

## Задание №2

Код для задания №2. Файл main.cpp:

|  |
| --- |
| import Math;  #include<math.h>  #include<Windows.h>  #include<iostream>  using namespace Math;  Complex f(const Complex& z)  {  Complex itog;  Complex a = 1 + 0i;  Complex u = z;  itog = a \* pow(double(u), 2) - cos(double(2 \* u));  return itog;  }  Rational f(const Rational& r)  {  Rational itog;  Rational a = (1, 1);  Rational u = r;  itog = a \* pow(double(u), 2) - cos(double(2 \* u));  return itog;  }  double f(double x)  {  double itog;  double a = 1.0 ;  double u = x;  itog = a \* pow(u, 2) - cos(double(2 \* u));  return itog;  }  void main()  {  setlocale(LC\_ALL, "rus");  int m, s;  double a, b, x;  std::cout << "Введите реальную часть комплексного числа ";  std::cin >> a;  std::cout << "Введите мнимую часть комплексного числа ";  std::cin >> b;  std::cout << "Введите числитель дроби ";  std::cin >> s;  std::cout << "Введите знаменатель ";  std::cin >> m;  std::cout << "Введите вещёственное число ";  std::cin >> x;  Complex z = Complex(a, b);  Rational r = Rational(s, m);  std::cout << "Результат с комплексным числом: " << f(z) << std::endl;  std::cout << "Результат с рациональным числом: " << f(r) << std::endl;  std::cout << "Результат с вещественным числом: " << f(x) << std::endl;  } |

Работа программы представлена на рисунке ниже:

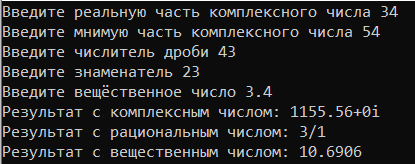


Рисунок 1 – Вывод задания

# Вывод

# Изучение создания классов, конструкторов, перегрузки операторов и работы с объектами классов в функциях дало нам ценные навыки, которые придают гибкость и структурированность нашему программированию. Это улучшает читаемость и чистоту кода, делает его более модульным и легким для будущего использования и расширения. Освоение этих концепций объектно-ориентированного программирования помогает решать задачи разной сложности более эффективно.