Федеральное агентство связи

ордена Трудового Красного Знамени федеральное государственное

бюджетное образовательное учреждение высшего образования

«Московский технический университет

связи и информатики»

Кафедра математической кибернетики и

информационных технологий

«Лабораторная работа №1»

Выполнил:

студент группы БФИ1701

Истомин М.И.

Москва 2019 г.

**Оглавление**

[Цель 3](#_Toc21309753)

[Задачи: 3](#_Toc21309754)

[Краткая теория 4](#_Toc21309755)

[Порядок выполнения работы: 6](#_Toc21309756)

[Вывод: 19](#_Toc21309757)

# **Цель**

Получить навыки и знания о работе с базовыми элементами среды разработки Qt. Изучив теоретическую часть, написать программы в рамках задания лабораторной работы, используя элементы, о которых была получена информация.

# **Задачи:**

1. Создать виджет с надписью hello world, задать размеры виджета, вывести его на экран. В качестве виджета использовать кнопку класса Qpushbutton.
2. Создать текстовую метку при помощи класса Qlabel, в текстовую метку записать текст hello Qt, вывести сообщение с использованием указателей и класса Qlabel.
3. Создать текстовую метку при помощи класса Qlabel, в текстовую метку записать текст hello Qt, вывести сообщение, не используя указателей.
4. Создать кнопку при помощи класса Qpushbutton с текстом quit. При нажатии на кнопку должно происходить завершение программы.
5. Создать объект класса Qdate содержащий текущую дату, создать объект класса Qtime, содержащий текущее время, вывести полученную информацию о дате и времени в текстовую метку с использованием объектов класса Qstring.
6. Используя события таймера через каждые 200мсек выводить в текстовую метку сообщение.
7. Используя QtDesigner поместить на форму объект класса Qlabel (текстовую метку), которую необходимо выводить текущее время в режиме онлайн.
8. Используя объекты класса Qfile скопировать содержимое одного бинарного файла в другой, при этом оссуществить проверку на возможность открытия файлов для чтения и записи.
9. Используя объекты класса Qbytearray скопировать содержимое одного бинарного файла в другой, при этом оссуществить проверку на возможность открытия файлов для чтения и записи.
10. Используя объекты класса Qbufer осуществить вывод текстовой строки в консоль.
11. Используя обхекты класса Qtextstream считать содержимое текстового файла и вывести его на консоль.
12. Используя метод readall считать содержимое текстового файла и вывести его на консоль.

# **Краткая теория**

Простейшие элеметы управления:

* QWidget – базовый класс для всех визуальных элементов;
* QLabel – текстовая метка или пиктограмма;   
  QComboBox – поле ввода с кнопкой выбора и раскрывающимся списком;
* QRadioButton – зависимый переключатель (радиокнопка);
* QPushButton – кнопка;
* QListWidget – список;
* QTableWidget – таблица.

Работа с датой и временем в Qt осуществляется с помощью классов **QDate**, **QTime** и **QDateTime**, которые предназначены для хранения дат и времени и проведения с ними различных операций. Чаще всего требуется получение текущей даты и времени. Эти классы предоставляют методы для преобразования даты и времени в строку определенного формата. Также есть методы для проведения обратного преобразования — из строки.

Таймер уведомляет приложение об истечении заданного промежутка времени. События таймера относятся к разряду внешних прерываний. Внешние прерывания — это прерывания, вызываемые асинхронными событиями, например, устройствами ввода/вывода или самим устройством таймера. Интервалы запуска таймера устанавливаются в миллисекундах. Недостаток состоит в том, что если программа занята интенсивными вычислениями, то события таймера могут быть обработаны по окончании процесса вычисления. При выходе из приложения таймеры автоматически уничтожаются.

Каждый класс, унаследованный от **QObject**, содержит свои собственные встроенные таймеры. Вызов метода QObject::startTimer() производит запуск таймера. В качестве параметра ему передается интервал запуска в миллисекундах. Метод startTimer() возвращает идентификатор, необходимый для распознавания таймеров, используемых в объекте. По истечении установленного интервала запуска генерируется событие **QTimerEvent**, которое передается В метод timerEvent(). Вызвав метод QTimerEvent::timerId() объекта события **QTimerEvent**, можно узнать идентификатор таймера, инициировавшего это событие.

Использование объекта класса **QTimer** гораздо проще, чем использование события таймера, определенного в классе **QObject**. К недостаткам работы с событием таймера относится необходимость наследования одного из классов наследующих **Qobject**, если в объекте создается более одного таймера, то возникает необходимость различать таймеры, чтобы узнать, который из них явился инициатором события.

Для ликвидации этих неудобств **Qt** предоставляет класс таймера **QTimer**, являющийся непосредственным наследником класса **QObject**. Чтобы запустить таймер, нужно создать объект класса **QTimer**, а затем вызвать метод start(). В параметре метода передается значение интервала запуска в миллисекундах.

Редко встречается приложение, которое не обращается к файлам. 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, состоящую из следующих классов:

* QDir — для работы с директориями;
* QFile — для работы с файлами;
* QFileInfо — для получения файловой информации;
* QIODevice — абстрактный класс для ввода/вывода;
* QBuffer — для эмуляции файлов в памяти компьютера.

**QIODevice** — это абстрактный класс, обобщающий устройство ввода/вывода, который содержит виртуальные методы для открытия и закрытия устройства ввода/вывода, а также для чтения и записи блоков данных или отдельных символов.

Реализация конкретного устройства происходит в унаследованных классах.

В Qt есть четыре класса, наследующие класс **QIODevice**:

* QFile — класс для проведения операций с файлами;
* QBuffer — позволяет записывать и считывать данные в массив **QByteArray**, как будто бы это устройство или файл;
* QAbstractSocket — базовый класс для сетевой коммуникации посредством сокетов.
* QProcess — этот класс предоставляет возможность запуска процессов с дополнительными аргументами и позволяет обмениваться информацией с этими процессами посредством методов, определенных в **QIODevice**.

Считывать и записывать данные можно с помощью методов read() и write()соответственно. Для чтения всех данных сразу определен метод readAll(), который возвращает их в объекте типа **QByteArray**. Строку или символ можно прочитать методами readLine() и getChar() соответственно.

Благодаря интерфейсу класса **QIODevice** можно работать со всеми устройствами одинаково, при этом не имеет значения, является ли устройство файлом, буфером или другим устройством

**Класс QBuffer** унаследован от QIODevice, и представляет собой эмуляцию файлов в памяти компьютера (memory mapped files). Это позволяет записывать информацию в оперативную память и использовать объекты как обычные файлы (открывать при помощи метода open() и закрывать методом close()). При помощи методов write() и read() можно считывать и записывать блоки данных. Можно это так же сделать при помощи потоков, которые будут рассмотрены далее.

Класс QBuffer полезен для проведения операций кэширования. Например, можно считывать файлы растровых изображений в объекты класса QBuffer, а затем, по необходимости, получать данные из них.

Класс **QTemporaryFile** представляет реализацию для временных файлов. Этот класс самостоятельно создает себе имя с гарантией его уникальности, для того чтобы не возникало конфликтов, в результате которых могли бы пострадать уже существующие файлы.

Класс **QDateStream** является гарантом того, что формат, в котором будут записаны данные, останется платформонезависимым и его можно будет считать и обработать на других платформах. Это делает класс незаменимым для обмена данными по сети с использованием сокетных соединений.

Формат данных, используемый **QDataStream**, в процессе разработки версии **Qt** претерпел множество изменений и продолжает изменяться. По этой причине этот класс знаком с различными типами версий, и для того чтобы заставить его использовать формат обмена, соответствующий определенной версии **Qt**, нужно вызвать метод setVersion(), передав ему идентификатор версии.

Класс поддерживает большое количество типов данных, к которым относятся: QByteArray, QFont, QImage, QMap, QPixmap, QString, QValueList и Variant.

# **Порядок выполнения работы:**

**1.1.1. Создать виджет с надписью hello world, задать размеры виджета, вывести его на экран. В качестве виджета использовать кнопку класса Qpushbutton.**

**Листинг 1.1.1.**

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

#include <QPushButton>

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

{

QApplication a(argc, argv);

QPushButton button("helloworld");//обьявление кнопки с именем

button.resize(200,60);//изменение размеров кнопки

button.show();//показать кнопку

return a.exec();

**Скриншот выполнения:**



Рисунок 1 – результат выполнения задания №1

**1.1.2. Создать текстовую метку при помощи класса Qlabel, в текстовую метку записать текст hello Qt, вывести сообщение с использованием указателей и класса Qlabel.**

**Листинг 1.1.2.**

**main.cpp**

#include "helloworld\_1\_2.h"

#include <QApplication>

#include <QLabel>

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

{

QApplication a(argc, argv);

QLabel \*label = new QLabel("<h2><i>Hello,</i>" "<font color = red> Qt!</font> </h2>");

label->show();

return a.exec();

}

**helloworld.cpp**

#include "helloworld\_1\_2.h"

#include "ui\_helloworld\_1\_2.h"

Helloworld\_1\_2::Helloworld\_1\_2(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::Helloworld\_1\_2)

{

ui->setupUi(this);

}

Helloworld\_1\_2::~Helloworld\_1\_2()

{

delete ui;

}

**Скриншот выполнения:**



Рисунок 2 - результат выполнения задания № 2

**1.1.3. Создать текстовую метку при помощи класса Qlabel, в текстовую метку записать текст hello Qt, вывести сообщение, не используя указателей.**

**Листинг 1.1.3.**

**main.cpp**

#include "helloworld\_1\_3.h"

#include <QApplication>

#include <QLabel>

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

{

QApplication a(argc, argv);

QLabel label("<h2><i>Hello,</i>" "<font color = red> Qt!</font> </h2>");

label.show();

return a.exec();

}

**helloworld.cpp**

#include "helloworld\_1\_3.h"

#include "ui\_helloworld\_1\_3.h"

helloworld\_1\_3::helloworld\_1\_3(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::helloworld\_1\_3)

{

ui->setupUi(this);

}

helloworld\_1\_3::~helloworld\_1\_3()

{

delete ui;

}

**1.2.1. Создать кнопку при помощи класса Qpushbutton с текстом quit. При нажатии на кнопку должно происходить завершение программы.**

**Листинг 1.2.1.**

**main.cpp**

#include "quit.h"

#include <QApplication>

#include <QPushButton>

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

{

QApplication a(argc, argv);

QPushButton \*button = new QPushButton("Quit");

QObject::connect(button,SIGNAL(clicked()),&a,SLOT(quit()));

button->resize(200,60);

button->show();

return a.exec();

}

**Скриншот выполнения:**

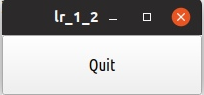


Рисунок 3 - результат выполнения задания № 4

**1.3.1. Создать объект класса Qdate содержащий текущую дату, создать объект класса Qtime, содержащий текущее время, вывести полученную информацию о дате и времени в текстовую метку с использованием объектов класса Qstring.**

**Листинг 1.3.1.**

**main.cpp**

#include "datetime.h"

#include <QApplication>

#include <QPushButton>

#include <QLabel>

#include <QDate>

#include <QTime>

#include <QDateTime>

#include <QTime>

#include <QTimeEdit>

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

{

QApplication a(argc, argv);

QDate dateToDay = QDate::currentDate();

QTime Now= QTime::currentTime();

QString str1,str2,str3;

str1=dateToDay.toString("d/M/yy");

str2=Now.toString("h:m:s");

str3= str1+" "+str2;

QLabel label("<H2><CENTER>"+ str3 + "<H2><CENTER>");

label.show();

return a.exec();

}

**Скриншот выполнения:**

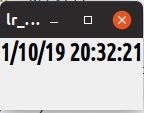


Рисунок 4 - результат выполнения задания № 5

**1.3.2. Используя события таймера через каждые 200мсек выводить в текстовую метку сообщение.**

**Листинг 1.3.2**

**main.cpp**

include "datetime.h"

#include <QApplication>

#include <QLabel>

#include <QDate>

#include <QTime>

#include <QDateTime>

#include <QTime>

#include <QTimerEvent>

class BlinkLabel : public QLabel

{

private:

bool m\_bBlink;

QString m\_strText;

protected:

virtual void timerEvent(QTimerEvent\*)

{

m\_bBlink = !m\_bBlink;

setText(m\_bBlink ? m\_strText : "");

}

public:

BlinkLabel(const QString& strText,

int nIntervel = 200,

QWidget\* pwgt =0

)

:QLabel(strText, pwgt)

, m\_bBlink(true)

,m\_strText(strText)

{

startTimer(nIntervel);

}

};

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

{

QApplication a(argc, argv);

BlinkLabel lbl("<FONT COLOR= RED> <CENTER>Blink</CENETER></FONT>");

lbl.resize(200,60);

lbl.show();

return a.exec();

**Скриншот выполнения:**

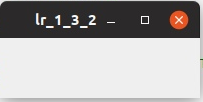


Рисунок 5 - результат выполнения задания № 6

**1.3.3. Используя QtDesigner поместить на форму объект класса Qlabel (текстовую метку), которую необходимо выводить текущее время в режиме онлайн.**

**Листинг 1.3.3.**

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

#include <QLabel>

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

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QLabel>

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

QFont font("Times",28,QFont::Bold);

ui->label->setFont(font);

ui->label->setText(QTime::currentTime().toString("hh:mm:ss"));

timer = new QTimer();

connect(timer,SIGNAL(timeout()),this,SLOT(slotTimerAlarm()));

timer->start(1000);

}

MainWindow::~MainWindow()

{

delete ui;

}

void MainWindow::slotTimerAlarm()

{

ui->label->setText(QTime::currentTime().toString("hh:mm:ss"));

}

**Скриншот выполнения:**

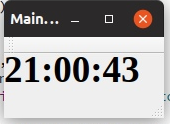


Рисунок 6 - результат выполнения задания № 7

**1.4.1. Используя объекты класса Qfile скопировать содержимое одного бинарного файла в другой, при этом оссуществить проверку на возможность открытия файлов для чтения и записи.**

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

#include <QFile>

#include <QBuffer>

#include <QDebug>

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

{

QApplication a(argc, argv);

QFile file1("file1.dat");

QFile file2("file2.dat");

if (file2.exists())

{

}

if (!file1.open(QIODevice::ReadOnly))

{

qDebug() << "ошибка открытия для чтения";

}

if (!file2.open(QIODevice::WriteOnly))

{

qDebug() << "ошибка открытия для чтения";

}

char b [1];

while (!file1.atEnd())

{

int bBlocksize = file1.read(b,sizeof(b));

file2.write(b, bBlocksize);

}

file1.close();

file2.close();

return a.exec();

}

**Скриншот выполнения:**

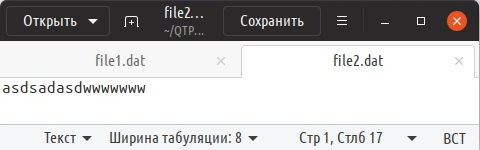
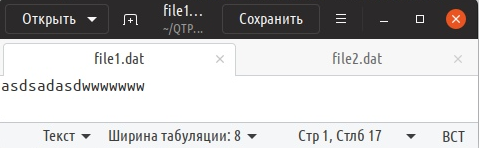


Рисунок 7 - результат выполнения задания №8

**1.4.2. Используя объекты класса Qbytearray скопировать содержимое одного бинарного файла в другой, при этом оссуществить проверку на возможность открытия файлов для чтения и записи.**

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

#include <QFile>

#include <QBuffer>

#include <QDebug>

#include <QByteArray>

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

{

QApplication a(argc, argv);

QFile file1("file1.dat");

QFile file2("file2.dat");

if (file2.exists())

{

}

if (!file1.open(QIODevice::ReadOnly))

{

qDebug() << "ошибка открытия для чтения";

}

if (!file2.open(QIODevice::WriteOnly))

{

qDebug() << "ошибка открытия для чтения";

}

QByteArray b = file1.readAll();

file2.write(b);

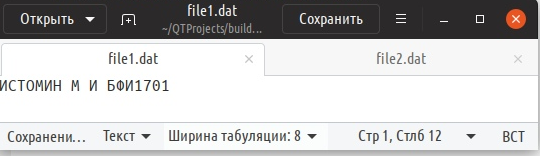
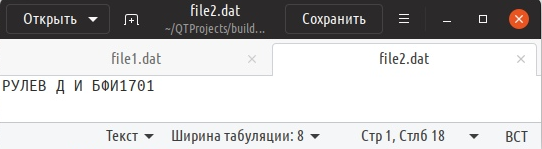
file1.close();

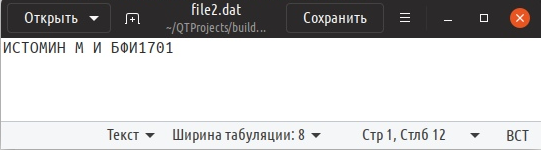
file2.close();

return a.exec();

}

**Скриншот выполнения:**





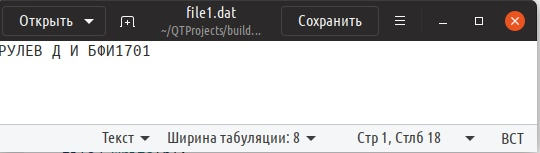


Рисунок 8 - результат выполнения задания № 9

**1.4.3. Используя объекты класса Qbufer осуществить вывод текстовой строки в консоль.**

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

#include <QLabel>

#include <QPushButton>

#include <QFile>

#include <QBuffer>

#include <QDebug>

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

{

QApplication a(argc, argv);

QByteArray arr;

QBuffer buffer(&arr);

buffer.open(QIODevice::WriteOnly);

QDataStream out(&buffer);

out << QString("1111");

qDebug()<<arr;

return a.exec();

}

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

}

MainWindow::~MainWindow()

{

delete ui;

}

**Скриншот выполнения:**



Рисунок 9 - результат выполнения задания № 10

**1.4.4 Используя объекты класса Qtextstream считать содержимое текстового файла и вывести его на консоль.**

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

#include <QPushButton>

#include <QLabel>

#include <QFile>

#include <QBuffer>

#include <QDebug>

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

{

QApplication a(argc, argv);

QFile file ("file.txt");

if (file.open(QIODevice::ReadOnly))

{

QTextStream stream(&file);

QString str;

while (!stream.atEnd())

{

str = stream.readLine();

qDebug() << str;

}

if (stream.status()!= QTextStream::Ok)

{

qDebug() << "Ошибка чтения файла";

}

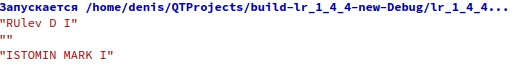
file.close();

}

return a.exec();

}

**Скриншот выполнения:**



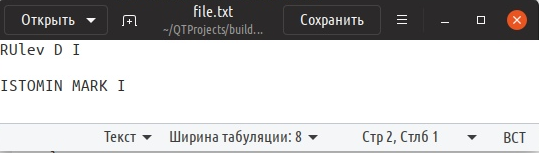


Рисунок 10 - результат выполнения задания № 11

**1.4.5 Используя обхекты класса Qtextstream считать содержимое текстового файла и вывести его на консоль.**

**mian.cpp**

#include "mainwindow.h"

#include <QApplication>

#include <QFile>

#include <QBuffer>

#include <QtDebug>

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

{

QApplication app(argc, argv);

QFile file("myfile.txt");

if (file.open(QIODevice::ReadOnly))

{

QTextStream stream(&file);

QString str = stream.readAll();

qDebug() << str;

if (stream.status() != QTextStream::Ok)

{

qDebug() << "Ошибка чтения файла";

}

}

QFile file2("file.txt");

QString str = "This is a text";

if (file2.open(QIODevice::WriteOnly))

{

QTextStream stream(&file2);

stream << str.toUpper();

file2.close();

if (stream.status() != QTextStream::Ok)

{

qDebug() << "Ошибка записи файла";

}

}

file2.close();

file.close();

return app.exec();

}

**Скриншот выполнения:**

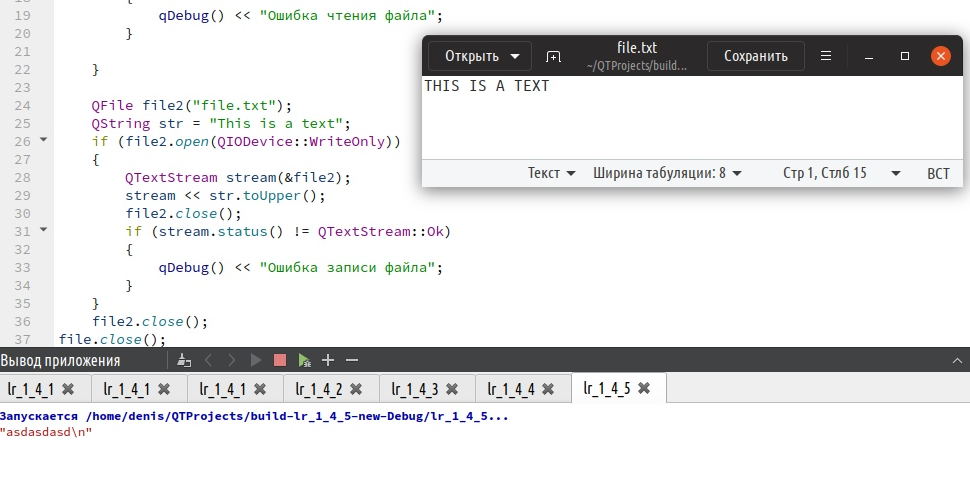


Рисунок 11 - результат выполнения задания № 12

**Вывод**: в ходе данной работы были получены навыки и знания о работе с базовыми элементами Qt. Изучив теоретическую часть, были написаны программы, в которых использовались полученные сведения и элементы среды разработки .