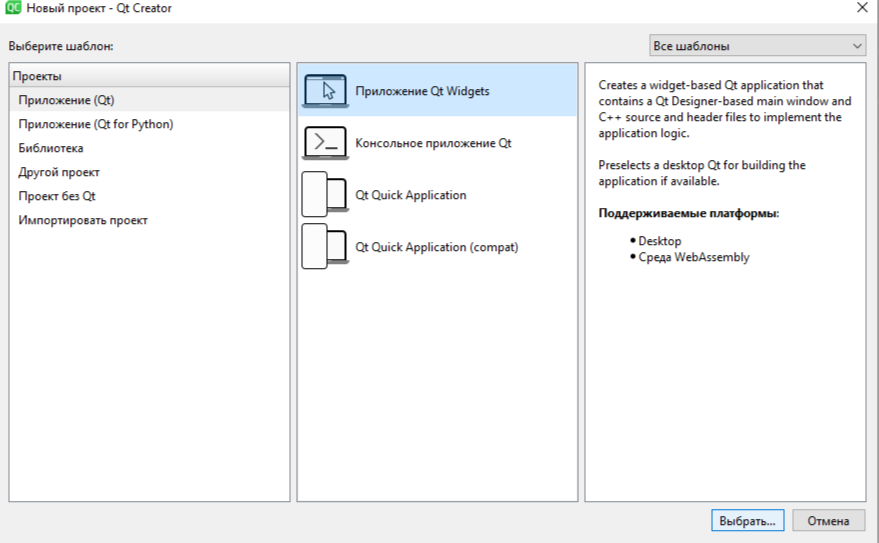
Лабораторная работа «Разработки калькулятора с GUI на С++»

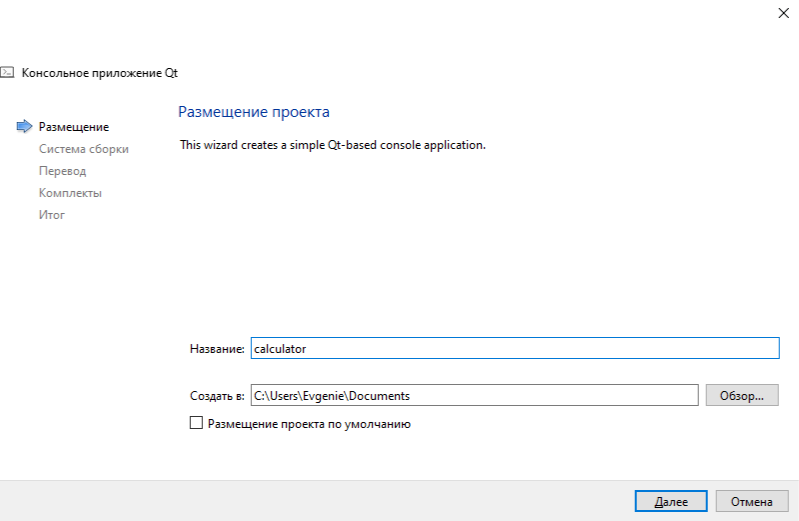
Оглавление   
**Создание проекта** ...................................................................................................................................... 2 **Проектирование интерфейса программы** ............................................................................................ 4 **Размерная политика элементов** ............................................................................................................ 14 **Стилизация калькулятора** .................................................................................................................... 15 **Стили для Line Edit и Label**................................................................................................................... 19 **Финальные штрихи** ................................................................................................................................ 20 **Обработка событий** ................................................................................................................................. 21 **Вычисление выражения** ........................................................................................................................ 24 **Требования к оформлению работы** ..................................................................................................... 33

1

**Создание проекта** 1.Откройте Qt Creator   
2.Нажмите «Создать проект» (Create Project)   
3.Приложение Qt -> Приложение Qt Windgets



4.Нажмите выбрать   
5.Назовите проект calculator

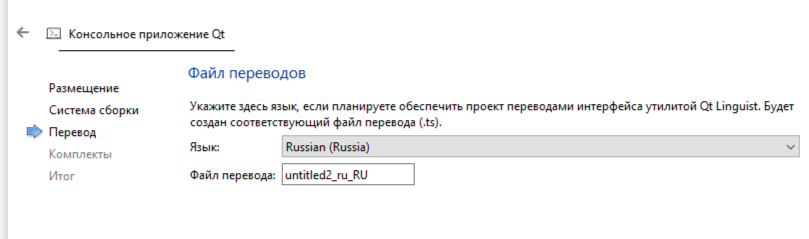


6.Вид сборки Cmake

2



7.Язык интерфейса – русский



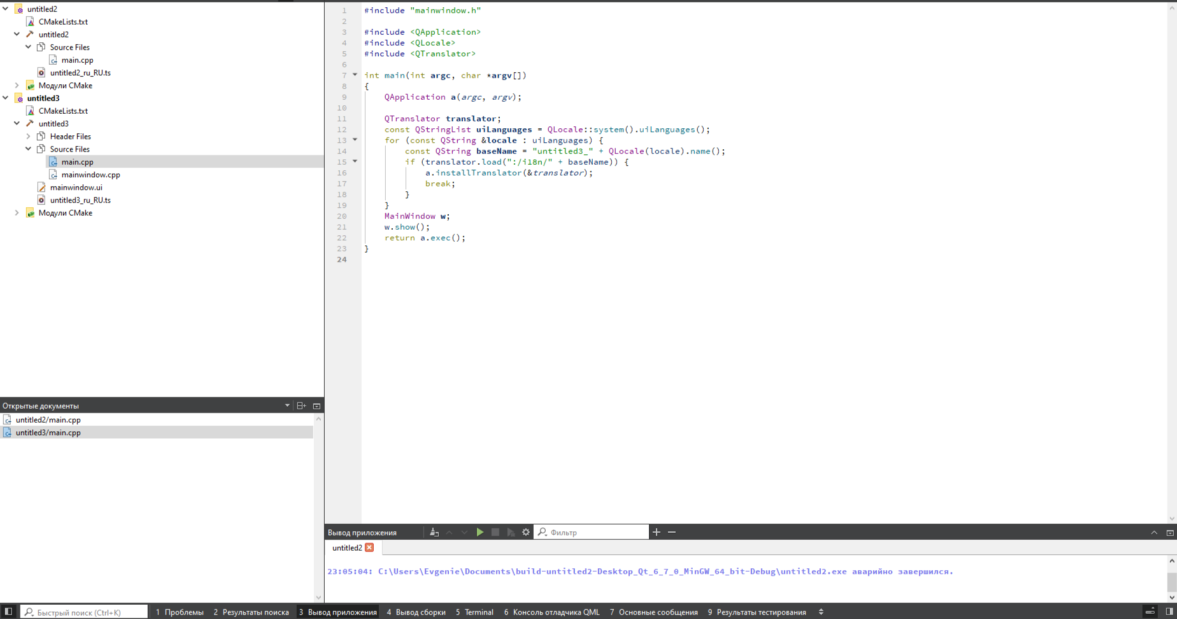
8.Комплект MinGW



9.Завершить (Enter)

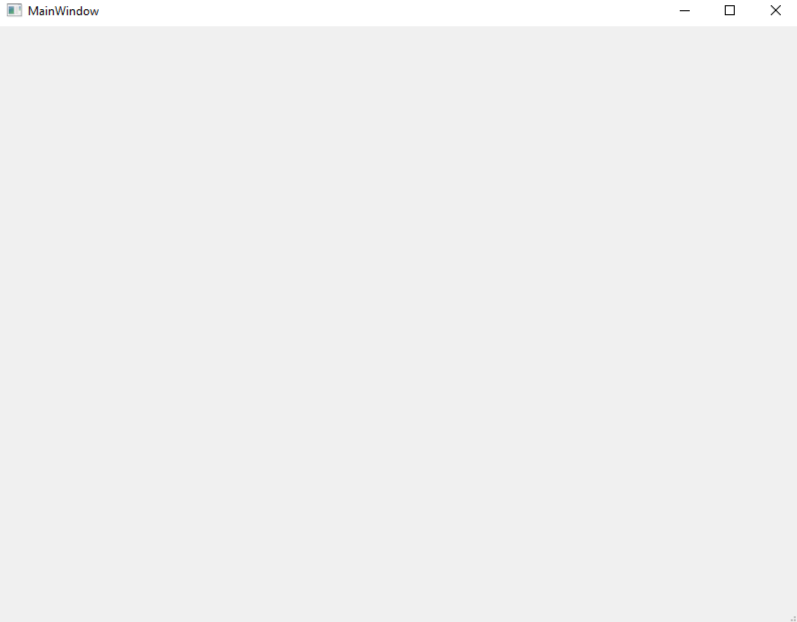
3

**Проектирование интерфейса программы**   
Если все действия в прошлом пункте выполнены верно, то вы увидите следующее окно:

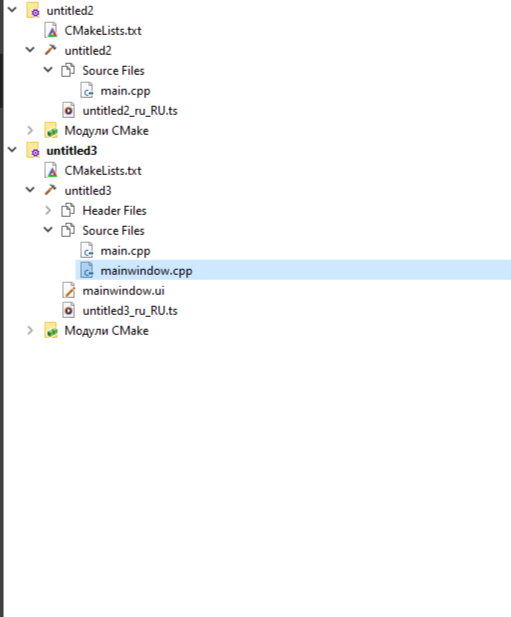


Не забудьте сохранить файл и запустить его (пиктограмма зеленый треугольник).

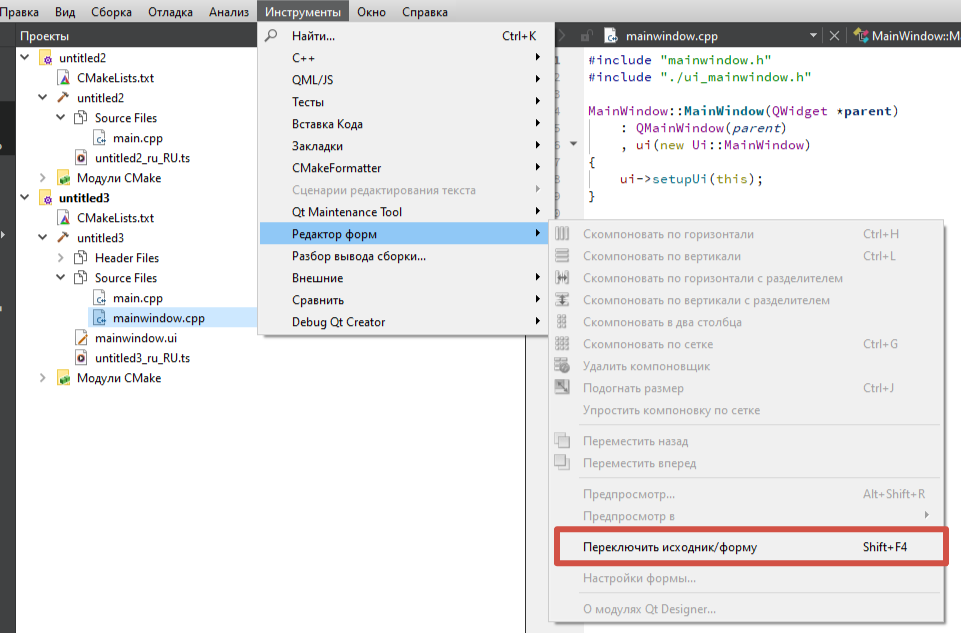
По окончанию сборки проекта вы увидите следующее окно:



1.В проводнике выберите (выделите) файл с главной формой (mainwindow.cpp)   
 4



2.Далее в меню выберите Инструменты -> Редактор форм -> Переключить исходник/форму (или нажмите Shift+F4)   
Обратите внимание, что mainwindow.cpp должен быть открыт!

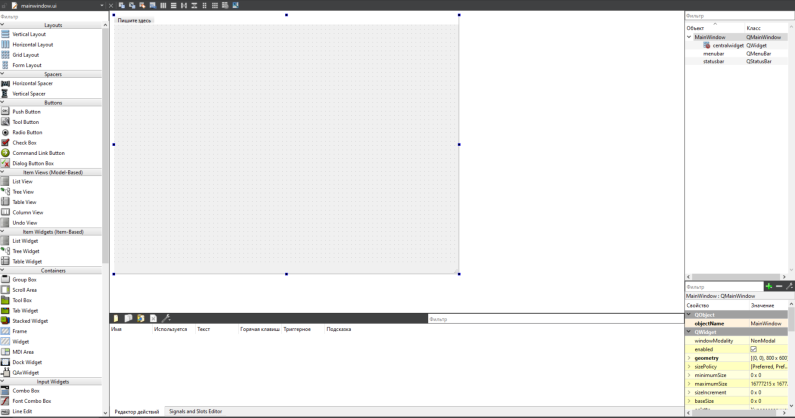


5

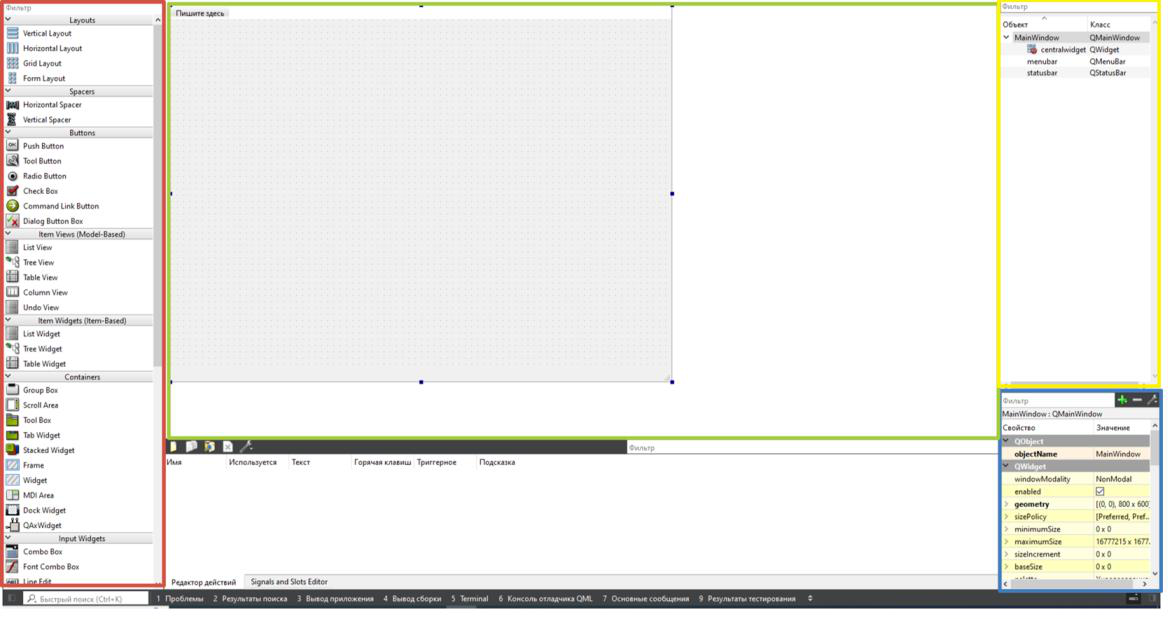
3.Вам откроется редактор форм.

Как мы отмечали ранее, есть два подхода, которые можно использовать при построении графического пользовательского интерфейса, используя виджеты Qt:

создать, настроить виджеты и разместить их на форме в соответствующих компоновках с помощью программного кода;   
воспользоваться визуальным редактором форм Qt Designer, который создаст файл формы (он будет описывать ее внешний вид, размещение, размеры, настройки, компонование и т.д.). В дальнейшем из файла формы на этапе компиляции будет создан файл с кодом программы, будет программно создавать этот интерфейс и предоставлять программисту доступ к элементам на форме.

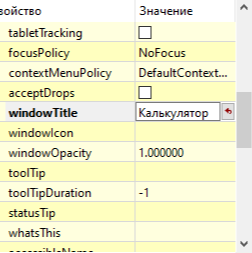


4.Рабочая область разбита на несколько частей:   
 a.Красная область – список доступных стандартных компонентов   
 b.Зеленая область – зона редактирования формы и её составных частей c.В синем окне отображаются свойства (характеристики) выбранного компонента, такие как цвет (color), длина (length), ширина (width) и проч. d.В желтом окне Инспектор объектов – список всех объектов на форме с их названиями и классами

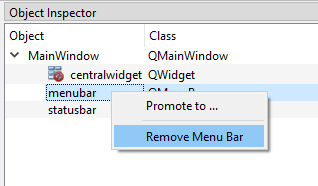


6

5.Выделите форму (Window) и замените WindowTitle на «Калькулятор»

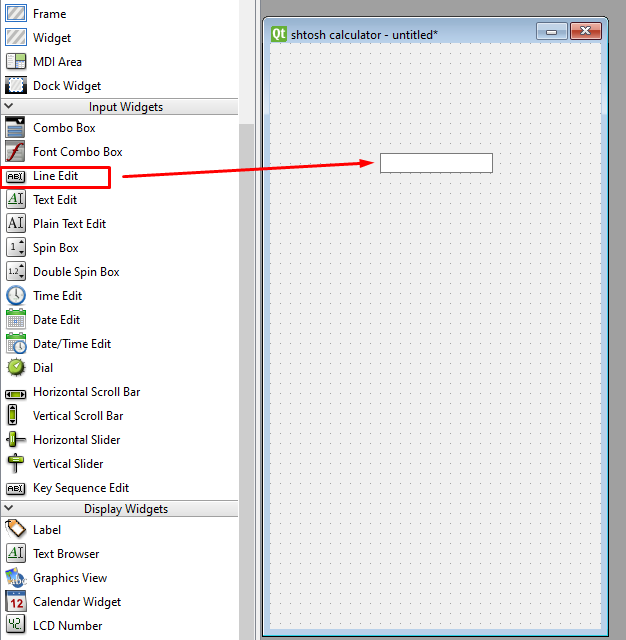


6.Убираем ненужные menubar и statusbar.

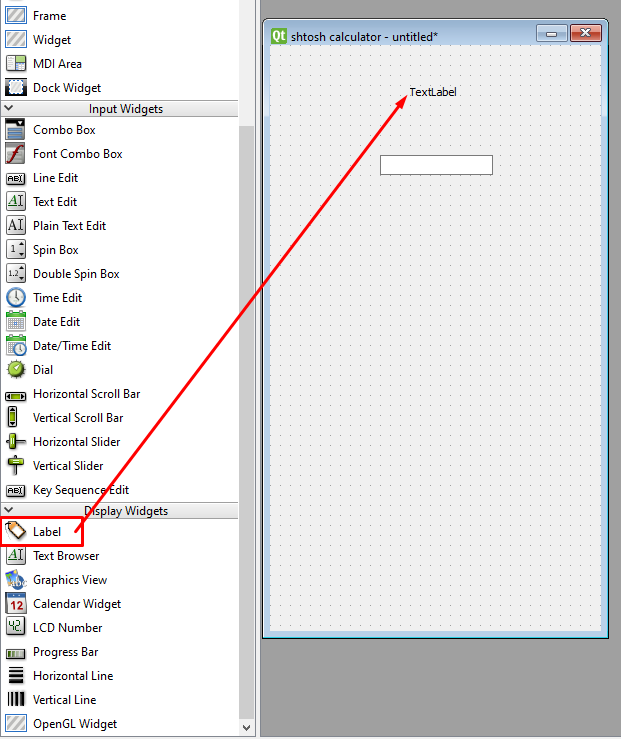


7

7.Перетащим нужные элементы в интерфейс. В нашем калькуляторе будет поле ввода Line Edit.

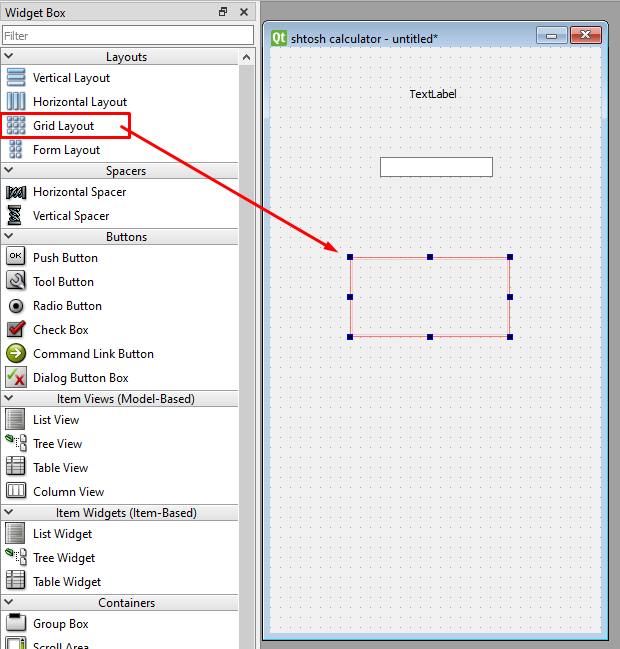


8.Label с временным выражением над этим полем ввода.

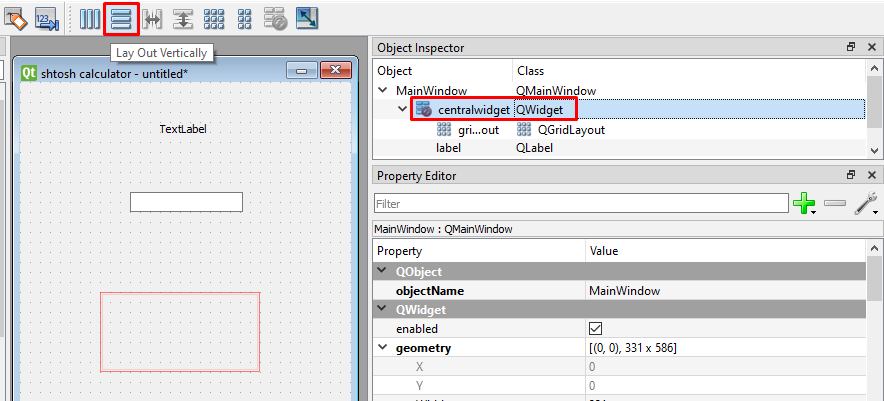


8

9.Grid Layout для кнопок.

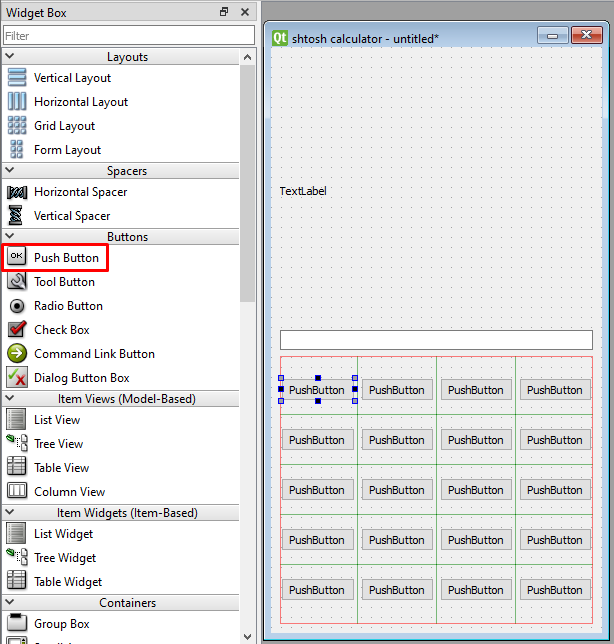


10.Просто закинем эти элементы и выберем Lay Out Vertically для центрального виджета.



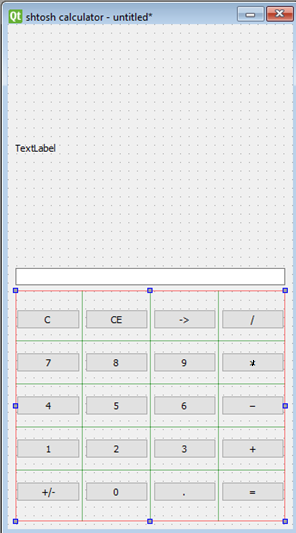
9

11.Теперь закинем кнопки в Grid Layout, у меня будет 4 колонки и 5 рядов. Чтобы скопировать и вставить элемент, можно перетащить его с зажатой клавишей Ctrl.



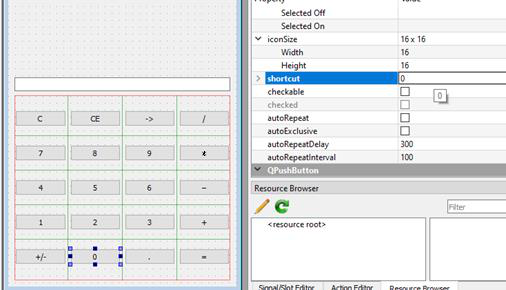
12.Поставим текст во все кнопки.

(ОБРАЩАЮ ВНИМАНИЕ, ЧТО ТЕКСТ КНОПОК ДОЛЖЕН УКАЗЫВАТЬСЯ БЕЗ ЛИШНИХ ПРОБЕЛОВ! УМНОЖЕНИЕ **ИСКЛЮЧИТЕЛЬНО СИМВОЛОМ\*** )

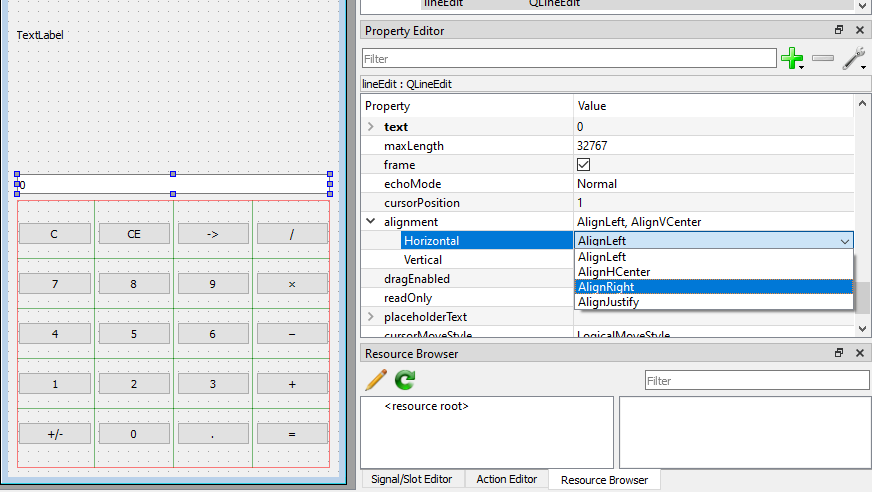


10

13.Проставим горячие клавиши для всех кнопок, кроме Clear и отрицания. За это отвечает свойство shortcut. К сожалению, в Qt Designer нельзя указать несколько горячих клавиш для одной кнопки. Хотелось бы, чтобы клавиши "Enter", "Return" и "=" выполняли вычисление. Мы сделаем это позже в коде. А пока поставим для   
вычисления одинокую клавишу =.

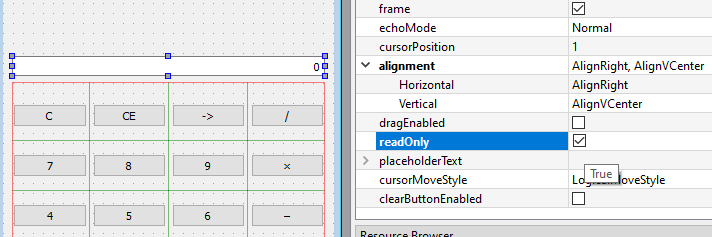


14.Запишем 0 в Line Edit и выберем правое горизонтальное выравнивание для текста.

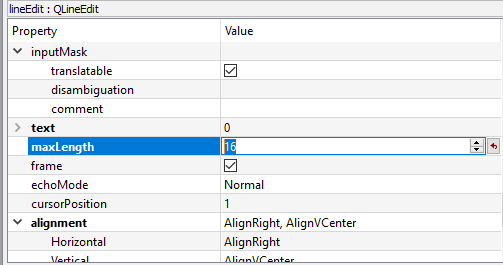


15.Нам нужно сделать так, чтобы пользователь не мог вводить что попало в это поле, чтобы он мог его только читать. Для этого существует свойство readOnly.

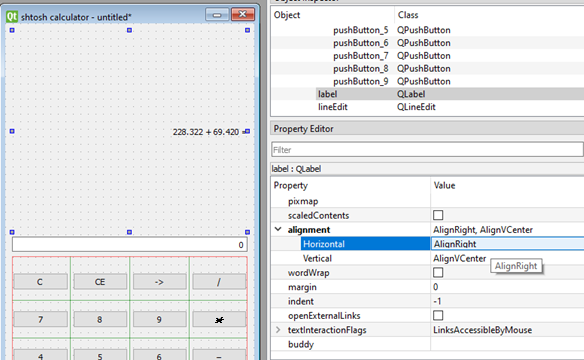
11



16.Укажем максимальную длину в 16 символов, как в калькуляторе Windows.

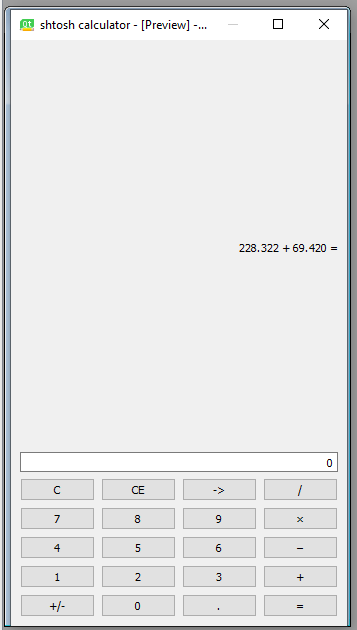


17.Запишем в лейбл какое-нибудь выражение и поставим правое выравнивание.



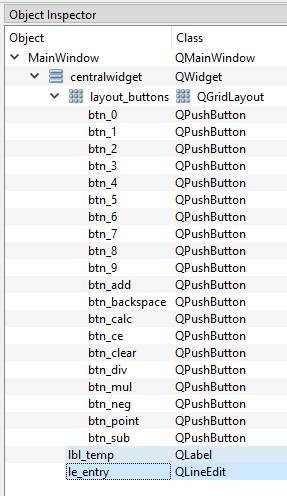
12

18.Чтобы посмотреть превью дизайна используйте сочетание клавиш Ctrl + R.



19.Назовем элементы, чтобы в коде было проще обращаться к ним.

13



**Размерная политика элементов**

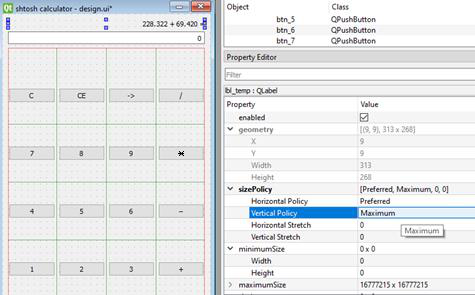
Вы спросите: "Почему интерфейс так плохо выглядит?". Все потому, что у элементов не

настроена вертикальная политика. Для лейбла и поля поставим **Maximum.**

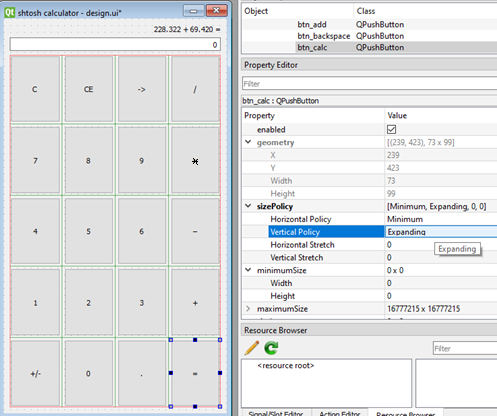
Конечно же не забываем сохранить файл интерфейса. Он имеет расширение ui. Обычно

называется файл design.ui

14



Для всех кнопок поставим Expanding.



**Стилизация калькулятора**   
Сначала нужно определиться с цветовой палитрой. Я буду использовать 4 цвета: 1.Почти черный #121212 для фона.

2.Белый #FFF для текста кнопок и поля ввода.

15

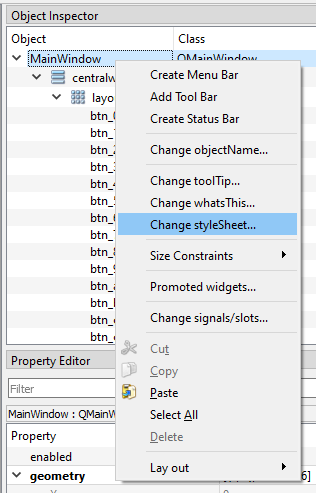
3.Серый #666 для фона кнопок при наведении.

4.Серый посветлее #888 для текста временного выражения и фона кнопок при нажатии.

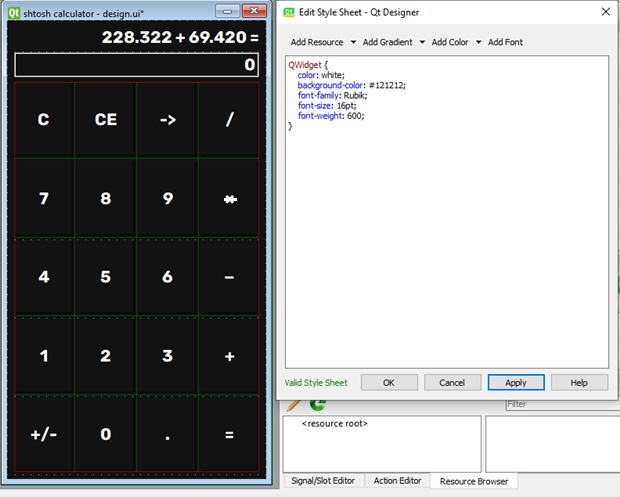
В Qt Designer поддерживается язык **css.** Напишем простенький stylesheet для главного окна. Для всего виджета указываем белый цвет текста и почти черный   
цвет #121212 для фона.

Я буду использовать бесплатный шрифт [Rubik](https://fonts.google.com/specimen/Rubik?query=rubik) из библиотеки Google Fonts. Он довольно приятный.

QWidget {   
 **color**: white;   
 **background-color**: #121212;   
 **font-family**: Rubik;   
 **font-size**: 16pt;   
 **font-weight**: 600;   
}



16



Давайте изменим кнопки на плоские с прозрачным фоном.

QPushButton {   
 **background-color**: transparent;   
 **border**: none;   
}

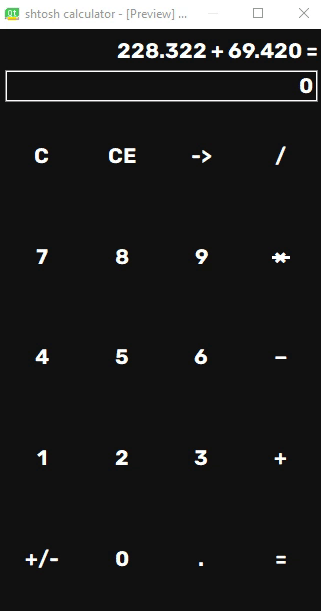
Теперь напишем изменение фона кнопок при наведении и нажатии. При наведении цвет фона будет меняться на серый #666, при нажатии на серый #888.

QPushButton:hover {   
 **background-color**: #666;   
}

QPushButton:pressed {   
 background-color: #888;   
}

17

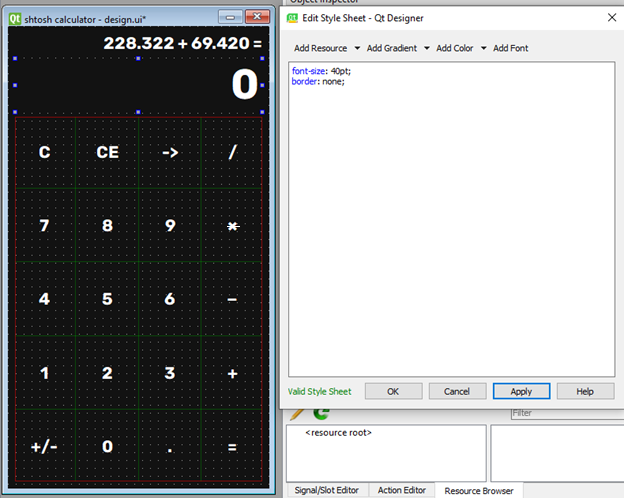
Посмотрим на результат.



18

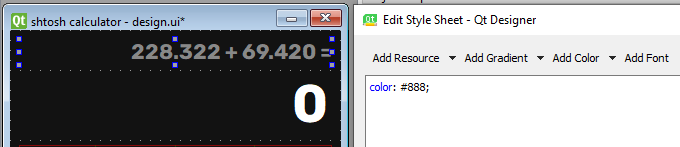
**Стили для Line Edit и Label**   
Сначала разберемся с Line Edit. Поставим размер шрифта 40pt и уберем границы. Я не буду делать какие-то изменения при наведении и нажатии, потому что пользователь не может взаимодействовать с этим полем.

**font-size**: 40pt;   
**border**: none;



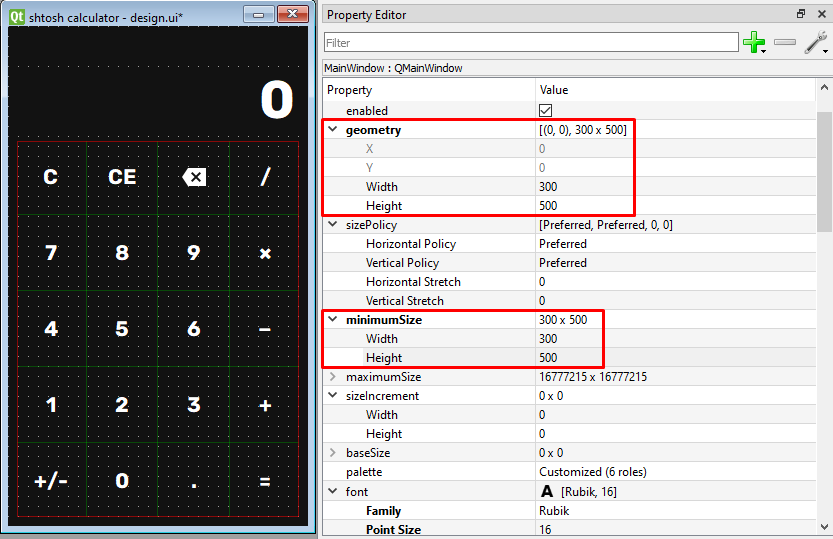
Для лейбла укажем только цвет #888. С этим элементом пользователь тоже не может взаимодействовать.

**color**: #888;

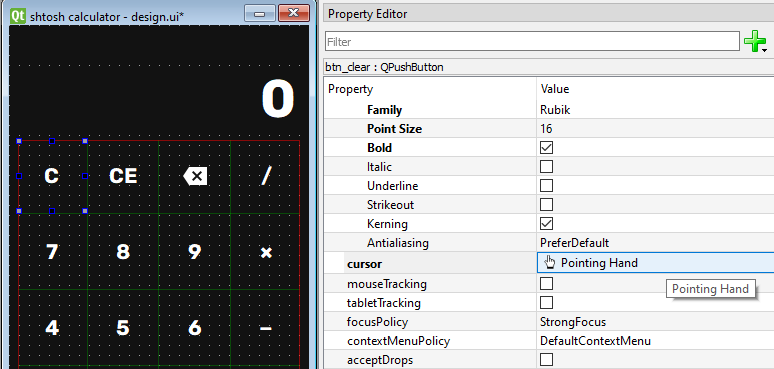


19

**Финальные штрихи**   
Почти готово. Убираем текст из лейбла. Ставим размер главного окна. У меня будет 300 на 500 пикселей. Такой же размер поставлю минимальным для приложения.



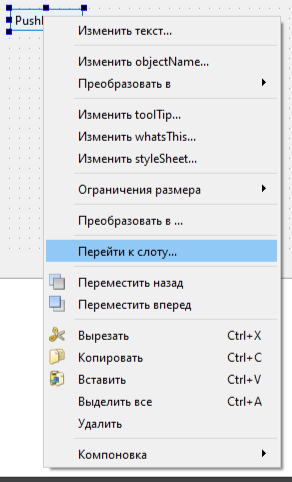
Еще добавлю такую фичу - курсор "указывающая рука" для кнопок. Поставлю только для одной кнопки, сейчас доделаем в коде.



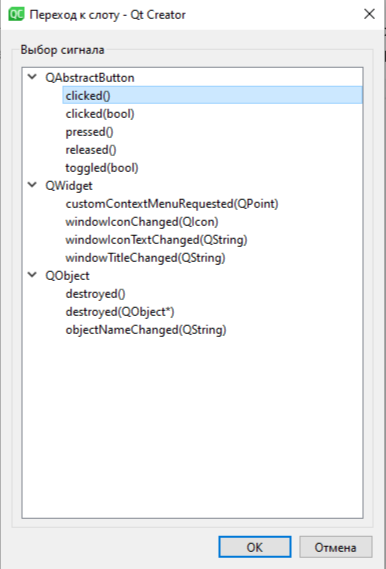
20

**Обработка событий**   
1.Чтобы привязать события к кнопке необходимо нажать правой кнопкой мышки по ней.

Откроется контекстное меню, в котором необходимо выбрать «Перейти к слоту…»



2.У QPushButton (а вообще говоря, у его базового класса QPushButton) есть сигнал clicked, который соответствует полноценному нажатию. При этом даже необязательно нажатие на кнопку должно осуществится, как результат взаимодействия пользователя с мышкой: программа сама может нажать на кнопку (метод click) или пользователь может нажать на пробел на клавиатуре, когда кнопка активна, сигнал clicked излучится и в этих случаях тоже.



21

3.При нажатии на clicked будет сгенерирован следующий код программы:

void MainWindow::on\_pushButton\_clicked()

{

}

4.Для добавления цифр lbl\_temp используйте следующее выражение:

lbl\_temp->setText(lbl\_temp->text() + button->text());

5.Для добавления операций используйте следующее выражение:

lbl\_temp->setText(lbl\_temp->text() + " " + button->text() + " ");

**ИСПОЛЬЗОВАНИЕ СИМВОЛОВ ПРОБЕЛА ПРИЦИПАЛЬНО ВАЖНО ДЛЯ ДАЛЬНЕЙШЕЙ ОБРАБОТКИ ВЫРАЖЕНИЯ!**

6.Добавление точки реализовано:   
 lbl\_temp->setText(lbl\_temp->text() + ".") 7.Очистка (СE) реализуется:

void MainWindow::clearAll() { le\_entry->clear();   
 lbl\_temp->clear();   
 }   
8.Backspace:   
 void MainWindow::backspaceClicked() { QString text\_entry = le\_entry->text(); QString text\_temp = lbl\_temp->text();

text\_entry.chop(1);   
 text\_temp.chop(1);

le\_entry->setText(text\_entry);   
 lbl\_temp->setText(text\_temp);   
}

22

9.Изменение знака числа (+/-)   
 void MainWindow::negateClicked() { QString text = le\_entry->text(); if (text.isEmpty()) {   
 return;   
 }

if (text.at(0) == '-') {   
 text.remove(0, 1);   
 } else {   
 text.prepend("-");   
 }

le\_entry->setText(text);   
}

23

**Вычисление выражения**

Ранее мы упоминали, что последовательность операций удобно реализовывать при помощи

списков. В самом деле нам пригодится данный структурированный тип, чтобы обработать

выражение.

Однако перед этим вспомним что такое файлы заголовков в С++ (.h).

Имена элементов программы, таких как переменные, функции, классы и т. д., должны быть

объявлены до их использования. Например, вы не можете просто написать x = 42 без

первого объявления "x".

Объявление сообщает компилятору, является **int** или элемент , функцией**double**,

или **class** другой вещью. Кроме того, каждое имя должно быть объявлено (прямо или

косвенно) в каждом CPP-файле, в котором он используется. При компиляции программы

каждый CPP-файл компилируется независимо в единицу компиляции. Компилятор не

знает, какие имена объявляются в других единицах компиляции. Это означает, что если вы

определяете класс или функцию или глобальную переменную, необходимо указать

объявление этой вещи в каждом дополнительном CPP-файле, который использует его.

Каждое объявление этой вещи должно быть точно идентичным во всех файлах. Небольшое

несоответствие приведет к ошибкам или непреднамеренное поведение, когда компоновщик

пытается объединить все единицы компиляции в одну программу.

Чтобы свести к минимуму потенциал ошибок, C++ принял соглашение об

**использовании *файлов* заголовков для хранения объявлений**. Вы делаете объявления в

файле заголовка, а затем используйте директиву #include в каждом CPP-файле или другом

файле заголовка, который требует этого объявления. Директива #include вставляет копию

файла заголовка непосредственно в CPP-файл перед компиляцией.

Для того, чтобы создать функцию для вычисления выражения включим ее в mainwindow.h

Назовем её ***calculateExpression*** на вход она будет получать строку из lbl и выдавать строку

на выход в edt.

***QString calculateExpression(const QString &expression);***

Чтобы вы не запутались я приведу полностью код своего mainwindow.h

Он может отличаться от вашего.

24

Содержание файла mainwindow.h   
(***добавлена функция QString calculateExpression(const QString &expression);***)

#ifndef MAINWINDOW\_H   
#define MAINWINDOW\_H

#include <QMainWindow>   
#include <QWidget>   
#include <QVBoxLayout>   
#include <QLineEdit>   
#include <QLabel>   
#include <QGridLayout>   
#include <QPushButton>

class MainWindow : public QMainWindow   
{   
 Q\_OBJECT

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

private slots:   
 void digitClicked();   
 void operatorClicked();   
 void pointClicked();   
 void equalClicked();   
 void clearEntry();   
 void clearAll();   
 void backspaceClicked();   
 void negateClicked();   
 QString calculateExpression(const QString &expression);   
private:   
 QWidget \*centralWidget;   
 QVBoxLayout \*layout;   
 QLineEdit \*le\_entry;   
 QLabel \*lbl\_temp;   
 QGridLayout \*gridLayout\_buttons;   
};

#endif // MAINWINDOW\_H

25

Для составления функции вычисления выражения рассмотрим внимательнее структуру выражения:   
 Expression = “9 + 9 \* 8”

Очевидно, что, благодаря знакам пробела данная строчка легко преобразуется в набор токенов при помощи команды split.   
 QStringList tokens = expression.split(" ");

Теперь необходимо отсортировать их по двум спискам: операторам и числам.

Объявим их:   
 QList<QString> numbers;   
 QList<QString> operators;   
В цикле рассортируем значения Токенов:   
 for (const QString &token : tokens) {   
 if (token != "+" && token != "-" && token != "\*" && token != "/") { // Если токен не является оператором, добавляем его в список чисел numbers.push\_back(token);   
 } else {   
 // Если токен - оператор, добавляем его в список операторов operators.push\_back(token);   
 }   
 }

26

Далее, предполагая, что **Оператор всегда находится между Числами**, извлечем сначала все знаки умножения (\*) и деления (/). Обратите внимание, что я делаю проверку на деление на 0!

// Выполняем операции умножения и деления for (int i = 0; i < operators.size(); ++i) {   
 if (operators.at(i) == "\*" || operators.at(i) == "/") { // Извлекаем первое число   
 double a = numbers.at(i).toDouble();

// Извлекаем оператор   
 QString op = operators.at(i);

// Извлекаем второе число   
 double b = numbers.at(i + 1).toDouble();

// Выполняем операцию в зависимости от оператора if (op == "\*") {   
 numbers[i] = QString::number(a \* b);   
 } else if (op == "/") {   
 if (b == 0.0) {   
 return "Error: Division by zero";   
 }   
 numbers[i] = QString::number(a / b);   
 }

// Удаляем использованный оператор и второе число operators.removeAt(i);   
 numbers.removeAt(i + 1);

// Уменьшаем индекс, чтобы не пропустить следующий оператор --i;

27

}

Далее, предполагая, что **Оператор всегда находится между Числами**, извлечем затем все знаки плюсы (+) и минусы (-).

// Выполняем операции сложения и вычитания while (!operators.isEmpty()) {   
 // Извлекаем первое число   
 double a = numbers.takeFirst().toDouble();

// Извлекаем оператор   
 QString op = operators.takeFirst();

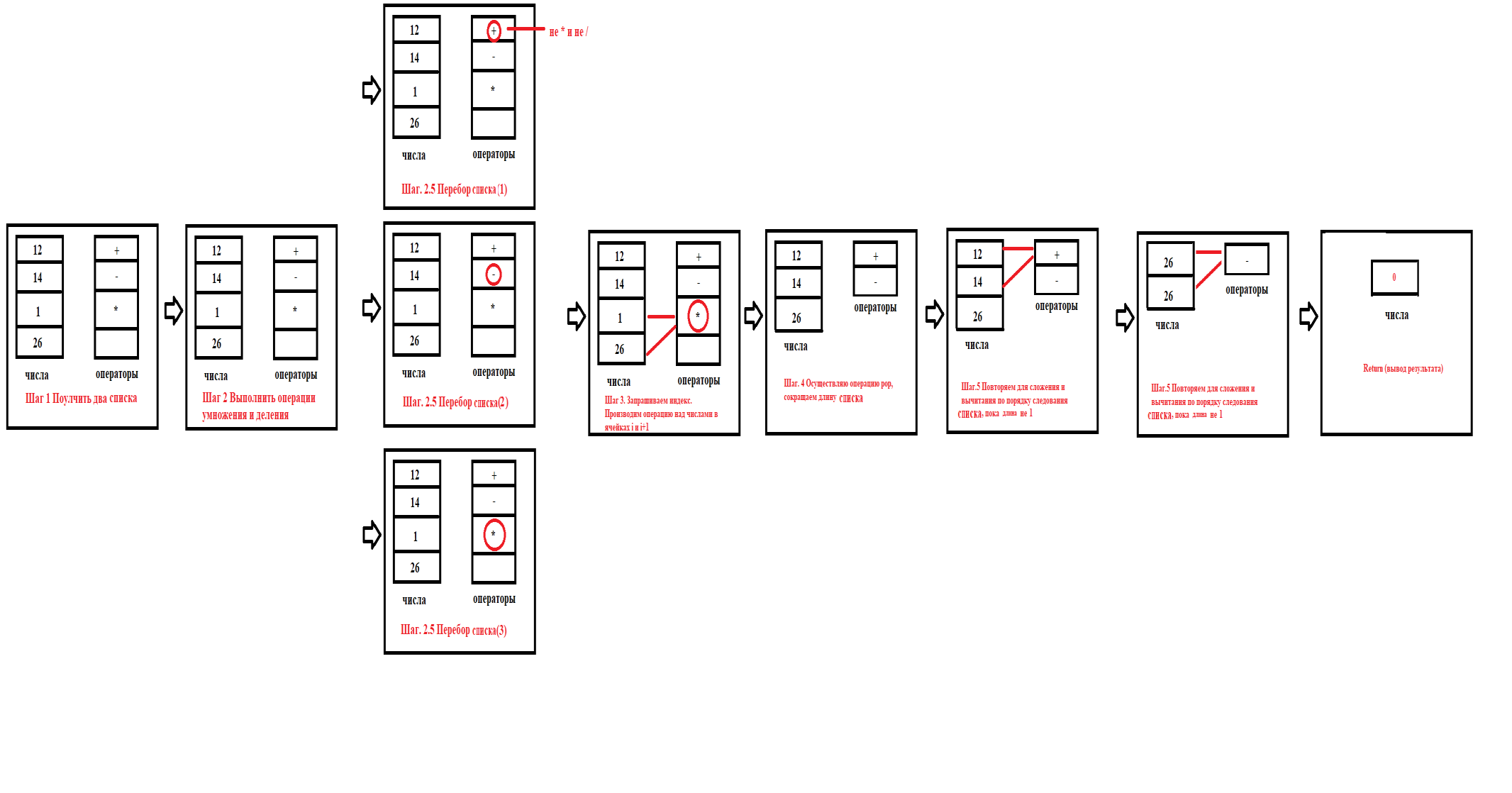
// Извлекаем второе число   
 double b = numbers.takeFirst().toDouble();

// Выполняем операцию в зависимости от оператора if (op == "+") {   
 numbers.push\_front(QString::number(a + b));   
 } else if (op == "-") {   
 numbers.push\_front(QString::number(a - b));   
 }   
 }

// В списке должен остаться один элемент - результат выражения if (numbers.size() == 1) {   
 QString result = numbers.takeFirst();   
 qDebug() << "Result:" << result;   
 return result;   
 } else {   
 return "Error";   
 }

28

Иллюстрация работы алгоритма:



29

Событие по нажати. На «=» и функция вычисления выражения приведены ниже:

void MainWindow::equalClicked() {   
 QString expression = lbl\_temp->text();   
 QString result = calculateExpression(expression);   
 le\_entry->setText(result);   
}   
QString MainWindow::calculateExpression(const QString &expression) { // Выводим токены для отладки   
 qDebug() << "Expression:" << expression;   
 QStringList tokens = expression.split(" ");   
 qDebug() << "Tokens:" << tokens;

QList<QString> numbers;   
 QList<QString> operators;

for (const QString &token : tokens) {   
 if (token != "+" && token != "-" && token != "\*" && token != "/") { // Если токен не является оператором, добавляем его в список чисел numbers.push\_back(token);   
 } else {   
 // Если токен - оператор, добавляем его в список операторов   
 operators.push\_back(token);   
 }   
 }

// Выполняем операции умножения и деления for (int i = 0; i < operators.size(); ++i) {   
 if (operators.at(i) == "\*" || operators.at(i) == "/") { // Извлекаем первое число

30

double a = numbers.at(i).toDouble();

// Извлекаем оператор   
 QString op = operators.at(i);

// Извлекаем второе число   
 double b = numbers.at(i + 1).toDouble();

// Выполняем операцию в зависимости от оператора if (op == "\*") {   
 numbers[i] = QString::number(a \* b);   
 } else if (op == "/") {   
 if (b == 0.0) {   
 return "Error: Division by zero";   
 }   
 numbers[i] = QString::number(a / b);   
 }

// Удаляем использованный оператор и второе число operators.removeAt(i);   
 numbers.removeAt(i + 1);

// Уменьшаем индекс, чтобы не пропустить следующий оператор --i;   
 }   
 }

// Выполняем операции сложения и вычитания while (!operators.isEmpty()) {   
 // Извлекаем первое число   
 double a = numbers.takeFirst().toDouble();

31

// Извлекаем оператор   
 QString op = operators.takeFirst();

// Извлекаем второе число   
 double b = numbers.takeFirst().toDouble();

// Выполняем операцию в зависимости от оператора if (op == "+") {   
 numbers.push\_front(QString::number(a + b));   
 } else if (op == "-") {   
 numbers.push\_front(QString::number(a - b));   
 }   
 }

// В списке должен остаться один элемент - результат выражения if (numbers.size() == 1) {   
 QString result = numbers.takeFirst();   
 qDebug() << "Result:" << result;   
 return result;   
 } else {   
 return "Error";   
 }   
}

32

**Требования к оформлению работы**   
1.Результаты лабораторной работы оформляются в виде doс или docx файла, состоящего из следующих разделов:

1.1.Титульный лист   
1.2.Задание: Разработка приложения «калькулятор», который будет выполнять основные арифметические операции (сложение, вычитание, умножение, деление). Калькулятор должен обладать простым и понятным пользовательским интерфейсом и обеспечивать корректное выполнение операций.

1.3.Глава 1. Разработка интерфейса (показываете проект в начале, середине и конце. Добавить комментарии по применению стилей для кнопок.

Дать комментарии к свойствам, которые вы изменяете) 1.4.Глава 2. Разработка калькулятора.

Скрин интерфейса программы. Под скрином подписываете функцию каждой кнопки. Например «Кнопка «1» добавляет 1 в строку.

Далее следует Листинг получившегося кода, разделенный над .h и .cpp

1.5. Тестирование

• Протестируйте каждую операцию калькулятора на различных входных данных. Составьте таблицу из 4 колонок (Описание теста, вводные данные, выходные данные, результат), где будет не менее 15 тестов.

33