Часть 1. Калькулятор

Цель работы: изучить основные принципы разработки программ с графическим интерфейсом с использованием Qt.

Выполнение: сначала была создана форма в Qt designer (рисунок 1)

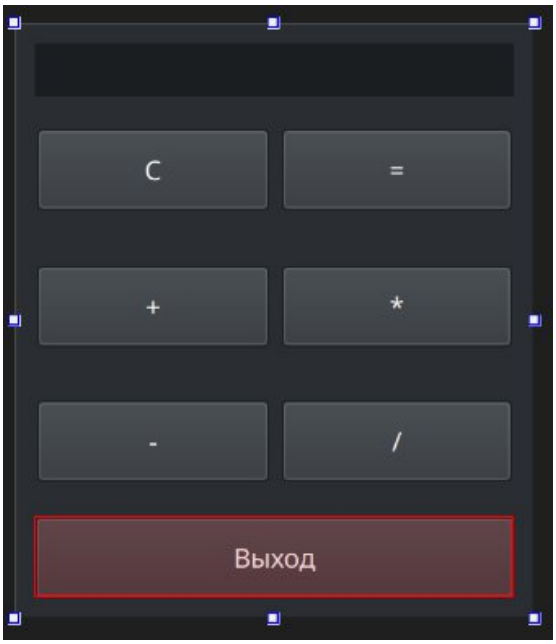

Рисунок 1 — Форма в qt creator

Затем, были созданы файлы mainwin.h и mainwin.cpp в Qt Creator (рисунки 2-3).

```h
#ifndef MAINWIN_H
#define MAINWIN_H

#include <QMainWindow>

QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE

class MainWindow : public QMainWindow {
  Q_OBJECT

public:
  MainWindow(QWidget *parent = nullptr);
  ~MainWindow();

private slots:
  void onActionClicked(char action);

  void on_minusButton_clicked();

  void on_clearButton_clicked();

  void on_equalButton_clicked();

  void on_plusButton_clicked();

  void on_multiButton_clicked();

  void on_divButton_clicked();

  void on_exitButton_clicked();

private:
  Ui::MainWindow *ui;
  double res;
  QChar oper;
};
#endif // MAINWIN_H
```

Рисунок 2 — Файл mainwin.h

```cpp
#include "mainwin.h"
#include "ui_mainwin.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent), ui(new Ui::MainWindow), res(0), oper('@') {
  ui->setupUi(this);
  ui->lineEdit->setFocus();
}

MainWindow::~MainWindow() { delete ui; }

void MainWindow::on_clearButton_clicked() {
  ui->lineEdit->clear();
  ui->lineEdit->setFocus();
  res = 0;
  oper = '@';
}

void MainWindow::on_equalButton_clicked() {
  double r = ui->lineEdit->text().toDouble();
  if (oper == '+')
    res += r;
  else if (oper == '-')
    res -= r;
  else if (oper == '*')
    res *= r;
  else if (oper == '/') {
    if (r == 0)
      res = 0;
    else
      res /= r;
  }
  QString q_str;
  ui->lineEdit->setText(q_str.setNum(res));
  ui->lineEdit->setFocus();
}

void MainWindow::onActionClicked(char action) {
  oper = action;
  res = ui->lineEdit->text().toDouble();
  ui->lineEdit->clear();
  ui->lineEdit->setFocus();
}

void MainWindow::on_plusButton_clicked() { onActionClicked('+'); }

void MainWindow::on_minusButton_clicked() { onActionClicked('-'); }

void MainWindow::on_multiButton_clicked() { onActionClicked('*'); }

void MainWindow::on_divButton_clicked() { onActionClicked('/'); }

void MainWindow::on_exitButton_clicked() { this->close(); }
```

Рисунок 3 — Файл mainwin.cpp

Получившийся результат изображён на рисунке 4.

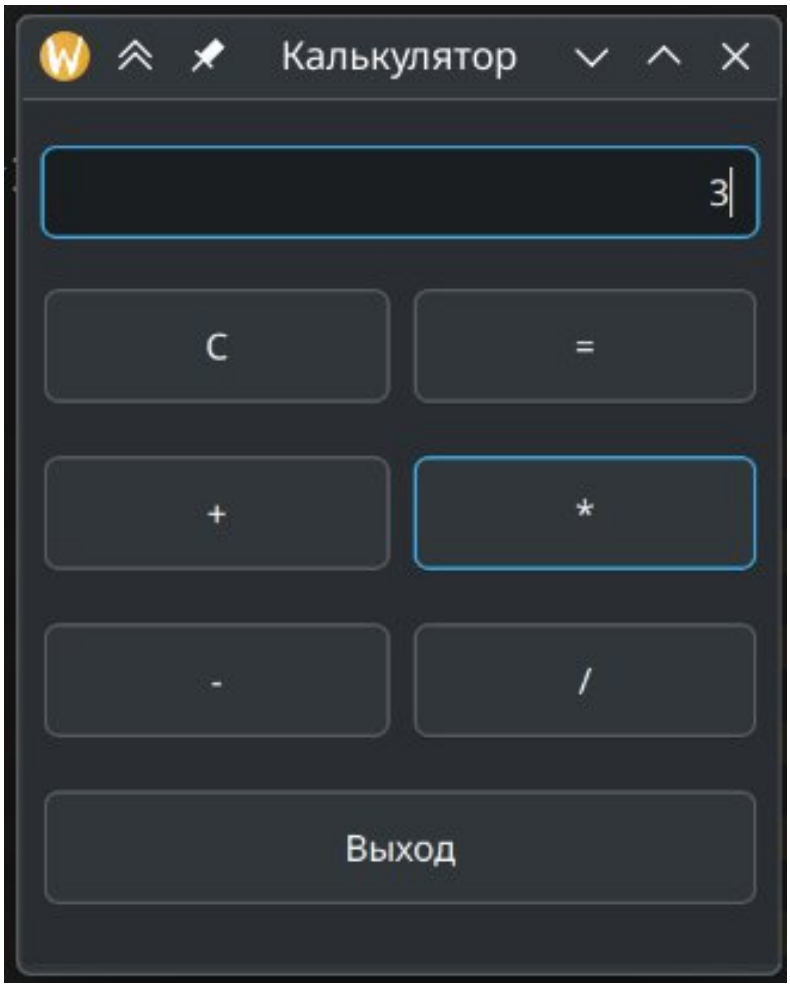

Рисунок 4 — Окно приложения

Вывод: Был разработан и протестирован простейший калькулятор с использованием фраемворка Qt и языка программирования C++.

Часть 2. Записная книжка

Цель работы: Разработать программу на языке программирования C++, используя фраемворк Qt. Реализовать такие функции, как поиск и удаление записей.

Ход работы: сначала была создана форма главного окна приложения и диалога (Рисунки 5-6).

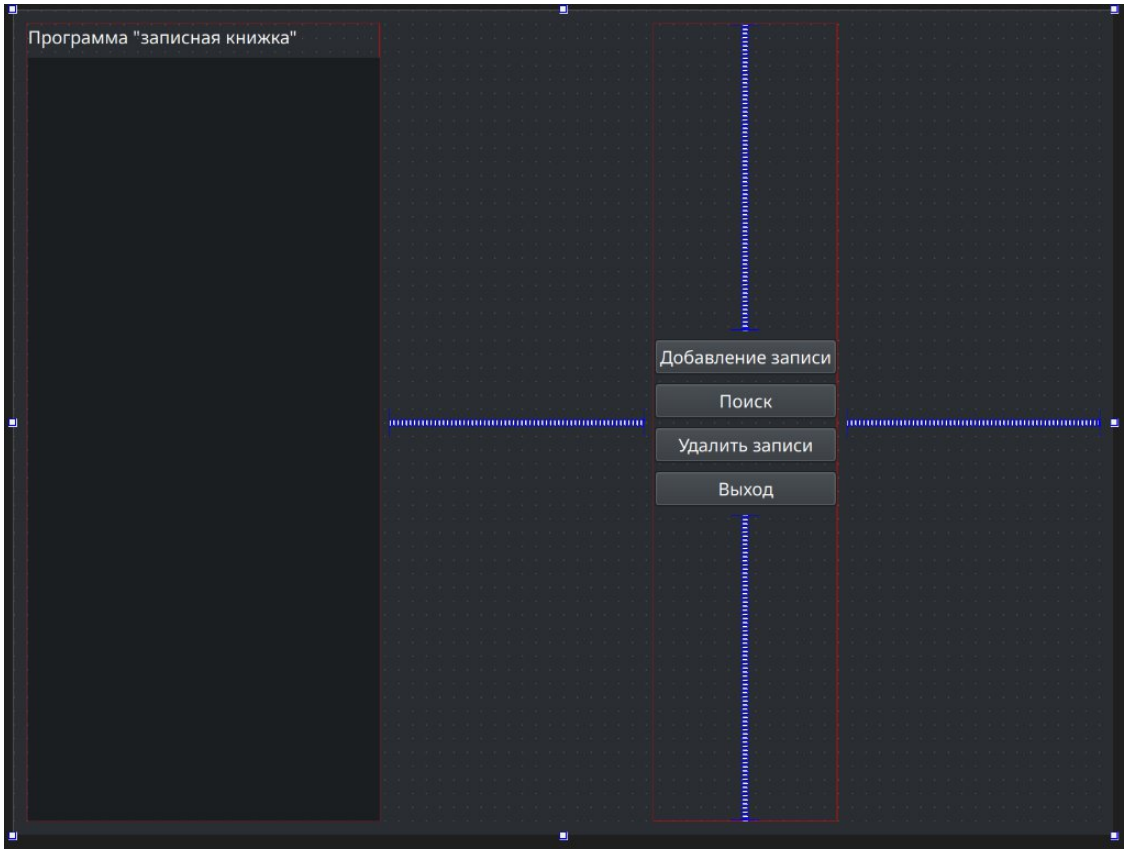

Рисунок 5 — Форма основного окна приложения

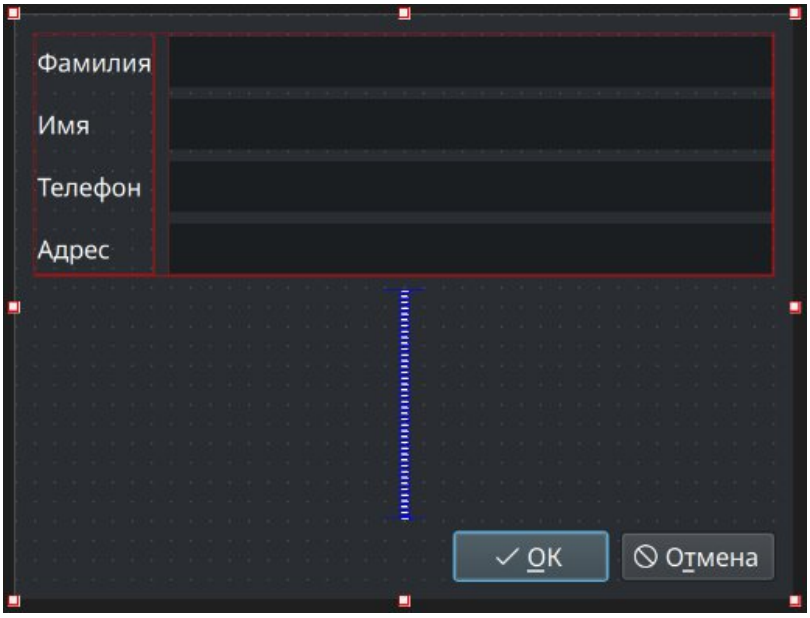

Рисунок 6 — Форма диалогового окна

Было принято решение создать структуру field для упрощения работы (рисунок 7)

```
struct field {
  QString last_name;
  QString first_name;
  QString phone_number;
  QString address;

  bool contains(field &other) {
    return last_name.contains(other.last_name) &&
           first_name.contains(other.first_name) &&
           phone_number.contains(other.phone_number) &&
           address.contains(other.address);
  }
};
```

Рисунок 7 — Описание структуры field

Затем была создана реализация все необходимых классов (рисунки 8-10)

```
#define ADD 1
#define SEARCH 2
#define DELETE 3

namespace Ui {
class StructDialog;
}
class StructDialog : public QDialog {
  Q_OBJECT;
  int dialogType;

public:
  explicit StructDialog(QWidget *parent = nullptr, int type = ADD);
  ~StructDialog();
  void setStruct(field &s);
  field getStruct();

private:
  Ui::StructDialog *ui;
  void accept() override;
};

#endif // STRUCT_DIALOG_H
```

Рисунок 8 — Заголовочный файл struct_dialog.h

```
StructDialog::StructDialog(QWidget *parent, int type)
    : QDialog(parent), ui(new Ui::StructDialog) {
  ui->setupUi(this);
  dialogType = type;
  if (dialogType == SEARCH)
    setWindowTitle("Поиск записей");
  else if (dialogType == DELETE) {
    setWindowTitle("Удаление записей");
  }
}

StructDialog::~StructDialog() { delete ui; }

void StructDialog::setStruct(field &s) {
  ui->edtLastName->setText(s.last_name);
  ui->edtFirstName->setText(s.first_name);
  ui->edtPhone->setText(s.phone_number);
  ui->edtAddress->setText(s.address);
}

field StructDialog::getStruct() {
  return {ui->edtLastName->text(), ui->edtFirstName->text(),
          ui->edtPhone->text(), ui->edtAddress->text()};
}

void StructDialog::accept() {
  auto lineEdits = {ui->edtLastName, ui->edtFirstName, ui->edtPhone,
                    ui->edtAddress};
  if (dialogType == ADD) {
    for (auto edt : lineEdits) {
      if (edt->text().isEmpty()) {
        edt->setFocus();
        return;
      }
    }
  } else if (dialogType == DELETE) {
    for (auto edt : lineEdits) {
      if (!edt->text().isEmpty()) {
        QDialog::accept();
        return;
      } else {
        ui->edtLastName->setFocus();
      }
    }
  }
  QDialog::accept();
}
```

Рисунок 9 — Реализация диалога

```
void MainWindow::on_btnAdd_clicked() {
  StructDialog addDialog{this, ADD};
  if (addDialog.exec() == QDialog::Accepted) {
    data.push_back(addDialog.getStruct());
    updateList();
  }
}

void MainWindow::on_btnSearch_clicked() {
  StructDialog searchDialog{this, SEARCH};
  searchDialog.setStruct(searchBy);
  if (searchDialog.exec() == QDialog::Accepted) {
    searchBy = searchDialog.getStruct();
    updateList();
  }
}

void MainWindow::on_btnDelete_clicked() {
  StructDialog deleteDialog(this, DELETE);
  if (deleteDialog.exec() == QDialog::Accepted) {
    auto toDelete = deleteDialog.getStruct();
    auto i = data.begin();
    while (i != data.end()) {
      if ((*i).contains(toDelete))
        i = data.erase(i);
      else
        i++;
    }
    updateList();
  }
}
void MainWindow::on_btnExit_clicked() { this->close(); }
```

Рисунок 10 — Обработка нажатий по кнопкам основной формы

Запуск и тестирование: Для тестирования были добалены два контакта (по умолчанию). Ход тестирования изображен на рисунках 11-15

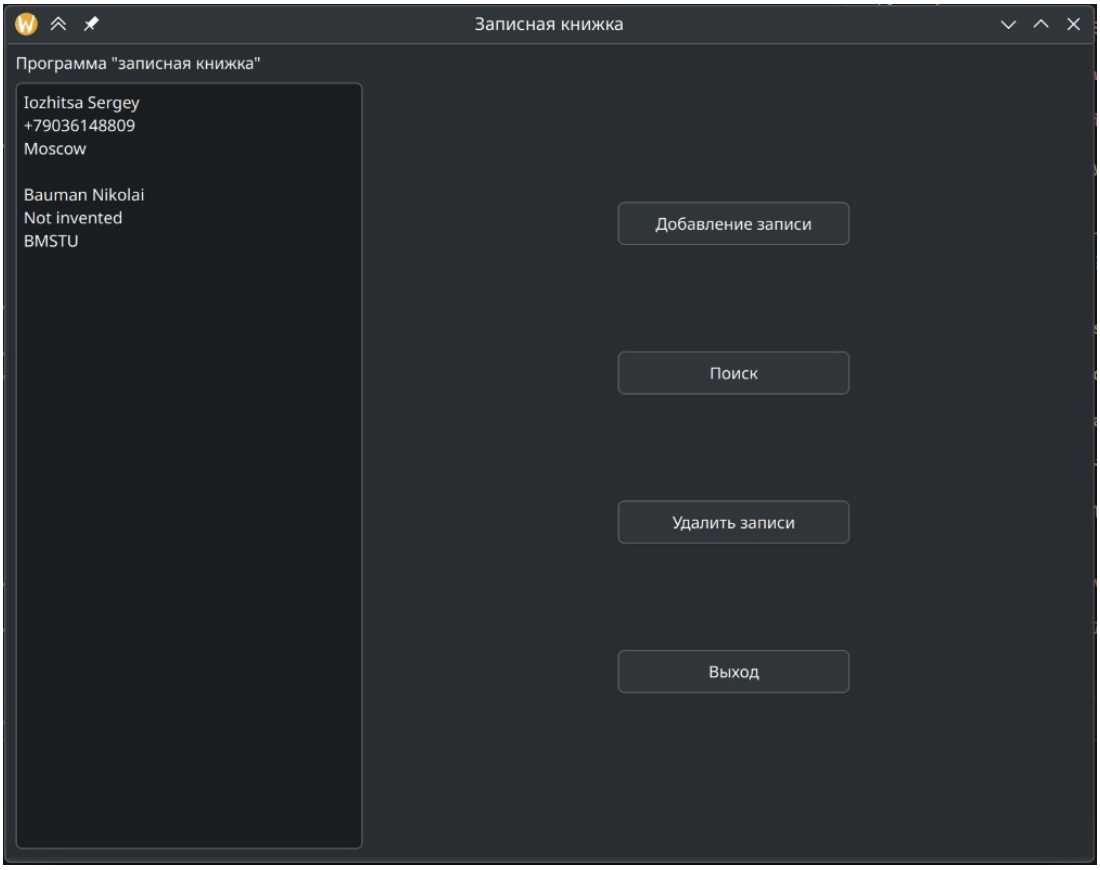

Рисунок 11 — Основное окно приложения

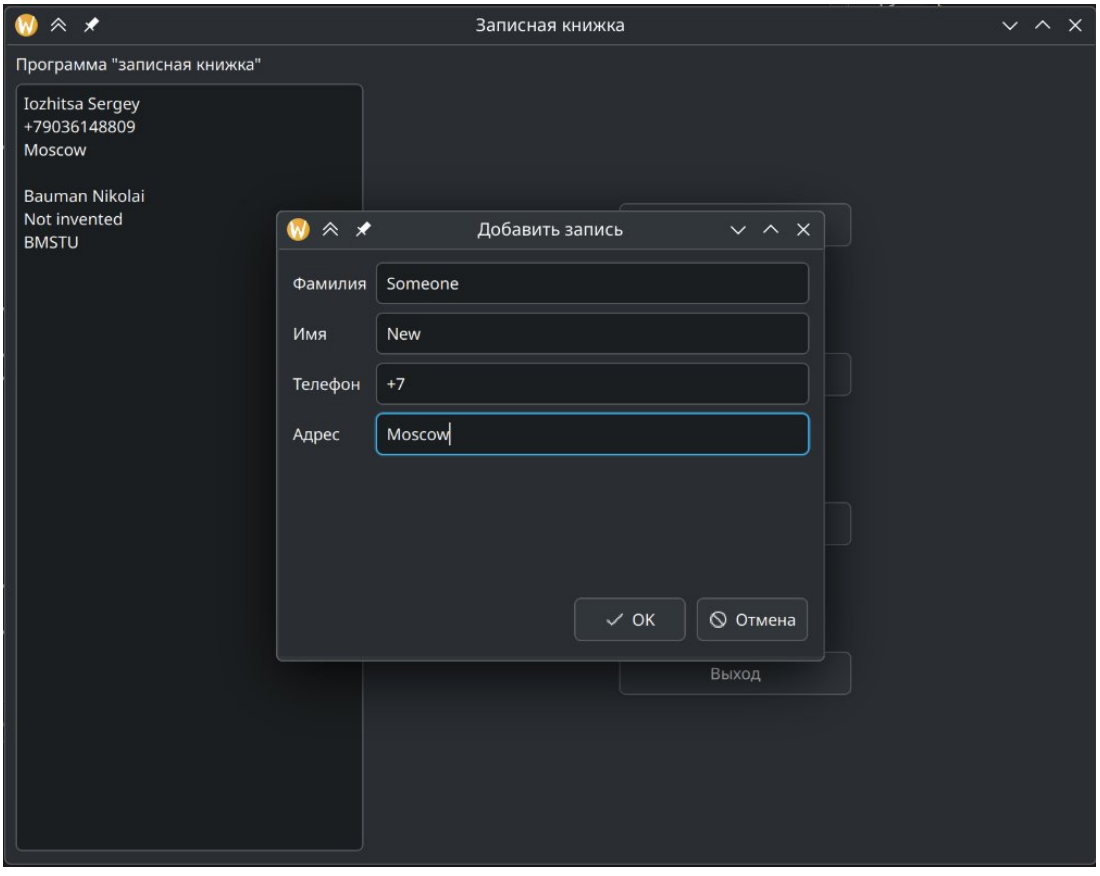

Рисунок 12 — Добавление нового контакта

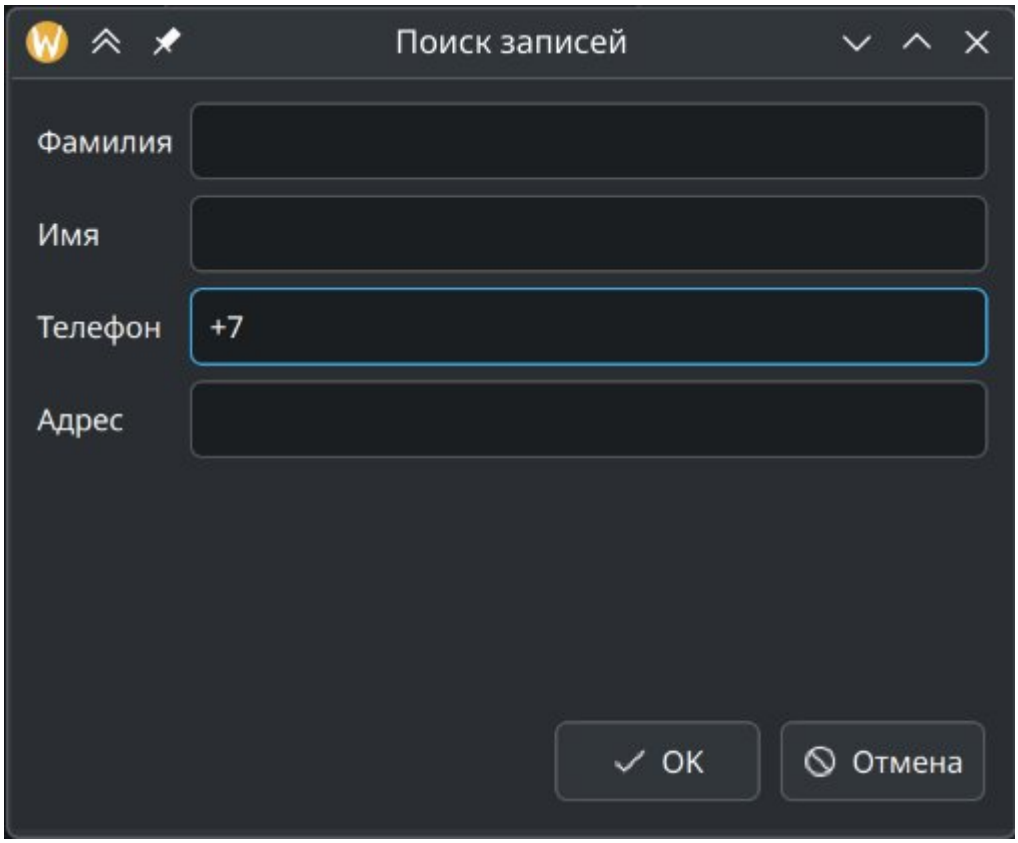

Рисунок 13 — Диалог поиска контактов

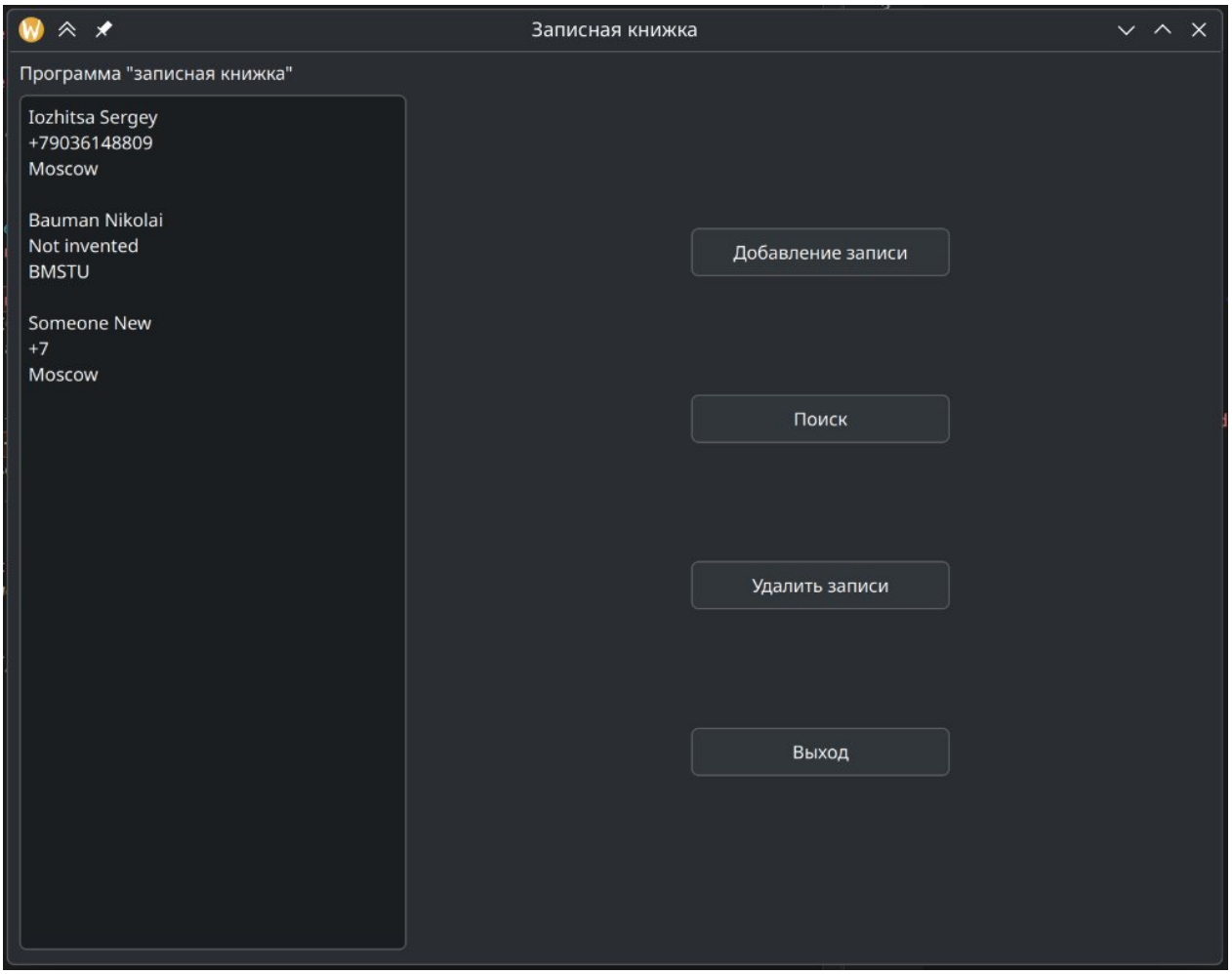

Рисунок 14 — Главное окно после добавления контакта

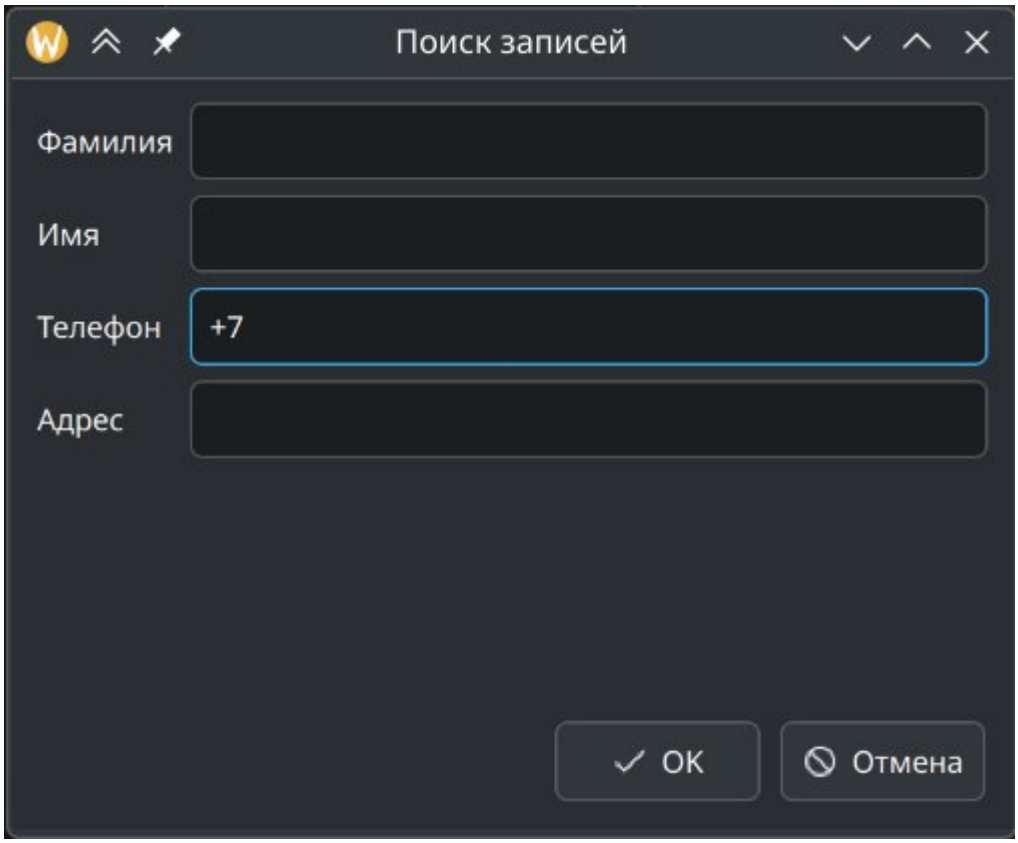

Рисунок 15 — Диалог поиска контактов

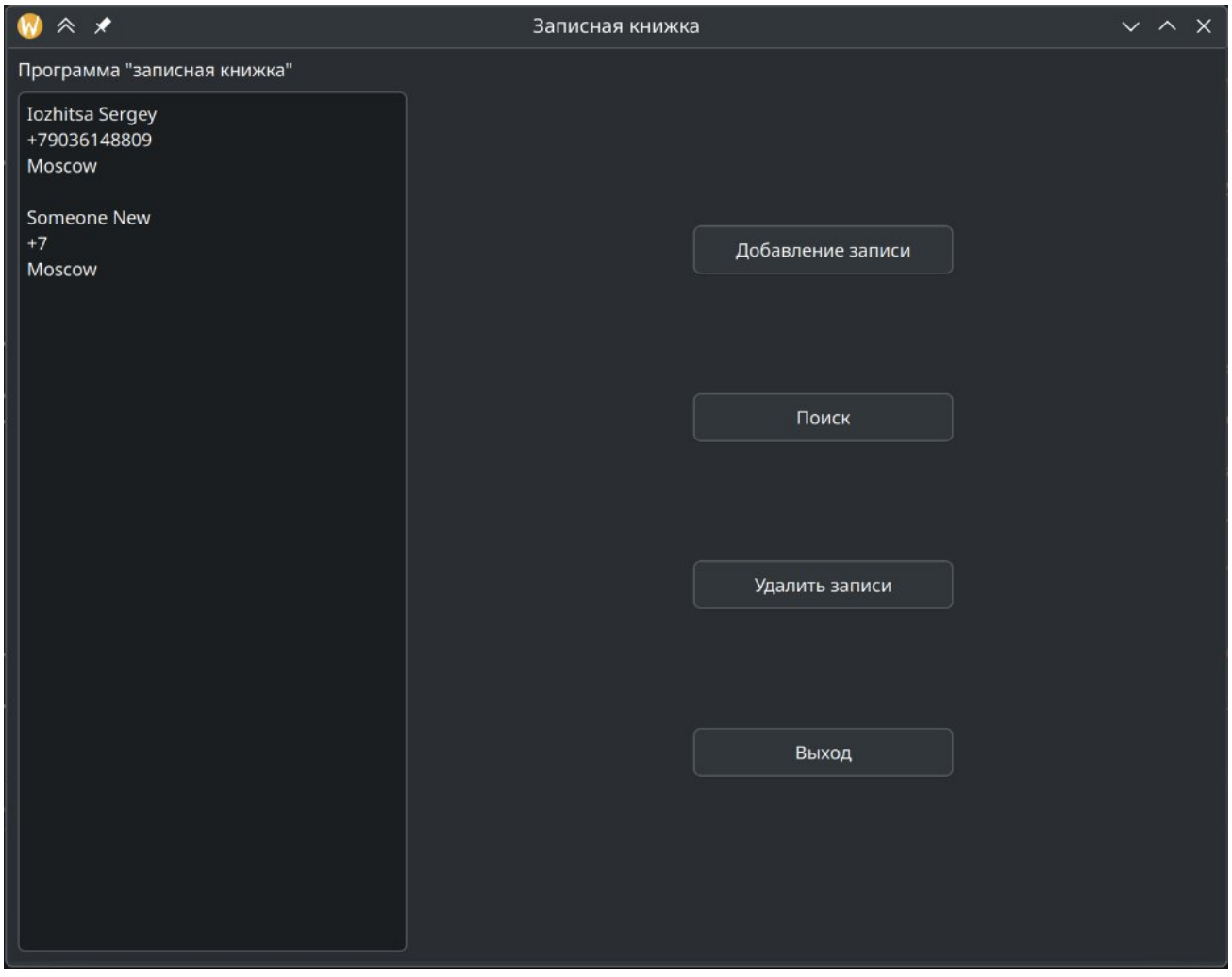

Рисунок 16 — Главное окно после нажатия “OK”

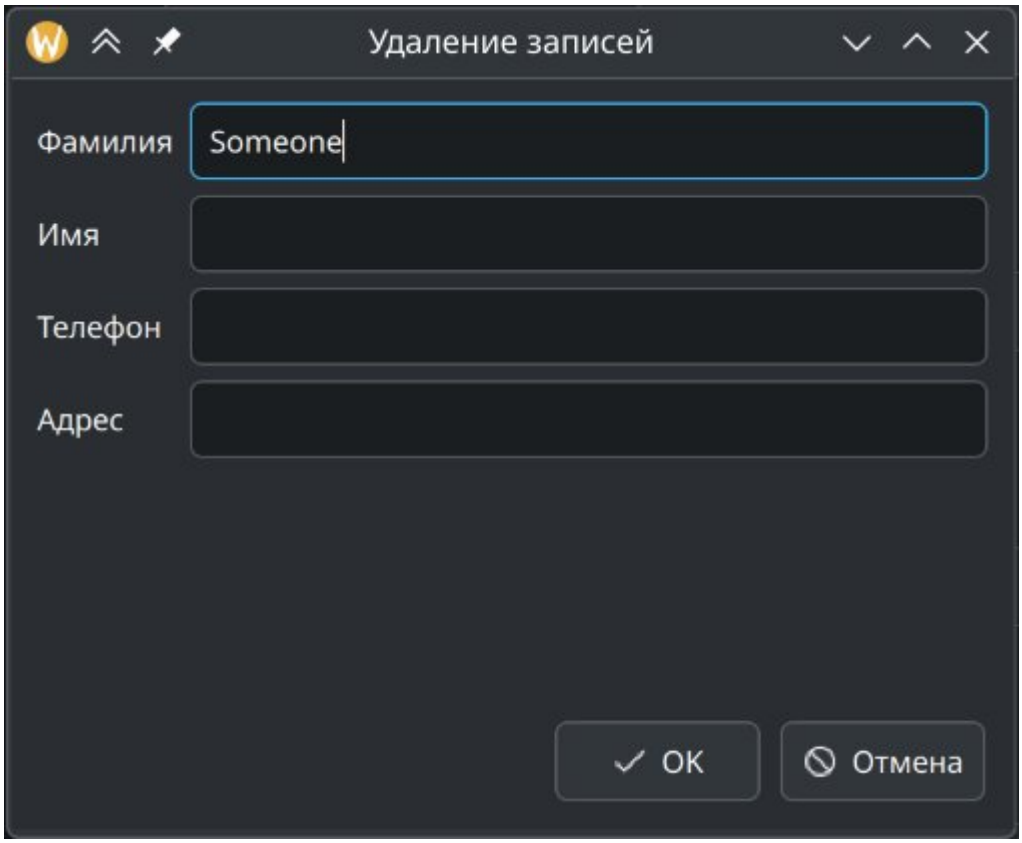

Рисунок 17 — Диалог удаления записей

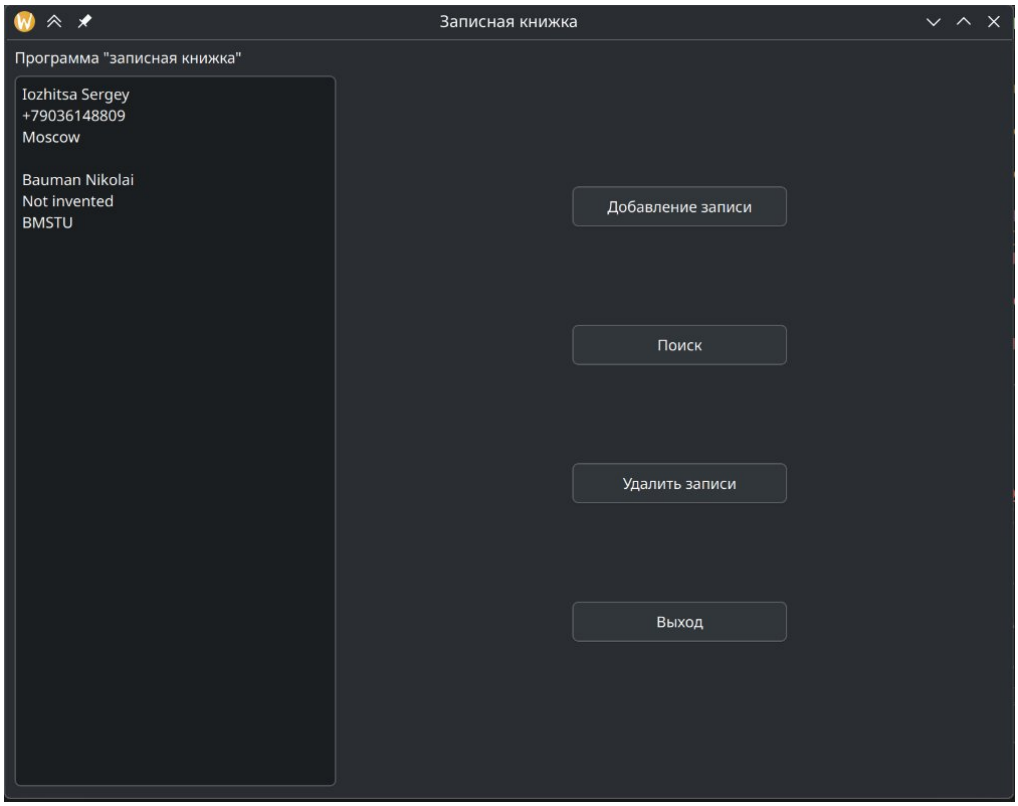

Рисунок 18 — Главное окно после удаления

Вывод: Было создано и протестировано приложение для хранения, создания, поиска и удаления контактов и их адресов.

Часть 3. Движение фигур

Цель работы: разработать на C++ программу, которая в окне отрисовывает вращение трёх фигур: отрезка, равнобедренного треугольника, стрелки компаса (ромб с короткой диагональю).

Выполнение: диаграмма классов изображена на рисунке 19

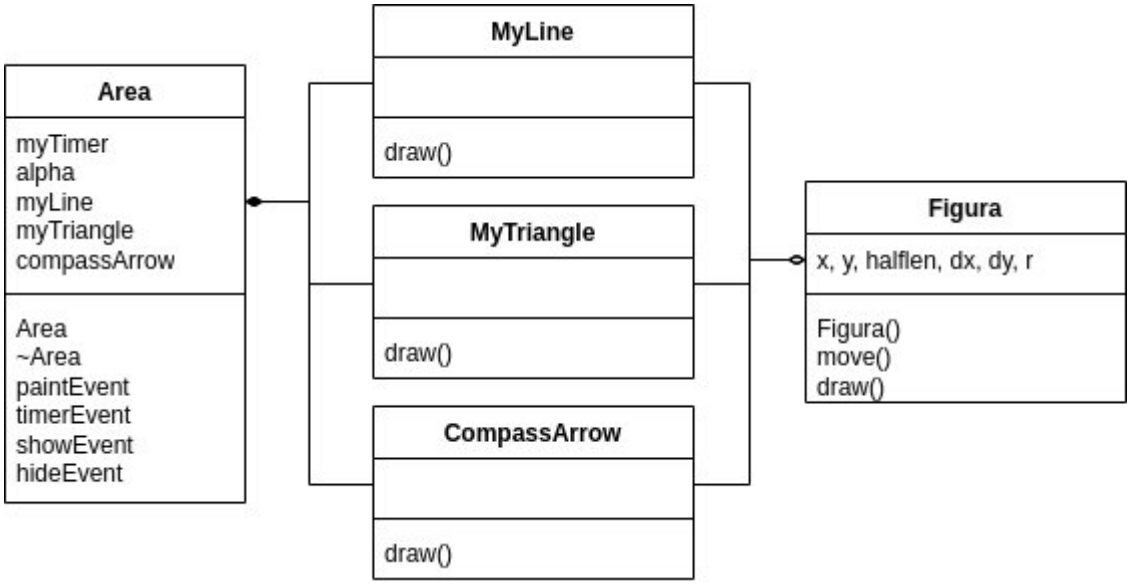

Рисунок 19 — Диаграмма классов

Код программы изображён на рисунках 20-24.

```
class Figura {
protected:
  int x, y, halflen, dx, dy, r;
  virtual void draw(QPainter *Painter) = 0;

public:
  Figura(int X, int Y, int Halflen) : x(X), y(Y), halflen(Halflen) {}
  void move(float Alpha, QPainter *Painter);
  virtual ~Figura() {} // необходим для полиморфного объекта
};

class MyLine : public Figura {
protected:
  void draw(QPainter *Painter);

public:
  MyLine(int x, int y, int halflen) : Figura(x, y, halflen) {}
};

class MyTriangle : public Figura {
protected:
  void draw(QPainter *Painter);

public:
  MyTriangle(int x, int y, int halflen) : Figura(x, y, halflen) {}
};

class CompassArrow : public Figura {
protected:
  void draw(QPainter *Painter);

public:
  CompassArrow(int x, int y, int halflen) : Figura(x, y, halflen) {}
};
```

Рисунок 20 — Заголовочный файл класса Figura

```cpp
void MyLine::draw(QPainter *Painter) {
  Painter->drawLine(x + dx, y + dy, x - dx, y - dy);
}

void MyTriangle::draw(QPainter *Painter) {
  const float sqrt3_2 = 0.8660254f; // sqrt(3)/2

  // Преобразуем в float для точности вычислений
  float fx = x;
  float fy = y;
  float fdx = dx;
  float fdy = dy;

  // Вычисляем координаты трёх вершин
  float x1 = fx + fdx;
  float y1 = fy + fdy;

  float x2 = fx + (-0.5f * fdx - sqrt3_2 * fdy);
  float y2 = fy + (-0.5f * fdy + sqrt3_2 * fdx);

  float x3 = fx + (-0.5f * fdx + sqrt3_2 * fdy);
  float y3 = fy + (-0.5f * fdy - sqrt3_2 * fdx);

  // Создаём массив точек и рисуем треугольник
  QPoint points[3] = {QPoint(static_cast<int>(x1), static_cast<int>(y1)),
                      QPoint(static_cast<int>(x2), static_cast<int>(y2)),
                      QPoint(static_cast<int>(x3), static_cast<int>(y3))};
  Painter->drawPolygon(points, 3);
}
```

Рисунок 21 — Реализция отрисовки MyLine и MyTriangle

```cpp
void CompassArrow::draw(QPainter *Painter) {
  const float wingScale = 0.3f; // Отношение длины крыльев к основной диагонали

  // Преобразование в float для точных вычислений
  float fx = x;
  float fy = y;
  float fdx = dx;
  float fdy = dy;

  // Вычисляем перпендикулярное направление (поворот на 90°) и масштабируем
  float wingX = -fdy * wingScale;
  float wingY = fdx * wingScale;

  // Создаем 4 вершины ромба
  QPoint points[4] = {
      QPoint(static_cast<int>(fx + fdx),
             static_cast<int>(fy + fdy)), // Острие стрелки
      QPoint(static_cast<int>(fx + wingX),
             static_cast<int>(fy + wingY)), // Правое крыло
      QPoint(static_cast<int>(fx - fdx), static_cast<int>(fy - fdy)), // Хвост
      QPoint(static_cast<int>(fx - wingX),
             static_cast<int>(fy - wingY)) // Левое крыло
  };

  Painter->drawPolygon(points, 4);
}
```

Рисунок 22 — Реализация отрисовки CompassArrow

```
class Area : public QWidget {
  int myTimer; // идентификатор таймера
  float alpha; // угол поворота
public:
  Area(QWidget *parent = nullptr);
  ~Area();
  MyLine *myLine; // указатели на объекты фигур
  MyTriangle *myTriangle;
  CompassArrow *compassArrow;

protected:
  // обработчики событий
  void paintEvent(QPaintEvent *event);
  void timerEvent(QTimerEvent *event);
  void showEvent(QShowEvent *event);
  void hideEvent(QHideEvent *event);
};
```

Рисунок 23 — Заголовочный файл холста Area

```
Area::Area(QWidget *parent) : QWidget(parent) {
  setFixedSize(QSize(400, 400)); // фиксируем размер Холста
  myLine = new MyLine(75, 75, 50);
  myTriangle = new MyTriangle(325, 75, 50);
  compassArrow = new CompassArrow(200, 200, 100);
  alpha = 0;
}

Area::~Area() {
  delete myLine;
  delete myTriangle;
}

void Area::showEvent(QShowEvent *) {
  myTimer = startTimer(50); // включаем таймер
}

void Area::paintEvent(QPaintEvent *) {
  QPainter painter(this);         // создаем контент рисования на Холсте
  painter.setPen(Qt::red);        // задаем красное Перо
  myLine->move(alpha, &painter); // рисуем Линию
  myTriangle->move(alpha * (-0.5), &painter); // рисуем Квадрат
  compassArrow->move(alpha * 1.5, &painter);
}

void Area::timerEvent(QTimerEvent *event) {
  if (event->timerId() == myTimer) // если наш таймер
  {
    alpha += 0.2;
    update(); // обновить внешний вид
  } else
    QWidget::timerEvent(event); // иначе событие передать далее
}
void Area::hideEvent(QHideEvent *) {
  killTimer(myTimer); // выключить таймер
}
```

Рисунок 24 — Реализация холста Area

Тестирование: тестирование программы изображено на рисунках 25-26

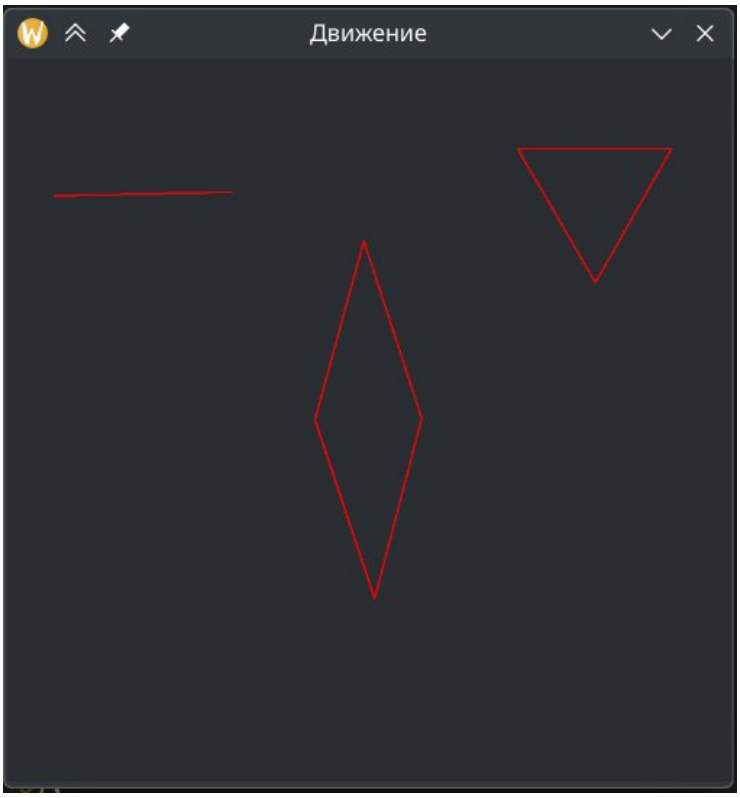

Рисунок 25 — Кадр из работы программы

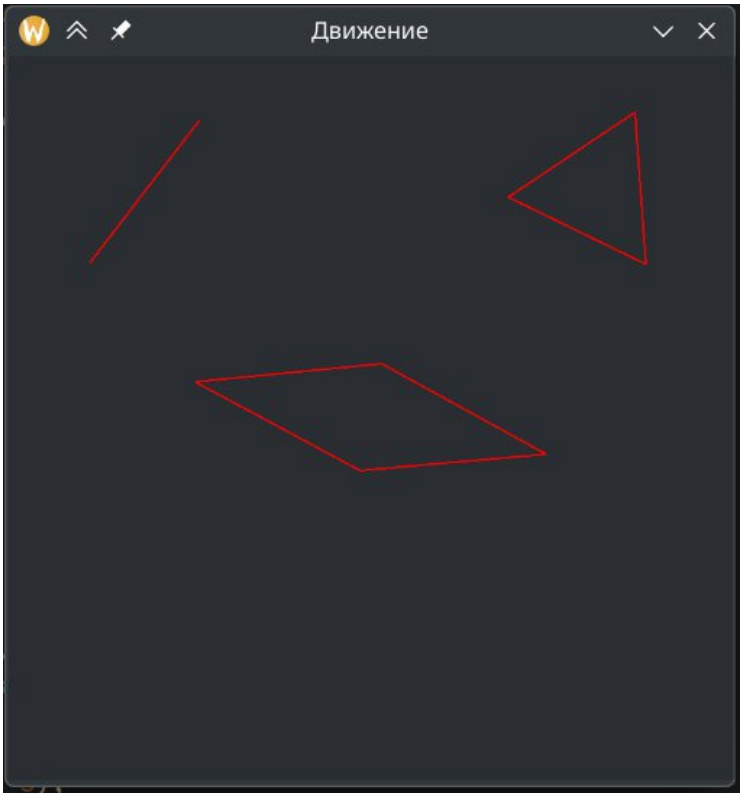

Рисунок 26 — Кадр из работы программы

Вывод: была разработана программа для отрисовки разных вращающихся геометрических примитивов с использванием Qt и C++