

# СЕМИНАР 1

---

Qt. Введение.

# Что такое Qt?

Qt - кроссплатформенный фреймворк для разработки приложений.

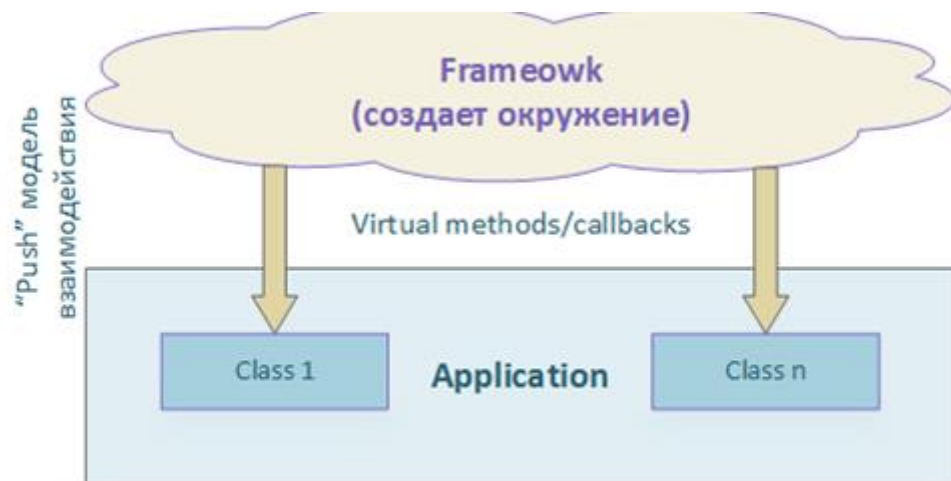


**Code less.  
Create more.  
Deploy everywhere.**

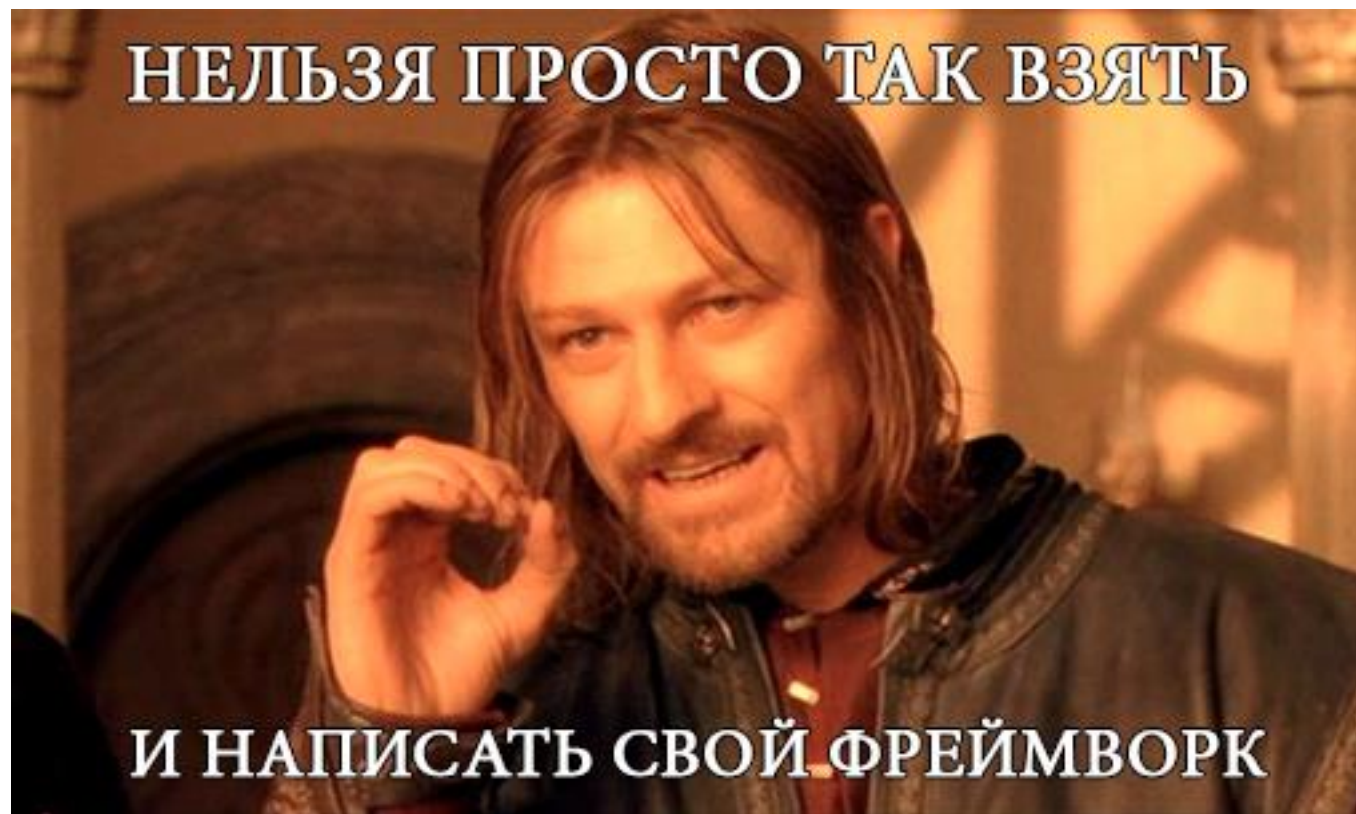
Фреймворк ( от *framework* — каркас, структура) — программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

# Чем отличается фреймворк от библиотеки?

Фреймворк	Библиотека
Каркасный подход к построению ПО. ПО состоит из постоянной части – каркаса и сменных модулей (или точек расширения)	Используется в ПО как набор подпрограмм, не влияя на архитектуру программного продукта.
Фреймворк вызывает функции и классы пользовательского кода.	Пользовательский код вызывает функции и классы библиотеки.
Содержит в себе различное число разных по тематике библиотек.	Объединяет набор модулей близкой функциональности.



# Преимущества Qt



# Преимущества Qt

Преимущества перед другими фреймворками:

1. Платформонезависимость;
2. Обширный инструментарий, гибкость;
3. QtDesigner для интерактивного создания GUI;
4. Широкая распространенность.

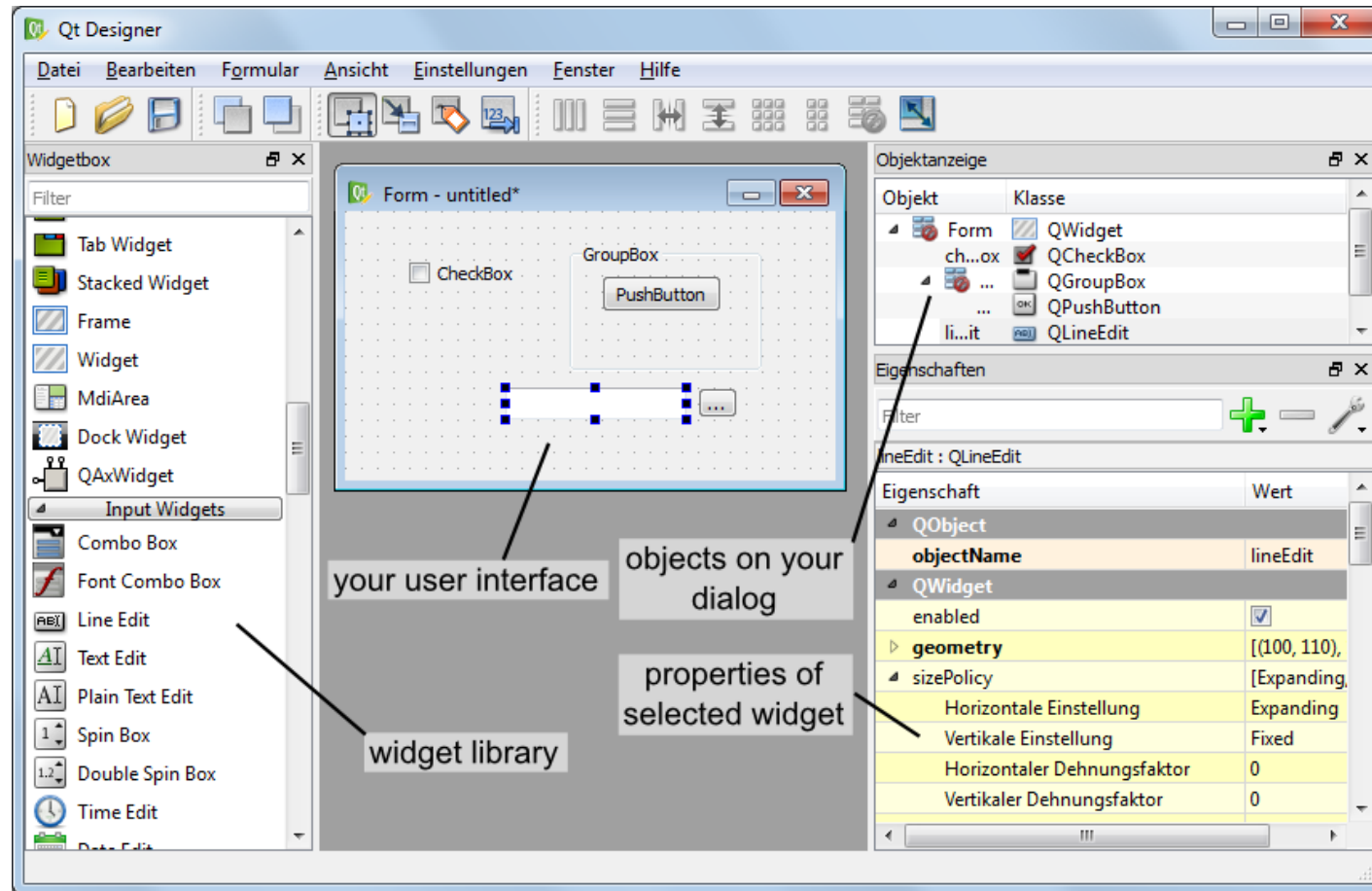
# Платформонезависимость

Предоставляет поддержку большинства операционных систем:

- Windows,
- Linux + \*NIX (Solaris, AIX, Irix, NetBSD, OpenBSD, HP-UX, FreeBSD и др. ),
- Mac OS,
- QNX,
- мобильные операционные системы: iOS, Android, Windows Phone, Windows RT, BlackBerry.



# QtDesigner для интерактивного создания GUI



**Qt Designer** — кроссплатформенная свободная среда для разработки графических интерфейсов (GUI) программ использующих библиотеку Qt.



# Широкая распространенность.

Qt используется разработчиками всего мира.

В число активных пользователей Qt входят такие компании как: Adobe, Amazon, AMD, Bosch, BlackBerry, Cannon, Cisco Systems, Disney, Intel, IBM, Panasonic, Google, NASA, и т.д.



Рис. 1. Редактор трехмерной графики Autodesk Maya

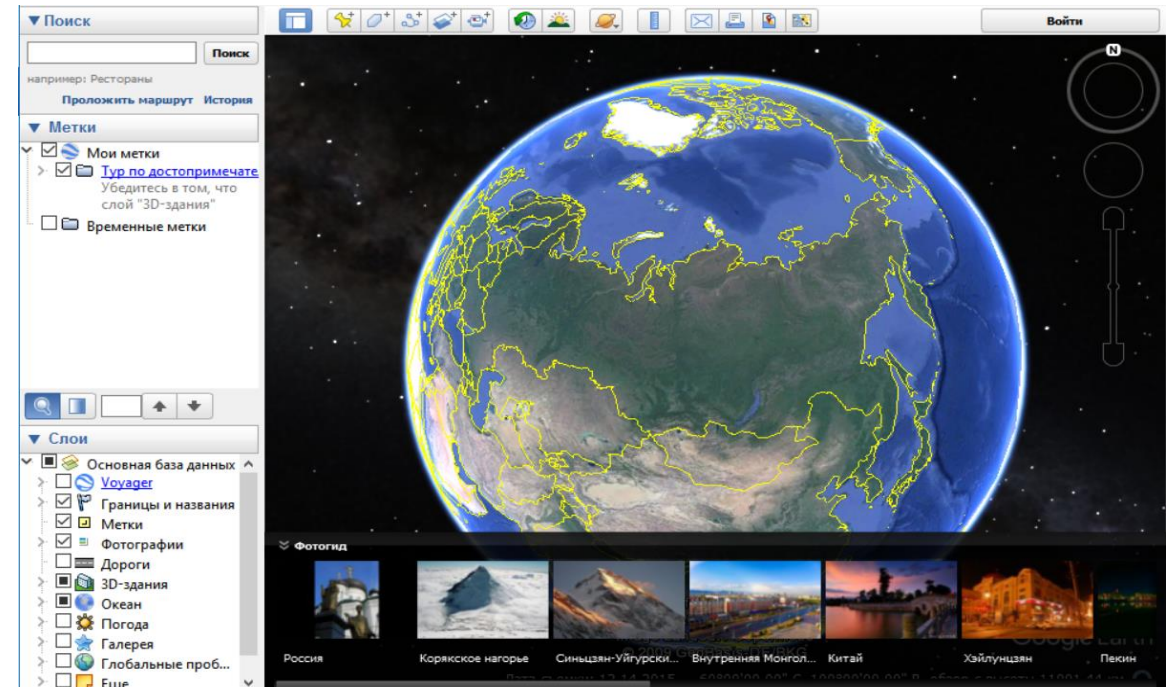


Рис. 2. Google Earth



# Qt. Широкая распространенность.

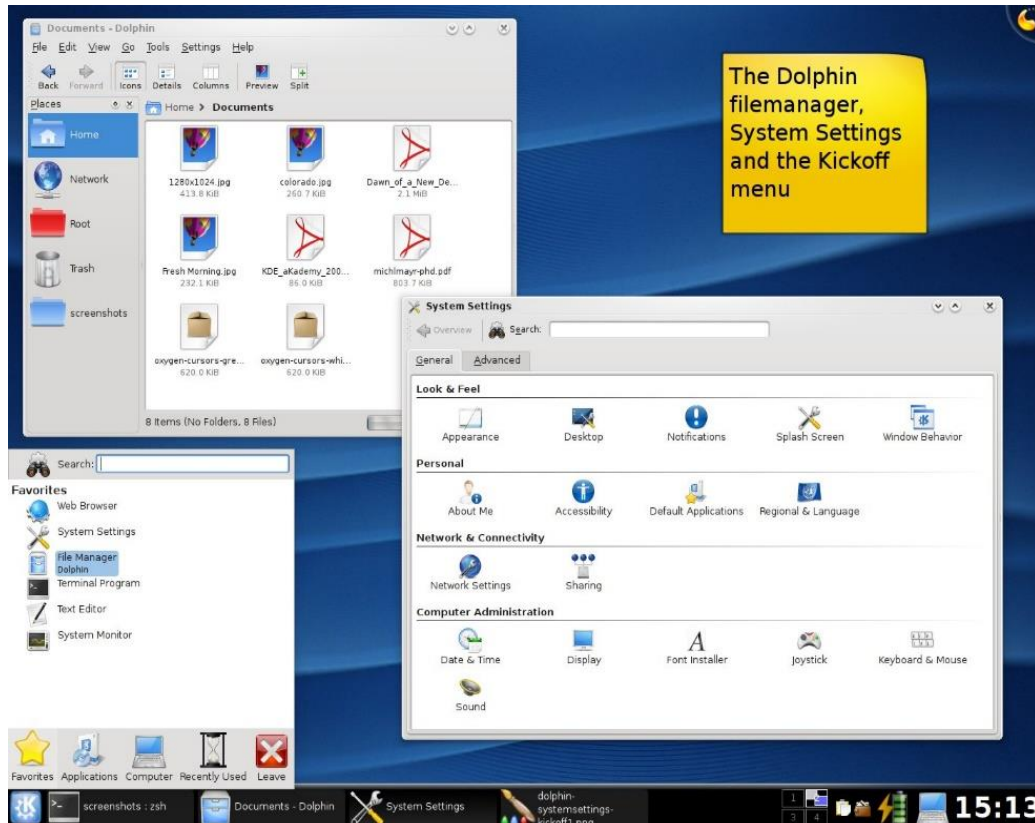


Рис. 3. Рабочий стол KDE Software Compilation 4

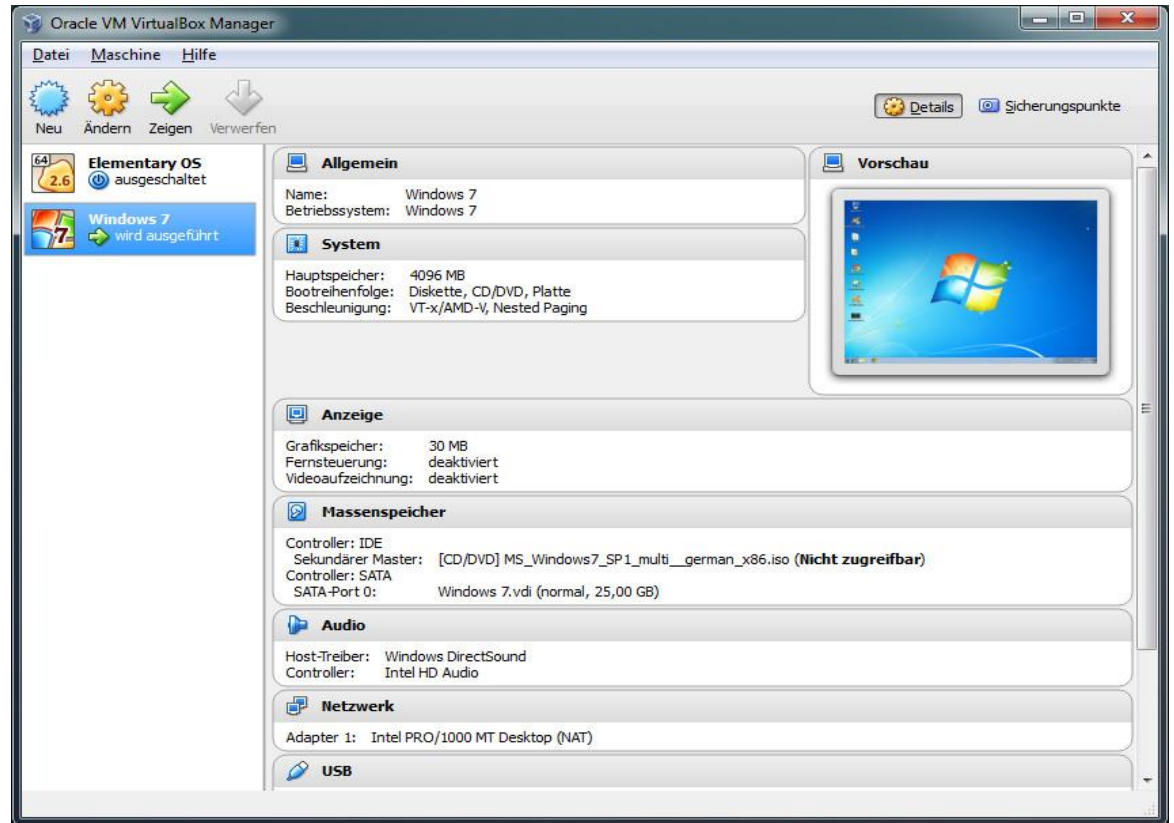


Рис. 4. Программа для виртуализации ОС VirtualBox

# Особенности фреймворка Qt

1. Объектные иерархии или модель «компонент-контейнер»;
2. Механизм «сигнал-слот»;
3. Модульность библиотеки Qt;
4. Типы разрабатываемых приложений;
5. Компилятор метаобъектов МОС;
6. Собственная система описания проекта.

# Особенности Qt. Объектные иерархии или модель «компонент-контейнер\*»

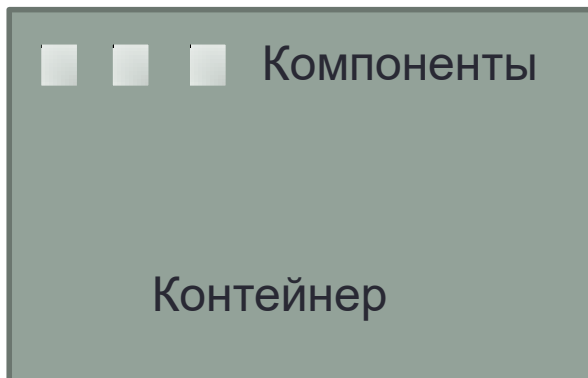
## В C++:

Для выделения памяти под объект используется оператор `new()`.

Для освобождения памяти используется оператор `delete()`.

## В Qt:

Контейнер – объект, который может содержать в своем составе другие объекты (компоненты). При уничтожении контейнера, уничтожаются все его компоненты.



## Ограничения:

- Один компонент не может принадлежать одновременно двум контейнерам.
- Любой компонент может быть контейнером.
- Любой контейнер может быть компонентом.

# Особенности Qt. Объектные иерархии или механизм «компонент-контейнер\*»

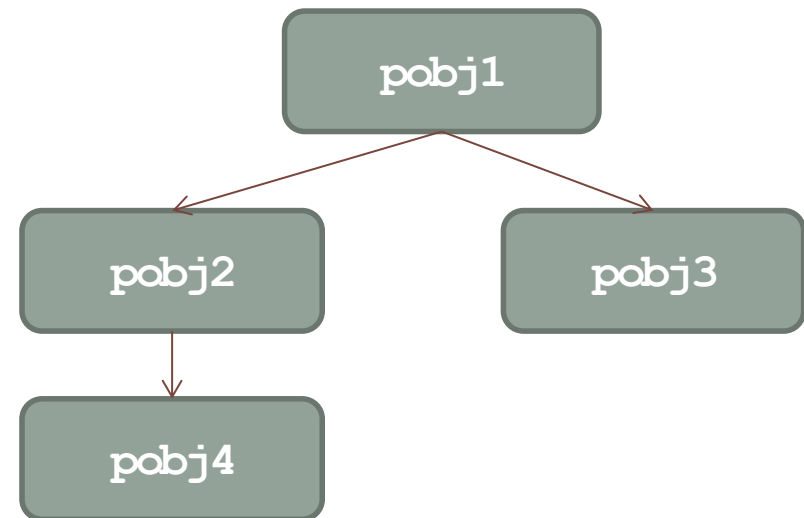
Механизм «компонент-контейнер» реализован в классе QObject. Производные от QObject классы наследуют данный механизм.

Конструктор класса QObject:

`QObject (QObject* pObj = 0);` // в параметре передается указатель на объект-предок (если передается 0, то у создаваемого объекта нет предка и он является объектом верхнего уровня).

Пример:

```
QObject* pObj1 = new QObject;  
QObject* pObj2= new QObject (pObj1);  
QObject* pObj3= new QObject (pObj1);  
QObject* pObj4= new QObject (pObj2);
```



*\*Не путать механизм «компонент-контейнер» с контейнерными классами Qt!*

# Особенности Qt. Механизм «сигнал-слот».

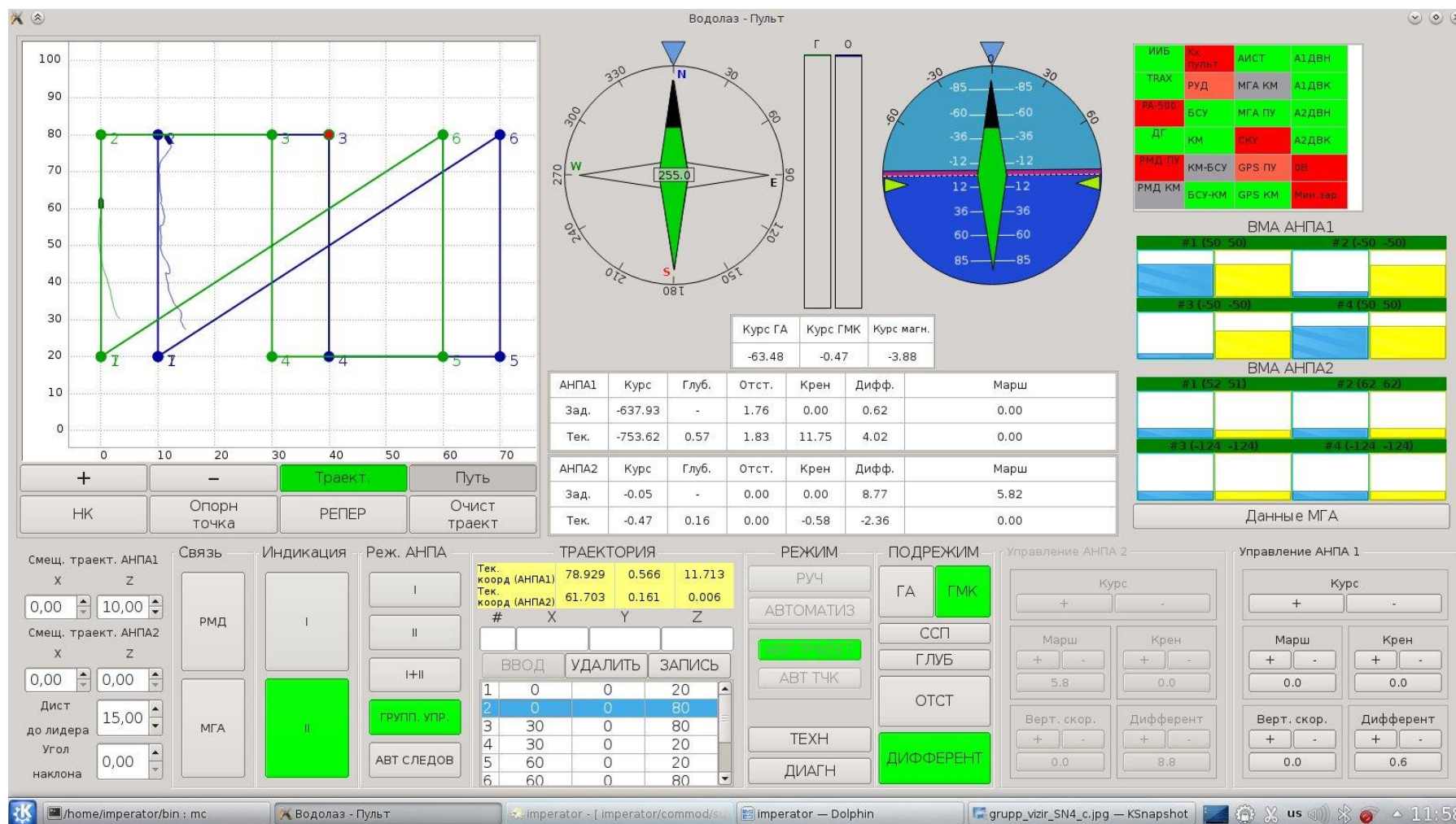


Рис. Пульт управления АНПА



# Особенности Qt. Механизм «сигнал-слот».

- Механизм «сигнал-слот» используется для коммуникации между объектами.

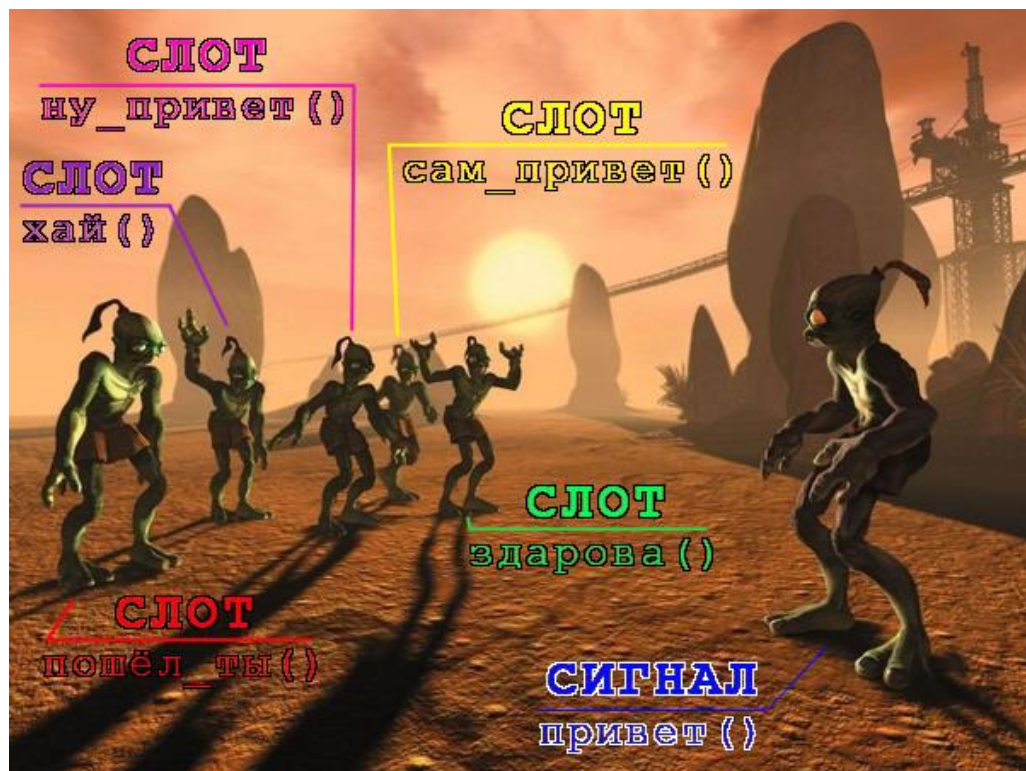


Рис. Механизм «сигнал-слот» в исполнении мудаконов

Сигнал — метод, который в состоянии осуществить пересылку сообщений.

Слот — метод, который присоединяется к сигналам (вызывается в ответ на определенный сигнал).

# Особенности Qt. Механизм «сигнал-слот».

Преимущества, которые дает программисту механизм «сигнал-слот»:

- Соединяемые сигналы и слоты абсолютно независимы и реализованы отдельно друг от друга. Следовательно, большой проект можно разделить на компоненты, которые будут реализовываться разными программистами по отдельности и будут соединяться при помощи сигналов и слотов вместе.
- Соединение сигналов и слотов можно производить в любой точке приложения.
- Соединение сигнала и слота можно осуществлять даже между объектами, которые находятся в различных потоках.
- При уничтожении объекта происходит автоматическое разъединение всех сигнально-слотовых связей.

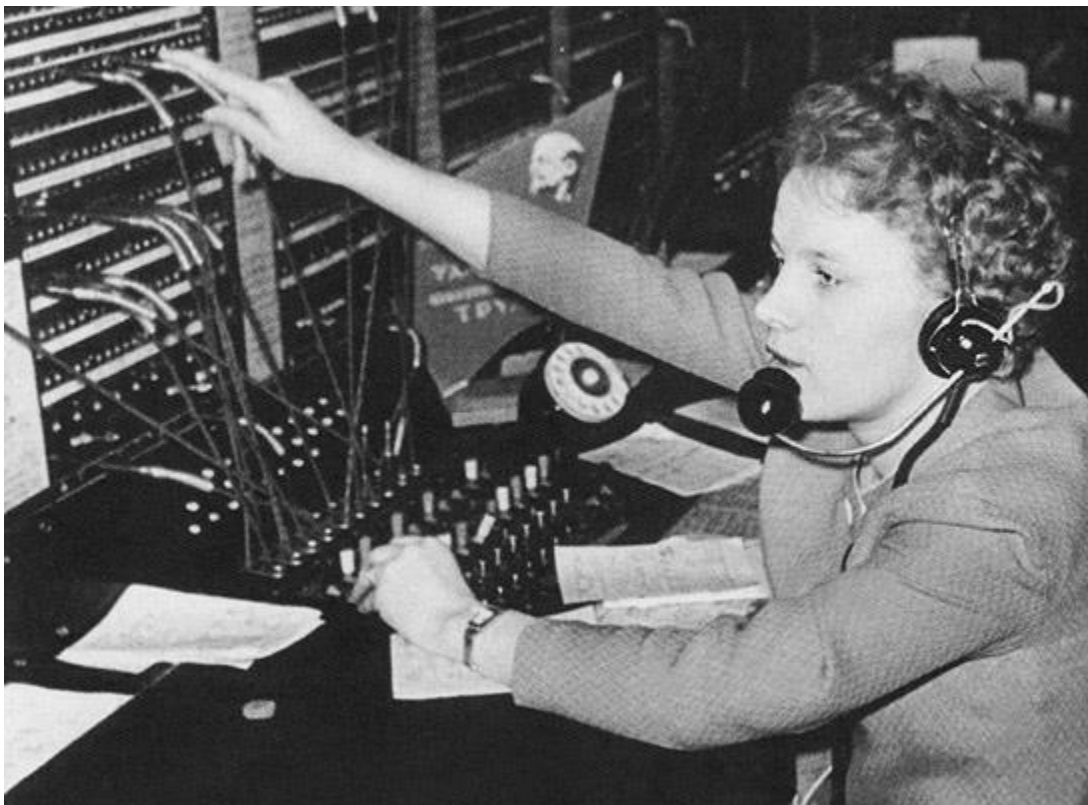


# Особенности Qt. Механизм «сигнал-слот».

Одному сигналу может  
соответствовать много слотов.



Одному слоту может  
соответствовать много сигналов.



# Особенности Qt. Механизм «сигнал-слот».

Недостатки, связанные с применением механизма «сигнал-слот»:

- Сигналы и слоты не являются частью языка C++, поэтому требуется запуск дополнительного препроцессора перед компиляцией программы.
- Данный механизм реализован в классе QObject. Для использования сигналов и слотов, класс должен быть унаследован от QObject, а в описании класса должен быть прописан макрос Q\_OBJECT.
- В процессе компиляции не производится никаких проверок: имеется ли сигнал или слот в соответствующих классах или нет; совместимы ли сигнал и слот друг с другом и могут ли они быть соединены вместе. (хотя в Qt5 реализован новый механизм, который частично устраняет этот недостаток).

# Модульность.

Библиотека Qt – это множество классов (более 500). Qt не является единым целым, она разбита на модули. В зависимости от требований к проекту вы можете выбрать какие модули подключить, но любая Qt-программа должна использовать хотя бы один из основных модулей – QtCore, QtGui или QtWidgets.

Основные модули	Дополнительные модули
QtCore	QtNetwork
	QtSerialPort
	QtQML
QtGui	QtXML
	QTSql
	QtOpenGL
QtWidgets	QtScript
	QtPy
	QtMultimedia и т.д.

# Особенности Qt. Типы разрабатываемых приложений

Типы приложений:

- Консольное приложение;
- Desktop GUI;
- Mobile GUI;
- CGI – приложения (WEB-приложения);
- Сервисные приложения, demon'ы (не взаимодействуют с пользователем вообще, взаимодействуют только с программами).

# Qt Особенности. Компилятор метаобъектов МОС

МОС (Meta Object Compiler) – метаобъектный компилятор.

МОС - механизм Qt, который добавляет необходимый код C++ в проект (реализует механизм сигнал-слот, создает код C++ для графического интерфейса, фалов ресурсов и т.п.).

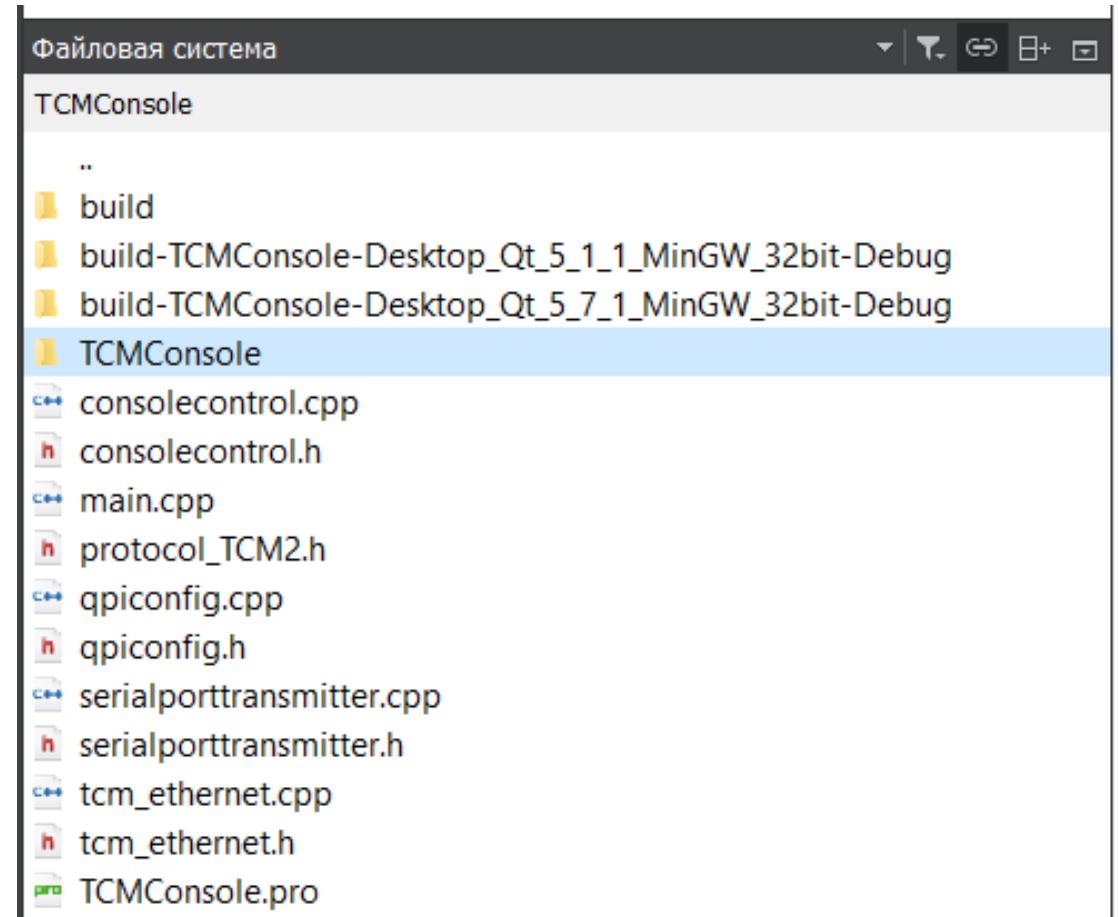
Полученный код сохраняется в файле с прототипом имени:  
`moc_<filename>.cpp`.

Для работы МОС в описании класса необходима директива `Q_OBJECT`.

# Особенности Qt. Своя система описания проекта.

Qt структура проекта:

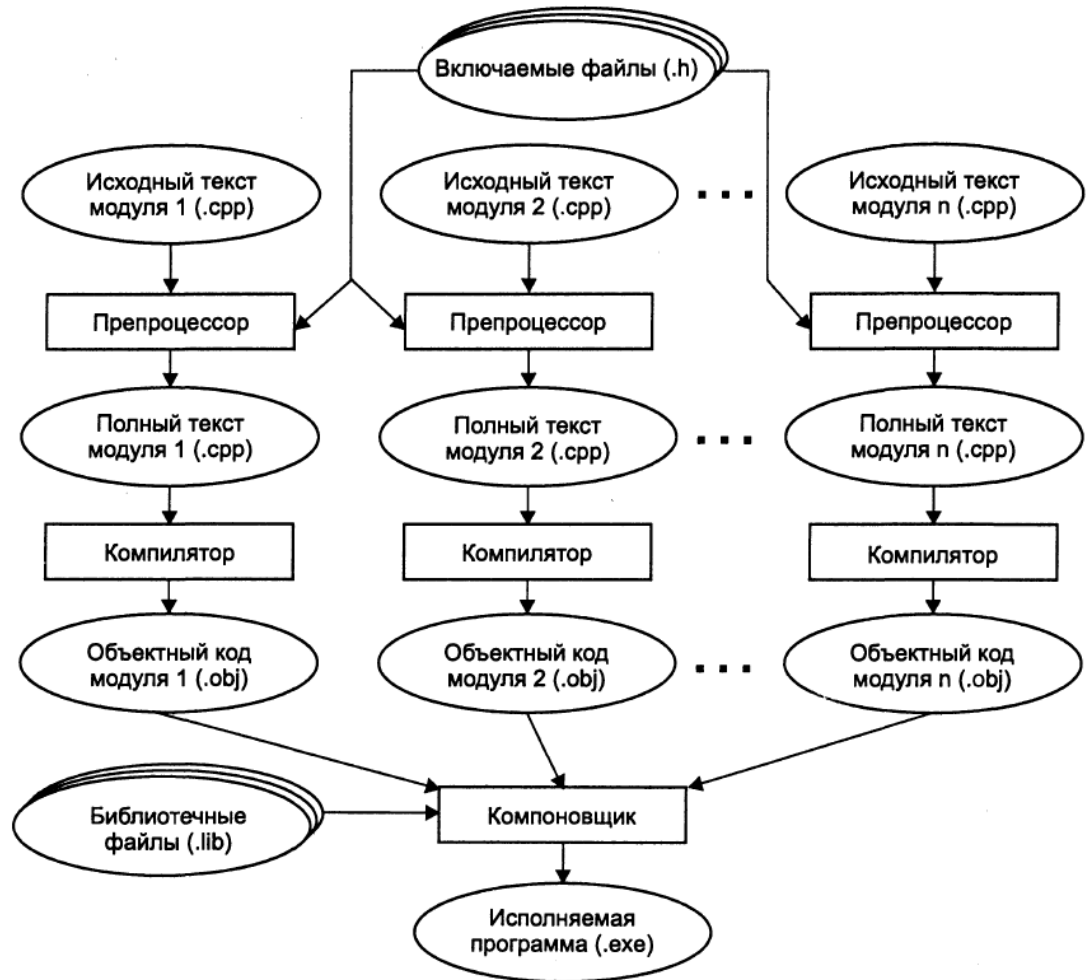
- Исходные коды;
- Файл описания проекта .pro.



# Особенности Qt. Собственная система описания проекта.

Этапы сборки проекта C++:

1. Преппроцессинг.
2. Ассемблирование.
3. Компилирование.
4. Линковка.



Этапы создания исполняемой программы



# Особенности Qt. Собственная система описания проекта.

Традиционно сборка проекта происходит средствами утилиты make, исполняющей сценарий описанный в Makefile.

Создание Makefile:

1. Вручную;
2. С помощью систем сборки:
  - GNU Toolchain;
  - CMake;
  - QMake;
  - etc...

# Qt Особенности. Собственная система описания проекта.

Этапы сборки проекта Qt:

- 1.1. Подготовка файла описания проекта .pro;
- 1.2. Предварительная обработка проекта с помощью qmake. (Создание make-файла и дополнительных исходных файлов);
- 2.1. Сборка проекта утилитой make, в make-файле содержится вызов MOC для создания дополнительного кода C++ и необходимых заголовочных файлов;
- 2.2. Если проект содержит qrc-файл, то также будет создан файл C++, содержащий данные ресурсов;
- 2.3. Все исходные коды компилируются C++-компилятором в файлы объектного кода;
- 2.4. Файлы объектного кода объединяются компоновщиком link в исполняемый модуль.

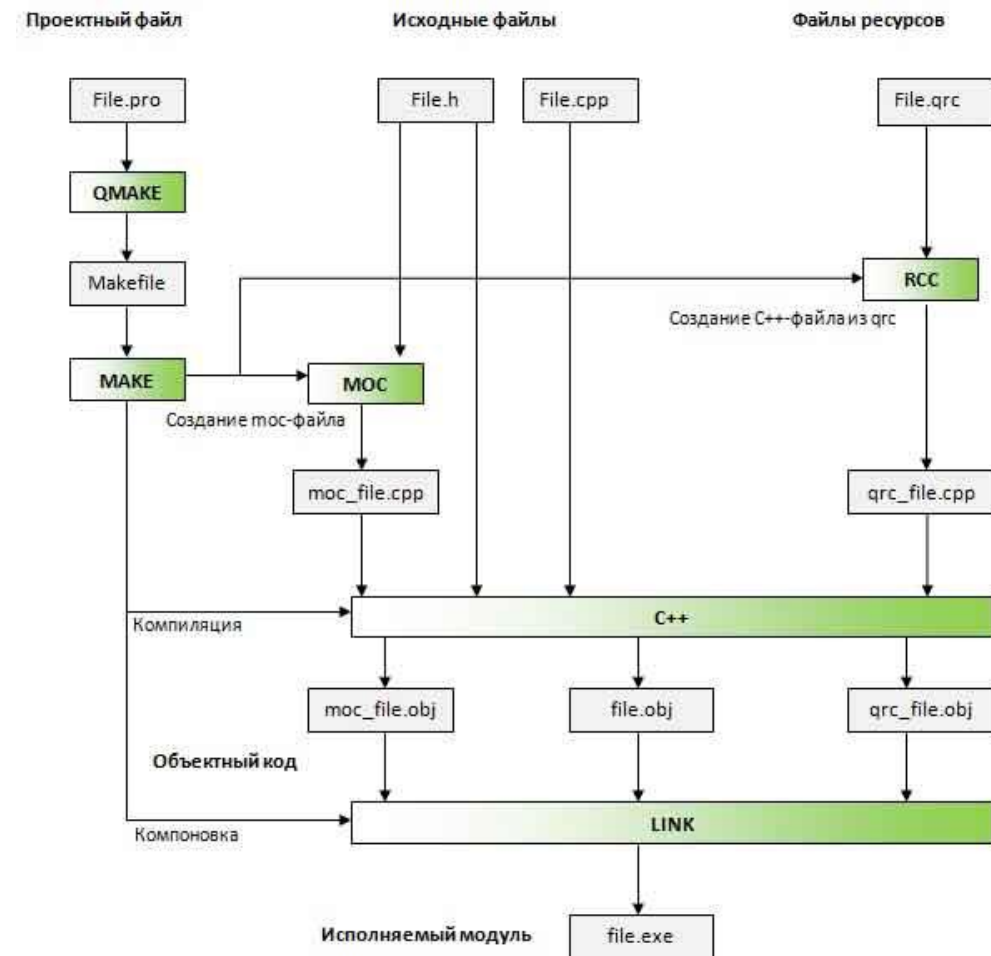
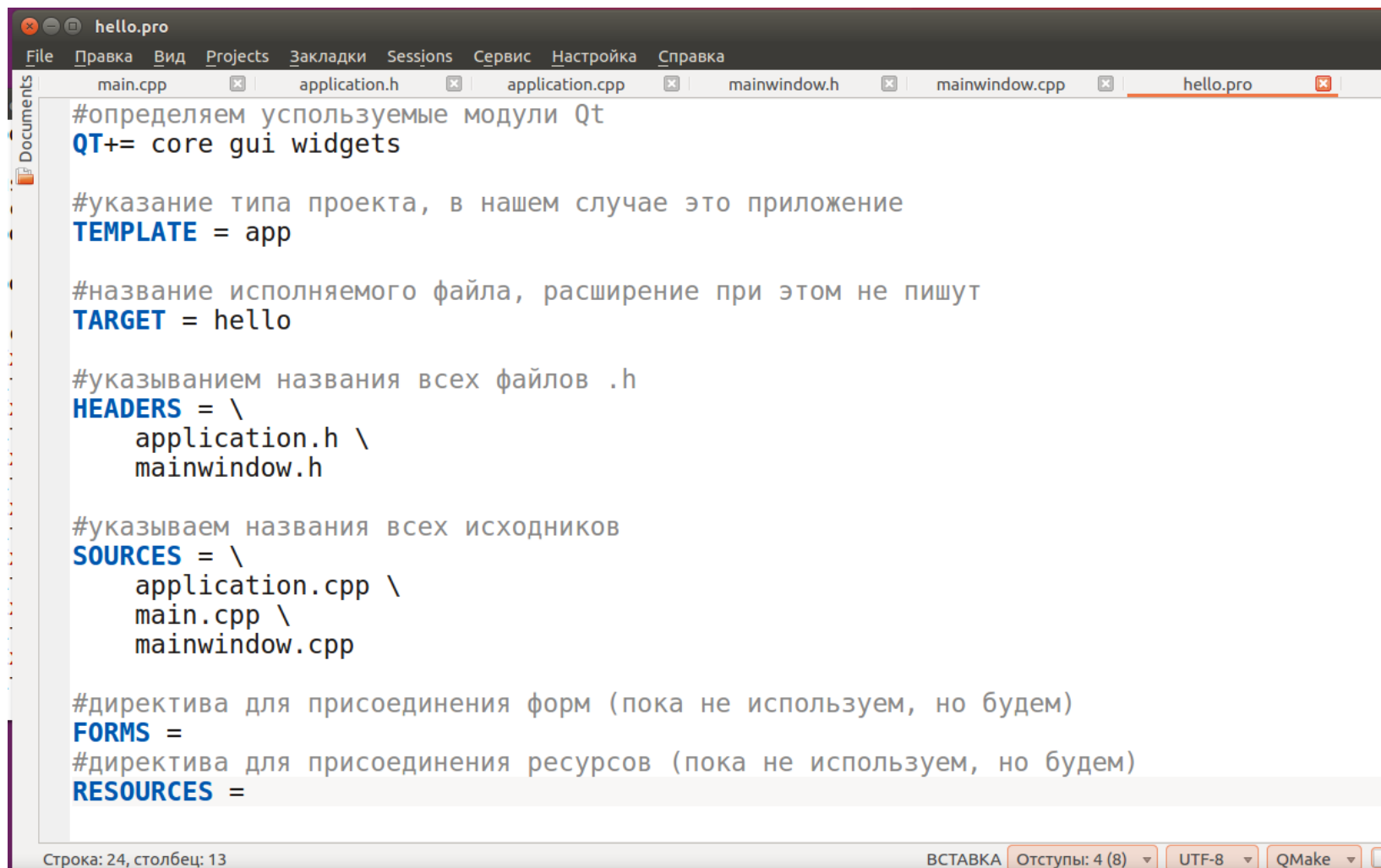


Рис 1. Процесс сборки проекта Qt

# Qt. Файл описания проекта .pro



```
hello.pro
File  Правка  Вид  Projects  Закладки  Sessions  Сервис  Настройка  Справка
main.cpp  application.h  application.cpp  mainwindow.h  mainwindow.cpp  hello.pro
#определяем используемые модули Qt
QT+= core gui widgets

#указание типа проекта, в нашем случае это приложение
TEMPLATE = app

#название исполняемого файла, расширение при этом не пишут
TARGET = hello

#указыванием названия всех файлов .h
HEADERS = \
    application.h \
    mainwindow.h

#указываем названия всех исходников
SOURCES = \
    application.cpp \
    main.cpp \
    mainwindow.cpp

#директива для присоединения форм (пока не используем, но будем)
FORMS =

#директива для присоединения ресурсов (пока не используем, но будем)
RESOURCES =

Строка: 24, столбец: 13  ВСТАВКА  Отступы: 4 (8)  UTF-8  QMake
```

# Qt. Файл описания проекта .pro

Настройки, которые задаются в .pro – файле:

- тип проекта (приложение, динамическая или статическая библиотека, проект, который состоит из подпроектов);
- общие настройки проекта;
- настройки компиляции;
- путь, где будет размещён исполняемый файл, библиотека или бинарный файл во время процесса компиляции;
- пути к файлам, библиотекам и другим частям проекта необходимым для компиляции;
  - файлы, входящие в проект;
- дополнительные действия, которые будут выполняться в процессе компиляции проекта.

# Лицензирование Qt

Вид лицензии	Условия
1. GNU GPL v3	Программа должна быть открыта, свободно распространяться, исходные тексты программы и все изменения в исходных текстах Qt должны пребывать в свободном доступе.
2. GNU LGPL v3	Исходные тексты программы могут быть как открытыми так и закрытыми. В случае, если программа является закрытой и планируется коммерческое использование программы — Qt должен связываться с программой в виде динамических библиотек. Конечно, в этом случае нельзя вставлять и использовать любые исходные тексты Qt в программе. Также любые изменения в исходных текстах Qt должны быть пребывать в свободном доступе.
3. Коммерческая лицензия	Кроме возможности закрывать, модифицировать любым образом текст программы, модифицировать или закрывать изменения в коде Qt и произвольно выбирать лицензию и способ распространения программы, предоставляется также поддержка и консультации по использованию Qt

# Лицензирование Qt



# Установка Qt

Варианты установки для Linux, Windows, Mac OS :  
(<https://www.qt.io/download-open-source/>)

## Qt Online Installers

Qt online installer is a small executable which downloads content over internet based on your selections. It provides binary and source packages for different Qt library versions and latest Qt Creator.

- › [Qt Online Installer for Linux 64-bit \(31 MB\)](#) (info)
- › [Qt Online Installer for Linux 32-bit \(33 MB\)](#) (info)
- › [Qt Online Installer for macOS \(12 MB\)](#) (info)
- › [Qt Online Installer for Windows \(17 MB\)](#) (info)

## Offline Installers

Qt offline installer is a stand-alone binary package including Qt libraries and Qt Creator.

### Linux Host

- › [Qt 5.8.0 for Linux 64-bit \(766 MB\)](#) (info)
- › [Qt 5.8.0 for Android \(Linux 64-bit, 817 MB\)](#) (info)

### macOS Host

- › [Qt 5.8.0 for macOS \(1.2 GB\)](#) (info)
- › [Qt 5.8.0 for Android \(armv7, x86\) \(1.4 GB\)](#) (info)
- › [Qt 5.8.0 for Android \(armv7, x86\) and iOS \(3.4 GB\)](#) (info)

### Windows Host

- › [Qt 5.8.0 for Windows 64-bit \(VS 2015, 1.0 GB\)](#) (info)
- › [Qt 5.8.0 for Windows 32-bit \(VS 2015, 1.0 GB\)](#) (info)
- › [Qt 5.8.0 for Windows 64-bit \(VS 2013, 958 MB\)](#) (info)
- › [Qt 5.8.0 for Windows 32-bit \(VS 2013, 947 MB\)](#) (info)
- › [Qt 5.8.0 for Windows 32-bit \(MinGW 5.3.0, 1.2 GB\)](#) (info)
- › [Qt 5.8.0 for Android \(Windows 32-bit, 1.3 GB\)](#) (info)



# Доступ к материалам по курсу

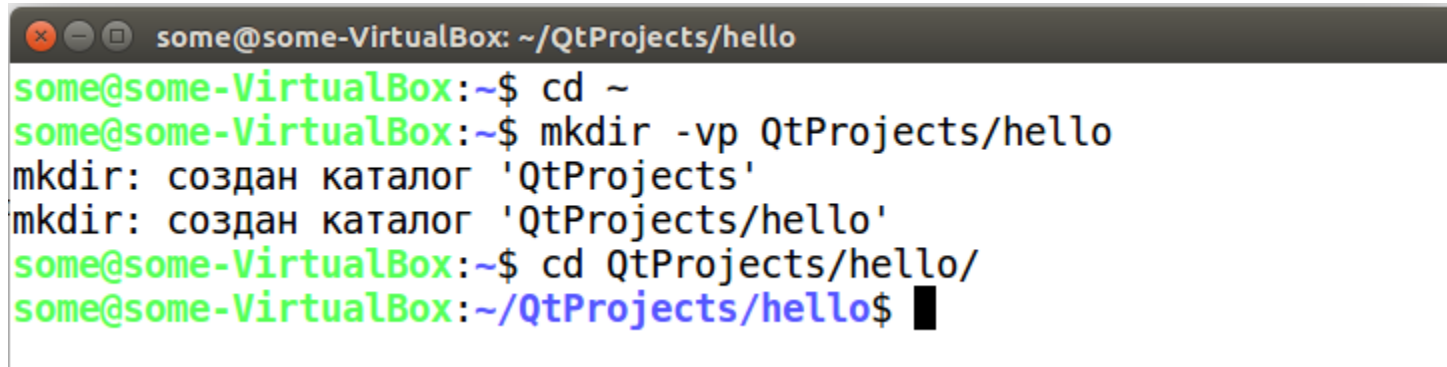
[https://gitlab.com/groups/bmstu\\_underwater\\_robotics](https://gitlab.com/groups/bmstu_underwater_robotics)

The screenshot shows the GitLab web interface for the group **bmstu\_underwater\_robotics**. The browser address bar displays the URL [https://gitlab.com/groups/bmstu\\_underwater\\_robotics](https://gitlab.com/groups/bmstu_underwater_robotics). The page header includes the group name, a search bar, and navigation links for Group, Issues (0), Merge Requests (0), Members, Contribution Analytics, and Settings. Below the header, the group's profile is shown with a globe icon, the name **bmstu\_underwater\_robotics**, and a description: "Group for BMSTU Underwater Robotics students. Группа для учебных проектов студентов кафедры 'Подводные роботы и аппараты' МГТУ им. Баумана." There are buttons for "Leave group" and "Global" notifications. The main content area lists projects under the "Projects" tab, with filters for "Filter by name..." and "Last updated". A "New Project" button is also present. The project list includes:

Project Name	Description	Updated
ni_prakt	Материалы по дисциплине "Научно-исследовательская практика"	updated about 16 hours ago
prts	Материалы по курсу "Проектирование подводных робототехнических систем"	updated about 16 hours ago
rpk_prts_2017	Проекты студентов по курсу "Разработка программных комплексов ПРТС" за 2017 год	updated 3 months ago

# Практическая часть::HelloWorld!

Создадим директории для будущего проекта:

A terminal window with a dark title bar containing the text 'some@some-VirtualBox: ~/QtProjects/hello'. The terminal shows a series of commands and their outputs. The first command is 'cd ~', followed by 'mkdir -vp QtProjects/hello'. The output for the mkdir command is split into two lines: 'mkdir: создан каталог 'QtProjects'' and 'mkdir: создан каталог 'QtProjects/hello''. The next command is 'cd QtProjects/hello/'. The final prompt is 'some@some-VirtualBox: ~/QtProjects/hello\$' with a black cursor block.

```
some@some-VirtualBox: ~/QtProjects/hello
some@some-VirtualBox:~$ cd ~
some@some-VirtualBox:~$ mkdir -vp QtProjects/hello
mkdir: создан каталог 'QtProjects'
mkdir: создан каталог 'QtProjects/hello'
some@some-VirtualBox:~$ cd QtProjects/hello/
some@some-VirtualBox:~/QtProjects/hello$
```

\*Команды:

\$cd — (**c**hange **d**irectory) — команда командной строки для изменения текущего рабочего каталога в Unix, DOS и других операционных системах.

\$mkdir - (**m**ake **d**irectory) в операционной системе Unix, Linux, DOS, Windows — команда для создания новых каталогов.

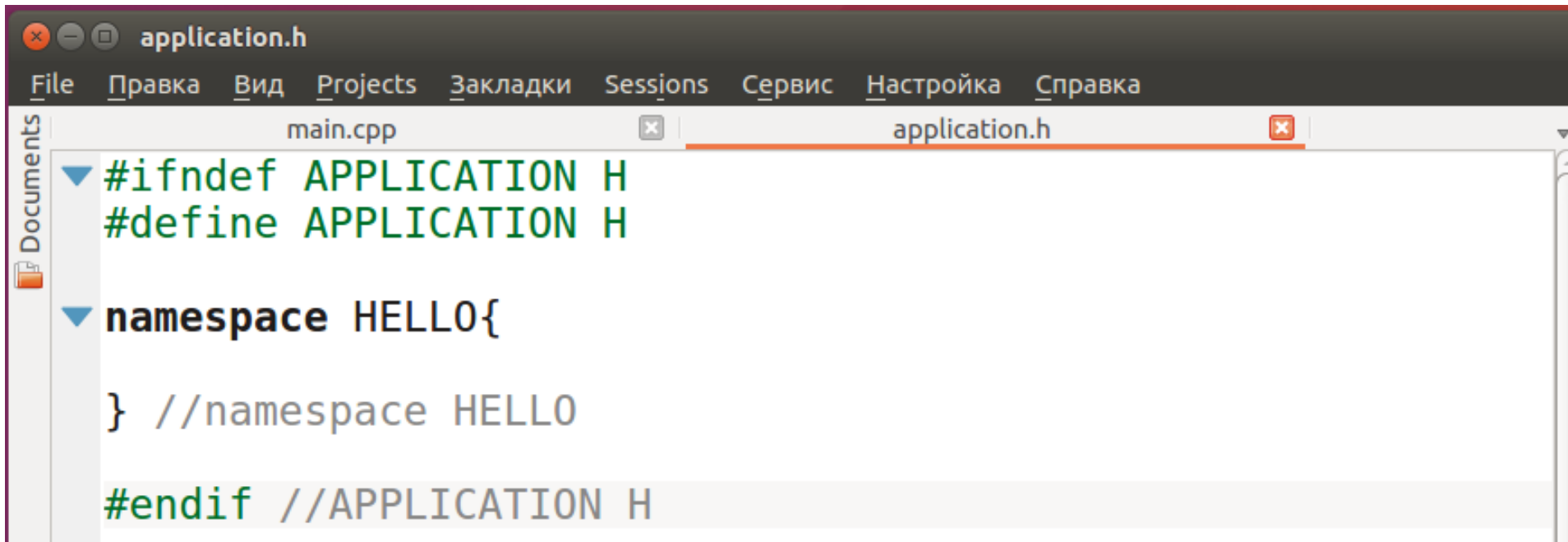
Аргументы:

- p, --parents не выдавать ошибок если существует, создавать родительские каталоги если необходимо;
- v, --verbose печатать сообщение о каждом созданном каталоге.

# Подготовка к работе

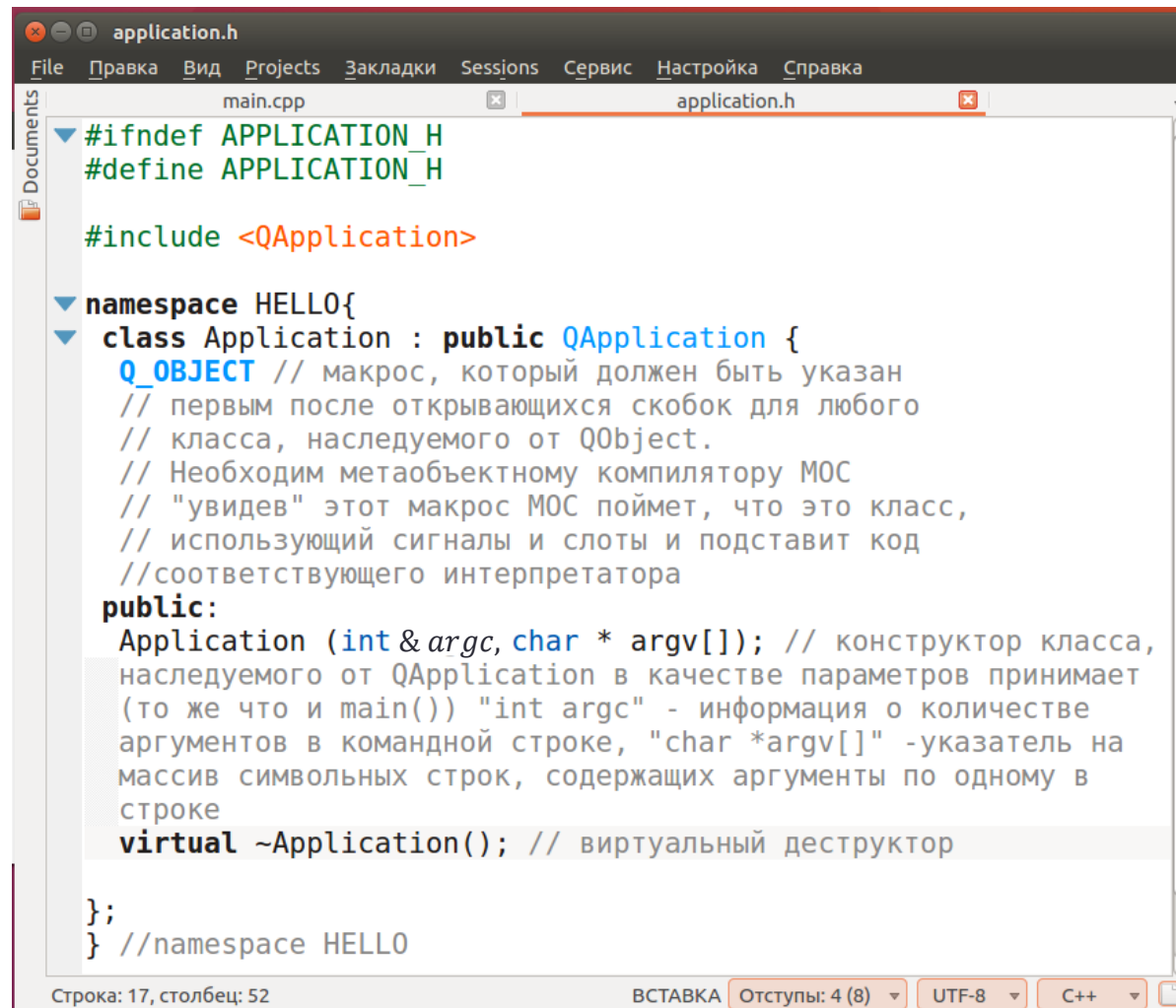
Создадим файлы main.cpp application.h и откроем их в редакторе kate:

```
some@some-VirtualBox: ~/QtProjects/hello
some@some-VirtualBox:~/QtProjects/hello$ touch main.cpp application.h
some@some-VirtualBox:~/QtProjects/hello$ ls
application.h  main.cpp
some@some-VirtualBox:~/QtProjects/hello$ kate main.cpp application.h
```



```
application.h
File  Правка  Вид  Projects  Закладки  Sessions  Сервис  Настройка  Справка
main.cpp  application.h
▼ #ifndef APPLICATION H
  #define APPLICATION H
▼ namespace HELLO{
    } //namespace HELLO
#endif //APPLICATION H
```

# Создание главного класса приложения (application.h)



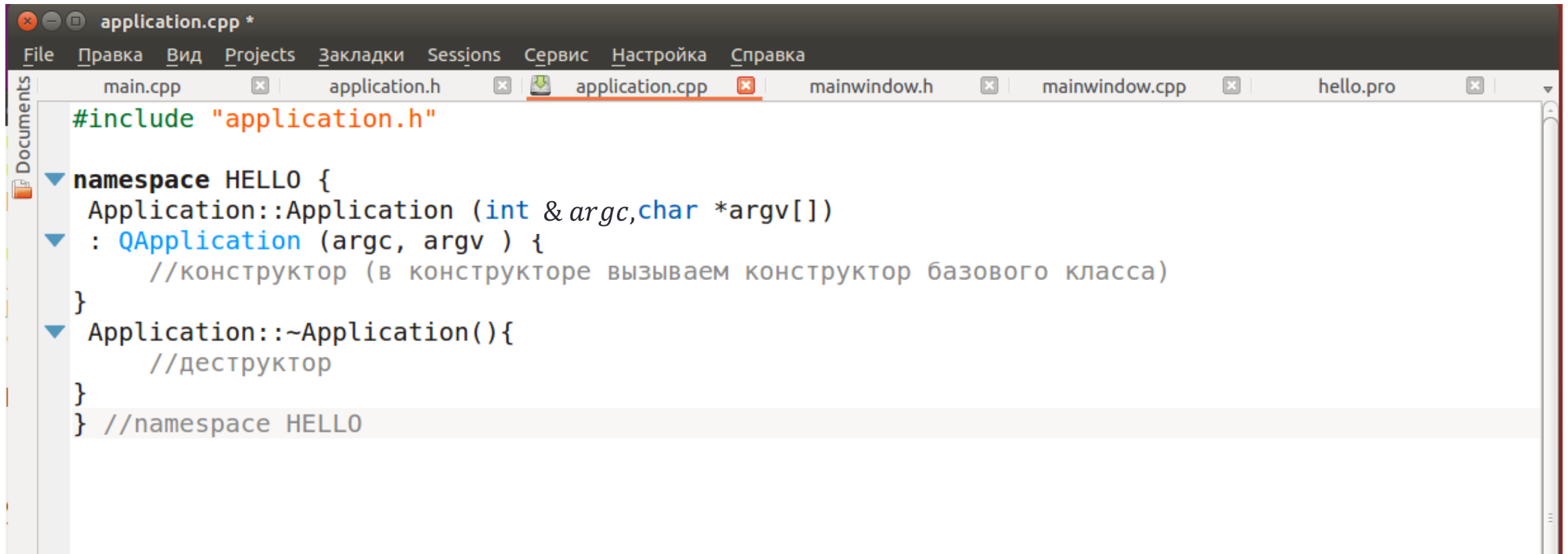
```
application.h
File Правка Вид Projects Закладки Sessions Сервис Настройка Справка
main.cpp application.h
#ifndef APPLICATION_H
#define APPLICATION_H

#include <QApplication>

namespace HELLO{
class Application : public QApplication {
    Q_OBJECT // макрос, который должен быть указан
              // первым после открывающихся скобок для любого
              // класса, наследуемого от QObject.
              // Необходим метаобъектному компилятору MOC
              // "увидев" этот макрос MOC поймет, что это класс,
              // использующий сигналы и слоты и подставит код
              // соответствующего интерпретатора
public:
    Application (int &argc, char * argv[]); // конструктор класса,
                                              наследуемого от QApplication в качестве параметров принимает
                                              (то же что и main()) "int argc" - информация о количестве
                                              аргументов в командной строке, "char *argv[]" -указатель на
                                              массив символьных строк, содержащих аргументы по одному в
                                              строке
    virtual ~Application(); // виртуальный деструктор
};
} //namespace HELLO

Строка: 17, столбец: 52
ВСТАВКА Отступы: 4 (8) UTF-8 C++
```

# Создание главного класса приложения (application.cpp)

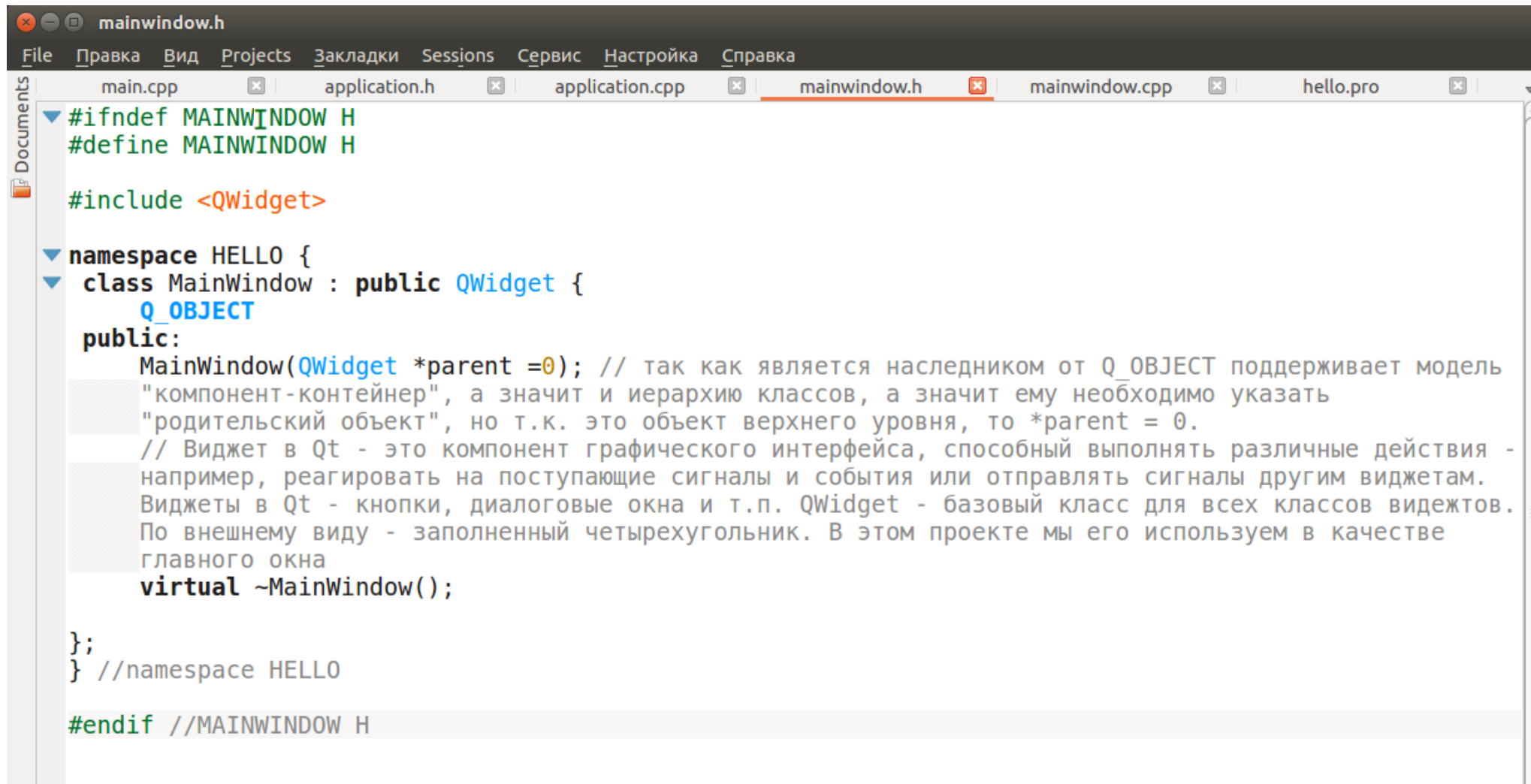


The screenshot shows a Qt IDE window titled "application.cpp \*". The menu bar includes "File", "Правка", "Вид", "Projects", "Закладки", "Sessions", "Сервис", "Настройка", and "Справка". The tab bar shows several open files: "main.cpp", "application.h", "application.cpp" (which is the active file and highlighted with a red underline), "mainwindow.h", "mainwindow.cpp", and "hello.pro". The left sidebar shows a "Documents" view with a folder icon. The main editor area displays the following C++ code:

```
#include "application.h"

namespace HELLO {
    Application::Application (int &argc, char *argv[])
    : QApplication (argc, argv ) {
        //конструктор (в конструкторе вызываем конструктор базового класса)
    }
    Application::~Application(){
        //деструктор
    }
} //namespace HELLO
```

# Создание главного окна приложения (mainwindow.h)



```
mainwindow.h
File  Правка  Вид  Projects  Закладки  Sessions  Сервис  Настройка  Справка
main.cpp  application.h  application.cpp  mainwindow.h  mainwindow.cpp  hello.pro
Documents
▼ #ifndef MAINWINDOW H
#define MAINWINDOW H

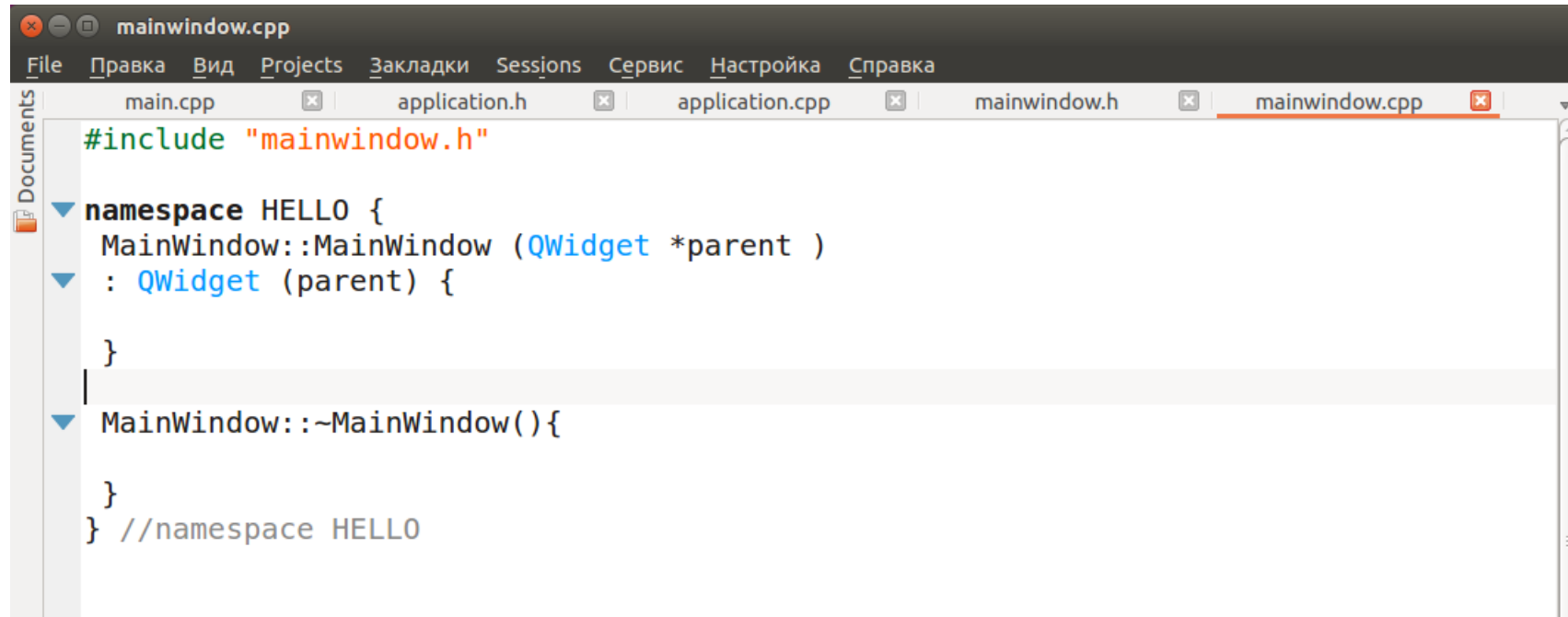
#include <QWidget>

▼ namespace HELLO {
▼ class MainWindow : public QWidget {
    Q_OBJECT
public:
    MainWindow(QWidget *parent =0); // так как является наследником от Q_OBJECT поддерживает модель
    "компонент-контейнер", а значит и иерархию классов, а значит ему необходимо указать
    "родительский объект", но т.к. это объект верхнего уровня, то *parent = 0.
    // Виджет в Qt - это компонент графического интерфейса, способный выполнять различные действия -
    например, реагировать на поступающие сигналы и события или отправлять сигналы другим виджетам.
    Виджеты в Qt - кнопки, диалоговые окна и т.п. QWidget - базовый класс для всех классов виджетов.
    По внешнему виду - заполненный четырехугольник. В этом проекте мы его используем в качестве
    главного окна
    virtual ~MainWindow();

};
} //namespace HELLO

#endif //MAINWINDOW H
```

# Создание главного окна приложения (mainwindow.cpp)



The screenshot shows the Qt Creator IDE with the 'mainwindow.cpp' file open. The interface includes a menu bar with options like 'File', 'Правка', 'Вид', 'Projects', 'Закладки', 'Sessions', 'Сервис', 'Настройка', and 'Справка'. Below the menu bar is a tab bar showing several open files: 'main.cpp', 'application.h', 'application.cpp', 'mainwindow.h', and 'mainwindow.cpp' (which is the active file). On the left side, there is a 'Documents' sidebar. The main editor area displays the following C++ code:

```
#include "mainwindow.h"

namespace HELLO {
    MainWindow::MainWindow (QWidget *parent )
    : QWidget (parent) {

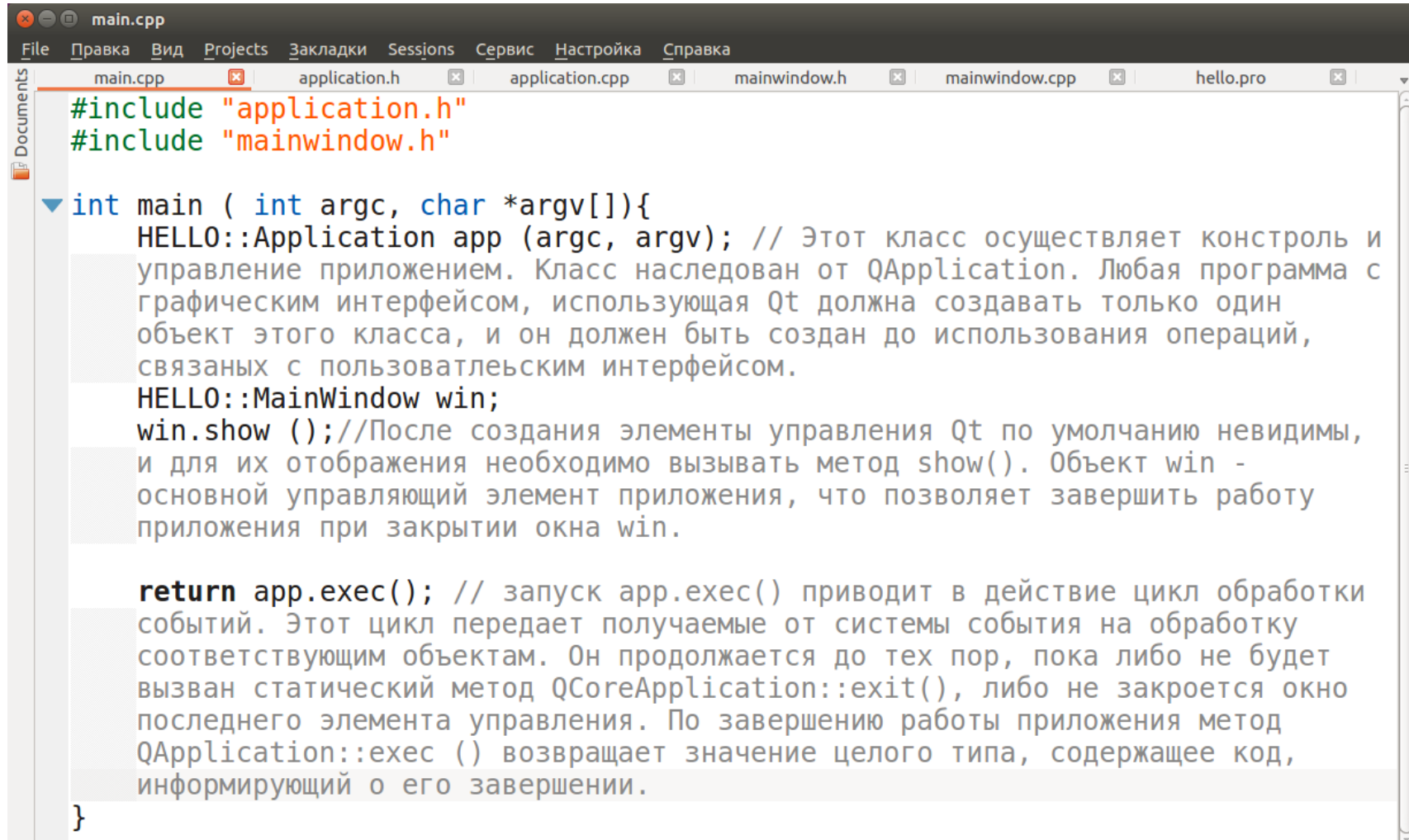
    }

    MainWindow::~MainWindow(){

    }
} //namespace HELLO
```



# main.cpp



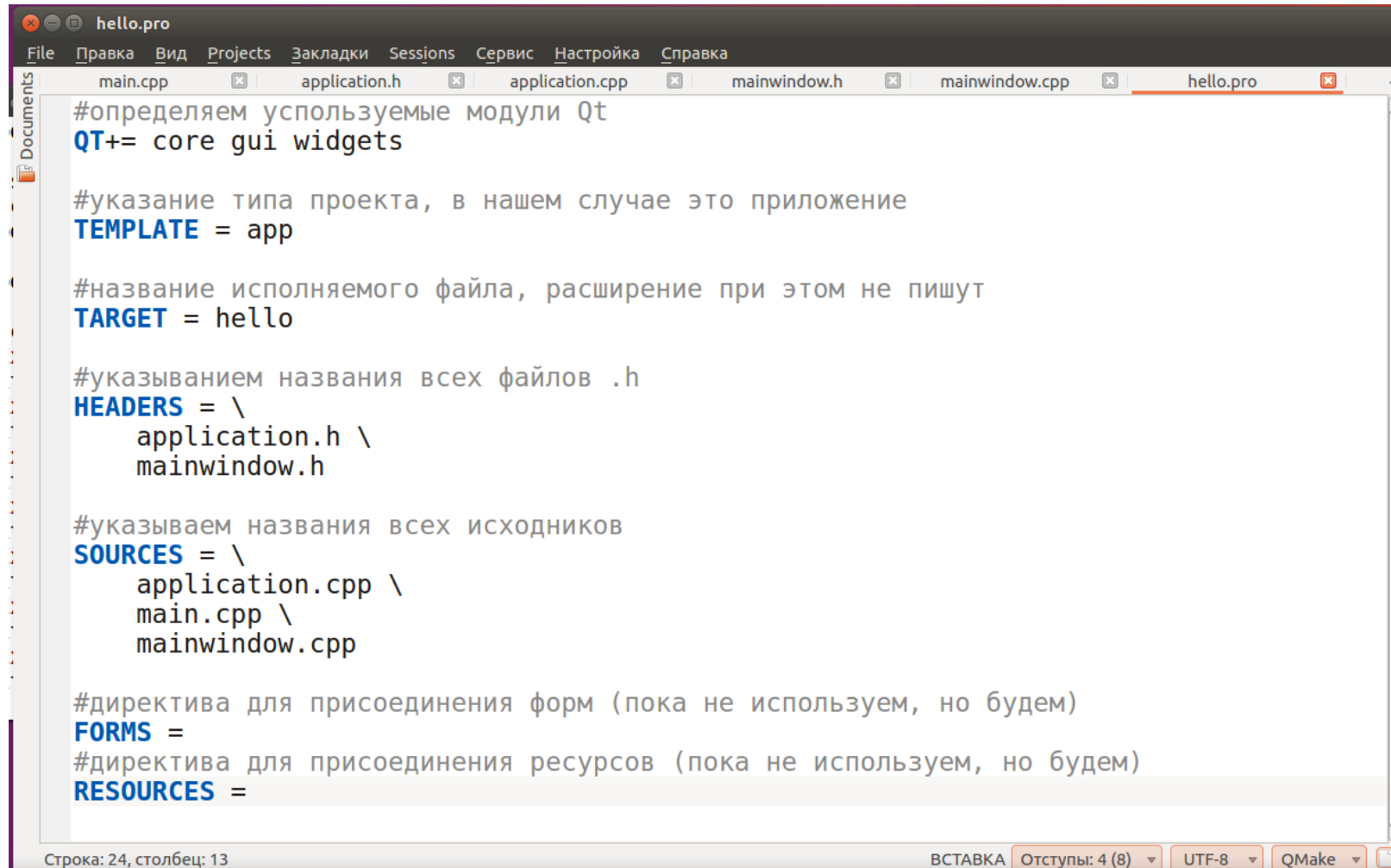
```
main.cpp
File  Правка  Вид  Projects  Закладки  Sessions  Сервис  Настройка  Справка
main.cpp  application.h  application.cpp  mainwindow.h  mainwindow.cpp  hello.pro

#include "application.h"
#include "mainwindow.h"

int main ( int argc, char *argv[]){
    HELLO::Application app (argc, argv); // Этот класс осуществляет контроль и
    управление приложением. Класс наследован от QApplication. Любая программа с
    графическим интерфейсом, использующая Qt должна создавать только один
    объект этого класса, и он должен быть создан до использования операций,
    связанных с пользовательским интерфейсом.
    HELLO::MainWindow win;
    win.show (); // После создания элементы управления Qt по умолчанию невидимы,
    и для их отображения необходимо вызывать метод show(). Объект win -
    основной управляющий элемент приложения, что позволяет завершить работу
    приложения при закрытии окна win.

    return app.exec(); // запуск app.exec() приводит в действие цикл обработки
    событий. Этот цикл передает получаемые от системы события на обработку
    соответствующим объектам. Он продолжается до тех пор, пока либо не будет
    вызван статический метод QApplication::exit(), либо не закроется окно
    последнего элемента управления. По завершению работы приложения метод
    QApplication::exec () возвращает значение целого типа, содержащее код,
    информирующий о его завершении.
}
```

# Файл описания проекта hello.pro



The screenshot shows a Qt IDE window titled 'hello.pro'. The menu bar includes 'File', 'Правка', 'Вид', 'Projects', 'Закладки', 'Sessions', 'Сервис', 'Настройка', and 'Справка'. The tab bar shows 'main.cpp', 'application.h', 'application.cpp', 'mainwindow.h', 'mainwindow.cpp', and 'hello.pro'. The editor displays the following QMake project file content:

```
#определяем используемые модули Qt
QT+= core gui widgets

#указание типа проекта, в нашем случае это приложение
TEMPLATE = app

#название исполняемого файла, расширение при этом не пишут
TARGET = hello

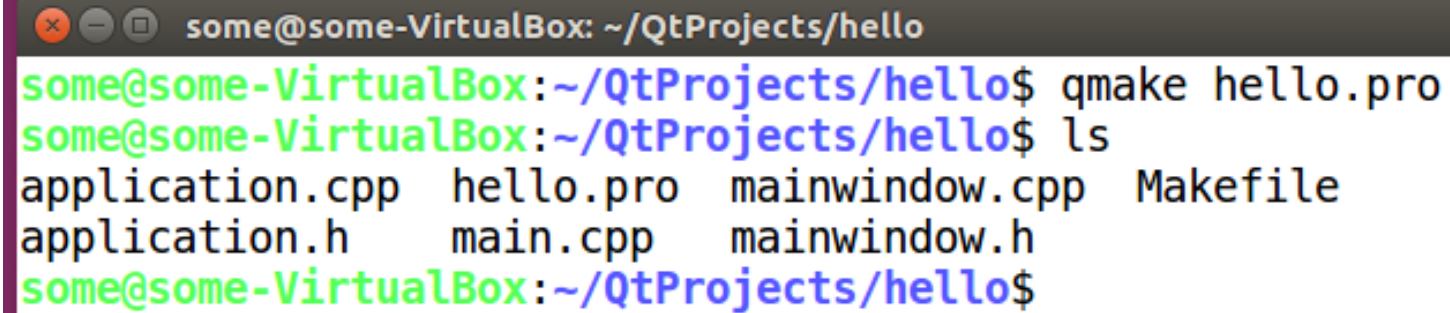
#указыванием названия всех файлов .h
HEADERS = \
    application.h \
    mainwindow.h

#указываем названия всех исходников
SOURCES = \
    application.cpp \
    main.cpp \
    mainwindow.cpp

#директива для присоединения форм (пока не используем, но будем)
FORMS =
#директива для присоединения ресурсов (пока не используем, но будем)
RESOURCES =
```

The status bar at the bottom indicates 'Строка: 24, столбец: 13', 'ВСТАВКА', 'Отступы: 4 (8)', 'UTF-8', and 'QMake'.

Для создания make-файла проекта, в директории с проектом необходимо выполнить команду `$qmake имя_файла_проекта.pro`



```
some@some-VirtualBox: ~/QtProjects/hello
some@some-VirtualBox:~/QtProjects/hello$ qmake hello.pro
some@some-VirtualBox:~/QtProjects/hello$ ls
application.cpp  hello.pro  mainwindow.cpp  Makefile
application.h    main.cpp  mainwindow.h
some@some-VirtualBox:~/QtProjects/hello$
```

Для создания исполняемого файла выполнить команду \$make

```
some@some-VirtualBox: ~/QtProjects/hello
application.h  main.cpp  mainwindow.h
some@some-VirtualBox:~/QtProjects/hello$ make
g++ -c -m64 -pipe -O2 -Wall -W -D REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++-64 -I. -I/usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. -o application.o application.cpp
g++ -c -m64 -pipe -O2 -Wall -W -D REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++-64 -I. -I/usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. -o main.o main.cpp
g++ -c -m64 -pipe -O2 -Wall -W -D REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++-64 -I. -I/usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. -o mainwindow.o mainwindow.cpp
/usr/lib/x86_64-linux-gnu/qt4/bin/moc -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++-64 -I. -I/usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. application.h -o moc_application.cpp
g++ -c -m64 -pipe -O2 -Wall -W -D REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++-64 -I. -I/usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. -o moc_application.o moc_application.cpp
/usr/lib/x86_64-linux-gnu/qt4/bin/moc -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++-64 -I. -I/usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. mainwindow.h -o moc_mainwindow.cpp
g++ -c -m64 -pipe -O2 -Wall -W -D REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++-64 -I. -I/usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. -o moc_mainwindow.o moc_mainwindow.cpp
g++ -m64 -Wl,-O1 -o hello application.o main.o mainwindow.o moc_application.o moc_mainwindow.o -L/usr/lib/x86_64-linux-gnu -lQtGui -lQtCore -lpthread
```

В случае успешной сборки проекта, для запуска программы выполнить  
\$./имя\_проекта

