ФГБОУ ВПО «Воронежский государственный технический университет»
Кафедра компьютерных интеллектуальных технологий проектирования

325-2014

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам № 9-13 по дисциплине
“Объектно-ориентированное программирование”
для студентов направления 230100.62
«Информатика и вычислительная техника»
очной формы обучения

Воронеж 2014

Составители: канд. техн. наук А.Н. Юров,
канд. техн. наук М.В. Паринов,
ст. преп. В.А. Рыжков,
ст. преп. А.А. Филимонова

УДК 004.9

Методические указания к лабораторным работам № 9-13 по дисциплине “Объектно-ориентированное программ-мирование” для студентов направления 230100.62 «Информатика и вычислительная техника» очной формы обучения / ФГБОУ ВПО «Воронежский государственный технический университет»; сост. А.Н. Юров, М.В. Паринов, В.А. Рыжков, А.А. Филимонова. Воронеж, 2014. 41 с.

Методические указания содержат материал по созданию кроссплатформенных приложений в среде QT, а также практические задачи и перечень заданий для выполнения лабораторных работ по дисциплине «Объектно-ориентированное программирование».

Предназначены для студентов 1,2 курсов.

Методические указания подготовлены в электронном виде в тектовом редакторе MS Word 2010 и содержатся в файле МУ 2014_3.docx.

Ил. 12. Библиогр.: 10 назв.

Рецензент канд. физ.-мат. наук, доц. Н.А. Тюкачев

Ответственный за выпуск зав. кафедрой д-р техн. наук, проф. М.И. Чижов

Издается по решению редакционно-издательского совета Воронежского государственного технического университета

ВВЕДЕНИЕ

Большинство решений в виде пакетов для разработчика и инструментальных средств создания приложений имеют весьма развитый функционал для записи и проверки выражений, работу с графикой, набор готовых решений по сортировке данных и численных методах на алгоритмическом языке программирования и многое другое. Среда Qt с набором классов кроме вышеперечисленных возможностей, содержит дополнительные решения, позволяющие стоить эффективные кроссплатформенные приложения для решения вычислительных задач.

В методических указаниях представлен материал по созданию приложений в среде QT как для разработки программ в консольном режиме, так и с графическим интерфейсом с использованием QT SDK. Реализация проектов позволит усвоить концепции объектно-ориентированного программирования на практике. Все примеры могут быть апробированы на известных операционных системах: Windows, Linux, Mac OS X, Android и ряда других.

ЛАБОРАТОРНАЯ РАБОТА № 9
РАБОТА С ФАЙЛАМИ И ПАПКАМИ В QT

Цель работы: освоить операции по работе с файлами и директориями, используя решения QT.

Задачи и требования к выполнению:

1.Изучить структуру классов и методов QT, позволяющую организовать работу на файловом уровне.

2. Использовать готовые решения и единый подход с файлами и каталогами в разработке приложений.

Теоретические сведения

Работа с директориями (папками, в терминологии ОС Windows) и файлами — это та область, в которой не все операции являются платформонезависимыми, поэтому Qt предоставляет свою собственную поддержку этих операций. В графическом виде каталоги в браузере могут выглядеть так, как показано на рис. 1, а отображение их в составе приложения может быть организовано согласно структуре, показанной на рис. 2.

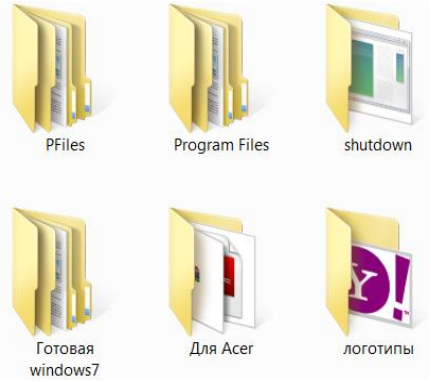

Рис. 1. Графический вид папок (каталогов) в браузере Windows

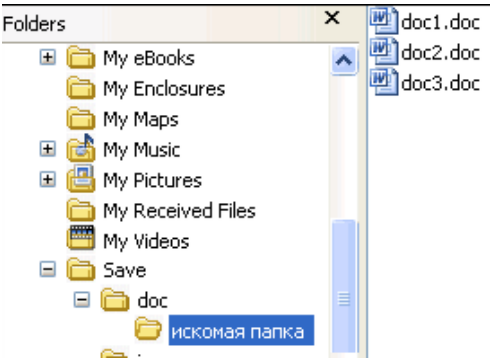

Рис. 2. Структура каталогов в приложении

Файлы делятся на два типа по их внутреннему содержимому: бинарные и текстовые. Текстовые файлы включают информацию, которую можно просмотреть и изменить любым текстовым редактором (например, блокнотом), как показано на рис. 3. К бинарным файлам относятся исполняемые файлы, которые обработаны некоторым компилятором и собраны компоновщиком с параметрами запуска под заданную операционную систему. Для файлов, кроме их содержимого, доступны режимы, установив которые, можно извлечь информацию о времени их создания, изменения, а также прочей служебной информации (рис. 4).

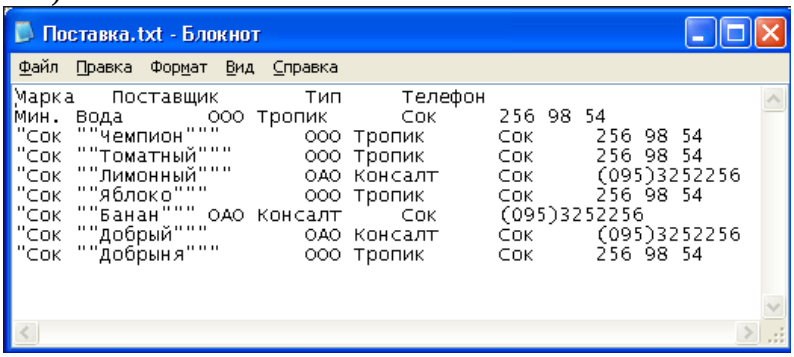

Рис. 3. Работа с текстовыми файлами

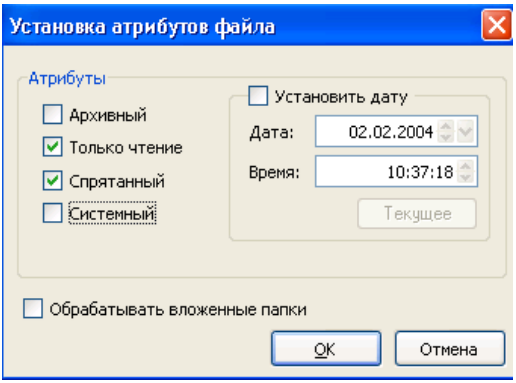

Рис. 4. Атрибуты файла

Для работы с файлами в проекте необходимо подключить класс QFile (#include <QFile>).

Для взаимодействия с файлами используются методы доступа к файлам:

QIODevice::ReadOnly // Открыть файл для чтения
QIODevice::WriteOnly // Открыть файл для записи
// (таким способом можно просто создать файл)
QIODevice::ReadWrite //Открыть файл для чтения и записи
QIODevice::Append //Открыть файл для дополнения файла в конец

На листинге 1 представлен пример записи данных в файл, а на рис. 5 показаны результаты работы программы.

Листинг 1. Запись строковых данных

```
QString temp="We learning QT in/out operation\n";
QString locale="Мы занимаемся работой на файловом
уровне";
QFile file("c://test.txt");
file.open(QIODevice::Append | QIODevice::Text);
QTextStream out(&file);
out << temp;
out << locale;
file.close();
QMessageBox *mes=new QMessageBox();
```

```
mes->setText("Операция выполнена!");
mes->exec();
```

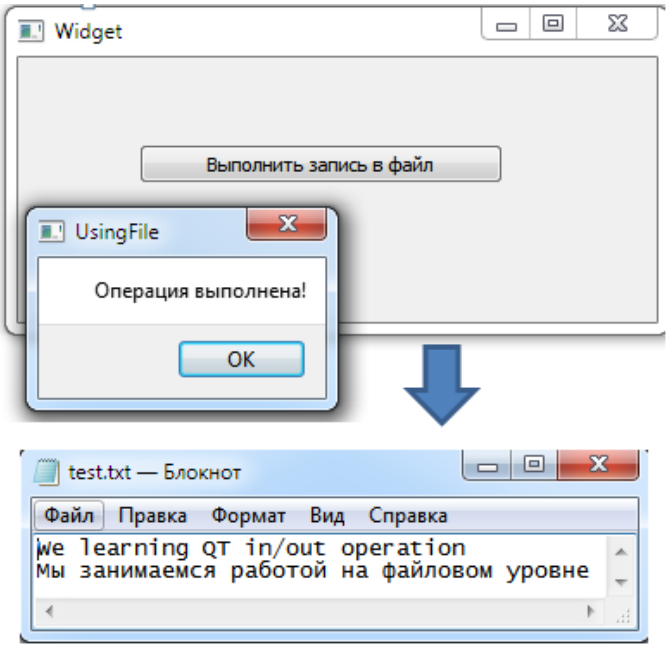

Рис. 5. Результаты работы приложения по записи данных на логический носитель информации

За получение информации о файлах отвечает класс QFileInfo. Задача этого класса состоит в предоставлении информации о свойствах файла, например: имя, размер, время последнего изменения, права доступа и т. д. Объект класса QFileInfo создается передачей в его конструктор пути к файлу, но можно передавать и объекты класса QFile. На листинге 2 представлен пример поиска файла (выполняется поиск приложения - калькулятор), а на рис. 6 показаны результаты работы программы с выводом в отладочном окне среды QT информационных сообщений.

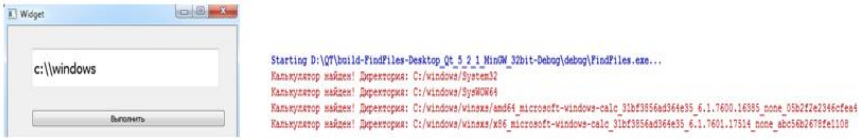

Рис. 6. Поиск файла calc.exe

Задания на самостоятельную работу:

Создать приложение, которое просматривает содержимое текстового файла, кроме того выводит информацию о файле: дату создания и тип файла (только для чтения, скрытый и т.д.). Для выбора файла необходимо воспользоваться стандартным диалогом открытия.

ЛАБОРАТОРНАЯ РАБОТА № 10
РАБОТА С ГРАФИЧЕСКИМИ ПРИМИТИВАМИ, ГОТОВЫЕ АЛГОРИТМИЧЕСКИЕ РЕШЕНИЯ В QT

Цель работы: изучить возможности по графическому выводу в Qt, ознакомиться с методами класса QtAlgorhms.

Задачи и требования к выполнению:

1.Научиться использовать готовые методы по сортировке и поиску данных;

2. Ознакомиться с типовыми приемами вывода простейших графических примитивов.

Теоретические сведения

Использование готовых алгоритмов значительно сокращает время разработки приложения, тем самым позволяя сконцентрироваться на решении конкретной задачи. В Qt имеется класс QtAlgorithms, в котором реализованы

общеизвестные алгоритмы. Рассмотрим некоторые методы из указанного класса.

qSort - Сортирует элементы в возрастающем порядке с помощью алгоритма быстрой сортировки. В Qt есть три варианта сортировки, которые представлены фрагментом листинга 2.

Листинг 2. Сортировка данных

```
//Пример использования первого способа:
QList<int> list;
list << 2 << 4 << 1 << 0 << 3;
qSort(list); // list: [ 0, 1, 2, 3, 4 ]

//Пример использования второго способа:
QList<int> list;
list << 2 << 4 << 1 << 0 << 3;
qSort(list.begin(), list.end()); // list: [ 0, 1,
2, 3, 4 ]

//Пример использования третьего способа
основанного на //применении перегруженной
функции:
bool mySort(const QString &s1, const QString &s2)
{
    return s1.toLower() < s2.toLower();
}
int doSomething()
{
    QStringList list;
    list << "One" << "Two" << "Four" << "Three";
    qSort(list.begin(), list.end(), mySort);
    // list: [ "One", "Two", "Three", "Four" ]
}
```

qFill – заполняет данными список в диапазоне: begin, end. В этом методе используют 2 варианта заполнения, как показано на примерах.

Пример первого варианта:

```
    QVector<QString> vect(4);
    qFill(vect, "hello world"); // vect: ["hello world", "hello world", "hello world", "hello world"]
```

Пример второго способа:

```
    QVector<QString> vect(4); // указываем диапазон самостоятельно
    qFill(vect.begin()+1, vect.end()-1, "hello world");
    // vect: ["", "hello world", "hello world", ""]
```

qFind – возвращает итератор на первое вхождение значения в контейнере в диапазоне: begin(), end(). Возвращает end(), если значение не найдено. Поиск может применяться двумя способами.

Пример использования первого варианта:

```
    QStringList list;
    list << "one" << "two" << "three";
    QStringList::iterator itr = qFind(list.begin(), list.end(), "two"); // itr == list.begin() + 1
    QStringList::iterator itr = qFind(list.begin(), list.end(), "nine"); // irt == list.end()
```

Пример использования второго варианта:

```
    QList<int> list;
    list << 2 << 13 << 47 << 234;
    QList<int>::const_iterator itr = qFind(list, 47);
    if (itr != list.end()) { QMessageBox::information(this, "Message", "Found: " + QString::number (*itr); }
    else { QMessageBox::information(this, "Message", "Not Found"); }
```

Функциональность остальных методов класса, реализующие типовые алгоритмы, можно узнать, обратившись к документации по Qt.

Графические возможности Qt по отображению геометрических примитивов на плоскости вынесены в класс QPainter. Дополнительные решения, которые реализуют выбор пера, определяют толщину линии, выполнены в отдельных классах.

Класс QPainter выполняет низкоуровневое рисование на виджетах и других устройствах рисования.

QPen – перо, определяет тип линии.

QPoint – класс определяет точку на плоскости, используя целочисленную точность, иными словами это координаты точки. На листинге 3 представлено приложение, в котором показано, как отобразить отрезок и выделить начальную и конечную точки этого отрезка.

Листинг 3. Графический вывод отрезка и точек на QT

```
.h файл
#ifndef DIALOG_H
#define DIALOG_H
#include <QDialog>
namespace Ui {
class Dialog;
}
```

```
class Dialog : public QDialog
{
    Q_OBJECT

public:
    explicit Dialog(QWidget *parent = 0);
    ~Dialog();

private:
    Ui::Dialog *ui;
// отрисовка формы начинается в этой функции
protected:
    void paintEvent(QPaintEvent *);
};
```

.cpp файл

```
#include "dialog.h"
#include "ui_dialog.h"
#include <QtCore>
#include <QtGui>
Dialog::Dialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Dialog)
{
    ui->setupUi(this);
}
Dialog::~Dialog()
{
    delete ui;
}
void Dialog::paintEvent(QPaintEvent *)
{
    QPainter painther(this);
    // создаем кисть(черного цвета)
    QPen penB(Qt::black);
    // ширина кисти в пикселях
    penB.setWidth(5);
    // создаем кисть(красного цвета)
    QPen penR(Qt::red);
```

```
    // ширина кисти в пикселях
    penR.setWidth(2);
    //    создаем    переменную    для    хранения
координат(начальная точка)
    QPoint p1;

 // x
    p1.setX(50);
    // y
    p1.setY(70);
    // вторая точка (конечная точка)
    QPoint p2;
    // x
    p2.setX(170);
    // y
    p2.setY(220);
    //   передаем   нашему   рисовальщику   кисть
красного цвета
    painther.setPen(penR);
    // рисуем линию по координатам красным цветом
    painther.drawLine(p1,p2);

    // выбираем кисть черного цвета
    painther.setPen(penB);
    // рисуем "жирные" точки в координатах начали
и окончания линии
    painther.drawPoint(p1);
    painther.drawPoint(p2);
}
```

Для вывода текста в графическом режиме можно воспользоваться следующим кодом (фрагмент листинга 4), а результаты показаны на рис. 7.

Листинг 4. Вывод текста на QT

```
    //берем зеленую кисть
    painther.setPen(Qt::green);
```

```
  // устанавливаем размер шрифта
painther.setFont(QFont("Arial", 30));
// пишем по центру формы слово "Qt"
painther.drawText(rect(),Qt::AlignCenter,"Qt");
```

Рис. 7. Вывод текста в графическом контексте Qt

Еще один пример, который демонстрирует работу по графическому выводу в Qt, представлен фрагментом листинга 5, результатом которого является узор, полученный из набора точек (рис. 8).

Листинг 5. Графический вывод объекта по точкам в QT

```
#include <cmath>
void Widget::paintEvent(QPaintEvent *)
{
    int ArrayX[16],ArrayY[16];
    QPainter *paint=new QPainter(this);
    QPen *penB=new QPen(Qt::blue);
    penB->setWidth(3);
    QPoint *p=new QPoint();
    paint->setPen(*penB);

    int k;
    for (int i=1;i<=4;i++)
    {
        for (int j=1;j<=4;j++)
        {
            k=4*i+j-4;
            ArrayX[k]=j-3;
```

```
            ArrayY[k]=i-3;
        }
    }
    ArrayX[2]=0;
    ArrayY[2]=-3;
    ArrayX[8]=2;
    ArrayY[8]=0;
    ArrayX[9]=-3;
    ArrayY[9]=-1;
    ArrayX[15]=-1;
    ArrayY[15]=2;
    for (int i=1;i<=16;i++)
    {
        for (int j=1;j<=16;j++)
        {
            for (int l=1;l<=16;l++)
            {
                p-
>setX(100+16*ArrayX[i]+4*ArrayX[j]+ArrayX[k]);
                p-
>setY(100+16*ArrayY[i]+4*ArrayY[j]+ArrayY[k]);

                paint->drawPoint(*p);
            }
        }
    }
}
```

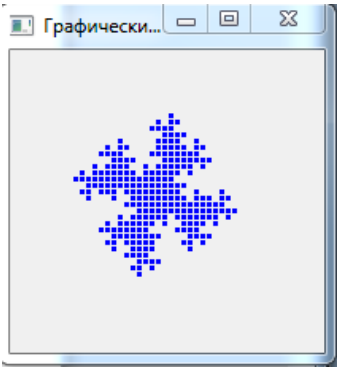

Рис. 8. Вывод объекта по точкам в графическом контексте Qt

Задания на самостоятельную работу:

1.Создать приложение, в котором числа от 1 до 20 случайным образом записываются в данном интервале, не повторяясь.

Пример для 5 чисел:

Входящие данные:

1,2,3,4,5

Результат:

4,1,3,5,2

2. Подготовить приложение, в котором имитируется работа секундной стрелки аналоговых часов.

ЛАБОРАТОРНАЯ РАБОТА № 11
ВВЕДЕНИЕ В OPENGL НА QT

Цель работы: получить представление и приобрести навыки работы с библиотекой OpenGL.

Задачи и требования к выполнению:

1.Изучить элементы и методы шаблонного проекта приложения OpenGL;

2.Изучить принципы работы и базовые функции библиотеки;

3.Ознакомиться с типовыми приемами вывода простейших графических примитивов;

4. Изучить дополнительные решения по расширению возможностей библиотеки OpenGL.

Теоретические сведения

OpenGL представляет собой кроссплатформенную библиотеку по работе с графическими объектами. Поскольку при помощи библиотеки OpenGL можно делать так много вещей, программа, использующая ее, может быть весьма

сложна. Однако базовая структура полезной программы может быть достаточно простой: ее задачами являются инициализация нескольких переменных или переключателей, контролирующих, как OpenGL осуществляет визуализацию изображения и, далее, указание объектов для отображения.

В Qt есть класс, в котором реализован набор функций OpenGL-QGLWidget. Работа с классом, а также возможности по передаче параметров с интерфейсной части приложения QT выполнено в проекте, представленным в листинге 6, в котором показано построение функции 2 переменных.

Листинг 6. Работа с OpenGL в QT

Файл .pro

```
QT       += core gui opengl

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = OpenGL_Appl
TEMPLATE = app


SOURCES += main.cpp\
        mainwindow.cpp \
    glwidget.cpp

HEADERS  += mainwindow.h \
    glwidget.h

FORMS    += mainwindow.ui
```

В данный файл необходимо добавить к строке запись opengl.

Файл main.cpp

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

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

```
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>


namespace Ui {
class MainWindow;
}
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
public:
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

Файл mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QtWidgets>
#include "glwidget.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    connect(ui->slNewDraw, SIGNAL(valueChanged(int)),
ui->widget, SLOT(ChangeStructFunc(int)));
```

```
    connect(ui->vsX, SIGNAL(valueChanged(int)), ui-
>widget, SLOT(ChangeX(int)));
    connect(ui->vsY, SIGNAL(valueChanged(int)), ui-
>widget, SLOT(ChangeY(int)));
    connect(ui->vsAngle, SIGNAL(valueChanged(int)),
ui->widget, SLOT(ChangeAngle(int)));
}
MainWindow::~MainWindow()
{
    delete ui;
}
```

Файл glwidget.h

```
#ifndef GLWIDGET_H
#define GLWIDGET_H
#include <QGLWidget>
#include <cmath>

class GLWidget : public QGLWidget
{
    Q_OBJECT
public:
   explicit GLWidget(QWidget *parent = 0);
   void initializeGL();
   void paintGL();
   void resizeGL(int w,int h);

public:
   static int curvature;

   static int rotateX,rotateY,angle;

   float func(float, float);

public slots:
   void ChangeStructFunc(int);
   void ChangeX(int);
   void ChangeY(int);
   void ChangeAngle(int);

};
#endif // GLWIDGET_H
```

Файл glwidget.cpp

```
#include "glwidget.h"
#include <QDebug>

int GLWidget::curvature=0;
int GLWidget::rotateX=0;
int GLWidget::rotateY=0;
int GLWidget::angle=45;
void GLWidget:: ChangeStructFunc(int slValue)
{
    curvature=slValue;
    //qDebug()<<curvature<<endl;
    updateGL();
}
void GLWidget:: ChangeX(int slValue)
{
    rotateX=slValue;
    updateGL();
}
void GLWidget:: ChangeY(int slValue)
{
    rotateY=slValue;
    updateGL();
}
void GLWidget:: ChangeAngle(int slValue)
{
    angle=slValue;
    updateGL();
}
GLWidget::GLWidget(QWidget *parent) :
    QGLWidget(parent)
{
}

void GLWidget::initializeGL()
{
   glClearColor(0,0,0,1);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glFrustum(-100, 100, -100, 100, 100, 2000);
```

```
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
void GLWidget::paintGL()
{
glClear(GL_COLOR_BUFFER_BIT);
glPushMatrix();
glTranslatef(0, 0, -800);
glRotatef(angle, rotateX, rotateY, 45);
glColor3f(1,1,0);

  double wScale=1.;
  int StepGrid=20;

  for (float x = -480; x < 480; x += StepGrid)
      {
          glBegin(GL_LINE_STRIP);
          for (float y = -480; y < 480; y += StepGrid)
          {
glVertex3f(x*wScale, y*wScale,
           func(x*wScale, y*wScale));
          }
          glEnd();
      }
      for (float y = -480; y < 480; y += StepGrid)
      {
          glBegin(GL_LINE_STRIP);
          for (float x = -480; x < 480; x += StepGrid)
          {
glVertex3f(x*wScale, y*wScale,
           func(x*wScale, y*wScale));
          }
          glEnd();
      }
glPopMatrix();
}

void GLWidget::resizeGL(int w,int h)
{
    glViewport(0, 0, (GLint)w, (GLint)h);
}
float GLWidget::func(float x, float y)
{
```

```
    return sin(x * y * 0.0001)*curvature;
}
```

Файл mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="windowModality">
   <enum>Qt::ApplicationModal</enum>
  </property>
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>448</width>
    <height>332</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Введение в OpenGL QT</string>
  </property>
  <widget class="QWidget" name="centralWidget">
   <layout class="QGridLayout" name="gridLayout_2">
    <item row="0" column="0">
     <layout class="QGridLayout" name="gridLayout">
      <item row="0" column="0">
       <layout class="QVBoxLayout"
name="verticalLayout_2">
        <item>
         <widget class="GLWidget" name="widget"
native="true">
          <property name="sizePolicy">
           <sizepolicy hsizetype="Expanding"
vsizetype="Preferred">
            <horstretch>0</horstretch>
            <verstretch>0</verstretch>
           </sizepolicy>
          </property>
         </widget>
        </item>
```

```
        <item>
         <widget class="QSlider" name="slNewDraw">
          <property name="maximum">
           <number>200</number>
          </property>
          <property name="orientation">
           <enum>Qt::Horizontal</enum>
          </property>
         </widget>
        </item>
       </layout>
      </item>
      <item row="0" column="1">
       <layout class="QVBoxLayout"
name="verticalLayout">
        <item>
         <layout class="QHBoxLayout"
name="horizontalLayout">
          <item>
           <widget class="QSlider" name="vsX">
            <property name="maximum">
             <number>250</number>
            </property>
            <property name="orientation">
             <enum>Qt::Vertical</enum>
            </property>
           </widget>
          </item>
          <item>
           <widget class="QSlider" name="vsY">
            <property name="maximum">
             <number>250</number>
            </property>
            <property name="orientation">
             <enum>Qt::Vertical</enum>
            </property>
           </widget>
          </item>
          <item>
           <widget class="QSlider" name="vsAngle">
            <property name="minimum">
             <number>-90</number>
            </property>
```

```
            <property name="maximum">
             <number>0</number>
            </property>
            <property name="pageStep">
             <number>0</number>
            </property>
            <property name="value">
             <number>-45</number>
            </property>
            <property name="orientation">
             <enum>Qt::Vertical</enum>
            </property>
           </widget>
          </item>
         </layout>
        </item>
        <item>
         <widget class="QPushButton"
name="pushButton">
          <property name="text">
           <string>Выход</string>
          </property>
         </widget>
        </item>
       </layout>
      </item>
     </layout>
    </item>
   </layout>
  </widget>
 </widget>
 <layoutdefault spacing="6" margin="11"/>
 <customwidgets>
  <customwidget>
   <class>GLWidget</class>
   <extends>QWidget</extends>
   <header>glwidget.h</header>
   <container>1</container>
  </customwidget>
 </customwidgets>
 <resources/>
 <connections>
  <connection>
```

```
   <sender>pushButton</sender>
   <signal>clicked()</signal>
   <receiver>MainWindow</receiver>
   <slot>close()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>365</x>
     <y>300</y>
    </hint>
    <hint type="destinationlabel">
     <x>407</x>
     <y>197</y>
    </hint>
   </hints>
  </connection>
 </connections>
</ui>
```

На рис. 9 представлены результаты работы приложения с отображением поверхности, заданной функцией вида sin(x * y * 0.0001)*curvature, где curvature-параметр искривления поверхности.

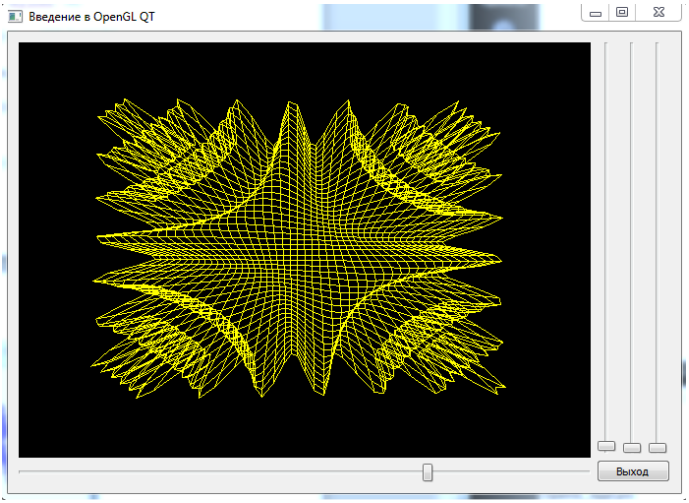

Рис. 9. Работа с OpenGL на Qt

Рассмотрим программный код проекта и приведем описание некоторых функций OpenGL.

Класс QGLWidget вынесен в отдельный набор файлов (glwidget.h и glwidget.cpp), содержит ряд методов и набор статических переменных для изменения координат при повороте графического объекта.

QGLWidget так устроен, что при первой инициализации класса он автоматически вызывает методы в следующем порядке:

При запуске: initializeGL()->resizeGL()->paintGL(). При изменении размера окна: resizeGL()->paintGL(). updateGL() вызывает paintGL()

initializeGL - необходимо использовать для задания всех параметров, изменение которых в дальнейшем, как правило, не производится.

resizeGL - служит для построения размера окна. Если в ходе работы изменится размер окна, но не изменить область просмотра, то при увеличении размера можно наблюдать непредсказуемые явления.

paintGL - этот метод будет выстраивать кадр каждый раз, когда поступит команда на обновление экрана.

В методе initializeGL используются:

glClearColor(0,0,0,1);-цвет в окне для вывода в OpenGL сбрасывается и производится установка в значение, соответствующее черному цвету (первые три параметра задают оттенки красного, зеленого и синего, а последний-прозрачность).

glMatrixMode(GL_PROJECTION); -определяется матрица в пространстве.

glFrustum()-определяет перспективную проекцию для создания реалистичного изображения.

glLoadIdentity()-производится инициализация текущей проекционной матрицы.

glViewport(0, 0, (GLint)w, (GLint)h); -используется для отсечения графического вывода на новым размерам окна.

glClear(GL_COLOR_BUFFER_BIT); - производится очистка буфера.

glPushMatrix() и glPopMatrix() используются в приложении совместно. Поскольку преобразования сохраняются в матрицах, матричный стек предоставляет идеальный механизм для подобного рода запоминаний, переносов и отбрасываний. Все ранее описанные матричные операции (glLoadMatrix(), glMultMatrix(), glLoadIdentity() и команды, создающие специфические матрицы) работают с текущей матрицей, то есть с верхней матрицей стека. С помощью команд управления стеком можно управлять зависимостями, в которых указано, какая матрица находится на вершине стека: glPushMatrix() копирует текущую матрицу и добавляет копию на вершину матричного стека, glPopMatrix() уничтожает верхнюю матрицу в стеке.

glTranslatef()-позволяет передвинуть текущую позицию.

glRotatef()-производит поворот пространства.

glColor3f(1,1,0);-определяет цвет объекта в окне OpenGL (в данном случае-желтый).

В секции glBegin() и glEnd() указывается набор точек, которые определяют объект построения. Кроме того, построение зависит от указания некоторого идентификатора, который в качестве параметра передается в первую функцию.

glBegin() может иметь следующие параметры:

GL_POINTS- определяется набор точек glVertex, которые необходимо отобразить на экране;

GL_LINES-каждая пара вершин позволяет построить отрезок;

GL_LINE_STRIP-отображается набор отрезков-начало каждого определяется конечными координатами предыдущего отрезка;

GL_LINE_LOOP-рисуется ломанная, причем ее последняя точка соединяется с первой;

GL_TRIANGLES-каждые три вызова glVertex задают треугольник, можно подготовить набор треугольников;

GL_TRIANGLE_STRIP-рисуются треугольники с общей стороной;

GL_TRIANGLE_FAN -тоже самое, но по другому правилу соединяются вершины;

GL_QUADS-каждые четыре вызова glVertex задают четырехугольник;

GL_QUAD_STRIP-четырехугольники с общей стороной;

GL_POLYGON-многоугольник задается набором вершин.

Остальные методы библиотеки OpenGL частично документированы в справочной системе Qt, а с дополнительными возможностями можно ознакомиться, обратившись к каталогу с примерами, которые поставляются вместе с Qt SDK.

Задания на самостоятельную работу:

1.Создать приложение по построению графика тригонометрической функции.

2.Разработать программу, в которой имитируется движение шарика на плоскости в замкнутом контуре.

ЛАБОРАТОРНАЯ РАБОТА № 12
РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ В QT

Цель работы: ознакомится в правилами ввода данных и поиска выражений по заданному шаблону в QT

Задачи и требования к выполнению:

1.Изучить подходы, которые определяют правила ввода символьных выражений;

2. Использовать выражения для поиска и ввода данных в проектах QT.

Теоретические сведения

Использование регулярных выражений позволяет задать шаблон набора строки, в которой будут присутствовать строго определенные символы, а также их количество. Таким образом, можно определить практически любое правило ввода строк, чисел с требуемым количеством разрядов и т.д. Целесообразно использовать регулярные выражения в проектах, где требуется авторизация пользователя, набор и ввод серийных номеров, ключей, а также сигнатур для активации программ. Рекомендуется использовать данные конструкции и там, где пользователь постоянно добавляет данные “вручную”, чтобы исключить случаи случайного ввода символов от рядом находящихся клавиш на клавиатуре.

В Qt определен класс по работе с регулярными выражениями - QRegExp. Данный класс QRegExp позволяет как находить, так и делать замену некоторого текста по заданному условию. Для этого он использует выражения, квантификаторы и утверждения. За основу языка взят порядок построения регулярных выражений языка Perl.

Один из примеров выражения - поиск символа, равного x или 5. Другой пример выражения - заключение в квадратные скобки. [ABCD] - означает A или B или C или D. Тоже выражение мы можем записать [A-D].

Квантификаторы определяют количество вхождений данного символа. Например x{1,1} означает только одно вхождение символа x. x{1, 5} - означает от одного до 5 x подряд.

Если рассмотреть пример, в котором требуется найти числа от 0 до 99, то формат регулярного выражения будет выглядеть как [0-9]{1,2}. При таком выражении найдутся все возможные числа от 0 до 99. В случае, когда надо найти строки, которые состоят лишь из одного числа от 0 до 99, то регулярное выражение будет ^[0-9]{1,2}$. Знак ^ в начале выражения означает что регулярное выражение должно

начинаться с начала строки, а знак $ в конце выражения - регулярное выражение должно заканчиваться в конце строки.

В случае поиска записи по квантификатору {1,1}, его указывать не обязательно, т. е. x{1,1} можно просто указать как x. Вместо квантификатора {0, 1} используют ?. Таким образом, для числа можно определить выражение следующим образом: ^[0-9][0-9]?$.

Для поиска цифры (выражение [0-9]) можно использовать символ \d. В итоге значения от 0 до 99 произведутся через выражение ^\d\d?$.

Пример по использованию регулярных выражений представлен листингом 7, в котором реализована проверка ввода данных, представленных в формате xxxx-x (где x-любое число от 0 до 9) со стороны пользователя. В случае верного набора чисел и совпадения выражения с шаблонной строкой, кнопка ввода становится активной, а для пользователя выводится информационное сообщение, как показано на рис. 10.

Листинг 7. Использование регулярных выражений

Файл проекта .pro

```
QT       += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = untitled
TEMPLATE = app
SOURCES += main.cpp\
        widget.cpp

HEADERS  += widget.h
FORMS    += widget.ui
```

Файл main.cpp

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

```
#include "ui_widget.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Widget *myWidget=new Widget();
    myWidget->setWindowTitle("Регулярные выражения");
    myWidget->setWindowIcon(QIcon("C://car.ico"));
    myWidget->show();
    return a.exec();
}
```

Файл widget.h

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QtCore>
#include <QtGui>
#include <QMessageBox>
namespace Ui {
class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();
private:
    Ui::Widget *ui;
 private slots:
    void on_lineEdit_textChanged(const QString &arg1);
    void on_EnterButton_clicked();
};
#endif // WIDGET_H
```

Файл widget.cpp

```
#include "widget.h"
#include "ui_widget.h"
```

```
Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui-
>lineEdit,SIGNAL(textEdited(QString)),this,SLOT(on_lin
eEdit_textChanged(QString)));

    ui->EnterButton->setEnabled(false);
    QRegExp myExpresion("[0-9]{4}[-]{1}[0-9]{1}");
    ui->lineEdit->setValidator(new
QRegExpValidator(myExpresion,this));
}

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

//

void Widget::on_lineEdit_textChanged(const QString &)
{
ui->EnterButton->setEnabled(ui->lineEdit-
>hasAcceptableInput());
}

void Widget::on_EnterButton_clicked()
{
    if (ui->lineEdit->text()=="1234-5")
QMessageBox::information(this,"Сообщение","Доступ
разрешен");
}
```

Файл widget.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>Widget</class>
 <widget class="QWidget" name="Widget">
  <property name="geometry">
   <rect>
    <x>0</x>
```

```
   <y>0</y>
   <width>346</width>
   <height>330</height>
  </rect>
 </property>
 <property name="windowTitle">
  <string>Widget</string>
 </property>
 <layout class="QGridLayout" name="gridLayout">
  <item row="0" column="0">
   <layout class="QVBoxLayout" name="verticalLayout">
    <item>
     <spacer name="verticalSpacer">
      <property name="orientation">
       <enum>Qt::Vertical</enum>
      </property>
      <property name="sizeHint" stdset="0">
       <size>
        <width>20</width>
        <height>40</height>
       </size>
      </property>
     </spacer>
    </item>
    <item>
     <widget class="QLineEdit" name="lineEdit"/>
    </item>
    <item>
     <widget class="QPushButton" name="EnterButton">
      <property name="font">
       <font>
        <pointsize>14</pointsize>
       </font>
      </property>
      <property name="text">
       <string>&amp;Выполнить ввод</string>
      </property>
     </widget>
    </item>
   </layout>
  </item>
 </layout>
</widget>
```

```
<layoutdefault spacing="6" margin="11"/>
<resources/>
<connections/>
    </ui>
```

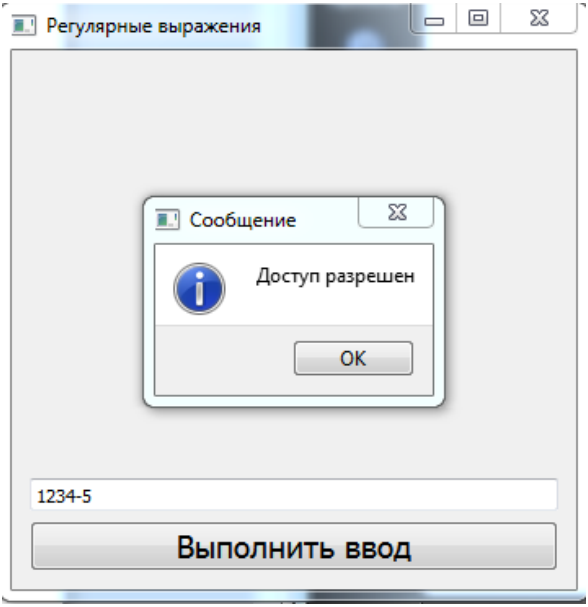

Рис. 10. Проверка ввода данных с использованием регулярных выражений

Задания на самостоятельную работу:

1.Разработать приложение, в котором производится авторизация пользователя с использованием регулярных выражений. Данные о пользователе сохранить в структуре, предусматривающей размещение сведений в количестве десяти.

2.Разработать программу, в которой производится поиск чисел (1,2..9) и их замена на текстовые строки (один, два…девять) с использованием регулярных выражений.

ЛАБОРАТОРНАЯ РАБОТА № 13
ПРИМЕНЕНИЕ БАЗ ДАННЫХ В QT

Цель работы: изучить приемы работы с SQLite в QT.

Задачи и требования к выполнению:

1.Изучить основные принципы работы с БД;

2.Изучить программные решения по подготовке БД;

3.Ознакомиться с программными разработками на QT по управлению БД SQLite.

Теоретические сведения

В QT имеется поддержка нескольких типов баз данных - реализация построена на базе некоторых наборов классов. Среди множества предлагаемых решений наиболее выгодным и простым с точки зрения программного использования в QT является база данных SQLite. SQLite - компактная встраиваемая реляционная база данных. Исходный код библиотеки передан в общественное достояние. Реляционная база данных представляет собой множество взаимосвязанных таблиц, каждая из которых содержит информацию об объектах определенного вида. Каждая строка таблицы содержит данные об одном объекте (например, архитектуре ЭВМ, студенте учебного заведения), а столбцы таблицы содержат различные характеристики этих объектов - атрибуты (например, марка процессора, телефоны домашний или мобильный).

Слово «встраиваемый» означает, что SQLite не использует парадигму клиент-сервер, то есть ядро SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а предоставляет библиотеку, с которой программа компонуется и ядро становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite. Такой подход уменьшает накладные расходы, время

отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа. Простота реализации достигается за счёт того, что перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется.

Проще подготовить БД с помощью SQLite Manager FireFox (рис. 11). Для примера необходимо создать таблицу, в которой будут размещены следующие записи: модель, масса, покрытие, материал. В каждую запись требуется внести ряд значений. После всех действий по внесению данных, файл с базой будет автоматически сохранен на логическом носителе после закрытия менеджера SQLite.

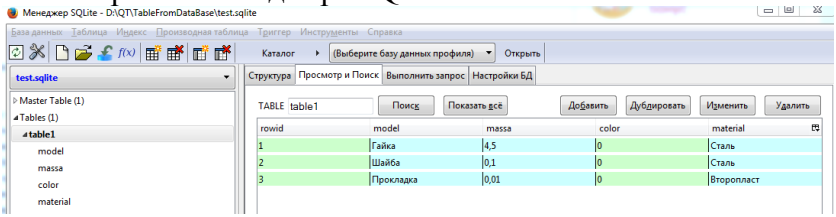

Рис. 11. Создание таблицы в FireFox менеджере SQLite

После создания файла подготавливается проект в Qt с поддержкой БД. Для поддержки БД в системе есть отдельный модуль, предоставляющий удобный «сервис» использования БД - QtSql, поэтому в в .pro файл нужно добавить следующую строку:

QT += sql

Для работы с классами БД нужно включать одноименный заголовок.

#include <QtSql>

Примеры SQL запросов, которые могут быть полезны в разработках:

Посмотреть все столбцы из таблицы table1: SELECT * FROM table1;

Посмотреть столбец model отдельно: SELECT model FROM table1;

Посмотреть набор столбцов (перечисляем через запятую): SELECT model, color FROM table1;

Выполнить сортировку для заданного столбца по алфавиту:

SELECT имя_столбца FROM имя_таблицы ORDER BY имя_столбца_сортировки;

Добавить в таблицу новую строку с заполненными данными:

INSERT INTO table1 (model, massa,color, material) VALUES ('Болт', '0,45','1','Сталь 30');

Далее приводится листинг 8 приложения по работе с БД SQLite. В приложении используются элементы с выпадающими списками, поля которых взяты из столбцов БД. При выборе пользователем в поле любого списка некоторого значения автоматически поставляются в остальные списки значения согласно выбранной строке табличных данных базы. На рис. 12 показан интерфейс приложения.

Листинг 8. Графический вывод объекта по точкам в QT

Содержимое файла widget.h

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QWidget>
#include <QtSql>
#include <QMessageBox>
#include <QFileInfo>
#include <QDebug>
#include <QDir>

namespace Ui {
class Widget;
```

```
}
class Widget : public QWidget
{
    Q_OBJECT
public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();
private slots:
    void on_cbType_currentIndexChanged(int
index);
    void on_cbMaterial_currentIndexChanged(int
index);
    void on_cbColor_currentIndexChanged(int
index);
    void on_cbMas_currentIndexChanged(int index);
private:
    Ui::Widget *ui;
    QSqlDatabase myDB;
};
#endif // WIDGET_H
```

Содержимое файла widget.cpp

```
#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);
    myDB = QSqlDatabase::addDatabase("QSQLITE");
    QString pathToDB =
QString("D:/QT/ExampleSQLite")+QString("/test.sql
ite");
    //QString pathToDB =
QDir::currentPath()+QString("/test.sqlite");
    myDB.setDatabaseName(pathToDB);
    QFileInfo checkFile(pathToDB);
       if (checkFile.isFile()) {
```

```
            if (myDB.open()) {

QMessageBox::information(this,"Сообщение","Подклю
чение выполнено");
            }
            else {

QMessageBox::warning(this,"Сообщение","Файл был
не открыт");
            }
        }
        else {
QMessageBox::critical(this,"Сообщение","Файл БД
не существует");
        }

        if (!myDB.isOpen()) {
                qDebug() << "No connection to
Database";
                return;
           }
        QSqlQuery qry;
        if (qry.exec("SELECT *FROM table1"))
        {
            while (qry.next())
            {
ui->cbType->addItem(qry.value(0).toString());
ui->cbMas->addItem(qry.value(1).toString());
ui->cbColor->addItem(qry.value(2).toString());
ui->cbMaterial->addItem(qry.value(3).toString());
            }
        }
}
Widget::~Widget()
{
    delete ui;
    myDB.close();
}
```

```
void Widget::on_cbType_currentIndexChanged(int
index)
{
     ui->cbMas->setCurrentIndex(index);
     ui->cbColor->setCurrentIndex(index);
     ui->cbMaterial->setCurrentIndex(index);
}
void
Widget::on_cbMaterial_currentIndexChanged(int
index)
{
    ui->cbMas->setCurrentIndex(index);
    ui->cbColor->setCurrentIndex(index);
    ui->cbType->setCurrentIndex(index);
}

void Widget::on_cbColor_currentIndexChanged(int
index)
{
    ui->cbMas->setCurrentIndex(index);
    ui->cbMaterial->setCurrentIndex(index);
    ui->cbType->setCurrentIndex(index);
}
void Widget::on_cbMas_currentIndexChanged(int
index)
{
    int cbIndex=ui->cbMas->currentIndex();
    ui->cbColor->setCurrentIndex(cbIndex);
    ui->cbMaterial->setCurrentIndex(cbIndex);
    ui->cbType->setCurrentIndex(cbIndex);
}
```

Содержимое файла .pro

```
QT       += core gui sql
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = ExampleSQLite
TEMPLATE = app
SOURCES += main.cpp\
```

```
        widget.cpp
HEADERS  += widget.h
FORMS    += widget.ui
```

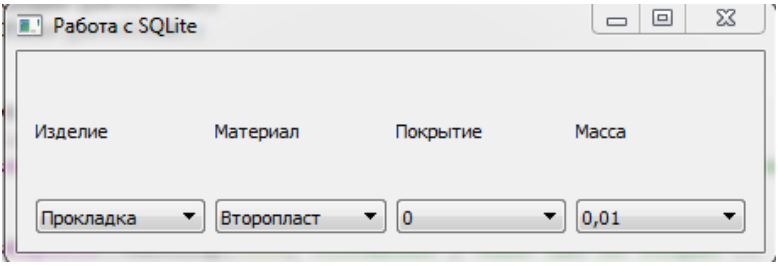

Рис. 12. Реализация приложения с подключением к БД SQLite

Задания на самостоятельную работу:

Разработать приложение с использованием SQLite, которое содержит базу с именами, адресами и телефонами абонентов, кроме того, имеются возможности по поиску строки из базы по одному из параметров.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Прата С. Язык программирования C++. Лекции и упражнения / С. Прата. 5-е изд. – М.: Вильямс, 2007. – 1184 с.
2. Страуструп Б Язык программирования C++ / Б. Страуструп. - М.: Бином, 2011. – 1136 с.
3. Шилдт Г. C++ Базовый курс / Г. Шилдт. 3-е изд. – М.: Вильямс, 2010. – 624 с.
4. Бланшет Ж. QT4: программирование GUI на C++ / Ж. Бланшет, М. Саммерфилд. - М.: Кудиц-Пресс, 2007. - 641 с.
5. Иванова Г.С. Создание пользовательских интерфейсов в программах на C++ с использованием библиотеки QT: учеб. пособие / Г. С. Иванова. - М.: МГТУ имени Н.Э. Баумана, 2011. - 52 с.
6. Шлее М. Qt 4.8 / Профессиональное программирование на C++/ М. Шлее. -СПб.: БХВ-Петербург, 2012. - 912 с.
7. Roberge J. A laboratory course in C++ structures. 2ed / J. Roberge, S. Brandl, D. Whittington. Jones and Bartlett, 2003. - 411 p.
8. London J. Modeling Derivatives in C++ / London J. Wiley, 2005. - 841p.
9. Документация библиотеки Qt [Электронный ресурс]. – Режим доступа: http://qt-project.org/doc/
10. Документация библиотеки Qt [Электронный ресурс]. – Режим доступа: http://qt-doc.ru/

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 1
ЛАБОРАТОРНАЯ РАБОТА № 9 2
ЛАБОРАТОРНАЯ РАБОТА № 10 6
ЛАБОРАТОРНАЯ РАБОТА № 11 14
ЛАБОРАТОРНАЯ РАБОТА № 12 26
ЛАБОРАТОРНАЯ РАБОТА № 13 33
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 40

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам № 9-13 по дисциплине
“Объектно-ориентированное программирование”
для студентов направления 230100.62
«Информатика и вычислительная техника»
очной формы обучения

Составители:
Юров Алексей Николаевич
Паринов Максим Викторович
Рыжков Владимир Анатольевич
Филимонова Анастасия Анатольевна

В авторской редакции

Компьютерный набор А.Н. Юрова

Подписано к изданию 28.11.2014.
Уч.-изд. л. 2,5. «С»

ФГБОУ ВПО «Воронежский государственный технический университет»
394026 Воронеж, Московский просп., 14