МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра: Программной инженерии**

Направление подготовки: «Программная инженерия»

«Визуальное программирование»

**Отчёт по лабораторной работе 3**

|  |  |
| --- | --- |
|  | Выполнил:  студент группы 381908-3  Гудков Андрей Вячеславович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  Проверил:  Карчков Д.А  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Нижний Новгород  
2020 г.

Содержание

[Описание задачи 3](#_Toc35011578)

[Код main.cpp 4](#_Toc35011579)

[Код fraction.h 6](#_Toc35011580)

[Код fraction.cpp 7](#_Toc35011581)

[Код Complex.h 11](#_Toc35011582)

[Код Complex.cpp 12](#_Toc35011583)

# Описание задачи

Требуется перегрузить 25 операторов используя предыдущую лабораторную работу с классом комплексного числа и классом дроби.

**Комплексные числа -** это выражение вида *a* + *bi*, где *a*, *b* — действительные числа, а *i* — так называемая *мнимая единица*, символ, квадрат которого равен –1, то есть *i*2 = –1. Число *a* называется *действительной частью*, а число *b* — *мнимой частью* комплексного числа *z* = *a* + *bi*. Если *b* = 0, то вместо *a* + 0*i* пишут просто *a*. Видно, что действительные числа — это частный случай комплексных чисел.

**Конструктор -** функция, предназначенная для инициализации объектов класса или иначе - специальный блок инструкций, вызываемый при создании объекта.

**Деструктор -** cпециальный метод [класса](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), служащий для [деинициализации](https://ru.wikipedia.org/w/index.php?title=%D0%94%D0%B5%D0%B8%D0%BD%D0%B8%D1%86%D0%B8%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F&action=edit&redlink=1" \o "Деинициализация (страница отсутствует)) [объекта](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) (например освобождения [памяти](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C)).

**Оператор присваивания -** К первому комплексному числу присваиваю первую дробь второго комплексного числа, потом присваиваю вторую дробь и вывожу первое комплексное число.

**Оператор умножения -** Присваиваю к 1 и 2 дроби произведение действительной части по формул. Затем нахожу сумму действительной части, сокращая дробь, если возможно, по нахождению наибольшего общего делителя, вместе с полученным результатом. Вывожу знак «+».Присваиваю к 1 и 2 дроби произведение мнимой части по формуле и выполняю те же самые действия уже для мнимой части. В конце вывожу «i».

**Оператор суммирования -** Присваиваю к 1 и 2 дроби значения действительных частей. Нахожу их сумму, если возможно, то сокращаю через НОД. Вывожу знак «+».Присваиваю к 1 и 2 дроби произведение мнимой части и выполняю те же самые действия уже для мнимой части. В конце вывожу «i».

**Оператор разности -** Присваиваю к 1 и 2 дроби значения действительных частей. Нахожу их разность, если возможно, то сокращаю через НОД. Вывожу знак «+».Присваиваю к 1 и 2 дроби произведение мнимой части и выполняю те же самые действия уже для мнимой части. В конце вывожу «i».

**Оператор деления -**  Присваиваю к 1 и 2 дроби произведение действительной части по формул. Затем нахожу сумму действительной части, сокращая дробь, если возможно, по нахождению наибольшего общего делителя, вместе с полученным результатом. Получившее значение делю на сумму квадратов. Вывожу знак «+».Присваиваю к 1 и 2 дроби произведение мнимой части по формуле и выполняю те же самые действия уже для мнимой части. В конце вывожу «i».

**Оператор суммы комплекса с числом -** Присваиваю к 1 дроби значение действительной части. Затем нахожу сумму дроби и числа, сокращая дробь, если возможно, по НОД. Вывожу знак «+» и вывожу мнимую часть. В конце вывожу «i».

**Оператор разности комплекса с числом -** Присваиваю к 1 дроби значение действительной части. Затем нахожу разность дроби и числа, сокращая дробь, если возможно, по НОД. Вывожу знак «+» и вывожу мнимую часть. В конце вывожу «i».

**Оператор произведения комплекса с числом -** Присваиваю к 1 дроби значение действительной части. Затем нахожу произведение дроби и числа, сокращая дробь, если возможно, по НОД. Вывожу знак «+» и вывожу мнимую часть. В конце вывожу «i».

**Оператор деления комплекса с числом -** Присваиваю к 1 дроби значение действительной части. Затем нахожу частное дроби и числа, сокращая дробь, если возможно, по НОД. Вывожу знак «+» и вывожу мнимую часть. В конце вывожу «i».

**Оператор сравнения больше** **-** Задаю булевою переменную. Присваиваю к переменным произведения числителя и знаменателя первой дроби комплекса и первой дроби 2 комплекса. Потом сравниваю и вывожу булевою переменную.

**Оператор сравнения меньше -** Задаю булевою переменную. Присваиваю к переменным произведения числителя и знаменателя первой дроби комплекса и первой дроби 2 комплекса. Потом сравниваю и вывожу булевою переменную.

**Оператор эквивалентности -** Задаю булевою переменную. Присваиваю к переменным произведения числителя и знаменателя первой дроби комплекса и первой дроби 2 комплекса. Потом присваиваю значение логического уравнения и вывожу булевою переменную.

**Оператор не эквивалентности -** Задаю булевою переменную. Присваиваю к переменным произведения числителя и знаменателя первой дроби комплекса и первой дроби 2 комплекса. Потом присваиваю значение логического уравнения и вывожу булевою переменную.

**Оператор сравнения больше или равно -** Задаю булевою переменную. Присваиваю к переменным произведения числителя и знаменателя первой дроби комплекса и первой дроби 2 комплекса. Потом присваиваю значение логического уравнения и вывожу булевою переменную.

**Оператор сравнения меньше или равно -** Задаю булевою переменную. Присваиваю к переменным произведения числителя и знаменателя первой дроби комплекса и первой дроби 2 комплекса. Потом присваиваю значение логического уравнения и вывожу булевою переменную.

**Оператор унарный плюс -**  Присваиваю к 1 и 2 дроби значения действительных частей. Нахожу их сумму, если возможно, то сокращаю через НОД. Вывожу знак «+».Присваиваю к 1 и 2 дроби произведение мнимой части и выполняю те же самые действия уже для мнимой части. В конце вывожу «i».

**Оператор унарный минус -**  Присваиваю к 1 и 2 дроби значения действительных частей. Нахожу их разность, если возможно, то сокращаю через НОД. Вывожу знак «+».Присваиваю к 1 и 2 дроби произведение мнимой части и выполняю те же самые действия уже для мнимой части. В конце вывожу «i».

**Оператор унарное умножение -** Присваиваю к 1 и 2 дроби произведение действительной части по формул. Затем нахожу сумму действительной части, сокращая дробь, если возможно, по нахождению наибольшего общего делителя, вместе с полученным результатом. Вывожу знак «+».Присваиваю к 1 и 2 дроби произведение мнимой части по формуле и выполняю те же самые действия уже для мнимой части. В конце вывожу «i».

**Оператор унарное деление -**  Присваиваю к 1 и 2 дроби произведение действительной части по формул. Затем нахожу сумму действительной части, сокращая дробь, если возможно, по нахождению наибольшего общего делителя, вместе с полученным результатом. Получившее значение делю на сумму квадратов. Вывожу знак «+».Присваиваю к 1 и 2 дроби произведение мнимой части по формуле и выполняю те же самые действия уже для мнимой части. В конце вывожу «i».

**Оператор вывода -** Присваиваю значения дробей действительной и мнимой комплексного числа, затем присваиваю значения к переменной и вывожу её.

**Оператор ввода -**  С начала перегружаю оператор ввода во fraction, присваивая значения числителя и знаменателя в объект класса и вывожу его, затем в complex передаю значения для первой дроби и второй дроби.

# Тестирование:

# Код main.cpp

#include "locale"

#include "Complex.h"

int main()

{

setlocale(LC\_ALL, "Russian");

int a;

cout << "Выберите:1.Обычные или 2.Комплексные дроби";

do {

cin >> a;

} while (a<1 || a>2);

if (a == 1) {

fraction test(0, 1);

fraction test1,rex;

int c, d;

cout << "Введите числитель" << endl;

cin >> c;

cout << "Введите знаменатель" << endl;

do

{

cin >> d;

if (d == 0)

cout << "Нельзя так" << endl;

} while (d == 0);

test.SetChisl(c);

test.SetZnam(d);

cout << "Введите числитель" << endl;

cin >> c;

cout << "Введите знаменатель" << endl;

do

{

cin >> d;

if (d == 0)

cout << "Нельзя так" << endl;

} while (d == 0);

test1.SetChisl(c);

test1.SetZnam(d);

test.Umn(test, test1);

cout << "перегрузка умножения" << endl;

rex = test \* test1;

rex.Out();

cout << endl;

cout << "перегрузка деления" << endl;

rex = test / test1;

rex.Out();

cout << endl;

cout << "перегрузка вычитания" << endl;

rex = test - test1;

rex.Out();

cout << endl;

cout << "перегрузка сложения" << endl;

rex = test + test1;

rex.Out();

cout << endl;

cout << endl;

test.Del(test, test1);

cout << endl;

cout << endl;

test.Sum(test, test1);

cout << endl;

cout << endl;

test.Vuch(test, test1);

}

else if (a == 2) {

Complex fr1,fr2,res;

cin >> fr1;

cout << fr1<<endl;

fr2 = fr1;

cout << fr2;

res = fr1 \* fr2;

cout << "Произвошло умножение " << res << endl;

res = fr1 \* 2;

cout << endl << res << endl;

}

}

# 

# Код fraction.h

#include <iostream>

using namespace std;

class fraction

{

private:

int chisl;

int znam;

public:

void SetChisl(int n);

void SetZnam(int n);

int GetChisl();

int GetZnam();

int Umn(fraction drob1, fraction drob2);

fraction operator\*(fraction& tmp);

int Del(fraction drob1, fraction drob2);

fraction operator/(fraction& tmp);

int Sum(fraction drob1, fraction drob2);

fraction operator-(fraction& tmp);

int Vuch(fraction drob1, fraction drob2);

fraction operator+(fraction& tmp);

friend istream& operator>>(istream& in, fraction& tmp);

void Out();

fraction(int i, int j);

fraction();

~fraction();

};

# 

# Код fraction.cpp

#include "Frac.h"

int found0(int c) {

if (c == 0) {

return 0;

}

else return 1;

}

void fraction::SetChisl(int n)

{

this->chisl = n;

}

void fraction::SetZnam(int n)

{

this->znam = n;

}

fraction::fraction(int i, int j)

{

this->chisl = i;

this->znam = j;

}

fraction::fraction()

{

this->chisl = 0;

this->znam = 1;

}

fraction::~fraction()

{

this->chisl = 0;

this->znam = 1;

}

int fraction::GetChisl()

{

return this->chisl;

}

int fraction::GetZnam()

{

return this->znam;

}

int fraction::Umn(fraction drob1, fraction drob2)

{

int c;

int d;

c = drob1.GetChisl() \* drob2.GetChisl();

if (found0(c) == 0)

{

cout << "0";

return 0;

}

else

{

d = drob1.GetZnam() \* drob2.GetZnam();

cout << c << '/' << d;

}

return 0;

}

fraction fraction::operator\*(fraction& tmp)

{

fraction res;

int c;

int d;

c = this->GetChisl() \* tmp.GetChisl();

if (found0(c) == 0)

{

cout << "0";

return res;

}

else

{

d = this->GetZnam() \* tmp.GetZnam();

}

res.SetChisl(c);

res.SetZnam(d);

return res;

}

int fraction::Del(fraction drob1, fraction drob2)

{

int c;

int d;

c = drob1.GetChisl() \* drob2.GetZnam();

if (found0(c) == 0)

{

cout << "0";

return 0;

}

else

{

d = drob1.GetZnam() \* drob2.GetChisl();

cout << c << '/' << d;

}

return 0;

}

fraction fraction::operator/(fraction& tmp)

{

int c;

int d;

fraction res;

c = this->GetChisl() \* tmp.GetZnam();

if (found0(c) == 0)

{

cout << "0";

return res;

}

else

{

d = this->GetZnam() \* tmp.GetChisl();

}

res.SetChisl(c);

res.SetZnam(d);

return res;

}

int fraction::Sum(fraction drob1, fraction drob2)

{

int c;

int d;

c = drob1.GetChisl() \* drob2.GetZnam() + drob1.GetZnam() \* drob2.GetChisl();

if (found0(c) == 0)

{

cout << "0";

return 0;

}

else

{

d = drob1.GetZnam() \* drob2.GetZnam();

cout << c << '/' << d;

}

return 0;

}

fraction fraction::operator-(fraction& tmp)

{

int c;

int d;

fraction res;

c = this->GetChisl() \* tmp.GetZnam() - this->GetZnam() \* tmp.GetChisl();

if (found0(c) == 0)

{

cout << "0";

return res;

}

else

{

d = this->GetZnam() \* tmp.GetZnam();

}

res.SetChisl(c);

res.SetZnam(d);

return res;

}

int fraction::Vuch(fraction drob1, fraction drob2)

{

int c;

int d;

c = drob1.GetChisl() \* drob2.GetZnam() - drob1.GetZnam() \* drob2.GetChisl();

if (found0(c) == 0)

{

cout << "0";

return 0;

}

else

{

d = drob1.GetZnam() \* drob2.GetZnam();

cout << c << '/' << d;

}

return 0;

}

fraction fraction::operator+(fraction& tmp)

{

int c;

int d;

fraction res;

c = this->GetChisl() \* tmp.GetZnam() + this->GetZnam() \* tmp.GetChisl();

if (found0(c) == 0)

{

cout << "0";

return res;

}

else

{

d = this->GetZnam() \* tmp.GetZnam();

}

res.SetChisl(c);

res.SetZnam(d);

return res;

}

void fraction::Out()

{

cout << this->GetChisl() << "/" << this->GetZnam()<<endl;

}

istream& operator>>(istream& in, fraction& tmp)

{

int c, z;

cout << "Введите дробь" << endl;

cout << "Числитель" << endl;

in >> c;

tmp.SetChisl(c);

cout << "Знаменатель" << endl;

in >> z;

tmp.SetZnam(z);

return in;

# }

# Код Complex.h

#include "Frac.h"

class Complex {

private:

fraction Re;

fraction Im;

public:

Complex();

~Complex();

void scan();

friend ostream& operator<<(ostream& out, Complex& tmp);

friend istream& operator>>(istream& in, Complex& tmp);

Complex& operator=(const Complex& tmp);

Complex operator\*(Complex& tmp);

Complex operator+(Complex& tmp);

Complex operator-(Complex& tmp);

Complex operator/(Complex& tmp);

Complex operator\*(int tmp);

Complex operator/(int tmp);

Complex operator+(int tmp);

Complex operator-(int tmp);

bool operator>(Complex& tmp);

bool operator<(Complex& tmp);

bool operator==(Complex& tmp);

bool operator!=(Complex& tmp);

bool operator>=(Complex& tmp);

bool operator<=(Complex& tmp);

Complex& operator+=(Complex& tmp);

Complex& operator-=(Complex& tmp);

Complex& operator/=(Complex& tmp);

Complex& operator\*=(Complex& tmp);

Complex& operator+=(int tmp);

Complex& operator-=(int tmp);

Complex& operator/=(int tmp);

Complex& operator\*=(int tmp);

};

# Код Complex.cpp

#include "Complex.h"

Complex::Complex()

{

this->Re = fraction(0, 1);

this->Im = fraction(0, 1);

}

Complex::~Complex()

{

this->Re = fraction(0, 1);

this->Im = fraction(0, 1);

}

void Complex::scan()

{

int c, d;

cout << "Введите первую дробь" << endl;

cout << "Введите числитель" << endl;

cin >> c;

cout << "Введите знаменатель" << endl;

do

{

cin >> d;

if (d == 0)

cout << "Нельзя так" << endl;

} while (d == 0);

this->Re = fraction(c,d);

cout << "Введите Вторую дробь" << endl;

cout << "Введите числитель" << endl;

cin >> c;

cout << "Введите знаменатель" << endl;

do

{

cin >> d;

if (d == 0)

cout << "Нельзя так" << endl;

} while (d == 0);

this->Im = fraction(c, d);

}

Complex& Complex::operator=(const Complex& tmp)

{

this->Re = tmp.Re;

this->Im= tmp.Im;

return \*this;

}

Complex Complex::operator\*(Complex& tmp)

{

Complex res;

fraction newRe, newIm, newRe1, newIm1;

newRe = this->Re \* tmp.Re;

newIm = this->Im \* tmp.Im;

newRe1 = this->Re \* tmp.Im;

newIm1 = this->Im \* tmp.Re;

res.Re = newRe - newIm;

res.Im = newRe1 + newIm1;

return res;

}

Complex Complex::operator+(Complex& tmp)

{

Complex res;

fraction newRe, newIm;

newRe = this->Re + tmp.Re;

newIm = this->Im + tmp.Im;

res.Re = newRe;

res.Im = newIm;

return res;

}

Complex Complex::operator-(Complex& tmp)

{

Complex res;

fraction newRe, newIm;

newRe = this->Re - tmp.Re;

newIm = this->Im - tmp.Im;

res.Re = newRe;

res.Im = newIm;

return res;

}

Complex Complex::operator/(Complex& tmp)

{

Complex res;

fraction newRe, newIm, newRe1, newIm1, newRe2, newIm2, newa, newb, newc;

newRe = this->Re \* tmp.Re;

newIm = this->Im \* tmp.Im;

newRe1 = tmp.Re \* tmp.Re;

newIm1 = tmp.Im \* tmp.Im;

newRe2 = this->Re \* tmp.Im;

newIm2 = this->Im \* tmp.Re;

newa = newRe + newIm;

newb = newRe1 + newIm1;

newc = newIm2 - newRe2;

res.Re = newa / newb;

res.Im = newc / newb;

return res;

}

Complex Complex::operator\*(int tmp)

{

Complex a;

a = \*this;

a.Re.SetChisl(this->Re.GetChisl() \* tmp);

return a;

}

Complex Complex::operator/(int tmp)

{

Complex a;

a = \*this;

a.Re.SetChisl(this->Re.GetChisl() / tmp);

return a;

}

Complex Complex::operator+(int tmp)

{

Complex a;

a = \*this;

a.Re.SetChisl(this->Re.GetChisl() + tmp\*this->Re.GetZnam());

return a;

}

Complex Complex::operator-(int tmp)

{

Complex a;

a = \*this;

a.Re.SetChisl(this->Re.GetChisl() - tmp \* this->Re.GetZnam());

return a;

}

bool Complex::operator>(Complex& tmp)

{

int n,m;

n = this->Re.GetChisl() \* tmp.Re.GetZnam();

m = this->Re.GetZnam() \* tmp.Re.GetChisl();

if (n > m)

return true;

else

return false;

}

bool Complex::operator<(Complex& tmp)

{

int n, m;

n = this->Re.GetChisl() \* tmp.Re.GetZnam();

m = this->Re.GetZnam() \* tmp.Re.GetChisl();

if (n < m)

return true;

else

return false;

}

bool Complex::operator==(Complex& tmp)

{

int n, m;

n = this->Re.GetChisl() \* tmp.Re.GetZnam();

m = this->Re.GetZnam() \* tmp.Re.GetChisl();

if (n == m)

return true;

else

return false;

}

bool Complex::operator!=(Complex& tmp)

{

int n, m;

n = this->Re.GetChisl() \* tmp.Re.GetZnam();

m = this->Re.GetZnam() \* tmp.Re.GetChisl();

if (n != m)

return true;

else

return false;

}

bool Complex::operator>=(Complex& tmp)

{

int n, m;

n = this->Re.GetChisl() \* tmp.Re.GetZnam();

m = this->Re.GetZnam() \* tmp.Re.GetChisl();

if (n >= m)

return true;

else

return false;

}

bool Complex::operator<=(Complex& tmp)

{

int n, m;

n = this->Re.GetChisl() \* tmp.Re.GetZnam();

m = this->Re.GetZnam() \* tmp.Re.GetChisl();

if (n <= m)

return true;

else

return false;

}

Complex& Complex::operator+=(Complex& tmp)

{

fraction newRe, newIm;

newRe = this->Re + tmp.Re;

newIm = this->Im + tmp.Im;

this->Re = newRe;

this->Im = newIm;

return \*this;

}

Complex& Complex::operator-=(Complex& tmp)

{

fraction newRe, newIm;

newRe = this->Re - tmp.Re;

newIm = this->Im - tmp.Im;

this->Re = newRe;

this->Im = newIm;

return \*this;

}

Complex& Complex::operator/=(Complex& tmp)

{

\*this = \*this / tmp;

return \*this;

}

Complex& Complex::operator\*=(Complex& tmp)

{

\*this = \*this \* tmp;

return \*this;

}

Complex& Complex::operator+=(int tmp)

{

\*this = \*this + tmp;

return \*this;

}

Complex& Complex::operator-=(int tmp)

{

\*this = \*this - tmp;

return \*this;

}

Complex& Complex::operator/=(int tmp)

{

\*this = \*this / tmp;

return \*this;

}

Complex& Complex::operator\*=(int tmp)

{

\*this = \*this \* tmp;

return \*this;

}

ostream& operator<<(ostream& out, Complex& tmp)

{

out << tmp.Re.GetChisl()<<"/"<< tmp.Re.GetZnam()<<"+"<<tmp.Im.GetChisl()<<"/"<< tmp.Im.GetZnam()<<"\*i";

return out;

}

istream& operator>>(istream& in, Complex& tmp)

{

cout << "Введите целую часть" << endl;

in >> tmp.Re;

cout << "Введите мнимую часть" << endl;

in >> tmp.Im;

return in;

}