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**

|  |  |
| --- | --- |
|  | Выполнил:  студент группы 381908-3  Кошелев С.А  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись)  Проверил:  Карчков Д.А  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |

Нижний Новгород  
2020 г.

Содержание

[Описание задачи 3](#_Toc35011578)

[Код main.cpp 4](#_Toc35011579)

[Код fraction.h 7](#_Toc35011580)

[Код fraction.cpp 8](#_Toc35011581)

[Код Complex.h 12](#_Toc35011582)

[Код Complex.cpp 13](#_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) [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)).

# Код main.cpp

#include <iostream>

#include"fraction.h"

#include "Komp.h"

using namespace std;

int main()

{

int a1,a2,b1, b2,g;

fraction\*dr1 = new fraction();

fraction\*dr2 = new fraction();

Kompl aa, bb, cc,c1;

cout << "Vvedite try complexnih cisla" << endl;

cin >> c1;

cout << "pervoe " << c1 << endl;

cin >> aa;

cout << "vtoroe " << aa << endl;

cin >> bb;

cout << "tret'e " << bb << endl;

cout << "Vvedite chislo " << endl;

cin >> g;

cout <<"proverka in u out" <<c1 << endl;

cc = aa + bb;

cout << "Otvet compl chislo pri slozh "<<cc<<endl;

cc = aa + g;

cout << "Otvet compl chislo pri slozh s int " << cc << endl;

cc = aa - bb;

cout << "Otvet compl chislo pri vicjit " << cc<<endl;

cc = aa / bb;

cout << "Otvet compl chislo pri delen " << cc<<endl;

cc = aa \* bb;

cout << "Otvet compl chislo pri umnozh " << cc << endl;

cc += aa + bb;

cout << "Otvet compl chislo pri slozh sovm s prisv " << cc << endl;

cc -= aa + bb;

cout << "Otvet compl chislo pri vicjit sovm s prisv " << cc << endl;

cc \*= aa + bb;

cout << "Otvet compl chislo pri umnozh sovm s prisv " << cc << endl;

cc /= aa + bb;

cout << "Otvet compl chislo pri delen sovm s prisv " << cc << endl;

cc = aa - g;

cout << "Otvet compl chislo pri vicjit s int " << cc << endl;

cc = aa / g;

cout << "Otvet compl chislo pri delen s int " << cc << endl;

cc = aa - g;

cout << "Otvet compl chislo pri umnozh s int " << cc << endl;

cc != aa;

cout << "Otvet compl chislo pri nerawenstwe " << cc << endl;

cc > aa;

cout << "Otvet compl chislo pri bol'she " << cc << endl;

cc < aa;

cout << "Otvet compl chislo pri men'she " << cc << endl;

cc == aa;

cout << "Otvet compl chislo pri ravenstve " << cc << endl;

cc == aa;

cout << "Otvet compl chislo pri bol'she ili ravno " << cc << endl;

cc == aa;

cout << "Otvet compl chislo pri men'she ili ravno " << cc << endl;

cc /= g;

cout << "Otvet compl chislo pri delen sovm s prisv int " << cc << endl;

cc \*= g;

cout << "Otvet compl chislo pri umnozh sovm s prisv int " << cc << endl;

do

{

cout << "vvedite chislit 1 drobi" << endl;

cin >> a1;

if (a1 == 0)

cout << "Vvedite chislo!=0" << endl;

} while (a1 == 0);

do

{

cout << "vvedite znam 1 drobi" << endl;

cin >> b1;

if (b1 == 0)

cout << "Vvedite chislo!=0" << endl;

} while (b1 == 0);

do

{

cout << "vvedite chislit 2 drobi" << endl;

cin >> a2;

if (a2 == 0)

cout << "Vvedite chislo!=0" << endl;

} while (a2 == 0);

do

{

cout << "vvedite znam 2 drobi" << endl;

cin >> b2;

if (b2 == 0)

cout << "Vvedite chislo!=0" << endl;

} while (b2 == 0);

dr1->SetCH(a1);

dr1->SetZN(b1);

dr2->SetCH(a2);

dr2->SetZN(b2);

Kompl\* zal = new Kompl(\*dr1, \*dr2);

dr1->Proizv(\*dr1, \*dr2);

dr1->Shastn(\*dr1, \*dr2);

dr1->Sum(\*dr1, \*dr2);

dr1->Razn(\*dr1, \*dr2);

zal->Otv();

}

# Код fraction.h

#pragma once

#include <iostream>

using namespace std;

class fraction

{

private:

int chisl;

int znam;

public:

fraction();

fraction(int a, int b);

int GetCH();

void SetCH(int a);

int GetZN();

void SetZN(int a);

void Proizv(fraction \_first, fraction \_second);

void Shastn(fraction \_first, fraction \_second);

void Sum(fraction \_first, fraction \_second);

void Razn(fraction \_first, fraction \_second);

fraction operator+(const fraction& tmp)const;

fraction operator-(const fraction& tmp)const;

fraction operator\*(const fraction& tmp)const;

fraction operator/(const fraction& tmp)const;

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

};

# Код fraction.cpp

#include "fraction.h"

#include <iostream>

using namespace std;

fraction::fraction()

{

this->chisl = 0;

this->znam = 1;

}

fraction::fraction(int a, int b)

{

this->chisl = a;

this->znam = b;

}

int fraction::GetCH()

{

return this->chisl;

}

void fraction::SetCH(int a)

{

this->chisl = a;

}

int fraction::GetZN()

{

return this->znam;

}

void fraction::SetZN(int a)

{

this->znam = a;

}

void fraction::Proizv(fraction \_first, fraction \_second)

{

if (\_first.chisl \* \_second.chisl % \_first.znam \* \_second.znam == 0

) {

cout << "proizvedenie =" << \_first.chisl \* \_second.chisl / \_first.znam \* \_second.znam << endl;

}

else

cout << "proizvedenie =" << \_first.chisl \* \_second.chisl << "/" << \_first.znam \* \_second.znam << endl;

}

void fraction::Shastn(fraction \_first, fraction \_second)

{

cout << "chastnoe =" << \_first.chisl \* \_second.chisl << "/" << \_first.znam \* \_second.znam << endl;

}

void fraction::Sum(fraction \_first, fraction \_second)

{

int b11;

b11 = \_first.znam \* \_second.znam;

\_first.chisl = \_first.chisl \* \_second.znam;

\_second.chisl = \_second.chisl \* \_first.znam;

cout << "summa =" << \_first.chisl + \_second.chisl << "/" << b11 << endl;

}

void fraction::Razn(fraction \_first, fraction \_second)

{

int b11;

b11 = \_first.znam \* \_second.znam;

\_first.chisl = \_first.chisl \* \_second.znam;

\_second.chisl = \_second.chisl \* \_first.znam;

cout << "raznost' =" << \_first.chisl - \_second.chisl << "/" << b11 << endl;

}

fraction fraction::operator+(const fraction & tmp) const

{

fraction res;

res.znam = this->znam\*tmp.znam;

res.chisl = this->chisl\*tmp.znam + this->znam\*tmp.chisl;

return res;

}

fraction fraction::operator-(const fraction & tmp) const

{

fraction res;

res.znam = this->znam\*tmp.znam;

res.chisl = this->chisl\*tmp.znam - this->znam\*tmp.chisl;

return res;

}

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

{

fraction res;

res.chisl = this->chisl \* tmp.chisl;

res.znam = this->znam \* tmp.znam;

return res;

}

fraction fraction::operator/(const fraction & tmp) const

{

fraction res;

res.chisl = this->chisl \* tmp.znam;

res.znam = this->znam \* tmp.chisl;

return res;

}

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

{

int a,b;

in >> a;

in >> b;

tmp.SetCH(a);

tmp.SetZN(b);

return in;

}

# Код Komp.h

#pragma once

#include<iostream>

#include "fraction.h"

class Kompl

{

public:

Kompl operator+(const Kompl& tmp)const;

Kompl& operator=(const Kompl& tmp);

Kompl operator+(int tmp);

Kompl operator-(const Kompl& tmp)const;

Kompl operator/(const Kompl& tmp)const;

Kompl operator\*(const Kompl& tmp)const;

Kompl& operator+=(const Kompl& tmp);

Kompl& operator-=(const Kompl& tmp);

Kompl& operator\*=(const Kompl& tmp);

Kompl& operator/=(const Kompl& tmp);

Kompl operator-(int tmp);

Kompl operator/(int tmp);

Kompl operator\*(int tmp);

bool operator!=(Kompl& tmp);

bool operator>(Kompl& tmp);

bool operator<(Kompl& tmp);

bool operator==(Kompl& tmp);

bool operator>=(Kompl& tmp);

bool operator<=(Kompl& tmp);

Kompl& operator+=(int tmp);

Kompl& operator-=(int tmp);

Kompl& operator/=(int tmp);

Kompl& operator\*=(int tmp);

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

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

Kompl(const Kompl & tmp);

Kompl(fraction \_first, fraction \_second);

Kompl();

void Otv();

~Kompl();

private:

fraction a;

fraction b;

};

# Код Komp.cpp

#include "Komp.h"

#include "fraction.h"

#include <iostream>

using namespace std;

Kompl::Kompl(const Kompl&tmp) {

this->a = tmp.a;

this->b = tmp.b;

}

Kompl::Kompl(fraction \_first, fraction \_second)

{

this->a = \_first;

this->b = \_second;

} //Конструктор сконструировался

Kompl::Kompl()

{

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

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

}//Конструктор 2 сконструировался

void Kompl::Otv()

{

int h1, h2;

int c1, c2;

h1 = this->a.GetCH();

h2 = this->b.GetCH();

c1 = this->a.GetZN();

c2 = this->b.GetZN();

cout << "Komlexnoe chislo=(" << h1 << "/" << c1 << ")+(i\*" << h2 << "/" << c2 << ")" << endl;

}

Kompl Kompl::operator+(const Kompl & tmp) const

{

Kompl res;

res.a = this->a + tmp.a; //комплексное число выглядит как a+b\*i

res.b = this->b + tmp.b; //комплексное число выглядит как a+b\*i

return res;

}

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

{

this->a = tmp.a; //a=4

this->b = tmp.b; //Перегрузка = перегрузилась

return \*this;

}

Kompl Kompl::operator+(int tmp)

{

Kompl res;

res = \*this;

res.a.SetCH(this->a.GetZN()\*tmp + this->a.GetCH());

return res;

}

Kompl Kompl::operator-(const Kompl & tmp) const

{

Kompl res;

res.a = this->a - tmp.a;

res.b = this->b - tmp.b;

return res;

}

Kompl Kompl::operator/(const Kompl & tmp) const

{

Kompl res;

res.a = (this->a\*tmp.a + this->b\*tmp.b) / (tmp.a\*tmp.a + tmp.b\*tmp.b);

res.a = (tmp.a\*this->b - tmp.b\*this->a) / (tmp.a\*tmp.a + tmp.b\*tmp.b);

return Kompl();

}

Kompl Kompl::operator\*(const Kompl & tmp) const

{

Kompl res;

res.a = this->a\*tmp.a - this->b\*tmp.b;

res.b = this->a\*tmp.b - this->b\*tmp.a;

return res;

}

Kompl& Kompl::operator+=(const Kompl & tmp)

{

\*this = \*this + tmp;

return \*this;

}

Kompl& Kompl::operator-=(const Kompl & tmp)

{

\*this = \*this - tmp;

return \*this;

}

Kompl& Kompl::operator\*=(const Kompl & tmp)

{

\*this = \*this \* tmp;

return \*this;

}

Kompl& Kompl::operator/=(const Kompl & tmp)

{

\*this = \*this / tmp;

return \*this;

}

Kompl Kompl::operator-(int tmp)

{

Kompl res;

res = \*this;

res.a.SetCH(this->a.GetCH() - this->a.GetZN()\*tmp);

return res;

}

Kompl Kompl::operator/(int tmp)

{

Kompl res;

res = \*this;

res.a.SetZN(this->a.GetZN()\*tmp);

return res;

}

Kompl Kompl::operator\*(int tmp)

{

Kompl res;

res = \*this;

res.a.SetCH(this->a.GetCH()\*tmp);

return res;

}

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

{

int a,b;

a = this->a.GetCH()\*tmp.a.GetZN();

b = tmp.a.GetCH()\*this->a.GetZN();

if (a != b)

return true;

else

return false;

}

bool Kompl::operator>(Kompl & tmp)

{

int a, b;

a = this->a.GetCH()\*tmp.a.GetZN();

b = tmp.a.GetCH()\*this->a.GetZN();

if (a > b)

return true;

else

return false;

return false;

}

bool Kompl::operator<(Kompl & tmp)

{

int a, b;

a = this->a.GetCH()\*tmp.a.GetZN();

b = tmp.a.GetCH()\*this->a.GetZN();

if (a < b)

return true;

else

return false;

}

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

{

int a, b;

a = this->a.GetCH()\*tmp.a.GetZN();

b = tmp.a.GetCH()\*this->a.GetZN();

if (a == b)

return true;

else

return false;

}

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

{

int a, b;

a = this->a.GetCH()\*tmp.a.GetZN();

b = tmp.a.GetCH()\*this->a.GetZN();

if (a >= b)

return true;

else

return false;

}

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

{

int a, b;

a = this->a.GetCH()\*tmp.a.GetZN();

b = tmp.a.GetCH()\*this->a.GetZN();

if (a <= b)

return true;

else

return false;

}

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

{

\*this = \*this + tmp;

return \*this;

}

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

{

\*this = \*this - tmp;

return \*this;

}

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

{

\*this = \*this / tmp;

return \*this;

}

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

{

\*this = \*this \* tmp;

return \*this;

}

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

{

in >> tmp.a;

in >> tmp.b;

return in;

}

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

{

out << tmp.a.GetCH() << "/" << tmp.a.GetZN() << "+" << tmp.b.GetCH() << "/" << tmp.b.GetZN()<<"\*i";

return out;

}

Kompl::~Kompl()

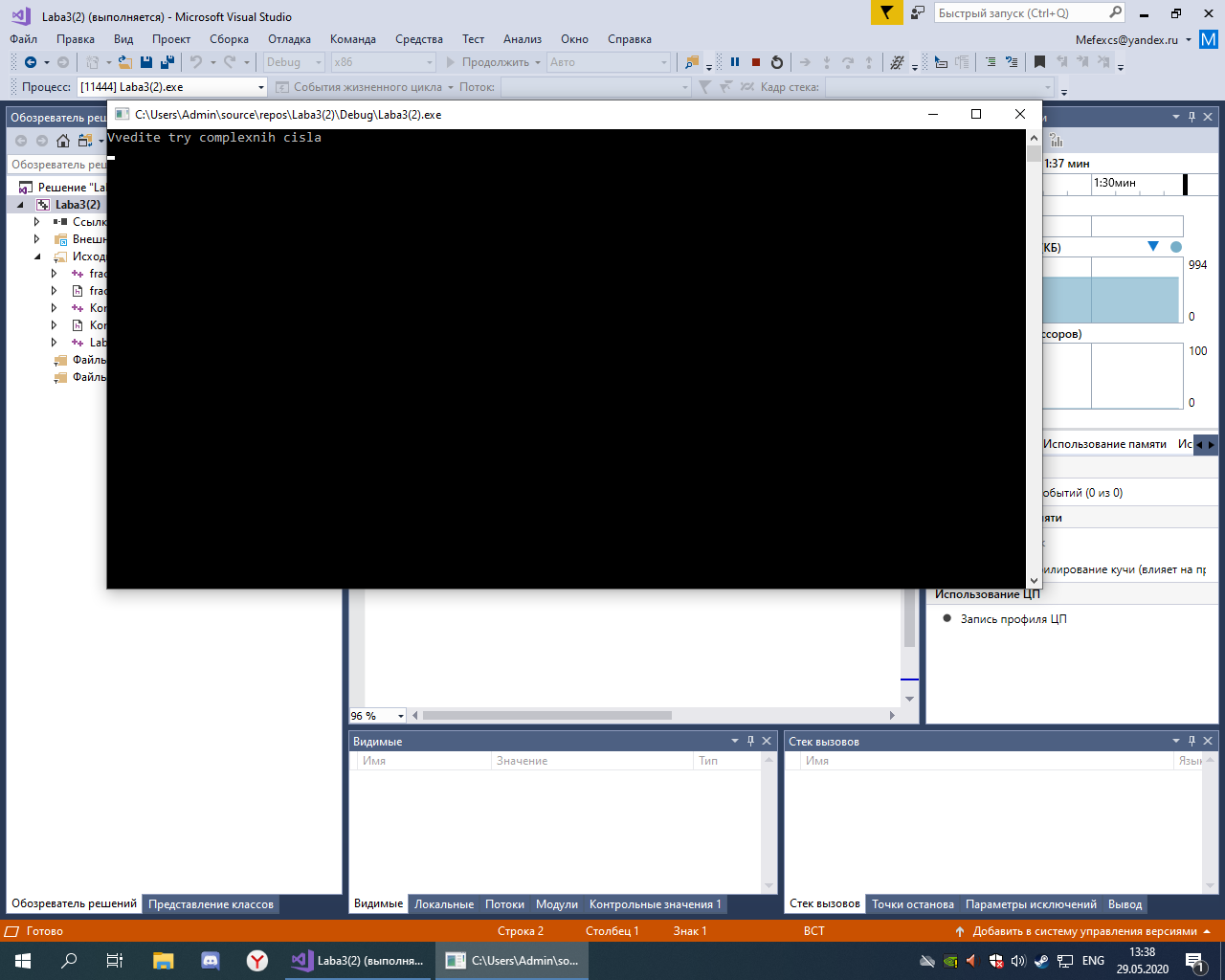
{

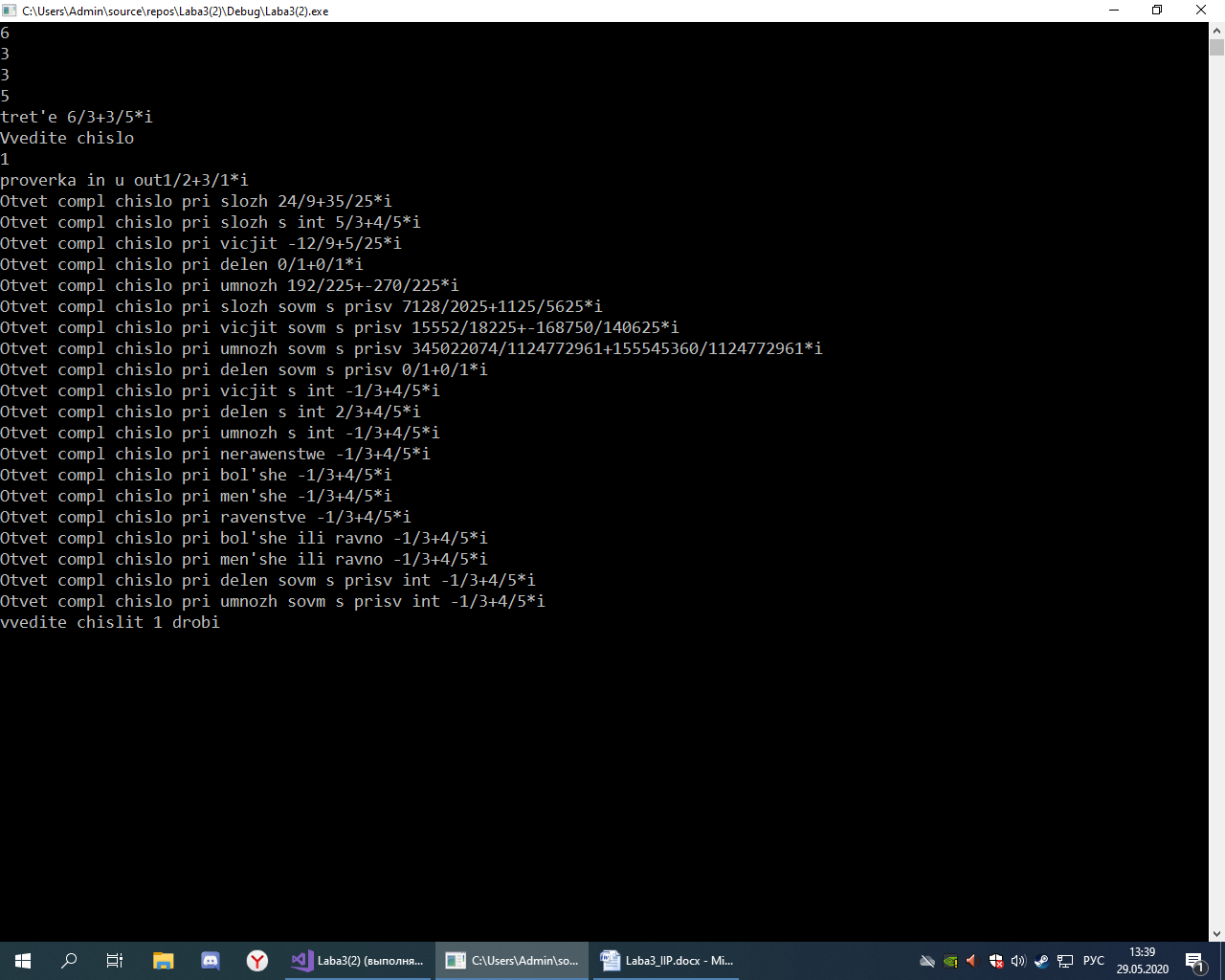
this->a = fraction(0, 1);

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

} //Деструктор сдеструктил

Скриншоты программы





# Вывод:

Я научился использовать перегрузки. Они нужны для того, чтобы выполнять операции с нестандартными типами (собственными классами).