МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Курганский государственный университет»

Кафедра информационных технологий
и методики преподавания информатики

РАЗРАБОТКА ГРАФИЧЕСКОГО ИНТЕРФЕЙСА С ПОМОЩЬЮ БИБЛИОТЕКИ QT

Методические рекомендации
для студентов направлений 44.03.01 «Педагогическое образование»
09.03.03 «Прикладная информатика»

Курган 2017

Кафедра: «Информационные технологии и методика преподавания информатики»

Дисциплина: «Объектно-ориентированное программирование»,
учебная практика
(направление 44.03.01, 09.03.03).

Составители: ст. преподаватель Ю.В. Адаменко,
студентка 4 курса К.А. Криница.

Утверждены на заседании кафедры «26» августа 2015 г.

Рекомендованы методическим советом университета «17» декабря 2015 г.

Лабораторная работа 1. Знакомство со средой Qt Creator и методами создания приложений

Для запуска среды разработки **Qt Creator** выберите **Пуск \ Qt \ Qt Creator**. Окно программы представлено на рисунке 1.1.

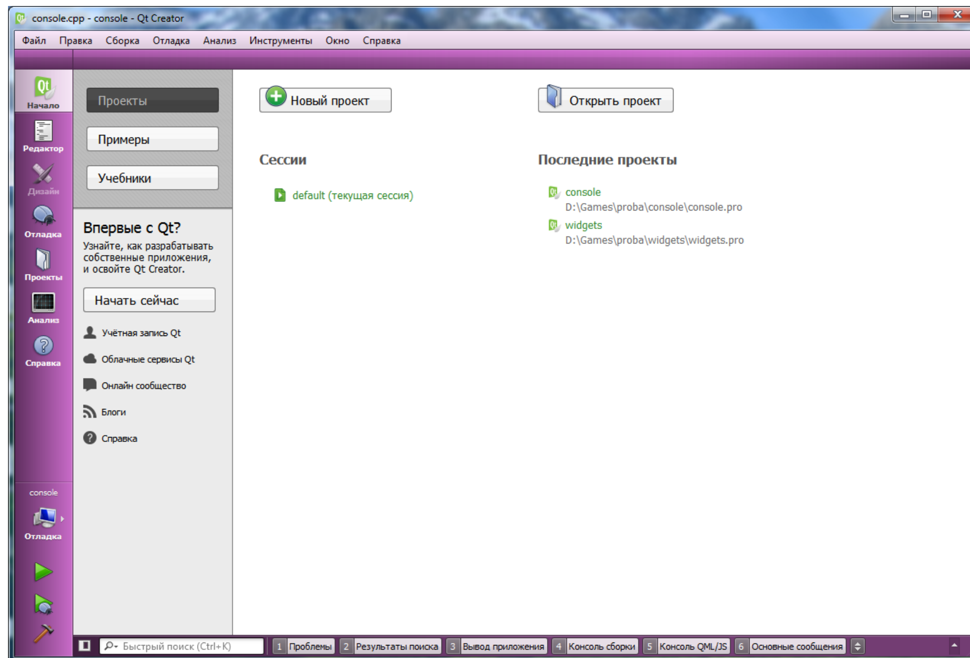


Рисунок 1.1 - Окно **Qt Creator**

При работе в **Qt Creator** вы находитесь в одном из режимов:

1 **Welcome** (Начало) - отображает экран приветствия, позволяя быстро загружать недавние сессии или отдельные проекты;

2 **Edit** (Редактор) - позволяет редактировать файлы проекта и исходных кодов. Боковая панель слева предоставляет различные виды для перемещения между файлами;

3 **Debug** (Отладка) - предоставляет различные способы для просмотра состояния программы при отладке;

4 **Projects** (Проекты) - используется для настройки сборки, запуска и редактирования кода;

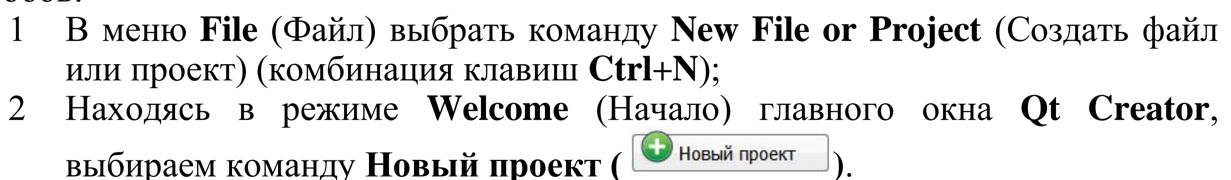
5 **Analyze** (Анализ) - в **Qt** интегрированы современные средства анализа кода разрабатываемого приложения;

6 **Help** (Справка) - используется для вывода документации библиотеки **Qt** и **Qt Creator**.

Задание 1.1. Запустите среду ***QtCreator****.*

Рассмотрим простейшие приёмы работы в среде **Qt Creator** на примере создания консольного приложения. Для этого можно поступить одним из способов:

1 В меню **File** (Файл) выбрать команду **New File or Project** (Создать файл или проект) (комбинация клавиш **Ctrl+N**);

2 Находясь в режиме **Welcome** (Начало) главного окна **Qt Creator**, выбираем команду **Новый проект (** ).

После этого откроется окно **Новый проект** для выбора одного из шаблонов. Для создания простейшего консольного приложения выбираем **Приложение** - **Консольное приложение Qt**.

Далее следует ввести **Название** (имя проекта) и выбрать каталог для его размещения (С**оздать в**). Обратите внимание, что при указании пути к проекту не должно встречаться русских букв и пробелов!

Затем **Комплекты для проекта** оставляем по умолчанию (Desktop).

Жмем «Далее» и «Завершить».

*Задание 1.2. Создайте консольный проект с именем **first**.*

В проекте ***first*** автоматически были созданы файлы **first.pro** (файл проекта):

```
QT += core                //Используется ядро функциональности,
                          //не касающейся GUI
QT - = gui                //Отказ от использования
                          //графического интерфейса
CONFIG += c++11           //Имя текущей версии стандарта C++
TARGET = first            //Имя приложения
CONFIG += console         //Использование консоли в
                          //приложении
TEMPLATE = app            //Тип программы - приложение
SOURCES += main.cpp       //Файлы реализации проекта
```

и **main.cpp** (исполняемый файл):

```
#include <QCoreApplication>      /*класс, представляющий обработку
                          сообщений для консольного приложения*/
int main(int argc, char *argv[])
{
   QCoreApplication a(argc, argv);          /*стандартное создание
                                            экземпляра приложения Qt*/
   return a.exec(); /*возвращаем результат выполнения приложения.
                    Функция exec() - это вход приложения
                    в цикл обработки сообщений */
}
```

Для сохранения текста программы можно воспользоваться командой **Сохранить всё** из меню **Файл**.

Откомпилировать и запустить программу можно одним из следующих способов:

1Пункт меню **Сборка-Запустить**.

2Нажать на клавиатуре комбинацию клавиш **Ctrl+R**.

3Щёлкнуть по кнопке **Запустить**.

*Задание 1.3. Запустите созданный проект **first**.* В результате будет открыто пустое окно. Отредактируем исполняемый файл этого приложения таким образом, чтобы при выполнении выводились строки «Hello Qt!» и «Привет Qt!».

Для этого добавьте следующие строки:

```
#include <QDebug>          //класс для вывода данных в консоль
int main(int argc, char *argv[])
{
   qDebug() << "Hello QT!";
}
```

Чтобы выводить текст в консоли **Windows** на кириллице необходимо:

1 Подключить класс **<qlocale>** и добавить строку **setlocale(LC_ALL, «»)**. Функция **char* setlocale (int category, const char* locale)** устанавливает в программе *локаль* - набор национальных параметров, включающий помимо форматов даты, времени, валюты и т.п., характерных для данной местности, кодировку символов, в ней употребляемую. Используемые нами аргументы в функции **setlocale(LC_ALL, «»)** задают языковой стандарт по умолчанию, т.е. заданную по умолчанию для пользователя кодовую страницу ANSI, полученную от операционной системы.

2 Добавить строку UTF8 **qDebug()<<QString::fromUtf8(«Привет qt!»)**, заранее подключив класс **QString**;

Сохраняем проект и запускаем приложение. Должна появиться консоль с двумя строчками (рисунок 1.2).

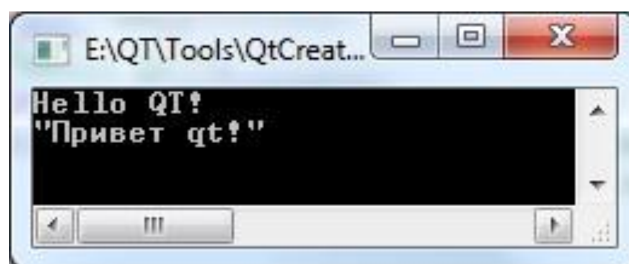


Рисунок 1.2 – Проект ***first***

Закрыть проект можно одним из следующих способов:

- выбрать пункт меню **Файл – Закрыть проект**;
- выбрать пункт меню **Файл – Закрыть все документы и проекты** (если открыто несколько проектов);
- нажать на клавиатуре комбинацию клавиш **Ctrl + Shift + W.**

Закройте проект ***first.***

Перейдём к созданию приложений с графическим интерфейсом.

Разработаем первое приложение, задача которого по нажатию кнопки отображать в надписи текст, введенный в поле ввода, изменяя его стиль (цвет текста, шрифт размер шрифта).

Задание 1.4. Создадим приложение, которое по нажатии на кнопку отображает в виджете надписи введенный текст.

1 Создайте приложение **Qt Widgets** с именем ***widgets*** аналогично шагам по созданию консольного приложения **Qt**. Когда откроется диалог **Информация о классе**, в поле **Имя класса** введите ***widgets***. В списке **Базовый класс** выберите **QMainWindow**. Нажмите кнопку Далее.

В проекте ***widgets*** автоматически будут созданы файлы: **widgets.pro, widgets.h, widgets.cpp, main.cpp, widgets.ui**.

2 Разработайте интерфейс пользователя:

1) в режиме **Редактор** дважды нажмите на файле **widgets.ui**;

2) перетащите следующие виджеты на форму: **Label** (метка), **Line Edit** (поле для ввода), **Push Button** (кнопка);

3) дважды нажмите на виджет **label** и введите текст **Text**;

4) дважды нажмите на виджет **pushButton** и введите текст **Start**;

5) выделите виджеты **lineEdit** и **pushButton** и выберите действие в меню над формой «Скомпоновать по горизонтали ()», затем выделите **label** и скомпонуйте элементы по вертикали (). Применение вертикальной и горизонтальной

компоновок обеспечивает масштабирование интерфейса приложения при различных разрешениях экрана;

6) в свойствах главного окна, выберите свойство *WindowTitle* и переименуйте окно в «Widgets»;

7) используя свойство *alignment* виджета **label** и свойство *styleSheet* виджетов **lineEdit** и **pushButton,** отредактируйте форму, как на рисунке 1.3.

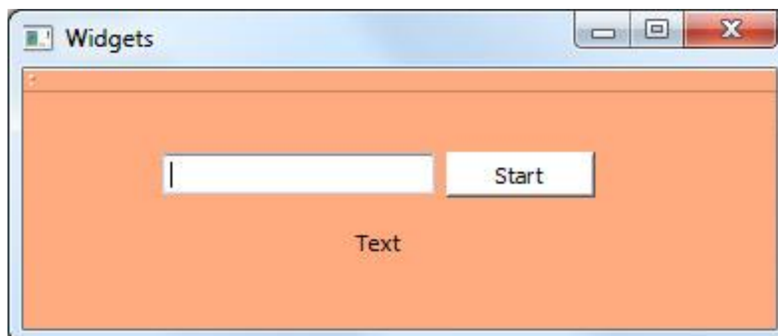


Рисунок 1.3 – Форма проекта ***widgets***

Сохраните проект и запустите для просмотра.

3 Перейдем к слоту, который будет вызываться в ответ на событие **clicked**().

При нажатии кнопки **Start** вы будете использовать механизм сигналов и слотов Qt. Сигнал вырабатывается, когда происходит определённое событие, а слот – это функция, которая вызывается в ответ на определённый сигнал. У виджетов Qt есть предопределённые сигналы и слоты которые вы можете использовать прямо из режима Дизайна. Чтобы перейти к слоту:

– нажмите правой кнопкой мыши на кнопке **Start** для открытия контекстного меню;
– выберите **Перейти к слоту... > clicked()** и нажмите **OK**.

Закрытый слот **`on_pushButton_clicked()`** будет добавлен в заголовочный файл **widgets.h**, и закрытая функция **`Widgets::on_pushButton_clicked()`** будет добавлена в файл исходных кодов **widgets.cpp**.

4 Перейдём к файлу исходных кодов **widgets.cpp**. Добавим в слот программный код:

```
ui→label→setText(ui→lineEdit→text());
ui→label→setStyleSheet("color: blue ;
                        font-family: Comic Sans MS;
                        font-size: 20px");
ui→lineEdit→clear();
```

В первой строке получаем доступ к виджету **Label** (**ui→**), расположенному на интерфейсе пользователя, а виджет **Label** получает текст, введенный пользователем из виджета **lineEdit**.

Во второй строке изменяем стиль для виджета **Label**.

И в последней строчке при нажатии на кнопку виджет **lineEdit** будет очищен от текста.

5 Сохраните изменения в проекте и запустите приложение. Результат работы приложения представлен на рисунке 1.4.

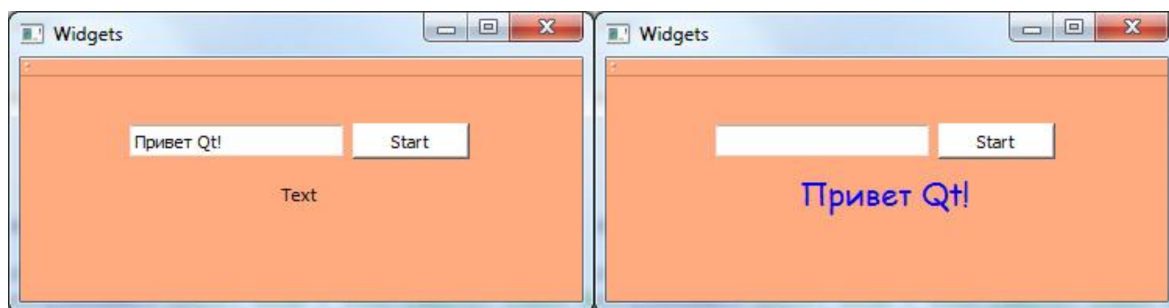


Рисунок 1.4 – Проект ***widgets***

Задание 1.5. Создадим приложение, которое позволит пользователю рассчитывать площадь прямоугольника.

1 Создайте приложение **Qt Widgets** с именем ***rectangle***. Имя класса также ***rectangle***.

2 Разработайте интерфейс пользователя, как показано на рисунке 1.5.

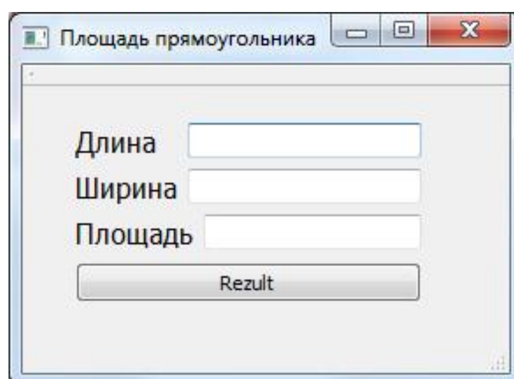


Рисунок 1.5 – Форма проекта ***rectangle***

3 Измените фон главного окна, шрифт и цвет виджетов **Label.**

4 В свойствах **lineEdit_3** поставьте галочку напротив свойства **ReadOnly**, чтобы пользователь не имел возможности изменять значение результата.

4 Перейдите к слоту функции расчета площади прямоугольника.

5 Добавьте программный код в слот:

```
QString firstNumStr = ui →lineEdit→text();        //Получаем текст из первой
QString secondNumStr = ui→lineEdit_2→text();   //и второй строк
bool ok;
```

/*Проводим проверку на ввод недопустимых символов (например, если пользователь не ввел число, ввел буквы или отрицательное число). Для этого переводим полученный текст в вещественное число и заносим в переменную вещественного типа. В случае ошибки выводится информационное окно класса *QMessageBox**/

```
float firstNum = firstNumStr.toFloat(&ok);
if (!ok) {
  QMessageBox::information(this,"Ошибка", "Введите число!");
  return; }
float secondNum = secondNumStr.toFloat();
if (!ok) {
  QMessageBox::information(this,"Ошибка", "Введите число!");
  return; }
```

```
if (secondNum <= 0 || firstNum <= 0) {
  QMessageBox::information(this,"Ошибка", "Отрицательное число!");
  return; }
//Создаем переменную для хранения результата
float result = firstNum * secondNum;
//Переводим результат в строку и записываем в виджет lineEdit_3
ui → lineEdit_3→setText(QString::number(result));
```

Сохраните изменения в проекте и запустите приложение.

Задание 1.6. Самостоятельное решение задач по вариантам. Реализовать приложения с графическим интерфейсом пользователя.

Вариант 1.1

1 Число, лежащее в диапазоне от -999 до 999, вводится в lineedit. Вывести информационное сообщение – словесное описание данного числа вида «отрицательное двузначное число», «нулевое число», «положительное однозначное число» и т.д.

2 Значения переменных X, Y, Z (переменные вводятся в lineedit) поменять местами так, чтобы они оказались упорядоченными по убыванию.

Вариант 1.2

1 Заменить наименьшее из трех чисел (числа вводятся в lineedit) суммой двух других чисел и вывести результат в label.

2 Даны две переменные целого типа: A и B (переменные вводятся в lineedit). Если их значения не равны, то присвоить каждой переменной сумму этих значений, а если равны, то присвоить переменным нулевые значения.

Вариант 1.3

1 Заменить наибольшее из трех чисел (числа вводятся в lineedit) разностью двух других чисел и вывести результат в label.

2 Даны две переменные целого типа: A и B (переменные вводятся в lineedit). Если их значения не равны, то присвоить каждой переменной максимальное из этих значений, а если равны, то присвоить переменным нулевые значения.

Вариант 1.4

1 Из трех данных чисел выбрать наименьшее и наибольшее (числа вводятся в lineedit), и заменить третье число их разностью (число должно быть изменено в текущем lineedit).

2 Даны три переменные: X, Y, Z (переменные вводятся в lineedit). Если их значения упорядочены по убыванию, то удвоить их; в противном случае заменить значение каждой переменной на противоположное.

Вариант 1.5

1 Перераспределить значения переменных X и Y (переменные вводятся в lineedit) так, чтобы в X оказалось меньшее из этих значений, а в Y – большее.

2 Даны три переменные: X, Y, Z (переменные вводятся в lineedit). Если их значения упорядочены по возрастанию или убыванию, то удвоить их; в противном случае заменить значение каждой переменной на противоположное.

Лабораторная работа 2. Работа с контейнерами в среде Qt Creator: QVector

Библиотека **Qt** предоставляет набор контейнерных классов общего назначения. Эти классы могут использоваться для хранения элементов

определенного типа. Например, если вам необходим массив изменяемого размера, содержащий **QString**, используйте **QVector**<**QString**>.

Контейнерные классы – классы с неявным совместным использованием данных, они оптимизированы для быстрой работы, низкого потребления памяти и минимального увеличения кода (**inline**), результат в меньшем исполняемом файле.

Qt предоставляет конструкцию **foreach**, которая позволяет очень легко перебрать все элементы, хранящиеся в контейнере.

Вектор – структура данных, очень похожая на обычный массив. Однако использование класса вектора предоставляет некоторые преимущества по сравнению с простым массивом. Например, можно узнать количество элементов внутри вектора (его размер), или динамически расширять его. Еще этот контейнер экономнее, чем другие виды контейнеров.

Добавлять элементы в вектор можно следующими способами:

1 Если нам заранее известно необходимое количество элементов в векторе, мы можем задать начальный размер при его определении и использовать оператор **[ ]** для заполнения вектора элементами:

Пример 2.1

```
QVector <float> vect1(3);
vect1[0] = 1.0;
vect1[1] = 0.5;
vect1[2] = -0.4;
qDebug() <<  vect1;
```

Пример 2.2

```
// QVector(3,3,3,3,3)
QVector <int> vect2(5,3);
qDebug() << vect2;
```

2 Для добавления элементов в конец последовательного контейнера необходимо объявить пустой вектор и использовать методы ***push_back()*** или ***append()***:

Пример 2.3

```
QVector<int> vect3;
vect3.push_back(10);
vect3.push_back(20);
vect3.push_back(30);
qDebug() << vect3;
```

Пример 2.4

```
QVector <float> vect4;
vect4.append(34.0);
vect4.append(0.5899);
vect4.append(-0.678);
qDebug() << vect4;
```

Пример 2.5

```
QVector<QString>vec5;
vect5.append("one");
vect5.append("two");
vect5.append("three");
qDebug() << vect5;
```

Задание 2.1. Создайте консольное приложение ***vector****. Подключите все необходимые классы и заполните векторы пятью способами. Сохраните изменения и запустите приложение на выполнение.*

Другие операции, доступные контейнеру **QVector**, приведены в таблице 2.1.

Таблица 2.1 – Часто используемые методы контейнера **QVector <T>**

Метод	**Описание**
push_back()	Добавление элемента в конец последовательного контейнера.
clear()	Удаление всех элементов из вектора и освобождение памяти, используемой вектором
remove()	Удаляет элемент в позиции с индексом *i*
replace()	Заменяет элемент в позиции с индексом *i* на *value*
insert()	Вставляет значение *value* в позицию с индексом *i* в векторе
size()	Возвращает количество элементов в векторе

Задание 2.2. Создайте консольное приложение ***vect_sum****, которое позволит находить сумму элементов вектора, а также продемонстрирует применение методов* ***replace()*** *и* ***size()****.*

Для этого добавьте следующие строки:

```
QVector<int> vec;                 // создаем вектор из 11 элементов от 1 до 10
for (int i=0; i<=10; i++ ) {
   vec.push_back(i); }
qDebug() <<  vec;
for (int i=0; i<=10; i++ ) {      // заменяем элемент вектора «2» на «65»
   vec.replace(2,65); }
qDebug() <<  vec;
int sum=0;                        // находим сумму элементов
for (int i=0; i<vec.size(); i++ ) {  // от 0 до количества элементов в векторе
   sum += vec[i]; }
qDebug() <<"Sum = " << sum;
```

Сохраните изменения и запустите приложение на выполнение.

Задание 2.3. Создайте консольное приложение ***vect_sum2****, которое позволит находить сумму положительных элементов вектора, введенных пользователем*

Для этого добавьте следующие строки:

```
qDebug() << "Введите пять целых чисел: ";
//Класс QTextStream используется для чтения и записи текста
QTextStream in(stdin);                 // указываем клавиатуру устройством ввода
QVector<int> vec2;
for (int i = 0; i < 5; i++) {
   int num = (in.readLine()).toInt();  // считываем строку и
                                       //преобразовываем её в целое
   vec2.append(num); }                 // добавляем элементы в конец массива
qDebug() << vec2;
int sum2=0;
for (int i=0; i<vec2.size(); i++ ) {
   if (vec2[i] > 0) {sum2 += vec2[i]; } }
qDebug() << "Сумма положительных чисел = " << sum2;
```

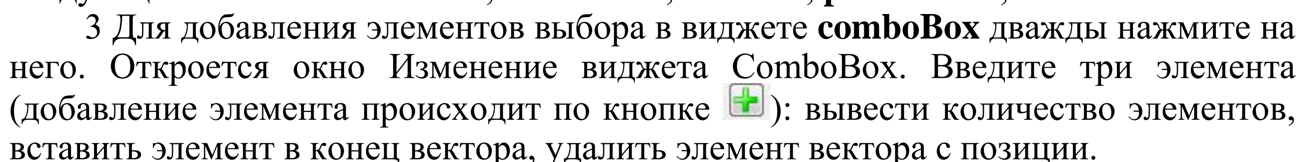
Сохраните изменения и запустите приложение на выполнение.

Использование других методов контейнера **QVector** рассмотрим в графическом приложении.

Задание 2.4. Создадим приложение, которое позволит узнать размер вектора, выведенного на экран, а также удалять и добавлять элементы.

1 Создайте приложение **Qt Widgets** с именем ***vectorgui***. Имя класса ***vectorgui***.

2 Разработайте интерфейс пользователя, для этого расположите на форме следующие элементы: **textEdit**, **comboBox**, **lineEdit**, **pushbutton**, **label**.



3 Для добавления элементов выбора в виджете **comboBox** дважды нажмите на него. Откроется окно Изменение виджета ComboBox. Введите три элемента (добавление элемента происходит по кнопке): вывести количество элементов, вставить элемент в конец вектора, удалить элемент вектора с позиции.

4 Запретим изменять размеры окна. Для этого в свойствах формы **minimumSize** и **maximumSize** установите нужные параметры. Если вы не знаете длину и ширину своей формы, то параметры можно посмотреть в свойстве **geometry**.

Пример формы представлен на рисунке 2.1 *(цвет фона, шрифт… установите по своему желанию).*

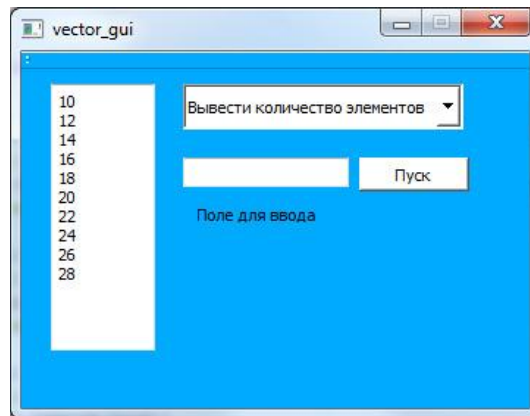


Рисунок 2.1 – Форма проекта ***vectorgui***

5 Сохраните изменения и запустите приложение для просмотра.

6 Перейдем к составлению программы. Добавим программный код для работы нашего приложения.

1) в переменных закрытого типа (**private**) заголовочного файла ***vectorgui.h*** объявите экземпляр класса **QVector**: QVector<int> vec;

2) в файле ***vectorgui.cpp,*** в конструкторе главного окна, добавьте код для заполнения вектора и вывода его в **textEdit** при запуске приложения:

```
QString st;
for (int i = 0; i < 10; i++) {
   vec.append(10+i*2);
   ui→textEdit→append(QString::number(vec[i]) + " "); }
ui→textEdit→setReadOnly(true); // textEdit  только для чтения
```

3) Сохраните изменения и запустите для просмотра.

7 Перейдите к слоту, который будет вызываться в ответ на событие ***clicked()***. В результате будет создана закрытая функция **vectorgui::on_pushButton_clicked()**.

Опишем в ней действия, которые должны происходить при выборе элемента **comboBox**:

```
// получение индекса выбранного элемента
switch (ui→comboBox→currentIndex()) {
   case 0: {
      int size = vec.size();
      // выводим размер вектора в lineEdit
      ui→lineEdit→setText("Vector size: " + QString::number(size));
      break;    }
   case 1: {     // получаем значение и добавляем в конец вектора
      int num = ui→lineEdit→text().toInt();
      vec.push_back(num);
      break;     }
   case 2: {
      int pos = ui→lineEdit→text().toInt();
      vec.remove(pos);    //удаляем элемент с заданной позиции
      break;    }  }
//    перезаписываем вектор
```

```
ui→textEdit→clear();
for (int i = 0; i < vec.size(); i++) {
   ui→textEdit→append(QString::number(vec[i]) + " "); }
```

8 Перейдите к слоту, который будет вызываться в ответ на событие ***currentIndexChange()*** виджета **comboBox,** и добавьте в слот код (для очищения виджета **lineEdit** и проверки условия):

```
ui→lineEdit→clear();
ui→comboBox→currentIndex() == 0 ?
      ui→lineEdit→setReadOnly(true) :
      ui→lineEdit→setReadOnly(false);
```

Сохраните изменения в приложении и просмотрите результат.

*Задание 2.5. Модернизируйте проект **vectorgui**:*

а) добавьте защиту на ввод недопустимых символов с применением класса QMessageBox;

*б) предусмотрите изменение текста виджета **label** при выборе 2 и 3 элементов **comboBox** («поле для ввода» → «введите значение элемента: » ...);*

в) при удалении элемента предусмотрите проверку, что элемент с введенным индексом существует.

До этого момента мы рассматривали только одномерные векторы, т.е. к элементу вектора мы обращались через один индекс. Двумерный вектор в **Qt** – это вектор одномерных векторов.

Рассмотрим консольный вариант двумерного вектора.

*Задание 2.6. Создайте консольное приложение **matrix**, которое позволит создавать двумерный вектор, содержащий случайны значения, и находить максимальный элемент.*

Для этого добавьте следующие строки:

```
QVector< QVector<int> > matrix;
srand(time(0));
for (int i = 1; i <= 3; i++) {
   QVector<int> line;
   for (int j =1; j <= 3; j++) {
      line.append(rand() % 100 + 100);    }
   matrix.append(line);}
QTextStream stream(stdout);
for (int i = 0; i < matrix.length(); i++) {
   QString str = "";
   for (int j = 0; j < matrix[i].length(); j++) {
      str += QString::number(matrix[i][j])+ " ";    }
   stream << str << "\n";
   stream.flush(); }
int max = matrix[0][0];
for (int i = 0; i < matrix.length(); i++) {
   for (int j = 0; j < matrix[i].length(); j++) {
      if (matrix[i][j] > max) {max = matrix[i][j];}    }  }
stream << "Максимальный элемент:" << max << "\n";
stream.flush();
```

Объявляем экземпляр класса **QVector**, содержащий другие вектора с целочисленными значениями.

Чтоб каждый раз генерировались новые случайные числа, необходимо, что бы менялся аргумент в функции ***srand()***. Чтобы производить рандомизацию автоматически, т.е., не меняя каждый раз аргумент в функции ***srand()***, нужно воспользоваться функцией ***time()*** с аргументом 0 (чтобы использовать функцию ***time()***, необходимо подключить заголовочный файл <***ctime***>). Теперь при каждом срабатывании программы будут генерироваться совершенно случайные числа.

Создаем экземпляр класса **QVector** и, двигаясь вправо (по столбцам), заполняем его случайными значениями в диапазоне от 100 до 200 включительно. И затем созданный вектор ***line*** добавляем в вектор ***matrix***.

В экземпляре класса **QTextStream** указываем, что монитор – устройство вывода. Выводим матрицу на экран (с пробелами между значениями). Функция ***flush()*** сбрасывает все буферы данных, ожидающие записи на устройство, т.е. выводит содержимое вне зависимости от того, насколько заполнен буфер.

Сохраните изменения и запустите приложение на выполнение. Находим максимальный элемент и выводим его на экран. Пример работы программы на рисунке 2.2.

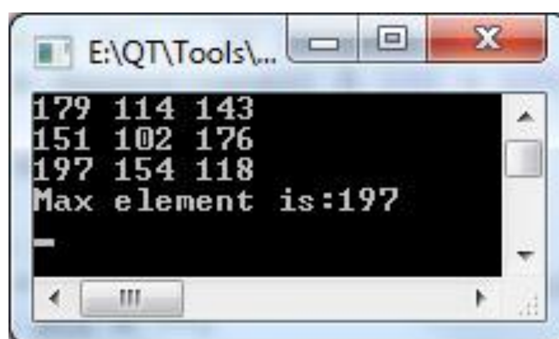


Рисунок 2.2 – Проект ***matrix***

И рассмотрим графический пример заполнения матрицы.

Задание 2.7. Создадим приложение, которое покажет, что каждый вектор, хранящийся в векторе матрицы, может быть произвольной длины.

1 Создайте приложение **Qt Widgets** с именем ***matrixgui***. Имя класса ***matrixgui***.

2 Разработайте интерфейс пользователя, для этого расположите на форме следующие элементы: **textEdit**, **pushbutton**.

3 Установите запрет на изменение размеров окна.

Пример формы на рисунке 2.3 *(цвет фона, шрифт… установите по своему желанию).*

4 Сохраните изменения и запустите для просмотра.

5 Перейдем к составлению программы. Добавим программный код для работы нашего приложения:

1) в переменных закрытого типа (**private**) заголовочного файла ***matrixgui.h*** объявите экземпляр класса **QVector**: **QVector <QVector <int> > matrix;** и переменную **int strCount**;

2) в файле ***matrixgui.cpp,*** в конструкторе главного окна, добавьте код для заполнения вектора и вывода его в **textEdit** при запуске приложения:

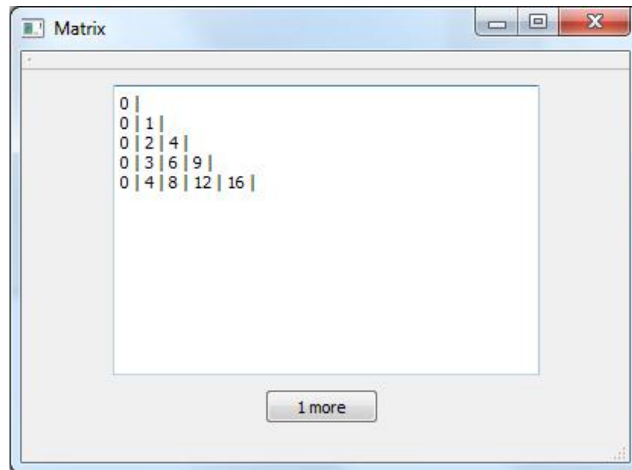


Рисунок 2.3 – Форма проекта ***matrixgui***

```
for (int i = 0; i < 5; i++) {
   QVector<int> vec;
   for (int j = 0; j <= i; j++) {
      vec.append(i*j);    }
   matrix.append(vec); }
strCount = 5;
ui→textEdit→clear();
for (int i = 0; i < matrix.length(); i++) {
   QString str = "";
   for (int j = 0; j < matrix[i].length(); j++) {
      str += QString::number(matrix[i][j])+" | ";    }
   ui→textEdit→append(str);}
```

Сохраните изменения и запустите для просмотра.

3) перейдите к слоту, который будет вызываться в ответ на событие ***clicked()*** и добавьте в него код для работы нашего приложения:

```
QVector<int> line;
for (int j = 0; j <= strCount; j++) {   // заполнение вектора значениями
   line.append(strCount*j); }
matrix.append(line);                    // добавление в матрицу вектора
strCount++;
if (strCount > 10) {                    //если количество строк >10 стираем матрицу
   matrix.clear();
   strCount = 0;}                       // количество строк устанавливаем в 0
ui→textEdit→setText("strCount: " + QString::number(strCount));
for (int i = 0; i < matrix.length(); i++) {
   QString str = "";
   for (int j = 0; j < matrix[i].length(); j++) {
      str += " "+QString::number(matrix[i][j])+" | ";    }
   ui->textEdit->append(str); }    // вывод в textEdit
```

Сохраните изменения в приложении и просмотрите результат.

Задание 2.8. Самостоятельное решение задач по вариантам. Реализовать для одномерных массивов консольный вариант приложения, для двумерных – с графическим интерфейсом. В консольном приложении значения элементов массива вводятся пользователем, в графическом – задаются произвольно.

Вариант 2.1

1 Дан массив размера N. Вывести его элементы в обратном порядке.

2 Дано число k (0 < k < 11) и матрица размера m x n. Найти сумму и произведение элементов k-го столбца данной матрицы (нахождение суммы или произведения определяется пользователем в виджете comboBox).

Вариант 2.2

1 Дан массив размера N. Вывести вначале его элементы с четными индексами, а затем – с нечетными.

2 Дана матрица размера m x n. Найти суммы элементов всех ее четных и нечетных столбцов (нахождение суммы четных/нечетных столбцов определяется пользователем в виджете comboBox).

Вариант 2.3

1 Дан целочисленный массив A. Вывести номер первого из тех его элементов A[i], которые удовлетворяют двойному неравенству: A[1] < A[i] < A[10]. Если таких элементов нет, то вывести 0.

2 Дана матрица размера m x n. Найти минимальное и максимальное значение в каждой строке (нахождение максимального/минимального значения определяется пользователем в виджете comboBox).

Вариант 2.4

1 Дан целочисленный массив размера N. Преобразовать его, прибавив к четным числам первый элемент. Первый и последний элементы массива не изменять.

2 Дана матрица размера m x n. В каждой строке найти количество элементов, больших среднего арифметического всех элементов этой строки.

Вариант 2.5

1 Дан целочисленный массив размера N. Вывести вначале все его четные элементы, а затем – нечетные, сохраняя порядок следования элементов.

2 Дана матрица размера m x n. Преобразовать матрицу, поменяв местами минимальный и максимальный элемент в каждой строке.

Лабораторная работа 3. Работа с контейнерами в среде Qt Creator: QList, QLinkedList

Для перемещения по элементам контейнера предназначен итератор. ***Итераторы*** позволяют абстрагироваться от структуры данных контейнеров, т.е. если в какой-либо момент вы решите, что применение другого типа контейнера было бы гораздо эффективнее, то все, что вам нужно будет сделать – это просто заменить тип контейнера. На остальном коде, использующем итераторы, это никак не отразится. **Qt** предоставляет два стиля итераторов: итераторы в стиле **Java**; итераторы в стиле **STL**.

Пример вывода элементов контейнера в прямом порядке:

```
QVector<QString> vec;
setlocale(LC_ALL,"");
vec <<"один" << "два" << "три";
QVector<QString>::iterator it = vec.begin();
for (; it != vec.end(); ++it) {
        qDebug() << "Элемент:"<< *it ;}
```

Вызов метода **begin()** из объекта контейнера возвращает итератор, указывающий на первый его элемент, а вызов метода **end()** возвращает итератор,

указывающий на воображаемый несуществующий элемент, следующий за последним.

Пример вывода элементов контейнера в обратном порядке:

```
QVector<QString>::iterator id = vec.end();
for (;id != vec.begin();) {
   --id;
   qDebug() << "Элемент:" << *id;}
```

Обратите внимание: прохождение элементов в обратном порядке при помощи оператора -- не симметрично с прохождением при помощи оператора ++.

Рассмотрим применение итераторов при работе со списками. **QList<T>** – наиболее часто используемый контейнер.

Список – это структура данных, представляющая собой упорядоченный набор связанных друг с другом элементов. Преимущество списков перед векторами и очередями состоит в том, что вставка и удаление элементов в любой позиции происходит эффективнее, так как для выполнения этих операций изменяется только минимальное количество указателей; исключение составляет только вставка элемента в центр списка. Но есть и недостаток: списки плохо приспособлены для поиска определенного элемента списка по индексу, и для этой цели лучше всего использовать вектор.

Списки реализует шаблонный класс **QList**. В общем виде данный класс представляет собой массив указателей на элементы.

Задание 3.1. Создайте консольное приложение ***iterators.*** *Добавьте программный код для вывода элементов контейнера* ***QVector*** *в прямом и обратном порядке. Просмотрите результат. Затем измените экземпляр класса QVector на QList и сравните результаты.*

Добавлять элементы в список можно следующими способами:

1 Если нам заранее известно необходимое количество элементов в списке и их значения:

```
QList <int> list;
list << 10 << 20 << 30;
qDebug() << list;
```

2 Для добавления элементов в конец последовательного контейнера необходимо объявить пустой список и использовать методы ***push_back()*** или ***append():***

```
QList <QString> list2;
list2.append( "one");
list2.append( "two" );
list2.push_back("three");
qDebug() << list2;
```

Задание 3.2. Создайте консольное приложение ***list****. Подключите все необходимые классы и заполните списки двумя способами (при выводе списка используйте итераторы)*.

Контейнер **QList** имеет следующие операции (таблица 3.1).

Таблица 3.1 – Операции контейнера **QList<T>**

Метод	Описание
move()	Перемещает элемент с одной позиции на другую.
removeFirst()	Выполняет удаление первого элемента списка.

Продолжение таблицы 3.1

Метод	Описание
removeLast()	Выполняет удаление последнего элемента списка.
swap()	Меняет местами два элемента списка на указанных позициях.
takeAt()	Возвращает элемент на указанной позиции и удаляет его из списка.
takeFirst()	Возвращает первый элемент и удаляет его из списка.
takeLast()	Возвращает последний элемент и удаляет его из списка.

*Задание 3.3. Создайте консольное приложение **list2**, в котором рассмотрим функции **move()** и **takeFirst()**, а также научимся менять значения элементов контейнера с использованием итераторов.*

Для этого добавьте следующие строки:

```
QList <int > list;
list << 3 << 4 << 5 << 6 << 7;              // вносим элементы в список
qDebug() << list;
int firstNumber = list.takeFirst();  //помещаем возвращенное значение в
                                     //переменную и удаляем из списка первый элемент
qDebug() <<"firstNumber:  " << firstNumber;
qDebug() << list;
list.move(0,1);              // перемещаем элемент с нулевой позиции на первую
QList<int>::iterator it = list.begin();
   while (it != list.end())   {
          qDebug() << *it;
          ++it;        }
qDebug() << endl;
QList<int>::iterator ip = list.begin()+1; // ставим итератор на первую позицию
   while (ip != list.end()-1) {   // пока не достигнем последнего элемента
           *ip *=(*ip);  // возводим в квадрат
            ++ip;        }
ip = list.begin();  // выводим список
while (ip != list.end())  {
    qDebug() << *ip;
     ++ip;          }
```

Сохраните изменения и запустите приложение на выполнение.

Использование других методов контейнера **QList** рассмотрим в графическом приложении.

Задание 3.4. Создадим приложение, которое позволит удалять и добавлять элементы, а также менять их местами.

1 Создайте приложение **Qt Widgets** с именем ***listgui***. Имя класса ***listgui***.

2 Разработайте интерфейс пользователя, для этого расположите на форме следующие элементы: **textEdit**, **pushbutton**, **radioButton (3)**, **lineEdit (2)**.

3 Переименовать виджеты **radioButton** можно двумя способами: двойным кликом по виджету или в свойстве **text**. Дайте названия виджетам: удалить последний элемент; вставить элемент в начало; поменять элементы на позициях.

4 Установите запрет на изменение размеров окна.

Пример формы на рисунке 3.1 *(цвет фона, шрифт… установите по своему желанию).*

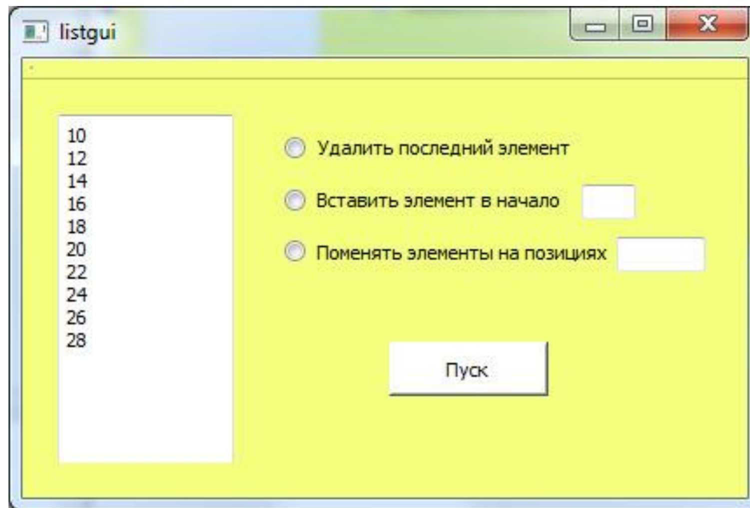


Рисунок 3.1 – Форма проекта ***listgui***

5 Перейдем к составлению программы. Добавим программный код для работы нашего приложения:

1) в заголовочном файле объявите экземпляр класса **QList**: **QList<int> list**.

2) добавьте код для заполнения списка и вывода его в **textEdit** при запуске приложения:

```
for (int i = 0; i < 10; i++) {
   list.append(10+i*2);
   ui→textEdit→append(QString::number(list.at(i)) + " ");    }
```

Поскольку мы не собираемся изменять значения элементов, то из соображений эффективности не рекомендуется использовать оператор индексации []. Вместо этого лучше будет воспользоваться методом ***at()***, т.к. этот метод возвращает константную ссылку на элемент.

3) сохраните изменения и запустите для просмотра.

4) перейдите к слоту, который будет вызываться в ответ на событие ***clicked()*** и добавьте в него код для работы нашего приложения:

```
if (ui→radioButton→isChecked()) {
  list.removeLast(); //если выбран первый radioButton, удаляем последний элемент
   } else if (ui→radioButton_2→isChecked()) {
      int n=ui→lineEdit→text().toInt();
     list.push_front(n);  //добавляем введенный в lineEdit элемент в начало
   } else if (ui→radioButton_3→isChecked()) {
      int a,b;
      QStringList st = ui→lineEdit_2→text().split(",");
      a = st[0].toInt();
      b = st[1].toInt();
      list.swap(a,b);   }
```

Функция ***split()*** разбивает строку на подстроки, как только доходит до запятой, и возвращает список этих строк.

```
ui→textEdit→clear();                // перезаписываем список
QList<int>::iterator it = list.begin();
while ( it != list.end()) {
   ui→textEdit→append(QString::number(*it) + " ");
   it++;    }
```

Сохраните изменения в приложении и просмотрите результат.

*Задание 3.5. Модернизируйте проект **listgui**:*

а) добавьте защиту на ввод недопустимых символов;

б) при перемещении элементов предусмотрите проверку, что элементы с введенными индексами существуют;

*в) виджет **lineEdit** должен очищаться при переходе на другой виджет **radioButton**;*

*г) при удалении последнего элемента предусмотрите проверку на наличие элементов в списке, используя функцию **isEmpty()**, в случае отсутствия элементов в списке, выведите надпись «Список пуст!».*

В классе **QList** не достаточно эффективно работает вставка элементов, поэтому если вы работаете с большими списками, и/или вам часто требуется вставлять элементы, то эффективнее будет использовать двусвязные списки **QLinkedList**. Хотя этот контейнер расходует больше памяти, чем **QList**, зато операции вставки и удаления сводятся к переопределению четырех указателей независимо от позиции удаляемого или вставляемого элемента.

Также **QLinkedList** отличается от **QList** тем, что при работе для доступа к элементам нужно использовать итераторы.

Примечательно, что итераторы можно использовать со стандартными алгоритмами **STL**, определенными в заголовочном файле **algorithm**.

*Задание 3.6. Создайте консольное приложение **linkedlist**. В приложении **linkedlist** мы рассмотрим некоторые стандартные алгоритмы **STL**, а именно:*

а) **`min_element`** *и* **`max_element`** *– нахождение минимального и максимального элемента ;*

*б) **find (first, last, value)** – возвращает итератор, указывающий на первый элемент, равный значению **value**;*

*в) **remove(first,last,value)** – удаление из диапазона всех значений, равных **value**. В действительности **remove** ничего не удаляет, так как ему не передается контейнер. Элементы могут удаляться лишь функциями контейнера, отсюда следует и главное правило: чтобы удалить элементы из контейнера, вызовите **`erase`** после **`remove`**.*

Для этого добавьте следующие строки:

```
using namespace std;
QLinkedList<int> list;   //заполнение списка
for (int i = 1; i < 10; i++) {
   list << abs(5-i);  }
QLinkedList<int>::iterator it;
it = list.begin();
QString str = " ";
while (it != list.end()) {
   str += QString::number(*it) + " "; // преобразовываем элементы списка
                                      // в строку и + к созданной строке
   it++;  }
qDebug() << str;           // вывод строки, содержащей элементы списка
//нахождение минимального и максимального элемента
int min = *min_element(list.begin(), list.end());
int max = *max_element(list.begin(), list.end());
qDebug() << "Минимальный элемент:" << min;
```

```
qDebug() << "Максимальный элемент:" << max;
list.erase(remove(list.begin(), list.end(),0),list.end()); // удаление 0
str = " ";
it = list.begin();
while (it != list.end()) {
   str += QString::number(*it) + " ";
   it++;  }
qDebug() << str;
QLinkedList<int>::iterator iter;
iter = find(list.begin(), list.end(), 4);      // находим «4»
*iter *= 10;                                    // умножаем на 10
iter = find(list.begin(), list.end(), 2);
*iter *= 10;
it = list.begin();
str = " ";
while (it != list.end()) {
   str += QString::number(*it) + " ";
   it++;  }
qDebug() << str;
```

Сохраните изменения и запустите приложение на выполнение.

*Задание 3.7. Создайте консольный проект **linkedlist2**, объявите экземпляр класса **QLinkedList <int> list**, заполните его случайными значениями и продемонстрируйте в нем работу других алгоритмов **STL**:*

*а) **count(first, last, value)** − возвращает, сколько раз элемент со значением **value** входит в последовательность, заданную итераторами;*

*б) **reverse (first, last)** − переставляет элементы в обратном порядке;*

*в) **iter_swap (first, last)** − меняет местами значения элементов, на которые указывают итераторы.*

Задание 3.8. Создадим графическое приложение, которое позволит пользователю выводить сумму элементов и произведение элементов на четных позициях.

1 Создайте приложение **Qt Widgets** с именем ***linkedlistgui***.

2 Разработайте интерфейс пользователя, для этого расположите на форме следующие элементы: **textEdit**, **pushbutton**, **checkBox (2)**.

3 Переименовать виджеты **checkBox** можно двумя способами: двойным кликом по виджету или в свойстве **text**. Дайте названия виджетам: вывод суммы элементов; вывод произведения элементов на четных позициях.

4 Установите запрет на изменение размеров окна.

Пример формы на рисунке 3.2 *(цвет фона, шрифт… установите по своему желанию)*.

Сохраните изменения и запустите для просмотра.

5 Перейдем к составлению программы. Добавим программный код для работы нашего приложения:

1) в заголовочном файле объявите экземпляр класса QLinkedList: **QLinkedList <int> Lili**;

2) добавьте код для заполнения вектора и вывода его в **textEdit** при запуске приложения:

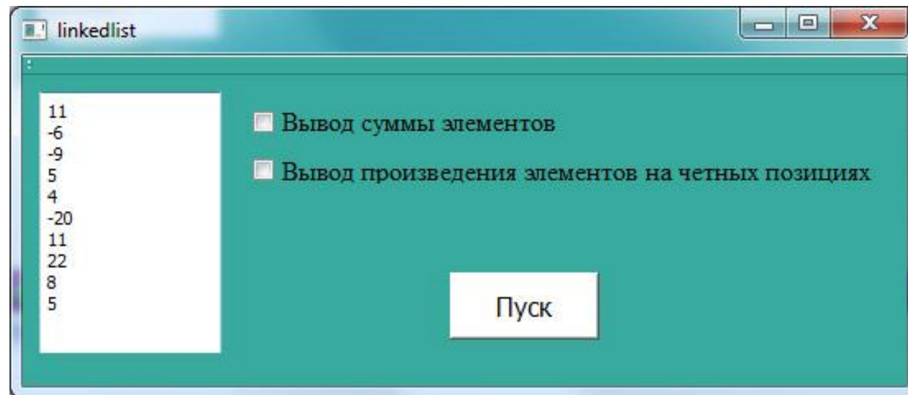


Рисунок 3.2 – Форма проекта ***linkedlistgui***

```
srand(time(0));
for (int i = 0; i < 10; i++) {
   Lili.append(rand() % 50 - 20);
   ui→textEdit→append(QString::number(Lili.last()) + " "); }
```

Функция ***last()*** возвращает ссылку на последний элемент в списке. Эта функция предполагает, что список не является пустым. Сохраните изменения и запустите для просмотра;

3) перейдите к слоту, который будет вызываться в ответ на событие ***clicked()***, и добавьте в него код для работы нашего приложения:

```
ui→textEdit→clear();
int sum = 0;
long mul = 1;
if (ui→checkBox→isChecked()) {
   QLinkedList<int>::iterator it = Lili.begin();
   while ( it != Lili.end()) {      // находим сумму элементов
      sum += *it;
      it++;      }
   ui→textEdit→append("Сумма: " + QString::number(sum)); }
//выводим сумму элементов в начале списка
if(ui→checkBox_2→isChecked()) {
   QLinkedList<int>::iterator it = Lili.begin();
   int pos = 0;
   while ( it != Lili.end()) {
      if (pos % 2 == 0) {         // произведение элементов
         mul *= *it;      }
      pos++;
      it++;   }
   ui→textEdit→append("Умножение: " + QString::number(mul));   }
QLinkedList<int>::iterator it = Lili.begin();
while ( it != Lili.end()) { //перезаписываем список
   ui→textEdit→append(QString::number(*it) + " ");
   it++;   } }
```

Сохраните изменения в приложении и просмотрите результат.

Задание 3.8. Самостоятельное решение задач по вариантам. Для работы со связным списком реализовать консольное приложение, со списком – использовать графический интерфейс. Перебор элементов осуществлять с помощью итераторов.

Вариант 3.1

1 Заполнить список случайными элементами. Реализовать добавление элемента в конец списка и удаления с конца (использовать RadioButton для выбора действия).
2 Создать два связных списка. Скопировать элементы первого во второй.

Вариант 3.2

1 Заполнить список случайными элементами. Реализовать добавление элемента в конец списка и удаления с начала (использовать RadioButton для выбора действия).
2 Создать два связных списка. Реализовать замену одного связного списка на другой.

Вариант 3.3

1 Заполнить 2 списка случайными элементами. Реализовать добавление введенного элемента в 1 список или второй, или в оба (использовать CheckBox).
2 Заполнить связный список случайными элементами и отсортировать их по возрастанию.

Вариант 3.4

1 Заполнить список случайными элементами и реализовать удаление элементов с позиций с N по K.
2 Заполнить связный список случайными элементами и отсортировать их по убыванию.

Вариант 3.5

1 Заполнить 2 списка случайными элементами и заменить все положительные элементы первого списка на значение минимального из второго списка.
2 Заполнить связный список случайными элементами. Удалить из списка все элементы, длина которых больше k.

Лабораторная работа 4. Работа с контейнерами в среде Qt Creator: QStack, QQueue

Стек реализует структуру данных, работающую по принципу Last In First Out: «последним пришел, первым ушел». То есть из стека первым удаляется элемент, который был вставлен позже всех остальных.

Класс ***QStack*** представляет собой реализацию стековой структуры. Этот класс унаследован от ***QVector***. Процесс помещения элементов в стек обычно называется **проталкиванием** (**pushing**), а извлечение из него верхнего элемента — **выталкиванием** (**poping**). Каждая операция проталкивания увеличивает размер стека на 1, а каждая операция выталкивания – уменьшает на 1. Для этих операций в классе ***QStack*** определены функции ***push()*** и ***pop()***. Метод ***top()*** возвращает ссылку на верхний элемент.

Прежде чем брать элемент необходимо убедиться, что стек не пустой (***!stack.isEmpty()***).

Задание 4.1. Создайте консольное приложение ***stack***. *Приложение должно увеличивать элементы с четными значениями на 3.*

```
QStack<int> stack_1, stack_2;  // объявление экземпляров класса QStack
qDebug() << "Задайте количество элементов: ";
QTextStream in(stdin);
int count = (in.readLine()).toInt(); //количество элементов стека
qDebug() << "Введите элементы:";     // ввод элементов с клавиатуры
```

```
for (int i = 0; i < count; i++) {
        int num = (in.readLine()).toInt();
        stack_1.push(num); }     // заносим введенные значения в стек
    qDebug() << "Стек 1 : " << stack_1;
    while (!stack_1.empty()) {
        int num = stack_1.pop();            // достаем элемент
        if (num % 2 == 0) { num += 3; } // проверяем на четность
        stack_2.push(num); }                // помещаем во второй стек
    qDebug() << "Стек 2 : " << stack_2;
    while (!stack_2.empty()) {
        stack_1.push(stack_2.pop());    } // перекладываем в первый стек
    qDebug() << "Стек 1 : " << stack_1;
```

Задание 4.2. Создадим графическое приложение, где продемонстрируем работу стека при решении головоломки «Ханойская башня».

1 Создайте Приложение **Qt Widgets** с именем ***tower***.

2 Разработайте интерфейс пользователя, для этого расположите на форме следующие элементы: **textEdit (3)**, **spinbox (2)**, **pushbutton**, **label (2)**.

Расположение виджетов представлено на рисунке 4.1.

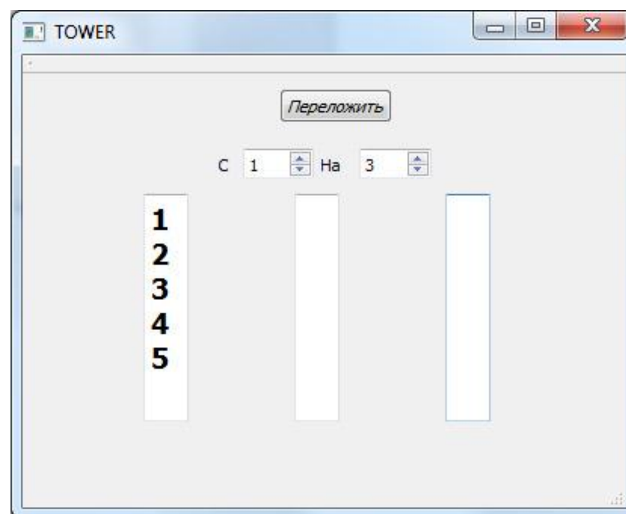


Рисунок 4.1 – Форма проекта ***tower***

3 В свойствах виджетов **spinBox** и **spinBox_2** задайте минимальное значение («1») и максимальное («3»). В виджете **spinBox** установите текущее значение (***value***) равным 1, в виджете **spinBox_2** равным 3.

4 Выделите **label, label_2, spinBox и spinBox_2** и скомпануйте по горизонтали с разделителем.

5 Установите запрет на изменение размеров окна.

6 Виджеты **textEdit** сделайте доступными только для чтения.

7 Сохраните изменения и запустите для просмотра.

8 Перейдем к составлению программы. Добавим программный код для работы нашего приложения:

1) объявите три объекта класса **QStack**;

2) добавьте код для заполнения первого стека и вывода его в **textEdit** при запуске приложения;

3) перейдите к слоту, который будет вызываться в ответ на событие ***clicked()***, и добавьте программный код для перемещения элементов стека:

```
//если значения компонентов spinBox и spinBox_2 совпадают,
//то выводим сообщение об ошибке
if (ui→spinBox→value() == ui→spinBox_2→value()) {
   QMessageBox::warning(this,"Предупреждение","Нельзя");}
else {
   QStack<int> *from_st;  //создаем указатели на 2 стека целых чисел
   QStack<int> *to_st;
   switch (ui→spinBox→value()) {
      case 1:{from_st = &stack_1; break;}          //в зависимости от выбранных
      case 2:{from_st = &stack_2; break;}          //значений спинбоксов,
      case 3:{from_st = &stack_3; break;}}         //присваиваем каждому из них
   switch (ui→spinBox_2→value()) {                 //ссылку на один из трех стеков
      case 1:{to_st = &stack_1; break;}
      case 2:{to_st = &stack_2; break;}
      case 3:{to_st = &stack_3; break;}}
//проверяем, что стек, из которого мы хотим перенести, не пустой
   if (from_st→isEmpty()) {
         QMessageBox::warning(this,"Предупреждение","Нельзя");}
   else
//проверяем, что стек, в который мы хотим записать элемент, пустой
   if ((!(to_st→isEmpty())) ||
// или его верхнее значение больше, чем то, которое мы хотим поместить
         (from_st→at(from_st→count()-1) >
         to_st→at(to_st→count()-1))) {
          QMessageBox::warning(this,"Предупреждение","Нельзя");}
   else {
      int val;
      val = from_st→pop(); /*значение извлекается из стека,
                              выбранного первым спинбоксом*/
      to_st→push(val); }} //и помещается в стек, выбранный вторым спинбоксом
ui→textEdit→clear();
ui→textEdit_2→clear();
ui→textEdit_3→clear();
for (int i = stack_1.count()-1; i >= 0; i--) {
     ui→textEdit→append(QString::number(stack_1.at(i))); }
for (int i = stack_2.count()-1; i >= 0; i--) {
     ui→textEdit_2→append(QString::number(stack_2.at(i)));}
for (int i = stack_3.count()-1; i >= 0; i--) {
     ui→textEdit_3→append(QString::number(stack_3.at(i)));}}
```

Сохраните изменения в приложении и просмотрите результат.

Очередь реализует структуру данных, работающую по принципу: «первым пришел, первым ушел». Для очереди определены операции добавления в очередь (**enqueue**) и извлечения элемента из очереди (**dequeue**). Реализована очередь в классе ***QQueue***, который унаследован от класса ***QList***.

*Задание 4.3. Создайте консольное приложение **queue**. Приложение должно увеличивать элементы с четными значениями на 3.*

```
QQueue<int> que;
```

```
QTextStream in(stdin);
qDebug() << "Количество элементов в очереди: ";
int count = (in.readLine()).toInt();
qDebug() << "Values: ";
for (int i = 0; i < count; i++) {
   int num = (in.readLine()).toInt();
   que.enqueue(num); }   // помещаем введенные значения в очередь
qDebug() << "Элементы очереди: " << que;
int len = que.length();    // заносим в переменную значение длины очереди
for (int i = 0; i < len; i++) {
   int num = que.dequeue();// извлекаем элемент из очереди
   if ( num % 2 == 0) {num += 3;}// проверка на четность
   que.enqueue(num); } // записываем обратно в очередь
qDebug() << "Очередь: " << que;
```

Сохраните изменения и запустите проект на выполнение.

Задание 4.4. Создадим графическое приложение (рисунок 4.2), которое продемонстрирует процесс извлечения и добавления элемента в очередь.

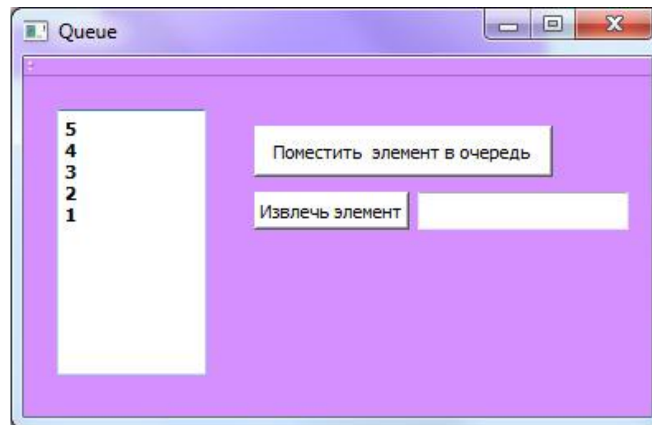


Рисунок 4.2 – Форма проекта ***queuegui***

Установите запрет на изменение размеров окна. Виджеты **lineEdit**, **textEdit** сделайте доступными только для чтения. Виджет **lineEdit** будет выводить текущий извлеченный элемент.

В файле ***queuegui.cpp*** в конструкторе главного окна добавьте код для заполнения очереди и вывода её в **textEdit** при запуске приложения:

```
for(int i = 1; i <= 5; i++) {
  que.enqueue(i);
  ui→textEdit→setText(QString::number(i) +"\n"+ ui→textEdit→toPlainText());}
n=5;
```

Функция ***toPlainText()*** возвращает текст, который находится в виджете.

Перейдите к слоту, который будет вызываться в ответ на событие ***clicked()*** и отвечать за добавление элемента в очередь:

```
n++;
que.enqueue(n);
ui→textEdit→clear();
for(int i = 0; i < que.count(); i++) {
  ui→textEdit→setText(QString::number(que[i]) +
                              "\n"+ ui→textEdit→toPlainText());    }
```

Перейдите к слоту, который будет вызываться в ответ на событие ***clicked()*** и отвечать за извлечение элемента из очереди:

```
ui→lineEdit_2→setText(QString::number(que.dequeue()));
ui→textEdit→clear();
for(int i = 0; i < que.count(); i++) {
   ui→textEdit→setText(QString::number(que[i]) +
                                   "\n"+ ui→textEdit→toPlainText());    }
```

Сохраните изменения в приложении и просмотрите результат.

Задание 4.5. Самостоятельное решение задач по вариантам. Реализовать приложения с графическим интерфейсом.

Вариант 4.1

1 Заполнить стек 10 случайными числами из интервала [-10; 20]. Просмотреть содержимое стека. Найти сумму положительных чисел, хранящихся в стеке.

2 Сформировать очередь из 8 чисел. Записать о очередь модуль разности между двумя соседними элементами очереди.

Вариант 4.2

1 Сформировать стек из 10 случайных целых чисел. Заменить в стеке все положительные значения на 1, а отрицательные - на -1.

2 Сформировать очередь из 10 чисел. Увеличить все значения в очереди на ее максимальный элемент. Результат поместить в очередь.

Вариант 4.3

1 Заполнить стек 10 случайными числами из интервала [-10; 80]. Заменить все значения остатками от деления на номер элемента в стеке.

2 Сформировать очередь из 8 чисел. Заменить значение первого элемента очереди суммой первого и последнего, значение второго элемента очереди – суммой второго и предпоследнего и т.д.

Вариант 4.4

1 Сформировать стек из 8 чисел. Заменить значение первого элемента стека произведением первого и последнего, значение второго элемента стека – произведением второго и предпоследнего и т.д.

2 Заполнить очередь 8 случайными числами из интервала [-20; 50]. Найти среднее арифметическое значений двух соседних элементов очереди. Результат поместить в очередь.

Вариант 4.5

1 Сформировать стек из 5 чисел. Поменять местами максимальный и минимальный элементы стека.

2 Заполнить очередь 8 случайными числами из интервала [0; 50]. Заменить все четные числа их средним арифметическим значением.

Лабораторная работа 5. Работа с контейнерами в среде Qt Creator: QMap, QSet

QMap<K,T> – ассоциативный массив, отображающий ключи типа **K** и значения типа **T**. Достоинство словаря в том, что он позволяет быстро получать значение, ассоциированное с заданным ключом. Необходимо следить за тем, чтобы не было занесено двух разных элементов с одинаковым ключом, ведь тогда не удастся извлечь один из этих элементов. Ключи должны быть уникальными!

Элементы сортируются по ключу, и проход по **QMap** всегда дает содержимое в отсортированном порядке.

Некоторые методы контейнера **QMap** представлены в таблице 5.1.

Таблица 5.1 – Методы контейнера ***QMap***

Метод	Описание
iowerBound()	Возвращает итератор на первый элемент с заданным ключом. Если такого ключа нет, то возвращается итератор, указывающий на ближайший элемент с большим ключом.
toStdMap()	Возвращает словарь STL с элементами нашего словаря.
upperBound()	Возвращает итератор, указывающий на последний элемент с ключом. Если такого ключа нет, то возвращается итератор, указывающий на ближайший элемент с большим ключом.

Ключ и значение можно получать через методы итератора: ***key()*** и ***value()***.

Добавлять элементы в словарь можно следующими способами:

Пример 5.1

```
QMap<QString, int> map;
map["ten"] = 10;
map["twenty"] = 20;
map["thirty"] = 30;
QMap<QString, int>::iterator it = map.begin();
for (;it != map.end(); ++it) {
   qDebug() << "key: " << it.key() << " value: " << it.value();}
```

Пример 5.2

```
QMap<QString, int> map2;
map2.insert("one",1);
map2.insert("two",2);
map2.insert("three",3);
QMap<QString, int>::iterator id = map2.begin();
for (;id != map2.end(); ++id) {
   qDebug() << "key: " << id.key() << " value: " << id.value();}
```

*Задание 5.1. Создайте консольное приложение **map**, подключите необходимые классы и заполните словарь двумя способами.*

*Задание 5.2. Создайте консольное приложение «Телефонный справочник», где будем хранить словарь из имен и прикрепленных к ним номеров, а также продемонстрируем варианты поиска записей по ключу или значению, и предусмотрим проверку на наличие записи в словаре с помощью оператора **[]** и функции **contains()**.*

Создайте консольное приложение ***phones***.

Для работы приложения добавьте следующие строки:

```
QMap<Qstring,Qstring> phones;          // объявляем экземпляр класса Qmap
phones["Anna"] = "79195679845";        // заполнение словаря
phones["Irina"] = "79128349028";
phones.insert("Alex","89632675687");
phones.insert("Danil", "89092345681");
Qmap<Qstring, Qstring>::iterator it = phones.begin();
for (;it != phones.end(); ++it)  {    //вывод словаря
   qDebug() << "Name: " << it.key() << " Phone: " << it.value(); }
qDebug() << endl;
```

```
qDebug() << “Phone Alex:  “ << phones.value(“Alex”);    //получаем значение
qDebug() << “Phone 89092345681:  “ << phones.key(“89092345681”);
if(phones.contains(“Oleg”)) {    // проверяем наличие записи в словаре
      qDebug() << “Phone:” << phones[“Oleg”];  }
if(phones.contains(“Irina”))    {
      qDebug() << “Phone:” << phones[“Irina”];    }
```

Примечание: обратите внимание на использование оператора **[]**, который может использоваться как для вставки, так и для получения значения элемента. Задание ключа, для которого элемент не существует, приведет к тому, что элемент будет создан. Чтобы избежать этого, нужно проверять существование элемента, привязанного к ключу. Подобную проверку мы осуществили при помощи метода ***contains()***.

Сохраните изменения и запустите приложение на выполнение.

*Задание 5.3. Самостоятельно разработайте графическое приложение **mapPhone.** Заранее заполните словарь 8 абонентами, при запуске приложения их список должен быть отображен в виджете **QTextEdit**. Добавьте 2 виджета **QRadioButton**, которые будут осуществлять переключение между поиском по ключу и поиском по значению. Также следует добавить 2 текстовых поля: для поиска и для вывода результата.*

Также предусмотрите удаление и добавление записей в справочник, если введенный ключ уже существует при добавлении записи, выведите информационное сообщение.

Контейнер **QSet** записывает элементы в некотором порядке и предоставляет возможность очень быстрого просмотра значений и выполнения с ними операций, характерных для множеств, таких как объединение, пересечение и разность. Необходимым условием является то, что ключи должны быть разными. Контейнер **QSet** можно использовать в качестве неупорядоченного списка для быстрого поиска данных.

Основные методы контейнера **QSet** представлены в таблице 5.2.

Таблица 5.2 – Методы контейнера **QSet**

Метод	Описание
unite()	Объединяет элементы множеств, т.е. элементы другого множества, которых нет в данном наборе элементов, вставляются в данное множество.
intersect()	Находит пересечение множеств. Удаляет те элементы из данного множества, которые не содержатся в другом множестве.
subtract()	Находит разность множеств. Удаляет те элементы данного множества, которые находится в другом множестве.

*Задание 5.4. Создайте консольное приложение, где введем два множества: танцоры и певцы, и продемонстрируем работу методов контейнера **QSet**.*

Для работы приложения добавьте следующие строки:

```
setlocale(LC_ALL,"");
QSet <QString> singers;
QSet <QString> dancers;
singers << "Иван" << "Евгений" <<
            "Анастасия" << "Александр" << "Ксения" ;
```

```
qDebug() << singers << endl;
dancers << "Марина" << "Петр" << "Иван" << "Ксения" ;
qDebug() << dancers << endl;
QSet<QString> result = singers;
result.unite(dancers);
qDebug() << "Объединение множеств:  ";
qDebug() << result << endl;
result = singers;
result.intersect(dancers);
qDebug() << "Пересечение множеств:  ";
qDebug() << result << endl;
result = singers;
result.subtract(dancers);
qDebug() << "Разность множеств singers - dancers:  ";
qDebug() << result << endl;
result = dancers;
result.subtract(singers);
qDebug() << "Разность множеств dancers - singers:  ";
qDebug() << result << endl;
```

Задание 5.5. Самостоятельно создайте графическое приложение ***setgui.*** *Добавьте на форму кнопку, 2 виджета* ***QSpinbox*** *и 3 виджета* ***QTextEdit****. Пользователь с помощью* ***QSpinbox*** *задает диапазон значений(от 1 до 100), при нажатии на кнопку 3 виджета* ***QTextEdit*** *должны быть заполнены:*

а) ***QTextEdit_1*** *множеством чисел кратных 3;*

б) ***QTextEdit_2*** *множеством простых чисел;*

в) ***QTextEdit_3*** *множеством составных чисел.*

Задание 5.6. Создайте две пустых телефонных книги для хранения записей. Заполните телефонные книги фамилиями абонентов и телефонными номерами. Выведите в ***textEdit*** *содержимое телефонных книг. Выполните обмен словарей и выведите их на экран. Очистите словари. Дополните телефонную книгу новыми записями и измените один из номеров телефонной книги. Снова выведите в* ***textEdit*** *содержимое телефонной книги.*

Задание 5.7. Самостоятельное решение задач по вариантам. Реализовать приложения с графическим интерфейсом.

Вариант 5.1

Имеется список класса (все имена различны). Определить, есть ли в классе человек, который побывал в гостях у всех. (Для каждого ученика составить множество побывавших у него в гостях друзей, сам ученик в это множество не входит.)

Вариант 5.2

Задан некоторый набор товаров. Определить для каждого товара, какие из них имеются в каждом из **n** магазинов, какие товары есть хотя бы в одном магазине, каких товаров нет ни в одном магазине.

Вариант 5.3

Заданы имена девочек. Определить, какие из этих имен встречаются во всех классах данной параллели, какие есть только в некоторых классах, какие из этих имен не встречаются ни в одном классе.

Вариант 5.4

Известны марки машин, изготовляемых в данной стране и импортируемых за рубеж. Даны некоторые N стран. Определить для каждой из марок, какие из них были доставлены во все страны, доставлены в некоторые из стран, не доставлены ни в одну страну.

Вариант 5.5

В озере водится несколько видов рыб. Три рыбака поймали рыб, представляющих некоторые из имеющихся видов. Определить, какие виды рыб есть у каждого рыбака, какие рыбы есть в озере, но нет ни у одного из рыбаков.

Лабораторная работа 6. Работа с файлами в среде Qt Creator

Класс **QFile** унаследован от класса **QIODevice**. В нем содержатся методы для работы с файлами (открытие, закрытие, чтение и запись данных). Создать объект можно, передав в конструкторе строку, содержащую имя файла. Можно ничего не передавать в конструкторе, а сделать это после создания объекта вызовом метода ***setName()***. Например: **QFile myFile ("D:\\file\\1.txt");** или

QFile myFile;

myFile.setName("1.txt ");

В процессе работы с файлами иногда требуется узнать, открыт файл или нет. Для этого вызывается метод ***QIODevice::isOpen()***, который вернет значение ***true***, если файл открыт, иначе - ***false***.

Чтобы закрыть файл, нужно вызвать метод ***close()***. С закрытием осуществляется запись всех данных буфера. Если требуется выполнить запись данных буфера в файл без его закрытия, то вызывается метод ***QFile::flush()***.

Проверить, существует ли нужный вам файл, можно статическим методом ***QFile::exists()***. Этот метод принимает строку, содержащую полный или относительный путь к файлу. Если файл найден, то метод возвратит значение ***true***, в противном случае - ***false***.

Методы ***QIODevice::read()*** и ***QIODevice::write()*** позволяют считывать и записывать файлы блоками. Если требуется считать или записать данные за один раз, то используют методы ***QIODevice::writeAll()*** и ***QIODevice::readAll()***.

Для удаления файла класс ***QFile*** содержит статический метод ***remove()***. В этот метод необходимо передать строку, содержащую полный или относительный путь удаляемого файла.

Задание 6.1. Создадим графическое приложение, в котором продемонстрируем основные методы при работе с файлами.

1 Создайте приложение **Qt Widgets** с именем ***file***.

2 Разработайте интерфейс пользователя, как показано на рисунке 6.1. Виджеты скомпонованы по горизонтали с разделителем. Установите запрет на изменение размеров окна. Сохраните изменения и запустите для просмотра.

3 Перейдите к слоту, который будет вызываться в ответ на событие ***clicked()*** и отвечать за чтение из файла, добавьте в него код:

```
QFile myFile ("D:\\proba\\file\\1.txt");  //прописываем путь к нашему файлу
if (!myFile.exists()) { //файл не найден
   QMessageBox::warning(this,"Ошибка","Файл не найден");
   return;    }
```

if (!myFile.open(QIODevice::ReadOnly)) { //файл нельзя открыть для чтения
QMessageBox::warning(this,"Ошибка","Файл нельзя открыть для чтения");
return; }
QTextStream stream(&myFile); /*создаем объект класса *QTextStream* и передаем в конструктор ссылку на файл, из которого нужно производить чтение*/
QString buffer = stream.readAll(); //считываем в объект класса *QString*
ui→textEdit→setText(buffer); //содержимое файла и помещаем в *QTextEdit*
myFile.close();

Рисунок 6.1 – Форма проекта ***file***

4 Перейдите к слоту, который будет вызываться в ответ на событие ***clicked()*** и отвечать за запись в файл, добавьте в него код:
QFile myFile2 ("D:\\proba\\file\\2.txt"); //путь к нашему файлу
if (!myFile2.exists()) { //файл не найден
QMessageBox::warning(this,"Ошибка","Файл не найден");
return; }
if (!myFile2.open(QIODevice::WriteOnly)) { //файл нельзя открыть для записи
QMessageBox::warning(this,"Ошибка","Файл нельзя открыть для записи");
return; }
QTextStream stream(&myFile2);
/*Преобразуем строку из **textedit** в массив при помощи метода *split*, используя в качестве разделителя между элементами массива пробел. Константа *SkipEmptyParts* отвечает за пропуск лишних пробелов*/
QStringList numbers =
ui→textEdit→toPlainText().split(" ",QString::SkipEmptyParts);
int val;
for (int i = 0; i < numbers.length(); i++) {
val = numbers[i].toInt()*2; /*преобразуем каждый элемент массива к целому числу и увеличиваем его вдвое*/
stream << QString::number(val) + " "; //помещаем во второй файл **}**
myFile2.close();

Сохраните изменения в приложении и просмотрите результат.

Задание 6.2. Самостоятельно создайте консольное приложение ***file_console***. Дан файл А, компоненты которого являются целыми числами. Найдите: 1) сумму компонент файла А и запишите её в файл В; 2) последний компонент файла А и

запишите его в файл C. Предусмотрите проверки на существование и на возможность чтения/записи.

Задание 6.3. Самостоятельное решение задач по вариантам. Реализовать приложения с графическим интерфейсом.

Вариант 6.1

Дан файл f, компоненты которого являются действительными числами. Найдите: а) наибольший компонент; б) наименьший компонент с четным номером; в) наибольший модуль компонента с нечётным номером; г) разность первого и последнего компонента файла.

Вариант 6.2

Дан файл f, компоненты которого являются целыми числами. Запишите в файл g наибольшее значение первых пяти компонентов файла f, затем - следующих пяти компонентов и т.д. Если в последней группе окажется менее пяти компонентов, то последний компонент файла g должен быть равен наибольшему из компонентов файла f, образующих последнюю (неполную) группу.

Вариант 6.3

Даны символьные файлы f1 и f2. Перепишите с сохранением порядка следования компоненты файла f1 в файл f2, а компоненты файла f2 – в файл f1. Используйте вспомогательный файл h.

Вариант 6.4

Дан файл f, компоненты которого являются целыми числами. Получите в файле g все компоненты файла f: а) являющиеся чётными числами; б) делящиеся на 3 и не делящиеся на 7; в) являющиеся точными квадратами.

Вариант 6.5

Дан файл f, компоненты которого являются целыми числами. Запишите в файл g все чётные числа файла f, а в файл h - все нечётные. Порядок следования чисел сохранить.

Лабораторная работа 7. Создание проекта «Калькулятор»

Цель проекта: разработать программу «Калькулятор» с расширенным функционалом.

1 Создайте графическое приложение **Qt Widgets**. Имя класса ***calculator.***

2 Разработка интерфейса, как показано на рисунке 7.1.

Задайте значения свойствам виджетов (таблица 7.1).

Таблица 7.1 – Значения свойств виджетов

Виджет	Свойство	Значение
lineEdit, *lineEdit_2*	maxlength alignment font→ размер enabled	10 alignRight 16 false

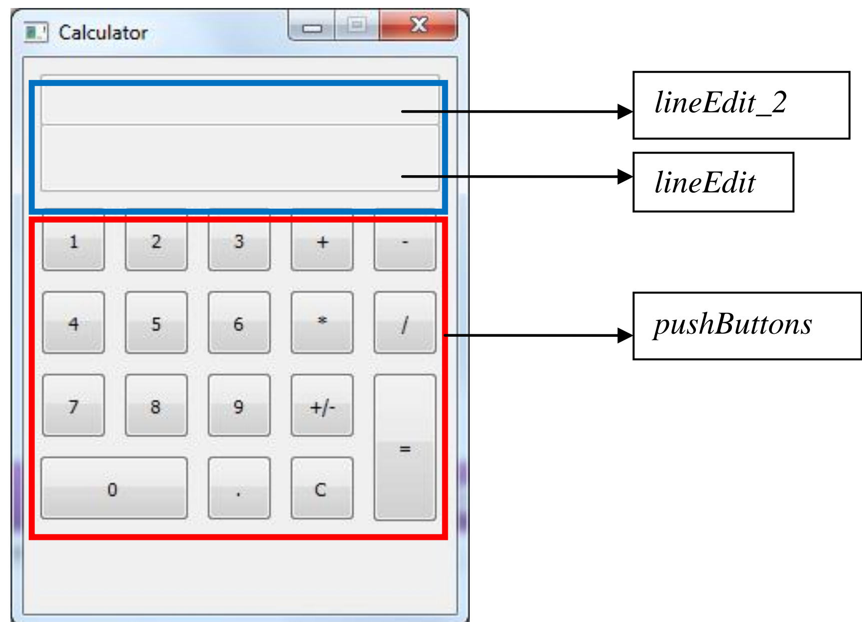


Рисунок 7.1 – Форма проекта ***calculator***

3 Добавим слоты для обработки нажатий на кнопки с цифрами, размещенные на форме **ui→lineEdit→insert("1");** Аналогично, добавьте слоты для остальных девяти кнопок.

4 В секции **private** заголовочного файла объявите экземпляр класса **QStack**, а также функцию для обработки арифметических действий

void calculate(QString sgn);

5 Добавим код в слот, который отвечает за сброс значений:

```
ui→lineEdit→clear();
ui→lineEdit_2→clear();
stack.clear();
```

5 Добавим код в слот, который отвечает за добавление разделителя целой и дробной части, т.е. знак «десятичная точка» :

```
if (ui→lineEdit→text().length() == 0 ||
    ui→lineEdit→text().at(ui→lineEdit→text().length()-1) == '-' ||
    ui→lineEdit→text().indexOf('.') != -1)      return;
ui→lineEdit→setText(ui→lineEdit→text()+'.');
```

Проверяем допустимость установки разделителя на текущей позиции. Точка не может стоять первым символом после знака «-» или если разделитель уже установлен.

6 Добавим код в слот, который отвечает за смену знака числа:

```
if (ui→lineEdit→text().length() == 0 || ui→lineEdit→text().at(0) != '-')
   ui→lineEdit→setText("-"+ui→lineEdit→text());
else
   ui→lineEdit→setText(ui→lineEdit→text().remove(0,1));
```

Если число ещё не введено или на нулевой позиции, в строке не введен знак «минус», то ставим знак минуса, иначе (если в строке уже присутствует знак «-») удаляем его.

7 Добавим код в функцию ***calculate()***:

if (ui→lineEdit→text().length() == 0) {return;} /*если ничего не введено, то выходим из функции*/

/*если в стеке не менее двух элементов, то извлекаем значения второго операнда и переводим в тип double, затем извлекаем знак и значение первого операнда*/

```
if (stack.length() >= 2) {
   double val2 = ui→lineEdit→text().toDouble();
   QString sign = stack.pop();
   double val1 = stack.pop().toDouble();
   if (sign == "+") {
         stack.push(QString::number(val1+val2));
   } else if (sign == "-") {
            stack.push(QString::number(val1-val2));
          } else if (sign == "*") {
                   stack.push(QString::number(val1*val2));
                } else if (sign == "/") {
                         if (val2 == 0) {
                            stack.push(QString::number(val1));
                         } else {
                             stack.push(QString::number(val1/val2));  }
                     }
                stack.push(sgn);
   } else {
        stack.push(ui→lineEdit→text());
        stack.push(sgn);
     }
ui→lineEdit→clear();
ui→lineEdit_2→setText(stack.toList().join(""));
```

В стек заносим результат соответствующей операции. Помещаем в стек только что нажатый знак. Если стек был пуст, то помещаем значение и знак.

Перезаписываем в верхний lineEdit новые значения первого операнда и знака. Преобразуем к списку и функцией join() собираем в строку или «склеиваем».

8 Добавим код в слот, который отвечает за равенство:

/*если есть первый операнд и знак и в строке lineedit не пусто, то помещаем из lineEdit в стек*/

```
if (ui→lineEdit→text().length() != 0 && stack.length() == 2) {
   stack.push(ui→lineEdit→text());}
if (stack.length() < 3) {return;}
```

//извлекаем из стека числа и знак операции

```
double val2 = stack.pop().toDouble();
QString sign = stack.pop();
double val1 = stack.pop().toDouble();
```

//помещаем в стек результат действия в зависимости от знака операции

```
if (sign == "+") {
   stack.push(QString::number(val1+val2));
} else if (sign == "-") {
         stack.push(QString::number(val1-val2));
      } else if (sign == "*") {
               stack.push(QString::number(val1*val2));
            } else if (sign == "/") {
```

```
                if (val2 == 0) {
                    stack.push(QString::number(val1));
                    stack.push(sign);
                    ui->lineEdit->clear();
                    return;}
                stack.push(QString::number(val1/val2));   }
    ui->lineEdit->setText(stack.pop());
    ui->lineEdit_2->setText(stack.toList().join(""));
}
```

9 Добавим программный код в слоты, которые отвечают за передачу аргумента (знака арифметическое операции) в функцию ***calculate()***: **calculate("+");** Аналогично добавьте код к остальным трем операциям (- , *, /).

Работа по группам. Организуйте работу по трем группам. Каждой группе необходимо будет модернизировать проект, добавив дополнительные функции.

Общее задание для всех групп: реализовать работу с памятью (MS, MR, MC, M+, M -).

Модернизировать проект следующими операциями:

1 группа: cos, lg, x^2, $\sqrt{x}$
2 группа: sin, ln, x^y, 1/x
3 группа: tg, 10^x, x^3, x!

Лабораторная работа 8. Создание проекта «Текстовый редактор»

Цель проекта: разработать программу «Текстовый редактор» со стандартным функционалом (копирование, открытие файла, работа со шрифтом).

1 Создайте графическое приложение **Qt Widgets**. Имя класса ***notepad.***

2 Разработать интерфейс приложения, как показано на рисунке 8.1.

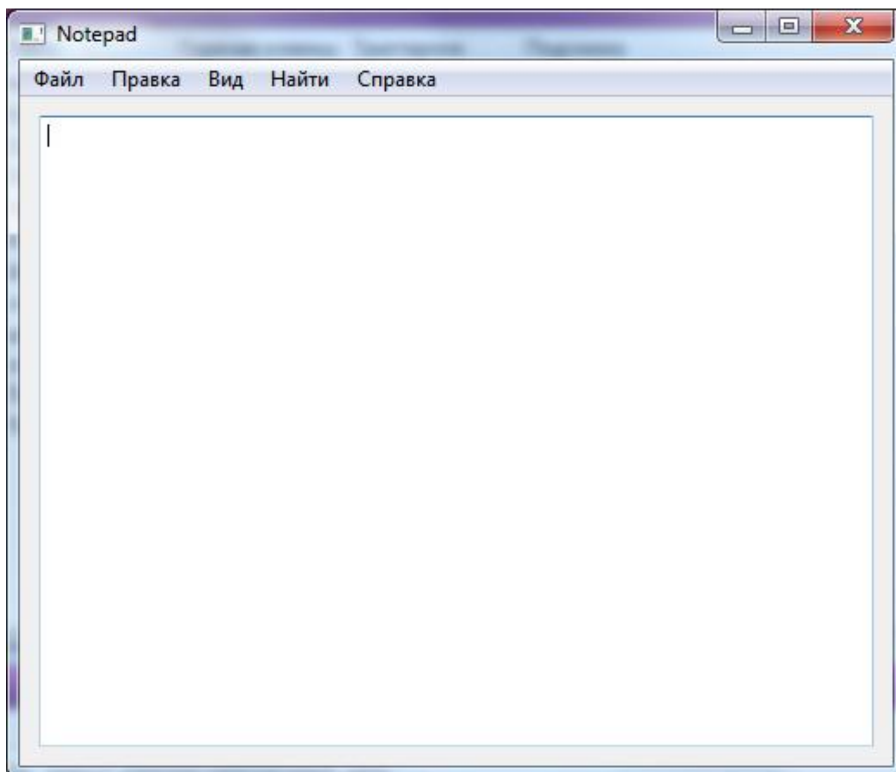


Рисунок 8.1 – Форма проекта ***notepad***

3 Разработать меню. Пункты меню:

1) Файл (Новый, Открыть, Сохранить как);

2) Правка (Отменить, Копировать, Вставить);

3) Вид (Шрифт, Выравнивание (Слева, По центру, Справа));

4) Найти (Поиск, Перейти к строке...);

5) Справка.

Виджет QTextEdit скомпонован «по горизонтали». Поставьте запрет на изменение размера окна.

4 Переход к слотам событий: **action_→ Перейти к слоту → triggered()**. Перейдем к слоту, который отвечает за открытие нового документа:

```
ui→textEdit→clear();
```

5 Перейдем к слоту, который отвечает за открытие ранее сохраненного документа:

```
QFileDialog dialog;
dialog.setFileMode(QFileDialog::AnyFile);
QString filename = dialog.getOpenFileName(NULL,"Open file","","");
QFile file(filename);
file.open(QFile::ReadOnly|QFile::Text);
QTextStream in(&file);
ui→textEdit→setText(in.readAll());
file.close();
```

Класс **QFileDialog** – стандартное диалоговое окно, которое позволяет пользователям выбирать файлы или каталоги.

FileMode используется для указания того, что пользователь может выбрать в данном окне, в нашем случае ***Имя файла***, если файл будет не найден, появится соответствующее информационное сообщение.

GetOpenFileName – удобная статистическая функция, которая возвращает существующий файл, выбранный пользователем. В параметрах указано, что заголовок окна «**Open File**», пустые кавычки обозначают, что пользователь может выбрать/найти файл в любом каталоге и с любым расширением.

В методах открытия файла перечислены режимы, в которых происходит открытие (для чтения и при окончании линии, переход на другую строку).

6 Перейдем к слоту, который отвечает за сохранение документа и выбора каталога для его сохранения:

```
QFileDialog dialog;
dialog.setFileMode(QFileDialog::AnyFile);
QString filename = dialog.getSaveFileName(NULL,"Save file","","");
QFile file(filename);
file.open(QFile::WriteOnly|QFile::Text);
QTextStream out(&file);
out << ui→textEdit→toHtml();
out.flush();
file.close();
```

getSaveFileName - это статическая функция, которая возвращает имя файла, выбранного пользователем. Файл не должен существовать.

Функция класса **QTextEdit - toHTML** возвращает текст как **html**, т.е. сохраняет все элементы форматирования.

7 Перейдем к слоту, который отвечает за отмену действия:

ui→textEdit→undo();

8 Перейдем к слотам, которые отвечают за копирование и вставку выделенного текста:

_copytext = ui→textEdit→textCursor().selectedText();

Объявим в секции **private** переменную строкового типа **_copytext**. Заносим в неё выделенный текст.

ui→textEdit→textCursor().insertHtml(_copytext);

Вставляем текст, сохраняя все элементы **HTML** форматирования с текущей позиции курсора.

9 Перейдем к слотам, которые отвечают за выравнивание текста:

слева **ui→textEdit→setAlignment(Qt::AlignLeft);**

по центру **ui→textEdit→setAlignment(Qt::AlignCenter);**

справа **ui→textEdit→setAlignment(Qt::AlignRight);**

10 Перейдем к слоту, который отвечает за изменение шрифта в тексте:

QFontDialog dialog;
QFont font = dialog.getFont(NULL);
QTextCharFormat format;
format.setFont(font);
ui→textEdit→textCursor().setCharFormat(format);

Класс **QFont** определяет шрифт, используемый для форматирования текста. Функция **getFont**() открывает модальный диалог выбора шрифта и возвращает шрифт.

QTextCharFormat класс предоставляет информацию о форматировании для символов. На формат символов текста в документе указывают визуальные свойства текста.

Функция **setFont**() устанавливает выбранный шрифт.

11 Перейдем к слоту, который отвечает за переход на нужную строку:

bool ok;
int num = QInputDialog::getInt(0, "Перейти", "На строку:", 1, 1,
ui→textEdit→document()→lineCount(),1,&ok);
QTextBlock block = ui→textEdit→document()→findBlockByLineNumber(num-1);
QTextCursor cursor(block);
ui→textEdit→setTextCursor(cursor);

QInputDialog класс, который предоставляет простой диалог для получения одного значения от пользователя.

Функция **getInt**() возвращает целое число, которое было введено пользователем. Параметры («Перейти» это текст, который отображается в строке заголовка диалогового окна. «На строку» – текст, который показывается пользователю (он должен сказать, что должно быть введено). 1 – является целым числом, по умолчанию с которого счётчик будет установлен. 1 – минимиальное значение, которое пользователь может выбрать. Функция **lineCount**() считает количество строк в документе и является максимальным значением, которое пользователь может выбрать. 1 – сумма, на которую значения изменяются при нажатии пользователем кнопки со стрелками для увеличения или уменьшения значения.

QTextBlock класс предоставляет контейнер для фрагментов текста.

Функция **findBlockByLineNumber**(**LineNumber**) возвращает текстовый блок, который содержит указанный **LineNumber**.

12 Перейдем к слоту, который отвечает за поиск слова в тексте:

```
bool bOk;
QString str = QInputDialog::getText(0,"Найти","Введите текст:",
                                    QLineEdit::Normal,"",&bOk);
if (!bOk) { return; }
QTextCursor oldCursor = ui→textEdit→textCursor();
ui→textEdit→setTextCursor(
        QTextCursor(ui→textEdit→document()→findBlockByLineNumber(0)));
bool finded = ui→textEdit→find(str);
if (!finded)   ui→textEdit→setTextCursor(oldCursor);
```

Самостоятельно доработайте проект, добавив в него следующие пункты главного меню: Файл (Сохранить, Печать, Выход); Правка (Повторить, Вырезать, Удалить, Вставить дату); Найти (Заменить); Справка (О програме).

Список литературы

1 Бланшет Ж., Саммерфилд М. Qt 4: Программирование GUI на C++. – Москва : Изд-во «КУДИЦ-ПРЕСС», 2007. - 629 с.

2 Саммерфилд М. Qt. Профессиональное программирование. Разработка кроссплатформенных приложений на C++. – Санкт-Петербург : Символ-Плюс, 2011. – 560 с.

3 Шлее М. Qt 4.8. Профессиональное программирование на C++. – Санкт-Петербург : БХВ-Петербург, 2012. – 912 с.

4 Сайт кафедры «Информационные технологии и методика преподавания информатики». URL: http://it.kgsu.ru/Qt/oglav.html (дата обращения 30.10.2016).

Содержание

Лабораторная работа 1. Знакомство со средой Qt Creator и методами создания приложений 3
Лабораторная работа 2. Работа с контейнерами в среде Qt Creator: QVector 8
Лабораторная работа 3. Работа с контейнерами в среде Qt Creator: QList, QLinkedList 15
Лабораторная работа 4. Работа с контейнерами в среде Qt Creator: QStack, QQueue 22
Лабораторная работа 5. Работа с контейнерами в среде Qt Creator: QMap, QSet 26
Лабораторная работа 6. Работа с файлами в среде Qt Creator....................30
Лабораторная работа 7. Создание проекта «Калькулятор»....................32
Лабораторная работа 8. Создание проекта «Текстовый редактор»....................35
Список литературы 38

Адаменко Юлия Владимировна

Криница Ксения Андреевна

РАЗРАБОТКА ГРАФИЧЕСКОГО ИНТЕРФЕЙСА С ПОМОЩЬЮ БИБЛИОТЕКИ QT

Методические рекомендации
для студентов направлений 44.03.01 «Педагогическое образование»,
09.03.03 «Прикладная информатика»

Редактор Г.В. Меньщикова

……………………………………………………………………………………………

Подписано в печать 02.11.17	Формат 60×84 1/16	Бумага 65 г/м^2
Печать цифровая	Усл. печ. л. 2,5	Уч.-изд. л. 2,5
Заказ № 188	Тираж 25	Не для продажи

……………………………………………………………………………………...

БИЦ Курганского государственного университета.
640020, г. Курган, ул.Советская, 63/4.
Курганский государственный университет.