МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра информационных систем

ОТЧЕТ

по практической работе №2

по дисциплине «Объектно-ориентированное программирование»

Студенты гр. 8363 _____ Нерсисян А.С.

_____ Панфилович А.И.

Преподаватель _____ Егоров С.С.

Санкт-Петербург

2021

Задание на практическую работу

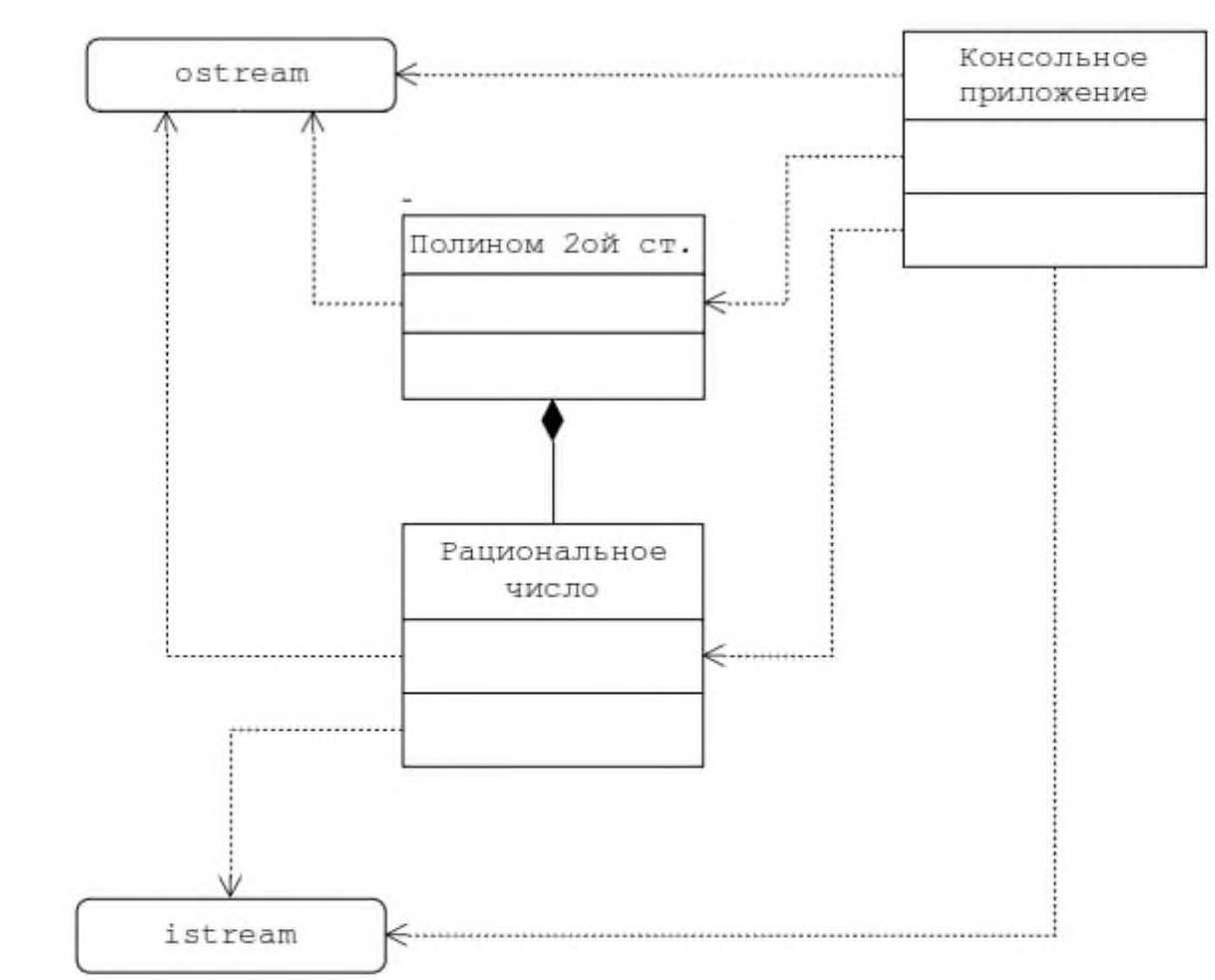

Рис.1. Диаграмма классов работы №2

Создать консольное приложение, реализующее функции перечисленные в описании работы №1 (вычисление корней, вычисление значения, представление полинома в классической и канонических формах) на множестве рациональных чисел.

Рациональное число – это **несократимая** дробь a/b, где a и b целые, причем $b > 0$.

Приложение должно включать основной модуль, модуль «application», модуль «polinom» и модуль «rational».

Для этого в проект лабораторной работы №1 следует добавить модуль с описанием и реализацией класса рациональных чисел Rational. Класс Rational

должен быть встроен в проект согласно диаграмме классов на рис.2. При этом основной модуль, модуль «application» и модуль «polinom» не должны изменяться. Изменения вносятся лишь в заголовочный файл number.h, где

```
typedef int number;
```

следует заменить на

```
#include «rational.h»
typedef Rational number;
```

В классе **Rational** следует определить только те члены класса и спецификации, которые необходимы для совместимости модулей проекта и реализации отношений, приведенных в ДК объектной модели.

Реализовать и отладить программу, удовлетворяющую сформулированным требованиям и заявленной цели. Разработать контрольные примеры и оттестировать на них программу. Оформить отчет, сделать выводы по работе.

Спецификации классов

Класс Rational

Атрибуты:

Тип	Наименование	Область видимости
int	a	private
unsigned int	b	private

Атрибуты a, b используется для хранения рациональных дробей.

Модуль **«Rational»** содержит спецификацию класса "Рациональное число" и реализацию его методов.

Методы **getA();** и **getB();** имеет тип возвращаемого значения int и unsigned int соответственно, не имеет формальных параметров, область видимости public, реализовывают доступ к атрибутам класса **Rational**;

Арифметические операторы

```
Rational operator* (int);          //умножение рационального числа на целое
Rational operator* (Rational);  //умножение рациональных чисел
Rational operator/ (Rational);   //деление рациональных чисел
Rational operator+ (Rational);  //сложение рациональных чисел
Rational operator- (Rational);   //вычитание рациональных чисел
Rational operator- (); //унарный оператор вычитания рационального числа
```

Операторы сравнения

```
bool operator== (Rational);      // равно
bool operator!= (Rational);      // не равно
bool operator>= (Rational);      // больше либо равно
bool operator<= (Rational);      //меньше либо равно
bool operator> (Rational);       //строго больше
bool operator< (Rational);       //строго меньше
```

Функции

friend **int getNod(int a, int b);** рекурсивная функция нахождения НОД, принимает два аргумента целого типа, возвращает наибольший общий делитель аргументов;

friend **std::istream& operator>> (std::istream&, Rational&);** определение и реализация стандартного входного потока для типа данных Rational.

friend **std::ostream& operator<< (std::ostream&, Rational);** определение и реализация стандартного выходного потока для типа данных Rational.

friend **Rational sqrt(Rational);** переопределение (перегрузка) глобальной функции ::sqrt(double& int) для типа данных Rational.

friend **Rational abs(Rational);** переопределение (перегрузка) глобальной функции ::abs(double& int) для типа данных Rational.

Описание контрольного примера с исходными и ожидаемыми (расчетными) данными

Пример: $P(x) = \frac{2}{5}x^2 + \frac{4}{5}x + \frac{112}{405} \quad x_0 = \frac{1}{8}$

$$a = \frac{2}{5}, \qquad b = \frac{4}{5}, \qquad c = \frac{112}{405}, \qquad x = \frac{1}{8}$$

$$D = \frac{16}{81}$$

$$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$$

$$x_1 = \frac{-\frac{4}{5} + \sqrt{\frac{16}{81}}}{2 * \frac{2}{5}} = -\frac{4}{9}$$

$$x_2 = \frac{-\frac{4}{5} - \sqrt{\frac{16}{81}}}{2 * \frac{2}{5}} = -\frac{14}{9}$$

$$P\left(\frac{1}{8}\right) = \frac{4961}{12960}$$

Скриншоты программы на контрольных примерах

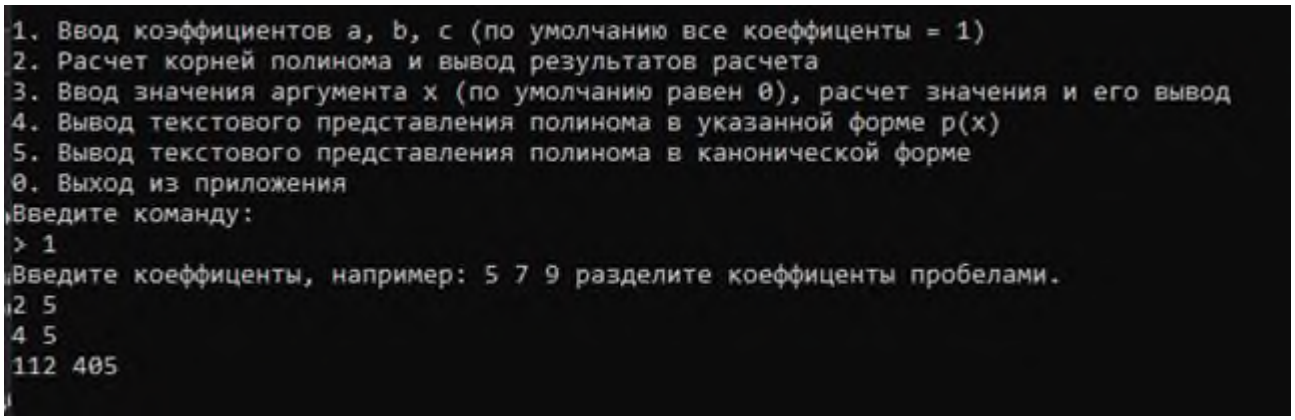

Рисунок 1. Ввод значений коэффициентов

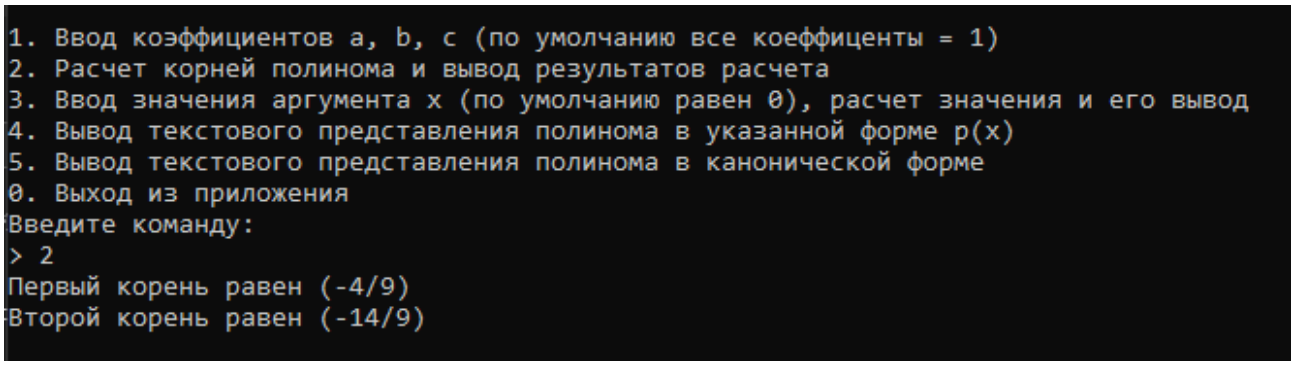

Рисунок 2. Расчет корней и вывод результатов расчета

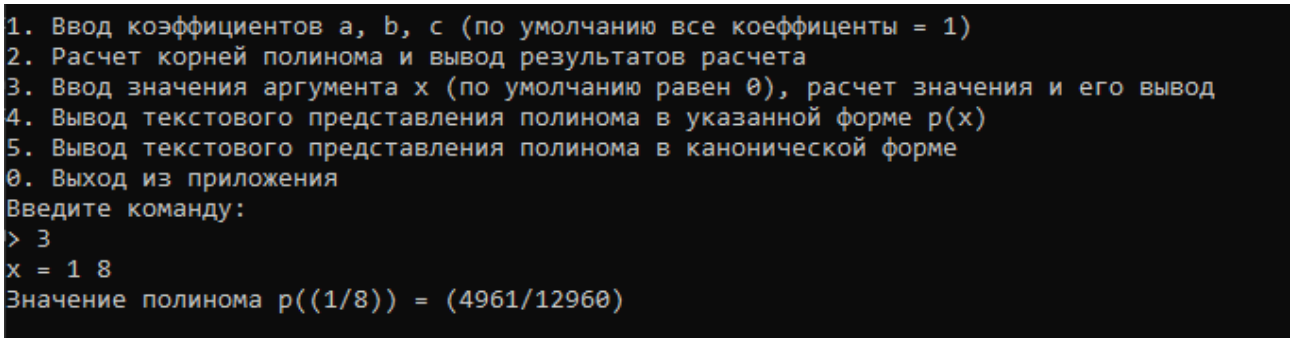

Рисунок 3. Ввод значения аргумента x, расчет значения и его вывод

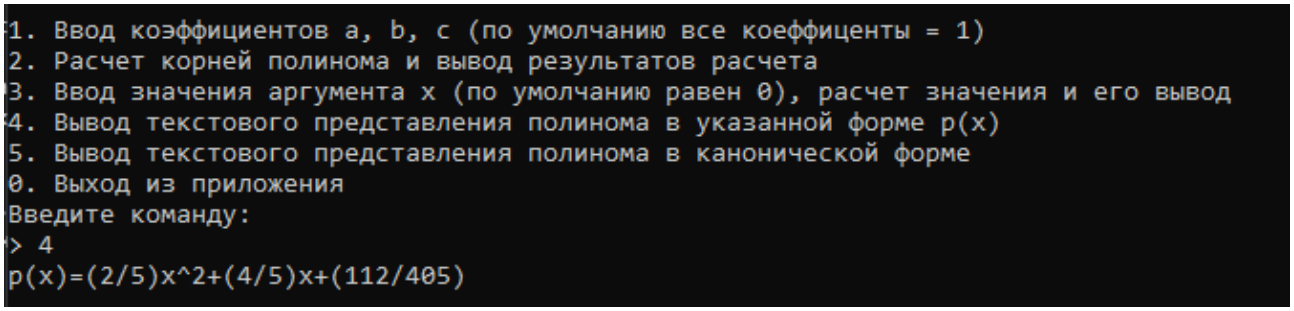

Рисунок 4. Вывод текстового представления полинома в указанной форме p(x)

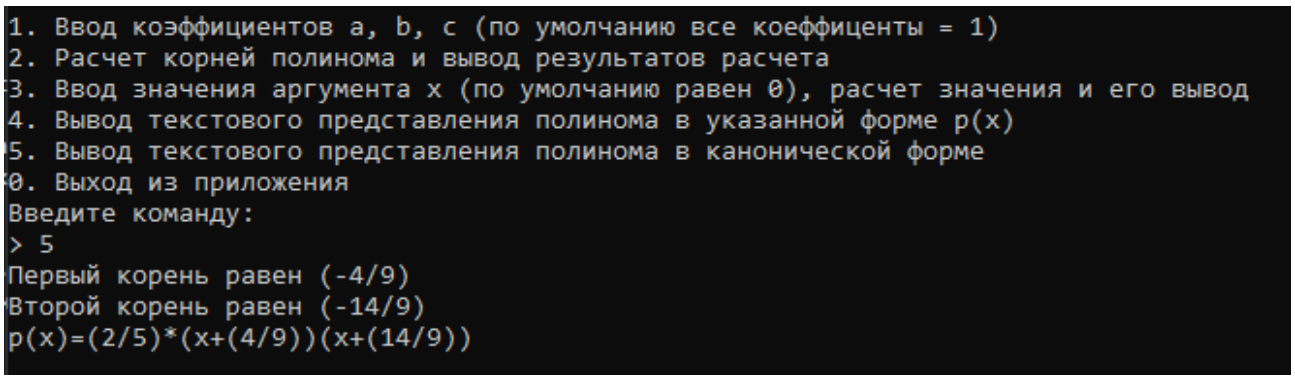

Рисунок 5. Вывод текстового представления полинома в канонической форме

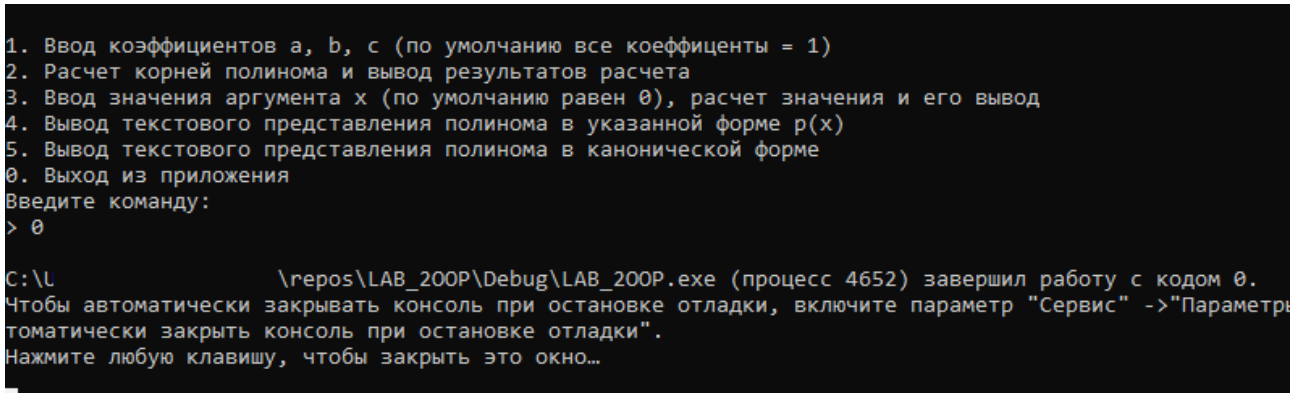

Рисунок 6. Выход из приложения

Выводы по выполнению работы

В рамках данной практической работы была реализована и отлажена программа, удовлетворяющая сформулированным требованиям и заявленным целям. Разработаны контрольные примеры, и программа оттестирована на них.

Тщательно изучил определение и перегрузку операторов потокового ввода и вывода, операторов сравнения, а также бинарных и унарных арифметических операторов.

На практике изучил важные аспекты и правила написания кода с использованием объектно-ориентированной модели программирования.

ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ КОД ПРОГРАММЫ

FILE number.h

```
#ifndef NUMBER_H
#define NUMBER_H

#include "rational.h"

typedef Rational number;

#endif
```

FILE rational.h

```
#include <iostream>
#include <cmath>

#ifndef RATIONAL_H
#define RATIONAL_H

int getNod(int a, int b);

class Rational {
private:
    int a;
    unsigned int b;

public:
    Rational();
    Rational(const int&);

    friend std::istream& operator>> (std::istream&, Rational&);
    friend std::ostream& operator<< (std::ostream&, Rational);
    friend int getNod(int, int);
    friend Rational sqrt(Rational);
    friend Rational abs(Rational);

    Rational operator* (int);
    Rational operator* (Rational);
    Rational operator/ (Rational);
    Rational operator+ (Rational);
    Rational operator- (Rational);
    Rational operator- ();
    bool operator== (Rational);
    bool operator!= (Rational);
    bool operator>= (Rational);
    bool operator<= (Rational);
    bool operator> (Rational);
```

```
    bool operator< (Rational);

    int getA();
    unsigned int getB();

};

#endif
```

FILE rational.cpp

```
#include "rational.h"

Rational::Rational() {};

Rational::Rational(const int& a_)
{
    a = a_;
    b = 1;
}

int Rational::getA()
{
    return a;
}

unsigned int Rational::getB()
{
    return b;
}

//НОД
int getNod(int a, int b)
{
    return b ? getNod(b, a % b) : a;
}


Rational sqrt(Rational rt)
{
    double a, b;
    a = ::sqrt(rt.a);
    b = ::sqrt(rt.b);
    // переобразовываем double a и b в int, далее снова в double
    // если переобразованый совпадает с исходным, значит число целое
    // например имеем раьиональное число 16/81
    // при извлечении квадратного корня получаем 4.0/9.0
    // if (((double)(int)4.0) == 4.0)
    // т.е 4.0 -> 4 -
> 4.0 == 4.0 значит можно записать в аттрибуты объекта (int)
```

```
    // заметим, что если бы a получилось скажем 4.254... то
    // при сравнении (((double)(int)4.254) == 4.254) получаем 4.0 == 4.25
    // и результат сравнения false
    if ((((double)(int)a) == a) && (((double)(int)b) == b))
    {
        rt.a = a;
        rt.b = b;
        return rt;
    }
    else
    {
        std::cout << "Ошибка, квадратный корень из дискриминанта не " <<
            "рациональная дробь!" << std::endl;
        rt.a = 1;
        rt.b = 2;
        return rt;
    }
}


Rational abs(Rational rt)
{
    rt.a = ::abs(rt.a);
    rt.b = rt.b;
    return rt;
}

std::istream& operator>> (std::istream& is, Rational& rt)
{
    do {
        is >> rt.a >> rt.b;
        if ((rt.b <= 1)) {
            std::cout << "Знаменатель не может быть 0, 1 или отрицательны
м, повторите попытку" <<
                std::endl;
            continue;
        }
        // если a без остатка делится на b
        if ((double(int((double(rt.a) / double(rt.b)))) == (double(rt.a)
/ double(rt.b))))
        {
            std::cout << "Введеные числа НЕ образуют рациональную дробь,
повторите попытку" <<
                std::endl;
            continue;
        }
        return is;
    } while (true);
}
```

```
std::ostream& operator<< (std::ostream& os, Rational rt)
{
    //вызвать функцию нахождения НОД-а
    //проверить можно ли сократить дробь
    //если да, то скоратить, потом вывести на экран
    int nod = ::abs(getNod(rt.a, rt.b));
    if (nod >= 1) // НОД есть, сокращаем
    {
        rt.a /= nod;
        rt.b /= nod;
    }
    os << "(" << rt.a << "/" << rt.b << ")";
    return os;
}


Rational Rational::operator- ()
{
    Rational rt;
    rt.a = a * (-1);
    rt.b = b;
    return rt;
}

//умножение рациональной дроби с целым
Rational Rational::operator* (int integer)
{
    Rational rational;
    rational.a = a * integer;
    rational.b = b;
    //вызвать функцию нахождения НОД-а
    //проверить можно ли сократить дробь после умножения
    int nod = getNod(rational.a, rational.b);
    if (nod >= 1) // НОД есть, сокращаем
    {
        rational.a /= nod;
        rational.b /= nod;
        return rational;
    }
    else
    {
        return rational;
    }
}

//сложение рациональных дробей
Rational Rational::operator+ (Rational rt)
{
    Rational rational;
```

```
    rational.a = a * rt.b + b * rt.a;
    rational.b = b * rt.b;
    //вызвать функцию нахождения НОД-а
    //проверить можно ли сократить дробь после сложения дробей
    int nod = getNod(rational.a, rational.b);
    if (nod >= 1) // НОД есть, сокращаем
    {
        rational.a /= nod;
        rational.b /= nod;
        return rational;
    }
    else
    {
        return rational;
    }
}

//вычитание рациональных дробей
Rational Rational::operator- (Rational rt)
{
    Rational rational;
    rational.a = a * rt.b - b * rt.a;
    rational.b = b * rt.b;
    //вызвать функцию нахождения НОД-а
    //проверить можно ли сократить дробь после сложения дробей
    int nod = getNod(rational.a, rational.b);
    if (nod >= 1) // НОД есть, сокращаем
    {
        rational.a /= nod;
        rational.b /= nod;
        return rational;
    }
    else
    {
        return rational;
    }
}

//умножение рациональных дробей
Rational Rational::operator* (Rational rt)
{
    Rational rational;
    rational.a = a * rt.a;
    rational.b = b * rt.b;
    //вызвать функцию нахождения НОД-а
    //проверить можно ли сократить дробь после умножения
    int nod = getNod(rational.a, rational.b);
    if (nod >= 1) // НОД есть, сокращаем
    {
        rational.a /= nod;
```

```
        rational.b /= nod;
        return rational;
    }
    else
    {
        return rational;
    }
}

//деление рациональных дробей
Rational Rational::operator/ (Rational rt)
{
    Rational rational;
    rational.a = a * rt.b;
    rational.b = b * rt.a;
    //вызвать функцию нахождения НОД-а
    //проверить можно ли сократить дробь после умножения
    int nod = getNod(rational.a, rational.b);
    if (nod >= 1) // НОД есть, сокращаем
    {
        rational.a /= nod;
        rational.b /= nod;
        return rational;
    }
    else
    {
        return rational;
    }
}

bool Rational::operator== (Rational rt)
{
    return (a == rt.a) && (b == rt.b);
}

bool Rational::operator!= (Rational rt)
{
    return (a != rt.a) || (b != rt.b);
}

bool Rational::operator>= (Rational rt)
{
    return ((double(a) / double(b)) >= (double(rt.a) / double(rt.b)));
}

bool Rational::operator<= (Rational rt)
{
    return ((double(a) / double(b)) >= (double(rt.a) / double(rt.b)));
}
```

```
bool Rational::operator> (Rational rt)
{
    return ((double(a) / double(b)) > (double(rt.a) / double(rt.b)));
}

bool Rational::operator< (Rational rt)
{
    return ((double(a) / double(b)) > (double(rt.a) / double(rt.b)));
}
```

FILE application.h

```
#include <iostream>
#include "number.h"
#include "polinom.h"

#ifndef APPLICATION_H
#define APPLICATION_H

class Application
{
private:
    int Menu();
public:
    int exec();
};

#endif
```

FILE application.cpp

```
#include "application.h"

int Application::Menu()
{
    int ch;
    std::cout << std::endl << std::endl <<
        "1. Ввод коэффициентов a, b, c (по умолчанию все коеффиценты = 1)
" << std::endl <<
        "2. Расчет корней полинома и вывод результатов расчета" << std::e
ndl <<
        "3. Ввод значения аргумента x (по умолчанию равен 0), расчет знач
ения и его вывод" << std::endl <<
        "4. Вывод текстового представления полинома в указанной форме p(x
)" << std::endl <<
        "5. Вывод текстового представления полинома в канонической форме"
 << std::endl <<
```

```
        "0. Выход из приложения" << std::endl << "Введите команду:" << st
d::endl << "> ";
    std::cin >> ch;
    return ch;
};

int Application::exec()
{
    setlocale(LC_ALL, "Russian");
    std::cout << "Практическая работа N1" << std::endl <<
        "Приложение для вычисления корней полинома 2-ой степени " <<
        "вида p(x) = a*x^2 + b*x + c (a!=0)" << std::endl <<
        "и его значения для заданного аргумента x " <<
        "на множестве целых чисел.";

    int ch, count = 0;
    number a = 1, b = 2, c = 1, x1, x2;
    number roots[2];
    while (true)
    {
        ch = Menu();
        switch (ch)
        {
        case 0:
            return 0;
        case 1:
            std::cout << "Введите коэффиценты, например: 5 7 9 " <<
                "разделите коэффиценты пробелами." << std::endl;
            std::cin >> a >> b >> c;
            break;
        case 2: {
            Polinom p(a, b, c);
            //прежде чем использовать указатель roots, надо проверить на
nullptr
            count = p.Calculate(roots);
            break;
        }
        case 3: {
            number x = 0;
            std::cout << "x = ";
            std::cin >> x;
            Polinom p(a, b, c);
            number v = p.value(x);
            std::cout << "Значение полинома p(" << x << ") = " << v << st
d::endl;
            break;
        }
        case 4: {
            Polinom p(a, b, c);
            p.setPrintMode(EPrintModeClassic);
```

```
            std::cout << p << std::endl;
            break;
        }
        case 5: {
            Polinom p(a, b, c);
            p.setPrintMode(EprintModeCanonical);
            std::cout << p << std::endl;
            break;
        }
        default:
            std::cout << "Ошибка, неверный ввод" << std::endl;
            break;
        }
    }
    return 0;
};
```

FILE polinom.h

```
#include <iostream>
#include "number.h"

#ifndef POLINOM_H
#define POLINOM_H

enum EPrintMode {
    EPrintModeClassic,
    EprintModeCanonical,
};

class Polinom
{
private:
    number a, b, c;
    EPrintMode printMode;
public:
    Polinom(number, number, number);
    friend std::ostream& operator<< (std::ostream&, Polinom&);
    number value(number);
    void setPrintMode(EPrintMode);
    int Calculate(number*);
};

#endif
```

FILE polinom.cpp

```
#include "polinom.h"
#include "number.h"
#include "math.h"

Polinom::Polinom(number inputA, number inputB, number inputC)
{
    printMode = EPrintModeClassic;
    a = inputA;
    b = inputB;
    c = inputC;
};

std::ostream& operator<< (std::ostream& os, Polinom& p) {
    if (p.printMode == EPrintModeClassic)
    {
        os << "p(x)=" << p.a << "x^2" << (p.b >= 0 ? "+" : "-
") << abs(p.b) <<
            "x" << (p.c >= 0 ? "+" : "-") << abs(p.c) << std::endl;
    }
    else
    {
        number roots[2];
        int count = p.Calculate(roots);
        if (count == 0) return os;
        //два корня
        if (count == 2) {
            std::cout << "p(x)=" << p.a << "*(x" << (roots[0] >= 0 ? "-
" : "+") << abs(roots[0]) <<
                ")(x" << (roots[1] >= 0 ? "-
" : "+") << abs(roots[1]) << ")" << std::endl;
        }

        //один корень
        if (count == 1) {
            std::cout << "p(x)=" << p.a << "(x" << (roots[0] >= 0 ? "-
" : "+") << abs(roots[0]) <<
                ")^2" << std::endl;
        }
    }
    return os;
};

number Polinom::value(number x)
{
    return a * x * x + b * x + c;
};

void Polinom::setPrintMode(EPrintMode mode)
```

```
{
    printMode = mode;
};

int Polinom::Calculate(number* roots)
{
    //при Д>0
    number d = ((b * b) - (a * c * 4));
    if (d > 0) //Если дискриминант больше 0
    {
        roots[0] = ((-b + sqrt(d)) / (a * 2));
        roots[1] = ((-b - sqrt(d)) / (a * 2));

        if (a * roots[0] * roots[0] + b * roots[0] + c == 0 &&
            a * roots[1] * roots[1] + b * roots[1] + c == 0) {
            std::cout << "Первый корень равен " << roots[0] << std::endl;
            std::cout << "Второй корень равен " << roots[1] << std::endl;
            return 2;
        }
        else
        {
            std::cout << "Полином не разложим над полем целых" << std::en
dl;
            return 0;
        }
    }

    //при Д=0
    if (d == 0)
    {
        roots[0] = ((-b) / (a * 2));
        if (a * roots[0] * roots[0] + b * roots[0] + c == 0) {
            std::cout << "Корень равен " << roots[0] << std::endl;
            return 1;
        }
        else {
            std::cout << "Полином не разложим над полем целых" << std::en
dl;
            return 0;
        }
    }

    //при Д<0
    else
    {
        std::cout << "Полином не разложим над полем целых" << std::endl;
        return 0;
    }
};
```

FILE main.cpp

```
//Object-Oriented Programming
//Practice Two

#include "application.h"

int main()
{
    Application app;
    return app.exec();
};
```

FILE makefile

```
all : main

.PHONY : all clean

CC = g++
LD = g++

main : main.o application.o polinom.o rational.o

main.o: main.cpp application.h polinom.h number.h rational.h

application.o: application.cpp application.h

polinom.o: polinom.cpp polinom.h

rational.o: rational.cpp rational.h
```