**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

Выполнили студенты группы: БКС2101

Кожемяка Е.В.

Руководитель: Барков В.В.

**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:

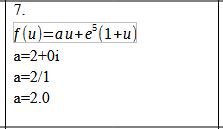
Complex f(const Complex &z);

Rational f(const Rational &r);

double f(double x);

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

**Индивидуальное задание (Вариант 7)**



**Ход работы**

**Задание №1**

Math.ixx:

|  |
| --- |
| module;  #include <math.h>  #include <ostream>  export module Math;  export class Complex  {  private:  double m\_re;  double m\_im;  public:  Complex(double real\_num)  {  m\_re = real\_num;  m\_im = 0;  }  Complex()  {  m\_re = 0;  m\_im = 0;  }  Complex(double real\_num, double img\_num)  {  m\_re = real\_num;  m\_im = img\_num;  }  static Complex FromExponentialForm(double Module, double Args)  {  Complex fef;  fef.m\_re = Module \* cos(Args);  fef.m\_im = Module \* sin(Args);  return fef;  }  static Complex FromAlgebraicForm(double real\_num, double img\_num)  {  Complex alg\_obj(real\_num, img\_num);  return alg\_obj;  }  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 (double)m\_re;  }  Complex operator-()  {  Complex object{\*this};  object.m\_im \*= -1;  object.m\_re \*= -1;  return object;  }  Complex& operator++()  {  m\_re++;  return \*this;  }  Complex operator++(int inc)  {  Complex object{\*this};  ++\*this;  return object;  }  Complex& operator--()  {  m\_re--;  return \*this;  }  Complex operator--(int dec)  {  Complex object{\*this};  --\*this;  return object;  }  Complex& operator+=(Complex var)  {  m\_re += var.m\_re;  m\_im += var.m\_im;  return \*this;  }  Complex& operator-=(Complex var) {  m\_re -= var.m\_re;  m\_im -= var.m\_im;  return \*this;  }  //DKA  Complex& operator\*=(Complex var) {  double realMI = m\_re;  double imagination = m\_im;  m\_re = realMI \* var.m\_re - imagination \* var.m\_im;  m\_im = realMI \* var.m\_im + imagination \* var.m\_re;  return \*this;  }  Complex& operator/=(Complex var) {  double tre1 = m\_re, tim1 = m\_im;  double tre2 = var.m\_re, tim2 = var.m\_im;  m\_re = (tre1 \* tre2 + tim1 \* tim2) / (pow(tre2, 2) + pow(tim2, 2));  m\_im = (tre2 \* tim1 - tre1 \* tim2) / (pow(tre2, 2) + pow(tim2, 2));  return \*this;  }  friend Complex operator+ (const Complex& one, const Complex& two);  friend Complex operator- (const Complex& one, const Complex& two);  friend Complex operator\* (const Complex& one, const Complex& two);  friend Complex operator/ (const Complex& one, const Complex& two);  friend Complex operator ""i(long double img\_num);  friend Complex operator ""i(unsigned long long img\_num);  friend std::ostream& operator<<(std::ostream& stream, const Complex& var);  };  export Complex operator+(const Complex& one, const Complex& two)  {  return Complex(one.m\_re + two.m\_re, one.m\_im + two.m\_im);  }  export Complex operator-(const Complex& one, const Complex& two)  {  return Complex(one.m\_re - two.m\_re, one.m\_im - two.m\_im);  }  export Complex operator\*(const Complex& one, const Complex& two)  {  return Complex((one.m\_re \* two.m\_re - one.m\_im \* two.m\_im),  (one.m\_re \* two.m\_im + one.m\_im \* two.m\_re));  }  export Complex operator/(const Complex& one, const Complex& two)  {  return Complex((one.m\_re \* two.m\_re + one.m\_im \* two.m\_im) /  (two.m\_re \* two.m\_re + two.m\_im \* two.m\_im),  (two.m\_re \* one.m\_im - one.m\_re \* two.m\_im) /  (two.m\_re \* two.m\_re + two.m\_im \* two.m\_im));  }  export Complex operator""i(long double img\_num)  {  return Complex(0.0, static\_cast<double>(img\_num));  }  export Complex operator""i(unsigned long long img\_num)  {  return Complex(0.0, static\_cast<double>(img\_num));  }  export std::ostream& operator<<(std::ostream& stream, const Complex& var)  {  if (var.m\_im < 0)  {  stream << var.m\_re << " " << var.m\_im << "i";  }  else  {  stream << var.m\_re << " + " << var.m\_im << "i";  }  return stream;  }  export int FindGreatestCommonDivisor(int a, int b)  {  int r;  if (a < 0)  a \*= -1;  if (b < 0)  b \*= -1;  while (true)  {  if (b == 0)  return a;  r = a % b;  a = b;  b = r;  }  }  export int FindLeastCommonMultiple(int a, int b) {  return abs(a \* b) / FindGreatestCommonDivisor(a, b);  }  export class Rational {  int m\_nominator;  int m\_denominator;    public:  void normalize()  {  int nod = FindGreatestCommonDivisor(m\_nominator, m\_denominator);  m\_nominator /= nod;  m\_denominator /= nod;  if (m\_denominator < 0) {  m\_denominator \*= -1;  m\_nominator \*= -1;  }  }  Rational()  {  m\_nominator = 0;  m\_denominator = 1;  }  Rational(int \_nominator, int \_denominator) {  m\_denominator = \_denominator;  m\_nominator = \_nominator;  normalize();  }  Rational(int \_nominator) {  m\_nominator = \_nominator;  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-() {  Rational object{\*this};  object.m\_nominator \*= -1;  return object;  }  Rational& operator++ () {  m\_nominator += m\_denominator;  return \*this;  }  Rational operator++ (int \_param) {  Rational object{\*this};  (\*this).m\_nominator += m\_denominator;  return object;  }  Rational& operator-- () {  m\_nominator -= m\_denominator;  return \*this;  }  Rational operator-- (int \_param) {  Rational object{\*this};  (\*this).m\_nominator -= m\_denominator;  return object;  }  Rational& operator+=(Rational var) {  int new\_den = FindLeastCommonMultiple(m\_denominator, var.m\_denominator);  m\_nominator = new\_den / m\_denominator \* m\_nominator;  m\_nominator += new\_den / var.m\_denominator \* var.m\_nominator;  m\_denominator = new\_den;  normalize();  return \*this;  }  Rational& operator-=(Rational var) {  int new\_d = FindGreatestCommonDivisor(m\_denominator, var.m\_denominator);  m\_nominator = new\_d / m\_denominator \* m\_nominator;  m\_nominator -= new\_d / var.m\_denominator \* var.m\_nominator;  m\_denominator = new\_d;  normalize();  return \*this;  }  Rational& operator\*=(Rational var) {  m\_denominator \*= var.m\_denominator;  m\_nominator \*= var.m\_nominator;  normalize();  return \*this;  }  Rational& operator/=(Rational var) {  m\_denominator \*= var.m\_nominator;  m\_nominator \*= var.m\_denominator;  normalize();  return \*this;  }  friend Rational operator+ (const Rational& one, const Rational& two);  friend Rational operator- (const Rational& one, const Rational& two);  friend Rational operator\* (const Rational& one, const Rational& two);  friend Rational operator/(const Rational& one, const Rational& two);  friend bool operator==(const Rational& one, const Rational& two);  friend bool operator>(const Rational& one, const Rational& two);  friend bool operator<(const Rational& one, const Rational& two);  friend bool operator>=(const Rational& one, const Rational& two);  friend bool operator<=(const Rational& one, const Rational& two);  friend std::ostream& operator<<(std::ostream& stream, const Rational& var);  };  export Rational operator+ (const Rational& one, const Rational& two) {  int denominator = FindLeastCommonMultiple(one.m\_denominator, two.m\_denominator);  int nominator = denominator / one.m\_denominator \* one.m\_nominator;  nominator += denominator / two.m\_denominator \* two.m\_nominator;  return Rational{ nominator, denominator };  }  export Rational operator-(const Rational& one, const Rational& two)  {  int denominator = FindLeastCommonMultiple(one.m\_denominator, two.m\_denominator);  int nominator = denominator / one.m\_denominator \* one.m\_nominator;  nominator -= denominator / two.m\_denominator \* two.m\_nominator;  return Rational{ nominator, denominator };  }  export Rational operator\*(const Rational& one, const Rational& two)  {  return Rational{ one.m\_nominator \* two.m\_nominator, two.m\_denominator \* one.m\_denominator };  }  export Rational operator/(const Rational& one, const Rational& two)  {  return Rational{ one.m\_nominator \* two.m\_denominator,one.m\_denominator \* two.m\_nominator };  }  export bool operator==(const Rational& one, const Rational& two)  {  return one.m\_nominator == two.m\_nominator && one.m\_denominator == two.m\_denominator;  }  export bool operator>(const Rational& one, const Rational& two)  {  int den = FindLeastCommonMultiple(one.m\_denominator, two.m\_denominator);  return den / one.m\_denominator \* one.m\_nominator > den / two.m\_denominator \* two.m\_nominator;  }  export bool operator<(const Rational& one, const Rational& two)  {  int den = FindLeastCommonMultiple(one.m\_denominator, two.m\_denominator);  return den / one.m\_denominator \* one.m\_nominator < den / two.m\_denominator \* two.m\_nominator;  }  export bool operator>=(const Rational& one, const Rational& two)  {  int den = FindLeastCommonMultiple(one.m\_denominator, two.m\_denominator);  return den / one.m\_denominator \* one.m\_nominator >= den / two.m\_denominator \* two.m\_nominator;  }  export bool operator<=(const Rational& one, const Rational& two)  {  int den = FindLeastCommonMultiple(one.m\_denominator, two.m\_denominator);  return den / one.m\_denominator \* one.m\_nominator <= den / two.m\_denominator \* two.m\_nominator;  }  export std::ostream& operator<<(std::ostream& stream, const Rational& var) {  stream << var.m\_nominator << "|" << var.m\_denominator;  return stream;  }  export Complex f(const Complex& z)  {  Complex a(2, 0);  Complex result = a \* z + std::exp(5) \* (1 + z);  return result;  }  export Rational f(const Rational& r)  {  Rational a(2, 1);  Rational result = a \* r + std::exp(5) \* (1 + r);  return result;  }  export double f(const double& d)  {  double a = 2.0;  int result = a \* d + std::exp(5) \* (1 + d);  return result;  } |

**Задание №2**

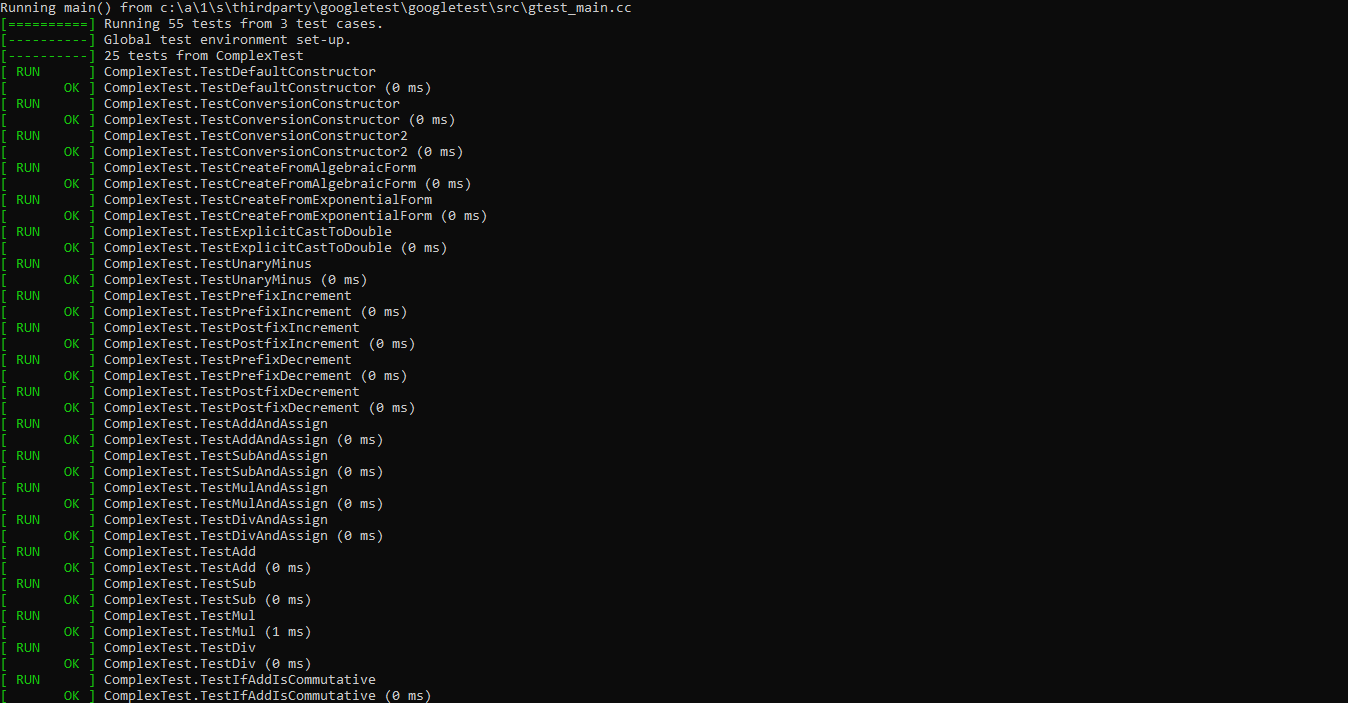
Разработайте для объектов вашего класса предложенные в каждом варианте пользовательские функции.

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

main.cpp:

|  |
| --- |
| #include <iostream>  import Math;  using namespace std;  int main()  {  double re, im;  cout << "Input Real and Imag num:" << endl;  cin >> re >> im;  Complex c\_num(re, im);  double chisl, znam;  cout << "Input Chisl and Znam num:" << endl;  cin >> chisl >> znam;  Rational r\_num(chisl, znam);  double d\_num;  cout << "Input double:" << endl;  cin >> d\_num;  Rational answ = f(r\_num);  double a = answ.Nominator(), b = answ.Denominator();  cout << "f(" << c\_num << ") = " << f(c\_num) << endl;  cout << "f(" << r\_num << ") = " << f(r\_num) << " = " << a/b << endl;  cout << "f(" << d\_num << ") = " << f(d\_num) << endl;  return 0;  } |

**Результат работы программы**

рисунок 1.1.(Результат выполнения автотестов)

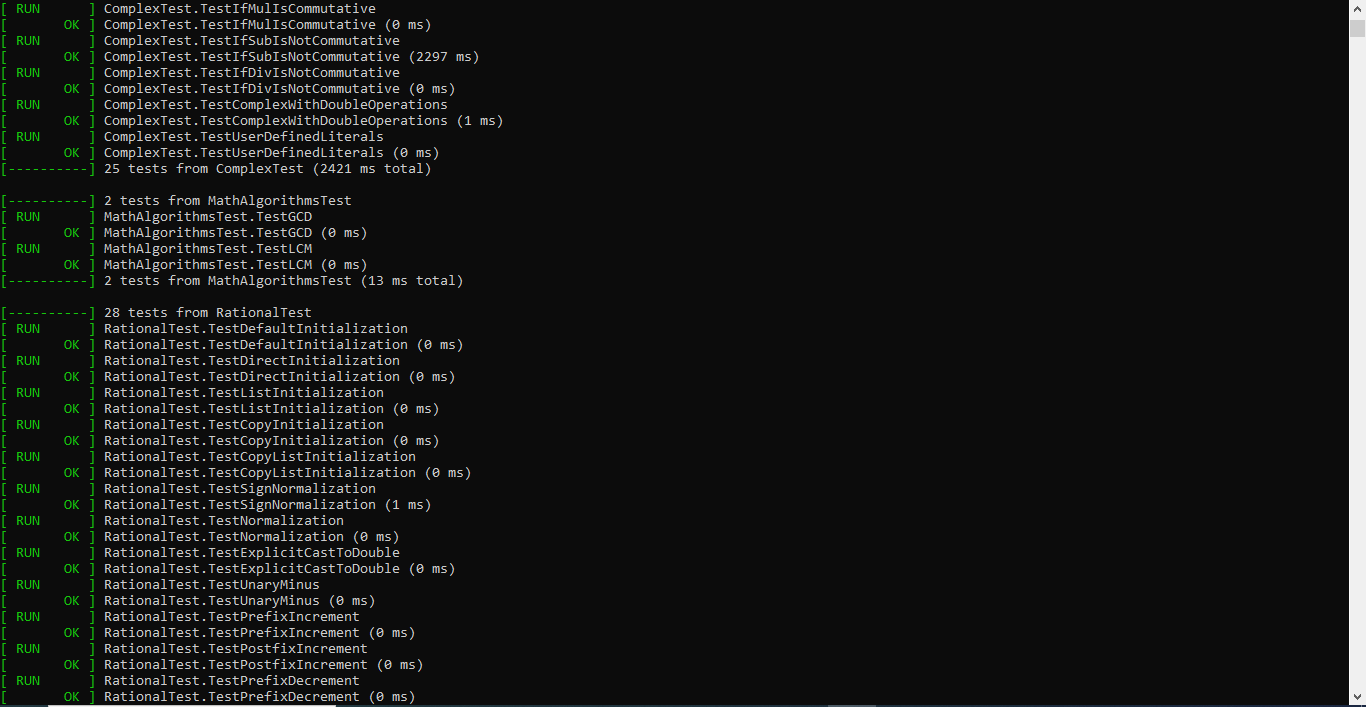
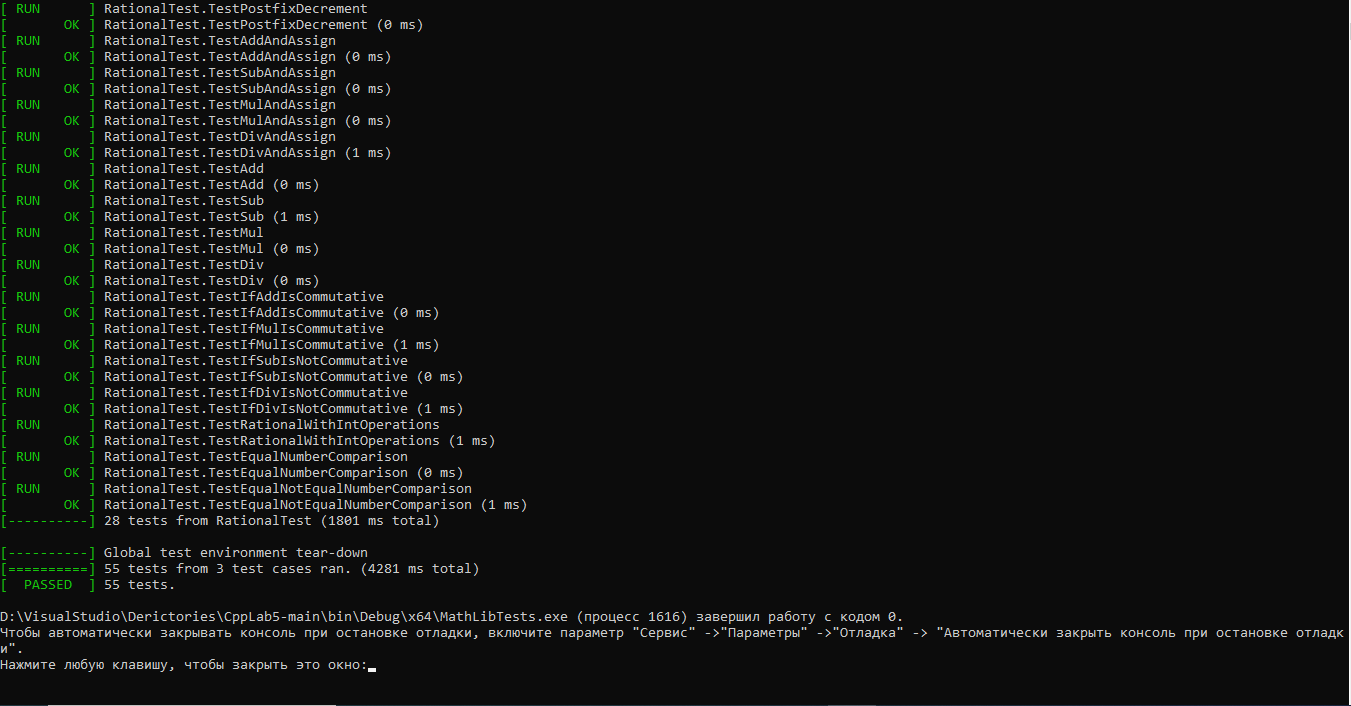
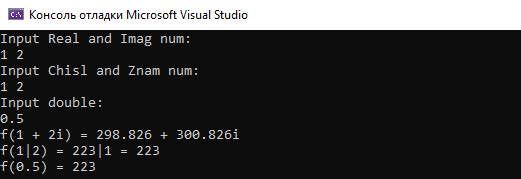


рисунок 1.2.(Результат выполнения автотестов)

рисунок 1.3.(Результат выполнения автотестов)

рисунок 2.(Результат выполнения файла main)

**Вывод**

В ходе выполнения лабораторной работы, я научился создавать простые классы, создавать конструкторы, перегружать операции, создавать объекты классов и передавать их в функции.