

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

ОТЧЕТ
по практической работе №4
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

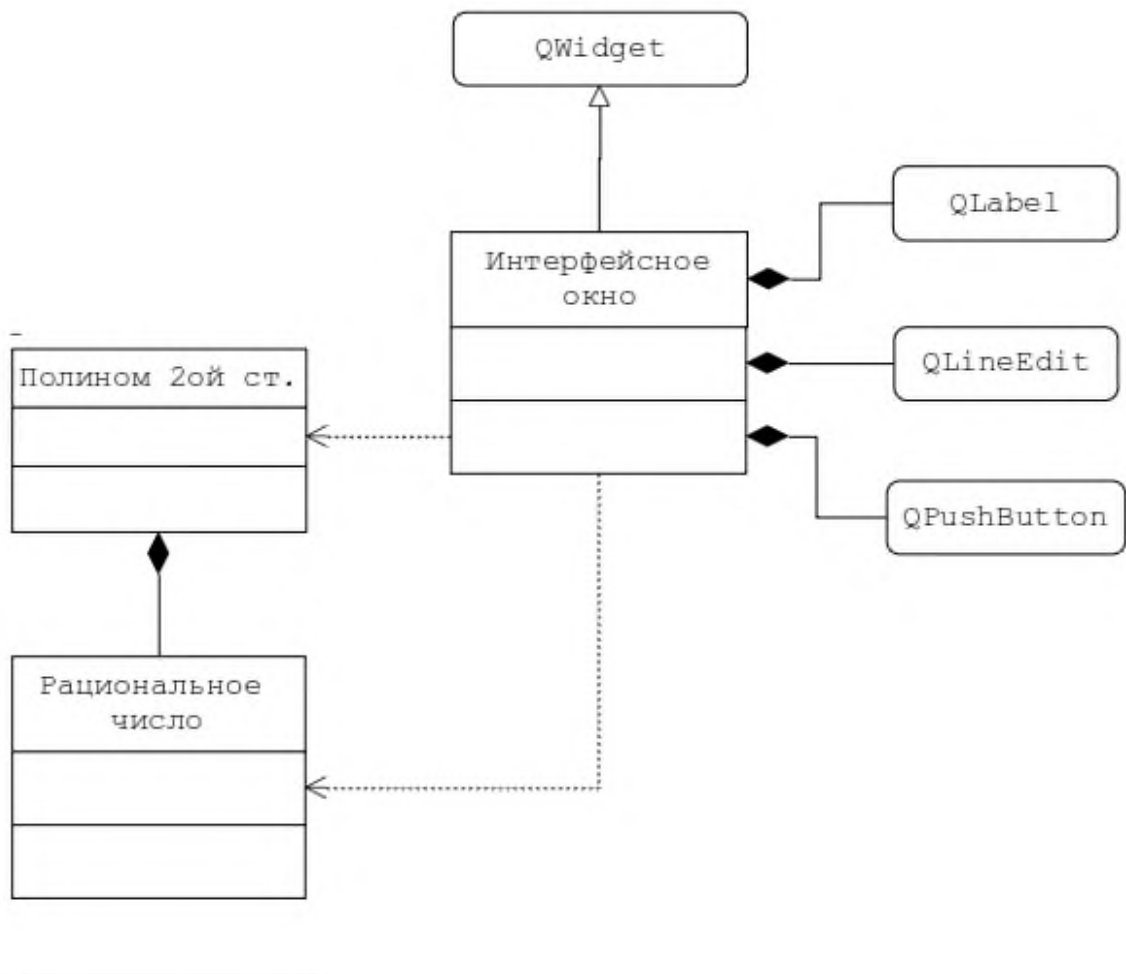


Рис.1. Диаграмма классов работы №4

Создать два приложения (клиент и сервер), работающих в паре.

Клиентское приложение (GUI приложение) должно выполнять следующие функции:

- ввод/вывод через интерфейсные элементы объекта класса интерфейс;
- формирование сообщения, состоящего из вида запроса (что от них надо) и параметров запроса (какие данные для этого есть);
- получать ответ на запрос не напрямую, а через ответное сообщение;
- транслировать сообщения объекта интерфейс в сеть и обратно.

Серверное приложение (консольное приложение) должно выполнять следующие функции:

- получать и принимать сообщения из сети (коммуникатор);
- преобразовывать полученные сообщения к виду, позволяющему обращаться к объектам полинома и рационального числа;
- напрямую обращаться с запросами к объектам полинома и рационального числа, что в едином приложении выполнялось в реализации методов объекта интерфейса;
- получать ответ от полинома;
- преобразовывать его в ответное сообщение для интерфейса клиентского приложения.

Клиент и сервер настроены на исполнение на одном компьютере по IP адресу 127.0.0.1 с перекрестным соединением портов 10000 → 10001 и 10001 ← 10000.

Реализовать и отладить программы, удовлетворяющие сформулированным требованиям и заявленным целям. Разработать контрольные примеры и протестировать на них программы. Оформить отчет, сделать выводы по работе.

Спецификации классов

Общие ресурсы

enum messages – перечисление передаваемых сообщений между программами клиент-сервер.

Класс TCommunicator

Атрибуты:

Тип	Наименование	Область видимости
bool	ready	private
struct TCommParams { QHostAddress rHost; quint16 rPort; QHostAddress sHost; quint16 sPort; };	params	private

Класс TCommunicator реализует сетевую коммуникацию между приложениями клиент и сервер.

Программа Server, 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 **CalculateRoots()** 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**.

Программа Server, 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

Программа Server, 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.

Программа Server, Класс TApplication

Атрибуты:

Тип	Наименование	Область видимости
TCommunicator*	comm	private

Класс TApplication является основным классом, управляющим серверную часть приложения, реализует получение запросов со стороны клиентского приложения, обработку необходимых данных, выполнение необходимых расчетов, формирование и отправка ответных сообщений.

Программа Client, Класс Interface

Атрибуты:

Тип	Наименование	Область видимости
QLabel*	name_a,	private
QLabel*	delimiter_a	private
QLineEdit*	a_numerator,	private
QLineEdit*	a_denominator	private
QLabel*	name_b,	private
QLabel*	delimiter_b	private
QLineEdit*	b_numerator,	private
QLineEdit*	b_denominator	private
QLabel*	name_c,	private
QLabel*	delimiter_c	private
QLineEdit*	c_numerator,	private
QLineEdit*	c_denominator	private

QLabel*	name_x,	private
QLabel*	delimiter_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 интерфейс программы.

Атрибуты delimiter_* используются для хранения текстовых меток, знаков разделителей, выводимых на графический интерфейс 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.

Программа Client, Класс TApplication

Атрибуты:

Тип	Наименование	Область видимости
TCommunicator*	comm	private
TInterface*	interface	private

Класс TApplication является основным классом, управляющим клиентскую часть приложения, реализует передачу и получение запросов с коммуникатора и в коммуникатор.

Описание контрольного примера с исходными и ожидаемыми (расчетными) данными

Пример: $P(x) = \frac{2}{5}x^2 + \frac{4}{5}x + \frac{112}{405}$ $x_0 = \frac{7}{8}$

$$a = \frac{2}{5}, \quad b = \frac{4}{5}, \quad c = \frac{112}{405}, \quad x = \frac{7}{8}$$

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

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

$$x_1 = \frac{-\frac{4}{5} + \sqrt{\frac{16}{81}}}{2 * \frac{2}{5}} = -\frac{4}{9} \quad x_2 = \frac{-\frac{4}{5} - \sqrt{\frac{16}{81}}}{2 * \frac{2}{5}} = -\frac{14}{9}$$

$$P\left(\frac{7}{8}\right) = \frac{3325}{2592}$$

Скриншоты программы на контрольных примерах

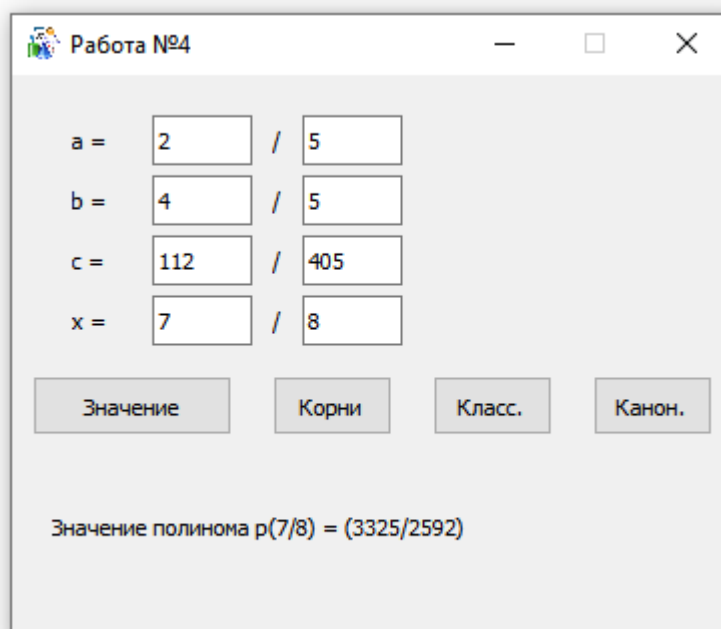


Рисунок 1. Ввод значения аргумента x, расчет значения и его вывод

Работа №4

a = /

b = /

c = /

x = /

Значение **Корни** Класс. Канон.

Первый корень равен (-4/9)
Второй корень равен (-14/9);

Рисунок 2. Расчет корней и вывод результатов расчета

Работа №4

a = /

b = /

c = /

x = /

Значение Корни **Класс.** Канон.

$p(x) = (2/5)x^2 + (4/5)x + (112/405);$

Рисунок 3. Вывод текстового представления полинома в классической форме

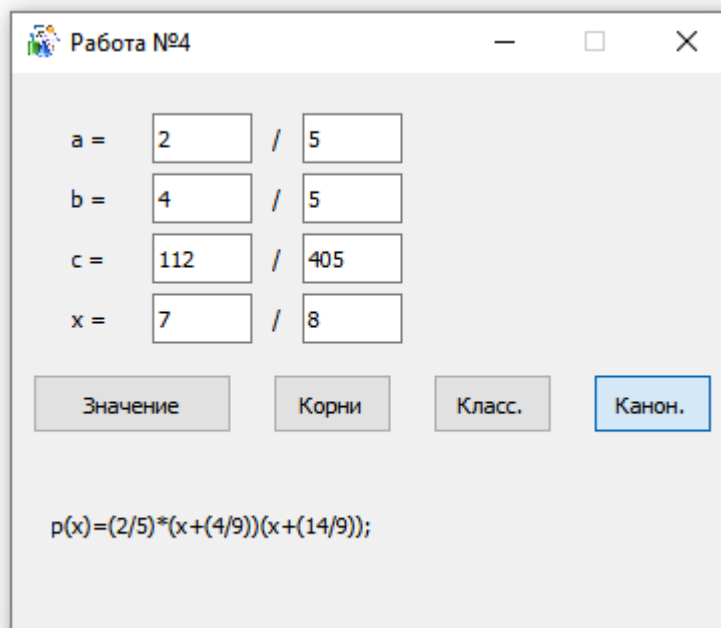


Рисунок 4. Вывод текстового представления полинома в канонической форме

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 протестирована на них.

Была реализована пара приложений (клиент-сервер). Все расчеты выполняются в серверной части приложения, а клиентская часть служит только для получения данных от пользователя и вывода итогового результата расчетов.

Получил практические навыки работы со средой разработки Qt Creator, с сетевым протоколом UDP и работы пары приложений через сеть.

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 (ОБЩИЙ)

FILE common/common.h

```
#ifndef COMMON_H
#define COMMON_H

#include <QString>

enum messages
{
    VALUE_REQUEST = 1,
    VALUE_ANSWER,
    PRINT_CLASSIC_REQUEST,
    PRINT_CANONIC_REQUEST,
    PRINT_ANSWER,
    ROOTS_REQUEST,
    ROOTS_ANSWER,
};

extern const QChar separator;
QString& operator<< (QString&, const QString&);

#endif // COMMON_H
```

FILE common/common.cpp

```
#include "common.h"

const QChar separator(';');

QString& operator<< (QString& m, const QString& s)
{
    m += s;
    m.append(separator);
    return m;
}
```

FILE common/communicator.h

```
#ifndef COMMUNICATOR_H
#define COMMUNICATOR_H

#include <QObject>
#include <QUdpSocket>
```

```

#include <QHostAddress>

struct TCommParams
{
    QHostAddress rHost;
    quint16      rPort;
    QHostAddress sHost;
    quint16      sPort;
};

class TCommunicator : public QUdpSocket
{
    Q_OBJECT

    bool      ready;
    TCommParams params;

public:
    TCommunicator(TCommParams&,QObject *parent = nullptr);
    bool isReady();

signals:
    void recieved(QByteArray);

public slots:
    void send(QByteArray);

private slots:
    void recieve();

};

#endif // COMMUNICATOR_H

```

FILE common/ communicator.cpp

```

#include "communicator.h"

TCommunicator::TCommunicator(TCommParams& pars, QObject *parent) : QUdpSo
cket(parent)
{
    params = pars;
    ready = bind(params.rHost, params.rPort,
                  QAbstractSocket::ShareAddress | QAbstractSocket::ReuseAd
dressHint);
    if (ready) connect(this,SIGNAL(readyRead()),this,SLOT(recieve()));
}

bool TCommunicator::isReady()
{

```

```

        return ready;
    }

    void TCommunicator::send(QByteArray msg)
    {
        if (ready)
        {
            writeDatagram(msg, params.sHost, params.sPort);
            //qDebug()<<"sended"<<msg;
        }
    }

    void TCommunicator::recieve()
    {
        if (hasPendingDatagrams())
        {
            quint64 size = pendingDatagramSize();
            QByteArray msg(size, '\0');
            readDatagram(msg.data(), size);
            //qDebug()<<"recieved"<<msg;
            emit recieved(msg);
        }
    }

```

ПРИЛОЖЕНИЕ 2. ИСХОДНЫЙ КОД ПРОГРАММЫ CLIENT

FILE client/client.pro

```
QT      += core gui network
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = client
TEMPLATE = app
DEFINES += QT_DEPRECATED_WARNINGS

#RC_ICONS += math.ico

INCLUDEPATH += ../common
SOURCES += \
    main.cpp \
    application.cpp \
    interface.cpp \
    ../common/communicator.cpp \
    ../common/common.cpp

HEADERS += \
    application.h \
    interface.h \
    ../common/communicator.h \
    ../common/common.h
```

FILE client/application.h

```
#ifndef APPLICATION_H
#define APPLICATION_H

#include <QObject>
#include <QApplication>

#include "interface.h"
#include "communicator.h"

class TApplication : public QApplication
{
```

```

Q_OBJECT

TCommunicator *comm;
TInterface     *interface;

public:
    TApplication(int, char**);

public slots:
    void fromCommunicator(QByteArray);
    void toCommunicator(QString);

};

#endif // APPLICATION_H

```

FILE client/application.cpp

```

#include "application.h"

TApplication::TApplication(int argc, char *argv[])
    : QApplication(argc,argv)
{
    TCommParams pars = { QHostAddress("127.0.0.1"), 10001,
                        QHostAddress("127.0.0.1"), 10000};
    comm = new TCommunicator(pars, this);

    interface = new TInterface();
    interface->show();

    connect(comm,SIGNAL(recieved(QByteArray)),this, SLOT(fromCommunicator(QByteArra
y)));
    connect(interface,SIGNAL(request(QString)),this, SLOT(toCommunicator(QString)))
;

}

void TApplication::fromCommunicator(QByteArray msg)
{
    interface->answer(QString(msg));
}

void TApplication::toCommunicator(QString msg)
{
    comm->send(QByteArray().append(msg));
}

```


FILE client/interface.h

```
#ifndef INTERFACE_H
#define INTERFACE_H

#include <QWidget>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>

class TInterface : public QWidget
{
    Q_OBJECT

    QLabel *name_a, *delimiter_a;
    QLineEdit *a_numerator, *a_denominator;
    QLabel *name_b, *delimiter_b;
    QLineEdit *b_numerator, *b_denominator;
    QLabel *name_c, *delimiter_c;
    QLineEdit *c_numerator, *c_denominator;
    QLabel *name_x, *delimiter_x;
    QLineEdit *x_numerator, *x_denominator;

    QPushButton *value_btn;
    QPushButton *root_btn;
    QPushButton *print_classic_btn;
    QPushButton *print_canonic_btn;

    QLabel *output;

public:
    TInterface(QWidget *parent = 0);
    ~TInterface();

public slots:
    void answer(QString);

private slots:
    void formRequest();

signals:
    void request(QString);

};

#endif // INTERFACE_H
```

FILE client/interface.cpp

```
#include "interface.h"
#include "common.h"

TInterface::TInterface(QWidget *parent) : QWidget(parent)
{
    setWindowTitle("Работа №4");
    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);
    delimiter_a = new QLabel("/", this);
    delimiter_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);
    delimiter_b = new QLabel("/", this);
    delimiter_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);
    delimiter_c = new QLabel("/", this);
    delimiter_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);
    delimiter_x = new QLabel("/", this);
    delimiter_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(formRequest()));
connect(root_btn,SIGNAL(pressed()),this,SLOT(formRequest()));
connect(print_classic_btn,SIGNAL(pressed()),this,SLOT(formRequest()));
connect(print_canonic_btn,SIGNAL(pressed()),this,SLOT(formRequest()));
}

TInterface::~TInterface()
{
    delete name_a;
    delete delimiter_a;
    delete a_numerator;
    delete a_denominator;

    delete name_b;
    delete delimiter_b;
    delete b_numerator;
    delete b_denominator;

    delete name_c;
    delete delimiter_c;
    delete c_numerator;
    delete c_denominator;

    delete name_x;
    delete delimiter_x;
    delete x_numerator;
    delete x_denominator;

    delete value_btn;
    delete root_btn;
    delete print_classic_btn;
    delete print_canonic_btn;

    delete output;
}

void TInterface::formRequest()
{
    QString msg;
    msg << a_numerator->text() << a_denominator->text();
    msg << b_numerator->text() << b_denominator->text();
    msg << c_numerator->text() << c_denominator->text();

```

```

QPushButton *btn = (QPushButton*)sender();
if (btn == value_btn)
{
    msg << QString().setNum(VALUE_REQUEST);
    msg << x_numerator->text() << x_denominator->text();
}
if (btn == root_btn)
{
    msg << QString().setNum(ROOTS_REQUEST);
}
if (btn == print_classic_btn)
{
    msg << QString().setNum(PRINT_CLASSIC_REQUEST);
}
if (btn == print_canonic_btn)
{
    msg << QString().setNum(PRINT_CANONIC_REQUEST);
}
emit request(msg);
}

void TInterface::answer(QString msg)
{
    QString text;
    int p = msg.indexOf(separator);
    int t = msg.left(p).toInt();
    msg = msg.mid(p+1, msg.length()-p-2);
    switch (t)
    {
        case VALUE_ANSWER:
            text = "Значение полинома p";
            p = msg.indexOf(separator);
            text += msg.left(p);
            text += " = ";
            text += msg.right(msg.length()-p-1);
            output->setText(text);
            break;
        case ROOTS_ANSWER:
            text = "";
            text<<msg;
            output->setText(text);
            break;
        case PRINT_ANSWER:
            text = "";
            text<<msg;
            output->setText(text);
            break;
        default: break;
    }
}
}

```

FILE client/main.cpp

```
#include "application.h"

int main(int argc, char *argv[])
{
    TApplication a(argc, argv);
    return a.exec();
}
```

ПРИЛОЖЕНИЕ 3. ИСХОДНЫЙ КОД ПРОГРАММЫ SERVER

FILE server/server.pro

```
QT -= gui
QT += network

CONFIG += c++11 console
CONFIG -= app_bundle

DEFINES += QT_DEPRECATED_WARNINGS

INCLUDEPATH += ../common

SOURCES += main.cpp \
    application.cpp \
    polinom.cpp \
    ../common/communicator.cpp \
    ../common/common.cpp \
    rational.cpp

HEADERS += \
    application.h \
    polinom.h \
    number.h \
    ../common/communicator.h \
    ../common/common.h \
    rational.h
```

FILE server/number.h

```
#ifndef NUMBER_H
#define NUMBER_H

#include "rational.h"

typedef Rational number;

#endif // NUMBER_H
```

FILE server/rational.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 CalculateRoots();
    friend QString& operator<< (QString&, Polinom&);
};

#endif // POLINOM_H

```

FILE server/rational.cpp

```

#include "rational.h"
#include "common.h"

QChar Rational::SEPARATOR = separator;

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;
}

Rational sqrt(Rational rt)
{
    double a, b;
    a = ::sqrt(rt.a);
    b = ::sqrt(rt.b);
    // преобразовываем double a и b в int, далее снова в double
    // если преобразованный совпадает с исходным, значит число целое
    // например имеем рациональное 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 атрибуты объ-
екта (int)
    // заметим, что если бы a получилось скажем 4.254... то
    // при сравнении (((double)(int)4.254) == 4.254) полу-
чаем 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 дискрими-
нанта 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::operator QString ()
{
    QString s = "(";
    s += QString().setNum(a);
    s += "/";
    s += QString().setNum(b);
    s += ")";
    return s;
}

Rational::Rational(const QByteArray& arr)
{
    int p = arr.indexOf(SEPARATOR);
    a = arr.left(p).toInt();
    b = arr.right(arr.length()-p-1).toUInt();
}

QString& operator<<(QString& qstr, Rational rt)
{
    //вызвать функцию нахождения НОД-а
    //проверить можно ли сократить дробь
    //если да, то сократить, 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;
}

QByteArray& operator>>(QByteArray& arr, Rational& c)
{
    int p = arr.indexOf(Rational::SEPARATOR);
    p = arr.indexOf(Rational::SEPARATOR,p+1);
    if (p > 0)
    {
        c = Rational(arr.left(p));
        arr = arr.right(arr.length()-p-1);
    }
}

```

```

    }
    return arr;
}

void Rational::setSeparator(QChar ch)
{
    SEPARATOR = ch;
}

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 отрицатель-
ным, 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 << "Введенные числа НЕ образуют рациональ-
ную 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 сократить, 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;
    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;
    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;
    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;
    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 server/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 CalculateRoots();
    friend QString& operator<< (QString&, Polinom&);
};

#endif // POLINOM_H

```

FILE server/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 += ")";
            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 d>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 D=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 D<0
    else {
        return 0;
    }
};

QString Polinom::CalculateRoots()
{
    QString qstr("");
    number roots[2];
    int count = Calculate(roots);

    if (count == 0) {
        qstr += "Полином не разложим над\нполем 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 server/application.h

```
#ifndef APPLICATION_H
#define APPLICATION_H

#include <QObject>
#include <QCoreApplication>

#include "communicator.h"

class TApplication : public QCoreApplication
{
    Q_OBJECT

    TCommunicator *comm;

public:

    TApplication(int, char**);

signals:

public slots:

    void recieve(QByteArray);

};

#endif // APPLICATION_H
```

FILE server/application.cpp

```
#include "application.h"
#include "polinom.h"
#include "common.h"

TApplication::TApplication(int argc, char *argv[])
    : QCoreApplication(argc,argv)
{
    TCommParams pars = { QHostAddress("127.0.0.1"), 10000,
                        QHostAddress("127.0.0.1"), 10001};
    comm = new TCommunicator(pars, this);

    connect(comm,SIGNAL(recieved(QByteArray)),this,SLOT(recieve(QByteArra
y)));
}

void TApplication::recieve(QByteArray msg)
{
    QString answer, s;
```

```

Rational a, b, c, x, v;
msg>>a>>b>>c;
Polinom p(a,b,c);
int pos = msg.indexOf(separator);
int t = msg.left(pos).toInt();
switch (t)
{
    case VALUE_REQUEST:
        msg = msg.right(msg.length()-pos-1);
        msg>>x;
        v = p.value(x);
        s<<(QString)x<<(QString)v;
        answer<<QString().setNum(VALUE_ANSWER);
        answer += s;
        break;
    case ROOTS_REQUEST:
        s += p.CalculateRoots();
        answer<<QString().setNum(ROOTS_ANSWER)<<s;
        break;
    case PRINT_CLASSIC_REQUEST:
        p.setPrintMode(EPrintModeClassic);
        s<<p;
        answer<<QString().setNum(PRINT_ANSWER)<<s;
        break;
    case PRINT_CANONIC_REQUEST:
        p.setPrintMode(EprintModeCanonical);
        s<<p;
        answer<<QString().setNum(PRINT_ANSWER)<<s;
        break;
    default: return;
}
comm->send(QByteArray().append(answer));
}

```

FILE server/main.cpp

```

#include "application.h"

int main(int argc, char *argv[])
{
    TApplication a(argc, argv);
    return a.exec();
}

```