МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Воронежский государственный технический университет»

Кафедра радиоэлектронных устройств и систем

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ №7-8
для студентов специальности 11.05.01
«Радиоэлектронные системы и комплексы»
очной формы обучения

Воронеж 2024

УДК 681.3.06(07)
ББК 32.97я7

Составитель А. И. Сукачев

Информационные технологии: методические указания к выполнению лабораторных работ №7-8 для студентов специальности 11.05.01 «Радиоэлектронные системы и комплексы» очной формы обучения / ФГБОУ ВО «Воронежский государственный технический университет»; сост.: А. И. Сукачев. – Воронеж: Изд-во ВГТУ, 2024. – 35 с.

В соответствии с рабочими учебными программами дисциплин приведены описания методов измерений и методик выполнения лабораторных работ, изложены теоретические сведения, лежащие в основе программирования на языке С++. По каждой лабораторной работе в описание включены: цель, основные теоретические сведения, порядок подготовки и проведения работы, перечень положений, которые необходимо отразить в выводах.

Предназначены для студентов специальности 11.05.01 «Радиоэлектронные системы и комплексы» очной формы обучения.

Методические указания подготовлены в электронном виде и содержатся в файле МУ_ИТ_ЛР7-8.pdf.

Ил. 49. Табл. 3. Библиогр.: 3 назв.

УДК 681.3.06(07)
ББК 32.97я7

Рецензент – А. В. Останков, д-р техн. наук, профессор кафедры радиотехники ВГТУ

Издается по решению редакционно-издательского совета Воронежского государственного технического университета

1. ЛАБОРАТОРНАЯ РАБОТА № 7
ПРОЕКТИРОВАНИЕ ПРОТОКОЛА ИНФОРМАЦИОННОГО ОБМЕНА С ИСПОЛЬЗОВАНИЕМ JSON

1.1. ОБЩИЕ УКАЗАНИЯ

1.1.1. ЦЕЛЬ РАБОТЫ

Получить базовые навыки проектирования протокола информационного обмена с использованием JSON.

1.1.2. ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Основным содержанием работы является создание приложения, способного создавать файл с названием, введенным пользователем с клавиатуры, записывать данные в файл, считывать их из файла и выводить на экран. В ходе работы идёт ознакомление с приведённым примером приложения, производится анализ используемых технологий. На основе полученных знаний выполняется индивидуальное задание.

1.2 ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

JSON (JavaScript Object Notation) – это простой формат обмена данными, лёгкий для чтения и представления как людьми, так и компьютером. Он основан на подмножестве языка программирования JavaScript стандарта ECMA-262 3rd Edition - December 1999. JSON - это текстовый формат, полностью независимый от языка программирования, но использующий соглашения, знакомые программистам языков семейства C, включая C, C ++, C #, Java, JavaScript, Perl, Python и многие другие. Эти свойства делают JSON идеальным языком обмена данными.

JSON построен на двух структурах:

- Коллекция пар ключ/значение. На разных языках это реализовано как объект, запись, структура, словарь, хеш-таблица, список ключей или ассоциативный массив.
- Упорядоченный список значений. В большинстве языков это реализовано как массив, вектор, список или последовательность.

Это универсальные структуры данных. Практически все современные языки программирования поддерживают их в той или иной форме. Очевидно, что формат данных, который не зависит от языка программирования, также должен быть основан на этих структурах.

В JSON они принимают следующие формы:

Объект является неупорядоченным набором пар ключ/значение. Объект начинается с левой фигурной скобки и заканчивается правой фигурной скобкой. Каждое имя сопровождается двоеточием, пары ключ/значение отделены друг от друга запятой.

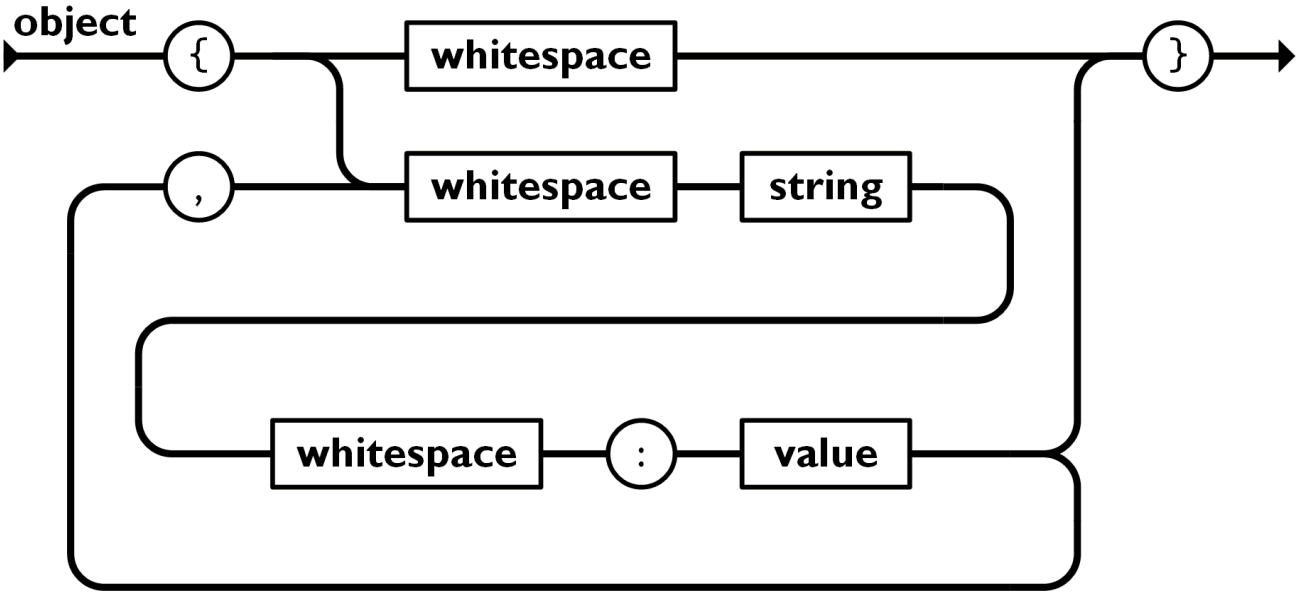

Рис. 1. Структура объекта

Массив представляет собой упорядоченную совокупность значений. Массив начинается с левой квадратной скобки и заканчивается правой квадратной скобкой. Значения отделены друг от друга запятой.

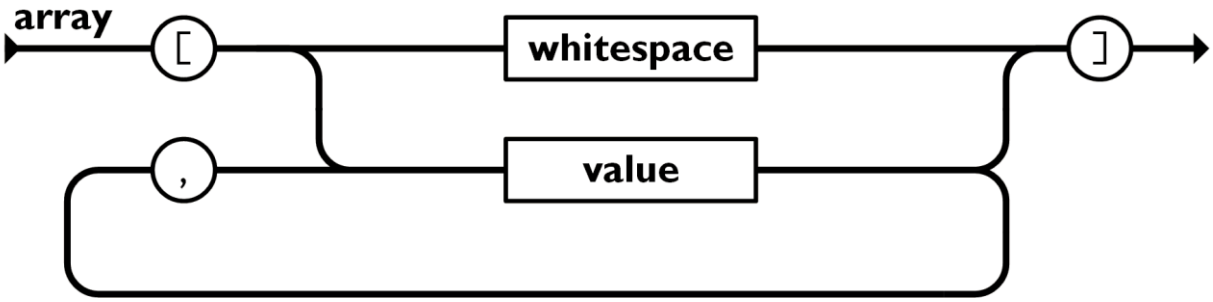

Рис. 2. Структура массива

Значение может быть строкой в двойных кавычках, числом, true, false, null, объектом или массивом. Эти структуры могут быть вложенными.

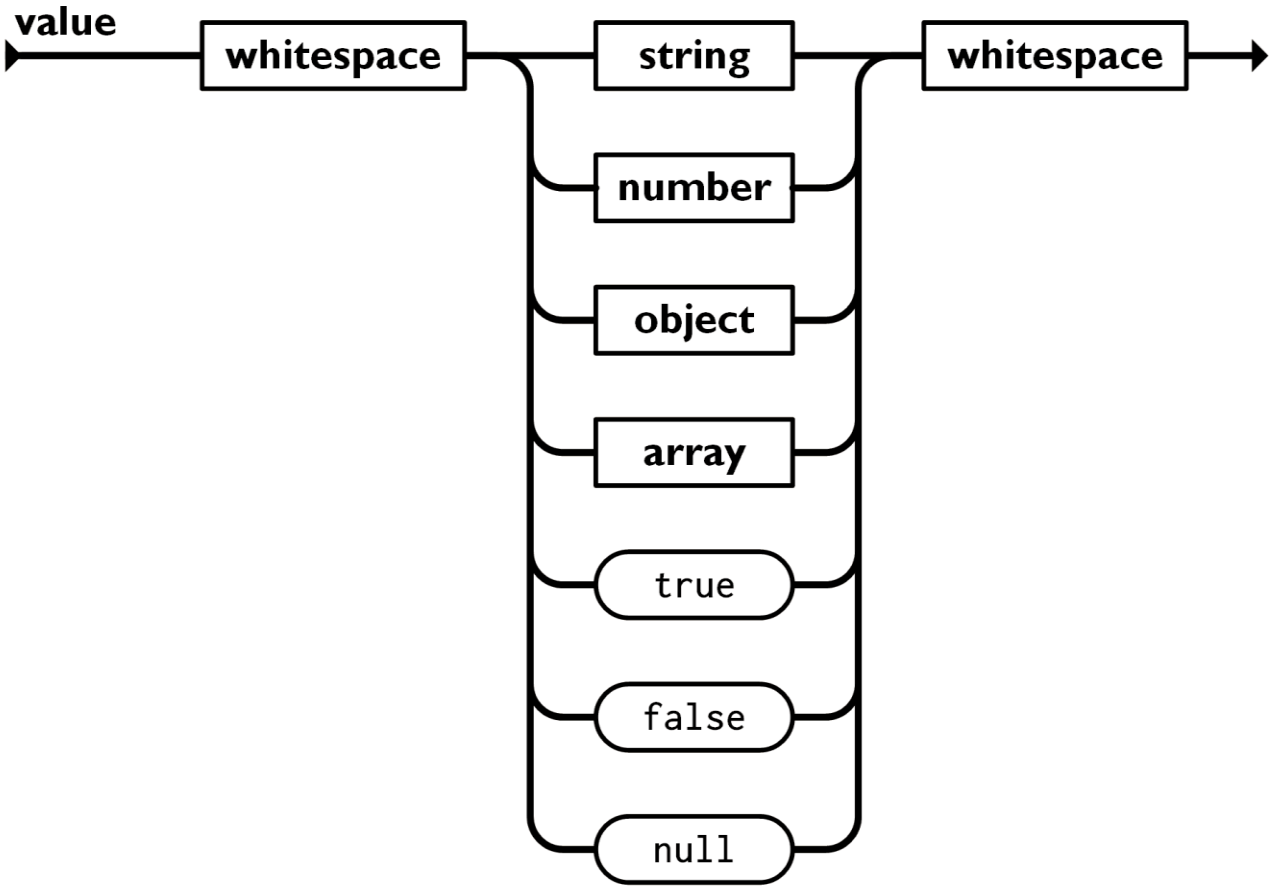

Рис. 3. Значения в JSON

Строка представляет собой последовательность, состоящую из нуля или более символов Unicode, заключённую в двойных кавычках, используя обратный слеш в качестве символа экранирования. Экранирование символов — замена в тексте управляющих символов на соответствующие текстовые подстановки. Один из видов управляющих последовательностей. Обычно языки программирования имеют дело со структурированным текстом, в котором некоторые символы (и их комбинации) используются в качестве управляющих, в том числе управляющих структурой текста. В ситуации, когда необходимо использовать такой символ в качестве «обычного символа языка», применяют экранирование. Символ представляется в виде строки, состоящей из одного символа. Строка JSON очень похожа на строку C или Java.

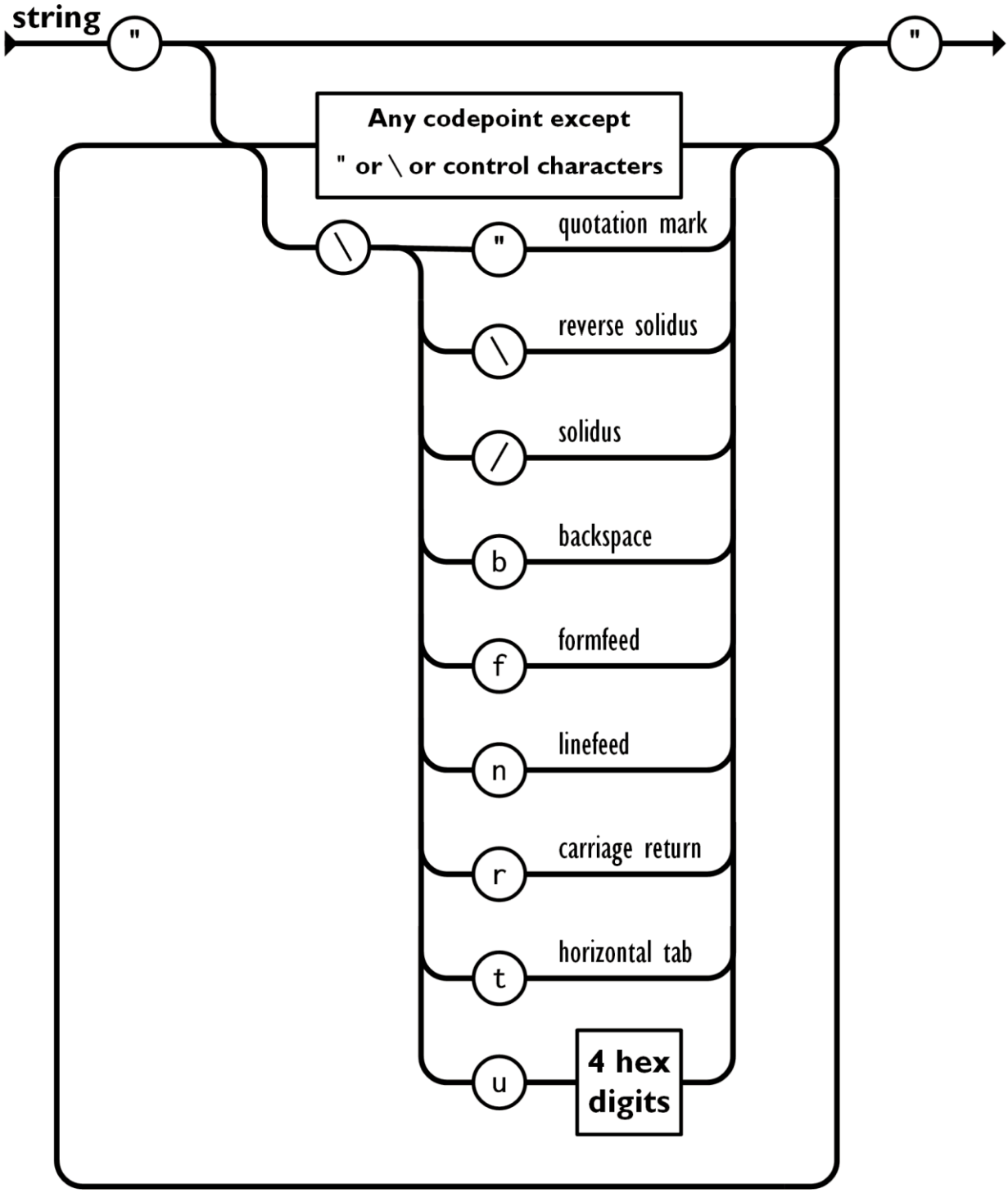

Рис. 4. Экранирование в строке

Число представляется как в C или Java, за исключением того, что восьмеричные и шестнадцатеричные форматы не используются.

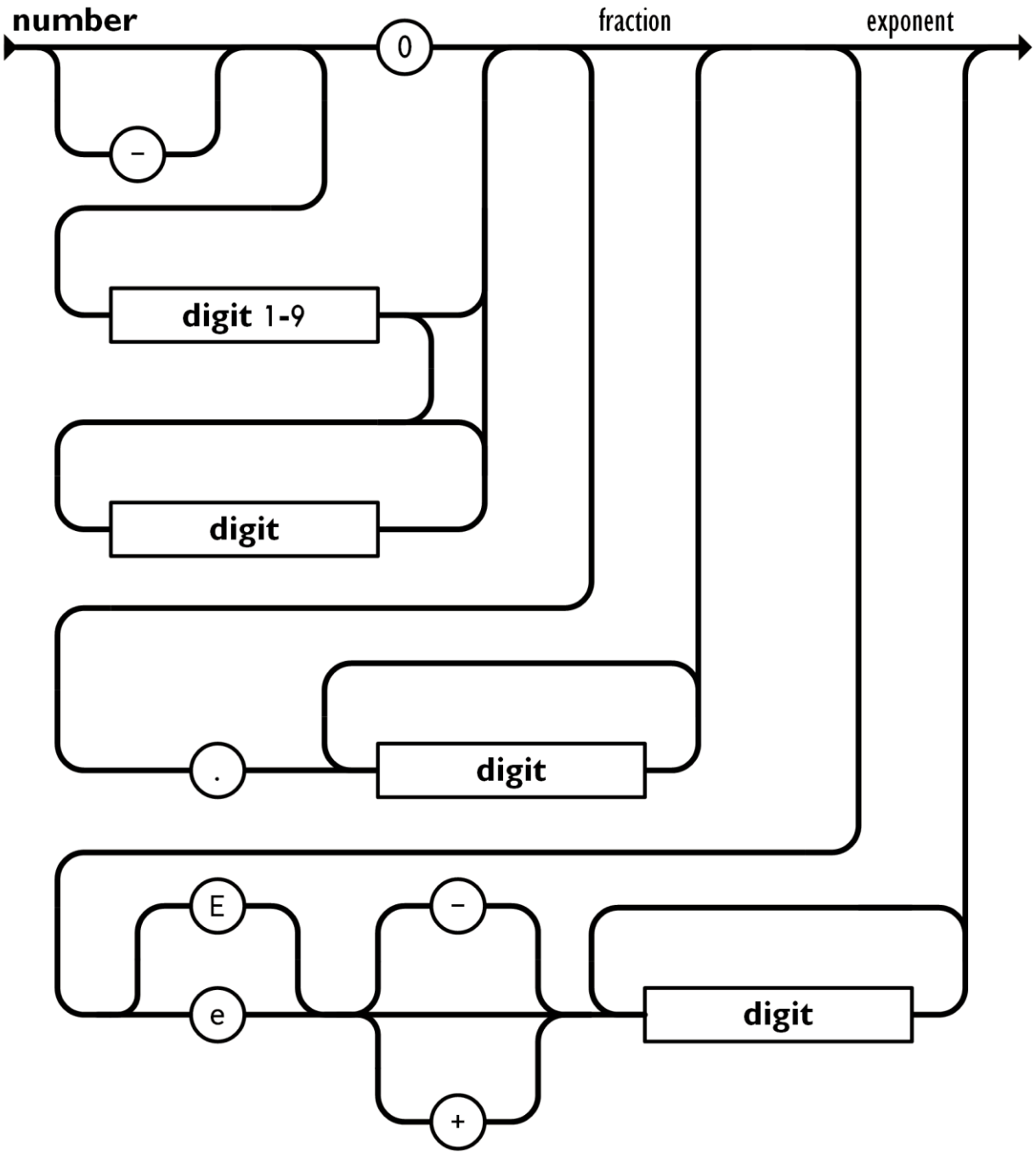

Рис. 5. Числа в JSON

Пробелы могут использоваться между любыми лексемами. За исключением некоторых деталей кодирования, вышеизложенное полностью описывает язык.

JSON имеет следующие преимущества:

- Компактность
- Лёгкость чтения и составления как человеком, так и компьютером.
- Лёгкость преобразования в структуру данных для большинства языков программирования (числа, строки, логические переменные, массивы и так далее)

- Многие языки программирования имеют функции и библиотеки для чтения и создания структур JSON.

Наиболее частое распространенное использование JSON - пересылка данных от сервера к браузеру. Также можно использовать JSON для отправки данных от браузера на сервер.

Qt предоставляет поддержку для работы с данными JSON.

Поддержка JSON в Qt предоставляет простой в использовании C ++ API для анализа, изменения и сохранения данных JSON. Он также содержит поддержку для сохранения этих данных в двоичном формате.

Простой JSON-документ, кодирующий человека, его возраст, адрес и номера телефонов, может выглядеть следующим образом:

{ "FirstName" : "John" , "LastName" : "Doe" , "Age" : 43 , "Address" : { "Street" : "Downing Street 10" , "City" : "London" , "Country" : "Великобритания" } , "Телефоны" : ["+44 1234567" , "+44 2345678"] }

Приведенный выше пример состоит из объекта с 5 парами ключ / значение. Два значения являются строками, одно - число, другое - другой объект, а последнее - массив.

Допустимый документ JSON - это либо массив, либо объект, поэтому документ всегда начинается с квадратной или фигурной скобки.

Классы JSON приведены в табл. 1.

Таблица 1

Классы JSON

QJsonArray	Инкапсулирует массив JSON
QJsonDocument	Способ чтения и записи документов JSON
QJsonParseError	Используется для сообщения об ошибках во время анализа JSON
QJsonObject	Инкапсулирует объект JSON
QJsonObject:: const_iterator	Класс QJsonObject :: const_iterator предоставляет константный итератор в стиле STL для QJsonObject
QJsonObject:: iterator	Класс QJsonObject :: iterator предоставляет неконстантный итератор в стиле STL для QJsonObject
QJsonValue	Инкапсулирует значение в JSON

1.3. ЛАБОРАТОРНЫЕ ЗАДАНИЯ И МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ИХ ВЫПОЛНЕНИЮ

Задание 1. Создание в папке проекта файл с названием и содержанием, введёнными с клавиатуры.

1. Создать проект с базовым классом QWidget (рис. 6).

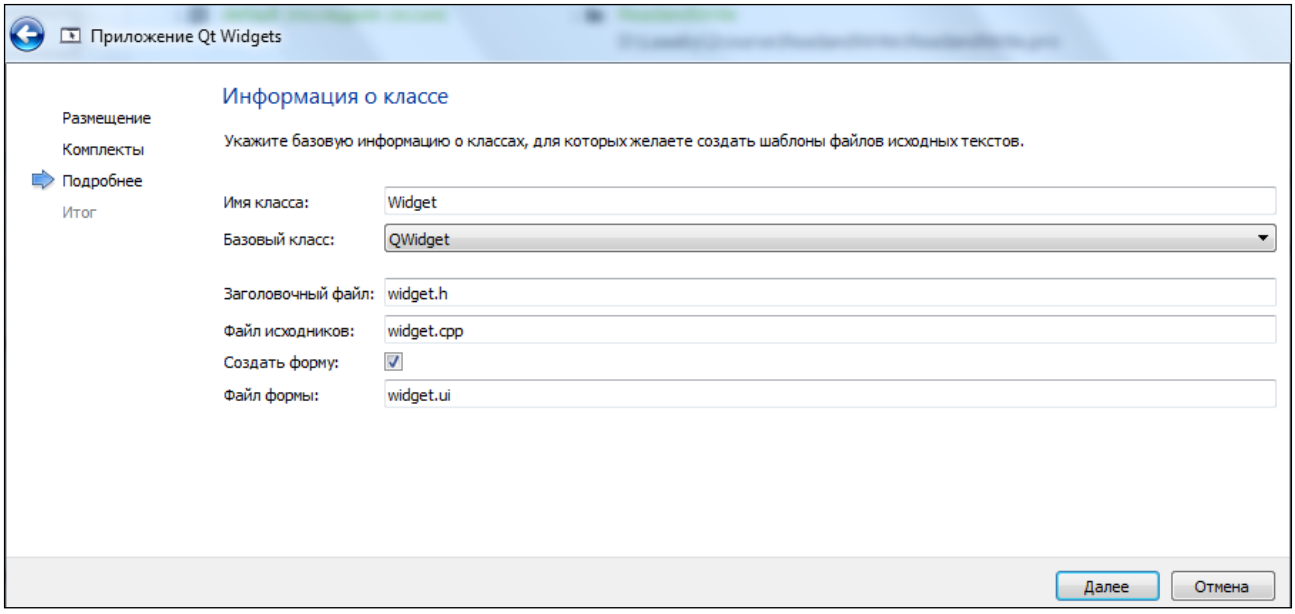

Рис. 6. Задание параметров проекта

2. В режиме «Дизайн» создать следующий интерфейс с помощью объектов классов QLable, QTextEdit и QPushButton (рис. 7)

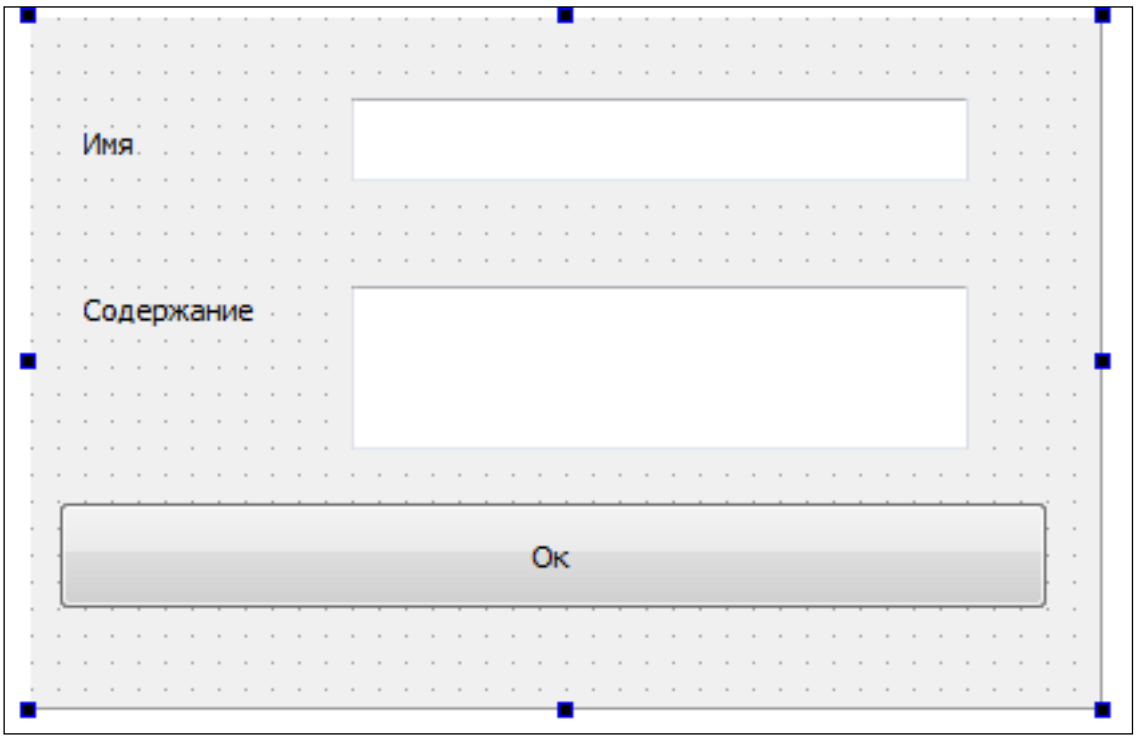

Рис. 7. Интерфейс

3. В файле *widget.h* объявить слот *on_pushButton_clicked* (рис. 8, листинг 1.1, строка 20) и подключить библиотеку класса QFile, которая понадобится для его реализации. Класс QFile предоставляет интерфейс для чтения и записи в файлы.

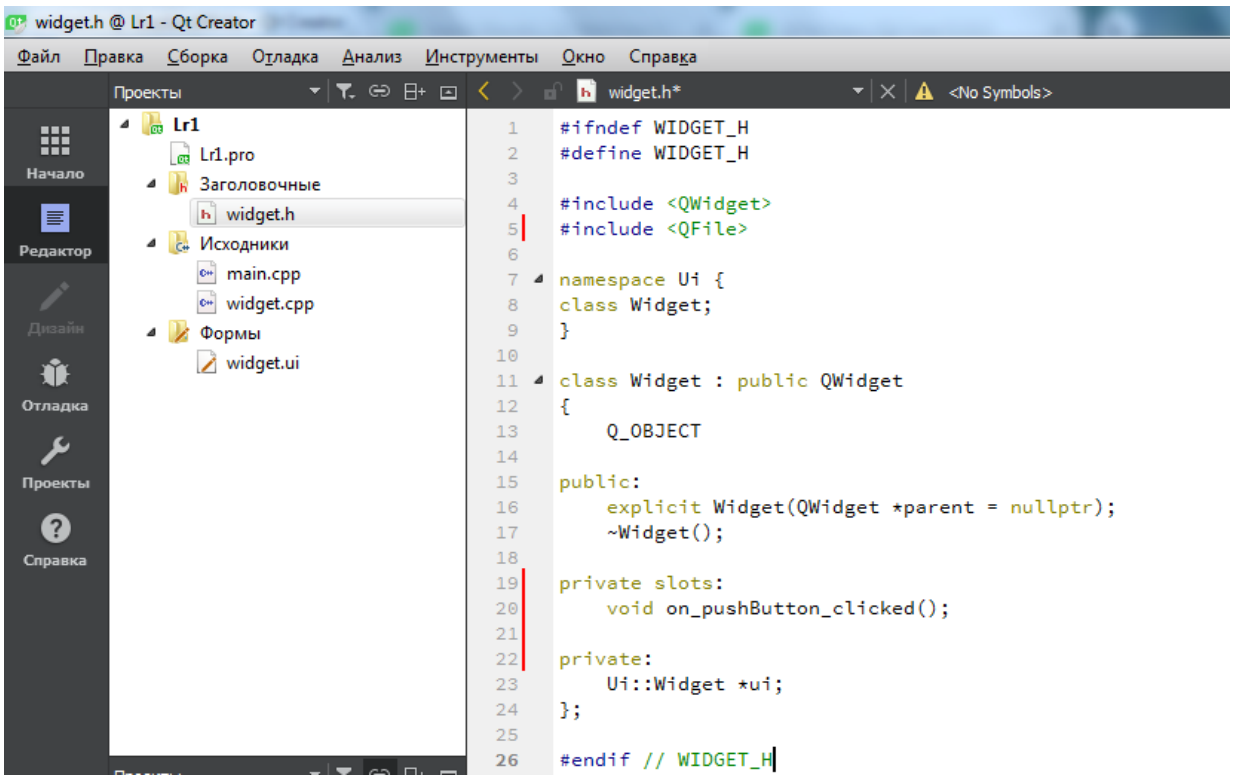

Рис. 8. Листинг 1.1; Файл widget.h

4. Перейти к реализации слота *on_pushButton_clicked* в файле *widget.cpp* (листинг 1.2, строки 16-27)

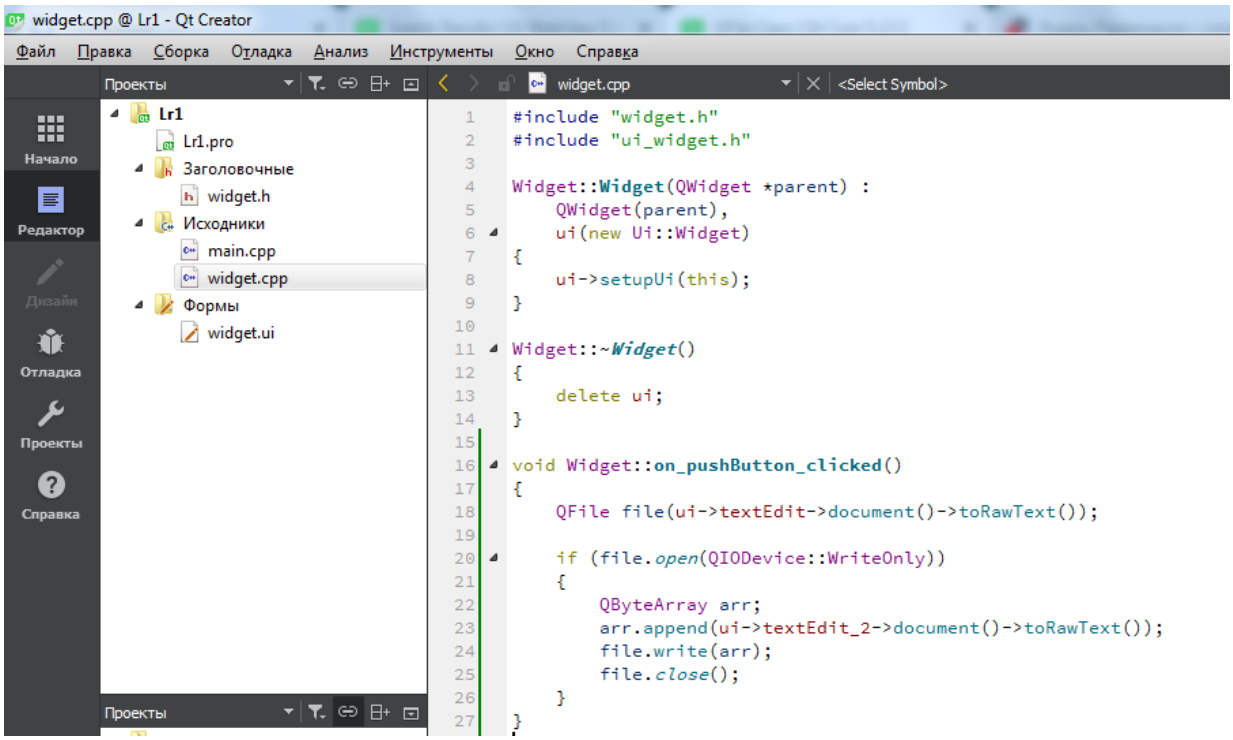

Рис. 9. Листинг 1.2; Файл widget.cpp

- В строке 18 создаём объект класса QFile, с помощью которого при нажатии на клавищу *Ok* в папке проекта создаётся файл с именем, введённым в поле *Имя* (рис. 10, 11).

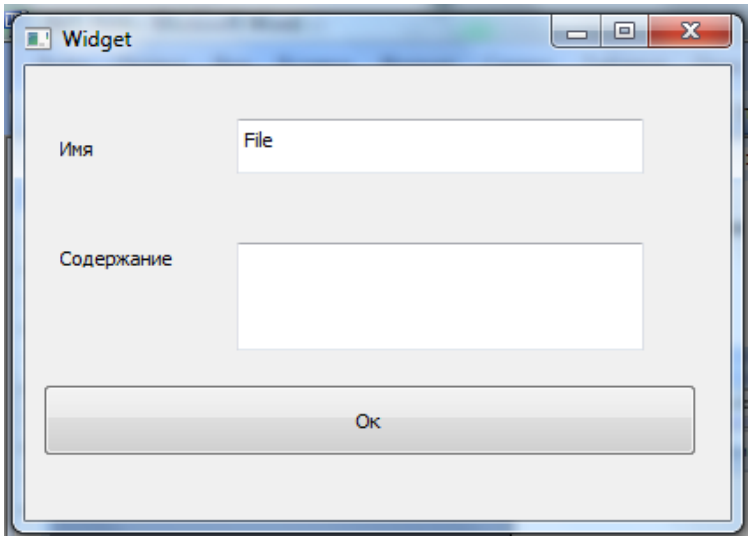

Рис. 10

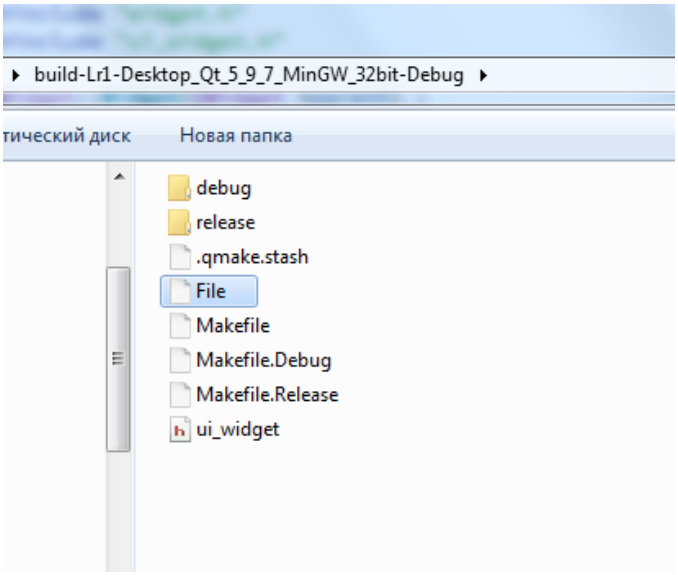

Рис. 11

- Открываем этот файл с флагом *WriteOnly*, что позволяет нам только записывать данные в файл (рис. 9, листинг 1.2, строка 18)
- При открытии файла создаём объект класса QByteArray, который представляет собой массив символов (рис. 9, листинг 1.2, строка 22)
- Добавляем в него строку, введённую в поле *Содержание,* с помощью метода *append* (рис. 9, листинг 1.2, строка 23)
- Записываем массив символов в созданный ранее файл, используя метод *write* (рис. 9, листинг 1.2, строка 24)
- Закрываем файл методом *close* (рис. 9, листинг 1.2, строка 25)

5. Созданная нами программа создаёт файл с введённым нами названием и содержанием (рис. 12, 13).

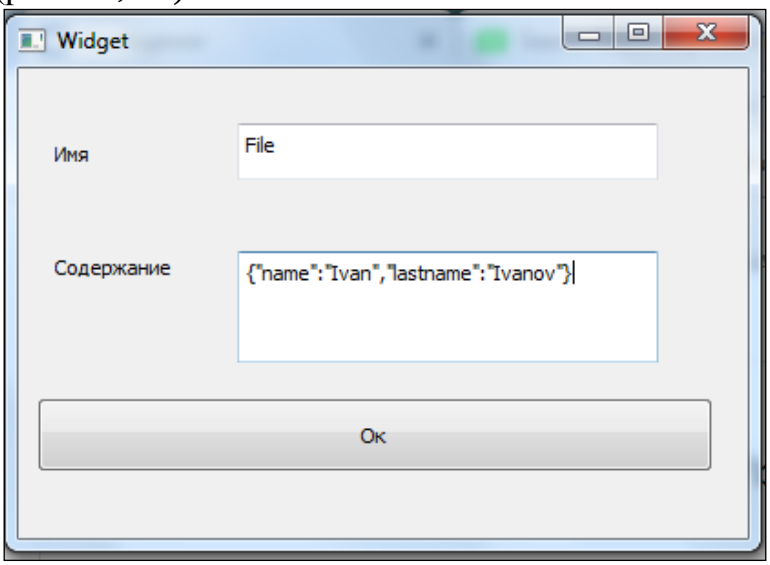

Рис. 12

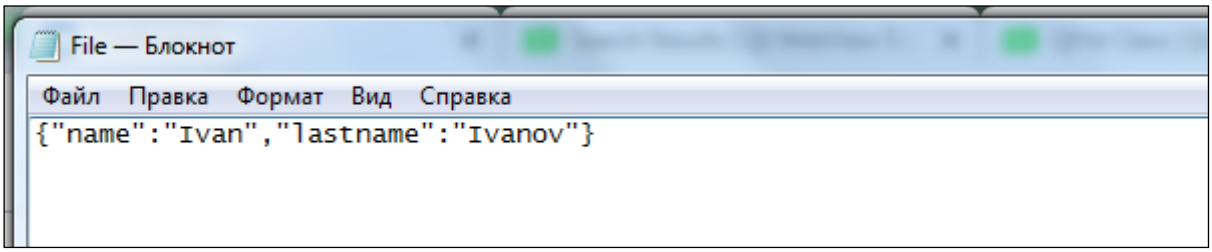

Рис. 13

Задание 2. Чтение данных из файла и вывод на экран

1. Ввести в поле *Содержание* объект JSON {"name":"Ivan","lastname":"Ivanov"} (рис.11).

Здесь строки "name" и "lastname" являются ключами, а строки "Ivan" и "Ivanov" – значениями.

2. В файле *widget.h* подключить библиотеки классов QJsonDocument, QJsonParseError и QJsonObject (рис. 14)

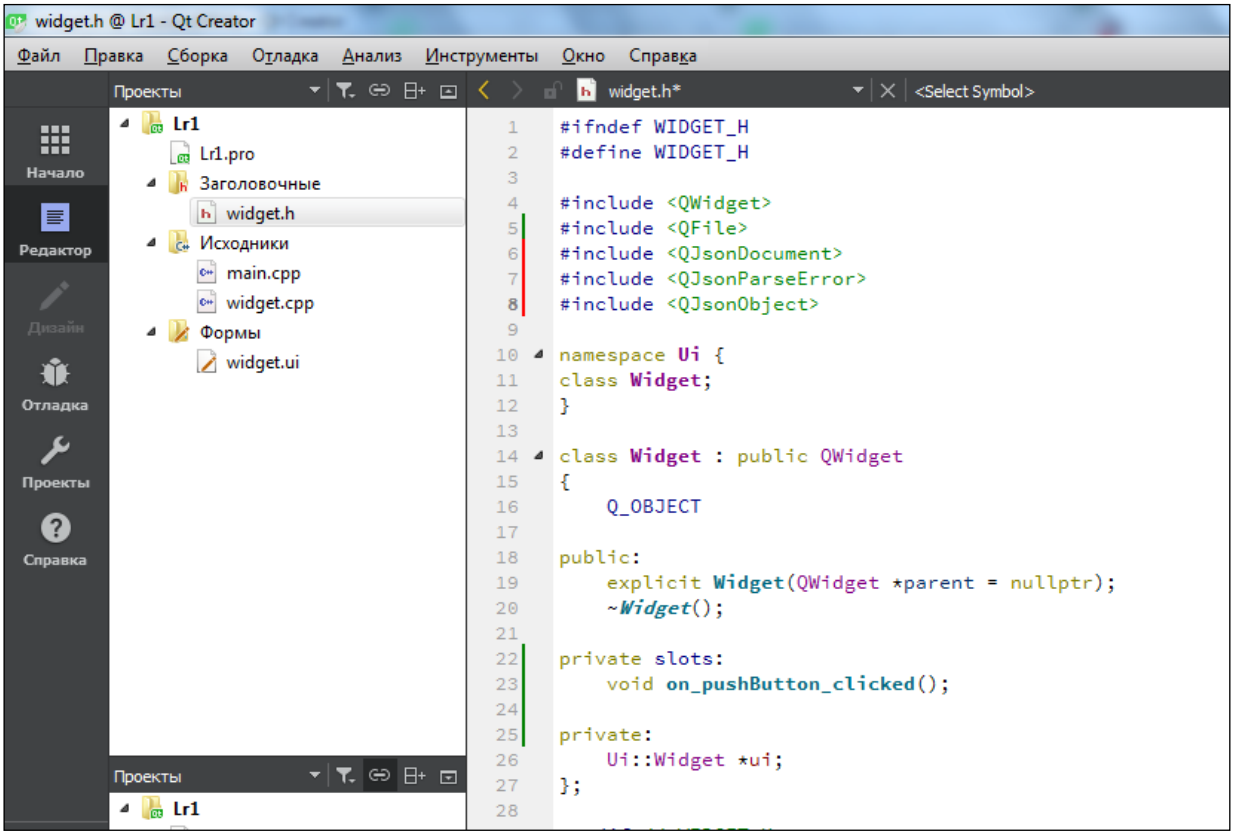

Рис. 14. Листинг 2.1; Файл widget.h

3. В режиме «Дизайн» добавить к интерфейсу два объекта класса QLable (рис. 15)

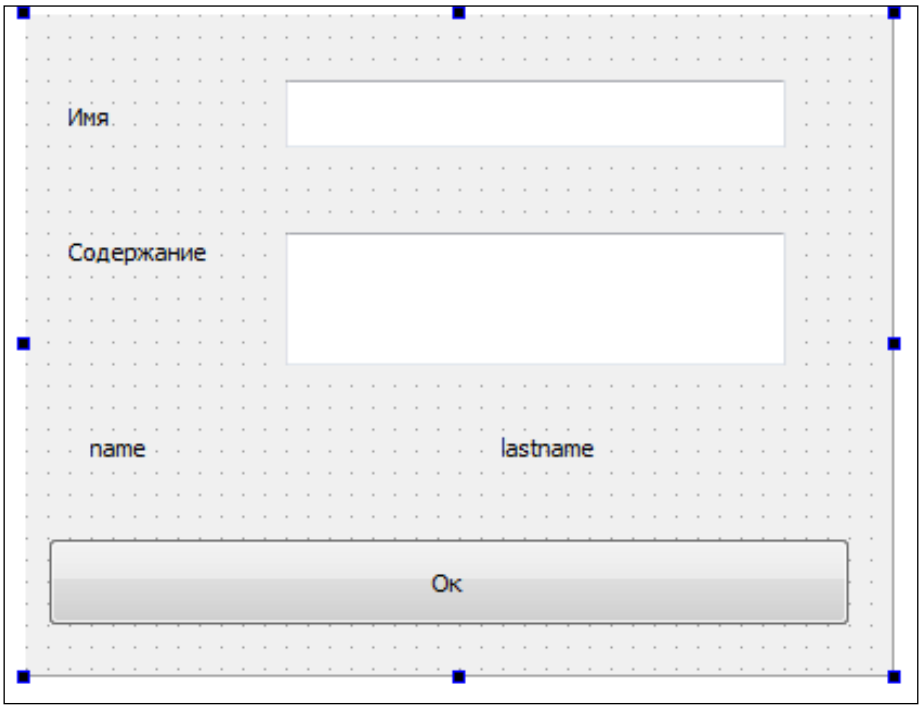

Рис. 15

4. Перейти к реализации новой функции в слоте *on_pushButton_clicked* в файле *widget.cpp* (рис. 16, листинг 2.2 строки 27-40)

```
14  }
15
16  void Widget::on_pushButton_clicked()
17  {
18      QFile file(ui->textEdit->document()->toRawText());
19
20      if (file.open(QIODevice::WriteOnly))
21      {
22          QByteArray arr;
23          arr.append(ui->textEdit_2->document()->toRawText());
24          file.write(arr);
25          file.close();
26      }
27      if (file.open(QIODevice::ReadOnly))
28      {
29          QByteArray arr;
30          arr = file.readAll();
31          file.close();
32          QJsonDocument doc;
33          QJsonParseError err;
34          doc = QJsonDocument::fromJson(arr,&err);
35          if (err.errorString().toInt()==QJsonParseError::NoError)
36          {
37              ui->label_3->setText(doc.object().value("name").toString());
38              ui->label_4->setText(doc.object().value("lastname").toString());
39          }
40      }
41  }
42
```

Рис. 16. Листинг 2.2; Реализация слота on_pushButton_clicked

- Открываем этот файл с флагом *ReadOnly*, что позволяет нам только считывать данные из файла (рис. 16; листинг 2.2, строка 27)
- При открытии файла создаём объект класса QByteArray, который представляет собой массив символов (рис. 16; листинг 2.2, строка 29)
- Записываем в массив содержание документа с помощью метода *readAll* класса QFile (рис. 16; листинг 2.2, строка 30)
- Закрываем файл методом *close* (рис. 16; листинг 2.2, строка 31)
- Создаём объект класса QJsonDocument, необходимый для чтения документа (рис.16; листинг 2.2, строка 32)
- Создаём объект класса QJsonParseError, необходимый для отслеживания ошибок (рис. 16; листинг 2.2, строка 33)
- Записываем в объект класса QJsonDocument данные из массива символов с помощью QJsonDocument::fromJson. В качестве аргументов записываем объект класса QByteArray и указатель на объект класса QJsonParseError (рис. 16; листинг 2.2, строка 34)

- Если ошибок при чтении не обнаружено, то в поле *name* (lable_3) записываем значение с ключом name, а в поле *lastname* (lable_4) - с ключом lastname, преобразовав их в строку методом *toString* (рис. 16; листинг 2.2, строки 35-39)

5. Теперь при нажатии на кнопку *Ok* на экран выводятся введённые значения.

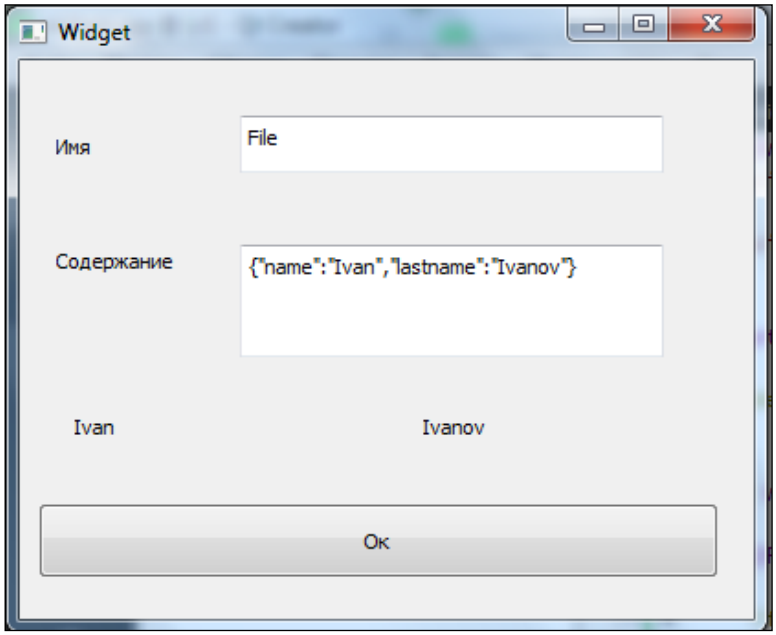

Рис. 17

1.4. ВАРИАНТЫ ЛАБОРАТОРНЫХ ЗАДАНИЙ

Создать приложение создающее файл с названием, введенным пользователем с клавиатуры. Записать в файл объект JSON, считать данные из файла и вывести на экран. Варианты заданий представлены в табл. 2.

Таблица 2

Варианты заданий

Вариант	Задание
1	Имя, фамилия, отчество, дата рождения человека
2	Адрес: город, улица, номер дома, номер квартиры
3	Обед в столовой: три блюда и салат
4	Фамилия студента, факультет, специальность, средний балл
5	Фамилия спортсмена, номер дорожки, время
6	Название предмета, фамилия преподавателя, время экзамена, номер аудитории
7	Наименование товара, цена, количество
8	Марка автомобиля, цвет, номер, год выпуска
9	Фамилия водителя, номер маршрута, номер автобуса
10	Любимый питомец, кличка, возраст

1.5. УКАЗАНИЯ ПО ОФОРМЛЕНИЮ ОТЧЕТА

Отчет должен содержать название и цель работы, краткие теоретические сведения, а также код с комментариями. В конце отчета должен быть вывод.

1.6. КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНЫМ ЗАДАНИЯМ

1. На каких структурах основан JSON?
2. Как формируется объект JSON?
3. Что представляет собой значение в JSON?
4. Что представляет собой экранирование символов?
5. Каковы преимущества JSON?
6. Для чего используется JSON?
7. Какие классы для работы с JSON есть в Qt?

2. ЛАБОРАТОРНАЯ РАБОТА№ 8 ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА МОБИЛЬНОГО КЛИЕНТСКОГО ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА QML

2.1. ОБЩИЕ УКАЗАНИЯ

2.1.1. ЦЕЛЬ РАБОТЫ

Познакомиться с основами языка QML, предназначенного для создания высоко динамичных приложений. На основе полученных знаний спроектировать простейший пользовательский интерфейс.

2.1.2. ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Основным содержанием работы является создание простейшего пользовательского интерфейса для мобильного клиентского приложения с использованием возможностей языка QML. В ходе работы идёт ознакомление с приведённым примером приложения, производится анализ используемых технологий. На основе полученных знаний выполняется индивидуальное задание по проектированию пользовательского интерфейса.

2.2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.2.1. ОБЩИЕ СВЕДЕНИЯ

QML – это многопарадигмальный язык для создания высоко динамичных приложений. С помощью QML объявляются строительные блоки приложения, такие как компоненты пользовательского интерфейса, и задаются различные свойства, определяющие поведение приложения. Поведение приложения может быть дополнительно описано с помощью JavaScript, который является подмножеством языка. Кроме того, QML активно использует Qt, что позволяет типам и другим функциям Qt быть доступными непосредственно из приложений QML.

2.2.2. ОСНОВЫ СИНТАКСИСА QML

QML позволяет определять объекты с точки зрения их атрибутов, а также того, как они связаны и реагируют на изменения в других объектах. В отличие от чисто императивного кода, где изменения в атрибутах и поведении выража-

ются с помощью ряда операторов, которые обрабатываются шаг за шагом, декларативный синтаксис QML интегрирует атрибутные и поведенческие изменения непосредственно в определения отдельных объектов. Эти определения атрибутов могут включать императивный код в случае, если необходимо сложное пользовательское поведение приложения.

Исходный код QML обычно загружается механизмом через документы QML, которые являются отдельными документами кода QML. Их можно использовать для определения типов объектов QML, которые затем можно повторно использовать в приложении. Обратите внимание, что имена типов должны начинаться с заглавной буквы, чтобы быть объявленными как типы объектов QML в файле QML.

2.2.3. ИМПОРТНАЯ ОТЧЕТНОСТЬ

Документ QML может иметь один или несколько импортов в верхней части файла. Импорт может быть любым из:

- версионное пространство имен, в которое были зарегистрированы типы (например, плагином)
- относительный каталог, который содержит определения типов в виде документов QML
- файл JavaScript

Импорт файлов JavaScript должен быть квалифицирован при импорте, чтобы можно было получить доступ к его свойствам и методам.

Примеры:

```
import QtQuick 2.0
import QtQuick.LocalStorage 2.0 as Database
import "../privateComponents"
import "somefile.js" as Script
```

2.2.4 ОБЪЯВЛЕНИЯ ОБЪЕКТОВ

Синтаксически, блок кода QML определяет дерево создаваемых объектов QML. Объекты определяются с помощью объявлений объектов, которые описывают тип создаваемого объекта, а также атрибуты, которые должны быть переданы объекту. Каждый объект может также объявлять дочерние объекты, используя объявления вложенных объектов.

Объявление объекта состоит из имени его типа объекта, за которым следует набор фигурных скобок. Все атрибуты и дочерние объекты затем объявляются в этих скобках.

Вот простое объявление объекта (рис. 18):

```
Rectangle {
    {
    width:  100
    height:  100
    color:  "red"
}}
```

Рис. 18. Объявление объекта

Это объявление объекта типа Прямоугольник, за которым следует набор фигурных скобок, охватывающих атрибуты, определенные для этого объекта. Тип прямоугольника – это тип, доступный модулю QtQuick, и атрибуты, определенные в этом случае, являются значениями свойств ширины, высоты и цвета прямоугольника.

Вышеуказанный объект может быть загружен движком, если он является частью документа QML. То есть, если исходный код дополняется инструкцией import, которая импортирует модуль QtQuick (как показано ниже на рис. 19):

```
import QtQuick 2.0

Rectangle {
    width: 100
    height: 100
    color: «red»
}
```

Рис. 19. Пример реализации объекта посредством движка

При помещении в .qmlфайл и загрузке механизмом QML приведенный выше код создает объект Rectangle с использованием типа Rectangle, предоставляемого QtQuickмодулем:

Рис. 20. Пример получившегося объекта

Примечание. Если определение объекта имеет только небольшое количество свойств, его можно записать в одну строку, например, со свойствами, разделенными точками с запятой (рис. 21):

```
Rectangle { width: 100; height: 100; color: "red" }
```

Рис. 21. Пример записи в одну строку

Очевидно, что объект Rectangle, объявленный в этом примере, действительно очень прост, поскольку он определяет не более, чем несколько значений свойств. Для создания более полезных объектов объявление объекта может определять многие другие типы атрибутов. Кроме того, объявление объекта может определять дочерние объекты, как описано ниже.

2.2.5. ДОЧЕРНИЕ ОБЪЕКТЫ

Любое объявление объекта может определять дочерние объекты через объявления вложенных объектов. Таким образом, любое объявление объекта неявно объявляет дерево объектов, которое может содержать любое количество дочерних объектов.

Например, приведенное ниже объявление объекта Rectangle включает объявление объекта Gradient, которое, в свою очередь, содержит два объявления GradientStop (рис. 22):

```
import QtQuick 2.0

Rectangle {
    width: 100
    height: 100

    gradient: Gradient {
        GradientStop { position: 0.0; color: “yellow” }
        GradientStop { position: 1.0; color: «green» }
    }
}
```

Рис. 22. Объявление вложенных объектов

Когда этот код загружается механизмом, он создает дерево объектов с объектом Rectangle в корне; у этого объекта есть дочерний объект Gradient, у которого, в свою очередь, есть два дочерних объекта GradientStop.

Однако обратите внимание, что это родительски-дочерние отношения в контексте дерева объектов QML, а не в контексте визуальной сцены. Концепция отношения родитель-потомок в визуальной сцене обеспечивается типом Item из QtQuick модуля, который является базовым типом для большинства типов QML, поскольку большинство объектов QML предназначены для визуальной визуализации. Например, Rectangle и Text являются типами, основанными на Item, и ниже объект Text объявлен как визуальный дочерний элемент объекта Rectangle (рис. 23):

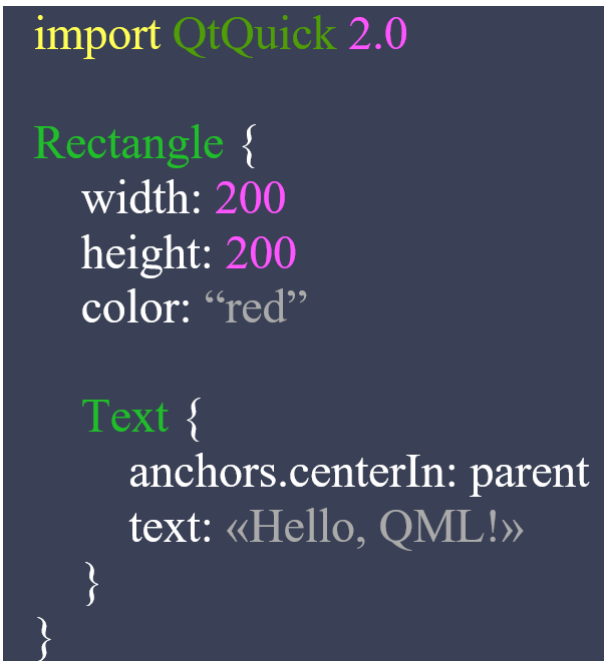

Рис. 23. Объявление дочернего объекта

Когда объект Text ссылается на свое родительское значение в приведенном выше коде, он ссылается на своего визуального родителя, а не на родителя в дереве объектов. В этом случае они одинаковы: объект Rectangle является родительским для объекта Text как в контексте дерева объектов QML, так и в контексте визуальной сцены. Однако, хотя родительское свойство может быть изменено для изменения визуального родителя, родительский объект в контексте дерева объектов не может быть изменен из QML.

(Кроме того, обратите внимание, что объект Text был объявлен без присвоения его свойству Rectangle, в отличие от более раннего примера, в котором объект Gradient назначался gradientсвойству прямоугольника. Это связано с тем, что для дочерних элементов Item было установлено значение по умолчанию для типа свойство, чтобы включить этот более удобный синтаксис.)

2.2.6. КОММЕНТАРИИ

Синтаксис комментирования в QML похож на синтаксис JavaScrip (рис. 24):

- Однострочные комментарии начинаются с // и заканчиваются в конце строки.
- Многострочные комментарии начинаются с / * и заканчиваются * /.

```
Текст { текст : "Привет, мир!" // базовое приветствие / *

    Мы хотим, чтобы этот текст выделялся среди остальных,
    мы даем ему большой размер и другой шрифт.
  * / шрифт . Семейство : шрифт "Helvetica" . pointSize : 24 }
```

Рис. 24. Пример комментариев

Комментарии игнорируются движком при обработке кода QML. Они полезны для объяснения того, что делает часть кода, для справки на более позднем этапе или для объяснения реализации другим.

Комментарии также можно использовать для предотвращения выполнения кода, что иногда полезно для отслеживания проблем.

2.2.7. ПОЗИЦИОНЕРЫ

Элементы позиционера – это элементы контейнера, которые управляют позициями элементов в декларативном пользовательском интерфейсе. Позиционеры ведут себя аналогично менеджерам компоновки, используемым со стандартными виджетами Qt, за исключением того, что они также являются контейнерами сами по себе.

Позиционеры облегчают работу со многими элементами, когда их необходимо расположить в обычном макете.

Qt Quick Layouts также можно использовать для организации элементов Qt Quick в пользовательском интерфейсе. Они управляют позициями и размерами элементов в декларативном пользовательском интерфейсе и хорошо подходят для пользовательских интерфейсов с изменяемыми размерами.

Набор стандартных позиционеров представлен в базовом наборе графических типов Qt Quick:

- Колонка позиционирует своих детей в столбце.
- Поток располагает своих детей рядом, оборачивая при необходимости.
- Сетка позиционирует своих детей в сетке.
- LayoutMirroring свойство, используемое для отражения поведения макета.
- Позиционер предоставляет вложенные свойства, которые содержат сведения о том, где находится элемент в позиционере.
- Ряд позиционирует своих детей подряд.
- Тип сетки QML.

2.2.7.1. ЭЛЕМЕНТЫ СТОЛБЦА

Элементы столбцов используются для вертикального расположения элементов(рис. 25).

Рис. 25. Пример столбцов

В следующем примере элемент Column используется для размещения трех элементов Rectangle в области, определенной внешним элементом. Свойство spacing установлено для включения небольшого пространства между прямоугольниками (рис. 26).

```
import QtQuick 2.0

Item {
    width: 310; height: 170

    Column {
        {
        anchors.horizontalCenter: horizontalCenter: parent.horizontalCenter
        anchors.verticalCenter: verticalCenter: parent.verticalCenter

        spacing: 5

        Rectangle { { color: "lightblue"; radius: 10.0
                width: 300; height: 50
                Text { { anchors.centerIn: centerIn: parent
                    font.pointSize: pointSize: 24; text: "Books" } }
        } }
        Rectangle { { color: "gold"; radius: 10.0
                width: 300; height: 50
                Text { { anchors.centerIn: centerIn: parent
                    font.pointSize: pointSize: 24; text: "Music" } }
        } }
        Rectangle { { color: "lightgreen"; radius: 10.0
                width: 300; height: 50
                Text { { anchors.centerIn: centerIn: parent
                    font.pointSize: pointSize: 24; text: "Movies" } } } }
    }}
}}
```

Рис. 26. Код для реализации столбцов

2.2.7.2. СТРОКА

Строка предметов используется для горизонтального расположения предметов (рис. 27).

Рис. 27. Пример строки

В следующем примере элемент Row используется для размещения трех округленных элементов Rectangle в области, определяемой внешним цветным Rectangle. Свойство spacing установлено для включения небольшого пространства между прямоугольниками (рис. 28).

```
import QtQuick 2.0 QtQuick 2.0

Rectangle {
   {
   width: 320; height: 110
   color: "#c0c0c0"

   Row {
      {
      anchors.horizontalCenter: horizontalCenter: parent.horizontalCenter
      anchors.verticalCenter: verticalCenter: parent.verticalCenter

      spacing: 5

      Rectangle { { width: 100; height: 100; radius: 20.0
               color: "#024c1c" }
      }
      Rectangle { { width: 100; height: 100; radius: 20.0
               color: "#42a51c" }
      }
      Rectangle { { width: 100; height: 100; radius: 20.0
               color: "white" }}
   }}
}}
```

Рис. 28. Пример кода для реализации строки

2.2.7.3. ЭЛЕМЕНТЫ СЕТКИ

Сетка – это тип, который позиционирует свои дочерние элементы при формировании сетки (рис. 29).

Рис. 29. Пример сетки

Сетка создает сетку ячеек, достаточно большую, чтобы вместить все ее дочерние элементы, и помещает эти элементы в ячейки слева направо и сверху вниз. Каждый элемент расположен в верхнем левом углу своей ячейки с позицией (0, 0).

По умолчанию для сетки используется четыре столбца, и создается столько строк, сколько необходимо для размещения всех ее дочерних элементов. Количество строк и столбцов можно ограничить, задав свойства строк и столбцов

Например, ниже приведена таблица, содержащая пять прямоугольников разных размеров (рис. 30):

```
import QtQuick 2.0 QtQuick 2.0

Grid {
    {
    columns: 3
    spacing: 2
    Rectangle { { color: “red”; width: 50; height: 50 }
    }
    Rectangle { { color: “green”; width: 20; height: 50 }
    }
    Rectangle { { color: “blue”; width: 50; height: 20 }
    }
    Rectangle { { color: “cyan”; width: 50; height: 50 }
    }
    Rectangle { { color: “magenta”; width: 10; height: 10 }}
}}
```

Рис. 30. Пример кода для реализации сетки

Сетка автоматически размещает дочерние элементы в сетке (рис. 31):

Рис. 31. Пример автоматического размещения элементов

2.2.7.4. ЭЛЕМЕНТЫ ПОТОКА

Элементы потока используются для размещения на странице таких элементов, как слова, со строками или столбцами непересекающихся элементов (рис. 32).

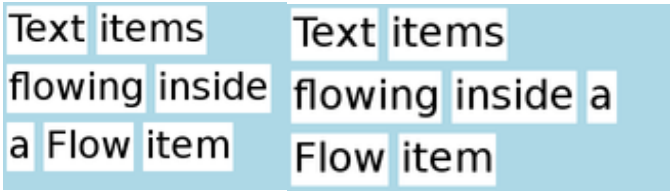

Рис. 32. Пример потока

Элементы потока располагают элементы аналогично элементам таблицы, элементы располагаются в виде линий вдоль одной оси (малой оси), а линии элементов располагаются рядом друг с другом вдоль другой оси (большой оси). Направление потока, а также расстояние между предметами контролируются свойствами потока и расстояния.

В следующем примере показан элемент Flow, содержащий несколько дочерних элементов Text. Они расположены так же, как показано на скриншотах (рис. 33).

```
Import QtQuick 2.0 QtQuick 2.0

Rectangle {
   {
   color: “lightblue”
   width: 300; height: 200

   Flow {
      {
      anchors.fill: fill: parent
      anchors.margins: margins: 4
      spacing: 10

      Text { { text: “Text”; font.pixelSize: pixelSize: 40 }
      }
      Text { { text: “items”; font.pixelSize: pixelSize: 40 }
      }
      Text { { text: “flowing”; font.pixelSize: pixelSize: 40 }
      }
      Text { { text: “inside”; font.pixelSize: pixelSize: 40 }
      }
      Text { { text: “a”; font.pixelSize: pixelSize: 40 }
      }
      Text { { text: “Flow”; font.pixelSize: pixelSize: 40 }
      }
      Text { { text: “item”; font.pixelSize: pixelSize: 40 }}
   }}
}}
```

Рис. 33. Пример кода для реализации потока

Основные различия между позиционерами Grid и Flow заключаются в том, что элементы внутри Flow будут переноситься, когда им не хватит места на вспомогательной оси, и элементы в одной строке могут не совпадать с элементами в другой строке, если элементы не имеют одинаковых размеров. Как и в случае с элементами сетки, независимый контроль расстояния между элементами и между строками элементов отсутствует.

2.2.7.5. ПОЗИЦИОНИРОВАНИЕ С ПОМОЩЬЮ ЯКОРЕЙ

В дополнение к более традиционным Grid, Row и Column, Qt Quick также предоставляет способ компоновки элементов, используя концепцию якорей. Каждый элемент можно рассматривать как имеющий набор из 7 невидимых «якорных линий»: слева, horizontalCenter, правые, верхние, verticalCenter, базовые и нижних (рис. 34).

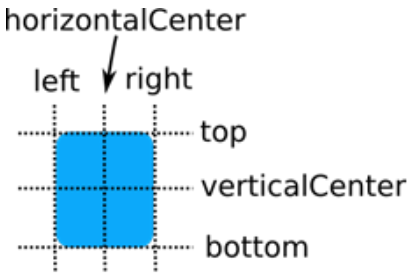

Рис. 34. Пример якорей

Базовая линия (не изображенная выше) соответствует воображаемой линии, на которой будет располагаться текст. Для элементов без текста это то же самое, что и top.

Система привязки Qt Quick позволяет вам определять отношения между линиями привязки различных элементов. Например, вы можете написать (рис. 35):

```
Rectangle { id: rect1; … }
Rectangle { id: rect2; anchors.left: rect1.right; … }
```

Рис. 35. Пример кода для определения привязки

В этом случае левый край rect2 привязан к правому краю rect1, создавая следующее (рис. 36):

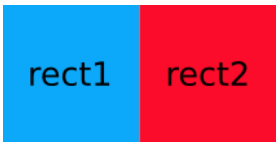

Рис. 36. Пример работы кода

Вы можете указать несколько якорей (рис. 37). Например:

```
Rectangle { id: rect1; … }
Rectangle { id: rect2; anchors.left: rect1.right; anchors.top: rect1.bottom; … }
```

Рис. 37. Пример указания кода для нескольких якорей

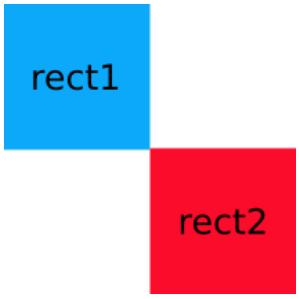

Рис. 38. Пример работы кода

Указав несколько горизонтальных или вертикальных якорей, вы можете контролировать размер элемента. Ниже, rect2 привязан справа от rect1 и слева

от rect3. Если любой из синих прямоугольников будет перемещен, rect2 будет растягиваться и уменьшаться по мере необходимости (рис. 39):

```
Rectangle { id: rect1; x: 0; … }
Rectangle { id: rect2; anchors.left: rect1.right; anchors.right: rect3.left; … }
Rectangle { id: rect3; x: 150; … }
```

Рис. 39. Пример кода с указанием размеров элементов

Рис. 40. Пример работы кода

Есть также несколько удобных анкеров. Aanchors.fill – это удобство, аналогичное установке левого, правого, верхнего и нижнего якорей слева, справа, сверху и снизу целевого элемента. anchors.centerIn - это еще одна удобная привязка, аналогичная настройке привязок verticalCenter и horizontalCenter для verticalCenter и horizontalCenter целевого элемента.

2.3. ПРИМЕР ЛАБОРАТОРНЫХ ЗАДАНИЙ С МЕТОДИЧЕСКИМИ УКАЗАНИЯМИ ПО ИХ ВЫПОЛНЕНИЮ

Задание № 1. Запускаем Qt Creator, выбираем пункт меню Файл –> Новый файл или проект. В открывшемся окне Приложение –> Приложение Qt Quick - Пустое, далее кнопка Выбрать (рис. 41).

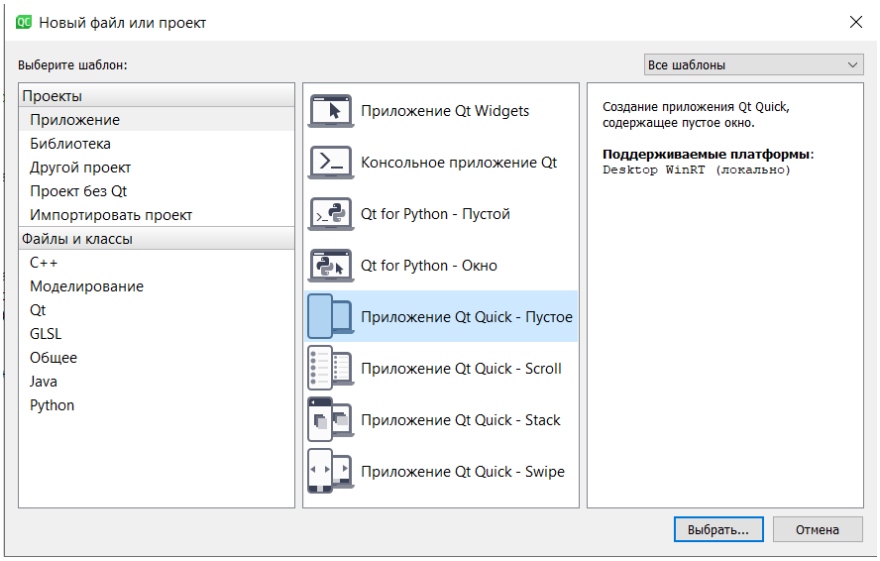

Рис. 41. Настройка проекта

В новом окне вводим название проекта, указываем путь к проекту, жмем Далее.

В следующем окне снимаем галочку Создать форму, она нам не пригодиться, Далее. И в последнем просто нажимаем кнопку Завершить. Все, каркас нашего приложения создан.

Если сейчас собрать и запустить проект, то мы увидим пустое окно. Так и должно быть, т.к. мы еще не создали и не инициализировали QML интерфейс (рис. 42).

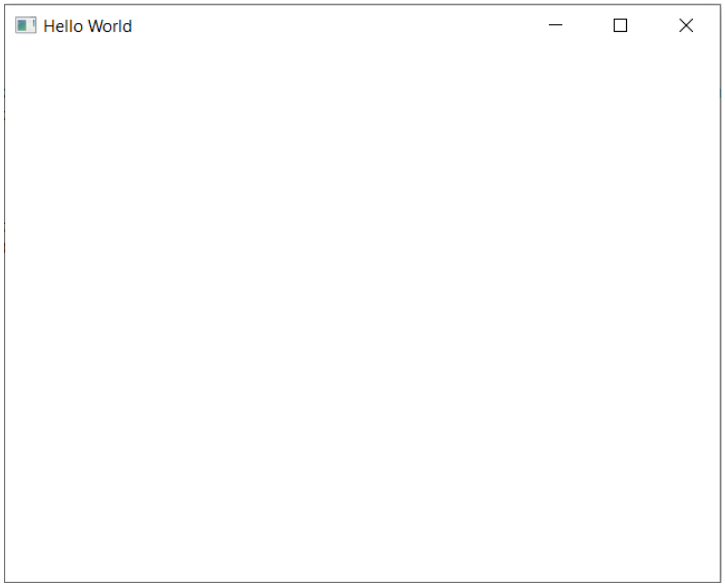

Рис. 42. Окно проекта

Window — наш корневой элемент, в который добавляются дочерние. Он обязательно должен присутствовать в главном QML-файле. Сразу пропишем внутри Window два поля: id и objectName: id будет использоваться для обращения к Window внутри QML-файла, objectName нужен, чтобы найти наше основное окно из QML в C++ коде.

Создаём строку, кнопку, прямоугольник внутри Window. Реализуем метод, который будет вызываться при нажатии на кнопку «BUtton»: в строку будут передаваться значения счётчика, увеличивающегося на 1. Для прямоугольника установим цвет. Пропишем две функции для нахождения наибольшего и наименьшего из хаданных значений. Создаём новые элементы (два прямоугольника и одну кнопку) и помещаем их в строку, задавая отступ между ними. Положение рассчитываем относительно положения главного окна: сначала устанавливаем центр строки по центру главного окна, а затем сдвигаем вниз на 150 пискелей. Реализуем метод, выводящий в консоль наибольшее из двух заданных чисел при нажатии на кнопку «but». Общий код файла *main.qml* представлен ниже (рис. 43):

main.qml* | Label

```
import QtQuick 2.12
import QtQuick.Window 2.12
import QtQuick.Controls 2.4

Window {
    visible: true // Сделать окно видимым
    width: 640
    height: 480
    title: qsTr("Hello World")

    Label {
        id: _label // Идентификатор, по нему будет происходить обращение к свойствам этого элемента
        anchors.centerIn: parent // Изменять размер под размеры родительского элемента
        text: qsTr("LABEL")
    }
    Button {
        id: _button
        anchors.centerIn: parent
        anchors.verticalCenterOffset: 40
        text: qsTr("BUtton")
        property int i: 0
// Метод, который будет вызываться при нажатии на кнопку левой клавишей мыши onClicked
        onClicked:{
            _label.text = i++ // Анимация с шагом в одно нажатие: с каждым нажатием счётчик будет увеличиваться на 1
        }
    }
    Rectangle{

        color: "red" // Цвет прямоугольника, красный
        width: 50
        height: 50
        x: 200
        y: 50
    }


    property string username: "Pavel"

    function max(val1, val2){
        return val1 > val2 ? val1 : val2
    }
    function min(val1, val2){
        return val1< val2 ? val1 : val2
    }
// Элемент позволяющий располагать элементы горизонтально
    Row{
        anchors.centerIn: parent
        anchors.verticalCenterOffset: 150
        spacing: 15 // Отступ между элементами
        Rectangle {color: "red"; width:58; height:76;}
        Rectangle {color: "green"; width:76; height:58}
        Button {
            width: 50
            height: 50
            text: qsTr("but")
// Метод, который будет вызываться при нажатии на кнопку левой клавишей мыши onClicked:
            onClicked: {
                console.log(max(15, 80)) //  При нажатии в консоль будет выводиться наибольшее из двух заданных чисел
            }
        }
    }
}
```

Рис. 43. Листинг файла *main.qml*

Выполнение программы (рис. 44):

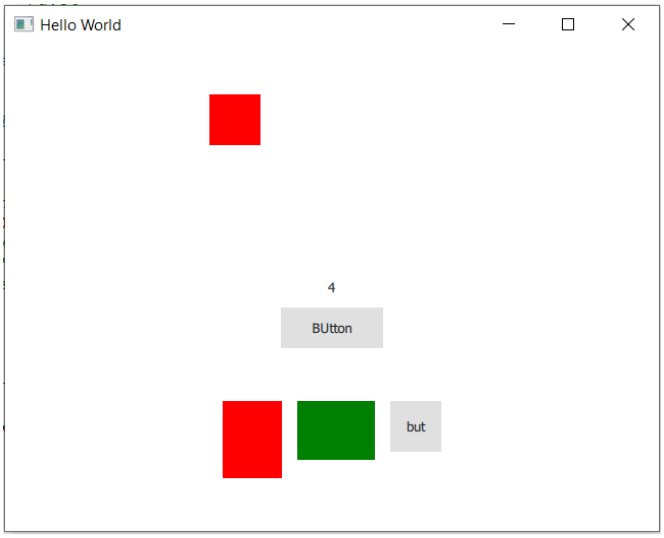

Рис. 44. Выполнение программы

Вывод в консоль наибольшего из двух заданных значений (рис. 45):

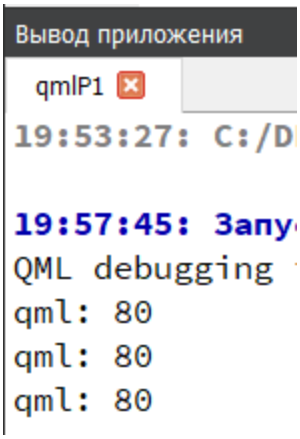

Рис. 45. Вывод в консоли

Задание № 2. Разработать интерфейс, согласно представленному рис. 46.

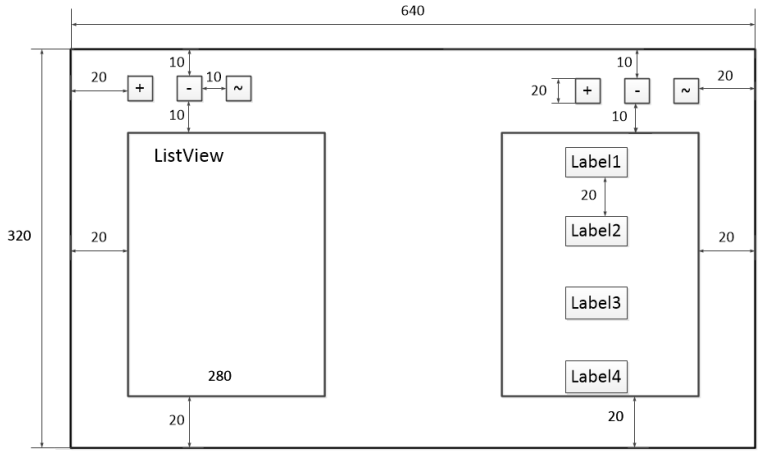

Рис. 46. Интерфейс программы

Подобным образом создаём новый проект.

В корневом элементе прописываем минимальные размеры окна. Создаём две строки с тремя кнопками в каждой, прямоугольник и объект ListView. По аналогии с предыдущим заданием задаём размеры и цвет объектов. С помощью сетки определяем положение элементов. Код *main.qml* (рис. 47):

qml5/main.qml — height

```
import QtQuick 2.12
import QtQuick.Window 2.12
import QtQuick.Controls 2.4

Window {
    visible: true
    minimumHeight: 320
    minimumWidth: 640
    title: qsTr("Hello World")
    Row {
        id: row1
        x: 20
        y: 10
        spacing: 10
        Button { width: 20; height: 20; text: "+" }
        Button { width: 20; height: 20; text: "-" }
        Button { width: 20; height: 20; text: "~" }
    }
    Row {
        id: row2
        x: okno.width +260
        y: 10
        spacing: 10
        Button { width: 20; height: 20; text: "+" }
        Button { width: 20; height: 20; text: "-" }
        Button { width: 20; height: 20; text: "~" }
    }
    ListView {
        id: okno
        x: 20
        y: 40
        height: parent.height -60
        width: parent.width -360
        Rectangle {
            height: parent.height
            width: parent.width
            color: "red"
        }
    }
    Rectangle {
        id: pravoeokno
        color: "#828282"
        x: okno.width +60
        y: 40
        width: 280
        height: parent.height -60
        Column {
            x: 40
            y: 20
            height: pravoeokno.height
            width: pravoeokno.width
            spacing: 20
            Label { text: "Label"; width: 200; height: 40}
            Label { text: "Label"; width: 200; height: 40}
            Label { text: "Label"; width: 200; height: 40}
            Label { text: "Label"; width: 200; height: 40}
        }
    }
```

Рис. 47. Листинг файла *main.qml*

Выполнение программы показаны ниже (рис. 48):

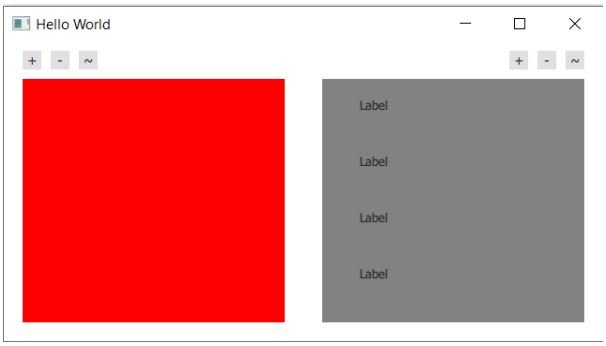

Рис. 48. Выполнение программы

2.4. ЛАБОРАТОРНЫЕ ЗАДАНИЯ

Выполните лабораторное задание в соответствии с полученным вариантом (рис. 49) (размеры и расположение элементов смотреть по табл. 3).

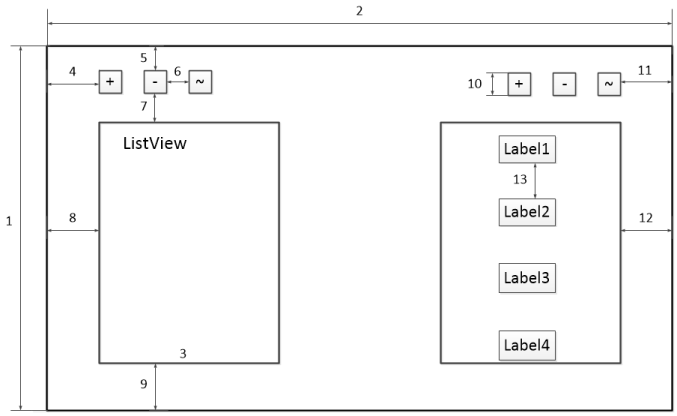

Рис. 49. Интерфейс программы

Таблица 3

Варианты заданий

Размер	1	2	3	4	5	6	7	8	9	10	11	12	13
№ В	Задание												
1	250	500	280	10	10	10	10	10	15	10	10	10	15
2	260	510	290	10	10	10	10	10	150	10	10	15	15
3	270	530	295	15	15	15	15	10	15	15	15	15	15
4	280	540	590	15	10	10	10	15	20	10	15	10	20
5	290	545	285	15	20	20	20	10	20	20	15	15	20
6	300	560	290	20	15	15	15	15	15	15	20	15	15
7	295	550	270	15	15	15	15	15	15	15	15	10	15
8	285	540	265	15	10	10	10	10	15	10	15	10	15
9	275	530	260	15	10	10	15	15	15	10	10	10	10
10	265	520	250	10	10	10	10	10	10	10	10	10	10

2.5. УКАЗАНИЯ ПО ОФОРМЛЕНИЮ ОТЧЕТА

Отчет оформляется в виде пояснительной записки на листах формата А4 (210 х 297 мм). Необходимо дома подготовить заготовку по всей работе. Заготовка должна содержать цель и содержание работы, основные теоретические сведения, все пункты лабораторных заданий, листинги проектов с комментариями. В конце отчета необходимо привести выводы по лабораторной работе и ответы на контрольные вопросы.

2.6. КОНТРОЛЬНЫЕ ВОПРОСЫ К ЛАБОРАТОРНЫМ ЗАДАНИЯМ

1. Язык QML и его характеристики. Основы синтаксиса QML.
2. Что называют импортной отчётностью? Какие виды импорта существуют? Приведите примеры.
3. Как происходит объявление объекта QML?
4. В чём особенность дочерних объектов?
5. Как реализуется комментирование?
6. Что называют позиционированием? Каковы его основные элементы? Как реализуется каждый из них?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Саммерфилд М. Qt. Профессиональное программирование. Разработка кроссплатформенных приложений на C++ / М. Саммерфилд – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 560 с.
2. Шлее М. Qt 5.10. Профессиональное программирование на C++ / М. Шлее – СПб: БХВ-Петербург, 2018. – 1072 с.
3. Бланшет Ж. QT 4: программирование GUI на C++ / Ж. Бланшет – М.: КУДИЦ-ПРЕСС, 2007. – 546 с.

ОГЛАВЛЕНИЕ

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ №7-8
для студентов специальности 11.05.01
«Радиоэлектронные системы и комплексы»
очной формы обучения

Составитель
Сукачев Александр Игоревич

Издается в авторской редакции

Подписано к изданию 18.03.2024.
Уч.-изд. л. 1,8.

ФГБОУ ВО «Воронежский государственный технический университет»
394006 Воронеж, ул. 20-летия Октября, 84