Министерство образования и науки Росссийской Федерации

Московский государственный институт электроники и математики

(технический университет)

Кафедра “Компьютерная безопасность”

ОТЧЕТ
К ЛАБОРАТОРНОЙ РАБОТЕ №4
по дисциплине
«Языки программирования

Работу выполнил
студент группы СКБ201 _____ Тур Т.В.
подпись, дата

Работу проверил _____ Булгаков С. А.
подпись, дата

Москва 2021

Содержание

Постановка задачи

Общее задание

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 описывающего их прямоугольника (по умолчанию против часовой стрелки). 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.

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** кнопкой мыши – открывается всплывающее меню. Меню должно содержать пункты «Удалить» и «Изменить».

При выборе пункта «Изменить» – 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.

Добавить в выпадающее меню кнопки «Переместить», начинающую процесс перетаскивания фигуры, а также кнопку «Повернуть», открывающее диалоговое окно с ползунком -180, 0, 180 градусов (текущий поворот фигуры отображается в реальном времени).

Создать выпадающее меню для основной части окна и добавить в него кнопки «Удалить все» и «Удалить пересекающиеся».

При перемещении фигур учитывать пересечения с другими фигурами. Добавить в выпадающее меню основной части окна кнопку «Уместить», изменяющую поворот и положение всех фигур, а при большом их количестве также и размер (на минимальное значение), так чтобы все фигуры разместились в окне.

Каждый студент выполняет лабораторную работу используя 2 фигуры. Первая фигура соответствует его номеру в общем списке, вторая на 10 больше. Пунктирные линии не отрисовываются. Студенты с номерами выше 80 выбирают первую фигуру как ***номер*** mod 80 +30, а вторую ***номер*** mod 80 +35. Список вариантов считать циклическим.

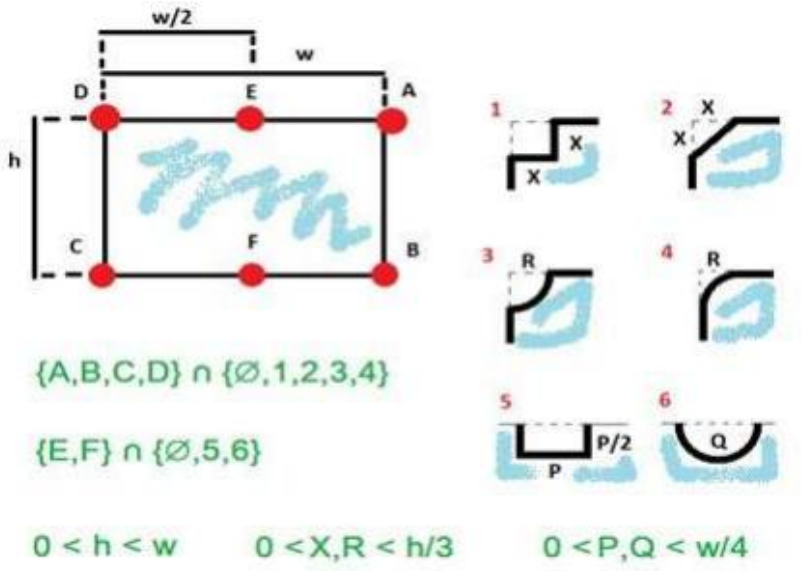
Параметры фигуры (1 – 6) изменяются независимо друг от друга. Например, фигура A1B1C1D1EF содержит 4 различных значения X (по одному на каждую точку).

Задание

Номер студента в общем списке: 105

Необходимые к реализации фигуры:

1. 55: A3B3C2D3EF6
2. 60: AB1C3D4E5F5



Варианты

№	Фигура	№	Фигура	№	Фигура	№	Фигура
1	A1B2CD1EF5	**21**	A1B2C3DEF6	**41**	AB4CD3EF	**61**	A2B2C2D3E6F5
2	A2BC1D1E5F5	**22**	A3B4C3D2EF6	**42**	ABC1D1E6F6	**62**	A3B4C3D3EF5
3	A1B1C4D1E6F	**23**	A3B2C4D3EF5	**43**	A2B1C4D2E5F	**63**	AB4C2D1E6F5
4	AB4C1DE5F5	**24**	A4B3C1D4E6F6	**44**	AB4CD4E5F	**64**	A3B3CD4EF
5	AB3C4DE5F6	**25**	A1B1C2D4EF6	**45**	A2B1CD1E5F	**65**	A1B3C2DE5F5
6	A3B3C3DE5F5	**26**	A2B2C2D3E5F6	**46**	AB2C3D1E5F	**66**	A1B3C1D4E6F
7	A2B1C2D2E6F5	**27**	A3B2C2D2E6F	**47**	AB4C3D4E5F6	**67**	A4B3C2D3E6F6
8	AB2C1D4E6F5	**28**	A3B3C1DE6F5	**48**	A3B2C1D3E5F	**68**	A2B4C4D3E6F6
9	A2B2CDE5F6	**29**	A3B2C4D2E5F	**49**	AB1C4D4EF5	**69**	AB2C3DE6F
10	AB4C2D4E6F5	**30**	A2B3C4D4E5F5	**50**	A3B3C2D3E6F6	**70**	A3B3C3D2E6F
11	AB4C4D4E5F6	**31**	A2BC1D1E6F5	**51**	A2BCD4EF5	**71**	A3B1CD3E6F6
12	A2B3C2D3EF6	**32**	A2BC3D2EF	**52**	A4B2C2D4EF	**72**	A4B3C4DE6F
13	A3B3C4D3EF5	**33**	A2B1C1D4E5F6	**53**	A3B3C3D2E5F6	**73**	AB3CD1E6F6
14	AB2C1D3E6F6	**34**	AB1C3D2EF5	**54**	A4B3C4D3EF	**74**	A3B4C3DEF
15	AB3C1D4EF6	**35**	A1B3C3D2E6F	**55**	A3B3C2D3EF6	**75**	AB4CD4E6F5
16	A3B4C1DE6F5	**36**	A3BC2D1E6F	**56**	AB2C1D2E5F	**76**	A2B3CD1E5F
17	A2B3C4D3E6F5	**37**	A4B3C1D3E5F6	**57**	A1B4C2D4E5F	**77**	AB1CDEF5
18	A3B1C2D4E6F6	**38**	A3B1C1D1EF5	**58**	A2B4C3D2E6F	**78**	A3B1C3D1E5F6
19	A1B4CD1EF5	**39**	A1B2C2DEF5	**59**	AB2C2D3E5F	**79**	A4B2C3DEF6
20	A4B1C4D4E5F	**40**	A4B4C3D2E6F	**60**	AB1C3D4E5F5	**80**	AB1C2D1EF5

Рисунок 1. Шаблоны фигур и варианты

Основная часть

Общая идея решения

Сперва написать файл (header) “mainwindow.h”, содержащий графический интерфейс и слоты основного окна выполнения программы. Написать файл (header) “figures.h”, содержащий основу класса фигур и 2 его наследника, для создания независимой формы фигуры, а так же классы диалоговых окон изменения параметров и поворота фигуры. Затем написать соответствующие реализации в файлы “mainwindow.cpp” и “figures.cpp”.

Решение задачи

Для решения задачи было разработано 6 классов: “MainWindow”, “EditDialog”, “RotateDialog”, “Figure”, “Figure1” и “Figure2”.

Класс “MainWindow” наслудется от “QMainWindow”, имеет 10 приватных полей: 4 для элементов меню типа “QAction*”; 1 для контекстного меню “QMenu*” и 3 для его элементов “QAction*”; массив динамического размера, содержащий в себе указатели на все фигуры, созданные в программе в момент, “ QVector<Figure*>”; и число, обозначающую активную фигуру в момент, “char”.

Класс “EditDialog” наследуется от “QDialog”, имеет 33 приватных поля: 9 “int” для обозначения изменяемых параметров фигуры; 2 “double” для площади и периметра фигуры; 1 “bool” для уточнения формы фигуры для правильных расчетов; по 9 “QLabel*” и “QSpinBox*”, по одному для каждого изменяемого параметра, и 2 дополнительных “QLabel*” для вывода площади и периметра; “QPushButton*” для кнопки подтверждения выбора параметров. Значения, допустимые для выбора ширины ограничены от 20 до 200, для высоты – от ширины/3 до ширины, для угловых точек – от 0 до высоты/3, для точек на ребрах – от 0 до ширины/4.

Класс “RotateDialog” наследуется от “QDialog” и имеет 3 приватных поля: “QLabel*” с текстом “Degree”, “QDial*” для задачи угла поворота круговым ползунком и “QPushButton*” с текстом “Accept”.

Класс “Figure” наследуется от “QWidget” и имеет 16 приватных полей: 9 “int” для ширины, высоты, параметров углов и ребер фигуры, угла поворота фигуры и размера виджета фигуры, 3 “bool”, для отслеживания, заблокирована ли фигура, двигается ли фигура и выбрана ли она, “QMenu*” для контекстного меню фигуры, “EditDialog*” и “RotateDialog*” для соответствующих модальных диалоговых окон.

Класс “Figure1” наследуется от “Figure”, дополнительных полей не имеет.

Класс “Figure2” наследуется от “Figure”, дополнительных полей не имеет.

Конструкторы

Конструктор “MainWindow” от одного “QWidget *” создает главное окно, согласно конструктору “QMainWindow”, указывает главное и контекстное меню и их кнопки, устанавливает размер окна в 800 на 623 пикселя.

Деструктор по умолчанию.

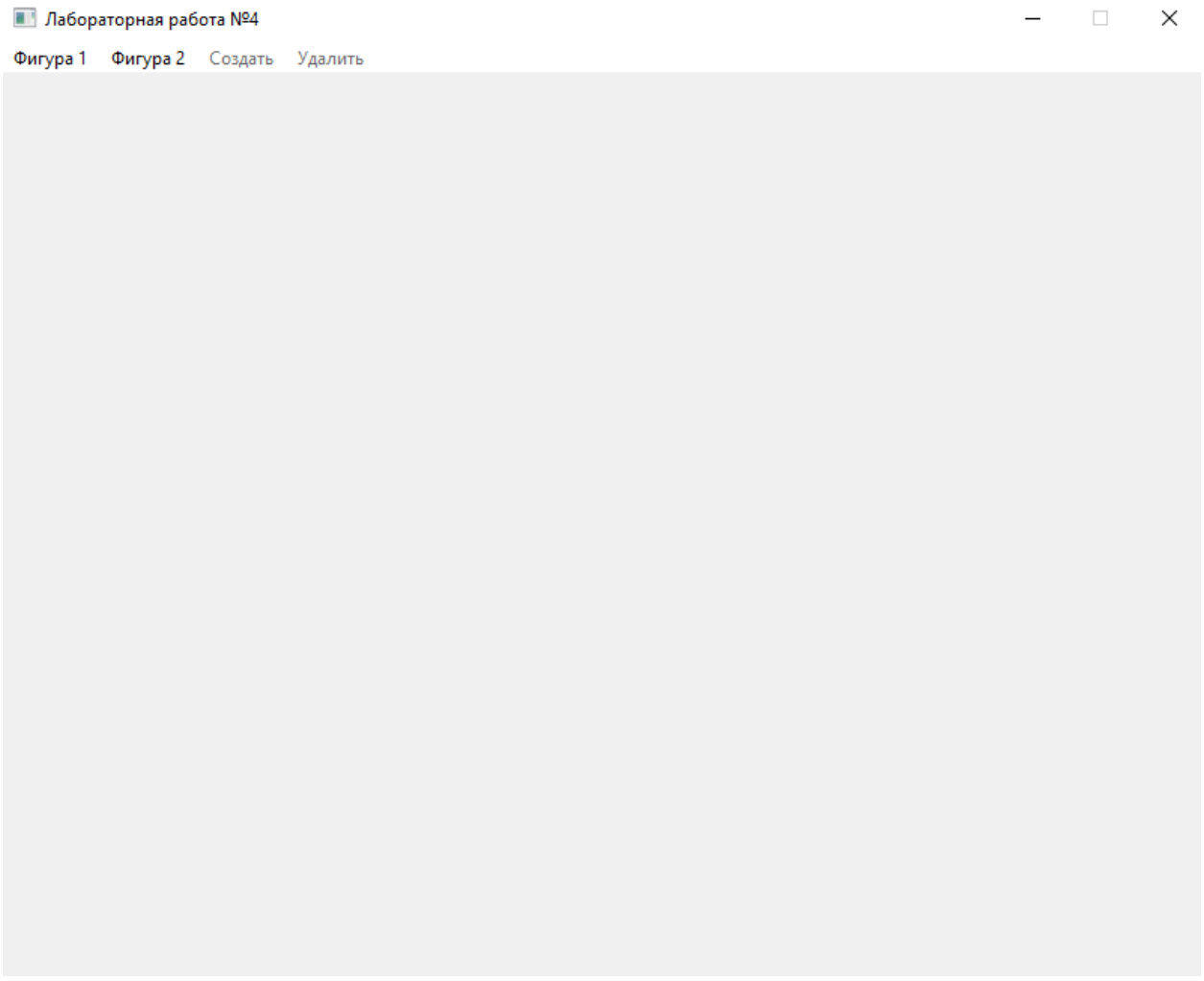


Рисунок 2 - Главное окно

Конструктор “EditDialog” от 9 “int”, “bool” и “QWidget*” создает диалоговое окно, согласно конструктору “QDialog”, указывает значения всех внутренних полей, в зависимости от вызывающей фигуры, а так же положение графический объектов.

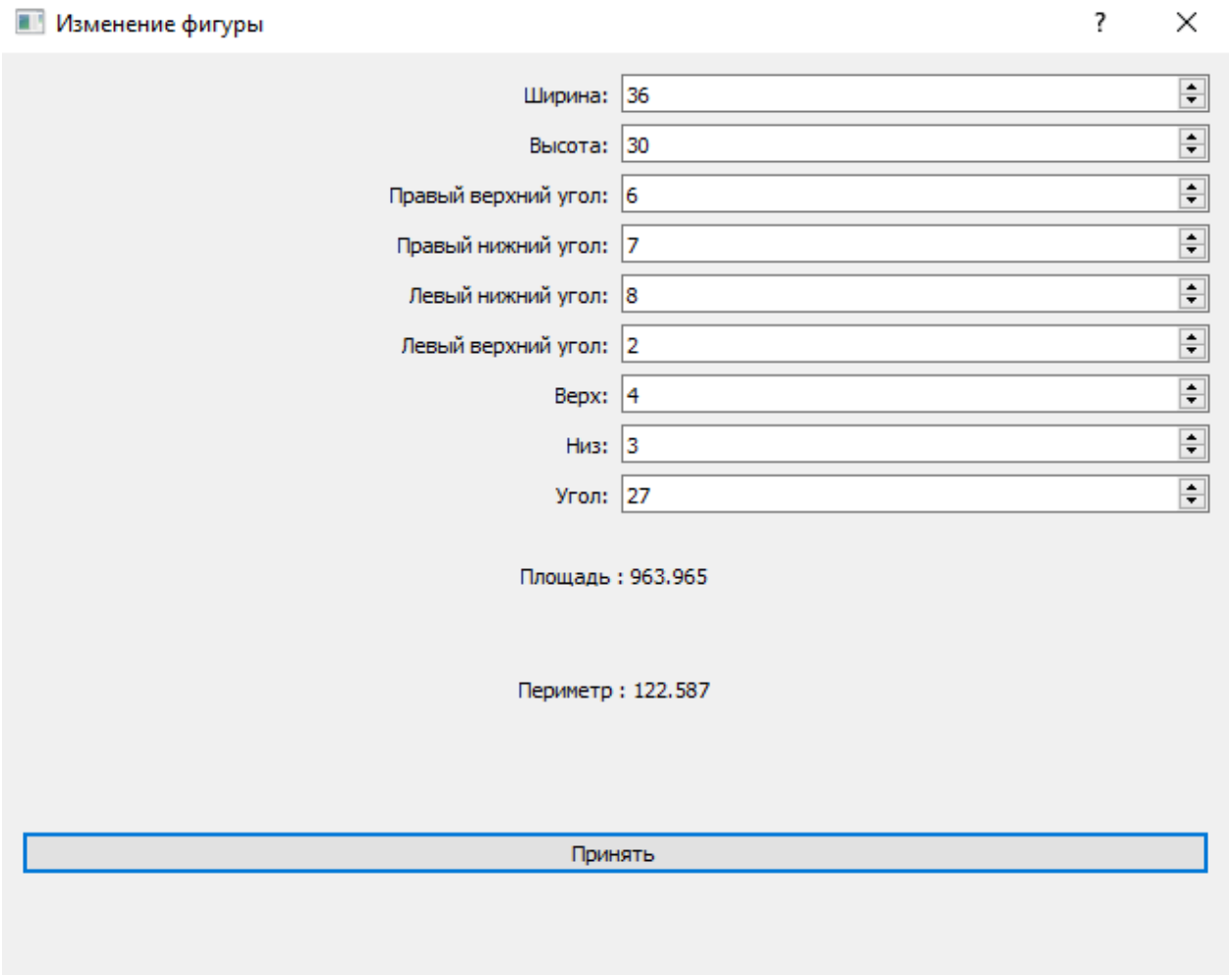


Рисунок 3 - Диалоговое окно изменения параметров фигуры

Конструктор “RotateDialog” от “int” и “ QWidget*” создает диалоговое окно, согласно конструктору “QDialog”, указывает значения всех внутренних полей, в зависимости от вызывающей фигуры, а так же положение графический объектов.

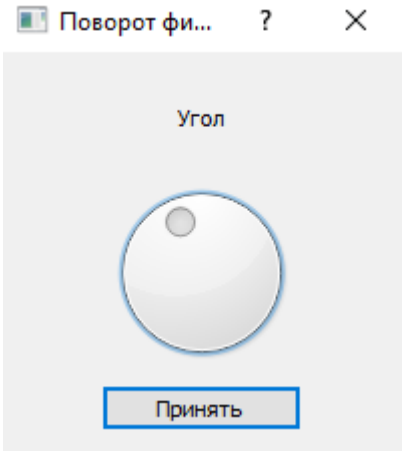


Рисунок 4. Диалоговое окно изменения поворота фигуры

Конструктор “Figure” от одного “QWidget *” создает виджет, согласно конструктору “QWidget”, указывает контекстное меню и его кнопки, выбирает случайные значения параметров, ширина ограничена от 20 до 200, высота – от ширины/3 до ширины, угол поворота ограничен от -180 до 180, остальные параметры ограничены согласно условиям задачи, значения всех “bool” задается как “false”, создается “RotateDialog*”. Если фигура при появлении находится полностью/частично в другой фигуре, то она блокирует свое перемещение.

Конструктор “Figure1” от одного “QWidget *” выполняет конструктор родителя, “Figure”, и создает диалоговое окно “ EditDialog*”, с указанием типа фигуры.

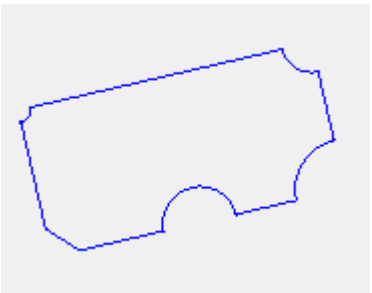


Рисунок 5. Форма фигуры 1

Конструктор “Figure2” от одного “QWidget *” выполняет конструктор родителя, “Figure”, и создает диалоговое окно “ EditDialog*”, с указанием типа фигуры.

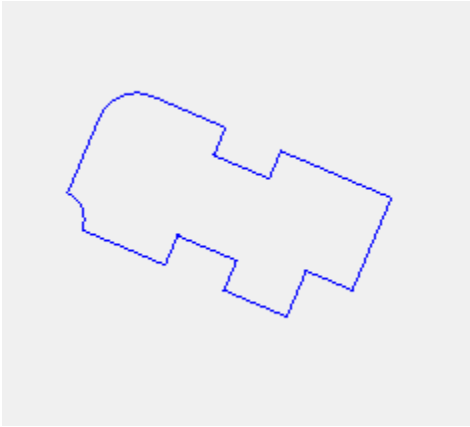


Рисунок 6. Форма фигуры 2

Методы и функционал “MainWindow”

Помимо унаследованного от “QMainWindow”, “MainWidow” не имеет ни одного публичного метода.

Кнопки меню “Создать” и “Удалить” поначалу не актины. “Фигура 1” и “ Фигура 2” изначально доступны, нажатие одной из них её заблокирует и разблокирует “Create”. Нажатие другой разблокирует первую. Кнопка “Удалить” доступна, когда выделена фигура. Если выделенной фигуры нет – кнопка “ Удалить ” заблокирована. В один момент времени может быть выделена только одна фигура, нажатие не по фигуре левой кнопной мыши удалит выделение. Нажатие кнопки “Создать” создаст выбранную фигуру в случайном месте. Нажатие “ Удалить ” удалит выделенную фигуру. Контекстное меню главного окна появляется при нажатии правой кнопки мыши не по фигуре. Оно содержит кнопки “Удалить все”, “Удалить пересекающиеся” и “Уместить”. Нажатие “ Удалить все ” удалит все существующие на экране фигуры. Нажатие “ Удалить пересекающиеся ” удалит все пересекающиеся фигуры. Нажатие “ Уместить ” уместит все фигуры на экране так, чтобы не было пересечений, а также разблокирует их перемещение.

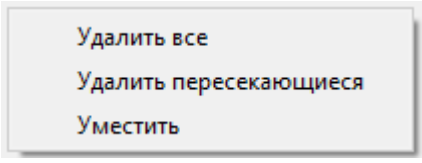


Рисунок 7. Контекстное меню главного окна

Методы и функционал “EditDialog”

Помимо унаследованного от “QDialog”, “EditDialog” имеет 9 геттеров: *getw, geth, geta, getb, getc, getd, gete, getf* и *getfi*, возвращающие значения соответствующих полей и 9 сеттеров: *setw, seth, seta, setb, setc, setd, sete, setf* и *setfi*, записывающих в соответствующие поля указанные значение и перемещающие счетчики.

В счетчики допустим ввод чисел вручную с клавиатуры или с помощью кнопок справа от соответствующего числа. Допустим ввод чисел, не выходящих за рамки допустимых значений. Изменение ширины и высоты изменит ограничения остальных счетчиков и изменит их значения, если они окажутся вне новых рамок. Текстовые поля площади и периметра изменяют свои значения динамически, вместе с изменениями параметров в диалоговом окне. Нажатие кнопки “Принять” сохранит выбранные параметры и передаст их фигуре, от которой класс был вызван, если та не заблокирована, и закроет окно. Закрытие окна через “крестик” не изменит параметры фигуры и вернет соответствующие параметры диалогового окна к изначальным.

Методы и функционал “RotateDialog”

Помимо унаследованного от “QDialog”, “RotateDialog” имеет геттер: *getFi*, возвращающий значение соответствующего поля, и сеттер *setFi*, записывающий в соответствующее поле указанное значение и перемещающий счетчик.

Поворот указателя ползунка меняет угол поворота фигуры сразу, если та не заблокирована. Кнопка “ Принять” закроет диалоговое окно.

Методы и функционал “Figure”

Помимо унаследованного от “QWidget”, “Figure” имеет геттеры: *isSelected, isBlocked* и *getForm*, возвращающие значения соответствующих полей, сеттеры: *deselect* – убирает выделение фигуры, *block* и *unblock* – блокируют и разблокировывают перемещение фигуры, а так же методы: *top, bottom, left* и *right,* возвращающие минимальную/максимальную координату по y/x фигуры внутри её виджета соответственно, и метод minimize, изменяющий параметры высоты и ширины до 20, устанавливающий поворот в 0 градусов и меняющий остальные параметры под их рамки.

Имеется внешний метод figuresColliding, который определяет, пересекается ли вторая указанная фигура с первой. Это уточнение имеет место быть, так как функция проверяет принадлежность некоторых “контрольных” точек второй фигуры первой, а не наоборот

Методы *top, bottom, left* и *right* и сеттер *getForm* являются виртуальными функциями, так как без формы фигуры, задаваемой наследниками, не могут иметь однозначных выводов.

Клик левой кнопкой мыши выделяет фигуру, снимает выделение с остальных фигур. Зажатие левой кнопки мыши начинает перемещение фигуры. Фигура не может переместиться дальше рамок главного окна, при столкновении с другой фигурой обе фигуры блокируются. Заблокированная фигура не может перемещаться.

Клик правой кнопкой мыши по фигуре открывает контекстное меню с опциями: “Удалить”, “Изменить”, “Переместить” и “Повернуть”. Нажатие “ Удалить” удаляет фигуру. Нажатие “ Изменить” открывает модальное диалоговое окнов “Изменени фигуры” соответствующей фигуры. Нажатие “ Переместить” начинает процесс отслеживания и перемещения к курсору мыши внутри рамок виджета. Следующее нажатие “ Переместить” отменит отслеживание и перемещение. Нажатие “ Повернуть” откроет модальное диалоговое окно “Поворот фигуры” соответствующей фигуры.

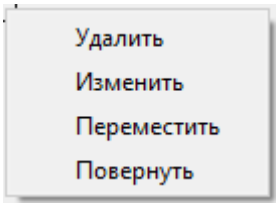


Рисунок 8. Контекстное меню фигуры

Методы и функционал “Figure1”

Помимо унаследованного от “Figure”, “Figure1” перегружает функции *getForm, top, bottom, left* и *right*, чтобы соответствовать свойствам свой формы. Также добавляет отрисовку фигуры и влияет на функции определения пересечения с точками.

Методы и функционал “Figure2”

Помимо унаследованного от “Figure”, “Figure2” перегружает функции *getForm, top, bottom, left* и *right*, чтобы соответствовать свойствам свой формы. Также добавляет отрисовку фигуры и влияет на функции определения пересечения с точками.

Сборка CMake

Проект компилируется с помощью системы сборки CMake. Файл конфигурации сгенерирован IDE “Qt Creator”. В нем установлен нужный компилятор и его версия,

установлены флаги для компиляции. Файл конфигурации CMake смотреть в Приложении Г.

Функция *main* в файле *main.cpp*

В функции *main* приведен пример создания разработанного окна MainWindow. Код с функцией *main* приведен в приложении В.

Приложение А – *mainwindow.h*

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QVector>

#include "figures.h"

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

protected:
    void contextMenuEvent(QContextMenuEvent* e);
    void mousePressEvent(QMouseEvent* e);

private slots:
    void deselect(Figure* figure);
    void moveFigure(Figure* figure, int dx, int dy);
    void select1();
    void select2();
    void create();
    void delSingle(Figure*);
    void delSelected();
    void callMousePress(QMouseEvent* e);
    void callContextMenu(QContextMenuEvent* e);
    void delAll();
    void delColliding();
    void fit();

private:
    QAction* _ActionFigure1;
    QAction* _ActionFigure2;
    QAction* _ActionCreate;
    QAction* _ActionDelete;

    QMenu* _BackgroundMenu;
    QAction* _ActionDeleteAll;
    QAction* _ActionDeleteColliding;
    QAction* _ActionFitAll;
    QVector<Figure*> figures;
    char selected;
};
#endif // MAINWINDOW_H
```

Приложение Б – *figures.h*

```
#ifndef FIGURES_H
#define FIGURES_H

#include <QWidget>
#include <QMenu>
#include <QDialog>
#include <QLabel>
#include <QSpinBox>
#include <QDial>

class EditDialog : public QDialog
{
    Q_OBJECT
public:
    EditDialog(int, int, int, int, int, int, int, int, int,
bool,
               QWidget *parent = nullptr);

    int getw(), geth(), geta(), getb(), getc(), getd(),
        gete(), getf(), getfi();
    void setw(int), seth(int), seta(int), setb(int), setc(int),
setd(int),
         sete(int), setf(int), setfi(int);

private slots:
    void wValueChanged(int val);
    void hValueChanged(int val);
    void aValueChanged(int val);
    void bValueChanged(int val);
    void cValueChanged(int val);
    void dValueChanged(int val);
    void eValueChanged(int val);
    void fValueChanged(int val);
    void fiValueChanged(int val);

private: // ыы
    int w, h,
        a, b, c, d, e, f,
        fi;
    double S, P;
    bool figureType; // 0 -> 1, 1 -> 2.
    QLabel* wLabel;
    QLabel* hLabel;
    QLabel* aLabel;
    QLabel* bLabel;
    QLabel* cLabel;
    QLabel* dLabel;
    QLabel* eLabel;
    QLabel* fLabel;
    QLabel* fiLabel;
```

```
    QLabel* SLabel;
    QLabel* PLabel;
    QSpinBox* wSpinBox;
    QSpinBox* hSpinBox;
    QSpinBox* aSpinBox;
    QSpinBox* bSpinBox;
    QSpinBox* cSpinBox;
    QSpinBox* dSpinBox;
    QSpinBox* eSpinBox;
    QSpinBox* fSpinBox;
    QSpinBox* fiSpinBox;
    QPushButton* acceptButton;
};

class RotateDialog : public QDialog
{
    Q_OBJECT
public:
    RotateDialog(int fi, QWidget* parent = nullptr);

    int getFi();
    void setFi(int);

private slots:
    void fiChanged(int);

private:
    QLabel* fiLabel;
    QDial* fiDial;
    QPushButton* acceptButton;

signals:
    void fiChangedSgn(int fi);
};

class Figure : public QWidget
{
    Q_OBJECT
public:
    explicit Figure(QWidget *parent = nullptr);

    void deselect();
    bool isSelected();
    virtual bool getForm() = 0;
    bool isBlocked();
    void block();
    void unblock();
    virtual int top() = 0;
    virtual int bottom() = 0;
    virtual int left() = 0;
    virtual int right() = 0;
    void minimize();
```

```
    friend bool figuresColliding(Figure*, Figure*, int, int);

protected:
    void contextMenuEvent(QContextMenuEvent* e);
    void mousePressEvent(QMouseEvent* e);
    void mouseReleaseEvent(QMouseEvent* e);
    void mouseMoveEvent(QMouseEvent* e);

    virtual bool collidingWithDot(QPoint p) = 0;
    virtual bool collidingWithDot(int x, int y) = 0;

    int w, h, s,
        a, b, c, d, e, f,
        fi;
    bool selected;
    bool lmHolds;
    QMenu* _FigureMenu;
    EditDialog* _EditDialog;
    RotateDialog* _RotateDialog;
    bool blocked;

private slots:
    void deleteFigure();
    void showFigureEdit();
    void figureChanged();
    void fiChanged(int val);
    void startMoving();
    void showFigureRotate();

private:
    QAction* _ActionFigureDelete;
    QAction* _ActionFigureEdit;
    QAction* _ActionFigureMove;
    QAction* _ActionFigureRotate;

signals:
    void selectedSgn(Figure*);
    void moveSgn(Figure*, int, int);
    void delSgn(Figure*);
    void contextMenuSgn(QContextMenuEvent*);
    void mousePressSgn(QMouseEvent*);
};

// фигура 55: A3B3C2D3EF6
class Figure1 : public Figure
{
    Q_OBJECT
public:
    explicit Figure1(QWidget *parent = nullptr);
    bool getForm();

    int top();
    int bottom();
```

```
    int left();
    int right();

protected:
    void paintEvent(QPaintEvent* event);

    bool collidingWithDot(QPoint p);
    bool collidingWithDot(int x, int y);
};

// фигура 60: AB1C3D4E5F5
class Figure2 : public Figure
{
    Q_OBJECT
public:
    explicit Figure2(QWidget *parent = nullptr);
    bool getForm();

    int top();
    int bottom();
    int left();
    int right();

protected:
    void paintEvent(QPaintEvent* event);

    bool collidingWithDot(QPoint p);
    bool collidingWithDot(int x, int y);
};

#endif // FIGURES_H
```

Приложение В – *main.cpp*

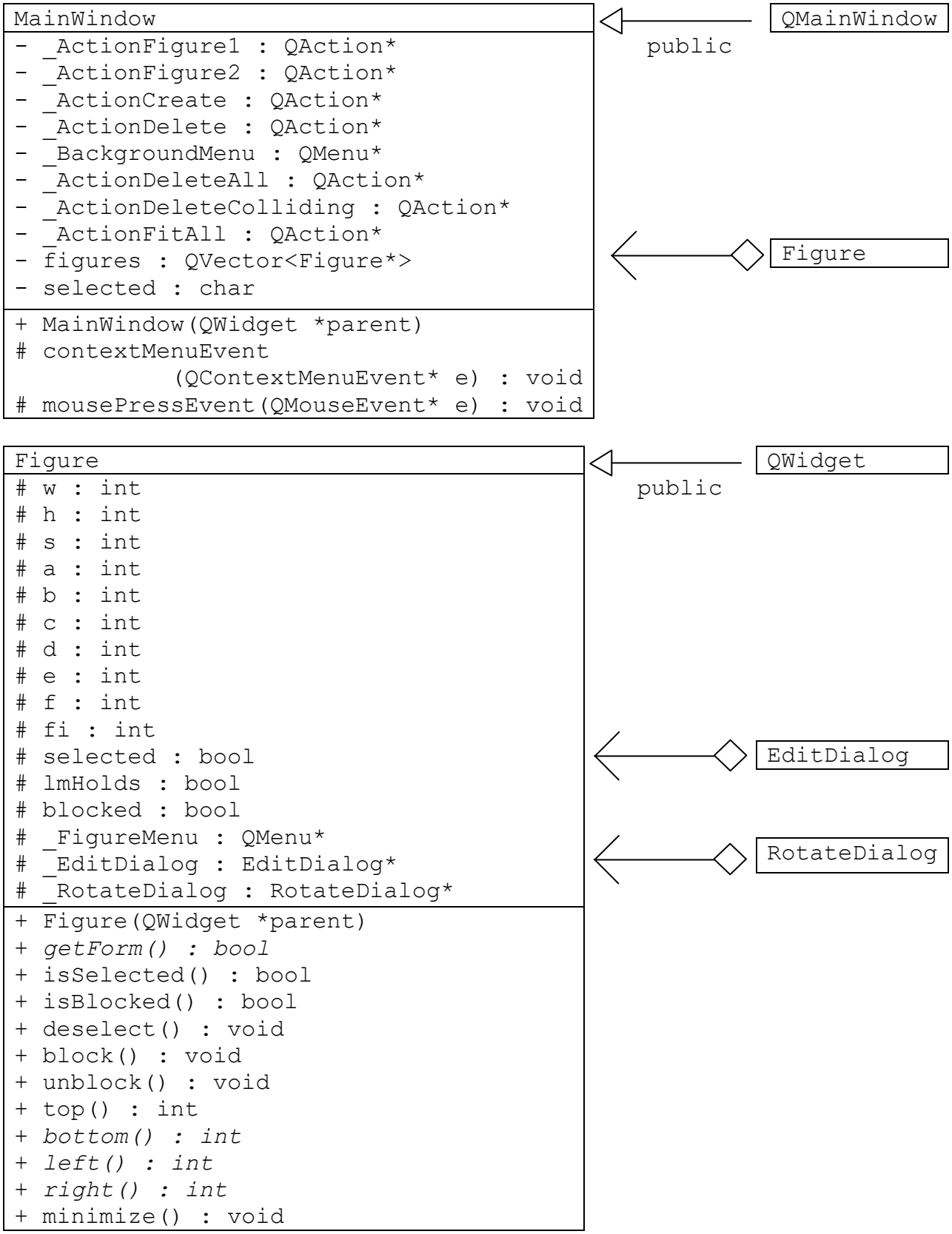
```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Приложение Г – CmakeLists.txt

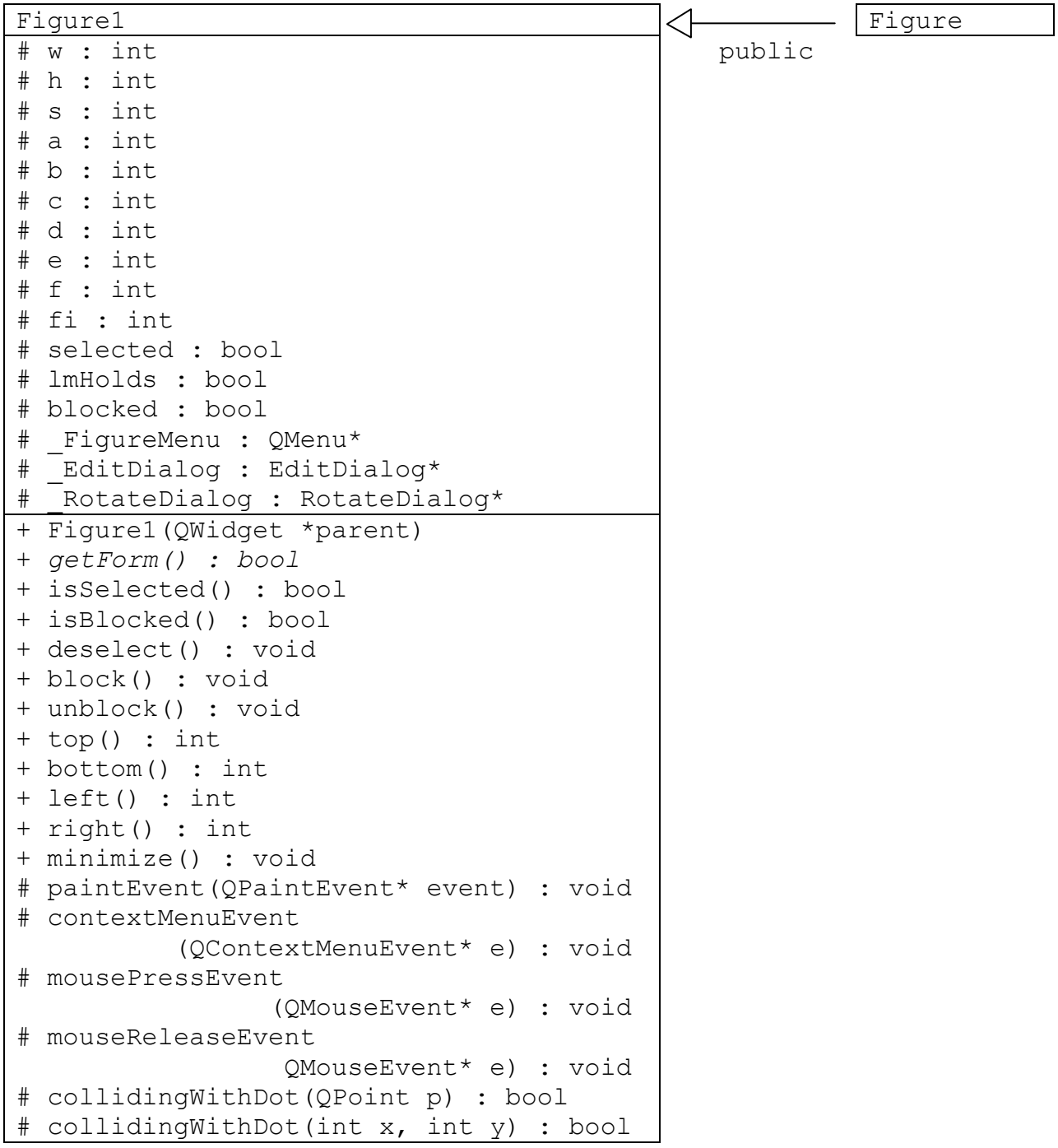
```
cmake_minimum_required(VERSION 3.5)

project(lw4 VERSION 0.1 LANGUAGES CXX)

set(CMAKE_INCLUDE_CURRENT_DIR ON)

set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)

set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(QT NAMES Qt6 Qt5 COMPONENTS Widgets REQUIRED)
find_package(Qt${QT_VERSION_MAJOR} COMPONENTS Widgets REQUIRED)

set(PROJECT_SOURCES
        main.cpp
        mainwindow.cpp
        mainwindow.h
        figures.h
        figures.cpp
)

if(${QT_VERSION_MAJOR} GREATER_EQUAL 6)
    qt_add_executable(lw4
        MANUAL_FINALIZATION
        ${PROJECT_SOURCES}
    )
# Define target properties for Android with Qt 6 as:
#    set_property(TARGET lw4 APPEND PROPERTY
QT_ANDROID_PACKAGE_SOURCE_DIR
#                 ${CMAKE_CURRENT_SOURCE_DIR}/android)
# For more information, see https://doc.qt.io/qt-6/qt-add-
executable.html#target-creation
else()
    if(ANDROID)
        add_library(lw4 SHARED
            ${PROJECT_SOURCES}
        )
# Define properties for Android with Qt 5 after find_package()
calls as:
#    set(ANDROID_PACKAGE_SOURCE_DIR
"${CMAKE_CURRENT_SOURCE_DIR}/android")
    else()
        add_executable(lw4
            ${PROJECT_SOURCES}
        )
    endif()
endif()
```

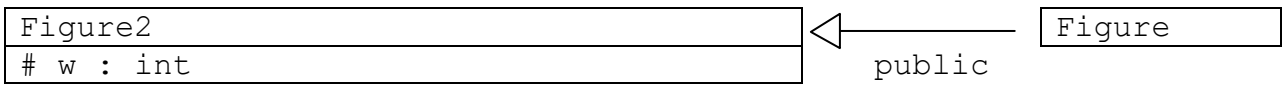
```
target_link_libraries(lw4 PRIVATE
Qt${QT_VERSION_MAJOR}::Widgets)

set_target_properties(lw4 PROPERTIES
    MACOSX_BUNDLE_GUI_IDENTIFIER my.example.com
    MACOSX_BUNDLE_BUNDLE_VERSION ${PROJECT_VERSION}
    MACOSX_BUNDLE_SHORT_VERSION_STRING
${PROJECT_VERSION_MAJOR}.${PROJECT_VERSION_MINOR}
)

if(QT_VERSION_MAJOR EQUAL 6)
    qt_finalize_executable(lw4)
endif()
```

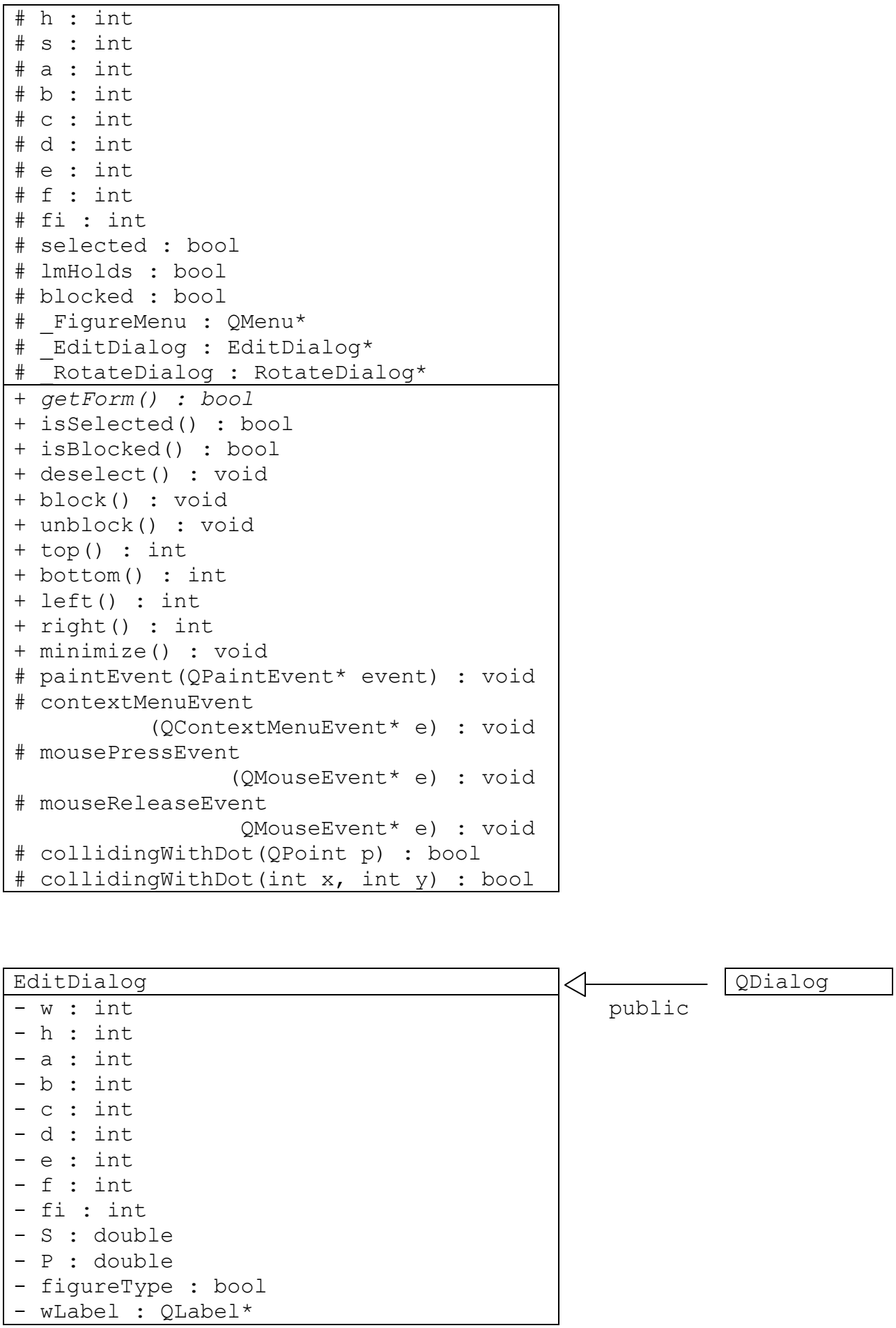
Приложение Д – UML 2.0 диаграмма классов

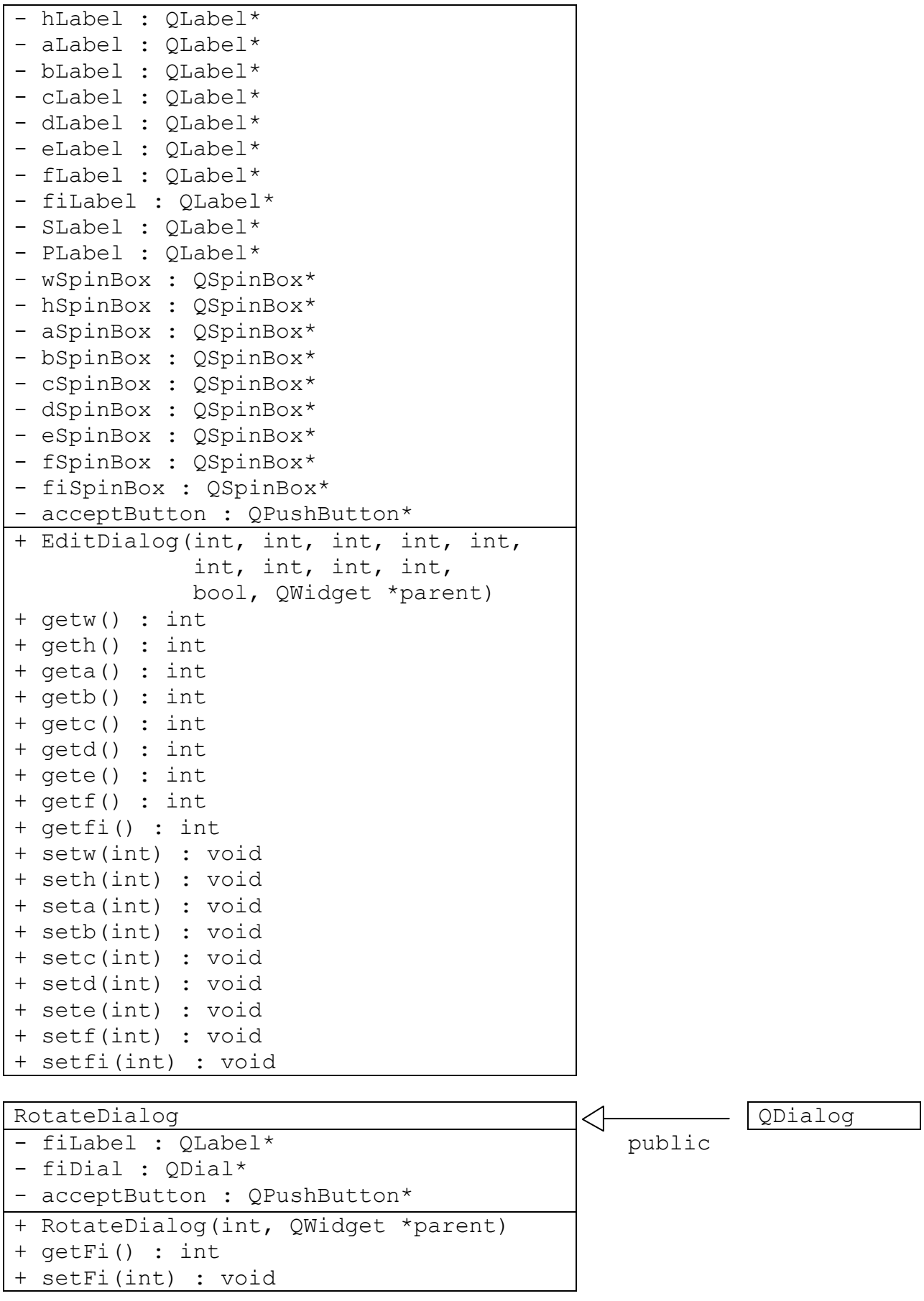


```
# contextMenuEvent
          (QContextMenuEvent* e) : void
# mousePressEvent
                 (QMouseEvent* e) : void
# mouseReleaseEvent
                  QMouseEvent* e) : void
# mouseMoveEvent(QMouseEvent* e) : void
# collidingWithDot(QPoint p) : bool
# collidingWithDot(int x, int y) : bool
```






h : int
s : int
a : int
b : int
c : int
d : int
e : int
f : int
fi : int
selected : bool
lmHolds : bool
blocked : bool
_FigureMenu : QMenu*
_EditDialog : EditDialog*
_RotateDialog : RotateDialog*
+ getForm() : bool
+ isSelected() : bool
+ isBlocked() : bool
+ deselect() : void
+ block() : void
+ unblock() : void
+ top() : int
+ bottom() : int
+ left() : int
+ right() : int
+ minimize() : void
# paintEvent(QPaintEvent* event) : void
# contextMenuEvent
(QContextMenuEvent* e) : void
mousePressEvent
(QMouseEvent* e) : void
mouseReleaseEvent
QMouseEvent* e) : void
collidingWithDot(QPoint p) : bool
collidingWithDot(int x, int y) : bool
EditDialog
- w : int
- h : int
- a : int
- b : int
- c : int
- d : int
- e : int
- f : int
- fi : int
- S : double
- P : double
- figureType : bool
- wLabel : QLabel*
public
QDialog


- hLabel : QLabel*
- aLabel : QLabel*
- bLabel : QLabel*
- cLabel : QLabel*
- dLabel : QLabel*
- eLabel : QLabel*
- fLabel : QLabel*
- fiLabel : QLabel*
- SLabel : QLabel*
- PLabel : QLabel*
- wSpinBox : QSpinBox*
- hSpinBox : QSpinBox*
- aSpinBox : QSpinBox*
- bSpinBox : QSpinBox*
- cSpinBox : QSpinBox*
- dSpinBox : QSpinBox*
- eSpinBox : QSpinBox*
- fSpinBox : QSpinBox*
- fiSpinBox : QSpinBox*
- acceptButton : QPushButton*
+ EditDialog(int, int, int, int, int,
int, int, int, int,
bool, QWidget *parent)
+ getw() : int
+ geth() : int
+ geta() : int
+ getb() : int
+ getc() : int
+ getd() : int
+ gete() : int
+ getf() : int
+ getfi() : int
+ setw(int) : void
+ seth(int) : void
+ seta(int) : void
+ setb(int) : void
+ setc(int) : void
+ setd(int) : void
+ sete(int) : void
+ setf(int) : void
+ setfi(int) : void
RotateDialog
- fiLabel : QLabel*
- fiDial : QDial*
- acceptButton : QPushButton*
+ RotateDialog(int, QWidget *parent)
+ getFi() : int
+ setFi(int) : void
public
QDialog

Приложение E – mainwindow.cpp

```
#include "mainwindow.h"
#include "figures.h"

#include <QMenuBar>
#include <QContextMenuEvent>
#include <QRandomGenerator>

#include <iostream>

//public:
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , selected(0)
{
    _ActionFigure1 = menuBar()->addAction(tr("&Фигура 1"),
                                          this,
SLOT(select1()));
    _ActionFigure2 = menuBar()->addAction(tr("&Фигура 2"),
                                          this,
SLOT(select2()));
    _ActionCreate = menuBar()->addAction(tr("&Создать"),
                                          this, SLOT(create()));
    _ActionDelete = menuBar()->addAction(tr("&Удалить"),
                                          this,
SLOT(delSelected()));
    _ActionCreate->setEnabled(false);
    _ActionDelete->setEnabled(false);

    _BackgroundMenu = new QMenu(this);
    _ActionDeleteAll =
            _BackgroundMenu->addAction(tr("&Удалить все"),
                                       this, SLOT(delAll()));
    _ActionDeleteColliding =
            _BackgroundMenu->addAction(tr("&Удалить
пересекающиеся"),
                                       this,
SLOT(delColliding()));
    _ActionFitAll =
            _BackgroundMenu->addAction(tr("&Уместить"),
                                       this, SLOT(fit()));
    setWindowTitle(tr("Лабораторная работа №4"));
    setFixedSize(800, 622);
}

MainWindow::~MainWindow() {}

//protected:
void MainWindow::contextMenuEvent(QContextMenuEvent* event) {
    _BackgroundMenu->exec(event->globalPos());
}
```

```
void MainWindow::mousePressEvent(QMouseEvent* event) {
    if(event->button() == Qt::LeftButton) {
        deselect(nullptr);
        _ActionDelete->setEnabled(false);
    }
}

//private slots:
void MainWindow::deselect(Figure* figure) {
    for(size_t i=0; i<figures.size(); ++i) {
        if(figures[i]!=figure)
            figures[i]->deselect();
    }
    if(figure != nullptr)
        _ActionDelete->setEnabled(true);
    update();
}
void MainWindow::moveFigure(Figure* figure, int dx, int dy) {
    if(not figure->isBlocked()) {
        int x = figure->geometry().left()+figure->left(),
            y = figure->geometry().top()+figure->top(),
            w = figure->right() - figure->left(),
            h = figure->bottom() - figure->top();
        x += dx; y += dy;
        if(x<0)
            x = 0;
        else if(x+w>800)
            x = 800-w;
        if(y<23)
            y = 23;
        else if(y+h>623)
            y = 623-h;
        bool toCheck;
        // hadooken.begin();
        for(int i=0; i<figures.size(); ++i) {
          toCheck = false;
          if(figures[i] != figure) {
            if(figure->geometry().left()>figures[i]-
>geometry().left() and
               figure->geometry().left()<figures[i]-
>geometry().right()) {
              if(figure->geometry().top()>figures[i]-
>geometry().top() and
                 figure->geometry().top()<figures[i]-
>geometry().bottom())
                toCheck = true;
              else if(
                 figure->geometry().bottom()>figures[i]-
>geometry().top() and
                 figure->geometry().bottom()<figures[i]-
>geometry().bottom())
                toCheck = true;
            }
```

```
            else if(
               figure->geometry().right()>figures[i]-
>geometry().left() and
               figure->geometry().right()<figures[i]-
>geometry().right()) {
              if(figure->geometry().top()>figures[i]-
>geometry().top() and
                  figure->geometry().top()<figures[i]-
>geometry().bottom())
                toCheck = true;
              else if(
                  figure->geometry().bottom()>figures[i]-
>geometry().top() and
                  figure->geometry().bottom()<figures[i]-
>geometry().bottom())
                toCheck = true;
            }
            if(toCheck) {
              if(figuresColliding(figure, figures[i],
                    figure->geometry().left()-figures[i]-
>geometry().left(),
                    figure->geometry().top()-figures[i]-
>geometry().top()) or
                  figuresColliding(figures[i], figure,
                    figures[i]->geometry().left()-figure-
>geometry().left(),
                    figures[i]->geometry().top()-figure-
>geometry().top())) {
                figure->block();
                figures[i]->block();
              }
            }
          }
        }
        //hadooken.end();
        if(not figure->isBlocked())
            figure->move(x-figure->left(), y-figure->top());
    }
}
void MainWindow::select1() {
    _ActionFigure1->setEnabled(false);
    _ActionFigure2->setEnabled(true);
    _ActionCreate->setEnabled(true);
    selected = 1;
}
void MainWindow::select2() {
    _ActionFigure1->setEnabled(true);
    _ActionFigure2->setEnabled(false);
    _ActionCreate->setEnabled(true);
    selected = 2;
}
void MainWindow::create() {
    Figure* f;
```

```
    if(selected==1)
        f = new Figure1(this);
    else if(selected==2)
        f = new Figure2(this);
    else
        return;
    f->move(QRandomGenerator::global()->bounded(500),
            QRandomGenerator::global()->bounded(23, 323));
    f->show();
    connect(f, SIGNAL(selectedSgn(Figure*)),
            this, SLOT(deselect(Figure*)));
    connect(f, SIGNAL(moveSgn(Figure*,int,int)),
            this, SLOT(moveFigure(Figure*,int,int)));
    connect(f, SIGNAL(delSgn(Figure*)),
            this, SLOT(delSingle(Figure*)));
    connect(f, SIGNAL(mousePressSgn(QMouseEvent*)),
            this, SLOT(callMousePress(QMouseEvent*)));
    connect(f, SIGNAL(contextMenuSgn(QContextMenuEvent*)),
            this, SLOT(callContextMenu(QContextMenuEvent*)));
    figures.push_back(f);

    bool toCheck;
    // hadooken.begin();
    for(int i=0; i<figures.size(); ++i) {
      toCheck = false;
      if(figures[i] != f) {
        if(f->geometry().left()>figures[i]->geometry().left()
and
           f->geometry().left()<figures[i]->geometry().right())
{
          if(f->geometry().top()>figures[i]->geometry().top()
and
             f->geometry().top()<figures[i]-
>geometry().bottom())
             toCheck = true;
          else if(
             f->geometry().bottom()>figures[i]->geometry().top()
and
             f->geometry().bottom()<figures[i]-
>geometry().bottom())
             toCheck = true;
        }
        else if(
           f->geometry().right()>figures[i]->geometry().left()
and
           f->geometry().right()<figures[i]->geometry().right())
{
          if(f->geometry().top()>figures[i]->geometry().top()
and
             f->geometry().top()<figures[i]-
>geometry().bottom())
             toCheck = true;
          else if(
```

```
             f->geometry().bottom()>figures[i]->geometry().top()
and
             f->geometry().bottom()<figures[i]-
>geometry().bottom())
            toCheck = true;
        }
        if(toCheck) {
          if(figuresColliding(f, figures[i],
                f->geometry().left()-figures[i]-
>geometry().left(),
                f->geometry().top()-figures[i]-
>geometry().top()) or
              figuresColliding(figures[i], f,
                figures[i]->geometry().left()-f-
>geometry().left(),
                figures[i]->geometry().top()-f-
>geometry().top())) {
            f->block();
            figures[i]->block();
          }
        }
      }
    }
    // hadooken.end();
    update();
}
void MainWindow::delSingle(Figure* figure) {
    int i = figures.indexOf(figure);
    delete figures[i];
    figures[i] = nullptr;
    figures.remove(i);
}
void MainWindow::delSelected() {
    for(int i=0; i<figures.size(); ++i) {
        if(figures[i]->isSelected()) {
            delete figures[i];
            figures[i] = nullptr;
            figures.remove(i);
        }
    }
    _ActionDelete->setEnabled(false);
    update();
}
void MainWindow::callMousePress(QMouseEvent* e) {
    mousePressEvent(e);
}
void MainWindow::callContextMenu(QContextMenuEvent* e) {
    contextMenuEvent(e);
}
void MainWindow::delAll() {
    int sz = figures.size();
    while(sz) {
        delete figures[sz-1];
```

```
        figures[sz-1] = nullptr;
        figures.pop_back();
        sz--;
    }
}
void MainWindow::delColliding() {
    int i=0;
    while(i!=figures.size()) {
        if(figures[i]->isBlocked()) {
            delete figures[i];
            figures[i] = nullptr;
            figures.remove(i);
        }
        else
            ++i;
    }
    update();
}
void MainWindow::fit() {
    int dx = 0, dy = 0;
    for(int i=0; i<figures.size(); ++i) {
        figures[i]->unblock();
        figures[i]->minimize();
        figures[i]->move(dx+1, dy+24);
        dx = figures[i]->geometry().right();
        if(dx > 800) {
            dx = 0;
            dy = figures[i]->geometry().bottom();
        }
    }
    update();
}
```

Приложение Ж – figures.cpp

```
#include "figures.h"

#include <QPainter>
#include <QContextMenuEvent>
#include <QRandomGenerator>
#include <QGridLayout>
#include <QPushButton>
#include <QtMath>

#include <iostream>

template<typename T>
T abs(T val) {
    if(val<0)
        return -val;
    return val;
}

// EditDialog
//public:
EditDialog::EditDialog(int w, int h, int a, int b, int c, int d,
int e, int f,
                       int fi, bool type, QWidget *parent)
    : QDialog(parent)
    , w(w), h(h), a(a), b(b), c(c), d(d), e(e), f(f),
figureType(type)
{
    setModal(true);

    wSpinBox = new QSpinBox;
    wSpinBox->setRange(20, 200);
    wSpinBox->setValue(w);
    wLabel = new QLabel(tr("&Ширина:"));
    wLabel->setBuddy(wSpinBox);

    hSpinBox = new QSpinBox;
    hSpinBox->setRange(w/3, w);
    hSpinBox->setValue(h);
    hLabel = new QLabel(tr("&Высота:"));
    hLabel->setBuddy(hSpinBox);

    aSpinBox = new QSpinBox;
    aSpinBox->setRange(0, h/3);
    aSpinBox->setValue(a);
    aLabel = new QLabel(tr("&Правый верхний угол:"));
    aLabel->setBuddy(aSpinBox);

    bSpinBox = new QSpinBox;
    bSpinBox->setRange(0, h/3);
    bSpinBox->setValue(b);
```

```
    bLabel = new QLabel(tr("&Правый нижний угол:"));
    bLabel->setBuddy(bSpinBox);

    cSpinBox = new QSpinBox;
    cSpinBox->setRange(0, h/3);
    cSpinBox->setValue(c);
    cLabel = new QLabel(tr("&Левый нижний угол:"));
    cLabel->setBuddy(cSpinBox);

    dSpinBox = new QSpinBox;
    dSpinBox->setRange(0, h/3);
    dSpinBox->setValue(d);
    dLabel = new QLabel(tr("&Левый верхний угол:"));
    dLabel->setBuddy(dSpinBox);

    eSpinBox = new QSpinBox;
    eSpinBox->setRange(0, w/4);
    eSpinBox->setValue(e);
    eLabel = new QLabel(tr("&Верх:"));
    eLabel->setBuddy(eSpinBox);

    fSpinBox = new QSpinBox;
    fSpinBox->setRange(0, w/4);
    fSpinBox->setValue(f);
    fLabel = new QLabel(tr("&Низ:"));
    fLabel->setBuddy(fSpinBox);

    fiSpinBox = new QSpinBox;
    fiSpinBox->setRange(-180, 180);
    fiSpinBox->setValue(-fi);
    fiLabel = new QLabel(tr("&Угол:"));
    fiLabel->setBuddy(fiSpinBox);

    if(figureType) {// 2
        S = w*h-b*b-f*f/2-0.785375*c*c-0.214625*d*d-e*e/2;
        P = 2*h+2*w+f+f-0.42925*(c+d);
    }
    else {// 1
        S = w*h-0.785375*a*a-0.785375*b*b-1.57075*f*f-c*c/2-
0.785375*d*d;
        P = 2*h+2*w-0.42925*(a+b+d)+0.57075*f-0.585786*c;
    }

    SLabel = new QLabel(tr("Площадь : ")+QString::number(S));
    PLabel = new QLabel(tr("Периметр : ")+QString::number(P));

    acceptButton = new QPushButton(tr("&Принять"), this);

    connect(wSpinBox, SIGNAL(valueChanged(int)),
            this,     SLOT(wValueChanged(int)));
    connect(hSpinBox, SIGNAL(valueChanged(int)),
            this,     SLOT(hValueChanged(int)));
    connect(aSpinBox, SIGNAL(valueChanged(int)),
```

```
            this,     SLOT(aValueChanged(int)));
    connect(bSpinBox, SIGNAL(valueChanged(int)),
            this,     SLOT(bValueChanged(int)));
    connect(cSpinBox, SIGNAL(valueChanged(int)),
            this,     SLOT(cValueChanged(int)));
    connect(dSpinBox, SIGNAL(valueChanged(int)),
            this,     SLOT(dValueChanged(int)));
    connect(eSpinBox, SIGNAL(valueChanged(int)),
            this,     SLOT(eValueChanged(int)));
    connect(fSpinBox, SIGNAL(valueChanged(int)),
            this,     SLOT(fValueChanged(int)));
    connect(fiSpinBox,SIGNAL(valueChanged(int)),
            this,     SLOT(fiValueChanged(int)));
    connect(acceptButton, SIGNAL(clicked()),
            this,         SLOT(accept()));

    QGridLayout* layout = new QGridLayout;
    layout->addWidget(wLabel,   0, 0, Qt::AlignRight);
    layout->addWidget(wSpinBox, 0, 1);
    layout->addWidget(hLabel,   1, 0, Qt::AlignRight);
    layout->addWidget(hSpinBox, 1, 1);
    layout->addWidget(aLabel,   2, 0, Qt::AlignRight);
    layout->addWidget(aSpinBox, 2, 1);
    layout->addWidget(bLabel,   3, 0, Qt::AlignRight);
    layout->addWidget(bSpinBox, 3, 1);
    layout->addWidget(cLabel,   4, 0, Qt::AlignRight);
    layout->addWidget(cSpinBox, 4, 1);
    layout->addWidget(dLabel,   5, 0, Qt::AlignRight);
    layout->addWidget(dSpinBox, 5, 1);
    layout->addWidget(eLabel,   6, 0, Qt::AlignRight);
    layout->addWidget(eSpinBox, 6, 1);
    layout->addWidget(fLabel,   7, 0, Qt::AlignRight);
    layout->addWidget(fSpinBox, 7, 1);
    layout->addWidget(fiLabel,  8, 0, Qt::AlignRight);
    layout->addWidget(fiSpinBox,8, 1);
    layout->addWidget(SLabel,   9, 0, 1, 2, Qt::AlignCenter);
    layout->addWidget(PLabel,  10, 0, 1, 2, Qt::AlignCenter);
    layout->addWidget(acceptButton, 11, 0, 2, 2);
    setLayout(layout);

    setFixedSize(aLabel->width(),
                 aLabel->height());

    setWindowTitle(tr("Изменение фигуры"));
}

int EditDialog::getw() {return w;}
int EditDialog::geth() {return h;}
int EditDialog::geta() {return a;}
int EditDialog::getb() {return b;}
int EditDialog::getc() {return c;}
int EditDialog::getd() {return d;}
int EditDialog::gete() {return e;}
```

```
int EditDialog::getf() {return f;}
int EditDialog::getfi() {return fi;}
void EditDialog::setw(int val) {
    w = val;
    wSpinBox->setValue(w);
}
void EditDialog::seth(int val) {
    h = val;
    hSpinBox->setValue(h);
}
void EditDialog::seta(int val) {
    a = val;
    aSpinBox->setValue(a);
}
void EditDialog::setb(int val) {
    b = val;
    bSpinBox->setValue(b);
}
void EditDialog::setc(int val) {
    c = val;
    cSpinBox->setValue(c);
}
void EditDialog::setd(int val) {
    d = val;
    dSpinBox->setValue(d);
}
void EditDialog::sete(int val) {
    e = val;
    eSpinBox->setValue(e);
}
void EditDialog::setf(int val) {
    f = val;
    fSpinBox->setValue(f);
}
void EditDialog::setfi(int val) {
    fi = val;
    fiSpinBox->setValue(-fi);
}

//private slots:
void EditDialog::wValueChanged(int val) {
    w = val;
    if(w>=20 and w<=200) { // ручной ввод
        if(h>=w)
            hSpinBox->setValue(w-1);
        else if(h<=w/3)
            hSpinBox->setValue(w/3);
        if(e>=w/4)
            eSpinBox->setValue(w/4-1);
        if(f>=w/4)
            fSpinBox->setValue(w/4-1);
        hSpinBox->setRange(w/3, w);
        eSpinBox->setMaximum(w/4);
```

```
        fSpinBox->setMaximum(w/4);
        if(figureType) {// 2
            S = w*h-b*b-f*f/2-0.785375*c*c-0.214625*d*d-e*e/2;
            P = 2*h+2*w+f+f-0.42925*(c+d);
        }
        else {// 1
            S = w*h-0.785375*a*a-0.785375*b*b-1.57075*f*f-c*c/2-
0.785375*d*d;
            P = 2*h+2*w-0.42925*(a+b+d)+0.57075*f-0.585786*c;
        }
        SLabel->setText(tr("Площадь : ")+QString::number(S));
        PLabel->setText(tr("Периметр : ")+QString::number(P));
    }
}
void EditDialog::hValueChanged(int val) {
    h = val;
    if(h>=w/3 and h<=w) {
        if(a>=h/3)
            aSpinBox->setValue(h/3-1);
        if(b>=h/3)
            bSpinBox->setValue(h/3-1);
        if(c>=h/3)
            cSpinBox->setValue(h/3-1);
        if(d>=h/3)
            dSpinBox->setValue(h/3-1);
        aSpinBox->setMaximum(h/3);
        bSpinBox->setMaximum(h/3);
        cSpinBox->setMaximum(h/3);
        dSpinBox->setMaximum(h/3);
        if(figureType) {// 2
            S = w*h-b*b-f*f/2-0.785375*c*c-0.214625*d*d-e*e/2;
            P = 2*h+2*w+f+f-0.42925*(c+d);
        }
        else {// 1
            S = w*h-0.785375*a*a-0.785375*b*b-1.57075*f*f-c*c/2-
0.785375*d*d;
            P = 2*h+2*w-0.42925*(a+b+d)+0.57075*f-0.585786*c;
        }
        SLabel->setText(tr("Площадь : ")+QString::number(S));
        PLabel->setText(tr("Периметр : ")+QString::number(P));
    }
}
void EditDialog::aValueChanged(int val) {
    a = val;
    if(figureType) {// 2
        S = w*h-b*b-f*f/2-0.785375*c*c-0.214625*d*d-e*e/2;
        P = 2*h+2*w+f+f-0.42925*(c+d);
    }
    else {// 1
        S = w*h-0.785375*a*a-0.785375*b*b-1.57075*f*f-c*c/2-
0.785375*d*d;
        P = 2*h+2*w-0.42925*(a+b+d)+0.57075*f-0.585786*c;
    }
```

```
    SLabel->setText(tr("Площадь : ")+QString::number(S));
    PLabel->setText(tr("Периметр : ")+QString::number(P));
}
void EditDialog::bValueChanged(int val) {
    b = val;
    if(figureType) {// 2
        S = w*h-b*b-f*f/2-0.785375*c*c-0.214625*d*d-e*e/2;
        P = 2*h+2*w+f+f-0.42925*(c+d);
    }
    else {// 1
        S = w*h-0.785375*a*a-0.785375*b*b-1.57075*f*f-c*c/2-
0.785375*d*d;
        P = 2*h+2*w-0.42925*(a+b+d)+0.57075*f-0.585786*c;
    }
    SLabel->setText(tr("Площадь : ")+QString::number(S));
    PLabel->setText(tr("Периметр : ")+QString::number(P));
}
void EditDialog::cValueChanged(int val) {
    c = val;
    if(figureType) {// 2
        S = w*h-b*b-f*f/2-0.785375*c*c-0.214625*d*d-e*e/2;
        P = 2*h+2*w+f+f-0.42925*(c+d);
    }
    else {// 1
        S = w*h-0.785375*a*a-0.785375*b*b-1.57075*f*f-c*c/2-
0.785375*d*d;
        P = 2*h+2*w-0.42925*(a+b+d)+0.57075*f-0.585786*c;
    }
    SLabel->setText(tr("Площадь : ")+QString::number(S));
    PLabel->setText(tr("Периметр : ")+QString::number(P));
}
void EditDialog::dValueChanged(int val) {
    d = val;
    if(figureType) {// 2
        S = w*h-b*b-f*f/2-0.785375*c*c-0.214625*d*d-e*e/2;
        P = 2*h+2*w+f+f-0.42925*(c+d);
    }
    else {// 1
        S = w*h-0.785375*a*a-0.785375*b*b-1.57075*f*f-c*c/2-
0.785375*d*d;
        P = 2*h+2*w-0.42925*(a+b+d)+0.57075*f-0.585786*c;
    }
    SLabel->setText(tr("Площадь : ")+QString::number(S));
    PLabel->setText(tr("Периметр : ")+QString::number(P));
}
void EditDialog::eValueChanged(int val) {
    e = val;
    if(figureType) {// 2
        S = w*h-b*b-f*f/2-0.785375*c*c-0.214625*d*d-e*e/2;
        P = 2*h+2*w+f+f-0.42925*(c+d);
    }
    else {// 1
```

```
        S = w*h-0.785375*a*a-0.785375*b*b-1.57075*f*f-c*c/2-
0.785375*d*d;
        P = 2*h+2*w-0.42925*(a+b+d)+0.57075*f-0.585786*c;
    }
    SLabel->setText(tr("Площадь : ")+QString::number(S));
    PLabel->setText(tr("Периметр : ")+QString::number(P));
}
void EditDialog::fValueChanged(int val) {
    f = val;
    if(figureType) {// 2
        S = w*h-b*b-f*f/2-0.785375*c*c-0.214625*d*d-e*e/2;
        P = 2*h+2*w+f+f-0.42925*(c+d);
    }
    else {// 1
        S = w*h-0.785375*a*a-0.785375*b*b-1.57075*f*f-c*c/2-
0.785375*d*d;
        P = 2*h+2*w-0.42925*(a+b+d)+0.57075*f-0.585786*c;
    }
    SLabel->setText(tr("Площадь : ")+QString::number(S));
    PLabel->setText(tr("Периметр : ")+QString::number(P));
}
void EditDialog::fiValueChanged(int val) {
    fi = -val;
}

//RotateDialog
RotateDialog::RotateDialog(int fi, QWidget* parent)
    : QDialog(parent)
{
    setModal(true);
    setFixedSize(200, 200);
    setWindowTitle(tr("Поворот фигуры"));

    fiDial = new QDial(this);
    fiDial->setFocusPolicy(Qt::StrongFocus);
    fiDial->setRange(-180, 180);
    fiDial->setSingleStep(1);
    fiDial->setValue(fi);
    fiDial->setFixedSize(100, 100);

    fiLabel = new QLabel(tr("&Угол"));
    fiLabel->setBuddy(fiDial);

    acceptButton = new QPushButton(tr("&Принять"), this);
    connect(acceptButton, SIGNAL(clicked()),
            this,         SLOT(accept()));

    QGridLayout* layout = new QGridLayout;
    layout->addWidget(fiLabel, 0, 0, Qt::AlignCenter);
    layout->addWidget(fiDial, 1, 0);
    layout->addWidget(acceptButton, 2, 0);
    setLayout(layout);
```

```
    connect(fiDial, SIGNAL(valueChanged(int)), this,
SLOT(fiChanged(int)));
}
int RotateDialog::getFi() {
    return fiDial->value();
}
void RotateDialog::setFi(int val) {
    fiDial->setValue(val);
}

//private slots:
void RotateDialog::fiChanged(int val) {
    emit fiChangedSgn(val);
}

// Figure
//public:
Figure::Figure(QWidget *parent)
    : QWidget(parent)
    , w(QRandomGenerator::global()->bounded(20, 200))
    , h(QRandomGenerator::global()->bounded(w/3, w))
    , s(qSqrt(w*w+h*h))
    , a(QRandomGenerator::global()->bounded(0, h/3))
    , b(QRandomGenerator::global()->bounded(0, h/3))
    , c(QRandomGenerator::global()->bounded(0, h/3))
    , d(QRandomGenerator::global()->bounded(0, h/3))
    , e(QRandomGenerator::global()->bounded(0, w/4))
    , f(QRandomGenerator::global()->bounded(0, w/4))
    , fi(QRandomGenerator::global()->bounded(-180, 180))
    , selected(false)
    , lmHolds(false)
    , blocked(false)
{
    setFixedSize(s, s);
    _FigureMenu = new QMenu(this);
    _ActionFigureDelete = _FigureMenu->addAction(tr("Удалить"),
                                                 this,

SLOT(deleteFigure()));
    _ActionFigureEdit = _FigureMenu->addAction(tr("Изменить"),
                                                 this,

SLOT(showFigureEdit()));
    _ActionFigureMove = _FigureMenu-
>addAction(tr("Переместить"),
                                               this,

SLOT(startMoving()));
    _ActionFigureRotate = _FigureMenu-
>addAction(tr("Повернуть"),
                                                 this,

SLOT(showFigureRotate()));
```

```
    _RotateDialog = new RotateDialog(fi, this);
    connect(_RotateDialog, SIGNAL(fiChangedSgn(int)),
            this,          SLOT(fiChanged(int)));
    update();
}

void Figure::deselect() {
    selected = false;
}
bool Figure::isSelected() {
    return selected;
}
bool Figure::isBlocked() {
    return blocked;
}
void Figure::block() {
    blocked = true;
    lmHolds = false;
    setMouseTracking(false);
}
void Figure::unblock() {
    blocked = false;
}
void Figure::minimize() {
    w = 20;
    h = 20;
    s = qSqrt(w*w+h*h);
    if(a>6)
        a = 6;
    if(b>6)
        b = 6;
    if(c>6)
        c = 6;
    if(d>6)
        d = 6;
    if(e>5)
        e = 5;
    if(f>5)
        f = 5;
    fi = 0;

    setFixedSize(s, s);
    update();
}

//protected:
void Figure::contextMenuEvent(QContextMenuEvent* e) {
    if(collidingWithDot(e->pos()))
        _FigureMenu->exec(e->globalPos());
    else
        emit contextMenuSgn(e);
}
void Figure::mousePressEvent(QMouseEvent* e) {
```

```
    if(collidingWithDot(e->pos())) {
        if(e->button()==Qt::LeftButton) {
            selected = true;
            emit selectedSgn(this);
            if(not blocked) {
                lmHolds = true;
                emit moveSgn(this, e->pos().x()-s/2, e-
>pos().y()-s/2);
            }
        }
        update();
    }
    else
        emit mousePressSgn(e);
}
void Figure::mouseMoveEvent(QMouseEvent* e) {
    if(not blocked) {
        if(hasMouseTracking() or lmHolds)
            emit moveSgn(this, e->pos().x()-s/2, e->pos().y()-
s/2);
        update();
    }
}
void Figure::mouseReleaseEvent(QMouseEvent* e) {
    if(not blocked) {
        if(e->button()==Qt::LeftButton)
            lmHolds = false;
    }
}

//private slots:
void Figure::deleteFigure() {
    emit delSgn(this);
}
void Figure::showFigureEdit() {
    _EditDialog->setw(w);
    _EditDialog->seth(h);
    _EditDialog->seta(a);
    _EditDialog->setb(b);
    _EditDialog->setc(c);
    _EditDialog->setd(d);
    _EditDialog->sete(e);
    _EditDialog->setf(f);
    _EditDialog->setfi(fi);
    _EditDialog->exec();
}
void Figure::figureChanged() {
    if(not blocked) {
        w = _EditDialog->getw();
        h = _EditDialog->geth();
        a = _EditDialog->geta();
        b = _EditDialog->getb();
        c = _EditDialog->getc();
```

```
        d = _EditDialog->getd();
        e = _EditDialog->gete();
        f = _EditDialog->getf();
        fi = _EditDialog->getfi();

        int news = qSqrt(w*w+h*h);
        setFixedSize(news, news);
        emit moveSgn(this, (s-news)/2, (s-news)/2);
        s = news;

        update();
    }
}
void Figure::fiChanged(int val) {
    if(not blocked) {
        fi = val;
        update();
    }
}
void Figure::startMoving() {
    if(not blocked) {
        if(hasMouseTracking()) {
            setMouseTracking(false);
        }
        else {
            setMouseTracking(true);
            QPoint p = this->mapFromGlobal(QCursor::pos());
            emit moveSgn(this, p.x()-s/2, p.y()-s/2);
        }
    }
}
void Figure::showFigureRotate() {
    _RotateDialog->setFi(fi);
    _RotateDialog->exec();
}

// Figure1
//public:
Figure1::Figure1(QWidget *parent) : Figure(parent) {
    _EditDialog = new EditDialog(w, h, a, b, c, d, e, f, fi,
false, this);
    connect(_EditDialog, SIGNAL(accepted()), this,
SLOT(figureChanged()));
}
bool Figure1::getForm() {
    return false;
}
int Figure1::top() {
    int ix[] = {
           d,                    w-a,
        0,                           w,
        0,                           w,
           c, w/2-f/2, w/2+f/2, w-b
```

```
    };
    int iy[] = {
             0,       0,
        d,               a,
        h-c,             h-b,
             h, h, h, h
    };
    int miny = s, dy;
    float radfi = qDegreesToRadians(static_cast<float>(fi));

    for(int i=0; i<10; ++i) {
        dy = ix[i]*qSin(radfi)+iy[i]*qCos(radfi)
                + s/2-w/2*qSin(radfi)-h/2*qCos(radfi);
        if(dy<miny)
            miny = dy;
    }
    return miny;
}
int Figure1::bottom() {
    int ix[] = {
           d,                   w-a,
        0,                           w,
        0,                           w,
           c, w/2-f/2, w/2+f/2, w-b
    };
    int iy[] = {
             0,       0,
        d,               a,
        h-c,             h-b,
             h, h, h, h
    };
    int maxy = 0, dy;
    float radfi = qDegreesToRadians(static_cast<float>(fi));

    for(int i=0; i<10; ++i) {
        dy = ix[i]*qSin(radfi)+iy[i]*qCos(radfi)
                + s/2-w/2*qSin(radfi)-h/2*qCos(radfi);
        if(dy>maxy)
            maxy = dy;
    }
    return maxy;
}
int Figure1::left() {
    int ix[] = {
           d,                   w-a,
        0,                           w,
        0,                           w,
           c, w/2-f/2, w/2+f/2, w-b
    };
    int iy[] = {
             0,       0,
        d,               a,
        h-c,             h-b,
```

```
            h, h, h, h
    };
    int minx = s, dx;
    float radfi = qDegreesToRadians(static_cast<float>(fi));

    for(int i=0; i<10; ++i) {
        dx = ix[i]*qCos(radfi)-iy[i]*qSin(radfi)
                + s/2-w/2*qCos(radfi)+h/2*qSin(radfi);
        if(dx<minx)
            minx = dx;
    }
    return minx;
}
int Figure1::right() {
    int ix[] = {
           d,                    w-a,
        0,                           w,
        0,                           w,
           c, w/2-f/2, w/2+f/2, w-b
    };
    int iy[] = {
            0,       0,
        d,               a,
        h-c,             h-b,
             h, h, h, h
    };
    int maxx = 0, dx;
    float radfi = qDegreesToRadians(static_cast<float>(fi));

    for(int i=0; i<10; ++i) {
        dx = ix[i]*qCos(radfi)-iy[i]*qSin(radfi)
                + s/2-w/2*qCos(radfi)+h/2*qSin(radfi);
        if(dx>maxx)
            maxx = dx;
    }
    return maxx;
}

//protected:
void Figure1::paintEvent(QPaintEvent* e) {
    QPainter painter(this);

    if(selected)
        painter.setPen(Qt::blue);
    else
        painter.setPen(Qt::black);
    painter.translate(s/2+
                      -
w/2*qCos(qDegreesToRadians(static_cast<float>(fi)))+

h/2*qSin(qDegreesToRadians(static_cast<float>(fi))),
                      s/2+
```

```
                        -
w/2*qSin(qDegreesToRadians(static_cast<float>(fi)))-

h/2*qCos(qDegreesToRadians(static_cast<float>(fi))));
    painter.rotate(fi);

    painter.drawArc(w-a, -a, 2*a, 2*a, 180*16, 90*16); // A
    painter.drawLine(w, a, w, h-b);
    painter.drawArc(w-b, h-b, 2*b, 2*b, 90*16, 90*16); // B
    painter.drawLine(w-b, h, w/2+f/2, h);
    painter.drawArc(w/2-f/2, h-f/2, f, f, 0, 180*16);  // F
    painter.drawLine(w/2-f/2, h, c, h);
    painter.drawLine(c, h, 0, h-c);                    // C
    painter.drawLine(0, h-c, 0, d);
    painter.drawArc(-d, -d, 2*d, 2*d, 270*16, 90*16);  // D
    painter.drawLine(d, 0, w-a, 0);                    // E

}
bool Figure1::collidingWithDot(QPoint p) {
    return collidingWithDot(p.x(), p.y());
}
bool Figure1::collidingWithDot(int x, int y) {
    float radfi = qDegreesToRadians(static_cast<float>(fi));
    x -= s/2+-w/2*qCos(radfi)+h/2*qSin(radfi);
    y -= s/2+ -w/2*qSin(radfi)-h/2*qCos(radfi);
    int t = x*qCos(radfi)+y*qSin(radfi);
    y = -x*qSin(radfi)+y*qCos(radfi);
    x = t;

    if(x<0 or x>w or y<0 or y>h)
        return false;

    if(x<d and y<d) { // D
        if(x*x+y*y>d*d)
            return true;
        return false;
    }
    //else if(x>w/2-e/2 and x<w/2+e/2 and y<e/2) { // E
    //   return true;
    //}
    else if(x>w-a and y<a) { // A
        int dx = x-w;
        if(dx*dx+y*y>a*a)
            return true;
        return false;
    }
    else if(x>w-b and y>h-b) { // B
        int dx = x-w, dy = y-h;
        if(dx*dx+dy*dy>b*b)
            return true;
        return false;
    }
    else if(x>w/2-f/2 and x<w/2+f/2 and y>h-f/2) { // F
```

```
        int dx = x-w/2, dy = y-h;
        if(dx*dx+dy*dy>f*f/4)
            return true;
        return false;
    }
    else if(x<c and y>h-c) { // C
        if(y-h+c<x)
            return true;
        return false;
    }
    return true;
}
// Figure2
//public:
Figure2::Figure2(QWidget *parent) : Figure(parent) {
    _EditDialog = new EditDialog(w, h, a, b, c, d, e, f, fi,
true, this);
    connect(_EditDialog, SIGNAL(accepted()), this,
SLOT(figureChanged()));
}
bool Figure2::getForm() {return true;}
int Figure2::top() {
    int ix[] = {
            d, w/2-e/2, w/2+e/2, w,
           static_cast<int>(0.61731656*d), // ободок
          static_cast<int>(0.29289321*d),
         static_cast<int>(0.07612046*d),
        0,
        0,                                  w,
           c, w/2-f/2, w/2+f/2, w-b
    };
    int iy[] = {
          0, 0, 0, 0,
        static_cast<int>(0.07612046*d), // ободок
       static_cast<int>(0.29289321*d),
      static_cast<int>(0.61731656*d),
     d,
     h-c,              h-b,
          h, h, h, h
    };
    int miny = s, dy;
    float radfi = qDegreesToRadians(static_cast<float>(fi));

    for(int i=0; i<14; ++i) {
        dy = ix[i]*qSin(radfi)+iy[i]*qCos(radfi)
                + s/2-w/2*qSin(radfi)-h/2*qCos(radfi);
        if(dy<miny)
            miny = dy;
    }
    return miny;
}
int Figure2::bottom() {
    int ix[] = {
```

```
            d, w/2-e/2, w/2+e/2, w,
           static_cast<int>(0.61731656*d), // ободок
          static_cast<int>(0.29289321*d),
         static_cast<int>(0.07612046*d),
        0,
        0,                              w,
           c, w/2-f/2, w/2+f/2, w-b
    };
    int iy[] = {
          0, 0, 0, 0,
        static_cast<int>(0.07612046*d), // ободок
       static_cast<int>(0.29289321*d),
      static_cast<int>(0.61731656*d),
     d,
     h-c,             h-b,
          h, h, h, h
    };
    int maxy = 0, dy;
    float radfi = qDegreesToRadians(static_cast<float>(fi));

    for(int i=0; i<14; ++i) {
        dy = ix[i]*qSin(radfi)+iy[i]*qCos(radfi)
                + s/2-w/2*qSin(radfi)-h/2*qCos(radfi);
        if(dy>maxy)
            maxy = dy;
    }
    return maxy;
}
int Figure2::left() {
    int ix[] = {
            d, w/2-e/2, w/2+e/2, w,
           static_cast<int>(0.61731656*d), // ободок
          static_cast<int>(0.29289321*d),
         static_cast<int>(0.07612046*d),
        0,
        0,                              w,
           c, w/2-f/2, w/2+f/2, w-b
    };
    int iy[] = {
          0, 0, 0, 0,
        static_cast<int>(0.07612046*d), // ободок
       static_cast<int>(0.29289321*d),
      static_cast<int>(0.61731656*d),
     d,
     h-c,             h-b,
          h, h, h, h
    };
    int minx = s, dx;
    float radfi = qDegreesToRadians(static_cast<float>(fi));

    for(int i=0; i<14; ++i) {
        dx = ix[i]*qCos(radfi)-iy[i]*qSin(radfi)
                + s/2-w/2*qCos(radfi)+h/2*qSin(radfi);
```

```
        if(dx<minx)
            minx = dx;
    }
    return minx;
}
int Figure2::right() {
    int ix[] = {
            d, w/2-e/2, w/2+e/2, w,
           static_cast<int>(0.61731656*d), // ободок
          static_cast<int>(0.29289321*d),
         static_cast<int>(0.07612046*d),
        0,
        0,                                  w,
           c, w/2-f/2, w/2+f/2, w-b
    };
    int iy[] = {
          0, 0, 0, 0,
        static_cast<int>(0.07612046*d), // ободок
       static_cast<int>(0.29289321*d),
      static_cast<int>(0.61731656*d),
     d,
     h-c,              h-b,
          h, h, h, h
    };
    int maxx = 0, dx;
    float radfi = qDegreesToRadians(static_cast<float>(fi));

    for(int i=0; i<14; ++i) {
        dx = ix[i]*qCos(radfi)-iy[i]*qSin(radfi)
                + s/2-w/2*qCos(radfi)+h/2*qSin(radfi);
        if(dx>maxx)
            maxx = dx;
    }
    return maxx;
}
//protevted:
void Figure2::paintEvent(QPaintEvent* event) {
    QPainter painter(this);

    if(selected)
        painter.setPen(Qt::blue);
    else
        painter.setPen(Qt::black);
    painter.translate(s/2+
                      -
w/2*qCos(qDegreesToRadians(static_cast<float>(fi)))+

h/2*qSin(qDegreesToRadians(static_cast<float>(fi))),
                      s/2+
                      -
w/2*qSin(qDegreesToRadians(static_cast<float>(fi)))-

h/2*qCos(qDegreesToRadians(static_cast<float>(fi))));
```

```
    painter.rotate(fi);

    painter.drawLine(w, 0, w, h-b);                    // A
    painter.drawLine(w,    h-b, w-b, h-b);             // B
    painter.drawLine(w-b, h-b, w-b, h   );             // B
    painter.drawLine(w-b, h, w/2+f/2, h);
    painter.drawLine(w/2+f/2, h,     w/2+f/2, h-f/2); // F
    painter.drawLine(w/2+f/2, h-f/2, w/2-f/2, h-f/2); // F
    painter.drawLine(w/2-f/2, h-f/2, w/2-f/2, h    ); // F
    painter.drawLine(w/2-f/2, h, c, h);
    painter.drawArc(-c, h-c, 2*c, 2*c, 0, 90*16);     // C
    painter.drawLine(0, h-c, 0, d);
    painter.drawArc(0, 0, 2*d, 2*d, 90*16, 90*16);    // D
    painter.drawLine(d, 0, w/2-e/2, 0);
    painter.drawLine(w/2-e/2, 0,   w/2-e/2, e/2);     // E
    painter.drawLine(w/2-e/2, e/2, w/2+e/2, e/2);     // E
    painter.drawLine(w/2+e/2, e/2, w/2+e/2, 0  );     // E
    painter.drawLine(w/2+e/2, 0, w, 0);               // A

}
bool Figure2::collidingWithDot(QPoint p) {
    return collidingWithDot(p.x(), p.y());
}
bool Figure2::collidingWithDot(int x, int y) {
    float radfi = qDegreesToRadians(static_cast<float>(fi));
    x -= s/2+-w/2*qCos(radfi)+h/2*qSin(radfi);
    y -= s/2+ -w/2*qSin(radfi)-h/2*qCos(radfi);
    int t = x*qCos(radfi)+y*qSin(radfi);
    y = -x*qSin(radfi)+y*qCos(radfi);
    x = t;


    if(x<0 or y<0 or x>w or y>h)
        return false;
    if(x<d and y<d) { // D
        int dx = x-d, dy = y-d;
        if(dx*dx+dy*dy<d*d)
            return true;
        return false;
    }
    else if(x>w/2-e/2 and x<w/2+e/2 and y<e/2) { // E
        return false;
    }
    //else if(x>w-a and y<a) { // A
    //    return true;
    //}
    else if(x>w-b and y>h-b) { // B
        return false;
    }
    else if(x>w/2-f/2 and x<w/2+f/2 and y>h-f/2) { // F
        return false;
    }
    else if(x<c and y>h-c) { // C
```

```
        int dy = y-h;
        if(x*x+dy*dy>c*c)
            return true;
        return false;
    }
    return true;
}

bool figuresColliding(Figure* f1, Figure* f2, int dx, int dy) {
    if(f2->getForm()) {  // 2
        int ix[] = {
               f2->d, f2->w/2-f2->e/2, f2->w/2+f2->e/2, f2->w,
               static_cast<int>(0.61731656*f2->d), // ободок
              static_cast<int>(0.29289321*f2->d),
             static_cast<int>(0.07612046*f2->d),
            0,
            0,
f2->w,
               f2->c, f2->w/2-f2->f/2, f2->w/2+f2->f/2, f2->w-
f2->b
        };
        int iy[] = {
                        0,     0,     0,     0,
            static_cast<int>(0.07612046*f2->d), // ободок
           static_cast<int>(0.29289321*f2->d),
          static_cast<int>(0.61731656*f2->d),
         f2->d,
         f2->h-f2->c,                                f2->h-f2->b,
                        f2->h, f2->h, f2->h, f2->h
        };
        float radfi = qDegreesToRadians(static_cast<float>(f2-
>fi));
        int addx = f2->s/2-f2->w/2*qCos(radfi)+f2-
>h/2*qSin(radfi);
        int addy = f2->s/2-f2->w/2*qSin(radfi)-f2-
>h/2*qCos(radfi);
        for(int i=0; i<14; ++i) {
            if(f1->collidingWithDot(
                        ix[i]*qCos(radfi)-iy[i]*qSin(radfi) - dx +
addx,
                        ix[i]*qSin(radfi)+iy[i]*qCos(radfi) - dy +
addy))
                 return true;
        }
        return false;
    }
    else { // 1
        int ix[] = {
               f2->d,                                       f2->w-
f2->a,
            0,
f2->w,
```

```
            0,
f2->w,
               f2->c, f2->w/2-f2->f/2, f2->w/2+f2->f/2, f2->w-
f2->b
        };
        int iy[] = {
                    0,                                  0,
            f2->d,                                         f2->a,
            f2->h-f2->c,                                   f2->h-f2-
>b,
                           f2->h, f2->h, f2->h, f2->h
        };
        float radfi = qDegreesToRadians(static_cast<float>(f2-
>fi));
        int addx = f2->s/2-f2->w/2*qCos(radfi)+f2-
>h/2*qSin(radfi);
        int addy = f2->s/2-f2->w/2*qSin(radfi)-f2-
>h/2*qCos(radfi);
        for(int i=0; i<10; ++i) {
            if(f1->collidingWithDot(
                       ix[i]*qCos(radfi)-iy[i]*qSin(radfi) - dx +
addx,
                       ix[i]*qSin(radfi)+iy[i]*qCos(radfi) - dy +
addy))
                return true;
        }
        return false;
    }
}
```