**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №3**

**по дисциплине «Объектно-ориентированное программирование»**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 8363 |  | Нерсисян А.С. |
|  |  | Панфилович А.И. |
| Преподаватель |  | Егоров С.С. |

Санкт-Петербург

2021

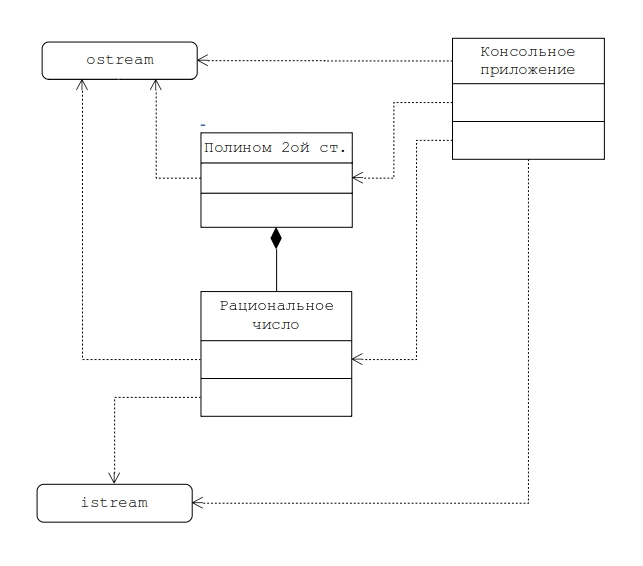
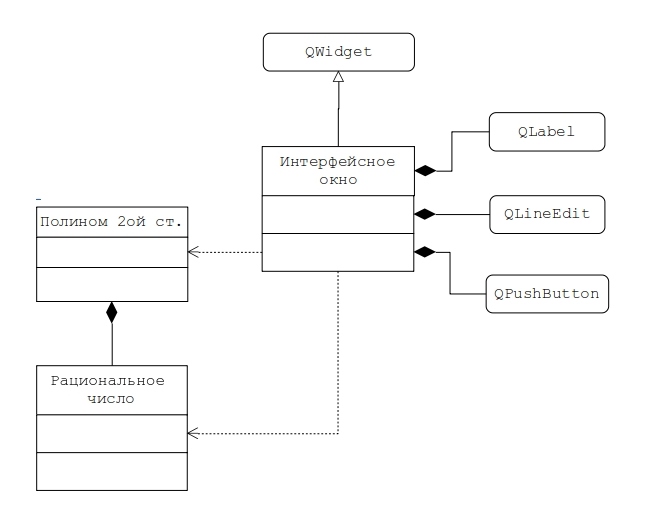
**Задание на практическую работу**

Рис.1. Диаграмма классов работы №3

Создать GUI приложение, реализующее функции перечисленные в описании работы №1 (вычисление корней, вычисление значения, представление полинома в классической и канонических формах) на множестве рациональных чисел. Приложение должно включать основной модуль, модуль «Interface», модуль «polinom» и модуль «Rational».

Основной модуль main.cpp GUI приложения может иметь вид:

#include <QApplication>

#include "interface.h"

int **main** (int argc, char \*argv[])

{

QApplication a (argc, argv);

TInterface interface;

interface.show();

return a.exec();

}

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

Спецификации классов

Класс Polinom

Атрибуты:

|  |  |  |
| --- | --- | --- |
| Тип | Наименование | Область видимости |
| number | a | private |
| number | b | private |
| number | c | private |
| enum EPrintMode | printMode | private |

Атрибуты a, b, c используется для хранения коэффициентов; printMode для хранения вида вывода на экран полинома P(x).

Модуль **«Polinom»** содержит спецификацию класса "Полином 2-ой степени" и реализацию его методов.

Конструктор **Polinom(number, number, number)** имеет 3 формальных параметров типа number, область видимости public, реализовывает создание объекта и задание значений коэффициентов полинома.

Метод **value(number)** имеет тип возвращаемого значения **number**, имеет 1 формальный параметр типа number, область видимости **public**, реализовывает расчет значения полинома по аргументу **x**.

Метод **Calculate()** имеет тип возвращаемого значения **number\***, не имеет формальных параметров, область видимости **public**, реализовывает расчет и вывод корней полинома, а также возвращает массив корней.

Метод **Calculate(Polinom&)** имеет тип возвращаемого значения **QString**, имеет один формальный параметр типа **Polinom**, область видимости **public**, реализовывает вывод корней полинома в строку **QString**, работает в паре с методом **int Calculate(number\*)**.

Метод **setPrintMode(EPrintMode)** не возвращает никакие параметры (void), имеет один формальный параметр EPrintMode, область видимости public, задает вид полинома выводимого на экран.

Перегрузка оператора **QString& operator<< (QString&, Polinom&)**, определяет оператор **<<** для вывода типа данных **Polinom** в **QString.**

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. Он имеет три конструктора:

* базовый, Rational();
* с одним аргументом, только числитель, знаменатель устанавливается в значение по умолчанию (1), Rational(const int&);
* с двумя аргументами, числитель и знаменатель, Rational(int, unsigned int).

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**

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

friend QString& operator<<(QString&, Rational); определение и реализация вывода строки в QString 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.

Класс Interface

Атрибуты:

|  |  |  |
| --- | --- | --- |
| Тип | Наименование | Область видимости |
| QLabel\* | name\_a, | private |
| QLabel\* | delimeter\_a | private |
| QLineEdit\* | a\_numerator, | private |
| QLineEdit\* | a\_denominator | private |
| QLabel\* | name\_b, | private |
| QLabel\* | delimeter\_b | private |
| QLineEdit\* | b\_numerator, | private |
| QLineEdit\* | b\_denominator | private |
| QLabel\* | name\_c, | private |
| QLabel\* | delimeter\_c | private |
| QLineEdit\* | c\_numerator, | private |
| QLineEdit\* | c\_denominator | private |
| QLabel\* | name\_x, | private |
| QLabel\* | delimeter\_x | private |
| QLineEdit\* | x\_numerator, | private |
| QLineEdit\* | x\_denominator | private |
| QPushButton\* | value\_btn | private |
| QPushButton\* | root\_btn | private |
| QPushButton\* | print\_classic\_btn | private |
| QPushButton\* | print\_canonic\_btn | private |
| QLabel\* | output | private |

Атрибуты name\_\* используются для хранения текстовых меток, выводимых на графический интерфейсе программы.

Атрибуты delimeter\_\* используются для хранения текстовых меток, знаков разделителей, выводимых на графический интерфейсе программы.

Атрибуты \*\_ numerator и \*\_denominator используются для хранения значений коэффициентов и аргумента полинома.

Атрибуты \*\_btn используются для хранения инструкций, выполняемых при нажатии кнопок в графическом интерфейсе.

Модуль **«Interface»** содержит спецификацию класса " **Interface**" и реализовывает графический интерфейс программы.

Описание контрольного примера с исходными и ожидаемыми (расчетными) данными

Пример:

Скриншоты программы на контрольных примерах

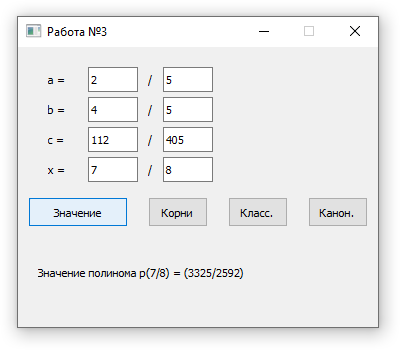


Рисунок 1. Ввод значения аргумента х, расчет значения и его вывод

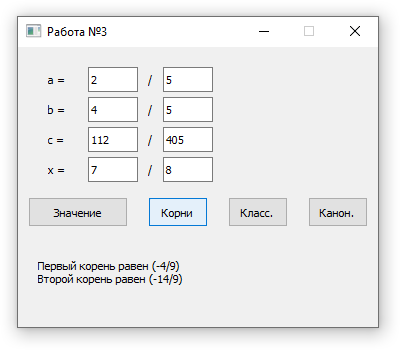


Рисунок 2. Расчет корней и вывод результатов расчета

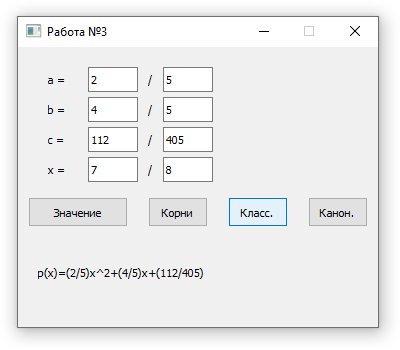


Рисунок 3. Вывод текстового представления полинома в классической форме

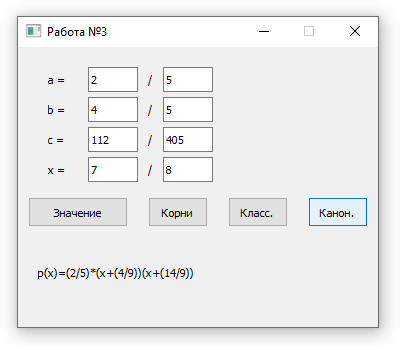


Рисунок 4. Вывод текстового представления полинома в канонической форме

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.

Получил практические навыки работы со средой разработки Qt Creator.

Познакомился с библиотекой классов Qt, с их помощью реализовал приложение с графическим интерфейсом на основе QWidget, на языке программирования C++.

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 // NUMBER\_H

FILE rational.h

#include <iostream>

#include <cmath>

#include <QString>

#include <QMessageBox>

#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&);

    Rational(int, unsigned int);

    friend Rational sqrt(Rational);

    friend Rational abs(Rational);

    friend QString& operator<<(QString&, 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 // RATIONAL\_H

FILE rational.cpp

#include "rational.h"

Rational::Rational() {};

Rational::Rational(const int& a\_)

{

    a = a\_;

    b = 1;

}

Rational::Rational(int a, unsigned int b)

{

    this->a = a;

    this->b = b;

}

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;

}

QString& operator<<(QString& qstr, 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;

    }

    qstr += "(";

    qstr += QString().setNum(rt.a);

    qstr += "/";

    qstr += QString().setNum(rt.b);

    qstr += ")";

    return qstr;

}

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 а получилось скажем 4.254... то

    // при сравнении (((double)(int)4.254) == 4.254) получаем 4.0 == 4.254

    // и результат сравнения false

    if ((((double)(int)a) == a) && (((double)(int)b) == b))

    {

        rt.a = a;

        rt.b = b;

        return rt;

    }

    else

    {

        QMessageBox messageBox;

        messageBox.critical(0,"Ошибка","Квадратный корень из дискриминанта не рациональная дробь!");

        messageBox.setFixedSize(500,200);

        rt.a = 1;

        rt.b = 2;

        return rt;

    }

}

Rational abs(Rational rt)

{

    rt.a = ::abs(rt.a);

    rt.b = rt.b;

    return rt;

}

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 interface.h

#ifndef INTERFACE\_H

#define INTERFACE\_H

#include <QWidget>

#include <QLabel>

#include <QLineEdit>

#include <QPushButton>

#include "polinom.h"

class Interface : public QWidget

{

    Q\_OBJECT

    QLabel \*name\_a, \*delimeter\_a;

    QLineEdit \*a\_numerator, \*a\_denominator;

    QLabel \*name\_b, \*delimeter\_b;

    QLineEdit \*b\_numerator, \*b\_denominator;

    QLabel \*name\_c, \*delimeter\_c;

    QLineEdit \*c\_numerator, \*c\_denominator;

    QLabel \*name\_x, \*delimeter\_x;

    QLineEdit \*x\_numerator, \*x\_denominator;

    QPushButton \*value\_btn;

    QPushButton \*root\_btn;

    QPushButton \*print\_classic\_btn;

    QPushButton \*print\_canonic\_btn;

    QLabel \*output;

public slots:

    void value();

    void root();

    void print\_classic();

    void print\_canonic();

public:

    Interface(QWidget \*parent = nullptr);

    ~Interface();

};

#endif // INTERFACE\_H

FILE interface.cpp

#include "interface.h"

Interface::Interface(QWidget \*parent)

    : QWidget(parent)

{

    setWindowTitle("Работа №3");

    setFixedSize(360,280);

    name\_a = new QLabel("a =", this);

    name\_a->setGeometry(30,20,30,25);

    a\_numerator = new QLineEdit("2", this);

    a\_numerator->setGeometry(70,20,50,25);

    delimeter\_a = new QLabel("/", this);

    delimeter\_a->setGeometry(130,20,30,25);

    a\_denominator = new QLineEdit("5", this);

    a\_denominator->setGeometry(145,20,50,25);

    name\_b = new QLabel("b =", this);

    name\_b->setGeometry(30,50,30,25);

    b\_numerator = new QLineEdit("4", this);

    b\_numerator->setGeometry(70,50,50,25);

    delimeter\_b = new QLabel("/", this);

    delimeter\_b->setGeometry(130,50,30,25);

    b\_denominator = new QLineEdit("5", this);

    b\_denominator->setGeometry(145,50,50,25);

    name\_c = new QLabel("c =", this);

    name\_c->setGeometry(30,80,30,25);

    c\_numerator = new QLineEdit("112", this);

    c\_numerator->setGeometry(70,80,50,25);

    delimeter\_c = new QLabel("/", this);

    delimeter\_c->setGeometry(130,80,30,25);

    c\_denominator = new QLineEdit("405", this);

    c\_denominator->setGeometry(145,80,50,25);

    name\_x = new QLabel("x =", this);

    name\_x->setGeometry(30,110,30,25);

    x\_numerator = new QLineEdit("7", this);

    x\_numerator->setGeometry(70,110,50,25);

    delimeter\_x = new QLabel("/", this);

    delimeter\_x->setGeometry(130,110,30,25);

    x\_denominator = new QLineEdit("8", this);

    x\_denominator->setGeometry(145,110,50,25);

    value\_btn = new QPushButton("Значение", this);

    value\_btn->setGeometry(10,150,100,30);

    root\_btn = new QPushButton("Корни", this);

    root\_btn->setGeometry(130,150,60,30);

    print\_classic\_btn = new QPushButton("Класс.", this);

    print\_classic\_btn->setGeometry(210,150,60,30);

    print\_canonic\_btn = new QPushButton("Канон.", this);

    print\_canonic\_btn->setGeometry(290,150,60,30);

    output = new QLabel(this);

    output->setGeometry(20,200,340,50);

    connect(value\_btn,SIGNAL(pressed()),this, SLOT(value()));

    connect(root\_btn,SIGNAL(pressed()),this, SLOT(root()));

    connect(print\_classic\_btn,SIGNAL(pressed()),this, SLOT(print\_classic()));

    connect(print\_canonic\_btn,SIGNAL(pressed()),this, SLOT(print\_canonic()));

}

Interface::~Interface()

{

    delete name\_a;

    delete delimeter\_a;

    delete a\_numerator;

    delete a\_denominator;

    delete name\_b;

    delete delimeter\_b;

    delete b\_numerator;

    delete b\_denominator;

    delete name\_c;

    delete delimeter\_c;

    delete c\_numerator;

    delete c\_denominator;

    delete name\_x;

    delete delimeter\_x;

    delete x\_numerator;

    delete x\_denominator;

    delete value\_btn;

    delete root\_btn;

    delete print\_classic\_btn;

    delete print\_canonic\_btn;

    delete output;

}

void Interface::value()

{

    number a(a\_numerator->text().toInt(), a\_denominator->text().toUInt());

    number b(b\_numerator->text().toInt(), b\_denominator->text().toUInt());

    number c(c\_numerator->text().toInt(), c\_denominator->text().toUInt());

    number x(x\_numerator->text().toInt(), x\_denominator->text().toUInt());

    Polinom p(a, b, c);

    number v = p.value(x);

    QString qstr("Значение полинома p");

    qstr << x;

    qstr += " = ";

    qstr << v;

    output->setText(qstr);

}

void Interface::root()

{   number a(a\_numerator->text().toInt(), a\_denominator->text().toUInt());

    number b(b\_numerator->text().toInt(), b\_denominator->text().toUInt());

    number c(c\_numerator->text().toInt(), c\_denominator->text().toUInt());

    Polinom p(a, b, c);

    QString qstr = p.Calculate(p);

    output->setText(qstr);

}

void Interface::print\_classic()

{

    number a(a\_numerator->text().toInt(), a\_denominator->text().toUInt());

    number b(b\_numerator->text().toInt(), b\_denominator->text().toUInt());

    number c(c\_numerator->text().toInt(), c\_denominator->text().toUInt());

    Polinom p(a, b, c);

    p.setPrintMode(EPrintModeClassic);

    QString qstr("");

    qstr << p;

    output->setText(qstr);

}

void Interface::print\_canonic()

{

    number a(a\_numerator->text().toInt(), a\_denominator->text().toUInt());

    number b(b\_numerator->text().toInt(), b\_denominator->text().toUInt());

    number c(c\_numerator->text().toInt(), c\_denominator->text().toUInt());

    Polinom p(a, b, c);

    p.setPrintMode(EprintModeCanonical);

    QString qstr("");

    qstr << p;

    output->setText(qstr);

}

FILE polinom.h

#include <iostream>

#include "number.h"

#include <QString>

#ifndef POLINOM\_H

#define POLINOM\_H

enum EPrintMode {

    EPrintModeClassic,

    EprintModeCanonical,

};

class Polinom

{

private:

    number a, b, c;

    EPrintMode printMode;

public:

    Polinom(number, number, number);

    number value(number);

    void setPrintMode(EPrintMode);

    int Calculate(number\*);

    QString Calculate(Polinom&);

    friend QString& operator<< (QString&, Polinom&);

};

#endif // POLINOM\_H

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;

};

QString& operator<< (QString& qstr, Polinom& p)

{

    if (p.printMode == EPrintModeClassic)

    {

        qstr += "p(x)=";

        qstr << p.a;

        qstr += "x^2";

        qstr += (p.b >= 0 ? "+" : "-");

        qstr << abs(p.b);

        qstr += "x";

        qstr += (p.c >= 0 ? "+" : "-");

        qstr << abs(p.c);

    }

    else

    {

        number roots[2];

        int count = p.Calculate(roots);

        if (count == 0) return qstr;

        //два корня

        if (count == 2) {

            qstr += "p(x)=";

            qstr << p.a;

            qstr += "\*(x";

            qstr += (roots[0] >= 0 ? "-" : "+");

            qstr << abs(roots[0]);

            qstr += ")(x";

            qstr += (roots[1] >= 0 ? "-" : "+");

            qstr << abs(roots[1]);

            qstr += ")";

        }

        //один корень

        if (count == 1) {

            qstr += "p(x)=";

            qstr << p.a;

            qstr += "(x";

            qstr += (roots[0] >= 0 ? "-" : "+");

            qstr << abs(roots[0]);

            qstr += ")^2";

        }

    }

    return qstr;

}

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) {

            return 2;

        }

        else

        {

            return 0;

        }

    }

    //при Д=0

    if (d == 0)

    {

        roots[0] = ((-b) / (a \* 2));

        if (a \* roots[0] \* roots[0] + b \* roots[0] + c == 0) {

            return 1;

        }

        else {

            return 0;

        }

    }

    //при Д<0

    else    {

        return 0;

    }

};

QString Polinom::Calculate(Polinom& p)

{

    QString qstr("");

    number roots[2];

    int count = p.Calculate(roots);

    if (count == 0) {

        qstr += "Полином не разложим над\nполем рациональных";

    }

    else if (count == 1) {

        qstr += "Корень равен ";

        qstr << roots[0];

    }

    else if (count == 2) {

        qstr += "Первый корень равен ";

        qstr << roots[0];

        qstr += "\nВторой корень равен ";

        qstr << roots[1];

    }

    return qstr;

}

FILE main.cpp

#include "interface.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

    QApplication a(argc, argv);

    Interface w;

    w.show();

    return a.exec();

}

FILE LAB\_3.pro

QT += core gui

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

CONFIG += c++11

# You can make your code fail to compile if it uses deprecated APIs.

# In order to do so, uncomment the following line.

#DEFINES += QT\_DISABLE\_DEPRECATED\_BEFORE=0x060000 # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \

main.cpp \

interface.cpp \

polinom.cpp \

rational.cpp

HEADERS += \

interface.h \

number.h \

polinom.h \

rational.h

# Default rules for deployment.

qnx: target.path = /tmp/$${TARGET}/bin

else: unix:!android: target.path = /opt/$${TARGET}/bin

!isEmpty(target.path): INSTALLS += target