Часть 1. Создание простого приложения

Цель работы: изучить основные принципы разработки программ с графическим интерфейсом с использованием Qt.

Ход работы: Была создана заготовка приложения Qt на C++ с системой сборки qmake, названная ex1 (Рисунки 1-3)

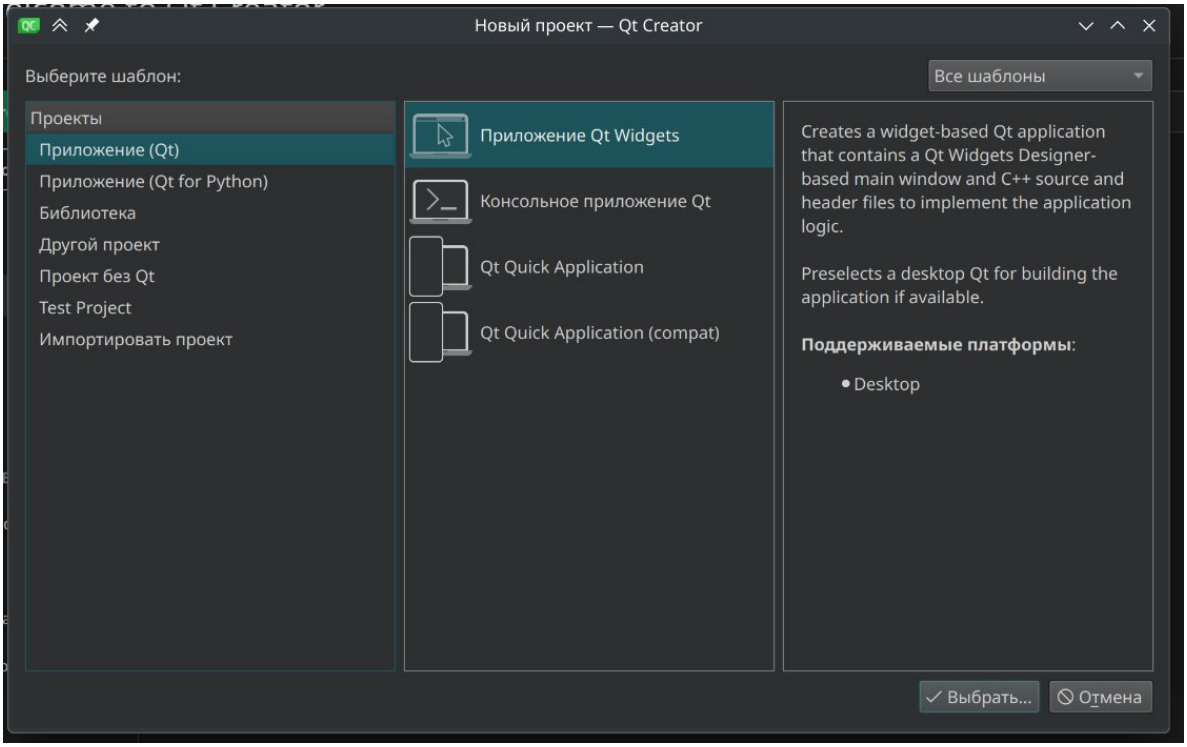

Рисунок 1 – Вид главного окна при создании проекта

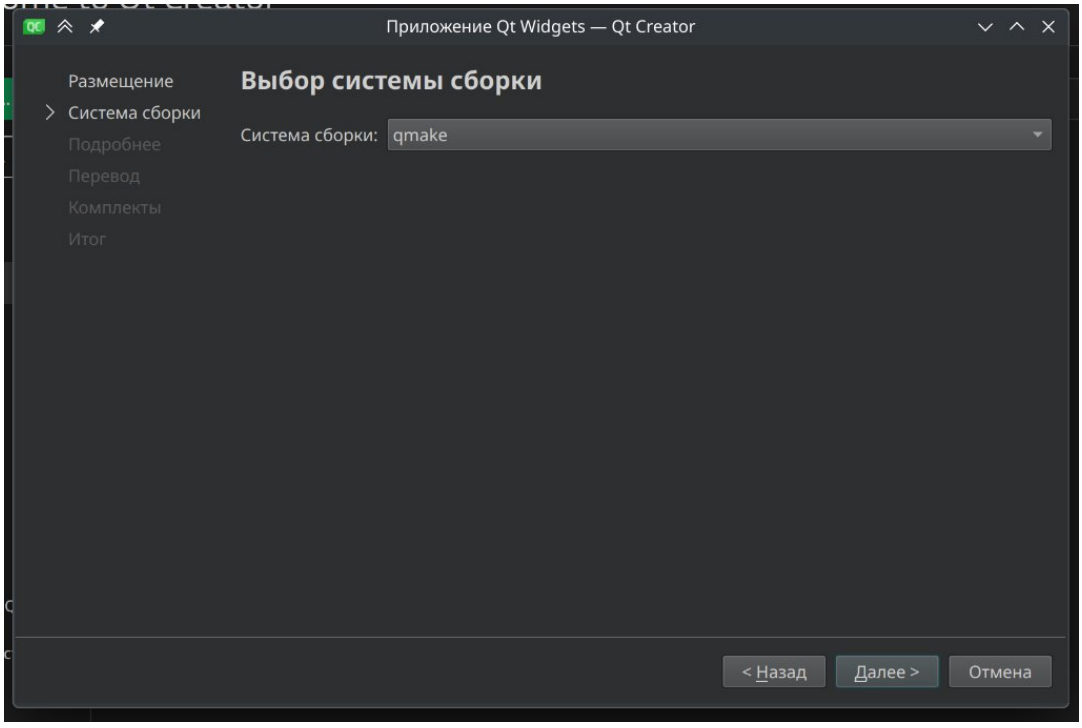

Рисунок 2 — Выбор системы сборки

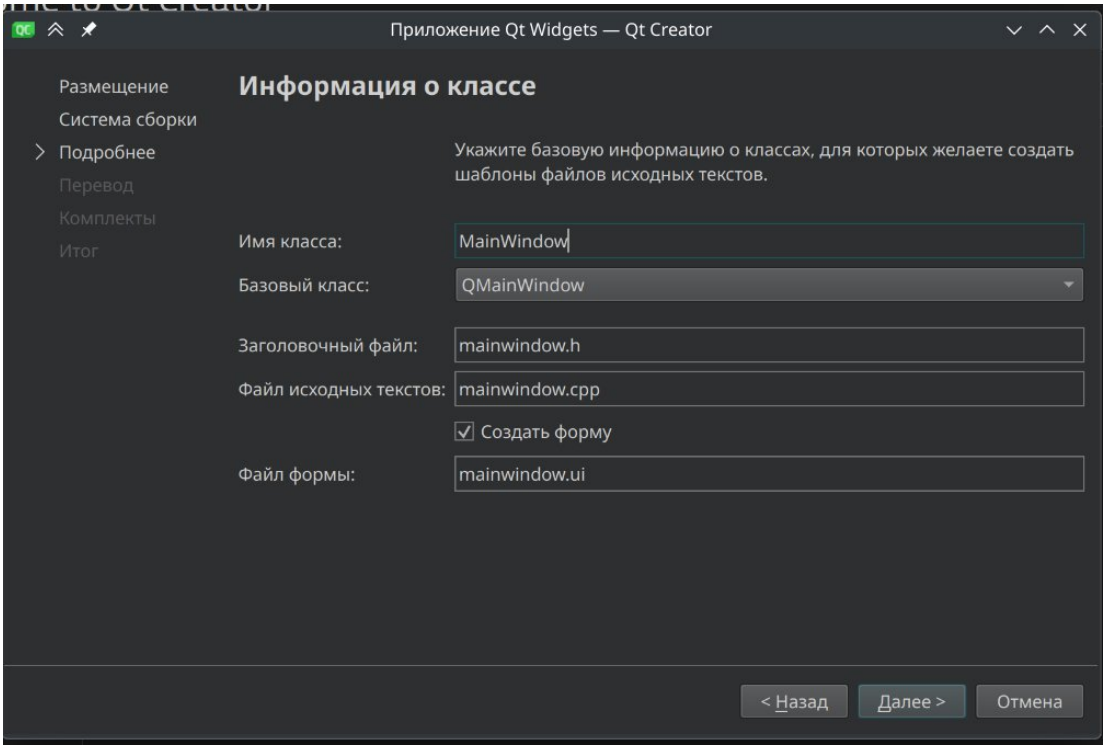

Рисунок 3 — Настройка проекта

Затем в окно редактора была введена программа для изучения принципов компоновки (layout). Затем контейнер QHBoxLayout был заменён на QVBoxLayout.

```
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    // Создаем главное окно
    QWidget *hbox = new QWidget();
    hbox->setWindowTitle( RUS("Введите Ваш Возраст") );
    QSpinBox *spinBox = new QSpinBox( hbox );
    QSlider *slider = new QSlider(Qt::Horizontal, hbox );
    spinBox->setRange(0, 130);
    slider->setRange(0, 130);
    spinBox->setValue(35);
    QPushButton * btn = new QPushButton( RUS("Завершение"), hbox );
    //*************************************************************************
    // QHBoxLayout *layout = new QHBoxLayout; // выравнивание по горизонтали
    QVBoxLayout *layout = new QVBoxLayout; // выравнивание по вертикали
    layout->setContentsMargins(5,5,5,5);
        // устанавливаем внешние границы
    layout->setSpacing(5); // устанавливаем интервал элементов внутри
    hbox->setLayout(layout);
        // связываем layout с hbox
    // устанавливаем порядок следования элементов
    layout->addWidget(spinBox);
    layout->addWidget(slider);
    layout->addWidget(btn);
    //*************************************************************************
    // связываем сигнал изменения spinBox со слотом slider
    QObject::connect(spinBox, SIGNAL(valueChanged(int)),
                     slider, SLOT(setValue(int)));
    // связываем сигнал изменения slider со слотом spinBox
    QObject::connect(slider, SIGNAL(valueChanged(int)),
                     spinBox, SLOT(setValue(int)));
    // связываем сигнал нажатия btn со слотом close главного окна
    QObject::connect(btn, SIGNAL(clicked(bool)),
                     hbox, SLOT(close()));
    hbox->show();
        // отображаем окно
    return app.exec();
        // запускаем цикл обработки сообщений
}
```

Рисунок 4 — Код программы

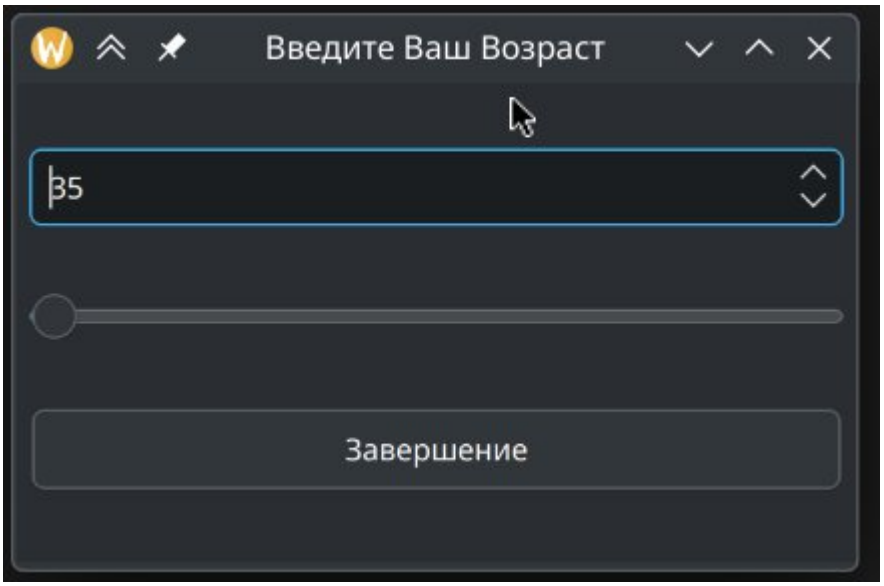

Рисунок 5 — Получившийся результат

Вывод: было получено представление о библиотеке Qt, принципов, на которых построен этот фраемворк, а также были получены базовые навыки работы с ним: компоновка, запуск, обработка ввода.

Часть 2. Создание простого приложения в Qt Designer

Цель: научиться создавать формы в Qt Designer, а также научиться работать с сигналами/слотами

Ход работы: в приложении Qt designer было выбрано “Dialog without buttons”, изменено objectName, добавлены Horizontal Spacer и Vertical Spacer для улушения макета. Были добавлены сигналы для связывания значений между компонентами (Рисунки 6-10).

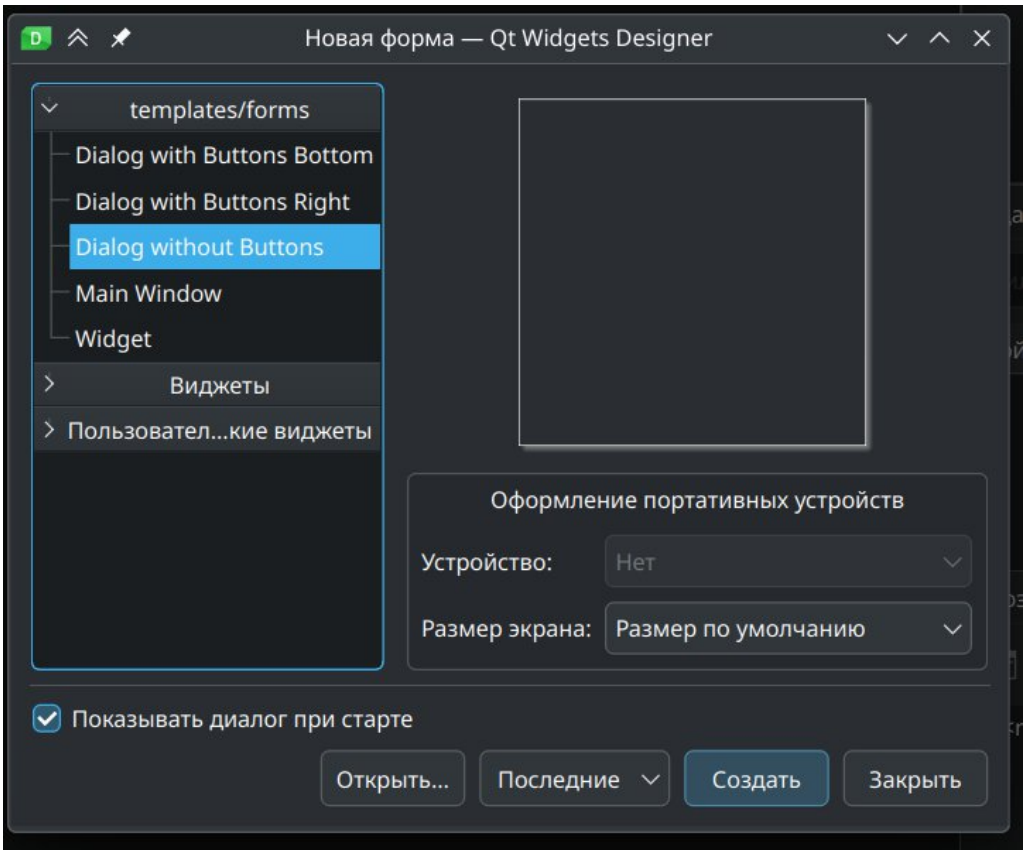

Рисунок 6 — Окно выбора типа формы

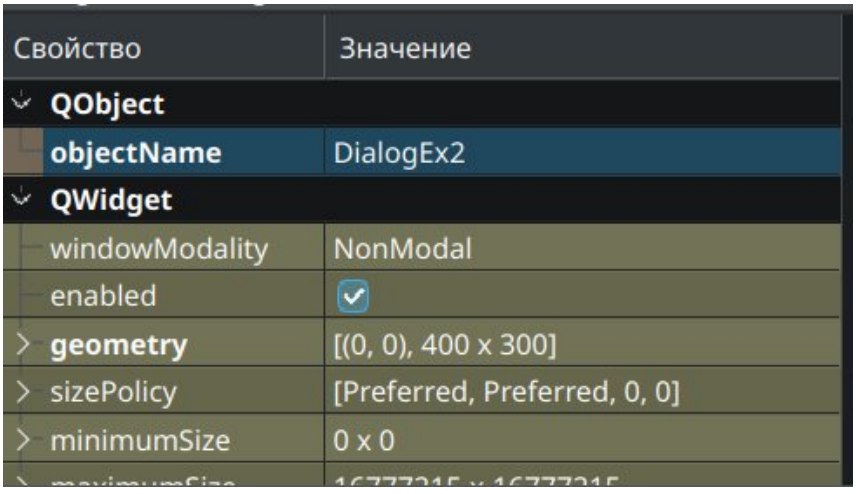

Рисунок 7 — окно выбора свойств

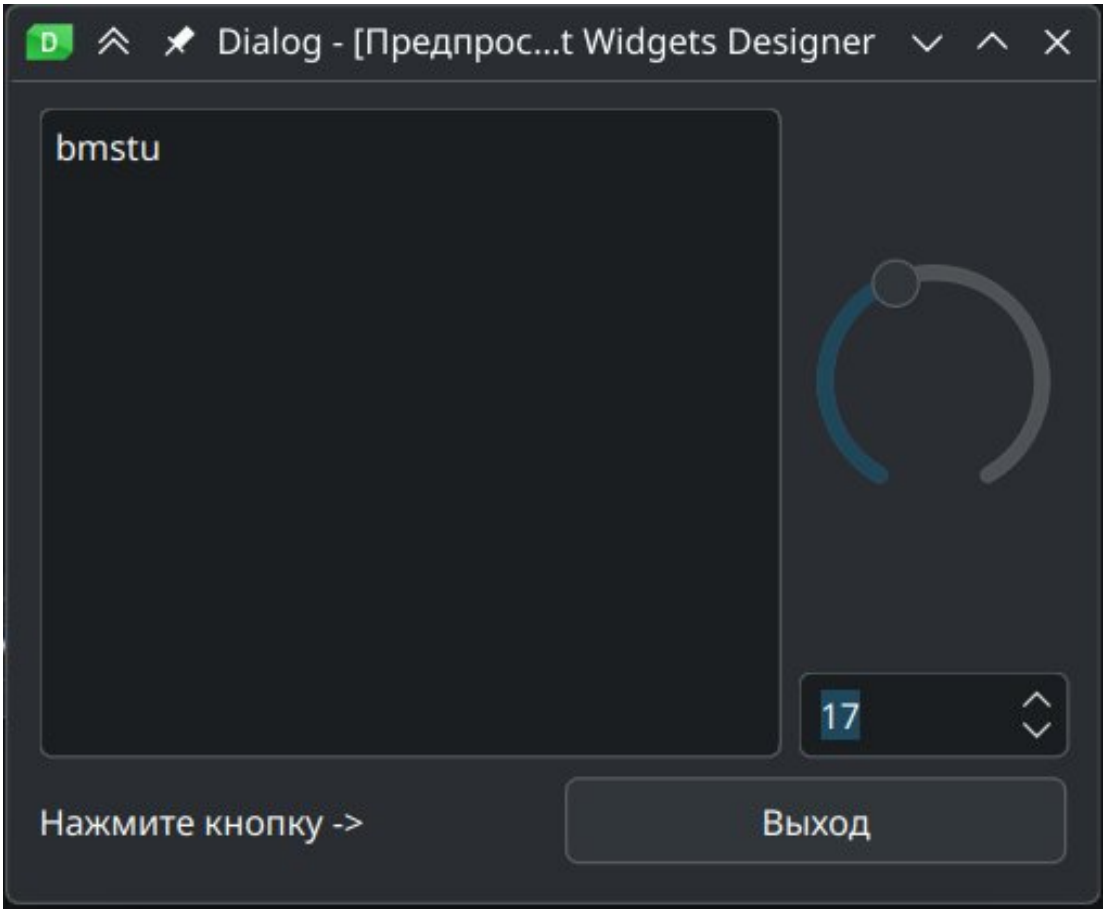

Рисунок 8 — Композиция до использования Spacer

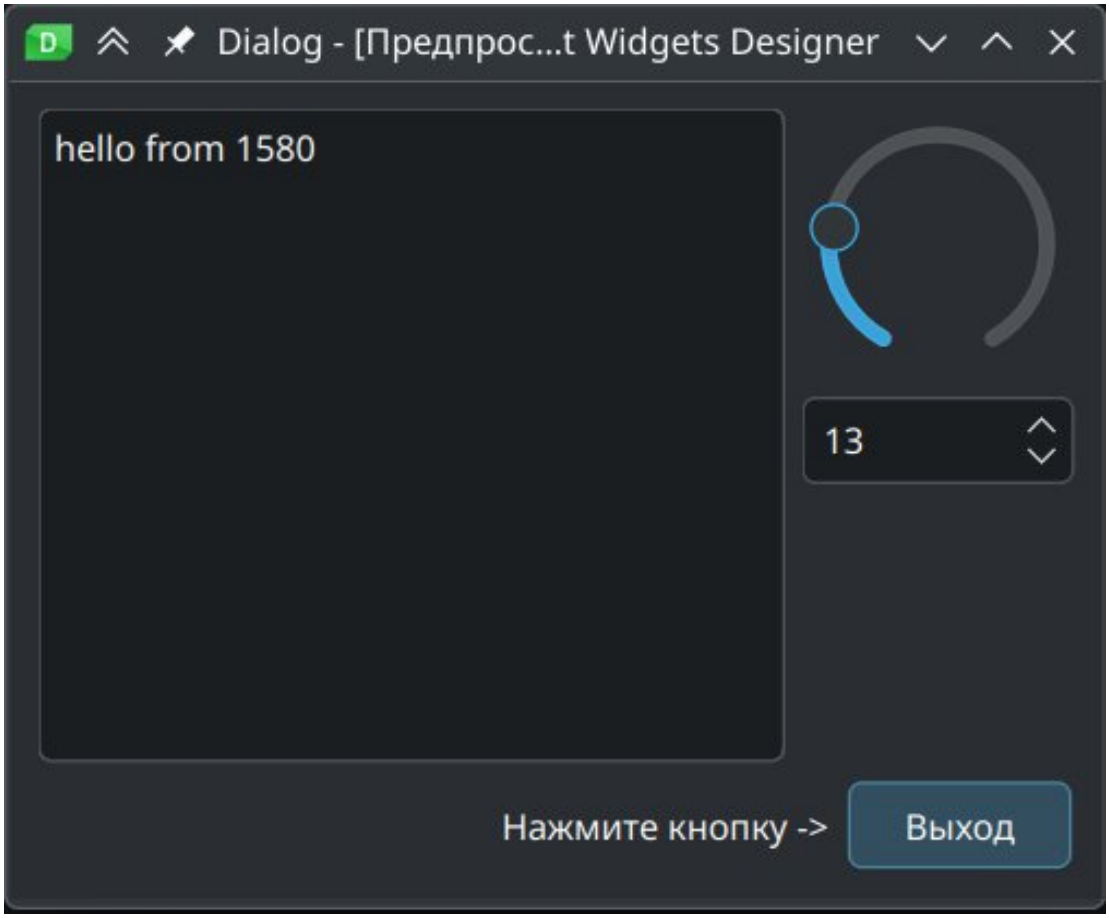

Рисунок 9 — Композиция после использования Spacer

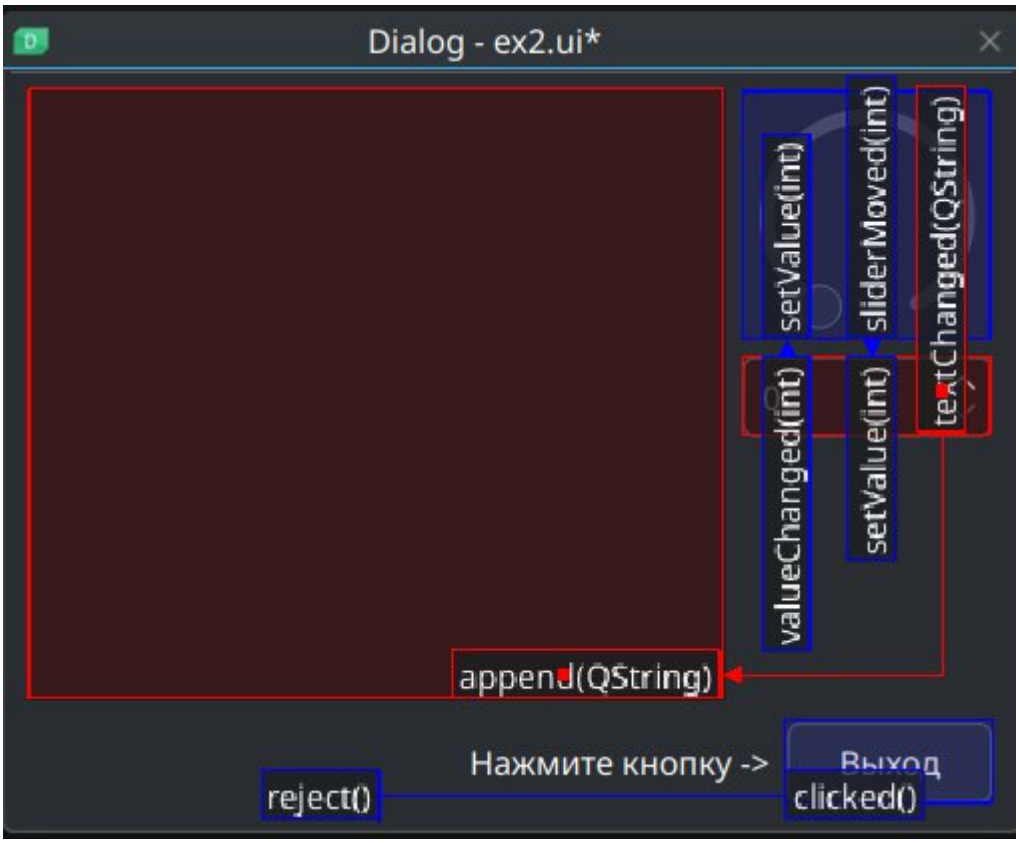

Рисунок 10 — Схема сигналов

С помощью кода было добавлено диалоговое окно, которое открывается при попытке закрыть приложение.

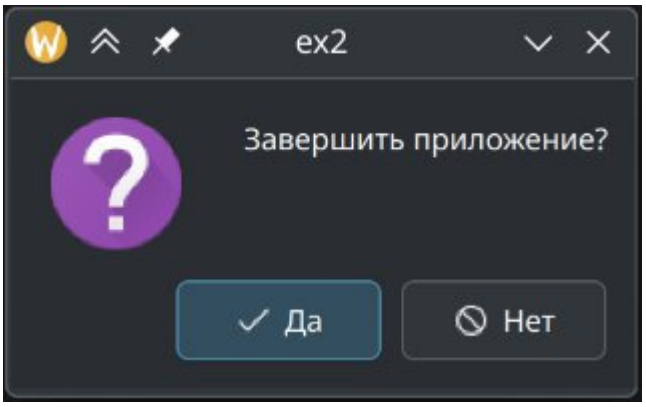

Рисунок 11 — Диалоговое окно

С помощью компонента QSplitter (рисунок 12) мы расположили рядом два одинаковых виджета (рисунок 13)

```cpp
#include "dialogex2.h"
#include <QApplication>
#include <QSplitter>
int main(int argc, char *argv[]) {
  QApplication app(argc, argv);

  /*
  // Отображаем форму так, как сделано в QtDesigner
  DialogEx2 *dialog1 = new DialogEx2();
  dialog1->show();   // отображаем окно
*/
  // Отображаем две формы горизонтально с вертикальным разделителем
  QSplitter *splitter = new QSplitter(Qt::Vertical);
  DialogEx2 *dialog1 = new DialogEx2();
  DialogEx2 *dialog2 = new DialogEx2();
  splitter->addWidget(dialog1);
  splitter->addWidget(dialog2);
  splitter->show();
  // отображаем окно
  return app.exec(); // запускаем цикл обработки сообщений
}
```

Рисунок 12 — Код программы с QSplitter

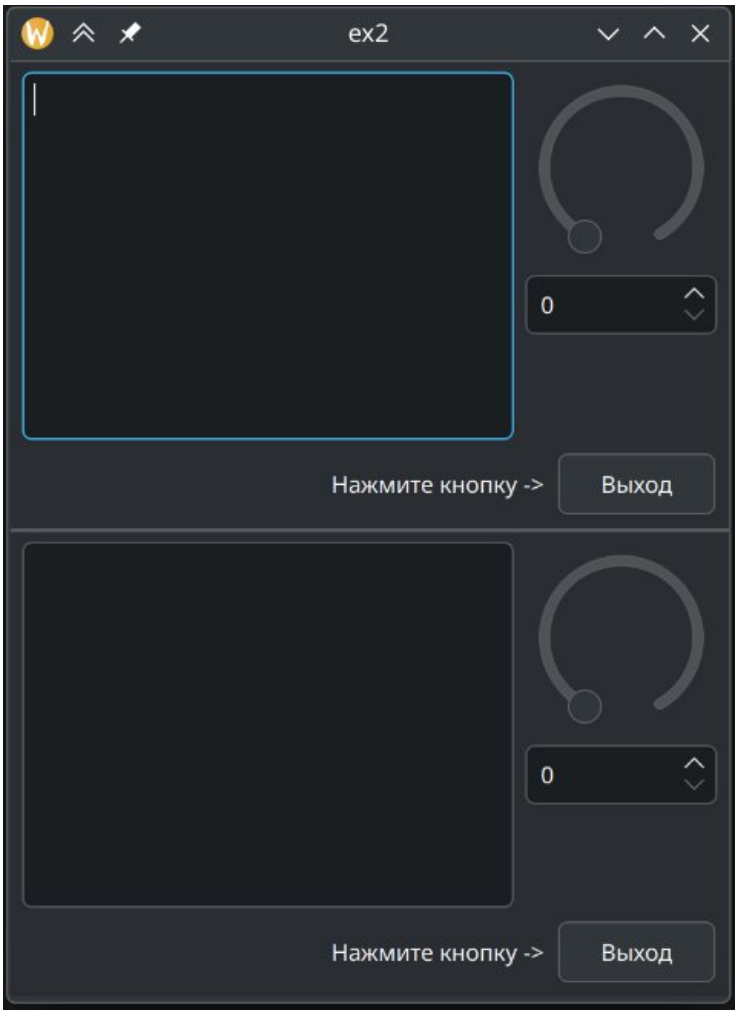

Рисунок 13 — Дизайн итогового окна

Вывод: было получено представление о работе с Qt Designer и слотами

Часть 3. Калькулятор

Цель работы: разработать и расширить функционал калькулятора, добавив новые бинарные (x^y, $\log_y(x)$) и унарные (sin(x), cos(x)) операции.

Выполнение: сначала был разработан дизайн базового калькулятора, представленного в примере (рисунок 14).

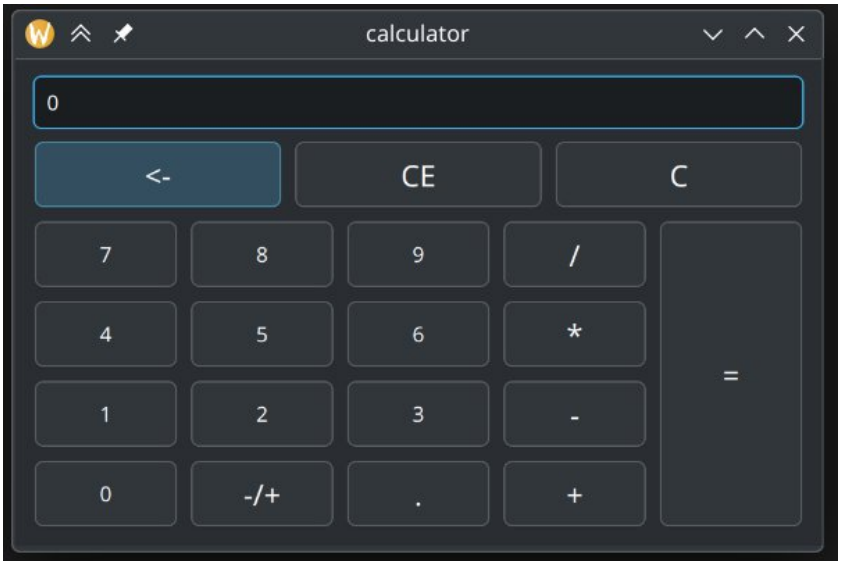

Рисунок 14 — Базовый дизайн калькулятора

Были добавлены и реализованы дополнительные операции и проверки, связанные с ними (рисунки 15-17).

```
_btnDescr.push_back(BtnDescr("x^y", POKAZ));
_btnDescr.push_back(BtnDescr("log y (x)", LOG));
_btnDescr.push_back(BtnDescr("sin(x)", SIN));
_btnDescr.push_back(BtnDescr("cos(x)", COS));
```

Рисунок 15 — Добавление кнопок новых действий

```
case SIN:
  setNumEdit(std::sin(getNumEdit()));
  break;
case COS:
  setNumEdit(std::cos(getNumEdit()));
  break;
```

Рисунок 16 — Реализация унарных операций

```
case POKAZ: {
  m_Val = std::pow(m_Val, num);
  break;
}
case LOG: {
  if (m_Val > 0 && num > 0 && num != 0)
    m_Val = std::log(m_Val) / std::log(num);
  else
    m_Val = 0;
  break;
}
```

Рисунок 17 — Релизация бинарных операций и проверок, связанных с ними

Итоговый результат изображен на рисунке 18.

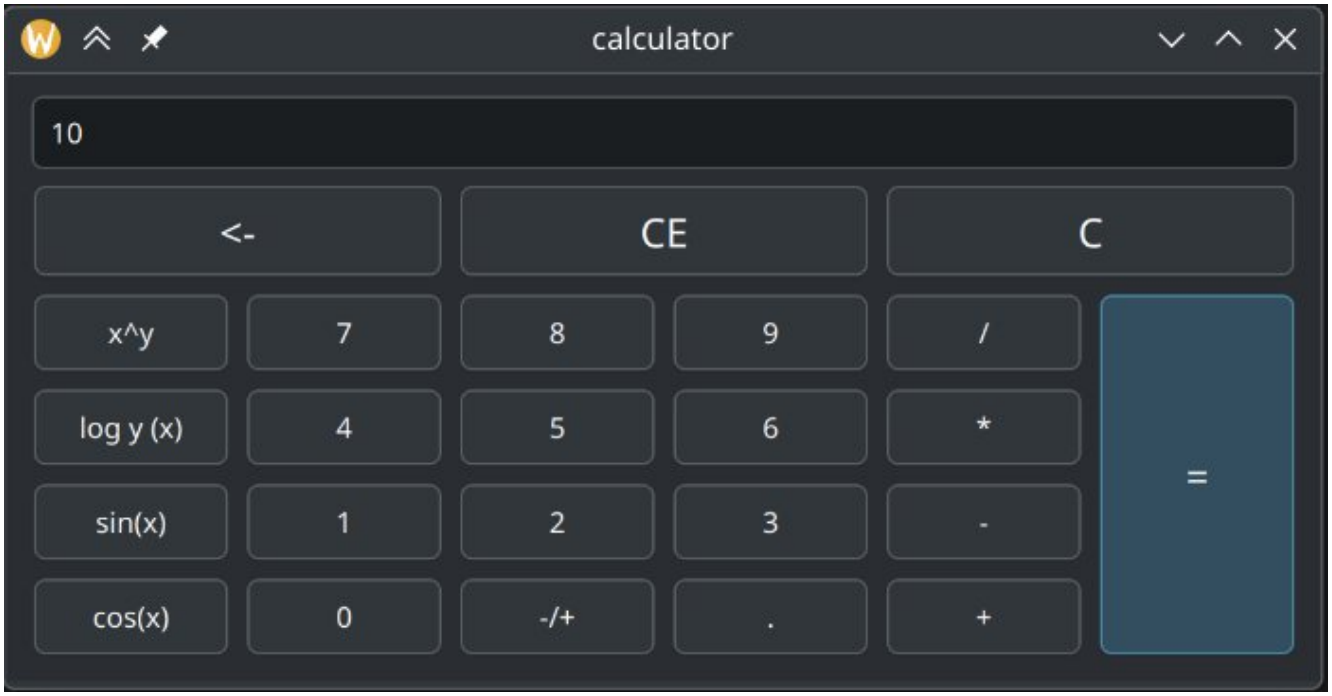

Рисунок 18 — Итоговый дизайн калькулятора

Вывод: был разработан калькулятор с базовыми функциями, используя библиотеку Qt и C++. Затем программа была усовершенствованна: были добавлены новые кнопки (компановка с помощью QVBoxLayout), а также действия, соответствующие им.QVBoxLayout

Часть 4. Конвертер строк

Цель работы: разработать программу, не используя Qt designer, которая выполняет следующие действия: при нажатии на кнопку введённая строка преобразуется в маленький регистр или большой (поочерёдно).

Выполнение: был разработан класс MainWindow, хранящий дизайн (рисунок 19) и логику программы (рисунки 20-21). Заголовочный файл изображен на рисунке 22.

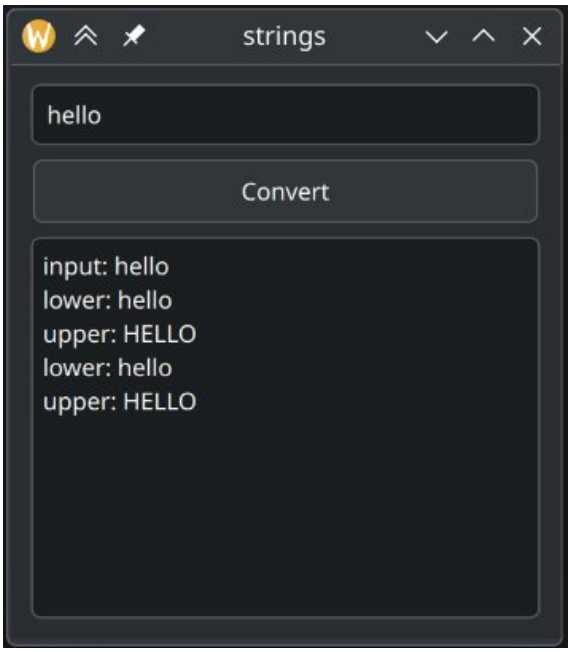

Рисунок 19 — Дизайн программы

```
void MainWindow::onButtonClick() {
  if (lastLine.isEmpty())
    return;
  if (lower)
    resultField->append("lower: " + lastLine.toLower());
  else
    resultField->append("upper: " + lastLine.toUpper());
  lower = !lower;
}

void MainWindow::onEditingFinished() {
  if (inputLine->text().isEmpty() || lastLine == inputLine->text())
    return;
  lastLine = inputLine->text();

  resultField->append("input: " + lastLine);
  lower = true;
}
```

Рисунок 20 — Логика программы

```cpp
MainWindow::MainWindow() { setupUI(); }

void MainWindow::setupUI() {
  mainLayout = new QVBoxLayout(this);

  inputLine = new QLineEdit();
  mainLayout->addWidget(inputLine);

  convertBtn = new QPushButton();
  convertBtn->setText("Convert");
  convertBtn->setAutoDefault(false);
  mainLayout->addWidget(convertBtn);

  resultField = new QTextEdit();
  resultField->setReadOnly(true);
  mainLayout->addWidget(resultField);

  setLayout(mainLayout);

  connect(convertBtn, SIGNAL(clicked()), this, SLOT(onButtonClick()));
  connect(inputLine, SIGNAL(editingFinished()), this,
          SLOT(onEditingFinished()));
}
```

Рисунок 21 — Настройка интерфеса программы

```cpp
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QDialog>
#include <QLineEdit>
#include <QPushButton>
#include <QTextEdit>
#include <QVBoxLayout>

class MainWindow : public QDialog {
  Q_OBJECT

  QLineEdit *inputLine;
  QPushButton *convertBtn;
  QTextEdit *resultField;
  QVBoxLayout *mainLayout;

  bool lower = true;
  QString lastLine;

public:
  MainWindow();
  void setupUI();

private slots:
  void onButtonClick();
  void onEditingFinished();
};

#endif // MAINWINDOW_H
```

Рисунок 22 — Заголовочный файл класса MainWindow

Вывод: в ходе выполнения задачи по работе со строками было успешно разработано приложение, удовлетворяющее всем требованиям задания: реакция на завершение ввода, преобразование регистра, интерфейс и компоновка.