Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
КАФЕДРА КОМПЬЮТЕРНОЙ БЕЗОПАСНОСТИ

ОТЧЁТ
К ЛАБОРАТОРНОЙ РАБОТЕ №11
по дисциплине
«Языки программирования»

Работу выполнил
студент группы СКБ-193 _____ М.А.Сурков
подпись, дата

Работу проверил _____ С.А.Булгаков
подпись, дата

Москва 2020

Содержание

Постановка задачи

Разработать графическое приложение с использованием библиотеки Qt. Приложение состоит из основного окна (наследовать QMainWindow), с панелью инструментов на которой расположены:
1)Залипающие кнопки с выбором типа фигуры (количество кнопок соответствует количеству фигур в варианте).
2)Кнопка добавления фигуры. Все фигуры поворачиваются относительно центра Параметры фигуры выбираются произвольно в допустимом диапазоне. Если ни одна фигура не выбрана, кнопка добавления не активна.
3)Кнопка удаления выделенной фигуры. Если фигура не выделена, кнопка не активна. описывающего их прямоугольника (по умолчению против часовой стрелки). Основная часть окна предназначена для размещения фигур (запрещается использовать QGraphicsScene). Фон основной части окна – белый, цвет отрисовки фигур – черный.

При нажатии на фигуре левой кнопкой мыши – фигура выделяется (отрисовывается синим цветом). При нажатии на фигуре правой открывается модальное диалоговое окно позволяющее изменить параметры фигуры, угол поворота, направление поворота, а также отображающее площадь и периметр фигуры.

При перетаскивании выделенной фигуры она меняет свое положение в рамках окна. При достижении края окна перемещение прекращается. Пересечение с другими фигурами не учитывается

Личный вариант №68

Фигура №68
A2B4C4D3E6F6
Фигура №78
A3B1C3D1E5F6

1. Алгоритм решения задачи

Для решения поставленной задачи были разработаны классы MainWindow, являющийся наследником класса QMainWindow, M_Window1 и M_Window2, являющиеся наследниками класса QDialog и классы Figure, Figure2, являющиеся наследниками QWidget, а так же реализованы методы в которых происходят действия необходимые для работы программы. UML диаграмма классов представлена на рисунке 1

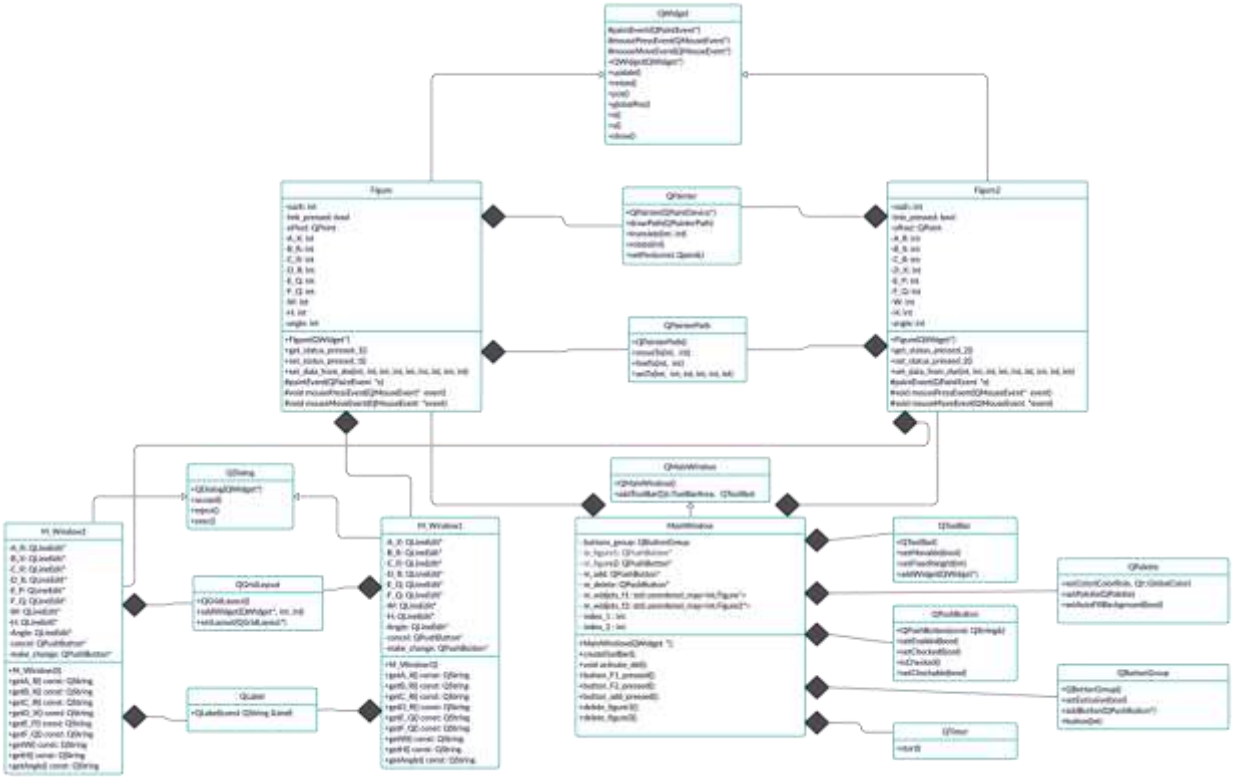


Рисунок1

2. Выполнение задания

2.1 Файл mainwindow.cpp

Данный файл содержит класс **MainWindow**, в котором реализованы панель инструментов и слоты, необходимые для ее работы (**void activate_del(), button_F1_pressed(), button_F2_pressed(), void button_add_pressed(), void delete_figure1(), void delete_figure2()**).

2.2 Файл figure.cpp

Этот файл состоит их двух классов **Figure** и **Figure2**, которые содержать методы при помощи которых происходит отрисовки фигур и все действия над ними.

2.3 Файл m_window1.cpp

Файл содержит класс **M_Window1**, который реализован для настройки модального диалогового окна и взятия данных, введенных пользователем, при помощи методов.

2.4 Файл m_window2.cpp

Файл содержит класс **M_Window2**, который реализован для настройки модального диалогового окна для фигуры 2 и взятия данных, введенных пользователем, при помощи методов.

2.5 Файл main.cpp

В данном файле подключается заголовочный файл класса **MainWindow**. создается обьект этого класса. Также создаётся экземпляр класса **QApplication**, после запускается цикл событий.

2.6 Поля класса MainWindow

2.6.1 QButtonGroup buttons_group

Контейнер для кнопок

2.6.2 QPushButton *m_figure1

Кнопка для фигуры A2B4C4D3E6F6.

2.6.3 QPushButton *m_figure2

Кнопка для фигуры A3B1C3D1E5F6.

2.6.4 QPushButton *m_add

Кнопка для добавления фигуры на экран(активируется при залипнувшей кнопки **m_figure1** или **m_figure2**).

2.6.5 QPushButton *m_delete

Кнопка для удаления фигур(активируется, если фигура выделена).

2.6.6 QTimer tmr

Таймер , издающий сигнал каждую 1мс и отслеживающий выделение фигур, для активации кнопки “-”.

2.6.7 std::unordered_map<int, Figure*> widjets_f1

Контейнер хранящий по ключу указатель на фигуру A2B4C4D3E6F6.

2.6.8 int index_1

Индекс для widjets_f.

2.6.9 std::unordered_map<int, Figure2*> widjets_f1

Контейнер хранящий по ключу указатель на фигуру A3B1C3D1E5F6.

2.6.10 int index_2

Индекс для widjets_f2.

2.7 Поля класса M_Window1

2.7.1 QLineEdit* W

Поле диалогового окна для изменения длины фигуры.

2.7.2 QLineEdit* H

Поле диалогового окна для изменения высоты фигуры.

2.7.3 QLineEdit* Angle

Поле диалогового окна для изменения угла поворота фигуры.

2.7.4 QLineEdit* A_X

Поле диалогового окна для изменения параметра A_X фигуры.

2.7.5 QLineEdit* B_R

Поле диалогового окна для изменения параметра B_R фигуры.

2.7.6 QLineEdit* C_R

Поле диалогового окна для изменения параметра C_R фигуры.

2.7.7 QLineEdit* D_R

Поле диалогового окна для изменения параметра D_R фигуры.

2.7.8 QLineEdit* E_Q

Поле диалогового окна для изменения параметра E_Q фигуры.

2.7.9 QLineEdit* F_Q

Поле диалогового окна для изменения параметра F_Q фигуры.

2.7.10 QPushButton* make_change

Кнопка для применения измененных параметров фигуры.

2.7.11 QPushButton* cancel

Кнопка для выхода из диалогового окна, если вы не хотите менять параметры фигуры.

2.8 Поля класса M_Window2

2.8.1 QLineEdit* W

Поле диалогового окна для изменения длины фигуры.

2.8.2 QLineEdit* H

Поле диалогового окна для изменения высоты фигуры.

2.8.3 QLineEdit* Angle

Поле диалогового окна для изменения угла поворота фигуры.

2.8.4 QLineEdit* A_X

Поле диалогового окна для изменения параметра A_X фигуры.

2.8.5 QLineEdit* B_R

Поле диалогового окна для изменения параметра B_R фигуры.

2.8.6 QLineEdit* C_R

Поле диалогового окна для изменения параметра C_R фигуры.

2.8.7 QLineEdit* D_R

Поле диалогового окна для изменения параметра D_R фигуры.

2.8.8 QLineEdit* E_Q

Поле диалогового окна для изменения параметра E_Q фигуры.

2.8.9 QLineEdit* F_Q

Поле диалогового окна для изменения параметра F_Q фигуры.

2.8.10 QPushButton* make_change

Кнопка для применения измененных параметров фигуры.

2.8.11 QPushButton* cancel

Кнопка для выхода из диалогового окна, если вы не хотите менять параметры фигуры.

2.9 Поля класса figure_1

2.9.1 int W

Длина фигуры.

2.9.2 int H

Высота фигуры.

2.9.3 int A_X

Длина линии для точки A.

2.9.4 int B_R

Длина радиуса для точки B.

2.9.5 int C_R

Длина радиуса для точки C.

2.9.5 int D_R

Радиус окружности для точки D.

2.9.6 int E_Q

Диаметр окружности для точки E.

2.9.7 int F_Q

Диаметр окружности для точки F.

2.9.8 bool lmd_pressed

Флаг который нужен для выделения фигуры синим(при выделении) или черным цветом.

2.9.9 int angle

Угол поворота фигуры.

2.9.10 QPoint offset

Точка, в которой последний раз было нажатие мышью

2.9.11 int nazh

Счетчик для подсчета кликов по фигуре.

2.10 Поля класса figure_2

2.10.1 int W

Длина фигуры.

2.10.2 int H

Высота фигуры.

2.10.3 int A_R

Длина радисуа для точки A.

2.10.4 int B_X

Длина линии для точки B.

2.10.5 int C_R

Длина радиуса для точки D.

2.10.6 int E_P

Длина стороны прямоугольника для точки E.

2.10.7 int F_Q

Длина диаметра для точки E.

2.10.8 bool lmb_pressed

Флаг который нужен для выделения фигуры синим(при выделении) или черным цветом.

2.10.9 int angle

Угол поворота фигуры.

2.10.10 QPoint offset

Точка, в которой последний раз было нажатие мышью

2.10.11 int nazh

Счетчик для подсчета кликов по фигуре.

2.11 Методы класса Main_Window

2.11.1 Метод MainWindow(QWidget *p) : QMainWindow(p)

Это конструктор класса, в котором происходит установка панели инструментов и соответствующих коннектов различных сигналов и слотов.

2.11.2 QToolBar* createToolBar()

Метод в котором настраивается панель инструментов (добавление кнопок и их характеристик)

2.11.3 void activate_del()

Слот класса необходимый для активации кнопки удаления фигуры

2.11.4 void button_F1_pressed()

Слот, который отвечает за залипание кнопки 2, и активацию кнопки добавления

2.11.5 void button_F2_pressed()

Слот, который отвечает за залипание кнопки 1, и активацию кнопки добавления

2.11.6 void button_add_pressed()

Слот отвечающий за добавлении фигур(все зависит от того какая кнопка нажата)

2.11.7 void delete_figure1();

Слот отвечающий за удаление фигуры 1.

2.11.8 void delete_figure2();

Слот отвечающий за удаление фигуры 2.

2.12 Методы класса M_Window1

2.12.1 QString getW() const

Возвращает строку введенную пользователем (длину фигуры).

2.12.2 QString getH() const

Возвращает строку введенную пользователем (высоту фигуры).

2.12.3 QString getA_X() const

Возвращает строку введенную пользователем (длину линии для точки A).

2.12.4 QString getB_R() const

Возвращает строку введенную пользователем (длина радиуса для точки B).

2.12.5 QString getC_R() const

Возвращает строку введенную пользователем (длина радиуса для точки C).

2.12.6 QString getD_R() const

Возвращает строку введенную пользователем (длина радиуса для точки D).

2.12.7 QString getE_Q() const

Возвращает строку введенную пользователем (длина диаметр для точки E).

2.12.8 QString getF_Q() const

Возвращает строку введенную пользователем (длина диаметр для точки F).

2.12.9 QString getAngle() const

Возвращает строку введенную пользователем (угол поворота фигуры).

2.12.10 M_Window1()

Конструктор класса M_Window1

2.13 Методы класса M_Window2

2.13.1 QString getW() const

Возвращает строку введенную пользователем (длину фигуры).

2.13.2 QString getH() const

Возвращает строку введенную пользователем (высоту фигуры).

2.13.3 QString getA_R() const

Возвращает строку введенную пользователем (длину радиуса для точки A).

2.13.4 QString getB_X() const

Возвращает строку введенную пользователем (длину линии для точки B).

2.13.5 QString getC_R() const

Возвращает строку введенную пользователем (длину радиуса для точки R).

2.13.6 QString getD_X() const

Возвращает строку введенную пользователем (длину линии для точки D).

2.13.7 QString getE_P() const

Возвращает строку введенную пользователем (длину стороны прямоугольника точки D).

2.13.8 QString getF_Q() const

Возвращает строку введенную пользователем (длину диаметра для точки D).

2.13.9 QString getAngle() const

Возвращает строку введенную пользователем (угол поворота фигуры).

2.13.10 M_Window2()

Конструктор класса M_Window2

2.14 Методы класса Figure

2.14.1 Figure(QWidget *parent) : QWidget(parent)

Является конструктором класса, в котором инициализируются параметры фигуры

2.14.2 void set_data_from_dw(int H, int W_, int A_X_, int B_R_, int C_R_, int D_R_, int E_Q_, int F_Q_, int angle_)

Устанавливает новые параметры фигуры

2.14.3 void paintEvent(QPaintEvent *e)

Является переопределенным методом класса-родителя, в котором происходит отрисовка фигуры.

2.14.4 void set_status_pressed_1(bool status)

Данный метод устанавливает значение флага выделения фигуры.

2.14.5 bool get_status_pressed_1()

Данный метод возвращает значение флага выделения фигуры.

2.14.6 void mouseMoveEvent(QMouseEvent *event)

Является переопределенным методом класса-родителя, в котором происходит передвижение фигуры.

2.14.7 void mousePressEvent(QMouseEvent* event)

Является переопределенным методом класса-родителя, в котором отслеживаются нажатия мыши по фигуре.

2.14 Методы класса Figure2

2.14.1 Figure2(QWidget *parent) : QWidget(parent)

Является конструктором класса, в котором инициализируются параметры фигуры

2.14.2 void set_data_from_dw(int H_, int W_, int A_R_, int B_X_, int C_R_, int D_X_, int E_P_, int F_Q_, int angle_)

Устанавливает новые параметры фигуры

2.14.3 void paintEvent(QPaintEvent *e)

Является переопределенным методом класса-родителя, в котором происходит отрисовка фигуры.

2.14.4 void set_status_pressed_2(bool status)

Данный метод устанавливает значение флага выделения фигуры.

2.14.5 bool get_status_pressed_2()

Данный метод возвращает значение флага выделения фигуры.

2.14.6 void mouseMoveEvent(QMouseEvent *event)

Является переопределенным методом класса-родителя, в котором происходит передвижение фигуры.

2.14.7 void mousePressEvent(QMouseEvent* event)

Является переопределенным методом класса-родителя, в котором отслеживаются нажатия мыши по фигуре.

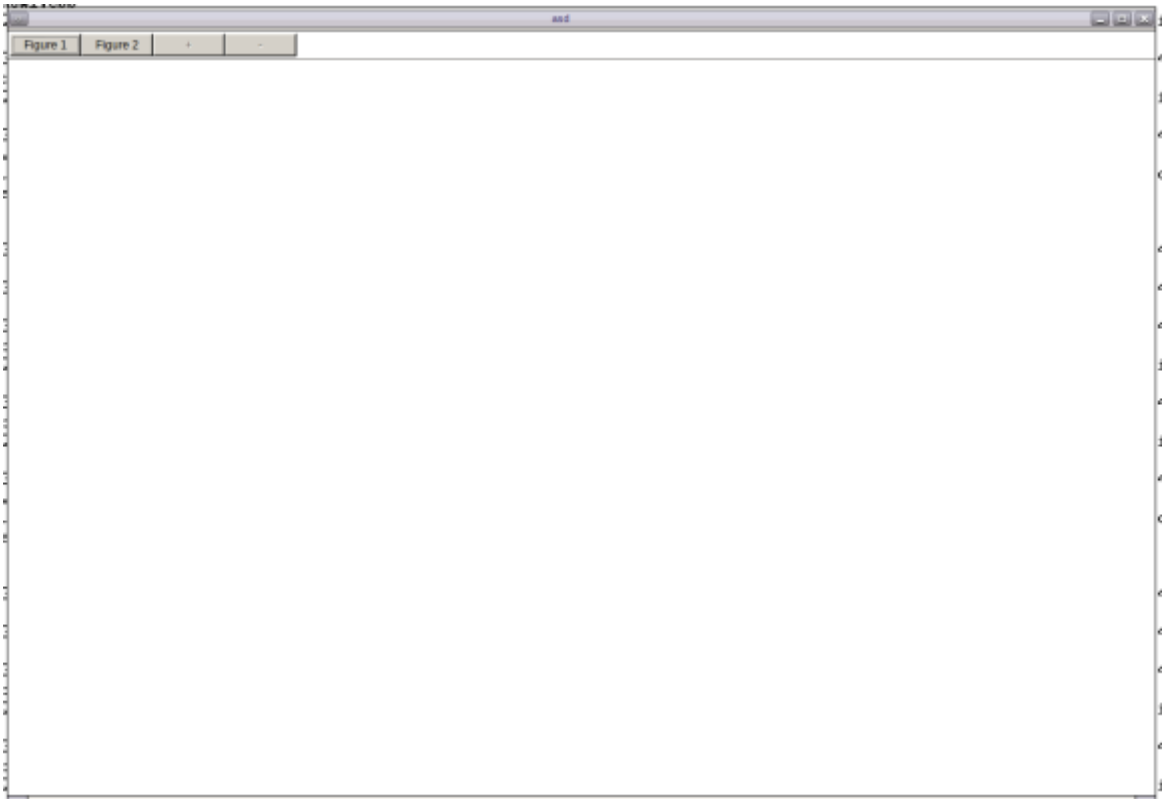
3. Процедура получения исполняемых файлов

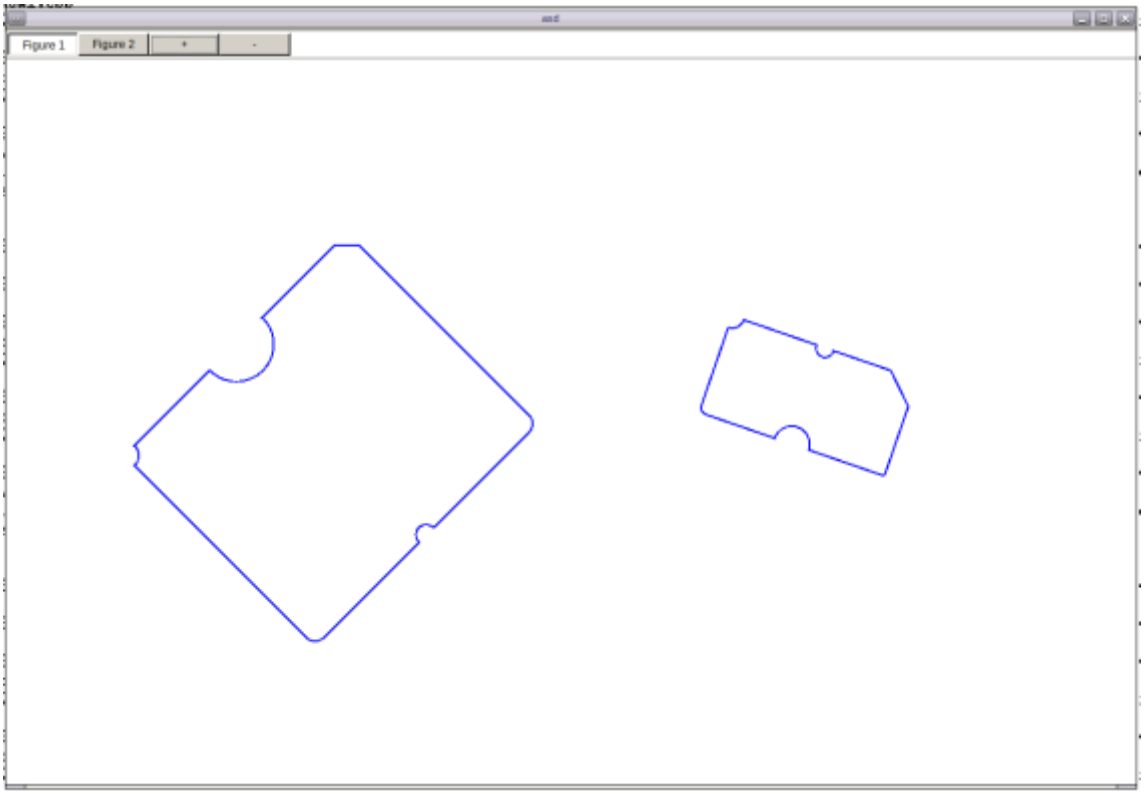
В программе есть файл asd.pro, который создает qmake, а сам qmake создает уже makefile.

4. Тестирование

Результат работы программы представлен в листинге 1.

Листинг 1





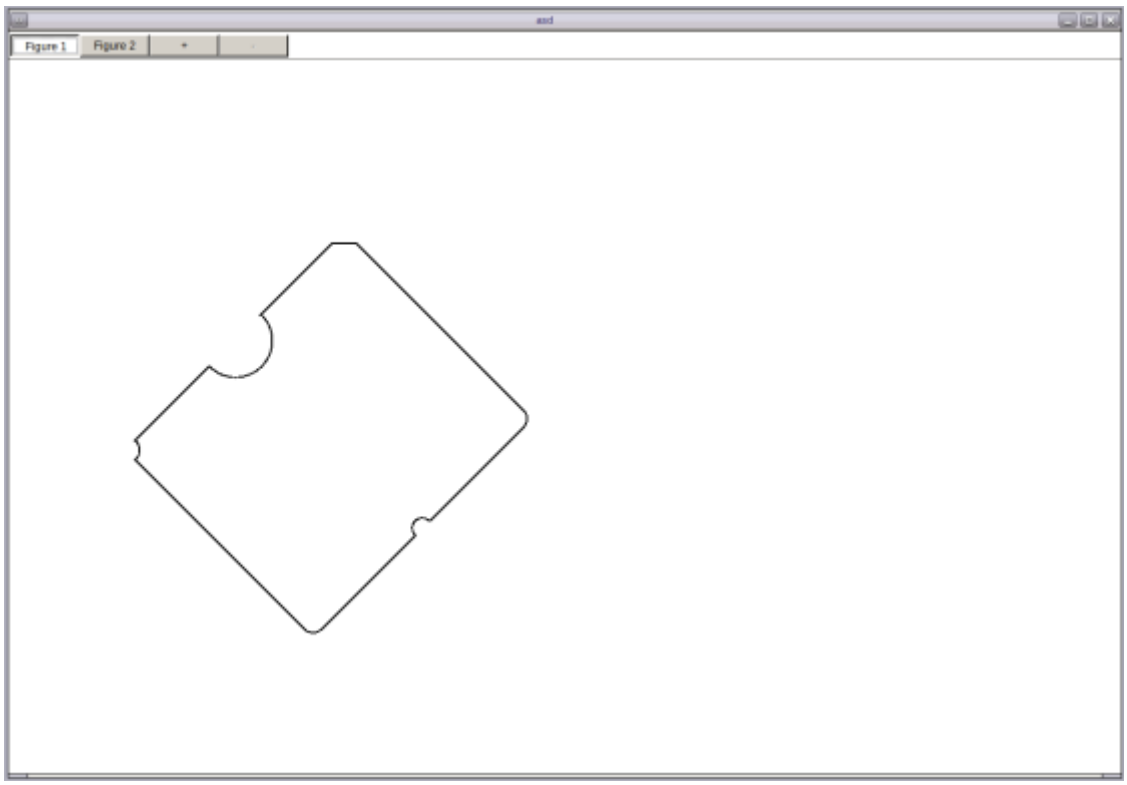


Figure 1
Figure 2
+
-

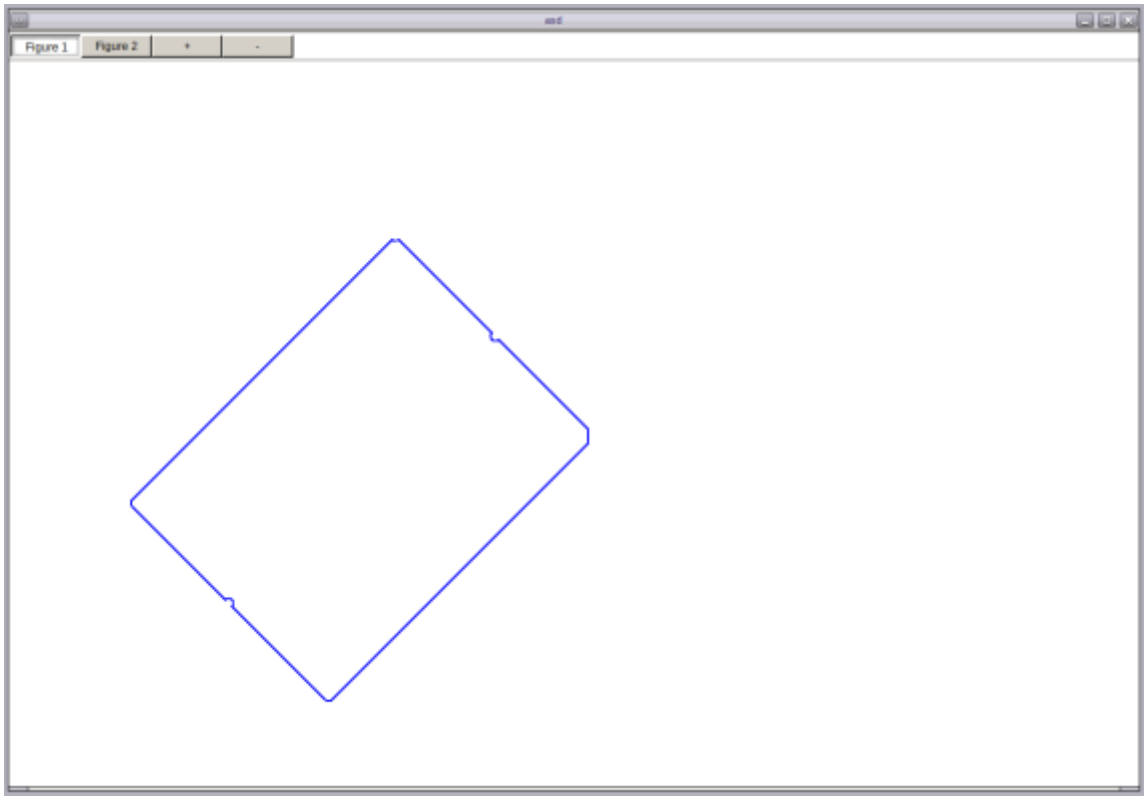


Figure 1
Figure 2
+
-

Приложение А

А1. Исходный код файла **main.cpp**

```
#include "mainwindow.h"
#include <QApplication>

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

    QApplication app(argc, argv);
    MainWindow a;
    a.resize(1200,800);
    a.show();
    return app.exec();

}
```

А2. Исходный код файла **figure.h**

```
#ifndef FIG
#define FIG

#include <QWidget>
#include <QPainter>
#include <QPainterPath>

class Figure : public QWidget {

    Q_OBJECT

private:
  bool lmb_pressed = false;
  QPoint offset;
  int H, W, A_X, B_R, C_R, D_R, E_Q, F_Q, angle = 0, nazh = 0;

public:
  Figure(QWidget *parent = 0);
  bool get_status_pressed_1();
  void set_status_pressed_1(bool);
  void set_data_from_dw(int, int, int, int, int, int, int, int,
int);

protected:
  void paintEvent(QPaintEvent *e);
  void mousePressEvent(QMouseEvent* event);
  void mouseMoveEvent(QMouseEvent *event);
};

class Figure2 : public QWidget {

  Q_OBJECT

private:
```

```
    bool lmb_pressed = false;
    QPoint offset;
    int W, H, A_R, B_X, C_R, D_X, E_P, F_Q, angle = 0, nazh = 0;

public:
  Figure2(QWidget *parent = 0);
  bool get_status_pressed_2();
  void set_status_pressed_2(bool);
  void set_data_from_dw(int, int, int, int, int, int, int, int,
int);

protected:
  void paintEvent(QPaintEvent *e);
  void mousePressEvent(QMouseEvent* event);
  void mouseMoveEvent(QMouseEvent *event);
};


#endif
```

А3. Исходный код файла **figure.cpp**

```
#include "figure.h"
#include <cmath>
#include <QMouseEvent>
#include "m_window1.h"
#include "m_window2.h"
#include <QMessageBox>

Figure::Figure(QWidget *parent) : QWidget(parent) {
    W = rand() % 300 + 50;
    H = rand() % (W - 50 ) + 50;

    A_X = rand()% (H / 3),
    B_R = rand()% (H / 3),
    C_R = rand()% (H / 3),
    D_R = rand()% (H / 3),
    E_Q = rand()% (W / 4),
    F_Q = rand()% (W / 4);
    angle = rand()% 360;
    setGeometry(rand() % 600 + 30, rand() % 400 + 30,
W*sqrt(2)+1,H*sqrt(2)+1);
}

void Figure::set_data_from_dw(int H_,int  W_, int A_X_, int
B_R_, int C_R_, int D_R_, int E_Q_, int F_Q_, int angle_){
    H = H_;
    W = W_;
    A_X = A_X_;
    B_R = B_R_;
    C_R = C_R_;
    D_R = D_R_;
    E_Q = E_Q_;
```

```
    F_Q = F_Q_;
    angle = angle_ % 360;

}

void Figure::paintEvent(QPaintEvent *e) {

    Q_UNUSED(e)

    this->resize(sqrt(W*W + H*H) + 1, sqrt(W*W + H*H) + 1);
    QPainterPath path_figure1;
    QPainter painter(this);
    painter.translate((sqrt(W*W + H*H) + 1)/2, (sqrt(W*W + H*H)
+ 1)/2);
    painter.rotate(angle);
    path_figure1.moveTo(0 + E_Q/2, -H/2);

    if (!lmb_pressed){
        QPen linepen(Qt::black);
        linepen.setWidth(2);
        painter.setPen(linepen);
    }
    else{
        QPen linepen(Qt::blue);
        linepen.setWidth(2);
        painter.setPen(linepen);

    }

    path_figure1.lineTo(W/2 - A_X, -H/2);
    path_figure1.lineTo(W/2, -H/2 + A_X);
    path_figure1.lineTo(W/2, H/2 - B_R/2);

    path_figure1.moveTo(W/2 - B_R,H/2);
    QRectF rectangle( W/2 - B_R,H/2 - B_R ,B_R, B_R);
    path_figure1.arcTo(rectangle, 270,90);

    path_figure1.moveTo(W/2 - B_R,H/2);
    path_figure1.lineTo(0 + F_Q/2,H/2);
    QRectF rectangle2( 0 - F_Q/2, H/2 - F_Q/2 ,F_Q, F_Q);
    path_figure1.arcTo(rectangle2, 0,180);

    path_figure1.moveTo(0 - F_Q/2,H/2);
    path_figure1.lineTo(0-W/2+C_R/2,H/2);
    path_figure1.moveTo(-W/2,H/2 - C_R);
    QRectF rectangle3(- W/2  , H/2 - C_R, C_R, C_R);
    path_figure1.arcTo(rectangle3, 180,90);
    path_figure1.moveTo(-W/2,H/2 - C_R);

    path_figure1.lineTo(-W/2, -H/2 + D_R/2 );
    QRectF rectangle4(- W/2 - D_R/2, -H/2 - D_R/2, D_R, D_R);
    path_figure1.arcTo(rectangle4, 270,90);
```

```
    path_figure1.moveTo(- W/2 + D_R/2, - H/2 );
    path_figure1.lineTo(0 - E_Q/2,-H/2);
    QRectF rectangle5(0 - E_Q/2,-H/2 - E_Q/2, E_Q, E_Q);
    path_figure1.arcTo(rectangle5, 180,180);
    painter.drawPath(path_figure1);
}

void Figure::set_status_pressed_1(bool status){
    lmb_pressed = status;
}

bool Figure::get_status_pressed_1(){
    return lmb_pressed;
}

void Figure::mouseMoveEvent(QMouseEvent *event){

    lmb_pressed = true;
    QWidget::update();
    if (event->buttons() & Qt::LeftButton){
        if (((event->globalPos() - offset).x() + width() > 1200)
||
            ((event->globalPos() - offset).y() + height() > 800)
||
            ((event->globalPos() - offset).x() <= 0 ) ||
            ((event->globalPos() - offset).y() <= 30))
                offset = event->globalPos() - pos();
        else
            move(event->globalPos() - offset);
    }

}

void Figure::mousePressEvent(QMouseEvent* event){

    if (event->button() & Qt::LeftButton){
        nazh++;
        if (nazh % 2 == 0)
            lmb_pressed = false;
        else
            lmb_pressed = true;
        QWidget::update();
        offset = event->globalPos() - pos();
    }
    if (event->button() & Qt::RightButton){
        if (!lmb_pressed){
              nazh++;
        }
        lmb_pressed = true;
        QWidget::update();
        M_Window1* w1 = new M_Window1;
        if(w1->exec() == QDialog::Accepted){
```

```
            if (w1->getW() == "0" || w1->getH() == "0" || w1-
>getA_X() == "0" ||
                w1->getB_R() == "0" || w1->getC_R() == "0" ||
w1->getD_R() == "0" ||
                    w1->getE_Q() == "0"){
                QMessageBox::information(0 ,"Information", "Some
data is zero");
            }
            else{
                if (((w1->getW()).toInt() <= (w1-
>getH()).toInt()) ||
                    ((w1->getA_X()).toInt() >= (w1-
>getH()).toInt()/3) ||
                    ((w1->getB_R()).toInt() >= (w1-
>getH()).toInt()/3) ||
                    ((w1->getC_R()).toInt() >= (w1-
>getH()).toInt()/3) ||
                    ((w1->getD_R()).toInt() >= (w1-
>getH()).toInt()/3) ||
                    ((w1->getW()).toInt() /4 <= (w1-
>getE_Q()).toInt()) ||
                    ((w1->getW()).toInt() /4 <= (w1-
>getF_Q()).toInt())){
                    QMessageBox::information(0,
                    "Information",
                    "Error entered"
                    );
                }
                else{
                    set_data_from_dw((w1->getW()).toInt(),
                            (w1->getH()).toInt(),
                            (w1->getA_X()).toInt(),
                            (w1->getB_R()).toInt(),
                            (w1->getC_R()).toInt(),
                            (w1->getD_R()).toInt(),
                            (w1->getE_Q()).toInt(),
                            (w1->getF_Q()).toInt(),
                            (w1->getAngle()).toInt());
                    double perimetr = (W/2 - A_X - E_Q/2) + (H/2
- A_X - B_R)
                            + 3.14*B_R/2 + (W/2 - F_Q/2 - B_R/2)
+ 3.14*F_Q/2
                            + (W/2 - C_R/2 - F_Q/2) + 3.14*C_R/2
                            + (H/2 - C_R - D_R/2) + (W/2 - D_R/2
- E_Q/2);
                    QString sp = QString::number(perimetr);
                    double square = W*H - A_X*A_X/2 -
3.14*B_R*B_R/4  - 3.14*F_Q*F_Q/2 - 3.14*C_R*C_R/4 -
3.14*D_R*D_R/4 - 3.14*E_Q*E_Q/2;
                    QString ss = QString::number(square);
                    QMessageBox::information(0,
                            "Information",
                            "Square: "
```

```
                                + ss
                                +"\nPerimetr: "
                                + sp
                                );
                        QWidget::update();
                    }
                }
            }
            else{
                double perimetr = (W/2 - A_X/2 - E_Q/2) + abs(-H/2 +
A_X) + (H/2 - B_R/2) + 3.14*B_R/2 +(W/2 - F_Q/2) + 3.14*F_Q/2 +
(W/2 - C_R/2 - F_Q/2) + 3.14*C_R/2 +
                        (H/2 - C_R - D_R/2) + (W/2 - D_R/2 - E_Q/2);
                QString sp = QString::number(perimetr);
                double square = W*H - A_X*A_X/2 - 3.14*B_R*B_R/4  -
3.14*F_Q*F_Q/2 - 3.14*C_R*C_R/4 - 3.14*D_R*D_R/4 -
3.14*3.14*E_Q*E_Q/2;
                QString ss = QString::number(square);
                QMessageBox::information(0,
                        "Information",
                        "Square: "
                        + ss
                        +"\nPerimetr: "
                        + sp
                        );
                QWidget::update();
            }
            delete w1;
        }
}

Figure2::Figure2(QWidget *parent) : QWidget(parent) {
    W = rand() % 300 + 50;
    H = rand() % (W - 50 ) + 50;

    A_R = rand()% (H / 3),
    B_X = rand()% (H / 3),
    C_R = rand()% (H / 3),
    D_X = rand()% (H / 3),
    E_P = rand()% (W / 4),
    F_Q = rand()% (W / 4);
    angle = rand()% 360;
    setGeometry(rand() % 600 + 30, rand() % 400 + 30,
W*sqrt(2)+1,H*sqrt(2)+1);
}

void Figure2::set_data_from_dw(int H_,int  W_, int A_R_, int
B_X_, int C_R_, int D_X_, int E_P_, int F_Q_, int angle_){
    H = H_;
    W = W_;
    A_R = A_R_;
    B_X = B_X_;
    C_R = C_R_;
```

```
    D_X = D_X_;
    E_P = E_P_;
    F_Q = F_Q_;
    angle = angle_ % 360;

}

void Figure2::paintEvent(QPaintEvent *e) {
    Q_UNUSED(e)
    this->resize(sqrt(W*W + H*H) + 1, sqrt(W*W + H*H) + 1);

    QPainterPath path_figure2;
    QPainter painter(this);
    painter.translate(sqrt(W*W + H*H)/2, sqrt(W*W + H*H)/2);
    painter.rotate(angle);
    path_figure2.moveTo(0 + E_P/2, -H/2);
    if (!lmb_pressed){
        QPen linepen(Qt::black);
        linepen.setWidth(2);
        painter.setPen(linepen);

    }
    else{
        QPen linepen(Qt::blue);
        linepen.setWidth(2);
        painter.setPen(linepen);
    }
    //A3
    path_figure2.lineTo(W/2 - A_R/2,-H/2);
    QRectF rectangle(W/2 - A_R/2, -H/2 - A_R/2, A_R, A_R);
    path_figure2.arcTo(rectangle, 180, 90);
    path_figure2.moveTo(W/2, -H/2 + A_R/2);
    //B1
    path_figure2.lineTo(W/2, H/2 - B_X);
    path_figure2.lineTo(W/2 - B_X, H/2 - B_X);
    path_figure2.lineTo(W/2 - B_X, H/2);
    //F6
    path_figure2.lineTo(0 + F_Q/2,H/2);
    QRectF rectangle2( 0 - F_Q/2, H/2 - F_Q/2 ,F_Q, F_Q);
    path_figure2.arcTo(rectangle2, 0,180);
    //C3
    path_figure2.moveTo(0 - F_Q/2, H/2);
    path_figure2.lineTo(-W/2 + C_R/2,H/2);
    QRectF rectangle3( -W/2 - C_R/2, H/2 - C_R/2 ,C_R, C_R);
    path_figure2.arcTo(rectangle3, 0,90);
    path_figure2.moveTo(-W/2, H/2 - C_R/2);
    //D1
    path_figure2.lineTo(-W/2, -H/2 + D_X);
    path_figure2.lineTo(-W/2 + D_X, -H/2 + D_X);
    path_figure2.lineTo(-W/2 + D_X, -H/2);
    //E5
     path_figure2.lineTo(-E_P/2, -H/2);
     path_figure2.lineTo(-E_P/2, -H/2 + E_P/2);
```

```
     path_figure2.lineTo(E_P/2, -H/2 + E_P/2);
     path_figure2.lineTo(E_P/2, -H/2);
     painter.drawPath(path_figure2);
}

void Figure2::mouseMoveEvent(QMouseEvent *event){
    lmb_pressed = true;
    QWidget::update();
    if (event->buttons() & Qt::LeftButton){
        if (((event->globalPos() - offset).x() + width() > 1200)
||
            ((event->globalPos() - offset).y() + height() > 800)
||
            ((event->globalPos() - offset).x() <= 0 ) ||
            ((event->globalPos() - offset).y() <= 30))
                offset = event->globalPos() - pos();
        else
            move(event->globalPos() - offset);
    }
}

void Figure2::mousePressEvent(QMouseEvent* event){
    if (event->button() & Qt::LeftButton){
        nazh++;
        if (nazh % 2 == 0)
            lmb_pressed = false;
        else
            lmb_pressed = true;
        QWidget::update();
        offset = event->globalPos() - pos();
    }
    if (event->button() & Qt::RightButton){
        if (!lmb_pressed){
              nazh++;
        }
        lmb_pressed = true;
        QWidget::update();
        M_Window2* w1 = new M_Window2;
        if(w1->exec() == QDialog::Accepted){
            if (w1->getW() == "0" || w1->getH() == "0" || w1-
>getA_R() == "0" ||
                w1->getB_X() == "0" || w1->getC_R() == "0" ||
w1->getD_X() == "0" ||
                    w1->getE_P() == "0"){
                QMessageBox::information(0 ,"Information", "Some
data is zero");
            }
            else{
                if (((w1->getW()).toInt() <= (w1-
>getH()).toInt()) ||
                   ((w1->getA_R()).toInt() >= (w1-
>getH()).toInt()/3) ||
```

```
                    ((w1->getB_X()).toInt() >= (w1-
>getH()).toInt()/3) ||
                    ((w1->getC_R()).toInt() >= (w1-
>getH()).toInt()/3) ||
                    ((w1->getD_X()).toInt() >= (w1-
>getH()).toInt()/3) ||
                    ((w1->getW()).toInt() /4 <= (w1-
>getE_P()).toInt()) ||
                    ((w1->getW()).toInt() /4 <= (w1-
>getF_Q()).toInt())){
                        QMessageBox::information(0,
                        "Information",
                        "Error entered"
                        );
                }
                else{
                    set_data_from_dw((w1->getW()).toInt(),
                            (w1->getH()).toInt(),
                            (w1->getA_R()).toInt(),
                            (w1->getB_X()).toInt(),
                            (w1->getC_R()).toInt(),
                            (w1->getD_X()).toInt(),
                            (w1->getE_P()).toInt(),
                            (w1->getF_Q()).toInt(),
                            (w1->getAngle()).toInt());
                    double perimetr = (W/2 - A_R/2 - E_P/2) +
abs(H/2 - A_R - B_X) + 2*B_X  + (W/2 - F_Q/2 - B_X ) +
3.14*F_Q/2 + (W/2 - C_R/2 - F_Q/2) + 3.14*C_R/2 +
                            (H/2 - C_R - D_X) + 2*D_X + (W/2 -
D_X/2 - E_P/2) + 2*E_P;
                    QString sp = QString::number(perimetr);
                    double square = W*H - 3.14*A_R*A_R/4 -
B_X*B_X  - 3.14*F_Q*F_Q/2 - 3.14*C_R*C_R/4 - D_X*D_X -
E_P*E_P/2;
                    QString ss = QString::number(square);
                    QMessageBox::information(0,
                            "Information",
                            "Square: "
                            + ss
                            +"\nPerimetr: "
                            + sp
                            );
                    QWidget::update();

                }
            }
        }
        else{
            double perimetr = (W/2 - A_R/2 - E_P/2) + abs(H/2 -
A_R - B_X) + 2*B_X  + (W/2 - F_Q/2 - B_X ) + 3.14*F_Q/2 + (W/2 -
C_R/2 - F_Q/2) + 3.14*C_R/2 +
                    (H/2 - C_R - D_X) + 2*D_X + (W/2 - D_X/2 -
E_P/2) + 2*E_P;
```

```
            QString sp = QString::number(perimetr);
            double square = W*H - 3.14*A_R*A_R/4 - B_X*B_X  -
3.14*F_Q*F_Q/2 - 3.14*C_R*C_R/4 - D_X*D_X - E_P*E_P/2;
            QString ss = QString::number(square);
            QMessageBox::information(0,
                    "Information",
                    "Square: "
                    + ss
                    +"\nPerimetr: "
                    + sp
                    );
            QWidget::update();
        }
        delete w1;
    }
}

bool Figure2::get_status_pressed_2(){
    return lmb_pressed;
}

void Figure2::set_status_pressed_2(bool status){
    lmb_pressed = status;
}
```

A4. Исходный код файла **m_window1.h**

```
#ifndef M_WINDOW1_H
#define M_WINDOW1_H
#include <QDialog>
#include <QLineEdit>
#include <QLabel>
#include <QPushButton>
#include <QGridLayout>

class M_Window1 : public QDialog{

    Q_OBJECT

private:
    QLineEdit* W;
    QLineEdit* H;
    QLineEdit* A_X;
    QLineEdit* B_R;
    QLineEdit* C_R;
    QLineEdit* D_R;
    QLineEdit* E_Q;
    QLineEdit* F_Q;
    QLineEdit* Angle;
    QPushButton *make_change;
    QPushButton *cancel;

public:
```

```
    QString getW() const;
    QString getH() const;
    QString getA_X() const;
    QString getB_R() const;
    QString getC_R() const;
    QString getD_R() const;
    QString getE_Q() const;
    QString getF_Q() const;
    QString getAngle() const;
    M_Window1();
};

#endif // M_WINDOW1_H
```

A5. Исходный код файла **m_window1.cpp**

```
#include "m_window1.h"

M_Window1::M_Window1() : QDialog(0){

    W = new QLineEdit;
    H = new QLineEdit;
    A_X = new QLineEdit;
    B_R = new QLineEdit;
    C_R = new QLineEdit;
    D_R = new QLineEdit;
    E_Q = new QLineEdit;
    F_Q = new QLineEdit;
    Angle = new QLineEdit;

    QLabel* W_lable = new QLabel("Width");
    QLabel* H_lable = new QLabel("Height");
    QLabel* A_X_lable = new QLabel("A_X");
    QLabel* B_R_lable = new QLabel("B_R");
    QLabel* C_R_lable = new QLabel("C_R");
    QLabel* D_R_lable = new QLabel("D_R");
    QLabel* E_Q_lable = new QLabel("E_Q");
    QLabel* F_Q_lable = new QLabel("F_Q");
    QLabel* Angle_lable = new QLabel("Angle");

    make_change = new QPushButton("Change");
    cancel = new QPushButton("Cancel");

    connect(make_change, SIGNAL(clicked()), SLOT(accept()));
    connect(cancel, SIGNAL(clicked()), SLOT(reject()));

    QGridLayout* ptopLayout = new QGridLayout;
    ptopLayout->addWidget(W_lable, 0, 0);
    ptopLayout->addWidget(H_lable, 1, 0);
    ptopLayout->addWidget(A_X_lable, 2, 0);
    ptopLayout->addWidget(B_R_lable, 3, 0);
    ptopLayout->addWidget(C_R_lable, 4, 0);
    ptopLayout->addWidget(D_R_lable, 5, 0);
```

```
    ptopLayout->addWidget(E_Q_lable, 6, 0);
    ptopLayout->addWidget(F_Q_lable, 7, 0);
    ptopLayout->addWidget(Angle_lable, 8, 0);

    ptopLayout->addWidget(W, 0, 1);
    ptopLayout->addWidget(H, 1, 1);
    ptopLayout->addWidget(A_X, 2, 1);
    ptopLayout->addWidget(B_R, 3, 1);
    ptopLayout->addWidget(C_R, 4, 1);
    ptopLayout->addWidget(D_R, 5, 1);
    ptopLayout->addWidget(E_Q, 6, 1);
    ptopLayout->addWidget(F_Q, 7, 1);
    ptopLayout->addWidget(Angle, 8, 1);

    ptopLayout->addWidget(make_change, 10, 1);
    ptopLayout->addWidget(cancel, 11, 1);

    setLayout(ptopLayout);
}

QString M_Window1::getH() const{
    QString s = H->text();
    if (s == ""){
        return "0";
    }
    else if ( s[0] != '0'){
        int size_str = 0;
        for (int i = 0; i < s.size(); i++){
            if (s[i] >= '0' && s[i] <= '9'){
                size_str++;
            }
        }
        if (size_str == s.size()){
            return s;
        }
        else{
            return "0";
        }
    }
    return H->text();
}

QString M_Window1::getW() const{
    QString s = W->text();
    if (s == ""){
        return "0";
    }
    else if ( s[0] != '0'){
        int size_str = 0;
        for (int i = 0; i < s.size(); i++){
            if (s[i] >= '0' && s[i] <= '9'){
                size_str++;
            }
```

```
        }
        if (size_str == s.size()){
            return s;
        }
        else{
            return "0";
        }
    }
    return W->text();
}

QString M_Window1::getA_X() const{
    QString s = A_X->text();
    if (s == ""){
        return "0";
    }
    else if ( s[0] != '0'){
        int size_str = 0;
        for (int i = 0; i < s.size(); i++){
            if (s[i] >= '0' && s[i] <= '9'){
                size_str++;
            }
        }
        if (size_str == s.size()){
            return s;
        }
        else{
            return "0";
        }
    }
    return A_X->text();
}

QString M_Window1::getB_R() const{
    QString s = B_R->text();
    if (s == ""){
        return "0";
    }
    else if ( s[0] != '0'){
        int size_str = 0;
        for (int i = 0; i < s.size(); i++){
            if (s[i] >= '0' && s[i] <= '9'){
                size_str++;
            }
        }
        if (size_str == s.size()){
            return s;
        }
        else{
            return "0";
        }
    }
    return B_R->text();
```

```
}

QString M_Window1::getC_R() const{
    QString s = C_R->text();
    if (s == ""){
        return "0";
    }
    else if ( s[0] != '0'){
        int size_str = 0;
        for (int i = 0; i < s.size(); i++){
            if (s[i] >= '0' && s[i] <= '9'){
                size_str++;
            }
        }
        if (size_str == s.size()){
            return s;
        }
        else{
            return "0";
        }
    }
    return C_R->text();
}

QString M_Window1::getD_R() const{
    QString s = D_R->text();
    if (s == ""){
        return "0";
    }
    else if ( s[0] != '0'){
        int size_str = 0;
        for (int i = 0; i < s.size(); i++){
            if (s[i] >= '0' && s[i] <= '9'){
                size_str++;
            }
        }
        if (size_str == s.size()){
            return s;
        }
        else{
            return "0";
        }
    }
    return D_R->text();
}

QString M_Window1::getE_Q() const{
    QString s = E_Q->text();
    if (s == ""){
        return "0";
    }
    else if ( s[0] != '0'){
        int size_str = 0;
```

```
        for (int i = 0; i < s.size(); i++){
            if (s[i] >= '0' && s[i] <= '9'){
                size_str++;
            }
        }
        if (size_str == s.size()){
            return s;
        }
        else{
            return "0";
        }
    }
    return E_Q->text();
}

QString M_Window1::getF_Q() const{
    QString s = F_Q->text();
    if (s == ""){
        return "0";
    }
    else if ( s[0] != '0'){
        int size_str = 0;
        for (int i = 0; i < s.size(); i++){
            if (s[i] >= '0' && s[i] <= '9'){
                size_str++;
            }
        }
        if (size_str == s.size()){
            return s;
        }
        else{
            return "0";
        }
    }
    return F_Q->text();
}

QString M_Window1::getAngle() const{
    return Angle->text();
}
```

А6. Исходный код файла **m_window2.h**

```
#ifndef M_WINDOW2_H
#define M_WINDOW2_H
#include <QDialog>
#include <QLineEdit>
#include <QLabel>
#include <QPushButton>
#include <QGridLayout>

class M_Window2 :public QDialog{
```

```
    Q_OBJECT

private:
    QLineEdit* W;
    QLineEdit* H;
    QLineEdit* A_R;
    QLineEdit* B_X;
    QLineEdit* C_R;
    QLineEdit* D_X;
    QLineEdit* E_P;
    QLineEdit* F_Q;
    QLineEdit* Angle;
    QPushButton *make_change;
    QPushButton *cancel;

public:
    QString getW() const;
    QString getH() const;
    QString getA_R() const;
    QString getB_X() const;
    QString getC_R() const;
    QString getD_X() const;
    QString getE_P() const;
    QString getF_Q() const;
    QString getAngle() const;
    M_Window2();

};

#endif // M_WINDOW2_H
```

А7. Исходный код файла **m_window2.cpp**

```
#include "m_window2.h"

M_Window2::M_Window2(): QDialog(0){
    W = new QLineEdit;
    H = new QLineEdit;
    A_R = new QLineEdit;
    B_X = new QLineEdit;
    C_R = new QLineEdit;
    D_X = new QLineEdit;
    E_P = new QLineEdit;
    F_Q = new QLineEdit;
    Angle = new QLineEdit;

    QLabel* W_lable = new QLabel("Width");
    QLabel* H_lable = new QLabel("Height");
    QLabel* A_R_lable = new QLabel("A_R");
    QLabel* B_X_lable = new QLabel("B_X");
    QLabel* C_R_lable = new QLabel("C_R");
    QLabel* D_X_lable = new QLabel("D_X");
    QLabel* E_P_lable = new QLabel("E_P");
```

```
    QLabel* F_Q_lable = new QLabel("F_Q");
    QLabel* Angle_lable = new QLabel("Angle");

    make_change = new QPushButton("Change");
    cancel = new QPushButton("Cancel");

    connect(make_change, SIGNAL(clicked()), SLOT(accept()));
    connect(cancel, SIGNAL(clicked()), SLOT(reject()));

    QGridLayout* ptopLayout = new QGridLayout;
    ptopLayout->addWidget(W_lable, 0, 0);
    ptopLayout->addWidget(H_lable, 1, 0);
    ptopLayout->addWidget(A_R_lable, 2, 0);
    ptopLayout->addWidget(B_X_lable, 3, 0);
    ptopLayout->addWidget(C_R_lable, 4, 0);
    ptopLayout->addWidget(D_X_lable, 5, 0);
    ptopLayout->addWidget(E_P_lable, 6, 0);
    ptopLayout->addWidget(F_Q_lable, 7, 0);
    ptopLayout->addWidget(Angle_lable, 8, 0);

    ptopLayout->addWidget(W, 0, 1);
    ptopLayout->addWidget(H, 1, 1);
    ptopLayout->addWidget(A_R, 2, 1);
    ptopLayout->addWidget(B_X, 3, 1);
    ptopLayout->addWidget(C_R, 4, 1);
    ptopLayout->addWidget(D_X, 5, 1);
    ptopLayout->addWidget(E_P, 6, 1);
    ptopLayout->addWidget(F_Q, 7, 1);
    ptopLayout->addWidget(Angle, 8, 1);

    ptopLayout->addWidget(make_change, 10, 1);
    ptopLayout->addWidget(cancel, 11, 1);

    setLayout(ptopLayout);
}

QString M_Window2::getH() const{
    QString s = H->text();
    if (s == ""){
        return "0";
    }
    else if ( s[0] != '0'){
        int size_str = 0;
        for (int i = 0; i < s.size(); i++){
            if (s[i] >= '0' && s[i] <= '9'){
                size_str++;
            }
        }
        if (size_str == s.size()){
            return s;
        }
        else{
            return "0";
```

```
        }
    }
    return H->text();
}

QString M_Window2::getW() const{
    QString s = W->text();
    if (s == ""){
        return "0";
    }
    else if ( s[0] != '0'){
        int size_str = 0;
        for (int i = 0; i < s.size(); i++){
            if (s[i] >= '0' && s[i] <= '9'){
                size_str++;
            }
        }
        if (size_str == s.size()){
            return s;
        }
        else{
            return "0";
        }
    }
    return W->text();
}

QString M_Window2::getA_R() const{
    QString s = A_R->text();
    if (s == ""){
        return "0";
    }
    else if ( s[0] != '0'){
        int size_str = 0;
        for (int i = 0; i < s.size(); i++){
            if (s[i] >= '0' && s[i] <= '9'){
                size_str++;
            }
        }
        if (size_str == s.size()){
            return s;
        }
        else{
            return "0";
        }
    }
    return A_R->text();
}

QString M_Window2::getB_X() const{
    QString s = B_X->text();
    if (s == ""){
        return "0";
```

```
    }
    else if ( s[0] != '0'){
        int size_str = 0;
        for (int i = 0; i < s.size(); i++){
            if (s[i] >= '0' && s[i] <= '9'){
                size_str++;
            }
        }
        if (size_str == s.size()){
            return s;
        }
        else{
            return "0";
        }
    }
    return B_X->text();
}

QString M_Window2::getC_R() const{
    QString s = C_R->text();
    if (s == ""){
        return "0";
    }
    else if ( s[0] != '0'){
        int size_str = 0;
        for (int i = 0; i < s.size(); i++){
            if (s[i] >= '0' && s[i] <= '9'){
                size_str++;
            }
        }
        if (size_str == s.size()){
            return s;
        }
        else{
            return "0";
        }
    }
    return C_R->text();
}

QString M_Window2::getD_X() const{
    QString s = D_X->text();
    if (s == ""){
        return "0";
    }
    else if ( s[0] != '0'){
        int size_str = 0;
        for (int i = 0; i < s.size(); i++){
            if (s[i] >= '0' && s[i] <= '9'){
                size_str++;
            }
        }
        if (size_str == s.size()){
```

```
            return s;
        }
        else{
            return "0";
        }
    }
    return D_X->text();
}

QString M_Window2::getE_P() const{
    QString s = E_P->text();
    if (s == ""){
        return "0";
    }
    else if ( s[0] != '0'){
        int size_str = 0;
        for (int i = 0; i < s.size(); i++){
            if (s[i] >= '0' && s[i] <= '9'){
                size_str++;
            }
        }
        if (size_str == s.size()){
            return s;
        }
        else{
            return "0";
        }
    }
    return E_P->text();
}

QString M_Window2::getF_Q() const{
    QString s = F_Q->text();
    if (s == ""){
        return "0";
    }
    else if ( s[0] != '0'){
        int size_str = 0;
        for (int i = 0; i < s.size(); i++){
            if (s[i] >= '0' && s[i] <= '9'){
                size_str++;
            }
        }
        if (size_str == s.size()){
            return s;
        }
        else{
            return "0";
        }
    }
    return F_Q->text();
}
```

```
QString M_Window2::getAngle() const{
    return Angle->text();
}
```

A8. Исходный код файла **mainwindow.h**

```
#ifndef mainwindow_h
#define mainwindow_h
#include "figure.h"
#include <QtGui>
#include <QPushButton>
#include <QMainWindow>
#include  <QButtonGroup>
#include <unordered_map>

class MainWindow : public QMainWindow {

  Q_OBJECT

private:

     QButtonGroup buttons_group;
     QPushButton *m_figure1 = new QPushButton("Figure 1");
     QPushButton *m_figure2 = new QPushButton("Figure 2");
     QPushButton *m_add = new QPushButton("+");
     QPushButton *m_delete = new QPushButton("-");
     QTimer tmr;
     std::unordered_map<int, Figure*> widjets_f1;
     int index_1 = 0;
     std::unordered_map<int, Figure2*> widjets_f2;
     int index_2 = 0;

public:
    MainWindow(QWidget *p = 0);
    QToolBar* createToolBar();

public slots:
    void activate_del();
    void button_F1_pressed();
    void button_F2_pressed();
    void button_add_pressed();
    void delete_figure1();
    void delete_figure2();
};

#endif
```

А9. Исходный код файла **mainwindow.cpp**

```
#include "mainwindow.h"
#include "figure.h"
MainWindow::MainWindow(QWidget *p) : QMainWindow(p) {
    QPalette pal;
    pal.setColor(this->backgroundRole(), Qt::white);
    this->setPalette(pal);
    addToolBar(Qt::TopToolBarArea, createToolBar());
    connect(m_figure1, SIGNAL(clicked()), this,
SLOT(button_F1_pressed()));
    connect(m_figure2, SIGNAL(clicked()), this,
SLOT(button_F2_pressed()));
    connect(m_add, SIGNAL(clicked()), this,
SLOT(button_add_pressed()));
    connect(m_delete, SIGNAL(clicked()), this,
SLOT(delete_figure1()));
    connect(m_delete, SIGNAL(clicked()), this,
SLOT(delete_figure2()));
    connect(&tmr, SIGNAL(timeout()), this,
SLOT(activate_del()));
    tmr.start(1);
}

QToolBar* MainWindow::createToolBar() {
    QToolBar *ptb = new QToolBar();
    ptb->setMovable(false);

    m_figure1->setCheckable(true);
    m_figure2->setCheckable(true);
    m_add->setEnabled(false);
    m_delete->setEnabled(false);
    buttons_group.addButton(m_figure1);
    buttons_group.addButton(m_figure2);
    buttons_group.setExclusive(false);
    ptb->setFixedHeight(30);
    ptb->addWidget(m_figure1);
    ptb->addWidget(m_figure2);
    ptb->addWidget(m_add);
    ptb->addWidget(m_delete);
    return ptb;
}

void MainWindow::activate_del(){
    bool fl_ = false;
        for (const auto i : widjets_f2){
            if (i.second->get_status_pressed_2()){
                fl_ = true;
                break;
            }
        }
        for (const auto i :widjets_f1){
            if (i.second->get_status_pressed_1()){
```

```
                fl_ = true;
                break;
            }
        }
        if (fl_)
            m_delete->setEnabled(true);
        else
            m_delete->setEnabled(false);
}

void MainWindow::button_F1_pressed(){
    if (buttons_group.button(-2)->isChecked() == true ){
        if (buttons_group.button(-3)->isChecked() == true){
            m_figure2->setChecked(false);
        }
        m_add->setEnabled(true);
    }
    else{
        m_add->setEnabled(false);
    }
}

void MainWindow::button_F2_pressed(){
    if (buttons_group.button(-3)->isChecked() == true){
        if (buttons_group.button(-2)->isChecked() == true){
            m_figure1->setChecked(false);
        }
        m_add->setEnabled(true);
    }
    else{
        m_add->setEnabled(false);
    }
}

void MainWindow::button_add_pressed(){

    if (buttons_group.button(-2)->isChecked() == true){

        Figure *f1 = new Figure(this);
        widjets_f1[index_1] = f1;
        f1->show();
        m_figure2->setChecked(false);
        index_1++;

    }
    if (buttons_group.button(-3)->isChecked() == true){
        Figure2 *f2 = new Figure2(this);
        widjets_f2[index_2] = f2;
        f2->show();
        m_figure1->setChecked(false);
        index_2++;
    }
}
```

```
void MainWindow::delete_figure1(){

    if (m_delete->isEnabled()){
        std::vector<int> key;
        for (auto i: widjets_f1){
            if (i.second->get_status_pressed_1()){
                key.push_back(i.first);
                i.second->set_status_pressed_1(false);
                i.second->hide();
              }
        }
    for (auto i : key){
           widjets_f1.erase(i);
     }


    }
}

void MainWindow::delete_figure2(){

    if (m_delete->isEnabled()){
        std::vector<int> key;
        for (auto i: widjets_f2){
            if (i.second->get_status_pressed_2()){
                key.push_back(i.first);
                i.second->set_status_pressed_2(false);
                i.second->hide();
             }
        }
    for (auto i : key){
           widjets_f2.erase(i);

     }

    }
}
```