МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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

Кафедра радиоэлектронных устройств и систем

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ №1-3
для студентов специальности 11.05.01
«Радиоэлектронные системы и комплексы»
очной формы обучения

Воронеж 2024

УДК 681.3.06(07)
ББК 32.97я7

Составитель А. И. Сукачев

Информационные технологии: методические указания к выполнению лабораторных работ №1-3 для студентов специальности 11.05.01 «Радиоэлектронные системы и комплексы» очной формы обучения / ФГБОУ ВО «Воронежский государственный технический университет»; сост.: А. И. Сукачев. – Воронеж: Изд-во ВГТУ, 2024. – 38 с.

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

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

Методические указания подготовлены в электронном виде и содержатся в файле МУ_ИТ_ЛР1-3.pdf.

Ил. 29. Табл. 15. Библиогр.: 3 назв.

УДК 681.3.06(07)
ББК 32.97я7

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

Издается по решению редакционно-издательского совета Воронежского государственного технического университета

1. ЛАБОРАТОРНАЯ РАБОТА № 1
ФРЕЙМВОРК QT

1.1. ОБЩИЕ УКАЗАНИЯ

1.1.1. ЦЕЛЬ РАБОТЫ

Целью лабораторной работы является разработка типовой интерфейса.

1.1.2. ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Основным содержанием работы является ознакомление со структурой фреймворка Qt, создание проекта с помощью фреймворка Qt.В ходе работы идёт ознакомление с приведённым примером приложения, производится анализ используемых технологий. На основе полученных знаний выполняется индивидуальное задание.

1.2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.2.1. СОЗДАНИЕ ПРИЛОЖЕНИЯ НА ОСНОВЕ ВИДЖЕТА QT

Далее приведено описание, как использовать Qt Creator для создания небольшого приложения Qt, TextFinder. Это упрощенная версия примера Qt UI Tools TextFinder. Пользовательский интерфейс приложения построен из виджетов Qt с помощью Qt Designer. Логика приложения написана на языке C++ с помощью редактора кода.

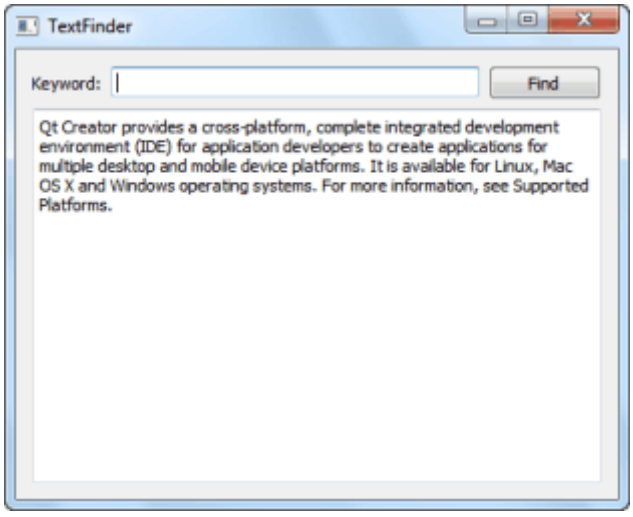


Рис. 1. Окно TextFinder

1.2.2. СОЗДАНИЕ ПРОЕКТА TEXT FINDER

Выберите File>New File или Project>Application>Qt Widgets Application>Choose… (рис. 2).

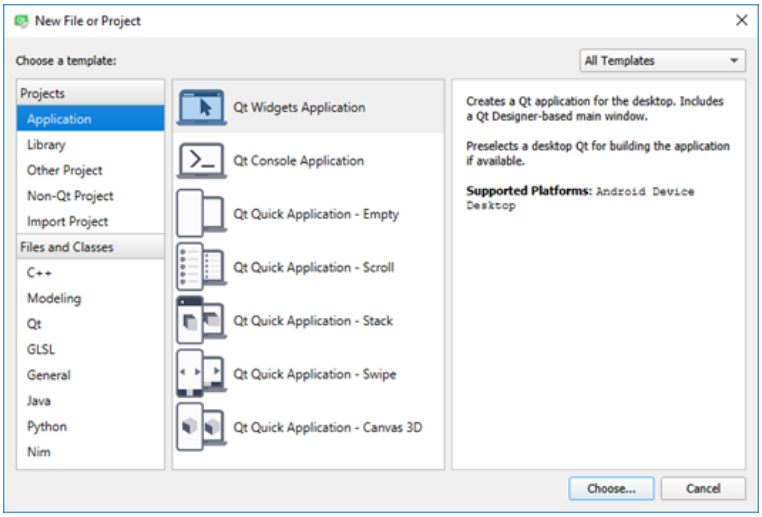


Рис. 2. Окно создания нового проекта

Откроется диалоговое окно введение название и расположение проекта.

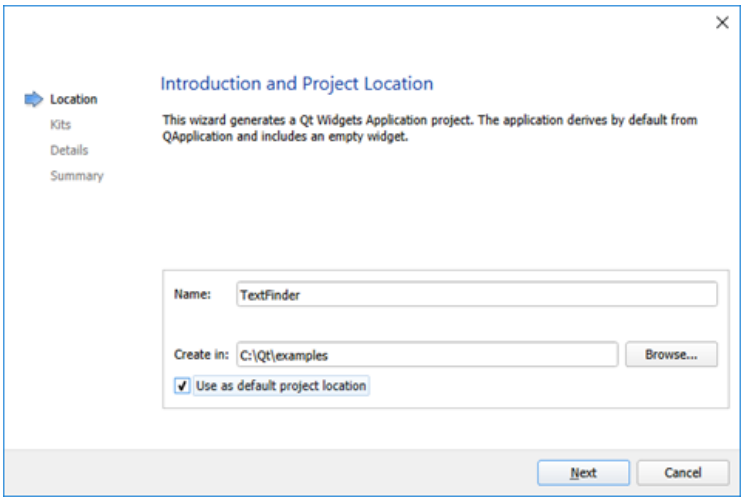


Рис. 3. Окно настроек проекта

В поле “Name” введите TextFinder. В поле “Create in” введите путь к файлам проекта. Например C:\Qt\examples, а затем нажмите кнопку Далее (в Windows и Linux) или продолжить (на macOS).

Откроется диалоговое окно выбора комплекта.

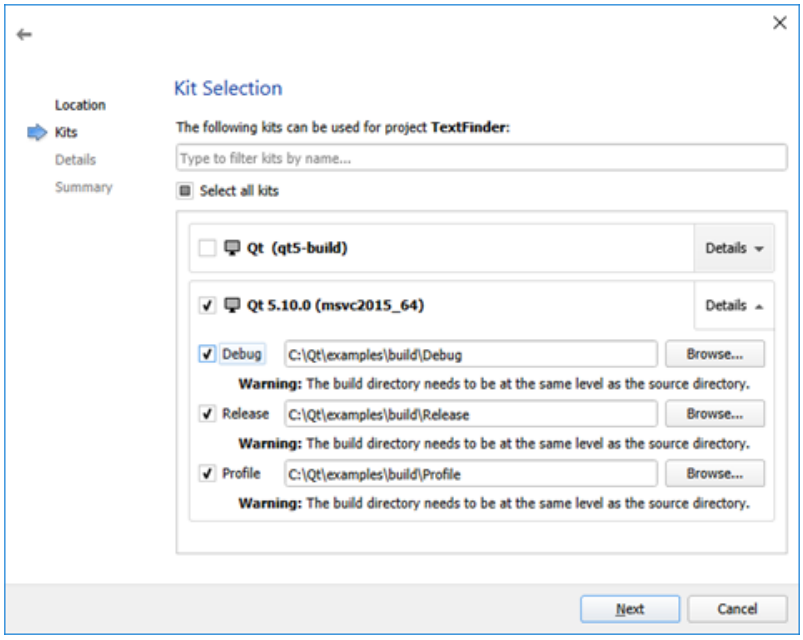


Рис. 4. Окно выбора комплекта

Выберите комплекты для вашего проекта, а затем нажмите кнопку “далее” или “продолжить”. Примечание: если указан только один комплект, этот диалог пропускается. Далее откроется диалоговое окно сведений о классе.

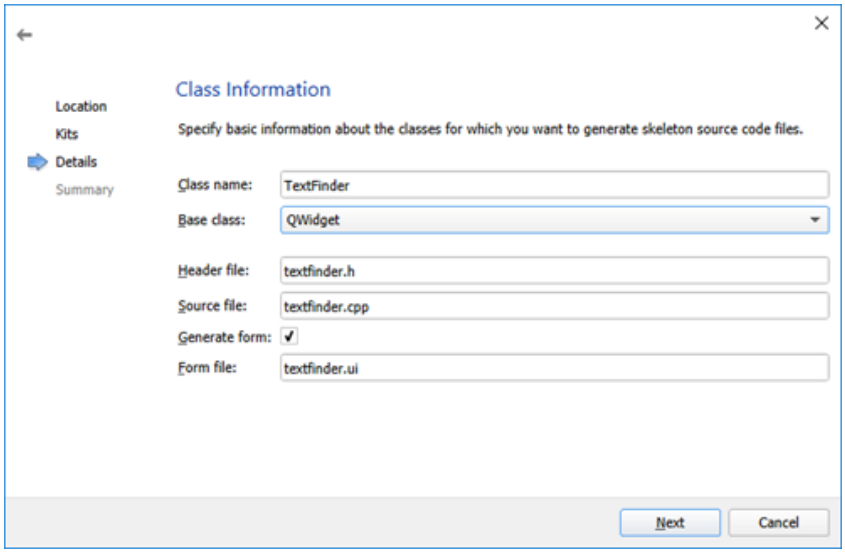


Рис. 5. Окно сведений о классе

В поле “имя класса” введите TextFinder в качестве имени класса. В списке “базовый класс” выберите “QWidget” в качестве типа базового класса. Примечание: поля “Header file”, “Source file” и “Form file” автоматически обновляются в соответствии с именем класса. Нажмите кнопку “далее” или “продолжить”. Откроется диалоговое окно “Управление проектами”.

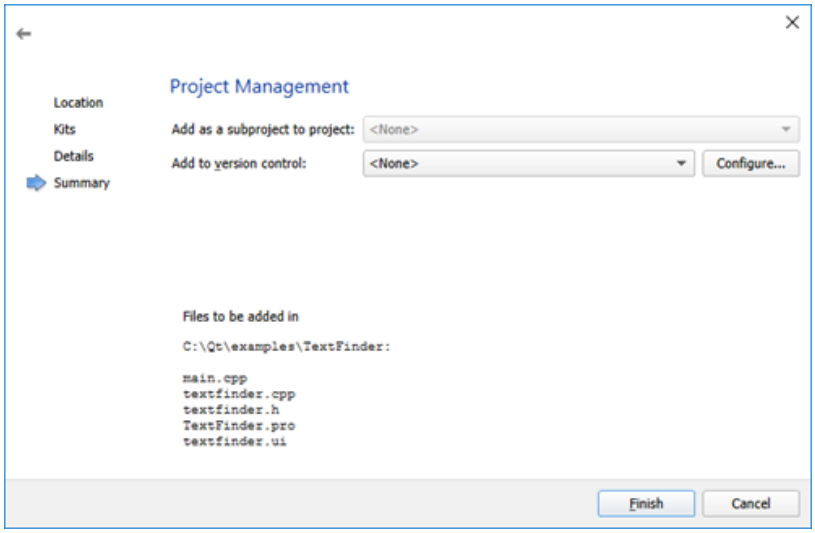


Рис. 6. Окно “Управление проектами”

Просмотрите параметры проекта и нажмите кнопку “Готово”, чтобы создать проект. Примечание: проект открывается в режиме редактирования, и эти инструкции скрыты. Чтобы вернуться к этим инструкциям, откройте режим справки.

Проект TextFinder теперь содержит следующие файлы:

textfinder.x

textfinder.CPP

Главная.CPP

textfinder.пользовательский интерфейс

textfinder.pro

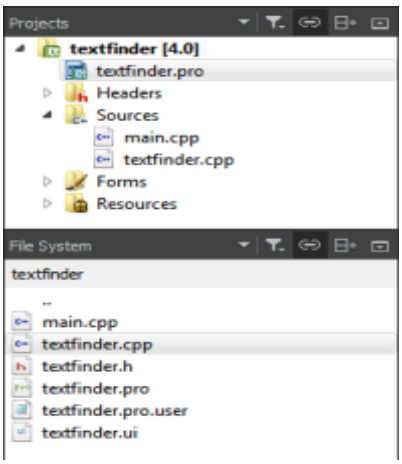


Рис. 7. Структура проекта

Файлы .h и файлы .cpp создаются с необходимым шаблонным кодом. Файл .pro является полным.

1.2.3. ЗАПОЛНЕНИЕ НЕДОСТАЮЩИХ ЧАСТЕЙ

Начнем с разработки пользовательского интерфейса, а затем перейдем к заполнению недостающего кода. В конце добавим функцию поиска.

Проектирование пользовательского интерфейса:

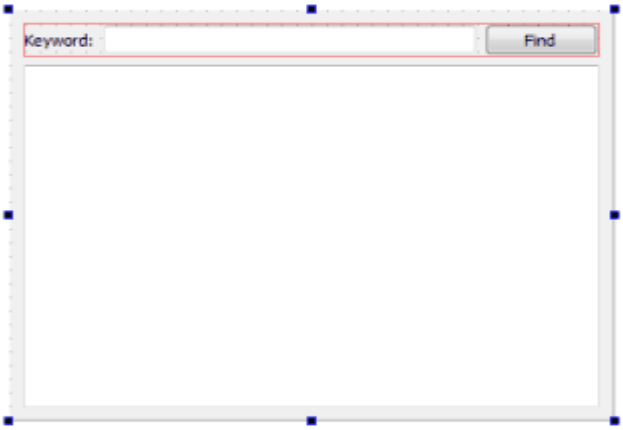


Рис. 8. Окно проектирования пользовательского интерфейса

В режиме редактора дважды щелкните на поле текстового поиска в файле пользовательского интерфейса в режиме проектирования проекта для запуска интегрированного Qt Designer.

Перетащите в форму следующие виджеты:

- Метка (QLabel)
- Редактирование строк (QLineEdit)
- Кнопка (QPushButton)

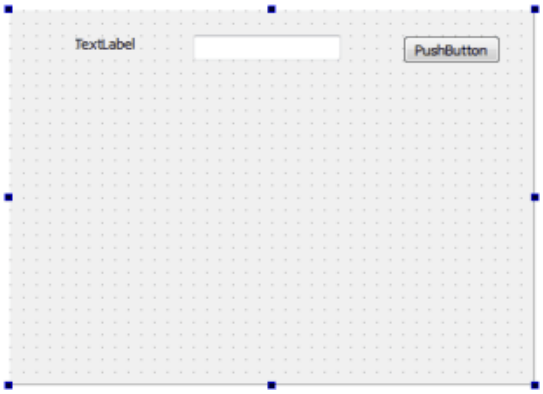


Рис. 9. Окно проекта с добавленными виджетами

Примечание: чтобы легко найти виджеты, используйте поле поиска в верхней части боковой панели.

Дважды щелкните виджет “метка” и введите ключевое слово текста. Затем дважды щелкните виджет “кнопка” и введите текст поиска. На панели “Свойства” измените имя объекта на findButton.

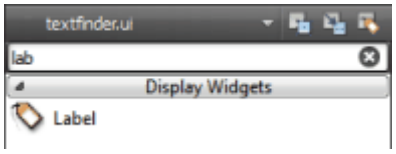


Рис. 10. Панель виджетов проекта

Нажмите Ctrl+A (или Cmd+A), чтобы выбрать виджеты, и нажмите “Lay out horizontally” (или нажмите Ctrl+H на Linux или Windows или Ctrl+Shift+H на macOS), чтобы применить горизонтальный макет (QHBoxLayout).

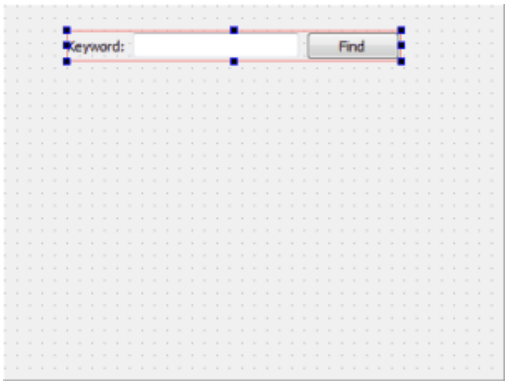


Рис. 11. Горизонтальное расположение виджетов

Далее переместите виджет редактирования текста (QTextEdit) в форму. Выберите область экрана и нажмите кнопку выкладки по вертикали (или нажмите Ctrl+L), чтобы применить вертикальную компоновку (QVBoxLayout).

Применение горизонтальных и вертикальных компоновок гарантирует, что пользовательский интерфейс приложения масштабируется до различных размеров экрана.

Чтобы вызвать функцию поиска, когда пользователи нажимают кнопку поиска, нужно использовать механизм сигналов и слотов Qt. Сигнал испускается, когда происходит определенное событие, и исполняется слот-функция, которая вызывается в ответ на определенный сигнал. Виджеты Qt имеют предопределенные сигналы и слоты, которые вы можете использовать непосредственно из Qt Designer. Чтобы добавить слот для функции поиска:

1. Щелкните правой кнопкой мыши кнопку “Найти”, чтобы открыть контекстное меню.
2. Выберите Перейти в слот > щелкните ()>, а затем нажмите кнопку ОК .
3. Частный слот, on_findButton_clicked(), добавляется в файл заголовка, textfinder.h и частная функция,TextFinder::on_findButton_clicked(), добавляется в исходный файл, textfinder.CPP.
4. Нажмите Ctrl+S (или Cmd+S), чтобы сохранить изменения.

С дополнительной информацией о проектировании форм с помощью Qt Designer можно ознакомиться в руководстве Qt Designer Manual.

1.2.4 ЗАВЕРШЕНИЕ ФАЙЛА ЗАГОЛОВКА

Файл textfinder.h уже имеет необходимые #includes, конструктор, деструктор и UI объект. Вам нужно добавить закрытую функцию loadTextFile(), чтобы прочитать и отобразить содержимое входного текстового файла в QTextEdit.

На панели проекта в режиме редактирования дважды щелкните файл textfinder.h, чтобы открыть его для редактирования.

Добавьте закрытую функцию в private раздел после Ui::TextFinderуказателя, как показано в следующем фрагменте кода:

```
        void on_findButton_clicked ();
private :
        Ui :: TextFinder  * ui ;
        void loadTextFile();
```

1.2.5. ЗАВЕРШЕНИЕ РАБОТЫ С ИСХОДНЫМ ФАЙЛОМ

Теперь, когда файл заголовка завершен, перейдите к исходному файлу, textfinder.CPP.

На панели проекты в режиме редактирования дважды щелкните файл textfinder.cpp, чтобы открыть его для редактирования.

Добавьте код для загрузки текстового файла с помощью QFile, прочитайте его с помощью QTextStream, а затем отобразите его textEditc помощью QTextEdit::setPlainText (). Это иллюстрируется следующим фрагментом кода:

```
void  TextFinder :: loadTextFile ()
{
        QFile inputFile ( ": / input.txt");
inputFile.open (QIODevice :: ReadOnly );

        QTextStream  in ( & inputFile);
        QString line =  in .readAll();
inputFile.close ();

ui - > > textEdit - > > setPlainText (line);
        qtextcursor cursor = ui - > > textEdit - > > textCursor ();
Cursor.movePosition ( QTextCursor :: Start,  QTextCursor :: MoveAnchor,  1);
}
```

Чтобы использовать QFile и QTextStream, добавьте в textfinder следующие #includes.CPP:

```
#include < QFile> >
#include < QTextStream>
```

Для on_findButton_clicked() слота добавьте код для извлечения строки поиска и используйте функцию QTextEdit::find () для поиска строки поиска в текстовом файле. Это иллюстрируется следующим фрагментом кода:

```
void  TextFinder :: on_findButton_clicked ()
{
        QString searchString = ui - > > lineEdit - > > text ();
ui - > > textEdit - > > find (  строка поиска, QTextDocument :: FindWholeWords );
}
```

Как только обе эти функции будут завершены, добавьте строку для вызова loadTextFile()в конструкторе, как показано в следующем фрагменте кода:

```
TextFinder :: TextFinder ( QWidget  * parent):
        QWidget ( parent), ui ( new  Ui:: TextFinder)
{
ui - > > setupUi (это);
loadTextFile();
}
```

Слот on_findButton_clicked() вызывается автоматически в UI файле ui_textfinder.h, сгенерированном по этой строке кода:

```
QMetaObject::connectSlotsByName(TextFinder);
```

1.2.6. СОЗДАНИЕ ФАЙЛА РЕСУРСОВ

Вам понадобится файл ресурсов (.qrc), в который будет вставлен входной текстовый файл. Входной файл может быть любым .txt файл с абзацем текста. Создайте текстовый файл с именем input.txt и сохраните его в папке textfinder.

Алгоритм добавления файла ресурсов:

1. Выберите Файл > Новый файл или проект > Qt > Qt Resource File > Choose.
2. Откроется диалоговое окно “Выбор местоположения”.
3. В поле “Имя” введите textfinder.
4. В поле “Путь” введите C:\Qt\examples\TextFinderи нажмите кнопку “Далее”.
5. Откроется диалоговое окно “Управление проектами”.
6. В поле “Добавить в проект” выберите textfinder.pro и нажмите кнопку “Готово”, чтобы открыть файл в редакторе кода.
7. Выберите Добавить > Добавить Префикс.
8. В поле префикс замените префикс по умолчанию косой чертой (/).
9. Выберите Добавить > добавить файлы, чтобы найти и добавить входные данные из input.txt.

1.3. ЛАБОРАТОРНЫЕ ЗАДАНИЯ И МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ИХ ВЫПОЛНЕНИЮ

Создайте новый проект согласно алгоритму: новый проект → приложение QtWidgets → введение и размещение проекта → выбор комплекта → информация о классе →завершить проект. Выберите имя проекта и каталог для его размещения. Выберите комплекты.

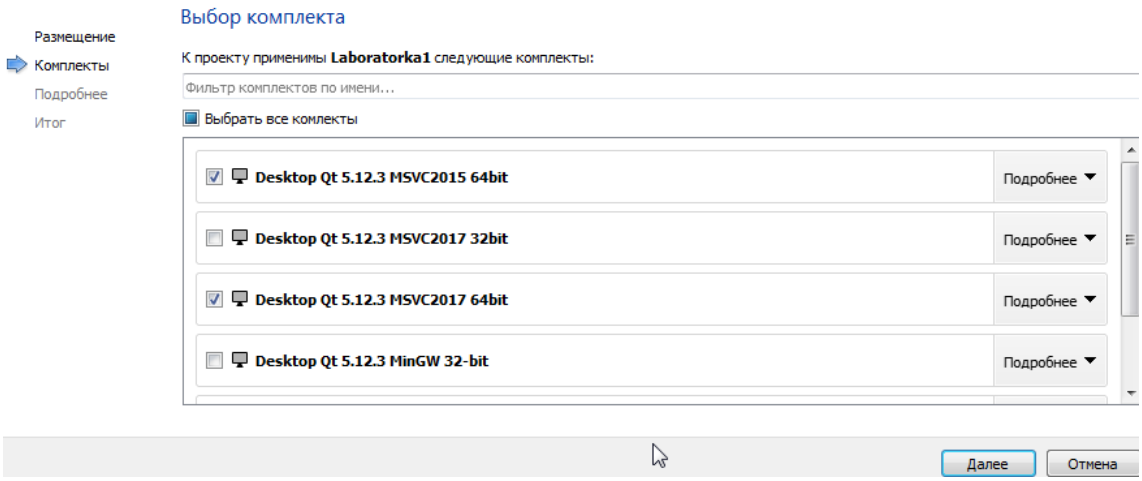


Рис. 12. Необходимые компоненты

Задайте базовый класс. Нажмите кнопку “Завершить”.

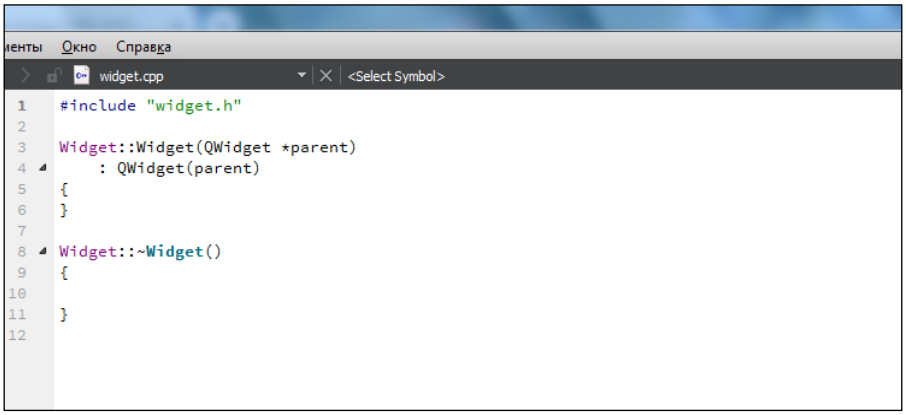


Рис. 13. Рабочее поле проекта

1.4. УКАЗАНИЯ ПО ОФОРМЛЕНИЮ ОТЧЕТА

Отчет должен содержать название и цель работы, краткие теоретические сведения, а также код с комментариями.

2. ЛАБОРАТОРНАЯ РАБОТА № 2
РАЗРАБОТКА ТИПОВЫХ ИНТЕРФЕЙСОВ С ПОМОЩЬЮ ФРЕЙМВОРКА QT

2.1. ОБЩИЕ УКАЗАНИЯ
2.1.1. ЦЕЛЬ РАБОТЫ

Целью лабораторной работы является разработка типового интерфейса.

2.1.2. ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Основным содержанием работы является разработка типовых интерфейсов с помощью фреймворка Qt. В ходе работы идёт ознакомление с приведённым примером приложения, производится анализ используемых технологий. На основе полученных знаний выполняется индивидуальное задание.

2.2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ
2.2.1. ВИДЖЕТЫ QT

Модуль виджетов Qt предоставляет набор элементов пользовательского интерфейса для создания классических пользовательских интерфейсов в стиле рабочего стола.

Для того, чтобы включить определения классов модуля, используйте следующую директиву:

```
#include <QtWidgets>
```

Чтобы установить связь с модулем, добавьте эту строку в свой .pro файл qmake:

```
Qt += widgets
```

Виджеты являются основными элементами для создания пользовательских интерфейсов в Qt. Виджеты могут отображать данные и информацию о состоянии, получать вводимые пользователем данные и предоставлять контейнер для других виджетов, которые должны быть сгруппированы вместе. Виджет, который не встроен в родительский виджет, называется окном.

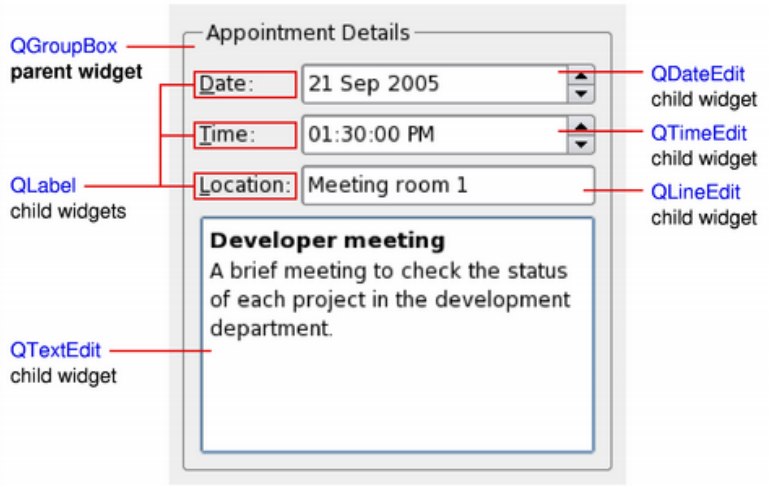


Рис. 14. Примеры виджетов пользовательском интерфейсе программы

Класс QWidget предоставляет базовую возможность визуализации на экране и обработки событий пользовательского ввода. Все элементы пользовательского интерфейса, предоставляемые Qt, являются либо подклассами QWidget, либо используются в связи с подклассом QWidget. Создание пользовательских виджетов выполняется путем создания подкласса QWidget или подходящего подкласса и переопределения виртуальных обработчиков событий.

Они могут быть следующих видов:

- Виджеты окон и диалоговых окон
- Главное окно приложения
- Диалоговое окно

Таблица 1

Основные классы виджетов

Класс	Назначение
QWidget	Базовый класс всех объектов пользовательского интерфейса
QCheckBox	Флажок с текстовой меткой
QComboBox	Комбинированная кнопка и всплывающий список
QCommandLinkButton	Кнопка связи команд стиля Vista
QDateEdit	Виджет для редактирования дат на основе виджета QDateTimeEdit
QDateTimeEdit	Виджет для редактирования дат и времени
QTimeEdit	Виджет для редактирования времени на основе виджета QDateTimeEdit
QDial	Округленная регулировка диапазона (например, спидометр или потенциометр)
QFocusFrame	Рамка фокусировки, которая может находиться за пределами обычной области рисования виджета
QFontComboBox	Combobox, который позволяет пользователю выбрать семейство шрифтов
QLabel	Дисплей текста или изображения

Окончание табл. 1

Класс	Назначение
QLCDNumber	Отображает число с ЖК-подобными цифрами
QLineEdit	Однострочный текстовый редактор
QMenu	Виджет меню для использования в строках меню, контекстных меню и других всплывающих меню
QProgressBar	Горизонтальный или вертикальный индикатор выполнения
QPushButton	Командная кнопка
QRadioButton	Переключатель с текстовой меткой
QScrollArea	Прокрутка вида на другой виджет
QScrollBar	Вертикальная или горизонтальная полоса прокрутки
QSizeGrip	Ручка изменения размера для изменения размера окон верхнего уровня
QSlider	Вертикальный или горизонтальный слайдер
QDoubleSpinBox	Spin box виджет для значений с плавающей запятой
QSpinBox	Виджет Spin box
QTabBar	Панель вкладок, например, для использования в диалогах с вкладками
QTabWidget	Стек виджетов с вкладками
QToolBox	Столбец элементов виджета с вкладками
QToolButton	Кнопка быстрого доступа к командам или параметрам, обычно используемым внутри QToolBar

Расширенные виджеты GUI (табл. 2), например вкладки виджетов и прогресс-баров, обеспечивают более сложные элементы управления пользовательского интерфейса.

Таблица 2

Расширенные виджеты

Класс	Назначение
QColumnView	Модель / представление реализация представления столбца
QDataWidgetMapper	Сопоставление раздела модели данных с виджетами
QListView	Представление списка или значка для модели
QTableView	Модель/представление по умолчанию реализация табличного представления
QTreeView	Модель/представление по умолчанию реализация представления в виде дерева
QUndoView	Отображает содержимое пакета QUndoStack
QCalendarWidget	Ежемесячный виджет календаря, позволяющий пользователю выбрать дату
QMacCocoaViewContainer	Виджет для macOS, который можно использовать для оберты-вания произвольных представлений Cocoa (т. е. подклассов NSView) и вставки их в иерархии Qt
QMacNativeWidget	Виджет для macOS, который предоставляет способ поместить виджеты Qt в иерархии Cocoa

Абстрактные классы виджетов (табл. 3) являются базовыми классами. Они не используются в качестве автономных классов, но обеспечивают функциональность, когда они являются подклассами.

Таблица 3

Абстрактные классы виджетов

Класс	Назначение
QAbstractButton	Абстрактный базовый класс виджетов кнопок, предоставляющий функции, общие для кнопок
QAbstractScrollArea	Область прокрутки с полосами прокрутки по требованию
QAbstractSlider	Целочисленное значение в пределах диапазона
QAbstractSpinBox	Spinbox и линия редактируют для отображения значений
QFrame	Базовый класс виджетов, которые могут иметь рамку
QDialog	Базовый класс диалоговых окон

Классы, такие как сплиттеры, панели вкладок, группы кнопок и т. д., используются для организации и группировки примитивов GUI в более сложные приложения и диалоговые окна. Они называются виджетами органайзера (табл. 4).

Таблица 4

Классы виджетов органайзера

Класс	Назначение
QButtonGroup	Контейнер для организации групп виджетов кнопок
QGroupBox	Рамка группового ящика с заголовком
QSplitter	Реализует виджет splitter
QSplitterHandle	Функциональность дескриптора для разделителя
QStackedWidget	Стек виджетов, где одновременно виден только один виджет
QTabWidget	Стек виджетов с вкладками

Далее приведем классы виджетов модели / представления (табл. 5).

Таблица 5

Классы виджетов модели / представления

Класс	Назначение
QAbstractItemModel	Абстрактный интерфейс для классов моделей элементов
QAbstractListModel	Абстрактная модель, которая может быть разделена на подклассы для создания одномерных моделей списков
QAbstractTableModel	Абстрактная модель, которая может быть разделена на подклассы для создания табличных моделей
QModelIndex	Используется для поиска данных в модели данных
QPersistentModelIndex	Используется для поиска данных в модели данных
QAbstractProxyModel	Базовый класс для моделей прокси-элементов, которые могут выполнять сортировку, фильтрацию или другие задачи обработки данных

Продолжение табл. 5

Класс	Назначение
QConcatenateTablesProxyModel	Прокси нескольких исходных моделей, объединяющих их строки
QIdentityProxyModel	Прокси его исходная модель немодифицированная
QItemSelection	Управление информацией о выбранных элементах в модели
QItemSelectionModel	Отслеживает выбранные элементы представления
QItemSelectionRange	Управляет информацией о диапазоне выбранных элементов в модели
QSortFilterProxyModel	Поддержка сортировки и фильтрации данных, передаваемых между другой моделью и видом
QStringListModel	Модель, предоставляющая строки для представлений
QStandardItem	Элемент для использования с классом QStandardItemModel
QStandardItemModel	Универсальная модель для хранения пользовательских данных
QAbstractItemDelegate	Используется для отображения и редактирования элементов данных из модели
QAbstractItemView	Основные функциональные возможности для классов представления элементов
QColumnView	Модель / представление реализация представления столбца
QDataWidgetMapper	Сопоставление раздела модели данных с виджетами
QHeaderView	Строка заголовка или столбец заголовка для представлений элементов
QItemDelegate	Средства отображения и редактирования элементов данных из модели
QItemEditorCreator	Позволяет создавать редактор элементов creator bases без подклассов QItemEditorCreatorBase
QItemEditorCreatorBase	Абстрактный базовый класс, который должен быть разделен на подклассы при реализации нового редактора элементов creators
QItemEditorFactory	Виджеты для редактирования данных элементов в представлениях и делегатах
QStandardItemEditorCreator	Возможность регистрации виджетов без необходимости подкласса QItemEditorCreatorBase
QListView	Представление списка или значка для модели
QListWidget	Виджет списка на основе элементов
QListWidgetItem	Элемент для использования с классом представления элемента QListWidget
QStyledItemDelegate	Средства отображения и редактирования элементов данных из модели
QTableView	Модель/представление по умолчанию реализация табличного представления
QTableWidget	Табличное представление на основе элементов с моделью по умолчанию
QTableWidgetItem	Элемент для использования с классом QTableWidget

Окончание табл. 5

Класс	Назначение
QTableWidgetSelectionRange	Способ взаимодействия с выделением в модели без использования индексов модели и модели выделения
QTreeView	Модель/представление по умолчанию реализация представления в виде дерева
QTreeWidget	Древовидное представление, использующее предопределенную модель дерева
QTreeWidgetItem	Предмет для использования с классом удобства QTreeWidget
QTreeWidgetItemIterator	Способ итерации по элементам в экземпляре QTreeWidget
QFileSystemModel	Модель данных для локальной файловой системы

Теперь приведем список классов виджетов главного окна и связанных классов (табл. 6).

Таблица 6

Классы виджетов главного окна и связанных классов

Класс	Название
QAction	Абстрактное действие пользовательского интерфейса, которое можно вставить в виджеты
QActionGroup	Группирует действия вместе
QWidgetAction	Расширяет QAction с помощью интерфейса для вставки пользовательских виджетов в контейнеры на основе действий, такие как панели инструментов
QDockWidget	Виджет, который может быть закреплен внутри QMainWindow или плавал как окно верхнего уровня на рабочем столе
QMainWindow	Главное окно приложения
QMdiArea	Область, в которой отображаются окна MDI
QMdiSubWindow	Класс подокна для QMdiArea
QMenu	Виджет меню для использования в строках меню, контекстных меню и других всплывающих меню
QMenuBar	Горизонтальная строка меню
QSizeGrip	Ручка изменения размера для изменения размера окон верхнего уровня
QStatusBar	Горизонтальная полоса подходит для представления информации о состоянии
QToolBar	Подвижная панель, содержащая набор элементов управления

Внешний вид виджета и связанные со стилем классы

Далее приведем список классов, используемых для настройки внешнего вида и стиля пользовательского интерфейса (табл. 7).

Таблица 7

Классы виджетов внешнего вида

Класс	Назначение
QCursor	Курсор мыши произвольной формы
QPalette	Содержит группы цветов для каждого состояния виджета
QColor	Цвета на основе значений RGB, HSV или CMYK
QFont	Задает запрос для шрифта, используемого для рисования текста
QFontDatabase	Информация о шрифтах, доступных в базовой оконной системе
QFontInfo	Общая информация о шрифтах
QGraphicsAnchor	Представляет якорь между двумя элементами в QGraphicsAnchorLayout
QGraphicsAnchorLayout	Макет, в котором можно закрепить виджеты вместе в виде графики
QCommonStyle	Инкапсулирует общий внешний вид графического интерфейса пользователя
QStyle	Абстрактный базовый класс, который инкапсулирует внешний вид графического интерфейса пользователя
QStyleFactory	Создает объекты QStyle
QStyleHintReturn	Подсказки стилей, возвращающие больше, чем основные типы данных
QStyleHintReturnMask	Подсказки стиля, возвращающие QRegion
QStyleHintReturnVariant	Подсказки стиля, которые возвращают QVariant
QStyleOption	Сохраняет параметры, используемые функциями QStyle
QStylePainter	Класс удобства для рисования элементов QStyle внутри виджета

Также приведем список виджетов классов компоновки (табл. 8).

Таблица 8

Классы виджетов компоновки

Класс	Назначение
QGraphicsAnchor	Представляет якорь между двумя элементами в QGraphicsAnchorLayout
QGraphicsAnchorLayout	Макет, в котором можно закрепить виджеты вместе в виде графики
QBoxLayout	Выравнивание дочерних виджетов по горизонтали или вертикали
QHBoxLayout	Выравнивание виджетов по горизонтали
QVBoxLayout	Выравнивание виджетов по вертикали
QFormLayout	Управляет формами входных виджетов и связанными с ними метками
QGridLayout	Выкладывает виджеты в сетку
QLayout	Базовый класс менеджеров геометрии
QLayoutItem	Абстрактный элемент, которым управляет QLayout
QSpacerItem	Пустое место в компоновке
QWidgetItem	Элемент макета, представляющий виджет
QSizePolicy	Атрибут макета, описывающий политику горизонтального и вертикального изменения размеров
QStackedLayout	Стек виджетов, где одновременно виден только один виджет

Окончание табл. 8

Класс	Назначение
QButtonGroup	Контейнер для организации групп виджетов кнопок
QGroupBox	Рамка группового ящика с заголовком
QStackedWidget	Стек виджетов, где одновременно виден только один виджет

Qt включает в себя набор классов управления компоновкой, которые используются для описания того, как виджеты размещаются в пользовательском интерфейсе приложения. Эти макеты автоматически позиционируют и изменяют размер виджетов, когда объем доступного для них пространства изменяется, гарантируя, что они последовательно расположены и что пользовательский интерфейс в целом остается пригодным для использования.

Все подклассы QWidget могут использовать макеты для управления своими дочерними элементами. Функция QWidget::setLayout () применяет макет к виджету. Когда макет задается на виджете таким образом, он выполняет следующие задачи:

- Позиционирование дочерних виджетов
- Разумные размеры по умолчанию для windows
- Разумные минимальные размеры для окон
- Изменение размера обработки
- Автоматическое обновление при изменении содержимого:
- Размер шрифта, текст или другое содержимое дочерних виджетов
- Скрытие или отображение дочернего виджета
- Удаление дочерних виджетов

Самый простой способ дать виджетам хороший макет – это использовать встроенные менеджеры макетов: QHBoxLayout, QVBoxLayout, QGridLayout и QFormLayout. Эти классы наследуются от QLayout, который, в свою очередь, происходит от QObject (не QWidget). Они заботятся об управлении геометрией для набора виджетов. Чтобы создать более сложные макеты, вы можете вложить менеджеры макетов друг в друга.

Рассмотрим работу менеджеров пакетов подробнее. QHBoxLayout раскладывает виджеты в горизонтальной строке слева направо (или справа налево для языков справа налево) (рис. 14).

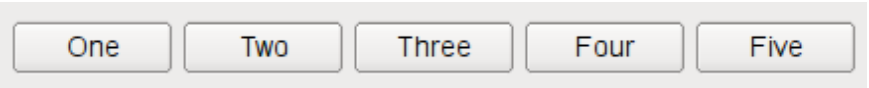


Рис. 15. Пример работы QHBoxLayout

QVBoxLayout раскладывает виджеты в вертикальном столбце, сверху вниз (рис. 16).

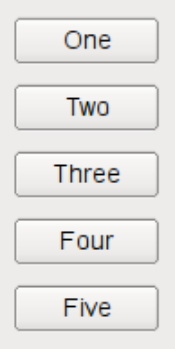


Рис. 16. Пример работы QVBoxLayout

QGridLayout раскладывает виджеты в двумерную сетку. Виджеты могут занимать несколько ячеек (рис. 17).

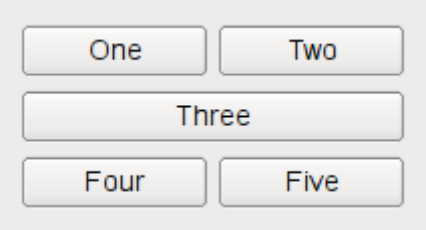


Рис. 17. Пример работы QGridLayout

QFormLayout раскладывает виджеты в 2-х столбцовом описательном стиле метка – поле (рис. 18).

Рис. 18. Пример работы QFormLayout

Для управления основными окнами и связанными компонентами пользовательского интерфейса Qt предоставляет следующие классы:

- QMainWindow – это центральный класс, вокруг которого можно создавать приложения. Наряду с сопутствующими классами QDockWidget и QToolBar, он представляет собой пользовательский интерфейс верхнего уровня приложения.
- QDockWidget предоставляет виджет, который может быть использован для создания съемных инструментальных палитр или вспомогательных окон. Виджеты док-станции отслеживают свои собственные свойства, и они могут быть перемещены, закрыты и перемещены как внешние окна.
- QToolBar предоставляет общий виджет панели инструментов, который может содержать несколько различных виджетов, связанных с действием, таких как кнопки, раскрывающиеся меню, комбо-боксы

и вращающиеся окна. Акцент на единую модель действий в Qt означает, что панели инструментов хорошо взаимодействуют с меню и сочетаниями клавиш.

QWidget предоставляет несколько функций, которые имеют дело с геометрией виджета. Некоторые из этих функций работают на чистой клиентской области (т. е. окно, исключающее оконную рамку), другие включают оконную рамку. Дифференциация осуществляется таким образом, чтобы наиболее распространенное использование было прозрачным.

- Включая оконную раму: x(), y(), frameGeometry(), pos () и move ().
- Исключая оконную раму: геометрия (), ширина(), высота(), прямая () и размер ().

Обратите внимание, что различие имеет значение только для украшенных виджетов верхнего уровня. Для всех дочерних виджетов геометрия рамки равна геометрии клиента виджета.

На диаграмме показано большинство используемых функций (рис. 19):

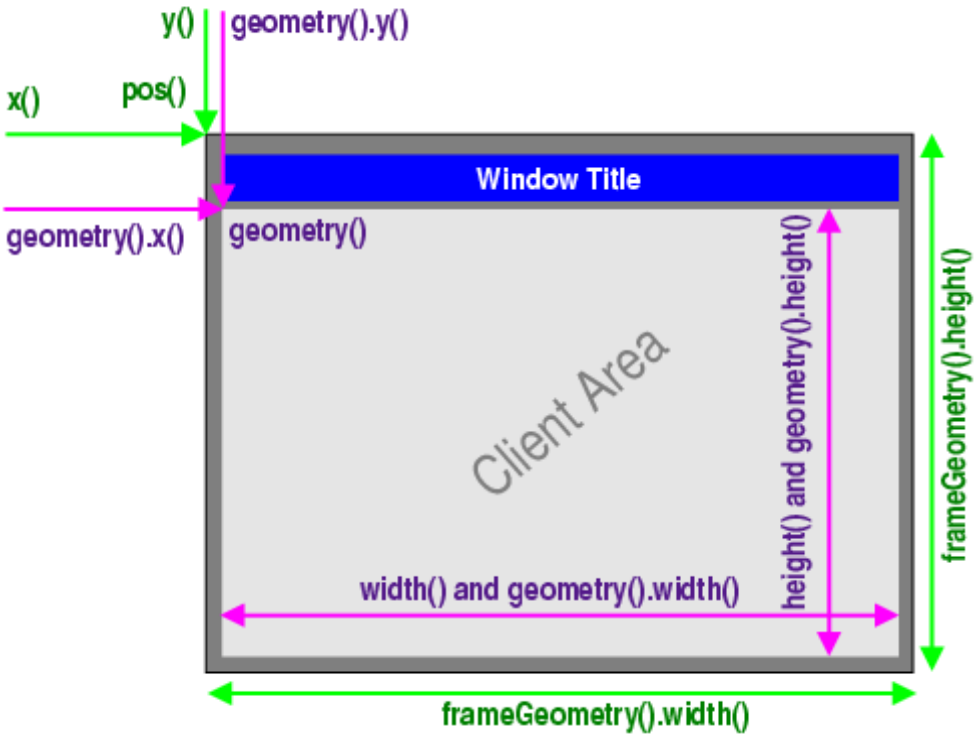


Рис. 19. Влияние функций на размеры окна

2.3 ЛАБОРАТОРНЫЕ ЗАДАНИЯ И МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ИХ ВЫПОЛНЕНИЮ

1. Подключить библиотеку QtWidgets (рис. 20, строка №5)

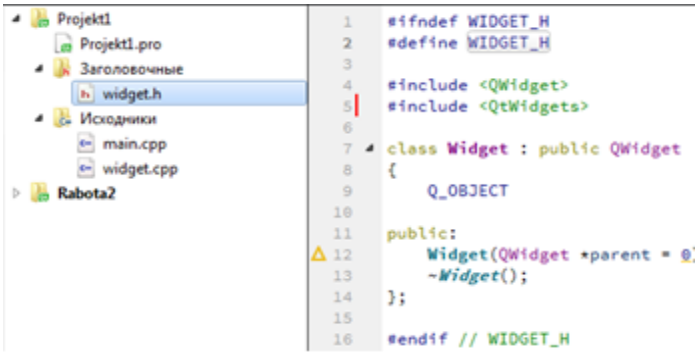


Рис. 20. Файл widget.h

2. В файле widget.cpp в конструкторе задать имя окна с помощью метода setWindowTitle.
3. Создать объект класса QLabel, затем при помощи функции setText создать надпись.
4. Добавить объект для ввода текса, класса QLineEdit.
5. Добавить две вертикальные разметки QVBoxLayout.
6. С помощью addWidget добавить в одну разметку надписи, а в другую разметку строки для ввода.

```
#include "widget.h"

Widget::Widget(QWidget *parent)
    : QWidget(parent)
{
    this->setWindowTitle("Анкета");

    QLabel * lbl1 = new QLabel;
    lbl1->setText("ФИО");
    QLabel * lbl2 = new QLabel("Дата рождения")
    QLabel * lbl3 = new QLabel("Адрес");
    QLabel * lbl4 = new QLabel("E-mail");
    QLabel * lbl5 = new QLabel("Телефон");
    QLabel * lbl6 = new QLabel("Пол");

    QLineEdit * fio = new QLineEdit;
    QLineEdit * data = new QLineEdit;
    QLineEdit * adress = new QLineEdit;
    QLineEdit * email = new QLineEdit;
    QLineEdit * tel = new QLineEdit;
    QComboBox * pol = new QComboBox;
    QStringList lst;
    lst << "М" << "Ж";
    pol->addItems(lst);

    QVBoxLayout * vlay1 = new QVBoxLayout;
    vlay1->addWidget(lbl1);
    vlay1->addWidget(lbl2);
    vlay1->addWidget(lbl3);
    vlay1->addWidget(lbl4);
    vlay1->addWidget(lbl5);
    vlay1->addWidget(lbl6);


    QVBoxLayout * vlay2 = new QVBoxLayout;
    vlay2->addWidget(fio);
    vlay2->addWidget(data);
    vlay2->addWidget(adress);
```

Рис. 21. Файл widget.cpp

7. Создать объекты, они же кнопки, класса QPushButton, в которые добавить текст.
8. Создать горизонтальную разметку QHBoxLayout в которую разместить эти кнопки.
9. В другую горизонтальную разметку вставить две вертикальные разметки с помощью метода addLayout.

10.Создать отдельно еще одну вертикальную(QVBoxLayout) разметку.
11.Вставить в нее две горизонтальных(QHBoxLayout).
12.Применить последнюю разметку this->setLayout.

```
QVBoxLayout * vlay2 = new QVBoxLayout;
vlay2->addWidget(fio);
vlay2->addWidget(data);
vlay2->addWidget(adress);
vlay2->addWidget(email);
vlay2->addWidget(tel);
vlay2->addWidget(pol);


QHBoxLayout * hlay1 = new QHBoxLayout;
hlay1->addLayout(vlay1);
hlay1->addLayout(vlay2);


QPushButton * ok = new QPushButton("Ok");
QPushButton * cancel = new QPushButton("Cancel");

QHBoxLayout * hlay2 = new QHBoxLayout;
hlay2->addWidget(ok);
hlay2->addWidget(cancel);


QVBoxLayout * vlay3 = new QVBoxLayout;
vlay3->addLayout(hlay1);
vlay3->addLayout(hlay2);


this->setLayout(vlay3);
```

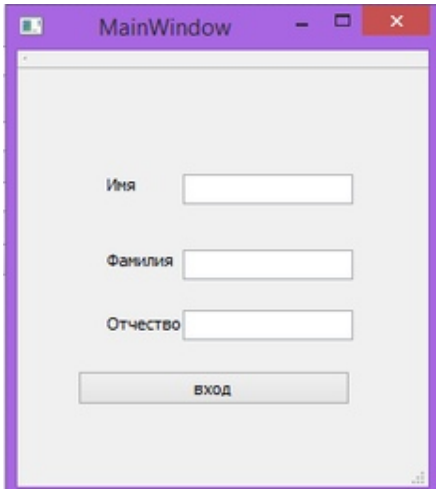
Рис. 22. Файл widget.cpp(продолжение)

2.4. ВАРИАНТЫ ЛАБОРАТОРНЫХ ЗАДАНИЙ

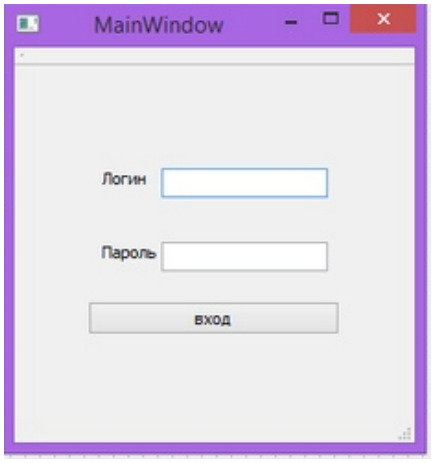
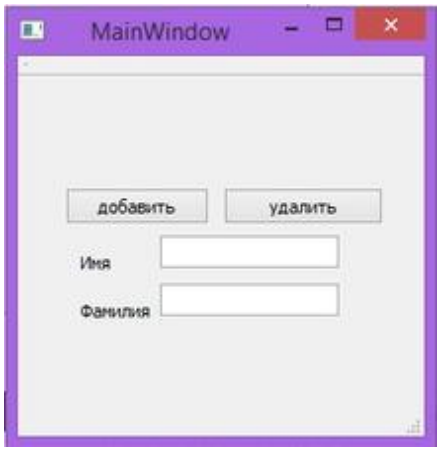
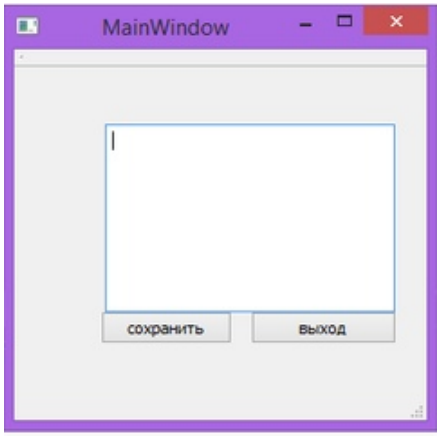
Необходимо создать программу с пользовательским интерфейсом, соответствующим варианту задания. Варианты лабораторных заданий приведены в таблице 9.

Таблица 9

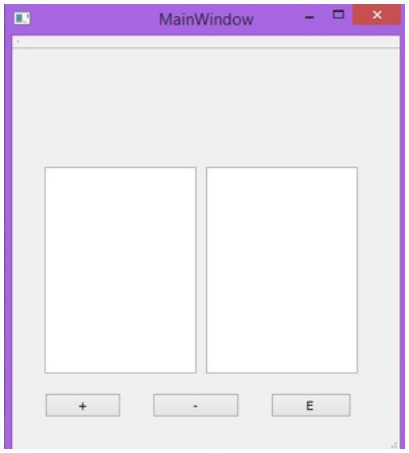
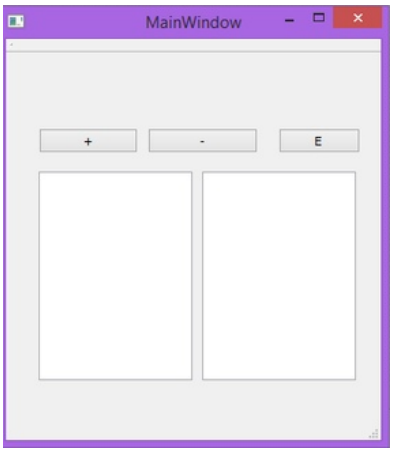
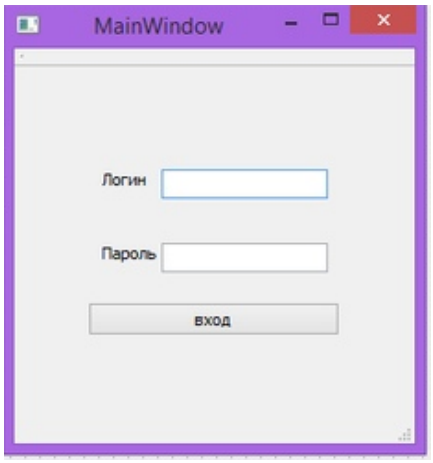
Варианты заданий

Вариант№1	

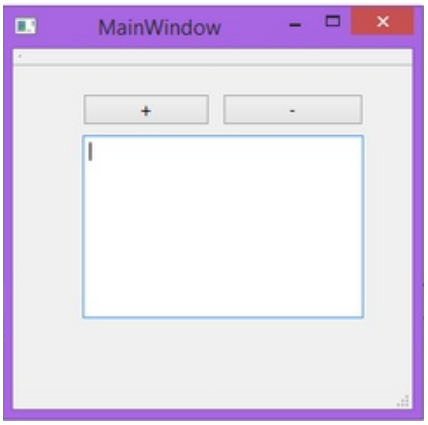
Продолжение табл. 9

Вариант№2	
Вариант№3	
Вариант№4	

Продолжение табл. 9

Вариант№5	
Вариант№6	
Вариант№7	

Окончание табл. 9

Вариант№8	

2.5. УКАЗАНИЯ ПО ОФОРМЛЕНИЮ ОТЧЕТА

Отчет должен содержать название и цель работы, краткие теоретические сведения, а также код с комментариями.

3. ЛАБОРАТОРНАЯ РАБОТА №3 МЕЖОКОННЫЕ ВЗАИМОДЕЙСТВИЯ

3.1. ОБЩИЕ УКАЗАНИЯ

3.1.1. ЦЕЛЬ РАБОТЫ

Целью работы является знакомство с особенностями межклассовых взаимодействий на примере фреймворкаQt.

3.1.2. ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Сначала рассмотрим работу класса QDialog. Он является базовым классом диалоговых окон. Диалоговое окно – это окно верхнего уровня, в основном используемое для краткосрочных задач и краткого общения с пользователем. QDialogs может быть модальным или немодальным. QDialogs может предоставить возвращаемое значение , и они могут иметь кнопки по умолчанию. QDialogs также может иметь QSizeGrip в правом нижнем углу, используя setSizeGripEnabled().

Обратите внимание, что QDialog (и любой другой виджет, имеющий тип Qt::Dialog) использует родительский виджет немного иначе, чем другие классы в Qt. Диалог всегда является виджетом верхнего уровня, но если у него есть родитель, его расположение по умолчанию центрируется поверх виджета верхнего уровня родителя (если это не сам верхний уровень). Он также поделится записью родительской панели задач.

Используйте перегрузку функции QWidget::setParent(), чтобы изменить владельца виджета QDialog. Эта функция позволяет вам явно установить флажки окна перекрашенного виджета; использование перегруженной функции очи-

стит флаги окна, определяющие свойства оконной системы для виджета (в частности, сбросит флаг Qt::Dialog).

Примечание. Родительское отношение диалогового окна *не* означает, что диалоговое окно всегда будет располагаться поверх родительского окна. Чтобы диалог всегда был сверху, сделайте его модальным. Это также относится к дочерним окнам самого диалога. Чтобы дочерние окна диалогового окна оставались поверх него, сделайте также модальные дочерние окна.

Теперь рассмотрим два типа диалоговых окон: модальное и немодальное.

Модальное диалоговое окно представляет собой диалог, который блокирует ввод в других видимых окон в одном приложении. Диалоги, которые используются для запроса имени файла у пользователя или которые используются для установки настроек приложения, обычно являются модальными. Диалоги могут быть модальными приложения (по умолчанию) или модальными окнами. Когда модальное диалоговое окно приложения открыто, пользователь должен закончить взаимодействие с диалоговым окном и закрыть его, прежде чем он сможет получить доступ к любому другому окну в приложении. Модальные диалоговые окна только блокируют доступ к окну, связанному с диалоговым окном, позволяя пользователю продолжать использовать другие окна в приложении. Самый распространенный способ отображения модального диалога - вызвать его функцию exec (). Когда пользователь закрывает диалоговое окно, exec() предоставит полезное возвращаемое значение . Чтобы закрыть диалоговое окно и вернуть соответствующее значение, необходимо подключить кнопку по умолчанию, например кнопку **ОК** , к гнезду accept() и кнопку **Отмена** к гнезду reject(). Кроме того, вы можете вызвать слот done() с помощью Acceptedили Rejected. Альтернативой является вызов setModal(true) или setWindowModality(), а затем show(). В отличие от exec(), show() немедленно возвращает управление вызывающей стороне. Вызов setModal(true) особенно полезен для диалогов прогресса, где пользователь должен иметь возможность взаимодействовать с диалогом, например, для отмены длительной операции. Если вы используете show() и setModal(true) вместе для выполнения длинной операции, вы должны периодически вызывать QCoreApplication::processEvents() во время обработки, чтобы позволить пользователю взаимодействовать с диалоговым окном. (См. QProgressDialog.)

Немодальным диалог представляет собой диалог, который работает независимо от других окон в одном приложении. Диалоги поиска и замены в текстовых процессорах часто не работают, чтобы позволить пользователю взаимодействовать как с главным окном приложения, так и с диалогом. Немодальные диалоги отображаются с использованием show(), которое немедленно возвращает управление вызывающей стороне. Если вы вызываете функцию show() после скрытия диалога, диалог будет отображаться в исходном положении. Это связано с тем, что диспетчер окон определяет положение окон, которые не были явно размещены программистом. Чтобы сохранить положение диалогового окна, которое было перемещено пользователем, сохраните его положение в об-

работчике closeEvent(), а затем переместите диалоговое окно в это положение, прежде чем снова его показывать.

Кнопка диалога по *умолчанию* – это кнопка, которая нажимается, когда пользователь нажимает Enter (Return). Эта кнопка используется для обозначения того, что пользователь принимает настройки диалога и хочет закрыть диалог. Используйте QPushButton::setDefault(), QPushButton::isDefault() и QPushButton::autoDefault() для установки и управления кнопкой по умолчанию для диалогового окна.

Если пользователь нажимает клавишу Esc в диалоговом окне, вызывается QDialog::reject(). Это приведет к закрытию окна: событие закрытия нельзя игнорировать .

Расширяемость – это возможность показать диалог двумя способами: частичный диалог, который показывает наиболее часто используемые параметры, и полный диалог, который показывает все параметры. Обычно расширяемый диалог первоначально отображается как частичный диалог, но с кнопкой переключения «**Больше»**. Если пользователь нажимает кнопку «**Дополнительно»**, диалоговое окно расширяется. Пример Расширение показывает, как достичь расширяемых диалогов с использованием Qt.

Модальные диалоги часто используются в ситуациях, когда требуется возвращаемое значение, например, чтобы указать, нажал ли пользователь **OK** или **Отмена** . Диалог можно закрыть, вызвав слоты accept() или reject(), и exec() вернется Acceptedили Rejected при необходимости. Exec() вызов возвращает результат диалога. Результат также доступен из result(), если диалог не был уничтожен.

Чтобы изменить поведение вашего диалогового окна, вы можете переопределить функции accept(), reject() или done(). Функция closeEvent() должна быть переопределена только для сохранения позиции диалога или для переопределения стандартного поведения закрытия или отклонения.

Далее приведем примеры кода.

Модальный диалог:

```
void EditorWindow::countWords()
{
    WordCountDialog dialog(this);
    dialog.setWordCount(document().wordCount());
    dialog.exec();
}
```

Немодальный диалог:

```
void EditorWindow::find()
{
    if (!findDialog) {
```

```
            findDialog = new FindDialog(this);
            connect(findDialog, &FindDialog::findNext,
            this, &EditorWindow::findNext);
        }

        findDialog->show();
        findDialog->raise();
        findDialog->activateWindow();
}

enum QDialog::DialogCode
```

Значения, возвращаемое модальным диалогом.

Таблица 10

Значения постоянных при модальном диалоге

Постоянная	Значение
QDialog::Accepted	1
QDialog::Rejected	0

3.1.3. ОПИСАНИЕ РАБОТЫ С МОДАЛЬНЫМ ДИАЛОГОМ

модальный : bool

Это свойство указывает, должно ли show () выводить диалоговое окно как модальное или немодальное.

По умолчанию это свойство false и show () выдает диалоговое окно как немодальное. Установка этого свойства в true эквивалентна установке QWidget :: windowModality в Qt :: ApplicationModal.

Функция exec () игнорирует значение этого свойства и всегда выдает диалог как модальный. Это свойство определяет, включена ли ручка размера QSizeGrip помещается в правом нижнем углу диалогового окна , когда это свойство включено. По умолчанию размер захвата отключен.

QDialog :: QDialog (QWidget * *parent* = nullptr, Qt :: WindowFlags *f* = Qt :: WindowFlags ()) Создает диалог с родительским *родителем*.

Диалог всегда является виджетом верхнего уровня, но, если у него есть родительский элемент, его расположение по умолчанию центрируется поверх родительского элемента. Он также поделится записью родительской панели задач.

Флаги виджета *f* передаются конструктору QWidget. Если, например, вам не нужна кнопка «Что это» в строке заголовка диалога, передайте
Qt :: WindowTitleHint | Qt :: WindowSystemMenuHint в *f* .

Скрывает модальное диалоговое окно и устанавливает код результата в Accepted.

Этот сигнал испускается, когда диалоговое окно было принято либо пользователем, либо путем вызова accept() или done() с аргументом QDialog :: Accepted.

Обратите внимание, что этот сигнал *не* генерируется при скрытии диалога с помощью hide () или setVisible (false). Это включает в себя удаление диалога, пока он виден.

Закрывает диалог и устанавливает его код результата на *r*. Закончил сигнал () будет излучать *г* ; если *г* является QDialog :: Принято или QDialog :: Отклонено, то приняты () или отклоненные сигналы () также будет излучаться, соответственно.

Если это диалоговое окно показано с помощью exec (), done () также приводит к завершению локального цикла обработки событий, а exec () возвращает *r*.

Как и в случае с QWidget :: close (), done () удаляет диалог, если установлен флаг Qt :: WA_DeleteOnClose . Если диалог является основным виджетом приложения, приложение закрывается. Если диалог является последним закрытым окном, посылается сигнал QApplication :: lastWindowClosed().

Показывает диалог как модальный диалог , блокируя, пока пользователь не закроет его. Функция возвращает результат DialogCode.

Если диалоговое окно является модальным приложения , пользователи не могут взаимодействовать с любым другим окном в том же приложении, пока они не закроют диалоговое окно. Если диалоговое окно является модальным окном , то только взаимодействие с родительским окном блокируется, пока диалоговое окно открыто. По умолчанию диалог является модальным для приложения.

Примечание: избегайте использования этой функции; вместо этого используйте open(). В отличие от exec (), open () является асинхронным и не вращает дополнительный цикл событий. Это предотвращает ряд опасных ошибок (например, удаление родителя диалога, когда диалог открыт через exec ()). При использовании открытой () , вы можете подключиться к готовой () сигнала QDialog получать уведомления , когда диалоговое окно закрывается.

Этот сигнал испускается, когда код результата диалога был задан пользователем или вызовом done (), accept () или reject ().

Обратите внимание, что этот сигнал не генерируется при скрытии диалога с помощью hide () или setVisible (false). Это включает в себя удаление диалога, пока он виден.

Показывает диалог как модальное диалоговое окно, возвращаясь немедленно.

Скрывает модальное диалоговое окно и устанавливает код результата в Rejected.

Этот сигнал испускается, когда диалоговое окно было отклонено пользователем или путем вызова reject () или done () с аргументом QDialog :: Rejected .

Обратите внимание, что этот сигнал *не* генерируется при скрытии диалога с помощью hide () или setVisible (false). Это включает в себя удаление диалога, пока он виден.

3.2. ПУБЛИЧНЫЕ ТИПЫ

Задается следующим образом: Enum DialogCode{Accepted, Rejected}.

И имеет свойства:

modal:bool

sizeGripEnabled:bool

3.2.1. ПУБЛИЧНЫЕ ФУНКЦИИ

Список публичных функций представлен в табл. 11.

Таблица 11

Список публичных функций

QDialog(QWidget **parent* = nullptr, Qt::WindowFlags *f* = Qt::WindowFlags())	
virtual	**~QDialog**()
bool	**isSizeGripEnabled**() const
int	**result**() const
void	**setModal**(bool *modal*)
void	**setResult**(int *i*)
void	**setSizeGripEnabled**(*bool*)

3.2.2 ПОВТОРНО РЕАЛИЗОВАННЫЕ ПУБЛИЧНЫЕ ФУНКЦИИ

Далее представлены повторно реализованные публичные функции (табл. 12).

Таблица 12

Список повторно реализованных публичных функций

virtual QSize	**minimumSizeHint**() const override
virtual void	**setVisible**(bool *visible*) override
virtual QSize	**sizeHint**() const override

3.2.3 ПУБЛИЧНЫЕ СЛОТЫ

В табл. 13 приведен список публичных слотов.

Таблица 13

Список публичных слотов

virtual void	**accept**()
virtual void	**done**(int *r*)
virtual int	**exec**()

virtual void	**open**()
virtual void	**reject**()

3.2.4 СИГНАЛЫ

Список сигналов приведен в табл. 14

Таблица 14

Список сигналов

void	**accepted**()
void	**finished**(int *result*)
void	**rejected**()

3.2.5 ПОВТОРНО РЕАЛИЗОВАННЫЕ ЗАЩИЩЕННЫЕ ФУНКЦИИ

Повторно реализованные защищенные функции приведены в табл. 15.

Таблица 15

Повторно реализованные защищенные функции

virtual void	**closeEvent**(QCloseEvent *e) override
virtual void	**contextMenuEvent**(QContextMenuEvent *e) override
virtual bool	**eventFilter**(QObject *o, QEvent *e) override
virtual void	**keyPressEvent**(QKeyEvent *e) override
virtual void	**resizeEvent**(*QResizeEvent* *) override
virtual void	**showEvent**(QShowEvent *event) override

3.3 ЛАБОРАТОРНЫЕ ЗАДАНИЯ И МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ИХ ВЫПОЛНЕНИЮ

1. В созданном ранее проекте создать новый класс TreeObj.
2. В заголовочном файле проекта treeobj.h объявить закрытые переменные QLineEdit * lin1; QLineEdit * lin2; QLineEdit * lin3; QComboBox * qcb; - строки фамилии, имени, отчества и выпадающего списка.
3. Объявить открытые методы setFam, setName, setOtche – сеттеры; getFam, getName, getOtche – геттеры.

4. Объявить конструктор от аргумента типа QString для релизации клавиши редактирования.
5. Перегрузить конструктор класса (создать конструкторы, содержащие дополнительные аргументы типов int и char).
   Пример выполняемого кода приведен на рис. 23.

```
#ifndef TREEOBJ_H
#define TREEOBJ_H

#include <QDialog>
#include <QtWidgets>

class TreeObj : public QDialog
{
    Q_OBJECT
public:
    explicit TreeObj(QWidget *parent = nullptr);
    TreeObj(QString sunameIn, QString nameIn, QString lastNameIn, QWidget *parent=0);
    TreeObj(bool perConst, QWidget *parent=0);
    void setFam(QString str1);
    QString getFam();
    void setName(QString str2);
    QString getName();
    void setOtche(QString str3);
    QString getOtche();
    bool getStatus();
private:
    QLineEdit * lin1;
    QLineEdit * lin2;
    QLineEdit * lin3;
    QComboBox * qcb;
signals:

public slots:

};

#endif // TREEOBJ_H
```

Рис. 23. Пример кода для пунктов 1-5

6. Реализовать сеттеры и геттеры в исходном файле widget.cpp, а также метод getStatus, через метод text() для закрытых переменных класса QLineEdit и метод currentText() для закрытой переменной класса QComboBox. Перегрузить конструктор
   На рис. 24 приведен пример выполняемого кода.

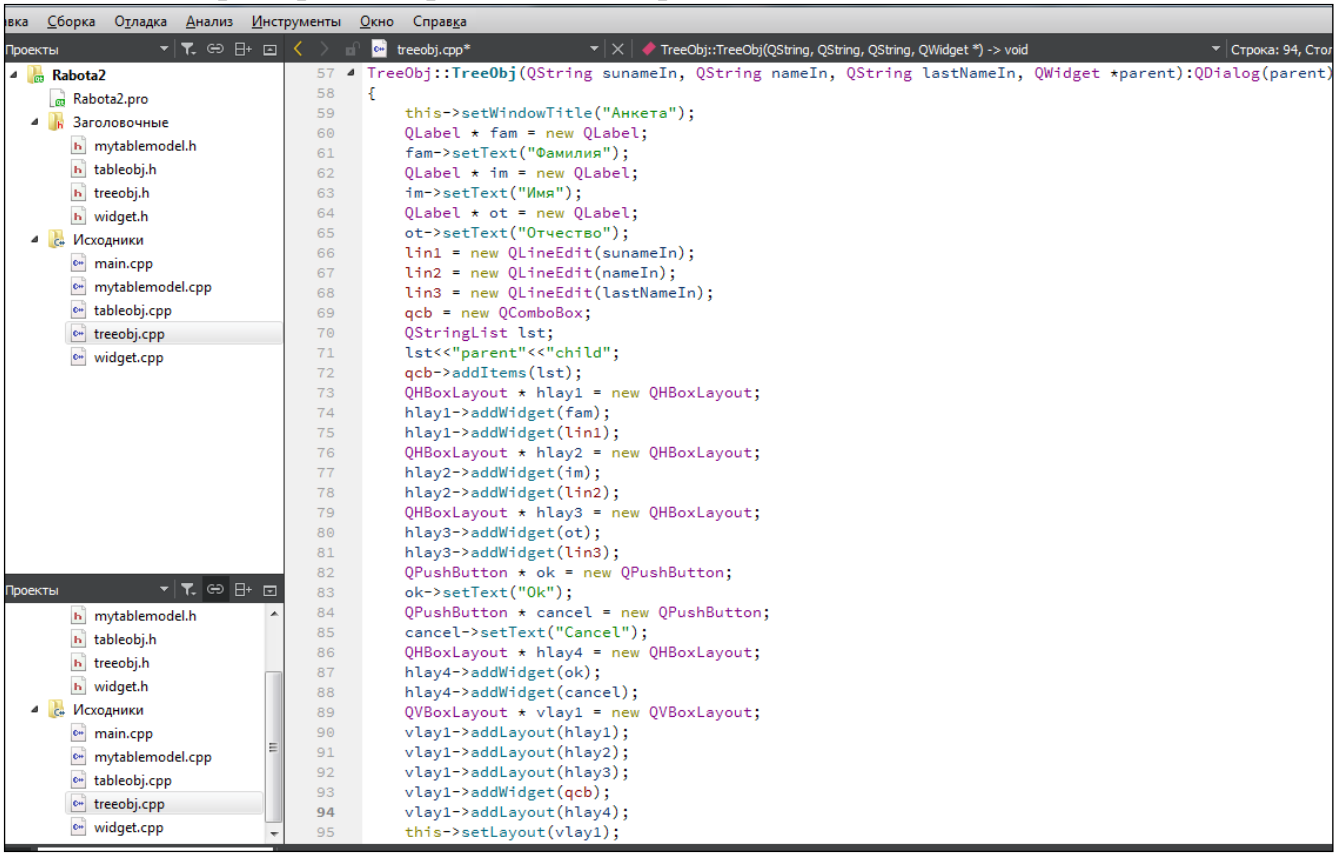
```
void TreeObj::setFam(QString str1)
{
    lin1->setText(str1);
}
QString TreeObj::getFam()
{
    return lin1->text();
}

void TreeObj::setName(QString str2)
{
    lin2->setText(str2);
}
QString TreeObj::getName()
{
    return lin2->text();
}

void TreeObj::setOtche(QString str3)
{
    lin3->setText(str3);
}
QString TreeObj::getOtche()
{
    return lin3->text();
}
```

Рис. 24. Пример выполняемого кода для пункта 6

7. Реализовать перегруженный процессор в файле treeobj.cpp и при помощи метода connect() связать слоты TreeObj::accept и TreeObj::reject.c кнопками Ok и Cancel.
   Пример кода представлен на рис. 25.




```
connect(ok,&QPushButton::clicked,this,&TreeObj::accept);
connect(cancel,&QPushButton::clicked,this,&TreeObj::reject);
```

Рис. 25. Пример кода для пункта 7

8. Реализовать слоты addTree и editTree в widget.cpp
   Пример кода представлен на рис. 26.

```
void Widget::addTree()
{

    TreeObj* add = new TreeObj();
    if (add->exec()==QDialog::Accepted)
    {

        delete add;
    }
else delete add;
}

void Widget::editTree()
{

    TreeObj* edit = new TreeObj("1","2","3");
    if (edit->exec()==QDialog::Accepted)
    {
       delete edit;
    }
    else delete edit;
}
```

Рис. 26. Пример кода для пункта 8

9. Связать кнопки but1 и but3 со слотами Widget::addTree и Widget::editTree при помощью метода connect().

Пример кода приведен на рис. 27.

```
connect(but1,&QPushButton::clicked,this,&Widget::addTree);

connect(but3,&QPushButton::clicked,this,&Widget::editTree);
```

Рис. 27. Пример кода для пункта 9

10. Когда нажимаем «+»-окно добавления, «E»- редактирование. При нажатии кнопки «ОК» каждого диалогового окна в отладчик передаётся строка «test»

Примеры работы окон приведены на рис. 28.

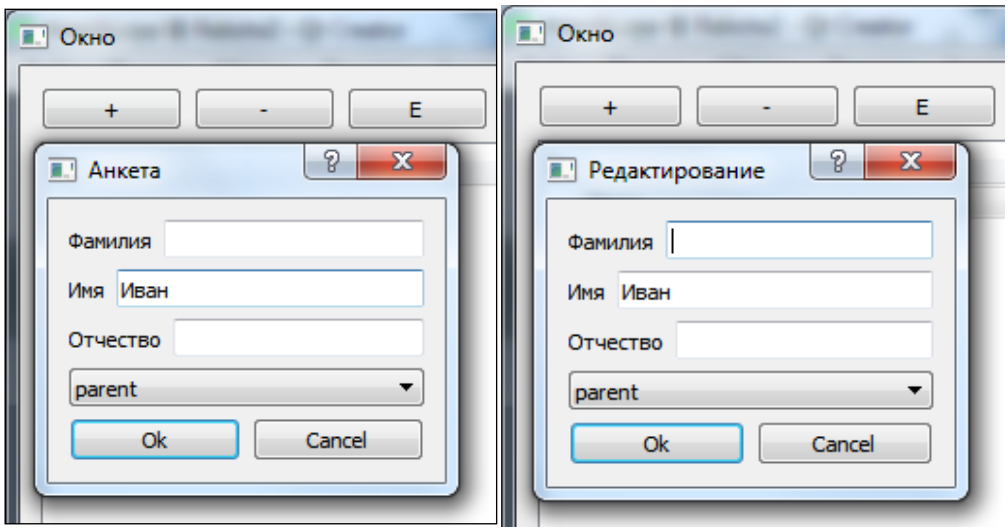


Рис. 28. Примеры работы окон

11. Передача строки в отладчик при нажатии кнопки «ОК».

Пример работы окна отладчика приведен на рис. 29.

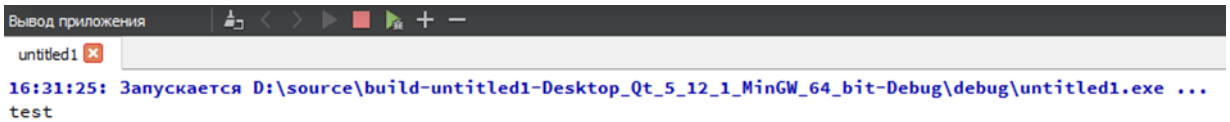


Рис. 29. Пример работы окна отладчика

3.4. УКАЗАНИЯ ПО ОФРМЛЕНИЮ ОТЧЕТА

Отчет должен содержать название и цель работы, краткие теоретические сведения, а также код с комментариями.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Саммерфилд М. Qt. Профессиональное программирование. Разработка кроссплатформенных приложений на C++ / М. Саммерфилд – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 560 с.

2. Шлее М. Qt 5.10. Профессиональное программирование на C++ / М. Шлее – СПб: БХВ-Петербург, 2018. – 1072 с.

3. Бланшет Ж. QT 4: программирование GUI на C++ / Ж. Бланшет – М.: КУДИЦ-ПРЕСС, 2007. – 546 с.

ОГЛАВЛЕНИЕ

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных работ №1-3
для студентов специальности 11.05.01
«Радиоэлектронные системы и комплексы»
очной формы обучения

Составитель
Сукачев Александр Игоревич

Издается в авторской редакции

Подписано к изданию 18.03.2024.
Уч.-изд. л. 2,0

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