بسم الله الرحمن الرحیم

گفتار نویسنده :

این کتاب بر گرفته از آموزش های خوب آقای کامبیز اسدزاده است طی صحبتی که با ایشون شد قرار شده که این کتاب را به صورت pdf انتشار بدیم این کتاب کاملا رایگان است

نویسنده : پژمان ابرا هیم پور(adonis27)

ایمیل نویسنده: pejman_ebrahimpor@yahoo.com

ایمیل آقای کامبیز اسدزاده: kambiz@nanosoftco.com

سایت رسمی : www.nanosoftco.com

اگر مشکلی بود با ایمیل های بالا تماس بگیرید تا در اولین فرصت اصلاح شود اگر هم غلط املایی هست به بزرگی خودتون ببخشید

مایلم از امروز طبق تعاریف و لیست بندی ذکر شده در ادامه توضیحات آموزش هایی رو برای برنامه نویسی ارائه بدیم که مایل بودم در وب سایت خودم اقدام به انتشار آموزش ها کنم که گفتم در انجمن Qt با C++ های برنامه نویس باشه بیشتر مورد استفاده قرار میگیره که انشالله همینطور هم خواهد بود
هدف از این آموزش ها آشنایی با امکاناتی هست که در QTمیتونیم استفاده کنیم که شامل کد نویسی کمتر / خروجی و طرح های بیشتر و در نهایت استفاده در پلتفرم های مورد نظر...

حال من لیستی از آموزش ها رو در نظر دارم که به صورت زیر شروع به آموزش خواهم کرد.

مقدمات و معرفی Qt برای شروع کار

راهنمایی برای دانلود مناسبترین نسخه ازQt

معرفی و کار با Signal و Slot ها

معرفی و کار با نمایشWindows

معرفی و کار با لایه ها زبانه ها و بدنه های در طراحی

معرفی و کار با قابلیت های HTML و CSS در طراحی

معرفی و کار با لایه های افقی و عمودی

معرفی و کار با لایه های Grid در طراحی فرم

معرفی و کار با جدا کننده هاSplitter

معرفی و کار با دایرکتوری ها

معرفی و کار با فایل ها / خواندن و نوشتن در آن ها

معرفی و کار با برچسب هاLabel

معرفی و کار با دکمه هاButton

معرفی و کار با کنترل ورودیLineEdit

معرفی و کار با چک باکسCheckBox

معرفی و کار باRadioButton

معرفی و کار با/Combobox

معرفی و کار با لیست ها/ListWidget

معرفی و کار با لیست های درختی/TreeWidge

معرفی و کار با Action ها

معرفی و کار با Slider و Progress ها

معرفی و کار با Statusbar در فرم

معرفی و کار باMessageBox

معرفی و کار باTimer

معرفی و کار با Thread ها

معرفی و کار باMap

معرفی و کار باHash

معرفی و کار با / String رشته ها

معرفی و کار با الگوریتم های مرتب سازی

معرفی و کار با الگوریتم های کپی کننده

معرفی و کار با الگوریتم های تنظیم کننده

معرفی و کار با الگوریتم های جستجو

معرفی و کار با Tooltip ها همراه با قابلیت های ویژه

معرفی و کار شبکه / دانلود فایل بر اساس پروتکل های HTTP وFTP

معرفی و کار با باینری و سریالیز کردن آبجکت ها

معرفی و کار با TextStream ها

مقایسه انواع حالت های کامپایل درQt

نحوه افزودن دیگر کتابخانه های C++ در محیط Qt Creator و استفاده همراه با کتابخانهQt

و اگه خدا بخواد بعد از به اتمام رسوندن این موارد میریم روی مباحث حرفه ای تر تا اطلاعات جامع تری در این رابطه در اختیار دوستان قرار بدیم.

نسخه Qt برای این آموزش خواهد بود Qt 5.2 : با کامپایلر MinGW و.MSVC2012

همچنین آموزش های لازم برای ارتباط برقرار کردن دو IDE قدرتمند C++ یعنی Qt و VS2012 و ۲۰۱۳ رو انشاالله آموزش خواهم داد.

سیستم عامل مناسب : لینوکس Ubuntu و ویندوز ۷ و ۸

خب مرحله ۱ : مقدمات و معرفی Qt برای شروع کار:

ابتدا لازمه توضیحاتی در رابطه با اینکه QTچیه و چرا باید ازش استفاده کنیم رو میدم:

همانطور که می دانید امروزه توسعه نرم افزار و به روز رسانی های آن در انواع پلتفرم ها از قبیل **Linux , Windows , Mac OS X** و همچنین پلتفرم های موبایلی و تبلتی از قبیل **Andoird , IOS ,**

Backberry و ... با سرعت بسیار زیادی دنبال می شود ؛ همچنین آرزوی اکثر برنامه نویسان این است که یک زبان ویژه با تمامی قابلیت ها و مهمتر از همه پشتیبانی از **Objective Oriented** و **Performance** بالا رو همراه با یک **IDE**همه کاره و جذاب در اختیار داشته باشند که در این صورت به جای تجربه کردن تمامی محیط های برنامه نویسی در این زمینه ها پیشنهاد میکنم خیلی راه دوری نروید زیرا با استفاده از محیط برنامه نویسی Qt که پشتشم یه زبان غولی مثل **C++** واستاده تقریبا همه آرزوهای شما در برنامه نویسی فراهم می شود.

در رابطه با توانایی ها و قدرت زبان C++ آشنا هستید اینکه یک زبان مادر (پایه) است و خود مستقیم بدون متکی و وابسته بودن یه به سیستم عامل یا زبان برنامه نویسی دیگری کار خودش رو انجام می دهد , در این میان طی این سال ها که زبان های برنامه نویسی پیشرفته بسیار زیادی رو داشتند طوری که برنامه نویسان به راحتی میتوانند توسط زبان های مثل C# و ... برنامه های مورد نظر خود را در کمترین زمان با محیط و گرافیک قابل توجهی تولید کنند که تمامی این موارد باعث شده بود زبان C++ از نظر برنامه نویسان بی حوصله یک زبان سخت و خواستار حال حوصله درست حسابی از طرف برنامه نویس می باشد بنا بر این دلایل سختی و همچنین ساده نبودن طراحی توسط این زبان شاید دلیلی برای کم رنگ شدن این زبان در طراحی و تولید فرم ها و خروجی های قوی بود که ذتا هر کسی که با این زبان واقعا کار کرده میداند که چنین نیست ! زیرا به تنهایی تکمیل و جامع است فقط وقت و کد نویسی بیشتری برای تولید خروجی های مشابه در زبان های دیگری همچون C# را دارد ؛ حال چکار کنیم ؟ آیا با این حال برنامه نویسی با سرعت کم در این زبان خواهان خودش رو داره ؟

جواب سوال رو اینگونه پاسخ میدم : اولا نیازی نیست به کد نویسی زیاد و سخت چون همونطور که زبان های دیگه در طی این سالها پیشرفت کرده زبان C++ هم خالی از پیشرفت نبوده و نسبت به قبل بسیار توانمند و خود کفاست , در این میان نه تنها در رابطه با قابلیت ها موارد زیادی در نسخه های ۱۱ این زبان رفع و توسعه داده شده است در کنار این IDE بسیار جذابی که به داد برنامه نویسان و مشتاقان این زبان اومده به نام ! Qt با آوای (کی یوت) یا کیوت؛

همه چیز ساده تر , روانتر و جذابتر شده و سرعت برنامه نویسی و طراحی فرم ها و قالب بندی های پیشرفته که قبلا نیاز به کد نویسی های بسیار زیادی داشت بسیار بهتر از قبل شده به طوری که به جرئت میتونم بگم من

خودم که بسیار سخت پسند هستم در نگاه اول کار با Qt رو پسندیدم , این محیط بر خلاف محیط های VS به هیچ عنوان سیاست انحصاری بودن رو ندارد و فقط ویندوز نیست که از قدرتمندترین IDE این دوره زمونه پشتیبانی میکند بلکه سیستم عامل های قدرتمند یونیکسی مثل Linux Ubuntu و Mac OS X این محیط برنامه نویسی خارق العاده رو کاملا پشتیبانی میکنن و این در ابتدای کار به تنهایی ارزشمند است.

از قابلیت هایی که نمیشه ازشون به این راحتی چشم پوشی کرد می توان به قابلیت Cross platform بودن برنامه های تولید شده توسط C++/Qt اشاره کرد که شما به راحتی میتونید خروجی رو در سیستم عامل مورد نظرتون دریافت و کامپایل کنید حتی سیستم عامل هایی مثل IOS و Android که امروزه برنامه نویسی موبایل هم یجورایی بازارش گرمه

قابلیت ها در طراحی : قابلیت طراحی فوق العاده با QML و همچنین پشتیبانی از CSS و HTML یکی دیگه از مزایای Qt هستش که میتونید برنامتون رو بترکونید مثلا فرض کنید یک فرم طراحی میکنید در حالت عادی خالی از style و افکت های ویژه هست برای این کار میتونید با استفاده از HTML و CSSبرنامتون رو به طور شگفت انگیزی طراحی کنید.

استفاده از قابلیت های HTML یکی از بهترین و جذابترین مواردی میتونه باشه که در برنامه نویسی Desktopو Mobile خیلی خیلی جذاب خواهد بود همه این قابلیت ها دست به دست هم میدن تا برنامه نویس سختی های C++ رو فراموش کنه و با علاقه تمام برنامش رو طراحی و در نهایت ری هر پلتفرمی که دوست داره کامپایل کنه.

یه چکیده کلی از تعریف این محیط : دیگه نیازی نخواهد بود برین سراغ Objective-C برای تولید نرم افزار های IOS و MAC OS X یا نیازی نیست برید سراغ Java برای تولید نرم افزار های Android یا نیازی نیست برید سراغ زبان های بدبختی مثل VB یا C# برای تولید برنامه های ویندوزی ! و در نهایت نیازی نیست فقط مجبور باشید برای صنعت سخت افزار از C++ استفاده کنید بلکه با داشتن علم یک زبان قدرتمندی مثل C++ میتونید با یک تیر چندین نشان رو همچین بزنید که خدا بداند که چه شود!!!

یادم نره بگم که میتونید طراحی وب هم داشته باشید هامنظورم از همه کاره بودن یعنی واقعا همه کارست

پس شد یاد گیری زبان **C++**و آشنایی با محیط **Qt**که یک نوع فریم ورک ویژه ای برای این زبان است برنامتون رو بترکونید حالا با در نظر داشتن اینکه اطلاعات HTML , CSS , QML , JavaScript رو هم دارید دیگه چه بهتر هرطور که مهربونتون می خواد مانور بدین که در نهایت منجر به یک خروجی مقابل میشود : قدرت , سرعت , کیفیت , ارتباط مستقیم با سخت افزار ! و در کنار این محیط با کیفیت بالا و همچنین طراحی مدرن همه و همه در خروجی نهایی برنامه شما حس خواهد شد.

مرحله ۲ : راهنمایی برای دانلود مناسبترین نسخه ازQt

قبل از شروع کار بهتره توضیحاتی در رابطه با اینکه کدام ویرایش از Qt رو باید دانلود کنید رو بدم به صورت زیر:

نکته :برای استفاده از Qt ما دو روش داریم که در این دو روش یکیش رایگان هست و یکیش شامل هزینه ای در رابطه با لیسانس IDE هستش , در رابطه با این دو مورد باید بگم هیچ تفاوتی آن چنانی بین این دو نسخه پولی و رایگان وجود نداره به جز نوع کامپایل اون که در نسخه پولی شما میتونید خروجی static بگیرید و تنها در صورت پرداخت هزینه میتونید برنامتون در حالت Static رو به فروش برسونید , فعلا میریم سراغ نسخه تحت مجوز LGPL که به شما اجازه کامپایل به صورت داینامیکی رو میده.

حالا از کجا دانلود کنیم و کدوم نسخه رو باید دانلود کنیم به صورت زیر هستش:

ما در بخش دانلود سایت رسمی Qt انواع مختلفی از ویرایش های اون را میبینیم که در لینک رسمی موجود هست:

http://qt-project.org/downloads

قبل از همه چیز باید سیستم عاملی که شما روی اون کار میکنید رو مشخص کنید هرچند الان ۹۹ درصد روی ویندوز کار میکنید ولی وقتی دیگه اسم C++میاد وسط تورو خدا ولش کنید این سیستم عامل رو برید روی Linuxیکم به خودتون تغییر تحولات بدین هم از لحاظ روحی تاثیر داره هم تجربه میکنید و این خوبه.

به طور کلی معنی عنوان لینک های قابل دریافت به صورت مقابل زیر معنی میشود:

مثلا Qt 5.3.0 for Windows 64-bit (VS 2013, OpenGL, 589 MB) : یعنی چی ؟

خب Qt 5.3.0 که نام و نسخه برنامه هست **for Windows** همون نوع سیستم عامل هست که روش باید نصب کنید **64-bit** معماری سیستم عامل نصب شده هستش دقت کنید شاید CPU شما ۶۴ بیتی باشه ولی اگه نسخه 32بیتی سیستم عامل رو نصب کردین باید در این حالت ۳۲ بیتی Qt رو دریافت و نصب کنید VS 2013 !و یا MinGW 4.8 نوع کامپایلر همراه با Qt هستش و مورد آخر اگه **OpenGL** رو نسخه شما باشه یعنی Qt شما از OpenGL هم پشتیبانی میکنه.

حالا بریم سر اصل مطلب:

در صورتی که از **سیستم عامل ویندوز** استفاده میکنید نسخه های **Qt 5.3.0 for Windows** رو باید دریافت و نصب کنید.

در صورتی که از **سیستم عامل لینوکس (Ubuntu)** استفاده میکنید نسخه های **Qt 5.3.0 for Linux**رو باید دریافت و نصب کنید.

در صورتی که از **سیستم عامل مکینتاش (Mac OS X)** استفاده میکنید نسخه های **Qt 5.3.0 for Mac**رو باید دریافت و نصب کنید.

حالا توضیحات در باره جزئیات نسخه ها برای دانلود : نسخه های ۳۲ بیتی همون معماری x86 هستند و نسخه های ۶۴ بیتی هم همون مهماری x64 هستند.

در حالت عادی شما میتونید روی سیستم عامل های دسکتاپی کار کنید و برنامتون رو کامپایل کنید ولی اگه نیاز به کامپایل برنامه روی سیستم عامل های IOS و Androidدارید باید نسخه های زیر رو دانلود کنید :

برای Android میتونید از نسخه Qt 5.3.0 for Android Windows برای استفاده در محیط **ویندوز** دانلودش کنید.

برای Android میتونید از نسخه Qt 5.3.0 for Android for Linux برای استفاده در محیط **لینوکس** دانلودش کنید.

برای IOSحتما باید سیستم عامل مک داشته باشید و نسخه Qt 5.3.0 for iOS رو دانلود کنید , همچنین میتونید Qt 5.3.0 for Android رو هم برای کامپایل برای اندروید در محیط مک دانلود کنید و یا هر دویه اینهارو میتونید به صورت Qt 5.3.0 for Android and iOSدریافت کنید.

یک نکته : چون سیستم عامل IOS و Mac بدتر از مایکروسافت انحصار طلب هستند به همین خاطر برای کامپایل برنامه های IOS و Mac حتما باید سیستم عامل مک داشته باشید تا بتونید برنامه های مربوط به Appleرو کامپایل کنید.

توجه : لازم نیست شما برای هر کدام از پلتفرم ها دوباره نویسی و کد نویسی مجدد انجام بدین خیالتون راحت شما میتونید برای شروعبرای ویندوز یا لینوکس یه نسخه مورد نظر رو دریافت کنید و برنامتون رو بنویسید در نهایت که دیدین نیاز هست روی سیستم های مثل IOS و MAC یا Android کامپایل و آزمایش کنید اونوقت بهتره نسخه مربوطه رو دریافت و پروژه خودتون رو به وسیله نسخه مورد نظرتون Import و کامپایل نمایید.

حالا در این آموزش من قصد دارم از نسخه های ویندوزی استفاده کنم تا راحت تر بتونید ازش استفاده کنید.(در رابطه با نصبش در ویندوز توضیحاتی نمیدم چون راحته ولی برای لینوکس بعد از اتمام آموزش ها دستورات و عملیات لازم رو خواهم گفت که چطور هستند)

خب برای شروع نسخه Qt 5.3.1 for Windows 32-bit (MinGW 4.8.2, OpenGL, 734 MB)

یا میتونید این رو دانلود کنیدQt 5.3.1 for Windows 32-bit (VS 2013, 626 MB)

فقط اگه نسخه VS2010 یا VS2012 و یا ۲۰۱۳ رو دانلود میکنید حتما باید Visual Studio مربوط به اون نسخه یا SDK های مربوطه روی سیستمتون نصب باشه.

من از کامپایلر **MinGW** استفاده میکنیم ذاتا مثل VS خیلی پر دردسر نیست برا همین!

دررابطه با لینوکس و مخصوصا **Ubuntu** هم به روش زیر عمل کنید :بر فرض اینکه فایل qt-opensource-linux-x64-5.3.0.runرو دانلود کرده باشید اگر قرارش بدین در پوشه Downloadsو دستور زیر رو تو Terminal بنویسید :

```
cd /home/USER/Downloads
```

به جای این USER که اینجا یوزر من هست یوزری که رو سیستم خودتون هست رو بنویسید البته دقت کنید قبلش باید توسط - su روت شده باشید!

بعد از اینکار تشریف ببرید روی فایل qt-opensource-linux-x64-5.3.0.run که تو پوشه Downloadsهستش راست کلیک و Properties کنید و در زبانه Permissions اون پایین گزینه Executeرو تیک بزنید و بعد برید ترمینال دستور زیر رو بنویسید...

```
./qt-opensource-linux-x64-5.3.0.run
```

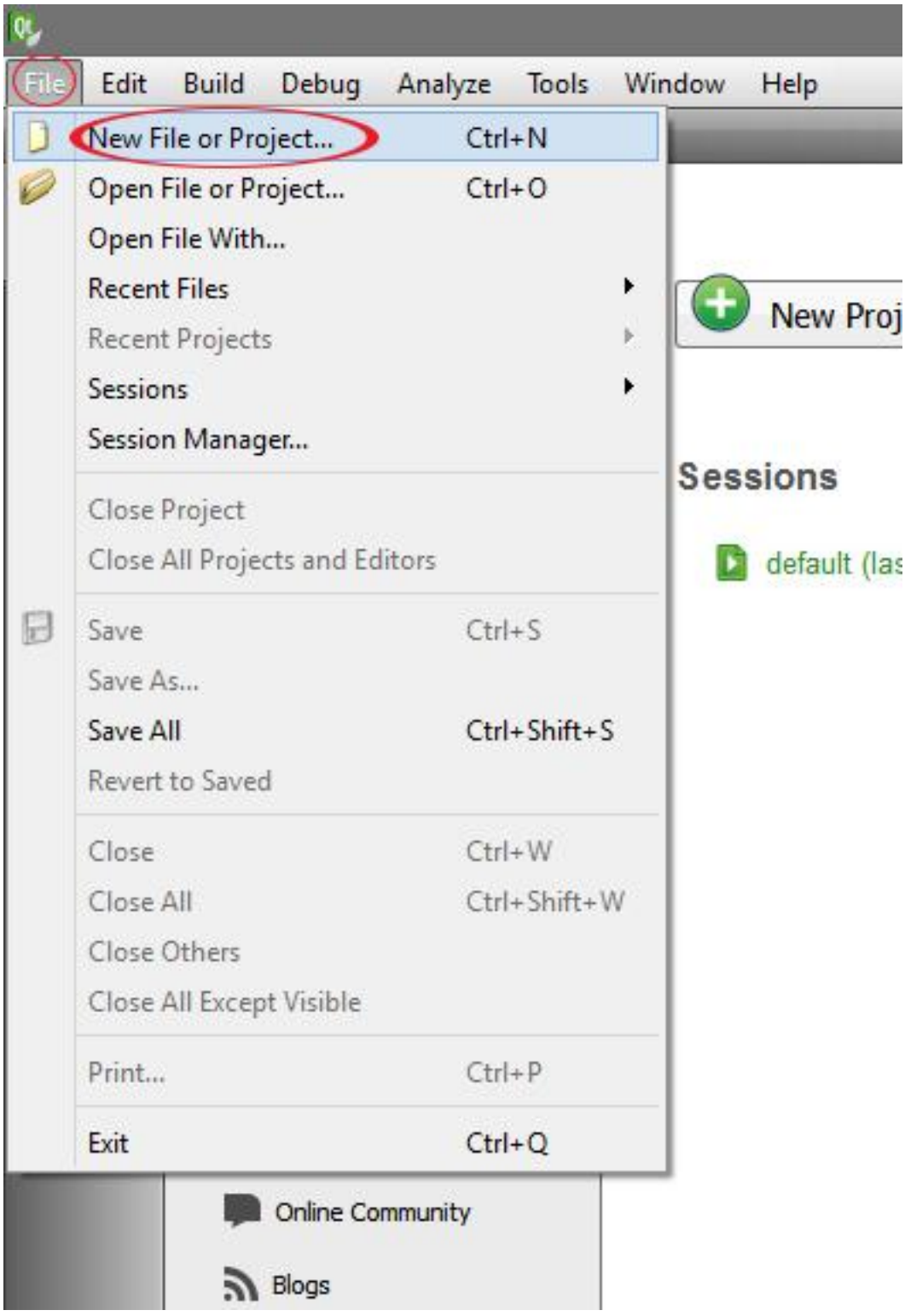
دستور جاری یه چیزی به این صورت هستش:

```
root@USER-Aspire-4720Z:/home/USER/Downloads# ./qt-opensource-linux-x64-5.3.0.run
```

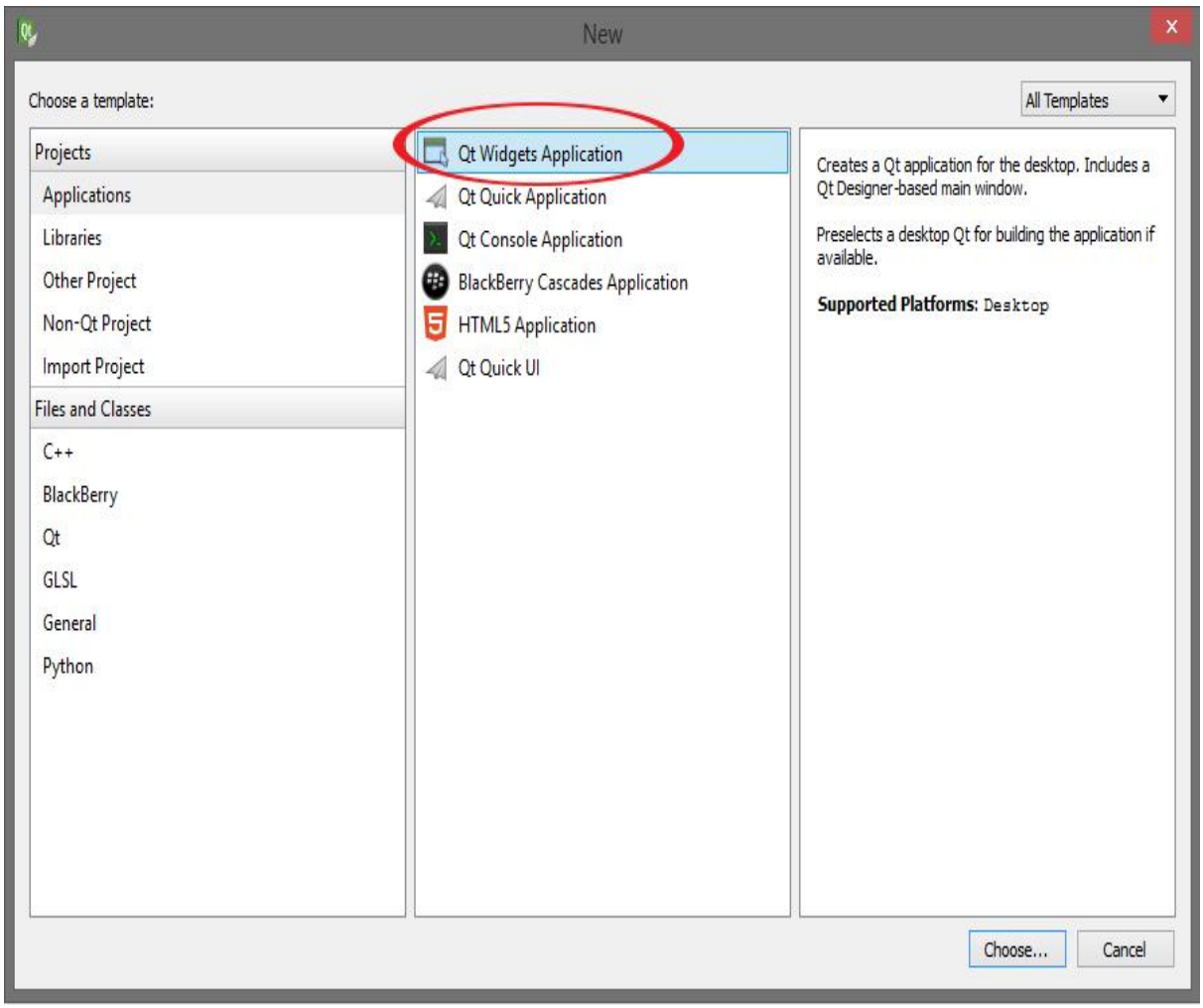
و در مراحل بعدی **Wizard**برای نصب میاد بالا و بقیه مراحلش همانند آموزش های ویندوزی هستش.

میریم سر اصل مطلب و کار با Qt و توضیحات در رابطه با انواع پروژه ها

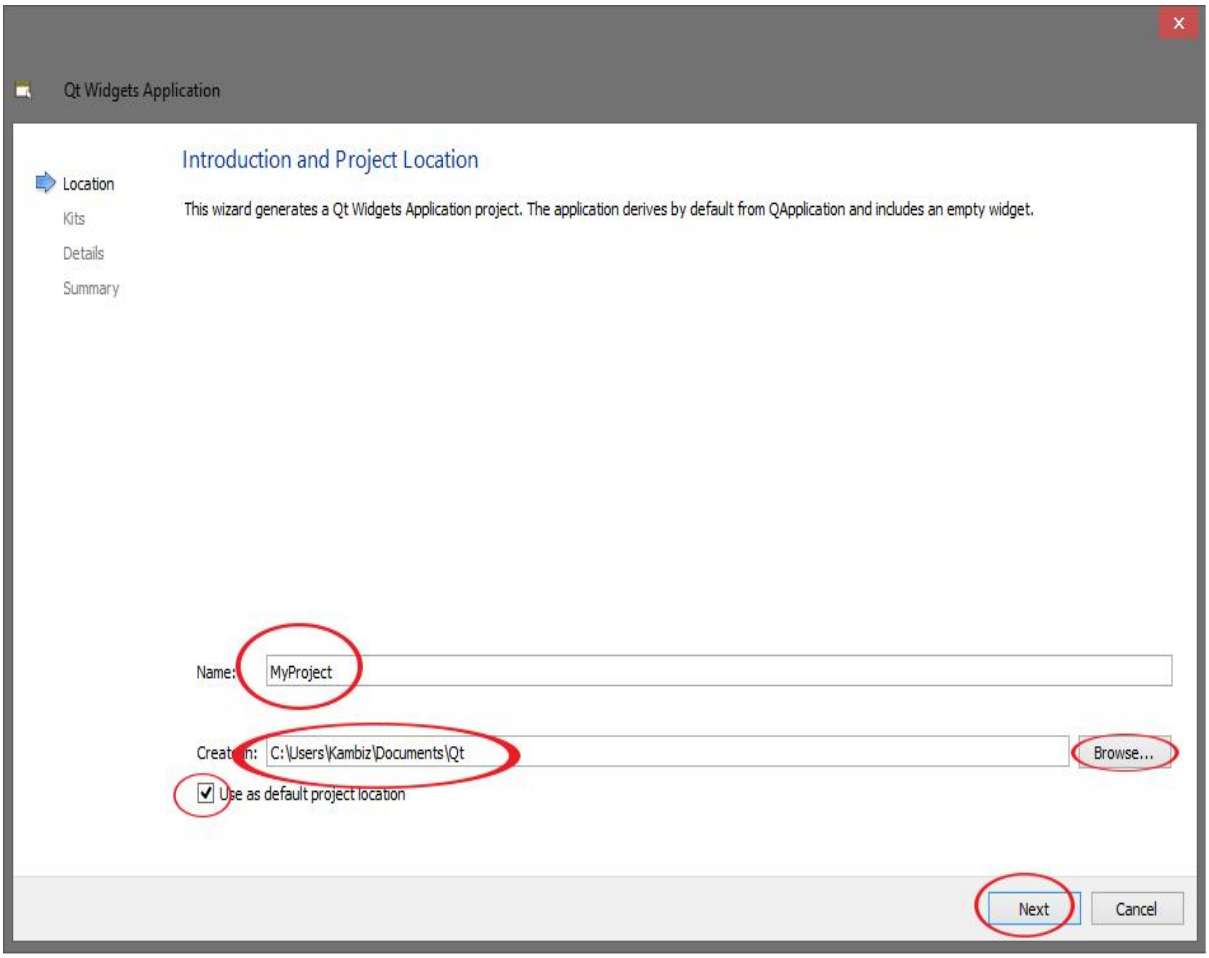
برای شروع به منوی File و گزینه New File or Project مراجعه کنید.



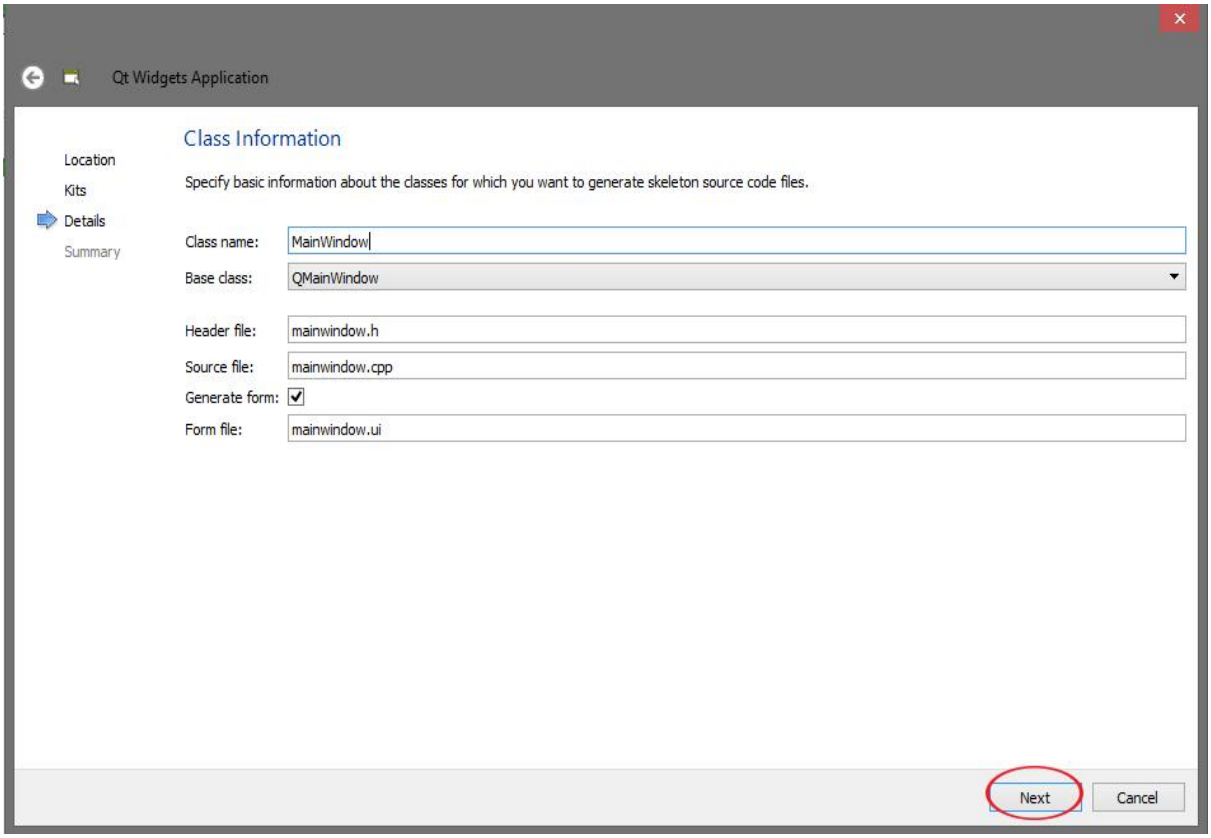
حالا در این قسمت شما انواع پروژه ها و فایل هایی که میتونید توسط Qt ایجاد کنید با توجه به نسخه ای که نصب کردین فعال و قابل انتخاب هستش که من در این اموزش ها از پروژه Qt Widgets Application استفاده میکنم و تقریبا میشه گفت استانداردترین حالت پروژه برای طراحی فرم هست با بقیه موارد از قبیل Qt Consoleیا ... فعلا کاری نداریم.



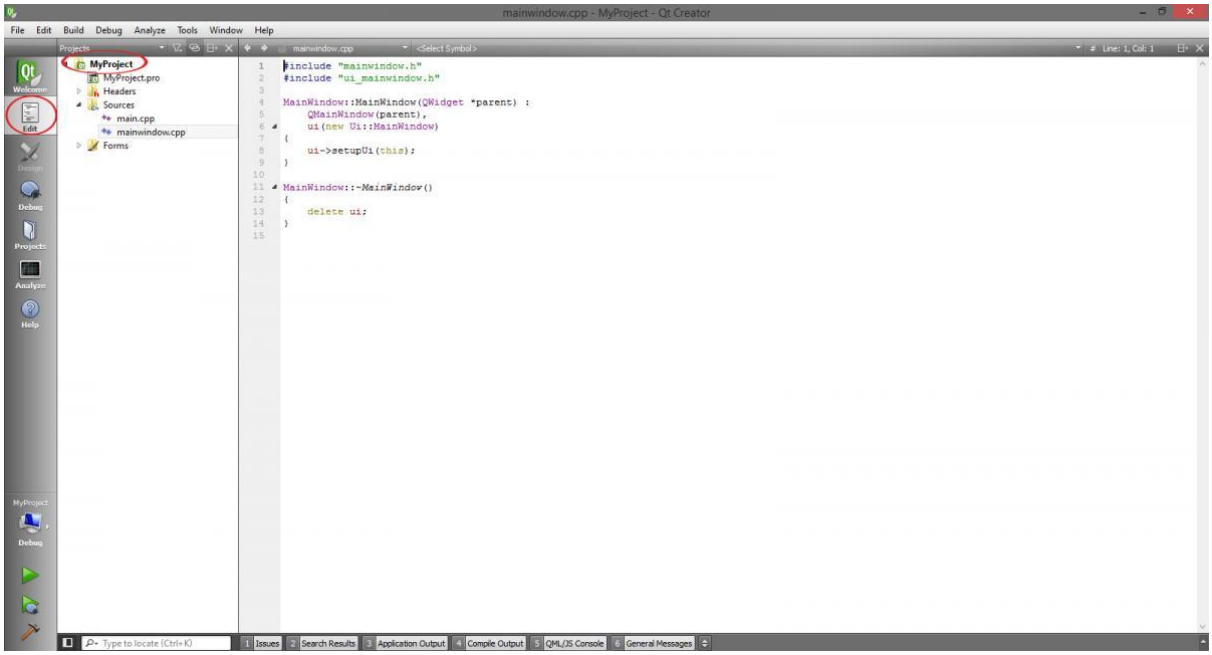
حال با ادامه این مرحله به صورت زیر نام پروژه و مسیری که مایل هستید پروژه در اون مکان ذخیره بشه رو انتخاب کنید



خب در این مرحله شما میتونید نوع کلاس و همچنین هیدر هارو مشخص کنید طبق تصویر زیر ادامه بدین:



در نهایت تایید کنید تا پروژه شما به صورت تصویر زیر ایجاد بشه:



خب تا اینجا ما فقط یه پروژه از نوع QtWidgets ایجاد کردیم که شامل Main Window و کلاس و هیدر های استاندارد برای شروع کار و برنامه نویسی هستش.

ببینید در پروژه های Qt ما چند نوع فایل پروژه ای داریم با پسوند های pri. و pro.که هر دوی این فایل ها توسط Qt قابل شناسایی هستند و به عنوان فایل اصلی پروژه شما در نظر گرفته میشوند.

اگر روی فایل MyProject.pro کلیک کنید به صورت زیر شامل کد هایی هستش:

```
#-------------------------------------------------
#
# Project created by QtCreator 2014-01-15T08:19:15
#
#-------------------------------------------------

QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = MyProject
TEMPLATE = app


SOURCES += main.cpp\
        mainwindow.cpp

HEADERS  += mainwindow.h

FORMS    += mainwindow.ui
```

حالا من به صورت زیر توضیحاتش رو میدم تا متوجه وجود این کد ها بشید ! فراموش نکنید فایل pro.

مهمترین قسمت پروژه هستش که در فراخوانی فایل ها و رفرنس ها مهمه مثلا من اگه بخوام از دیتابیس و دستورات SQL استفاده کنم در این قسمت باید اول فراخوانیش کنم.

```
#-------------------------------------------------
#
# Project created by QtCreator 2014-01-15T08:19:15 ( پروژه ساخته شده توسط کیوت در تاریخ
فلان)
#
#-------------------------------------------------

QT       += core gui  ( این قسمت مربوط به فراخوانی موارد لازم هست یه چیزی مثل رفرنس گیری در ویژوال استدیو که به صورت
کد نوشته میشه)

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets  (مربوط به نسخه ویدجت در کیوت هستش)

TARGET = MyProject  (نام پروژه رو به صورت Namespace که در ویژوال استدیو هم دیدین فراخوانی میکنه)
TEMPLATE = app  (قالب بندی)


SOURCES += main.cpp\
        mainwindow.cpp  (سورس اصلی که شامل تابع main هستش رو فراخوانی میکند)

HEADERS  += mainwindow.h  (هیدر پنجره اصلی رو فراخوانی میکنه)

FORMS    += mainwindow.ui  (فایل و قسمتی که شامل طراحی هستش رو فراخوانی میکند)
```

خب بعد از این فایل ما چند پوشه به نامهای Sources , Headers و Forms داریم که به صورت زیر توضیحاتشونو میدم.

پوشه Headers وظیفه نگه داری تمام فایل های C++ از نوع h. یا همان heder رو بر عهده دارد.

پوشه Sources وظیفه نگه داری تمام فایل های C++ از نوع cpp. یا همان Source رو بر عهده دارد.

پوشه Forms وظیفه نگه داری تمام فایل های مربوط به طراحی رو داره پسوند فایل های طراحی در کیوت uiهستند به صورت mainwindow.ui که فایل دیزاین پروژه شما به عنوان یک فرم در نظر گرفته شده است.

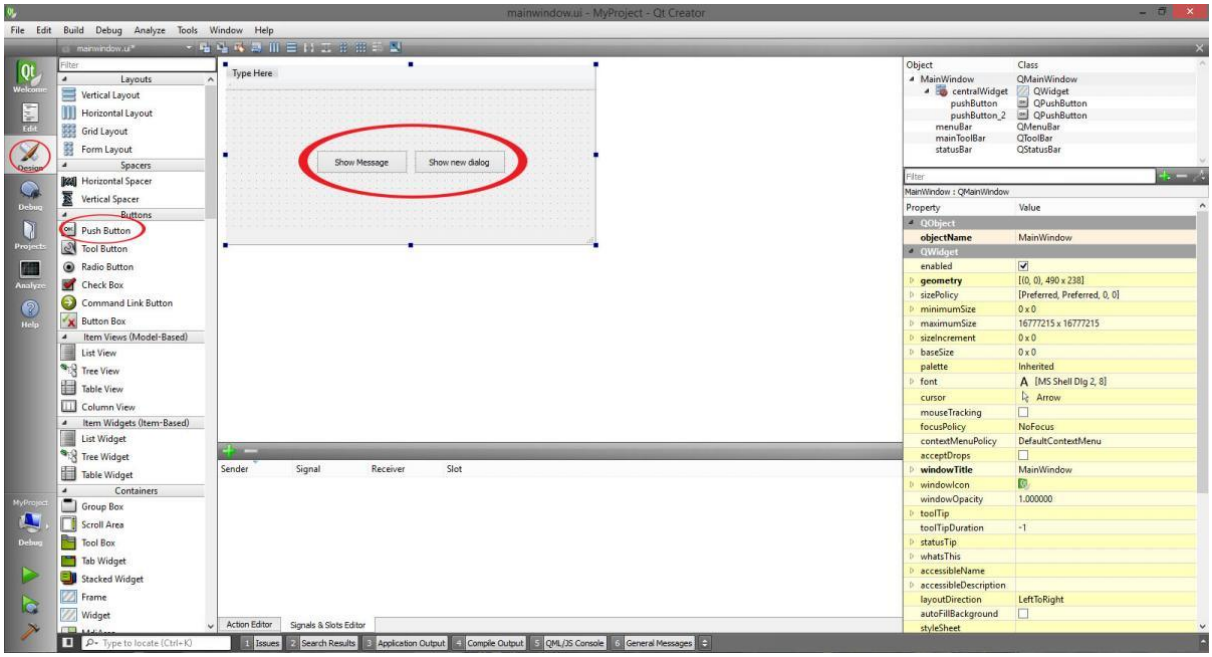
مرحله سوم : معرفی و کار با Signal و Slot ها و Event ها

نکته : من قصدم اینه که در ساده ترین حالت آموزشرو ارائه بدم به خاطر اینکه افراد شاکی از کد نویسی زیاد C++کمی دیدگاهشون رو در رابطه با این زبان تغییر بدن بد نیست... مخصوصا اونایی ک میگن کی حوصله داره تو C++ هزار خط کد بنویسه تا یه پیغام نمایش بده!!!

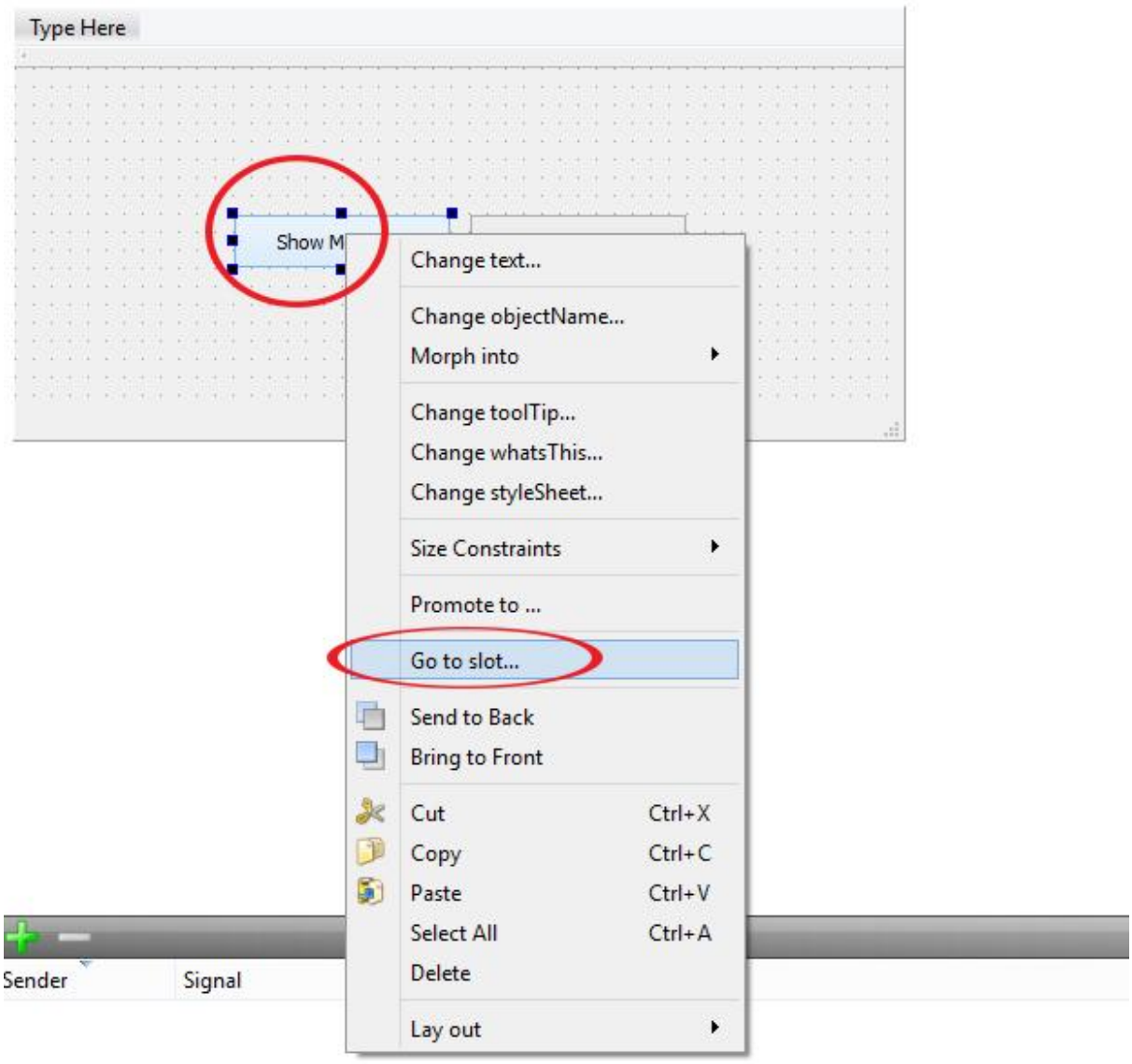
سیگنال یا اسلات ها چین ؟ به چه دردی میخورن ؟ مسلما شما برای ایجاد یک رخداد در یک زمان با اثر گذاری بر روی یک کنترل یا Object انتظار اینو دارین که اون شیء در قبال فشرده شدن یا هر رویداد دیگه ای یک عکس العملی نشون بده مثلا روی یک دکمه ای کلیک میکنید انتظار دارید یه پیغامی در رویداد Clickedاون نمایش داده بشه منظور از Signal و Slot در رابطه با این مسائل هستش که حالا در این میان برای صدا زدن از Signal استفاده میکنیم و از Slot برای دریافت دستور و عمل کردن.

من قصد دارم برای شروع کار روی یک فرمی ۲ تا دکمه ایجاد کنم و روی این دوتا دکمه از۲ رویداد (Clicked)کلیک شدن استفاده کنم.

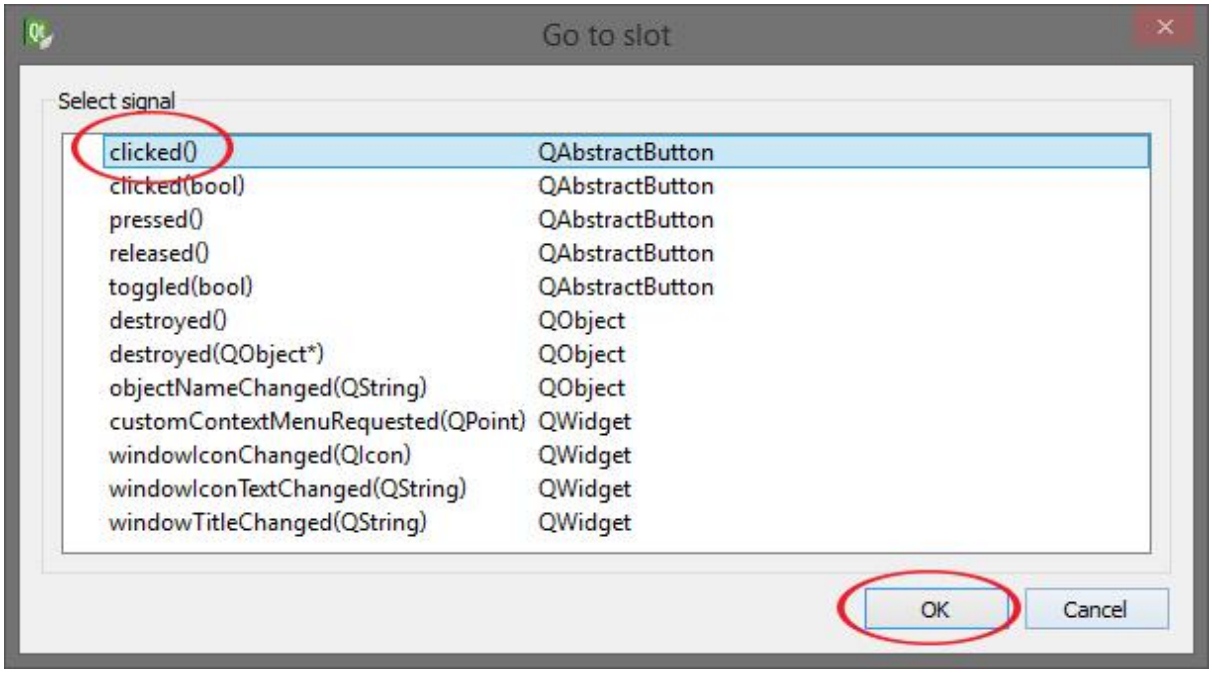
خب به قسمت Forms در پروژه برید و روی فایل mainwindow.ui دوبار کلیک کنید تا محیط طراحی فرمتون رو ببینید به صورت زیر دو تا دکمه با عمل Drag and Drop روی فرم خودم ایجاد میکنم در صورتی که کامپایل و اجرا کنید دکمه های ایجاد شده رو روی فرمتون میبینید ولی بدون هیچ عملی !



بر خلاف Visual Studio و زبان C# که روی Object ها کلیک میکنید و تابع مربوط به رویداد اون ساخته میشه و شما شروع میکنید به برنامه نویسی در Qt چنین چیزی وجود نداره در کل توی C++ کلیک کد کد بنویس بی معنیه باید کوهو بکنی تا بتونی دستور بدی به رویداد ها.... ولی من یه روش ساده تری رو بهتون میگم به صورت زیر روی کنترل راست کلیک میکنیم و گزینه Go to Slot رو انتخاب میکنیم.



خب حالا طبق تصویر زیر مشاهده میکنید که تمام رخداد های مربوط به کنترل مورد نظر شما قابل انتخاب هستند ما در این بخش لازمه از رویداده Click استفاده کنیم پس روش کلیک کنید و برید مرحله بعد.



خب حالا اینجا تابع مربوط به عمل کلیک شدن روی این دکمه ایجاد شده.

```
void MainWindow::on_pushButton_clicked()
{

}
```

در فایل mainwindow.h کد زیر اضافه شده که تعریف Slot مربوط به کنترل میباشد.

```
private slots:
    void on_pushButton_clicked();
```

منظور از pushButton نام همان دکمه ای هستش که روش کلیک میشه مسلما نام دکمه بعدی pushButton_2خواهد بود ! شما میتونید این تابع رو بدون عملیاتی که در تصویر دیدین ایجاد و به هر یک از کنترل های خودتون اختصاص بدین.

حالا این بخش مشخص کرد که ایجاد Event های مربوط به کنترلر ها چطور صورت میگیره به صورت کد نویسی این روش هارو میشد بریم ولی خواستم ساده ترین حالتش رو نشونتون بدم تا عوض اونهمه کد نویسی در C++به صورت عادی در بیاد.

برای اینکه این بحث رو یجورایی کامل کنم میخوام یه پیغامی رو با کلیک روی این دکمه نمایش بدم پس طبق قوانین C++ ابتدا هیدر Message رو فراخوانی میکنم که در کیوت هستش QMessageBox به صورت زیر:

```
#include "qmessagebox.h"
```

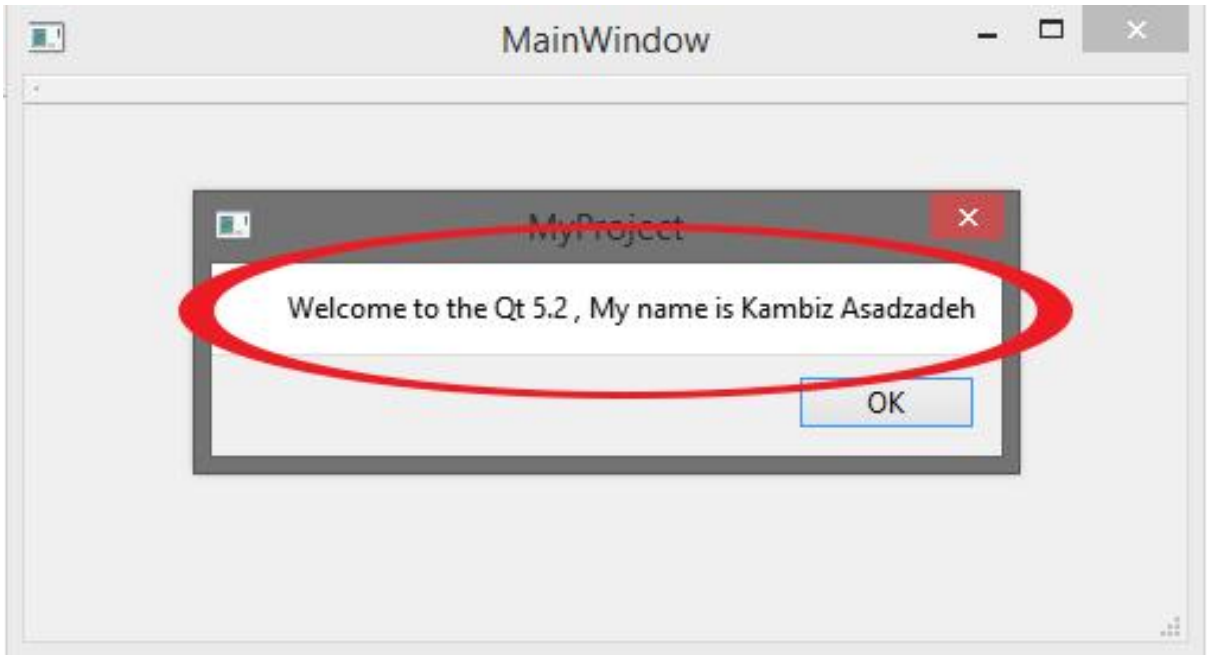
و کد زیر رو در داخل تابع رخداد کلیک مینویسم:

```
    QMessageBox msgBox;
    msgBox.setText("Welcome to the Qt 5.2 , My name is Kambiz Asadzadeh");
    msgBox.exec();
```

کاملش هست:

```
void MainWindow::on_pushButton_clicked()
{

    QMessageBox msgBox;
    msgBox.setText("Welcome to the Qt 5.2 , My name is Kambiz Asadzadeh");
    msgBox.exec();

}
```

خب حالا باید بعد از کامپایل و اجرا هنگام کلیک روی دکمه Show Message خروجی بشه مثل زیر :



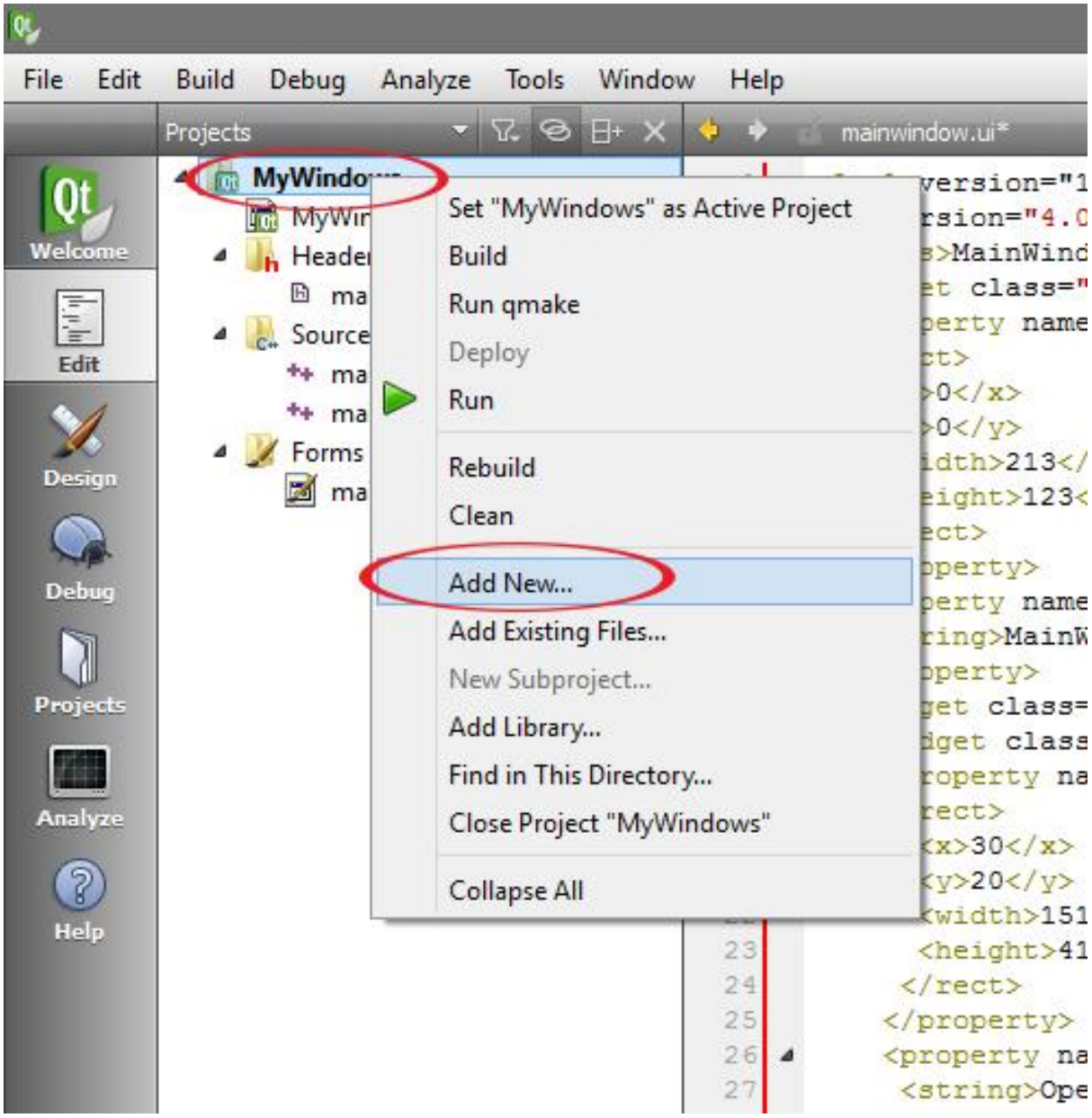
مرحله چهارم : معرفی و کار با نمایشWindows

قصد دارم در رابطه با پنجره ها یا همان دیالوگ ها یا در زبان Qt همون Widgets ها توضیحاتی بدم.

طبق روال عادی یک پروژه به نام MyWindows ایجاد میکنیم که در حالت عادی اگه توجه کنید پروژه دارای یک MainWindow یا همون فرم اصلی هستش ایجاد میکنه.

حال ما برای اینکه از فرم ها و دیالوگ های دیگه ای در پروژمون استفاده کنیم باید چکار کنیم ؟

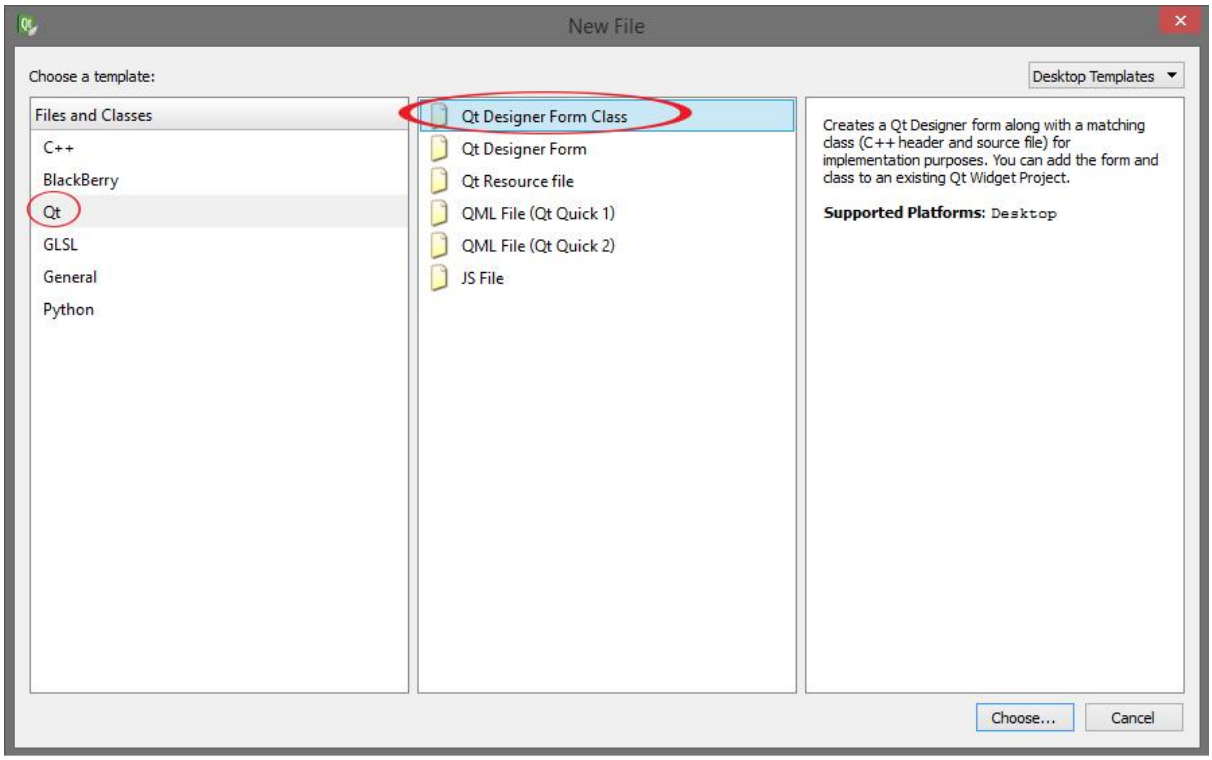
خیلی راحته برای این کار روی پروژه یعنی MyWindows راست کلیک میکنیم و گزینه Add new رو طبق تصویر زیر میزنیم.



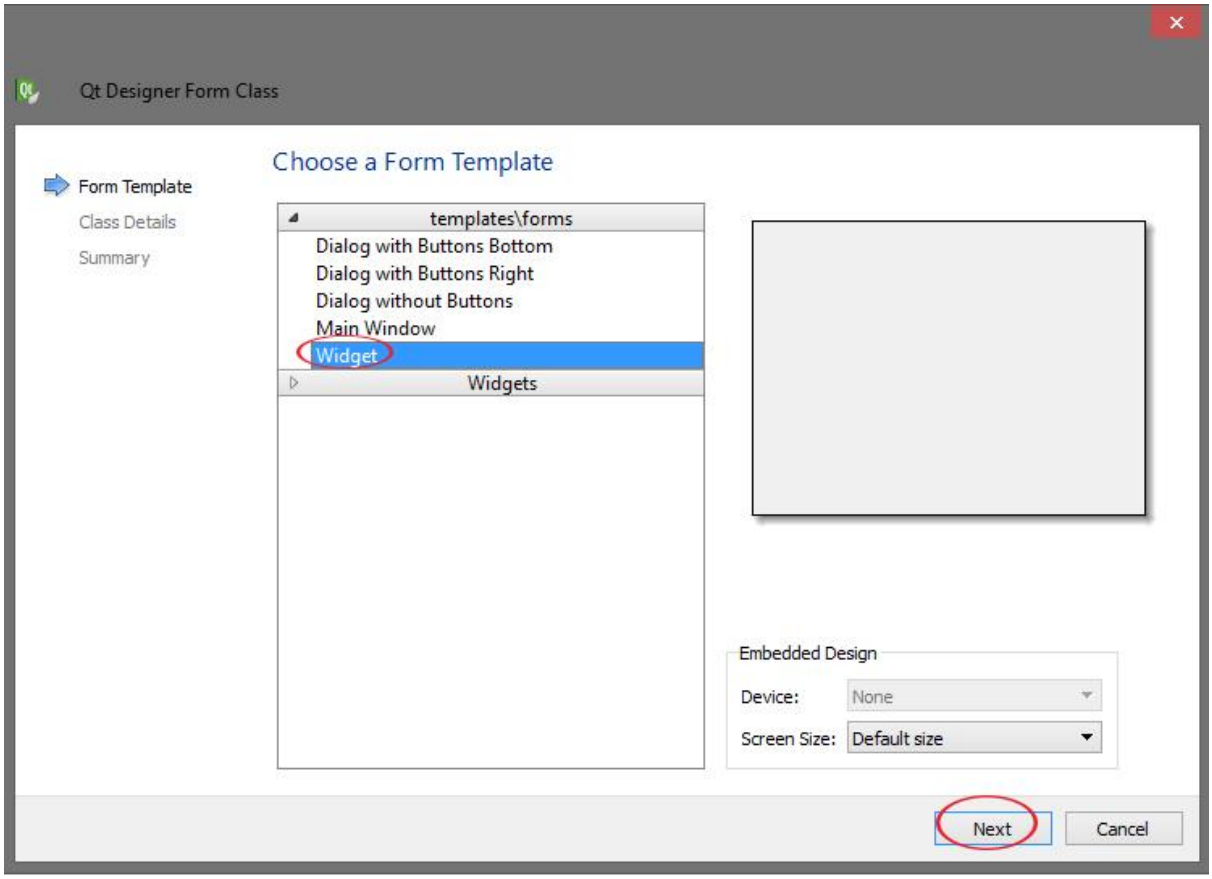
در مرحله بعد طبق تصویر زیر گزینه های متنوعی رو نسبت به نیاز کاری روی پروژه در اختیار قرار میده که ما هدفمون ساخت فرم یا همون پنرجره هست برای اینکار به شاخه Qt میرم که در اینا قبل از هر چیز بذارید یه توضیحی بدم...

انواع گزینه های مختلفی وجود داره کدومو باید انتخاب کنیم و چرا ؟ خب ببینید شما بر اساس نیاز اگر میخواهید فرم تولید شده شما دارای هیدر و کلاس اختصاصی خودش باشه میتونید در همین مرحله اون رو تعیین

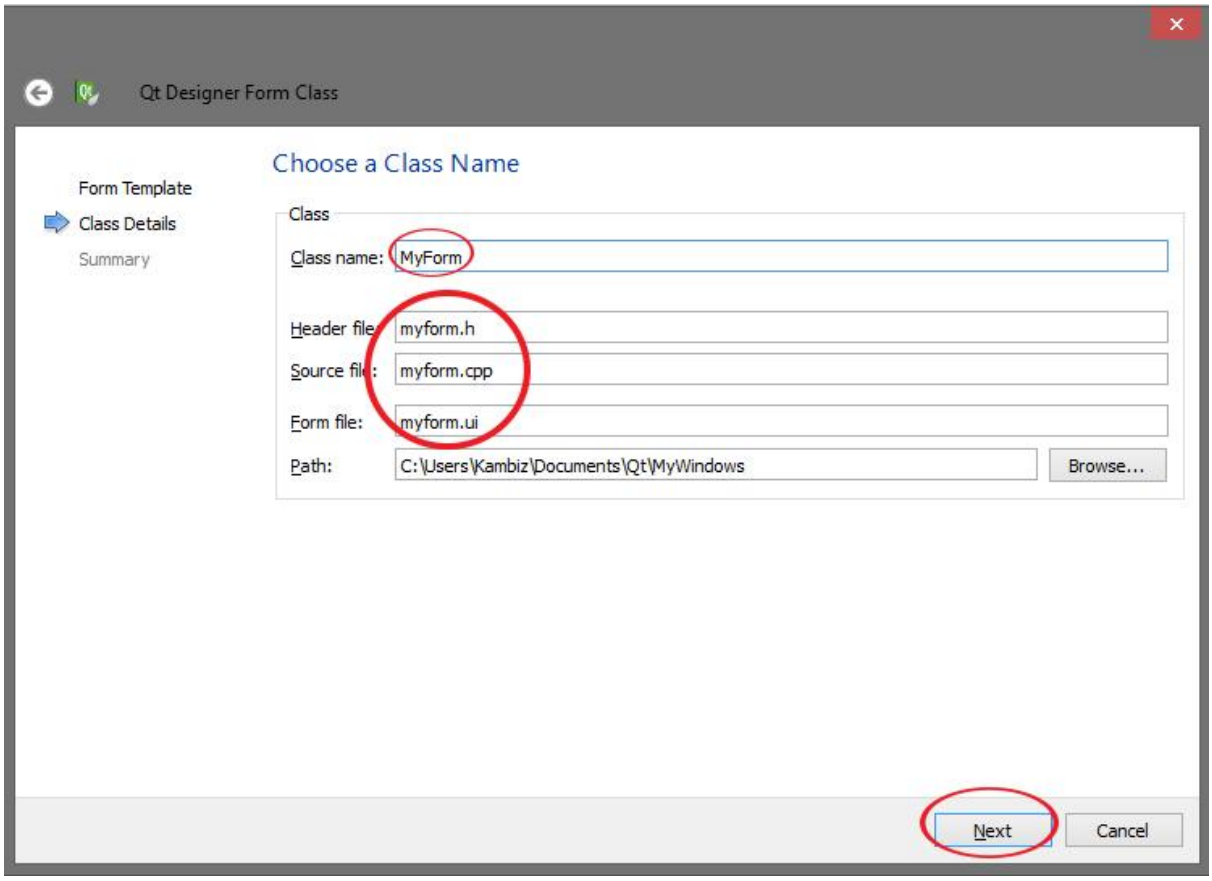
کنید که آیا هیدر و کلاس مخصوص فرم من ایجاد بشه یا خیر ! در اکثر موارد لازمه پس من از Qt Designer Form classاستفاده میکنم...



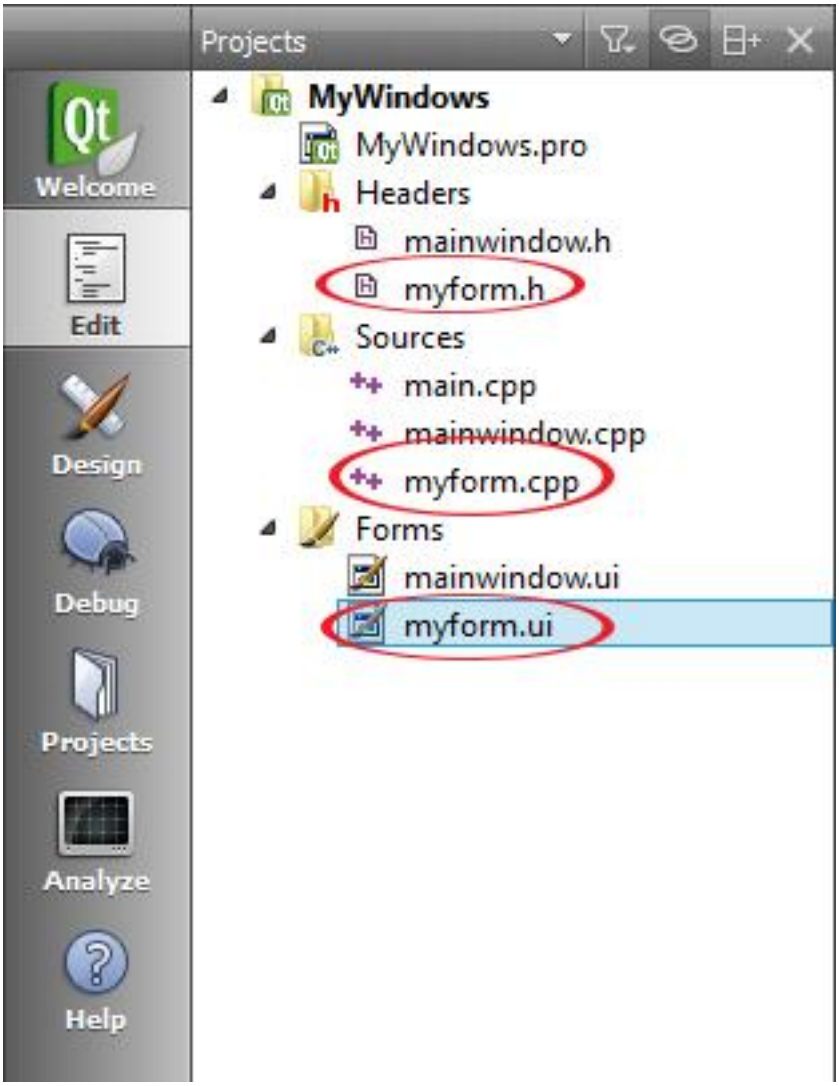
در مرحله بعد طبق تصویر زیر ما میتونیم نوع فرم رو سریعا انتخاب کنیم که که فرم دیالوگ باشه یا فرم معمولی باشه و یا فرم اصلی ؛ چون قصد ما استفاده از Widgets هستش من از نوع ویدجت یا همون فرم معمولی استفاده میکنم.



در مرحله بعد شما میتونید به صورت سفارشی نام کلاس , نام هیدر و نام فایل اصلی و فایل طراحی رو تعیین کنید که در خروجی بر اساس نام انتخاب شده شما مورد استفاده قرار میگیره که در این آموزش من اسم فرم خودم رو میزارم MyFormو به صورت زیر تایید میکنم.



حالا فرم ما ایجاد شده و ما میتونیم ازش استفاده کنیم برای اینکه مطمئن بشیم به تصویر زیر نگاهی کنید دقیقا طبق نام گذاری و انتخابی که کردم هم فایل طراحی و هم فایل های اصلی و هیدر همراه با کلاس های مربوط داخلی ایجاد شده اند.



خب حالا فرم ما تا اینجا ایجاد شده من میخوام فرمم رو داخل فرم اصلی توسط عمل کلیک شدن روی یک کنترل دکمه (Button) فراخوانی و نمایش بدم.

باید چکار کنیم ؟ من میخوام در فرم اصلی یه دکمه ای ایجاد کنم و در رویداد کلیک شدن اون فرم دوم خودم یعنی MyForm رو صدا بزنم تا Show بشه.

پس یا به صورت سریع از Got to Slot و کلیک استفاده میکنم روی دکمه ایجاد شده و یا در فایل هیدر فرم اصلی این کد رو مینویسم.

```
void MainWindow::on_pushButton_clicked()
{
```

```
}
```

در هر دو روش این کد رویداد کلیک رو ایجاد میکنه حالا من بخوام فرم دومم رو صدا بزنم چکار بدید کنم...

به این کد نگاه کنید:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_pushButton_clicked()
{



}
```

همه چیز حله ولی برای شناسایی فرم دوم باید یه کارایی کنم و بعد ازش کپی بسازم و اونو نمایش بدم.

پس ابتدا هیدر فرم دوم رو باید اینکلود کنم با این کد:

```
#include "myform.h"
```

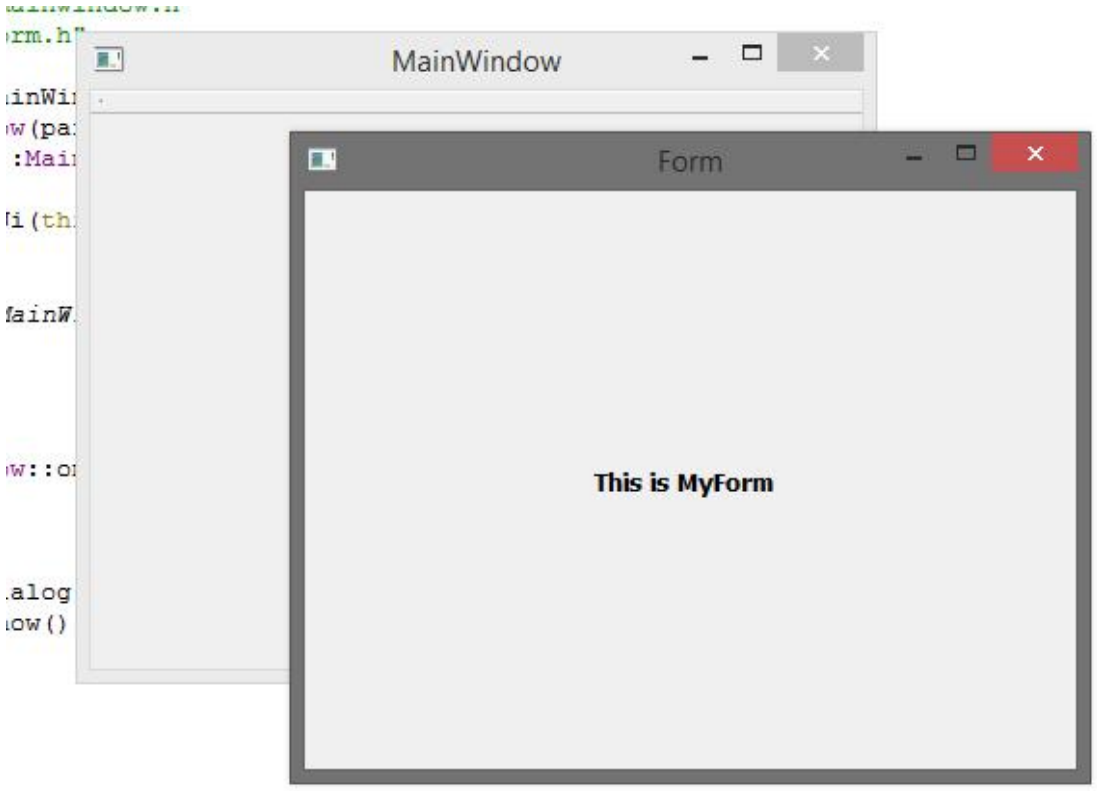
و در نهایت در تابع کلیک کد زیر رو مینویسم:

```
void MainWindow::on_pushButton_clicked()
{


    MyForm *dialog = new MyForm();
    dialog->show();

}
```

کد اصلی کلی میشه به صورت زیر :

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "myform.h"

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_pushButton_clicked()
{


    MyForm *dialog = new MyForm();
    dialog->show();

}
```

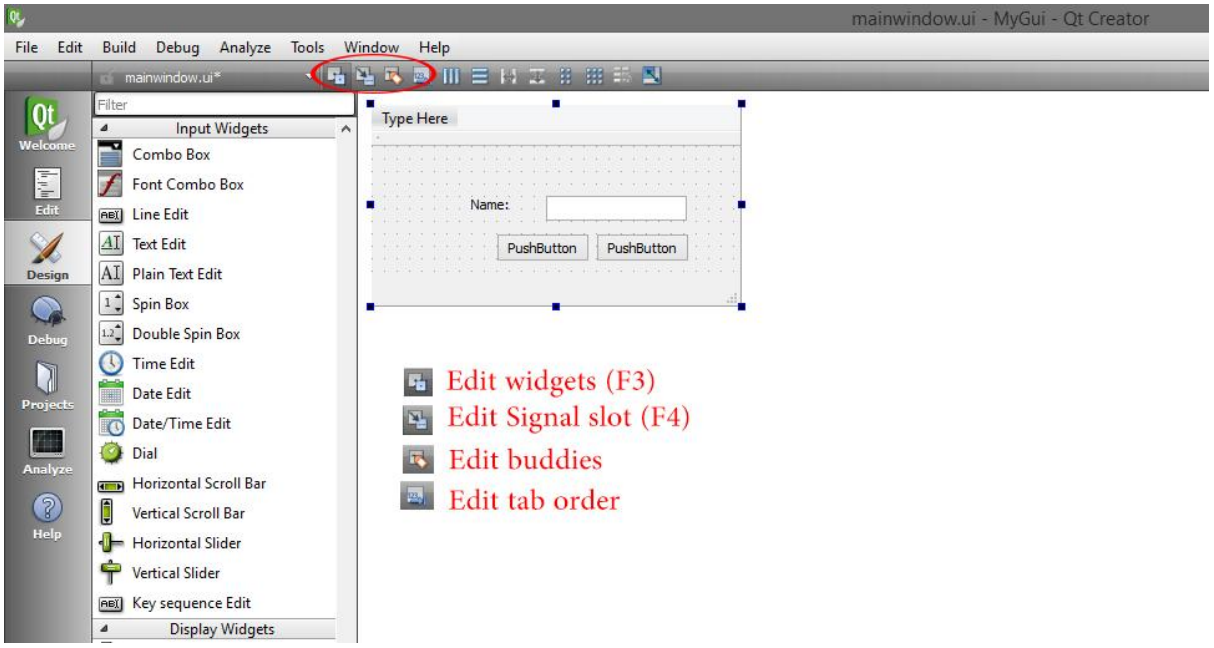
خروجی هم بعد از کلیک روی دکمه میشه به صورت زیر:



قسمت پنجم : معرفی و کار با لایه ها زبانه ها و بدنه های در طراحی

در این مرحله در رابطه با لایه های طراحی و امکانات توضیحاتی بدم که طبق روال قبلی پروژه ای ایجاد میکنیم به نام MyGui و میریم سر اصل مطلب...

خب طبق تصویر زیر ما یه قسمتی داریم که به صورت خیلی راحت امکانات لایه به لایه در اختیار ما میزاره:



ابتدا توضیحات قسمت اول رو به صورت زیر میدم:

گزینه : Edit Widgets در حالت کلی وقتی لازمه ابزار ها و کنترل هایی رو روی فرم قرار بدیم یا به طور ساده بخوام بگم وقتی نیاز باشه روی فرمتون شیئی رو درج کنید حتما باید در این لایه کار کنید.

گزینه : Edit Signals/Slots به صورت کلی این لایه وظیفه در اختیار گذاشتن سریع دسترسی های مربوط به Signal ها و Slot ها رو در اختیار شما میزاره که با عمل Drag و Drop اشیاء روی همدیگه مراحل بعدیش نمایان میشود.

گزینه : Edit buddies این لایه امکان تنظیم اندازه فرم رو براتون فراهم میکنه.

گزینه : Edit tab order در این لایه از طراحی شما میتونید به راحتی شماره ایندسک یا همان Index numberاختصاصی هر یک از اشیائ رو روی فرم مشخص کنید که با عمل Tab نیز بر اساس اولویت انتخاب می شوند.

یه نکته مهم در Qt وجود داره اونم اینه که عملیات Drag و Drop بر خلاف IDE های دیگه امکانات بسیار زیادی رو برای راحتی کار ایجاد میکنند.

مرحله ششم : معرفی و کار با قابلیت های HTML و CSS در طراحی

همانطور که میدونید امروزه داشتن گرافیک و طراحی های درست حسابی روی نرم افزار نویسی خودش یکی از مهمترین قسمت های تولید یک برنامه هستش که اکثرا به خاطر ای دلیل که C++ رو فقط یک محیط Consoleمیدونن سراغ این زبان نیومدن نمیدونم چرا شاید براشون سخته یا شایدم اطلاعات کافی ندارن...

ولی در کل اگه با زبان های راحتی مثل C# کار کردین میدونید که برای طراحی فناوری هایی مثل WPF رو دارند که بجای CSS از XAML استفاده میکنه و در نوبت خودش در طراحی کیفیت خوبی رو در خروجی ارائه میده.

ولی هرچی باشه خودتون میدونید طرح هایی که در برنامه های تحت وب تولید میشن همشون به کمک CSS و HTML هستش و همچنین JavaScript و ... حالا اگه این فناوری ها روی پلتفرم های دسکتاپی باشه چی میشه ؟ مسلما یکی من خودم خیلی خوشحال میشم و WPF رو میزارم کنار و میام سراغ همین.Qt

قبل از هرچیز یه تعریفی در رابطه به CSS و HTML بدم بعد بریم سر اصل مطلب....

فناوری CSS با آوای (سی‌اس‌اس :)(روشی ساده برای نمایش چیدمان و جلوه‌های تصویری (مانند نوع قلم، رنگ و اندازه‌ها) بر صفحه‌های وب است. شیوه‌نامهٔ آبشاری از جنس زبان‌های نشانه‌گذاری، با ساختار متن سادهٔ رایانه هستند و درون هرکدام، دستورهایی آبشار مانند و پی‌درپی، برای چگونگی نمایش هر صفحه وب افزوده می‌شود. به گفته‌ای ساده تر، این دستورها روش نشان داده شدن قلم‌ها و اندازه شان، رنگ‌ها و پس زمینه‌ها، روش چیدمان موزاییک‌های دربرگیرنده داده‌ها (دیواره‌ها)، و بسیاری دیگر از عنصرهای ساختار هر صفحه وب را، درون خود جای می‌دهند.

فناوری : HTML زبان امتدادپذیر نشانه‌گذاری فرامتن یا اکس‌اچ‌تی‌ام‌ال (Extensible HyperText Markup Language - XHTML)همان اچ‌تی‌ام‌ال است به همراه رعایت دقیق تمامی قواعد و دستورات نحوی نزدیک‌تر به زبان اکس‌ام‌ال که موجبات افزایش اطمینان از عمل‌کرد صحیح سندها در شرایط پیچیده‌تر موجود در اینترنت امروزین را فراهم می‌سازدXHTML .ها، نوع‌های سندها و ماژول‌ها در حال حاضر و در آینده هستند که در واقع زیر مجموعه و گسترش یافته HTML4 است. این نوع اسناد بر پایه

XMLهستند و برای کار در ترکیب با عامل کاربر مبتنی بر XML طراحی شده‌اند. و در حال حاظر نیز جدیدترین آن HTML5 هستش.

در رابطه با این مسئاله من خودم خیلی به این قسمت علاقه مندم دلیلشم شاید واضح باشه کار کردن با دستورات CSSو HTML در تولید صفحات وب خروجی خیلی قابل توجهی رو میده حال آنکه از این دستورات در تولید برنامه های تحت دسکتاپ استفاده کنیم چه شود !!! در اینجاست که باید بگم بنامتون رو از لحاظ گرافیکی میتونید منفجرش کنید.

سوال .... مگر HTML و CSS مختص طراحی وب نیست ؟ در جواب باید گفت بله... ولی از این به بعد نه همونطور که متوجه شدید در Qt ما قابلیت این رو خواهیم داشت از دستورات HTML و CSS در طراحی استفاده کنیم برای همینه که دارم میگم بهتر از این نمیشه

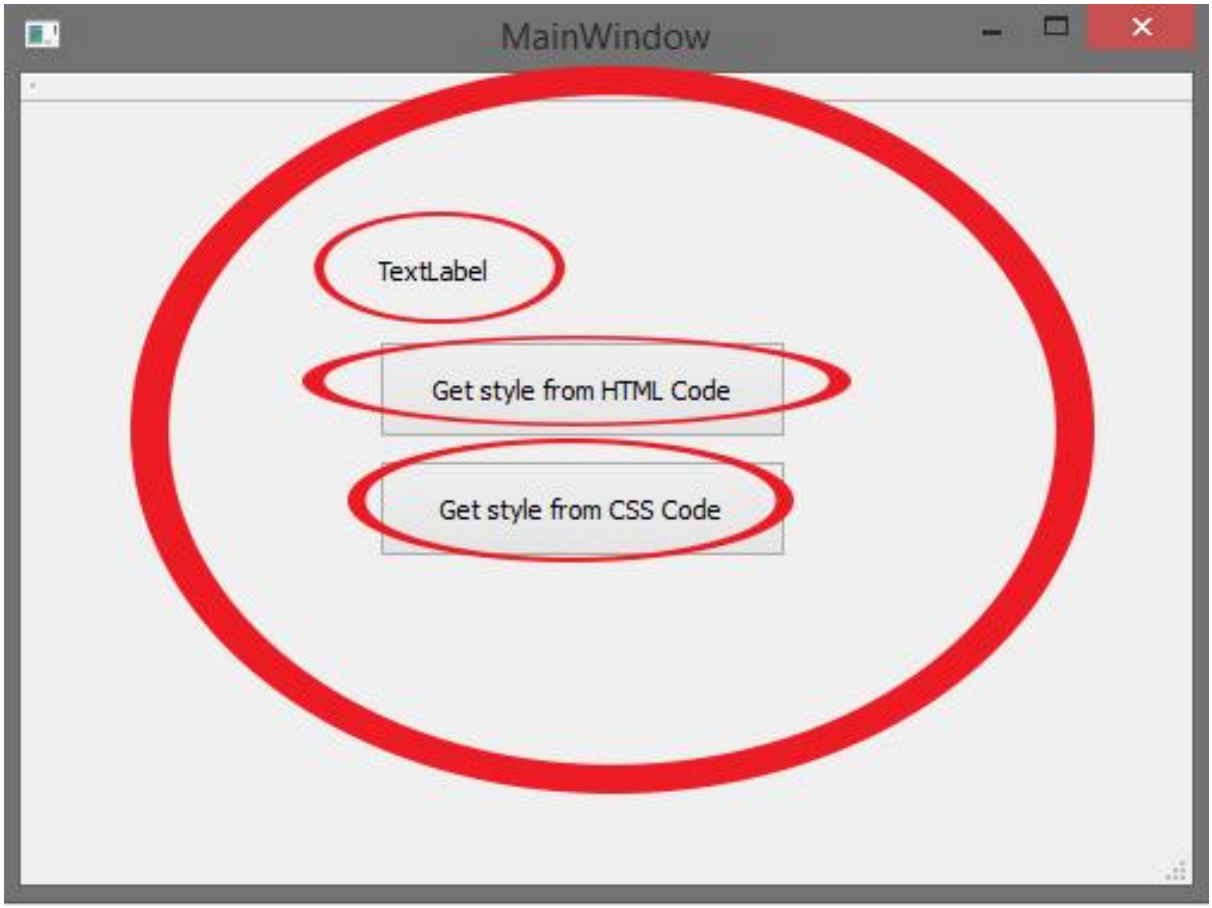
بریم سر پروژه ... مثل قبل یه پروژه ایجاد میکنیم با نام MyDesign و از نوع همون... Widgets

خب حالا برای شروع کار از HTML من میخوام شروع کنم که چطوری میشه دستورات HTML رو توی کد های C++ استفاده کرد.

فرض کنید من یه متنی هست در روی یه label یا هر شیئ دیگری میخوام توسط کد های HTML بهش ویژگی های خاصی بدم...

میام کد رو به صورت زیر در نظر میگیرم:

یه Label و ۲ تا دکمه ایجادمیکنم که در حالت عادی به صورت زیر دارای افکت و شکل ظاهرای استانداردی هستند که از استایل پیشفرض سیستم عامل تبعیت کردن ...



الا من میخوام از HTML استفاده کنم و دیگه این وابستگی رو از سیستم عامل بگیرم ... چکار میکنیم به صورت زیر باید دقت کنید ما برای دستور نویسی HTML باید بین کتیشن و دابل کتیشن از کد های HTML استفاده کنیم که درستش هم همینه به صورت زیر.

```
"<b>This is C++/Qt HTML Code</b>"
```

میبینید این کد در بین تگ های دستور bold که با اغاز کننده و خاتمه دهنده شروع شده قرار گرفته و تمامی اونهارم در بین "" و یا ' قرار میدیم تا قابل شناسایی برای کامپایلر باشه.

حالا ما چطور باید این نوع نویسه رو روی یک لیبل ستش کنیم اینم خیلی راحت میشه به صورت زیر انجامش داد و خروجی رو مشاهده کرد.

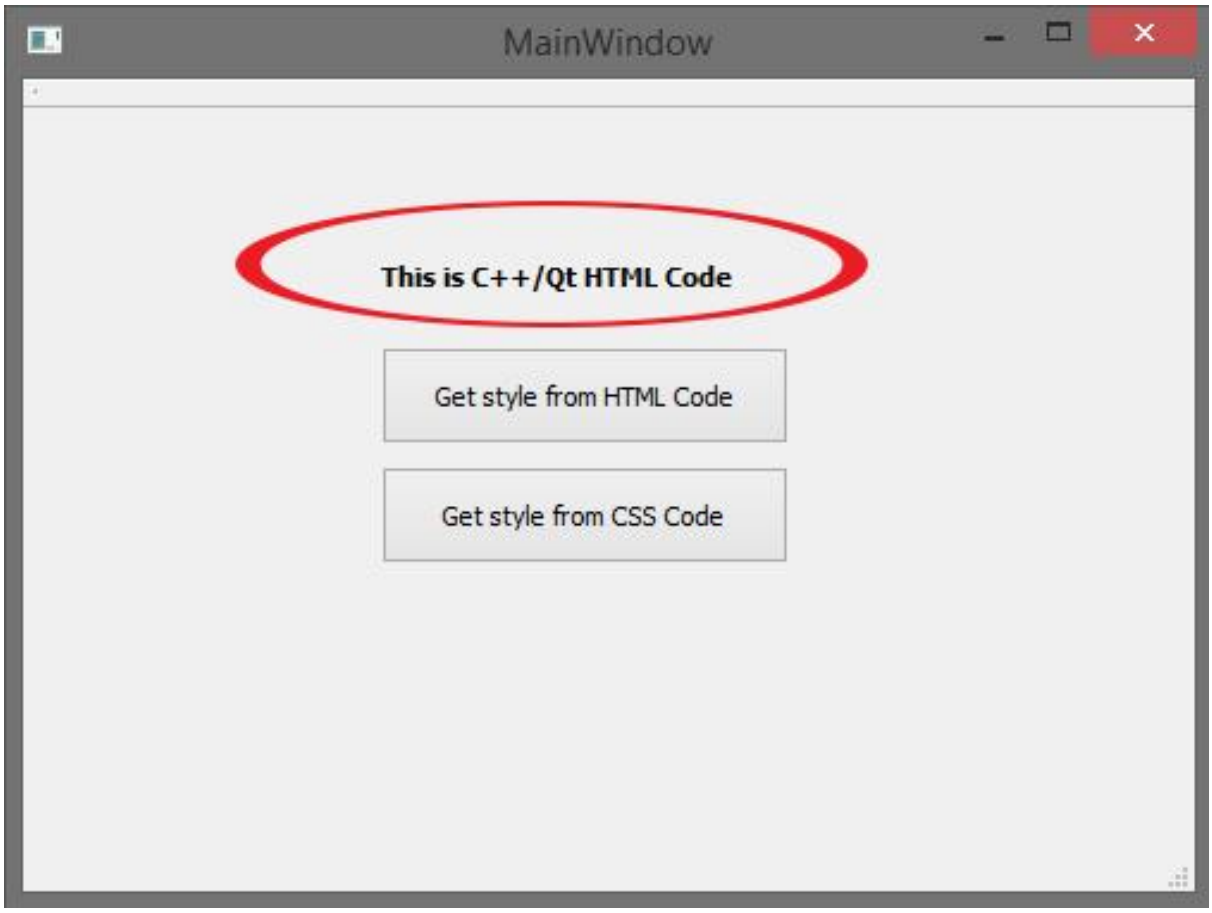
```
ui->label->setText("<b>This is C++/Qt HTML Code</b>");
```

این کد چی میگه؟ خیلی راحت داره میگه آقا جان از کلاس فرم من شیئ label رو انتخاب کن و متن یا همون محتویاتش رو برام در بین کد HTML ست کن. که رد حالت عادی به صورت زیر هستش.

```
ui->label->setText("This is C++/Qt HTML Code");
```

تنها چیزی که من اضافه کردم چیه ؟ فقط تگ های شروع و پایان دهنده Bold هست پس در نتیجه شما با وارد کردن تگ های HTML به صورت استاندارد در کدتون میتونید ازشوتن استفاده کنید.

نتیجه خروجی:



میبینید طبق تصویر مقایسه کنید متن label در حالت HTML پر رنگ شده به حالت bold تغییر کرده.

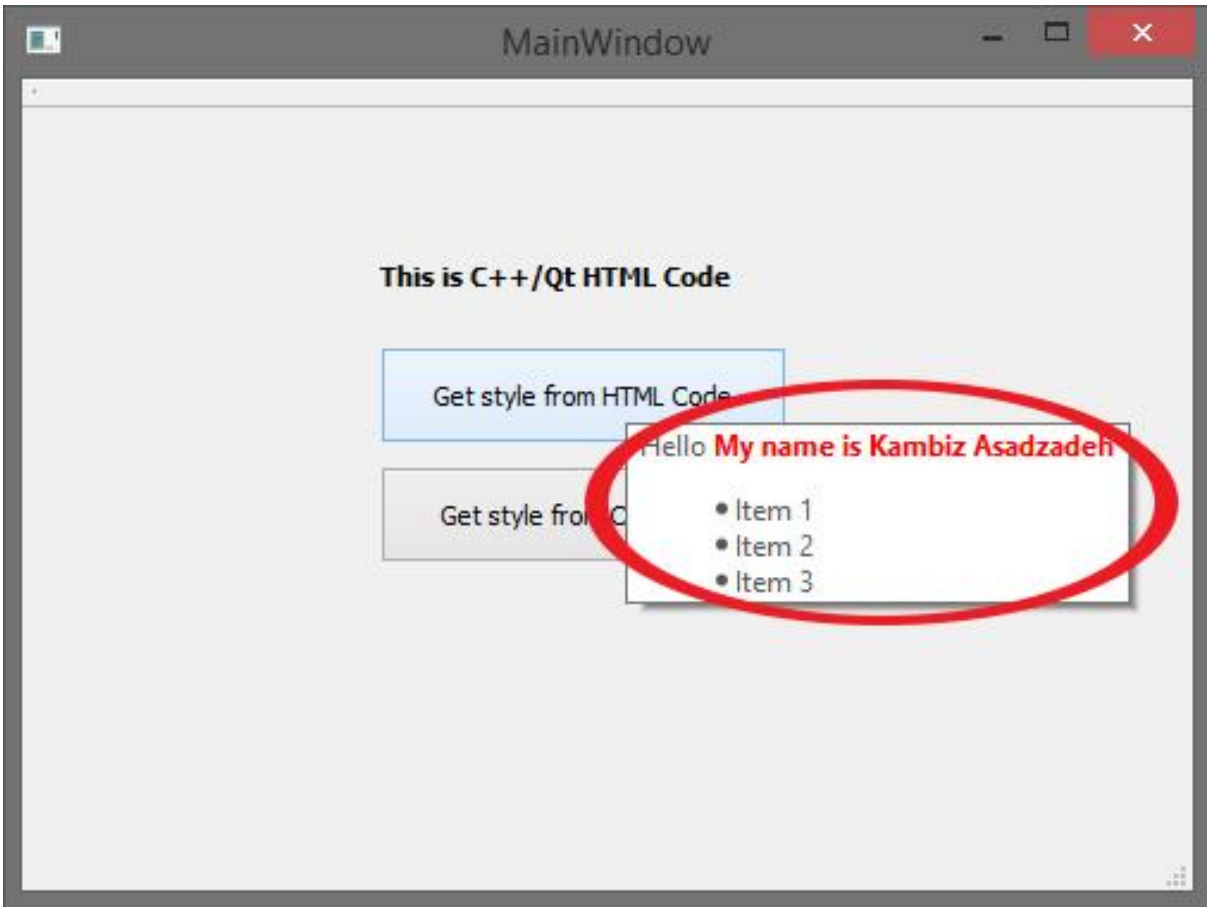
یه مثالی میزنم در رابطه با Tooltip با ... HTML طوری که وقتی روی دکمه اول ماوس رو نگه داریم یه Tooltipباز بشه که توسط HTML نوشتیم.

برای اینکار باید از Propertie چی استفاده کنیم ؟ مسلما باید از setToolTip استفاده کنیم

به این روش:

```
ui->pushButton->setToolTip("Hello <font color='red'><b>My name is    Kambiz
Asadzadeh</b></font>"
                                "<ul>"
                                "<li>Item 1</li>"
                                "<li>Item 2</li>"
                                "<li>Item 3</li>"
                                "</ul>");
```

خروجی شد به صورت زیر:



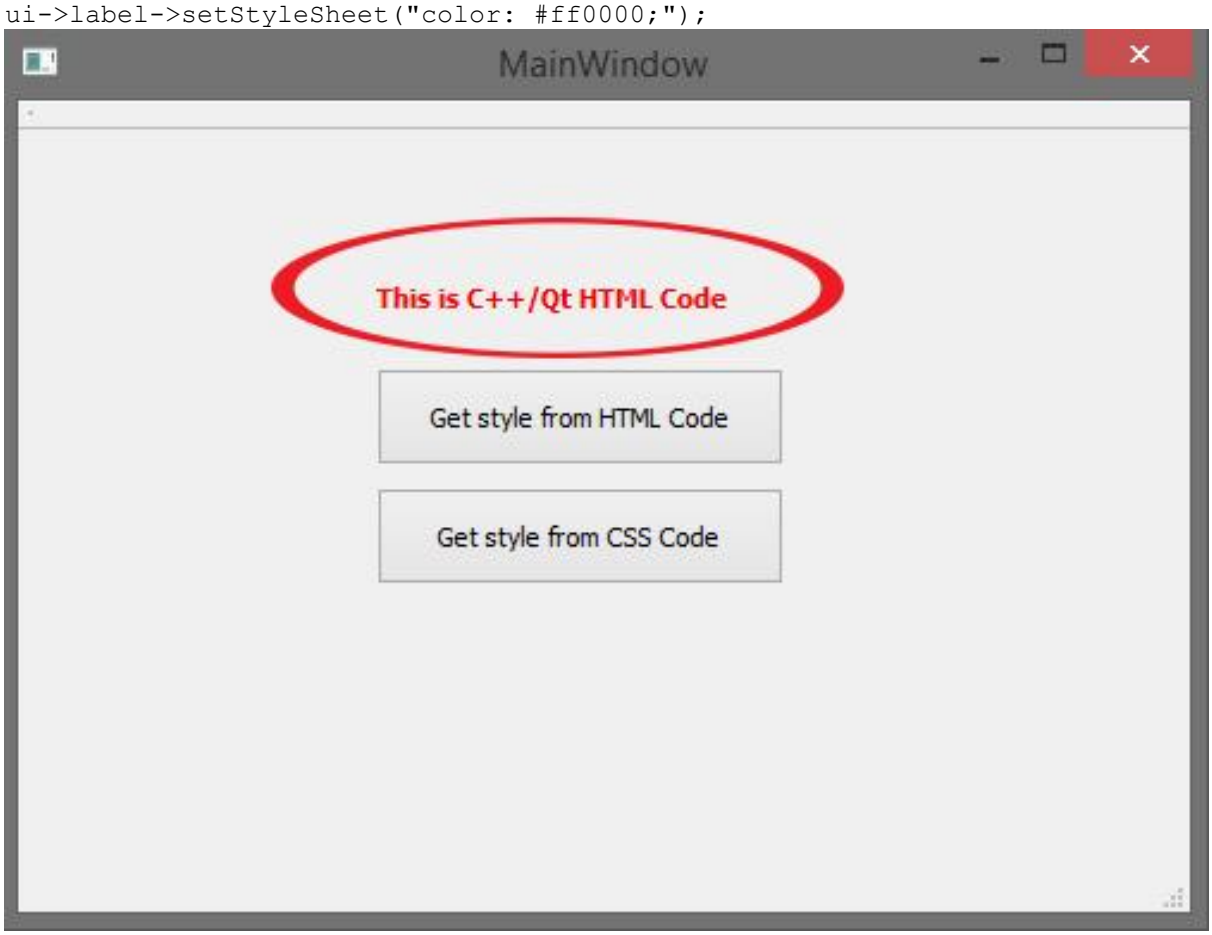
حالا یه نکته ای هم اشاره کنم که اگه دقت کنید در فرم بالا من از CSS هم در داخل همون HTML تگ استفاده کردم ... این یعنی چی یعنی HTML به طور کلی قابل پشتیبانی هست چون اگه نبود ویژگی خاصش رو اجرا نمیکرد.

در رابطه با CSS یکم توضیح بدم چطور میتونیم یه CSS کد به فرممون نسبت بدیم.

قبل از هر چیز باید اینو بگم شما میتونید هم با این روشی که گفتم در بین تگ های HTML از CSS استفاده کنید هم روش های دیگه در کد نویسی خود C++ و همچنین روش دستی در قسمت طراحی توسط Qt به صورت های زیر..

روش اول رو که دیدین...روش دوم چطوریه در روش دوم شما میتونید از خاصیت setStyleSheetبرای کنترل ها و اشیائ استفاده کنید مثلا به صورت زیر:

```
ui->label->setStyleSheet("color: #ff0000;");
```



تغییری که کرده رنگ محتویات label قرمز شده که توسط خاصیت CSS بهش ربط دادیم.

حالا میشه با یه روش دیگه ای هم این کار رو کرد اونم اینه روی فرم و روی labelراست کلیک میکنیم و گزینه Change StyleSheet رو انتخاب میکنیم در این حالت میتونید بر اساس نام کلاس و شناسه هر یک خاصیت و استایل رو ست کنید.

مثلا کد زیر رو در کادر باز شده وارد کنید و تایید کنید:

```
color: rgb(0, 85, 255);
```

رنگ خروجی میشه آبی...

در کل باید بگم شما میتویید کلا کدهای CSS رو در مرحله طراحی و همچنین کد نویسی به برنامتون ربط بدین و یا در نهایت یه کلاس های درست حسابی بنویسید برای استایل برنامتون و کلی کارای دیگه.

مرحله هفتم : معرفی و کار با لایه های افقی و عمودی

در رابطه با این قسمت یکم توضیحاتی بدم که هدف چه چیزی هست و بعد بریم سراغ آموزش.

خب تاحالا شنیدین و دیدین که در نرم افزار ها تنظیمات پنجره ها و همچنین لایه ها توسط Re-size شدن صورت میگیرن و در عین تغییر اندازه آبجکت ها و کنترل های موجود بر روی فرم متناسب با تغییرات اندازه به خوبی منظم و تغییر میکنند در این حالت ما میگیم تمامی کنترل های موجود روی لایه تنظیم شده اند.

برای شروع پروژه ای رو ایجاد میکنم و نامش رو میزارم MyLayout میتونیم این پروژه رو بدون widget هم ایجاد کنیم چون در این مرحله از کد نویسی برای ایجاد فرم و لایه ها استفاده خواهیم کرد.

ببینید قبل از هر چیزی من میخوام یک فرمی به عنوان پنجره ایجاد کنم و همچنین کنترل های لازم رو بر روی اون فراخوانی خواهم کرد.

برای این کار ما باید موارد لازم رو Include کنیم که در این مرحله ما نیاز به کلاس QWidget و QPushButtonو QHBoxLayout و QVBoxLayout هستیم.

حالا چطوری از این موارد استفاده خواهیم کرد به صورت زیر هستش.

ابتدا من اقدام به ساختن فرمم میکنم توسط کد زیر:

```
QWidget *window = new QWidget;
window->setWindowTitle("Window Layout");
```

در این کد من یه نمونه از QWidget میگیرم و نامش رو window میزارم و در خط بعدیش عنوانی برای این فرم خودم توسط خاصیت setWindowsTitle براش در نظر میگیرم.

حالا میخوام ۳ تا دکمه در فرمم قرار بدم به صورت زیر برای هر کدوم از QPushButton نمونه میگیرم.

```
QPushButton *button = new QPushButton("Button One");
 QPushButton *button1 = new QPushButton("Button Two");
```

```
QPushButton *button2 = new QPushButton("Button Three");
```

ببینید در هر ۳ تا خط هم من ۳ تا نمونه گرفتم با نام های مختلف ولی همشون از نوع دکمه یا همان Button هستند.

حالا من اگه فرم رو همینجوری show کنم دکمه های نمونه گرفته شده من در داخل فرم نمایش داده نخواهند شد و فرم من به صورت پیشفرض در حداکثر اندازه نمایش داده خواهد شد ! ولی هدف من اینه که به کمک کلاس های افقی و یا عمودی Buttonها رو در داخل فرم نمایش بدم و هچنین در این حالت اندازه فرم من بر اساس این کلاس متناسب با ۳ دکمه نمونه گرفته شده در داخل لایه Re-Size خواهد شد.

بر اساس نیاز نوع لایه رو انتخاب میکنیم که عمودی باشه یا افقی...

من لایه افقی رو انتخاب میکنم و ازش به صورت زیر نمونه میگیرم:

```
QHBoxLayout *hlayout = new QHBoxLayout;
```

حالا تا اینجا لایه من ساخته شده ولی بازهم در این مرحله چیزی نمایش داده نخواهد شد برای این امر باید به صورت زیر کنترل های خودم رو بر اساس لایه تعریف شده صدا بزنم.

```
hlayout->addWidget(button);
hlayout->addWidget(button1);
hlayout->addWidget(button2);
```

دقت کنید ... اینجا من میگم دکمه ۱..۲...۳ رو هر ۳ تاشون رو به لایه من اضافه کن.

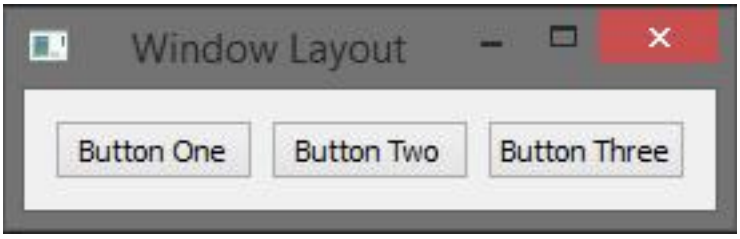
حالا تا اینجا کنترل های من به لایه اختصاص داده شده ولی خود لایه یه موردی داره !!! اونم اینه که برای نمایش لایه در داخل window که از نوع QWidget در نظر گرفتم نمایش داده نخواهد شد ! پس باید چکار کنم ؟ خب خیلی راحت باید من لایه ای که تعریف کردم رو به window تخصیص بدم به صورت زیر...

```
window->setLayout(hlayout);
```

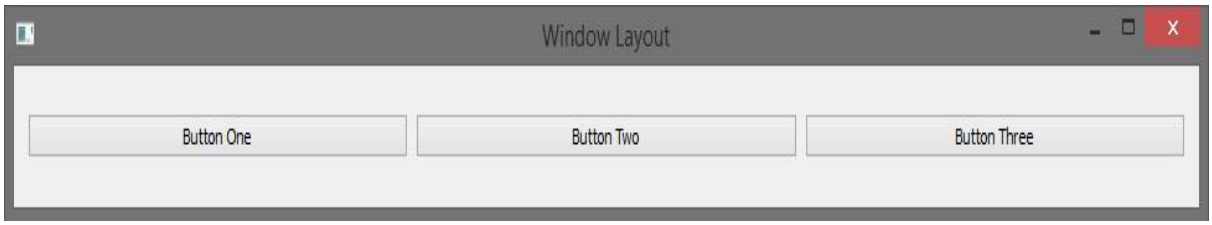
و در انتها برای اینکه فرم من به نمایش در بیاد باید به صورت زیر این کد رو هم فراموش نکنم:

```
window->show();
```

نتیجه این کارا میشه به صورت زیر:



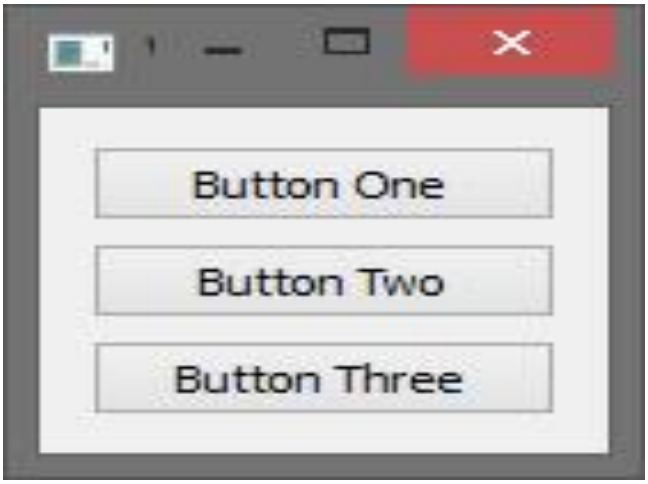
و



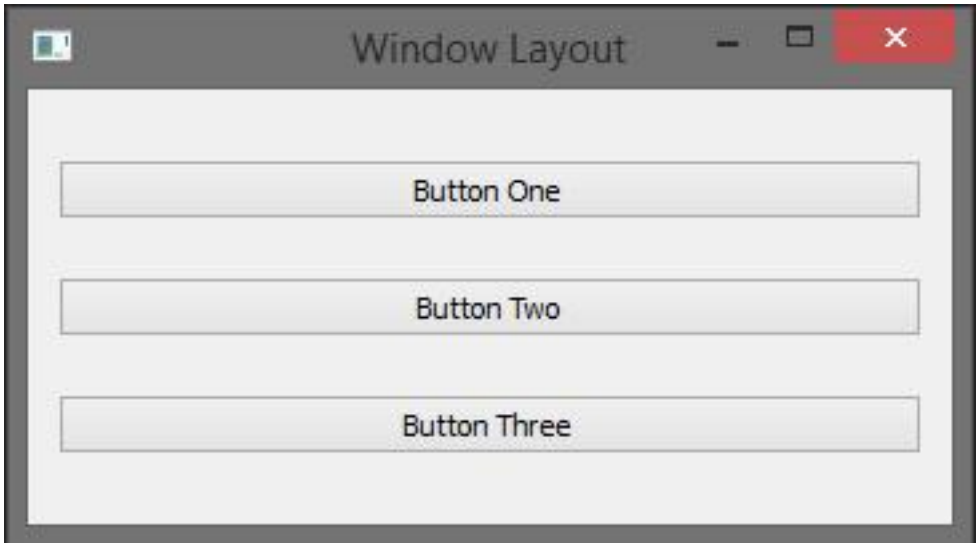
و اگه از حالت عمودی استفاده کنیم به صورت زیر کد ها تغییر میکنند:

```
QVBoxLayout *vlayout = new QVBoxLayout;

    vlayout->addWidget(button);
    vlayout->addWidget(button1);
    vlayout->addWidget(button2);

    window->setLayout(vlayout);
```

خروجی :



و



کد کلی به صورت زیر:

```
#include "mainwindow.h"
#include <QApplication>
#include <QPushButton>
#include <QHBoxLayout>
#include <QVBoxLayout>


int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    QWidget *window = new QWidget;
    window->setWindowTitle("Window Layout");

    QPushButton *button = new QPushButton("Button One");
    QPushButton *button1 = new QPushButton("Button Two");
    QPushButton *button2 = new QPushButton("Button Three");

    QHBoxLayout *hlayout = new QHBoxLayout;

    hlayout->addWidget(button);
    hlayout->addWidget(button1);
    hlayout->addWidget(button2);

    window->setLayout(hlayout);
    window->show();

    return a.exec();
}
```

مرحله هشتم : معرفی و کار با لایه های Grid در طراحی فرم

در این مرحله قصد داریم با لایه های Grid کار کنیم این لایه چی هستش و به چه دردی میخوره ؟

خب باید اینطور بگم شما روی فرم هر کنترلی رو درج میکنید مسلما انتظار این رو دارید که کنترل های شما و اشیائی که روی فرم صدا زده شده به صورت منظم بر اساس لایه های توری (Grid) منظم بشه یعنی هر جور که فرم رو طراحی کنیم برای هر کدام از کنترل ها یک مکانی رو روی Grid اختصاص میدیم.

میریم سراغ یه پروژه با نام MyGridLayout در محله اول فرمم رو ایجاد میکنم طبق کد زیر:

```
QWidget *window = new QWidget;
window->setWindowTitle("Grid Window");
```

حالا نیاز به لایه توری یا همون Grid دارم به صورت زیر اضافه میکنم:

```
QGridLayout *Layout = new QGridLayout;
```

حالا چند تا کنترل میخوام ایجاد کنم تا توی فرمم و داخل Grid درجشون کنم به صورت زیر:

```
QLabel *label1 = new QLabel("First name:");
QLineEdit *txtFirstName = new QLineEdit;
QLabel *label2 = new QLabel("Last name:");
QLineEdit *txtLastName = new QLineEdit;
QPushButton *button = new QPushButton("OK");
```

در این بخش ما یه خصوصیاتی رو باید توجه کنیم که Grid بر اساس اون کنترل رو نمایش میده...

تابع به صورت زیر هستش:

```
void QGridLayout::addWidget(QWidget * widget, int fromRow, int fromColumn,
int rowSpan, int columnSpan, Qt::Alignment alignment = 0)
```

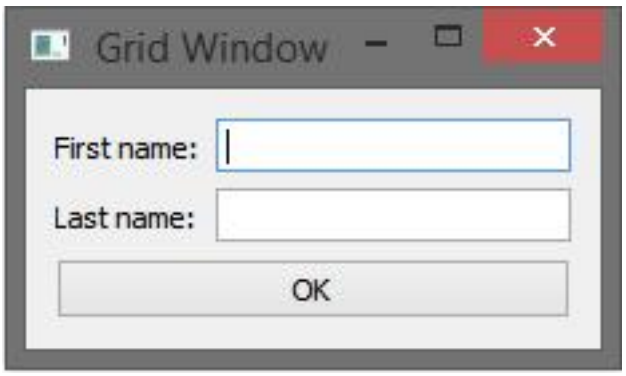
خب حالا بحث اصلی اینجاست چطوری باید تعیین کنم که در کدوم قسمت از Grid کنترل های من باید نمایش پیدا کنن...

در زیر ما بر اساس همین قانون تابع Grid برای هر یک از کنترل ها شماره ایندکس سلول و ستونش رو مشخص کردیم.

```
Layout->addWidget(label1,0,0);
Layout->addWidget(txtFirstName,0,1);
```

```
    Layout->addWidget(label2,1,0);
    Layout->addWidget(txtLastName,1,1);
    Layout->addWidget(button,2,0,1,2);
```

هر جا که لازم نباشه از عدد ۰ به منظور در نظر نگرفتن مقدار میتونید استفاده کنید در غیر این صورت باید مقدار رو بر اساس اولویت یعنی ۱...۲...۳ و ... وارد کنید و نتیجش میشه به صورت زیر:



کد کامل همراه با اینکلود ها و دستورات تکمیلی کار...

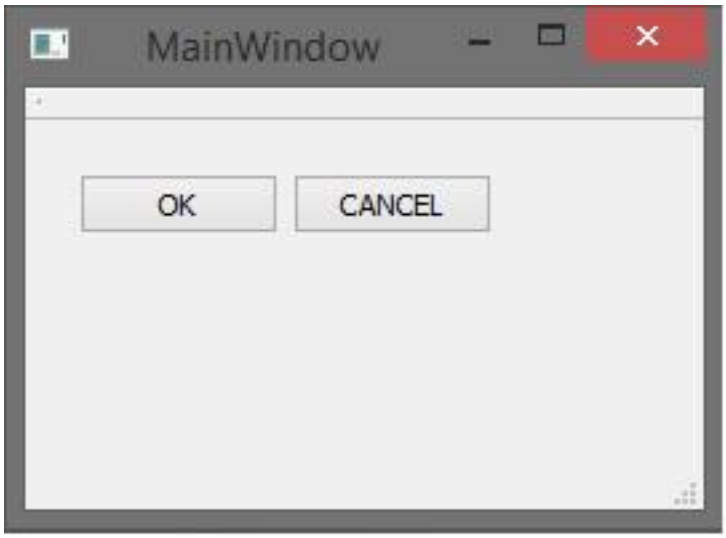
```
#include <QApplication>
#include "QLabel"
#include "QLineEdit"
#include "QLayout"
#include "QPushButton"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    QWidget *window = new QWidget;
    window->setWindowTitle("Grid Window");

    QGridLayout *Layout = new QGridLayout;

    QLabel *label1 = new QLabel("First name:");
    QLineEdit *txtFirstName = new QLineEdit;
    QLabel *label2 = new QLabel("Last name:");
    QLineEdit *txtLastName = new QLineEdit;
    QPushButton *button = new QPushButton("OK");

    Layout->addWidget(label1,0,0);
    Layout->addWidget(txtFirstName,0,1);
    Layout->addWidget(label2,1,0);
    Layout->addWidget(txtLastName,1,1);
    Layout->addWidget(button,2,0,1,2);
```

```
    window->setLayout(Layout);

    window->show();


    return a.exec();
}
```

مرحله نهم : معرفی و کار با جدا کننده هاSplitter

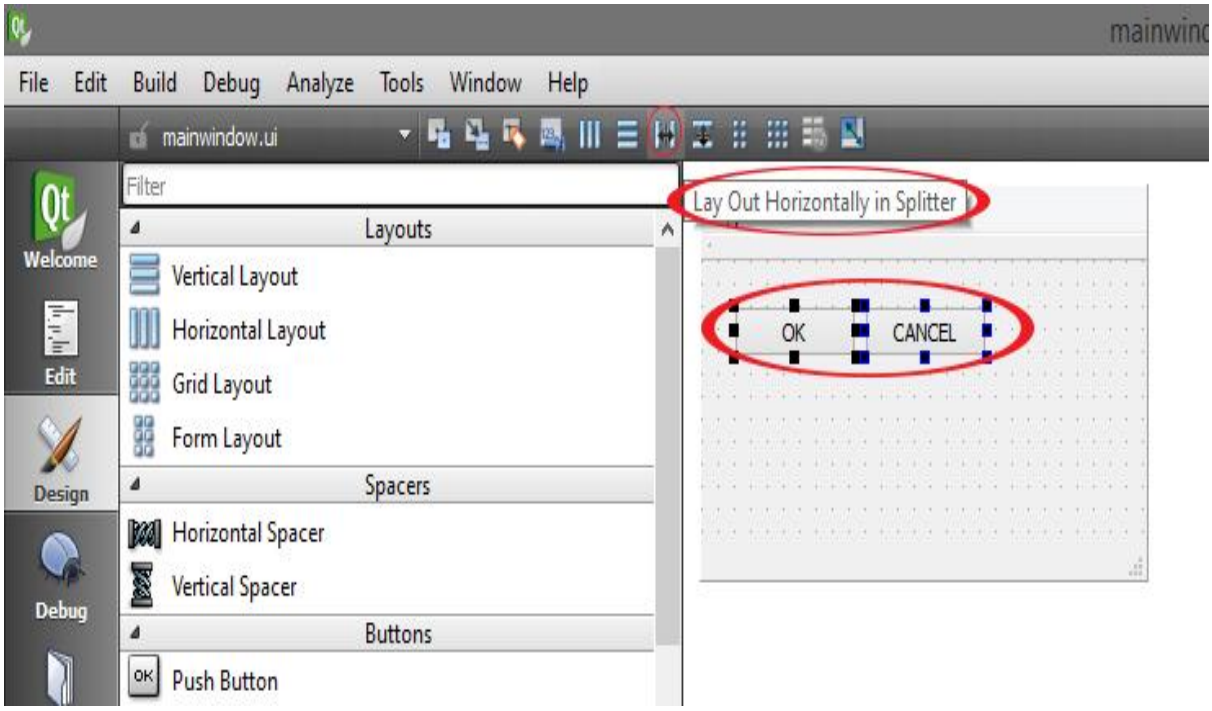
در رابطه با جداکننده یا همون Splitter یکم توضیحات بدم ... منظور از این اصطلاح چیه و تو برنامه نویسی به چه دردی میخوره ؟ در رابطه با این گزینه باید اینطور بگم فرض کنید شما فرمی دارید و فضای فرم شما هنگام تغییر اندازه کلی فرق میکنه و کنترل های موجو در اون هیچ تغییری هنگام تغییر اندازه فرم نمیکنند ! برای رفع این مشکل در صورت نیاز میتونیم به صورت زیر عمل کنیم.

خب یه پروژه از نوع Widget ایجاد میکنم اسمشو میزارم Splitter و به صورت زیر دو تا کنترل توش ایجاد میکنم.

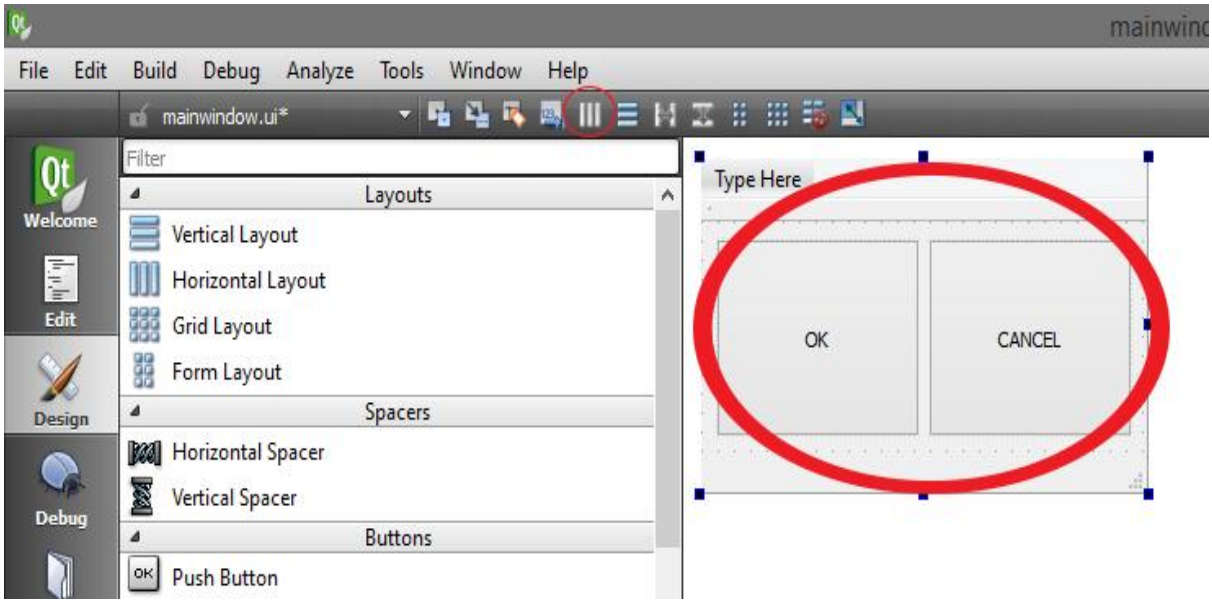


اگه دقت کنید در مرحله اول دکمه ها و فرم ما مرتب نیستند و هنگام تغییر سایز هم کنترل ها همراه با فرم تغییری نمیکنند.

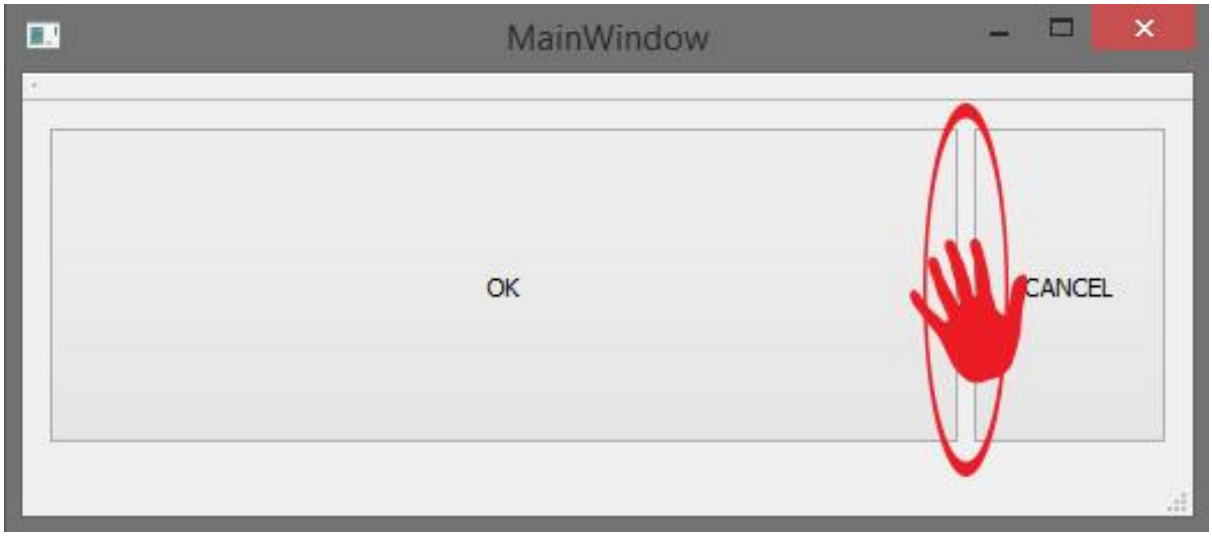
برای حل این مورد باید چکار کنیم ؟ خب طبق تصویر زیر هر دو کنترل رو انتخاب میکنم و بر اساس محور افقی و یا عمودی ستشون میکنم که من در این مثال از نوع Horizontal همون افقی برای Splitter انتخابشون کردم.



حالا روی فرم کلیک میکنیم و گزینه Layout Horizontally رو انتخاب میکنیم تا به صورت زیر کنترل ها با فرم ما هماهنگ بشن.



خب حالا بعد از اجرای فرم من همراه با کنترل هاش هماهنگ میشه و هنگام تغییر اندازه فرم کنترل هامم تغییر اندازه میدن و حتی اگه از حالت Drag و Move استفاده کنید میبینید که خاصیت Splitter فعال هست و راحت میشه تغییرش داد.



شما طبق این آموزش میتونید نه تنها کنترل بلکه تو برنامه هاتون نوار ابزار هایی تولید کنید که مثل همین نوار ابزار های محیط برنامه نویسی سفارشی و قابلیت Splitter داشته باشند.

مرحله دهم : معرفی و کار با دایرکتوری ها

میریم سر کار با دایرکتوری ! برای شروع یه پروژه ایجاد کنیم اینبار از نوع Consoleچون نیازی به کار روی فرم نداریم فعلا فقط میخوام در رابطه با نحوه بررسی دایرکتوری توضیح بدم.

در Qt برای صدا زدن توابع کار با دایرکتوری باید از QDir استفاده کنیم.

برای مثال من ابتدا اینکلود میکنم QDir رو و ازش یه نمونه میسازم و مسیری که میخوام بررسی کنم رو مشخص میکنم به صورت زیر:

```
#include <QCoreApplication>
```

```
#include <QDebug>
#include <QDir>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    QDir MyDir("c:/");
       qDebug() << MyDir.exists();

    return a.exec();
}
```

در اینجا چه کردیم ؟ من از کلاس QDir یک نمونه گرفتم و مسیری که لازم دارم برای بررسی که همون دایرکتوری هستش رو بهش دادم و در مرحله بعدش اگر مسیری که من دادم رو تشخیص بده برام مقدار بولین trueرو و اگر نه مقدار بولین falseرو میده.

ساختارش به صورت زیر هست:

```
bool QDir::exists(const QString & name) const
```

اگر مسیر موجود باشه True و اگر نباشه False رو برگشت میده.

```
qDebug() << MyDir.exists();
```

qDebugهمون عمل چاپ رو انجام میده در مورد این توضیحات ندادم ولی کارش دقیقا مثل std::cout هستش که در حالت استاندارد میشه به این صورت ازش چاپ گرفت:

```
std::cout << MyDir.exists();
```

خب در این مرحله قصد دارم تعداد درایو هایی که روی سیستم من فعال هستند رو نمایش بدم.

به صورت زیر یه نمونه از QDir و یه نمونه برای لیست کردن درایو هام از QFileInfoو گرفتن اطلاعاتشون میگیرم و داخل یک حلقه روند رو تا جایی که لازمه تکرار میکنم.

```
    QDir MyDir;
        foreach (QFileInfo MyItem,MyDir.drives())
        {
            qDebug() << MyItem.absoluteFilePath();
```

```
        }
```

نتیجش میشه به صورت زیر البته من چون ۳ تا درایو دارم ۳ تا نشون میده نه بیشتر که این وسط وظیفه absoluteFilePathهستش که اطلاعات مربوطه رو بازگشت میده به خروجی.

```
"C:/"
"D:/"
"E:/"
```

کد کلی به صورت زیر:

```
#include <QCoreApplication>
#include <QDebug>
#include <QDir>
#include <QFileInfo>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    QDir MyDir;
        foreach (QFileInfo MyItem,MyDir.drives())
        {
            qDebug() << MyItem.absoluteFilePath();

        }

    return a.exec();
}
```

مرحله یازدهم : معرفی و کار با فایل ها / خواندن و نوشتن در آن ها

در این قسمت میریم سراغ کار با فایل ها در C++/Qt و خواندن محتویات فایل(Read)

ابتدا کلاس های مورد نظرم رو فراخوانی میکنم به صورت زیر:

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

حالا در نظر دارم یه فایلی ایجاد کنم و محتویات داخل اون رو بخونم برای این منظور یه تابع Read میسازم به صورت زیر...

```
void Read(QString Filename)
{
QFile MyFile(Filename);

if(!MyFile.open(QFile::ReadOnly | QFile::Text))
{
qDebug() << "could not open file for reading";
return;
}

QTextStream in(&MyFile);
QString MyText = in.readAll();

qDebug() << MyText;

MyFile.close();
}
```

حالا در این قسمت تابعی که نوشتم رو فراخوانی میکنم و ادرس مسیر و فایلم رو مشخص میکنم تا محتویات اون رو برام نمایش بده به صورت زیر:

```
int main(int argc, char *argv[])
{
QCoreApplication a(argc, argv);

Read("C:/TEST/test.txt");

return a.exec();
}
```

خروجی شد به صورت زیر هر چیزی که در داخل فایل هستش:

```
"My name is Kambiz Asadzadeh :)"
```

قبل از هر چیز توضیح بدم که برای دریافت نام و مسیر فایل مورد نظر در تابع به صورت زیر باید نوشته بشه:

```
void Read(QString Filename)
```

یعنی نام و مسیر فایل رو از نوع رشته در نظر بگیر و وارد تابع برای پردازش های بعدی بکن.

حالا بریم سراغ توضیحات.... در مرحله اول در رابطه با تابع Read توضیح بدم....

من ابتدا برای نوشتن و یا خواندن در داخل یک فایل باید از کلاس QFile یک نمونه بگیرم تا بتونم این کار رو انجام بدم پس به صورت زیر ابتدا یه نمونه گرفتم:

```
QFile MyFile(Filename);
```

در خط بعدی یه دستور شرطی آوردم و گفتم که اگر فایلم رو از نوع متنی و در حالت فقط خواندنی در نظر بگیر اگر فایل من رو بر اساس این شرایط تشخیص و قادر به خواندن محتویات آن نشد پیغام عدم توانایی در باز کردن و خواندن فایل رو بهم بده.

به صورت زیر:

```
if(!MyFile.open(QFile::ReadOnly | QFile::Text))
{
qDebug() << "could not open file for reading";
return;
}
```

در قسمت بعد من برای اینکه بتونم با یک فایل متنی و رشته ای کار کنم که میتونه خواندن یا نوشتن باشه باید از کلاس **QTextStream** یه نمونه ای داشته باشم وظیفه این کلاس ارائه یک رابط مناسب برای خواندن و نوشتن در فایل هستنش به صورت زیر:

```
QTextStream in(&MyFile);
```

حالا در قسمت بعد میگم تمام محتویاتی که در فایل وجود داره رو برام بگیر به صورت زیر:

```
QString MyText = in.readAll();
```

در قسمت بعد دستور چاپ محتوای فایل رو دادم و در نهایتش هم دستور بستن فایل رو براش ارسال کردم به صورت زیر:

```
qDebug() << MyText;

MyFile.close();
```

تا اینجا من نه فایل رو و نه مسیر فایلم رو مشخص نکردم البته این میشد من در داخل خود تابع این کار رو کنم ولی برای بهینه شدن مایلم به صورت تابع و فقط هنگام صدا زدن تابع مشخص کنم که از چه مسیری چه فایلی باز کنه و محتوای اون رو برام نمایش بده برای این ار به صورت زیر عمل میکنم:

```
Read("C:/TEST/test.txt");
```

ابتدا نام تابعی که ساختم و بعد مسیرش همراه با نام و پسوند فایلم رو مشخص کردم.

کد کامل برای خواندن یک فایل در یک مسیر مشخص به صورت زیر:

```
#include <QCoreApplication>
#include <QDebug>
```

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

void Read(QString Filename)
{
QFile MyFile(Filename);

if(!MyFile.open(QFile::ReadOnly | QFile::Text))
{
qDebug() << "could not open file for reading";
return;
}

QTextStream in(&MyFile);
QString MyText = in.readAll();

qDebug() << MyText;

MyFile.close();
}


int main(int argc, char *argv[])
{
QCoreApplication a(argc, argv);

Read("C:/TEST/test.txt");

return a.exec();
}
```

در این قسمت میریم سراغ کار با فایل ها در C++/Qt و نوشتن در داخل فایل(Write)

ابتدا کلاس های مورد نظرم رو فراخوانی میکنم به صورت زیر:

```
#include <QFile>

#include <QString>

#include <QTextStream>
```

حالا در نظر دارم یه فایلی ایجاد کنم و محتویات داخل اون رو بخونم برای این منظور یه تابع Write میسازم به صورت زیر...

```
void Write(QString Filename)
{
```

```
QFile MyFile(Filename);

if(!MyFile.open(QFile::WriteOnly | QFile::Text))
{
qDebug() << "could not open file for reading";
return;
}

QTextStream out(&MyFile);
out << "Hello my friend :D";
std::cout << "The text is written!";

MyFile.flush();
MyFile.close();
}
```

حالا در این قسمت تابعی که نوشتم رو فراخوانی میکنم و آدرس مسیر و فایلم رو مشخص میکنم تا محتویات اون رو برام نمایش بده به صورت زیر:

```
int main(int argc, char *argv[])
{
QCoreApplication a(argc, argv);

Write("C:/TEST/test.txt");

return a.exec();
}
```

خروجی شد به صورت زیر هر چیزی که در داخل فایل هستش:

```
The text is written!
```

و اگر داخل فایل خودتون را باز کنید خواهید دید که متن زیر در درون ان نوشته شده است:

```
Hello my friend :D
```

حالا بریم سراغ توضیحات.... در مرحله اول در رابطه با تابع Write توضیح بدم....

من ابتدا برای نوشتن و یا خواندن در داخل یک فایل باید از کلاس QFile یک نمونه بگیرم تا بتونم این کار رو انجام بدم پس به صورت زیر ابتدا یه نمونه گرفتم:

```
QFile MyFile(Filename);
```

در خط بعدی یه دستور شرطی آوردم و گفتم که اگر فایلم رو از نوع متنی و در حالت فقط نوشتنی در نظر بگیر اگر فایل من رو بر اساس این شرایط تشخیص و قادر به نوشتن محتویات آن نشد پیغام عدم توانایی در باز کردن و نوشتن فایل رو بهم بده.

به صورت زیر:

```
if(!MyFile.open(QFile::WriteOnly | QFile::Text))
{
qDebug() << "could not open file for writing";
```

```
return;
}
```

در قسمت بعد من برای اینکه بتونم با یک فایل متنی و رشته ای کار کنم که میتونه خواندن یا نوشتن باشه باید از کلاس QTextStream یه نمونه ای داشته باشم وظیفه این کلاس ارائه یک رابط مناسب برای خواندن و نوشتن در فایل هستنش به صورت زیر:

```
QTextStream out(&MyFile);
```

حالا در قسمت بعد میگم متن مورد نظر رو در داخل فایل من بنویس به صورت زیر:

```
out << "Hello my friend :D";
```

در قسمت بعد دستور چاپ پیغام نوشته شد رو دادم و در نهایتش هم دستور بستن فایل رو براش ارسال کردم به صورت زیر:

```
out << "Hello my friend :D";
std::cout << "The text is written!";

MyFile.flush();
MyFile.close();
```

قبل از بستن فایل گفتم که هرگونه متنی رو که در فایل میخوای بنویسی قبلش در حافظه Buffer قرار بده در این صورت مقدار ۱ رو برام بازگشت میده و فایل رو میبنده در غیر اینصورت ۰ و خطا!

تا اینجا من نه فایل رو و نه مسیر فایلم رو مشخص نکردم البته این میشد من در داخل خود تابع این کار رو کنم ولی برای بهینه شدن مایلم به صورت تابع و فقط هنگام صدا زدن تابع مشخص کنم که از چه مسیری چه فایلی باز کنه و متنی رو در داخل اون بنویسه برای این ار به صورت زیر عمل میکنم:

```
[CPP]Write("C:/TEST/test.txt");
```

ابتدا نام تابعی که ساختم و بعد مسیرش همراه با نام و پسوند فایلم رو مشخص کردم.

کد کامل برای نوشتن در یک فایل در یک مسیر مشخص به صورت زیر:

```
#include <QCoreApplication>
#include <QDebug>
#include <QFile>
#include <QString>
#include <QTextStream>

void Write(QString Filename)
{
QFile MyFile(Filename);

if(!MyFile.open(QFile::WriteOnly | QFile::Text))
{
qDebug() << "could not open file for reading";
return;
}

QTextStream out(&MyFile);
out << "Hello my friend :D";
std::cout << "The text is written!";

MyFile.flush();
MyFile.close();
}


int main(int argc, char *argv[])
{
QCoreApplication a(argc, argv);

Write("C:/TEST/test.txt");

return a.exec();
}
```

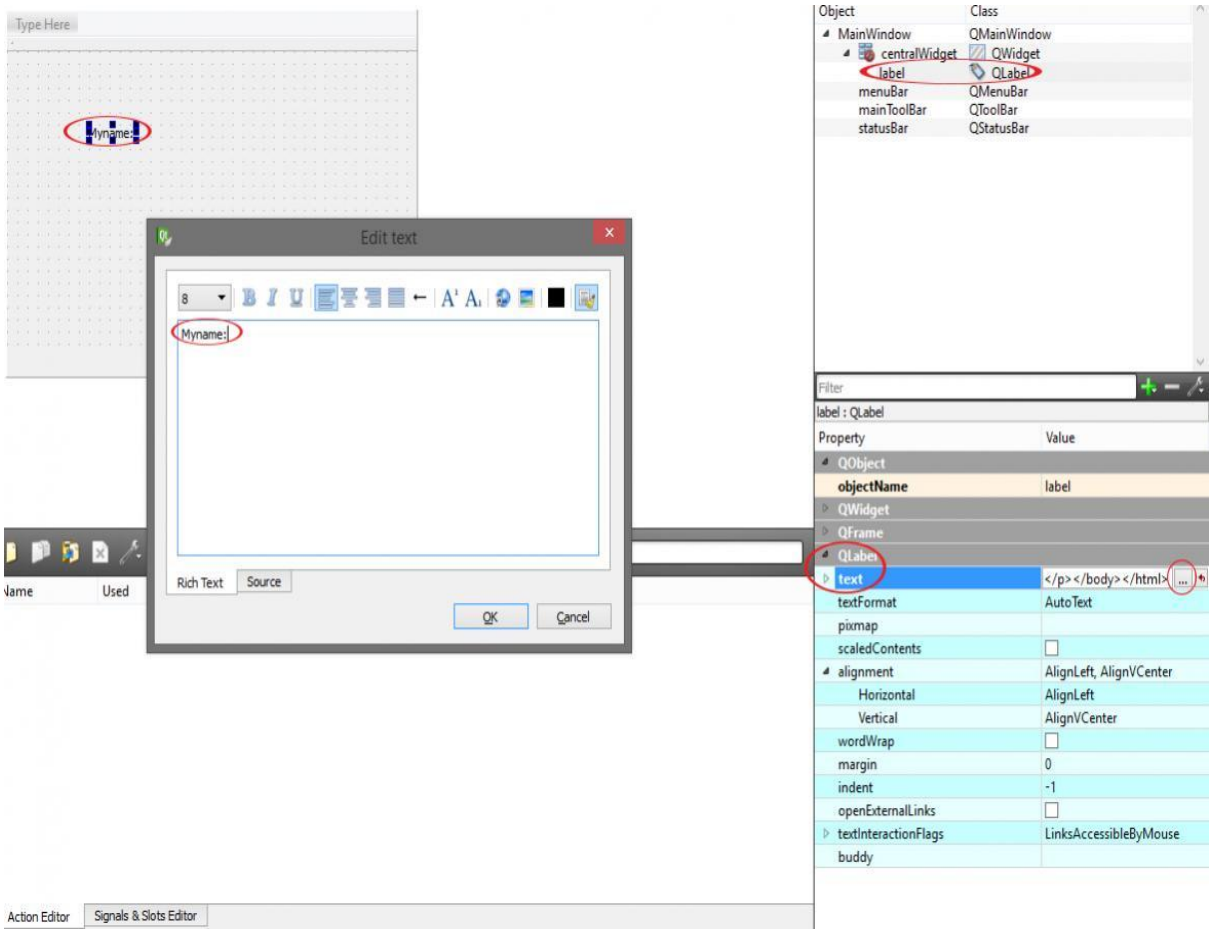
مرحله دوازدهم : معرفی و کار با برچسب هاLabel

میخوام در رابطه با کنترل برچسب (label) یا همان QLabel توضیح بدم.

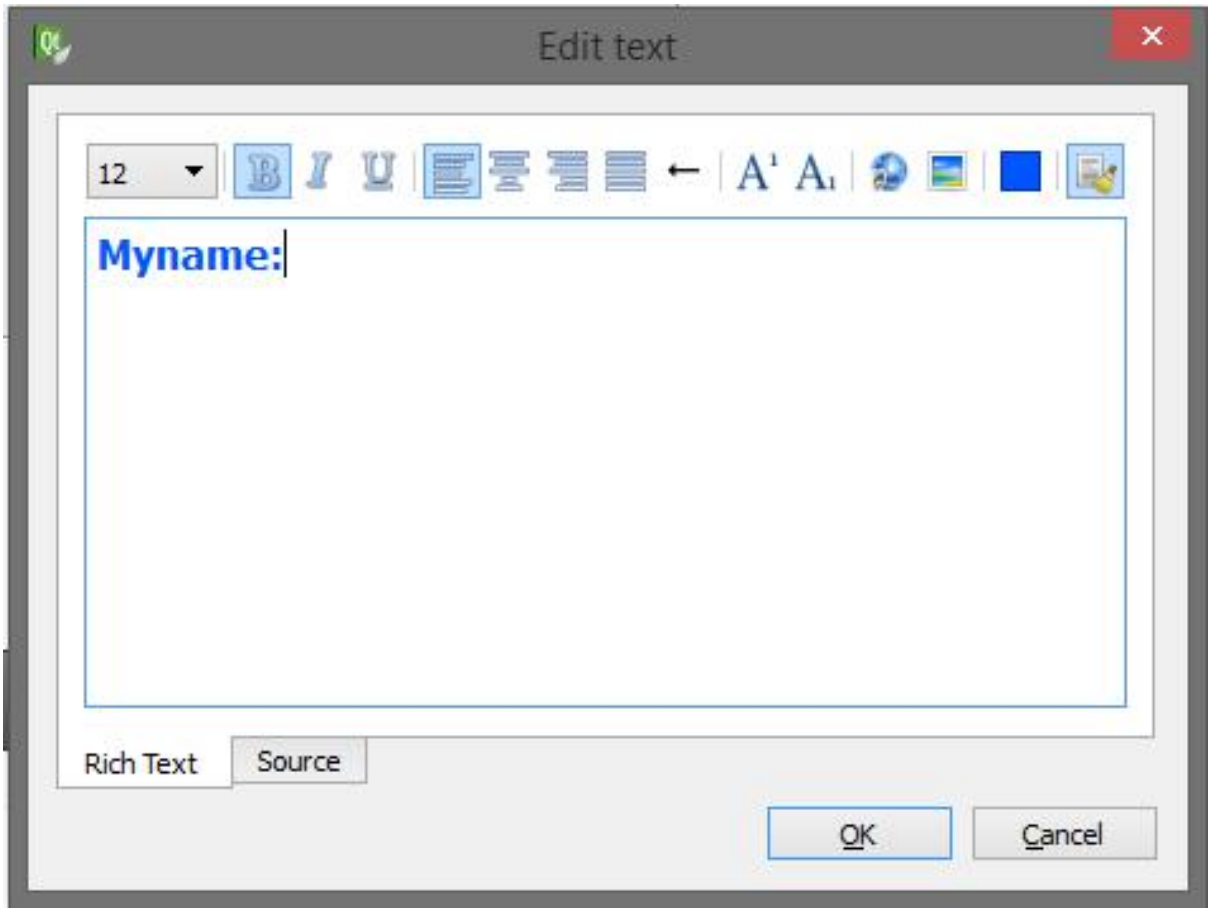
همونطور که میدونید برای نمایش متن و عنوان گزاری در طراحی از لیبل استفاده میشه که در کیوت به راحتی با درگ دراپ کردن این شیئ روی فرم اون رو میتونید ایجاد کنید.

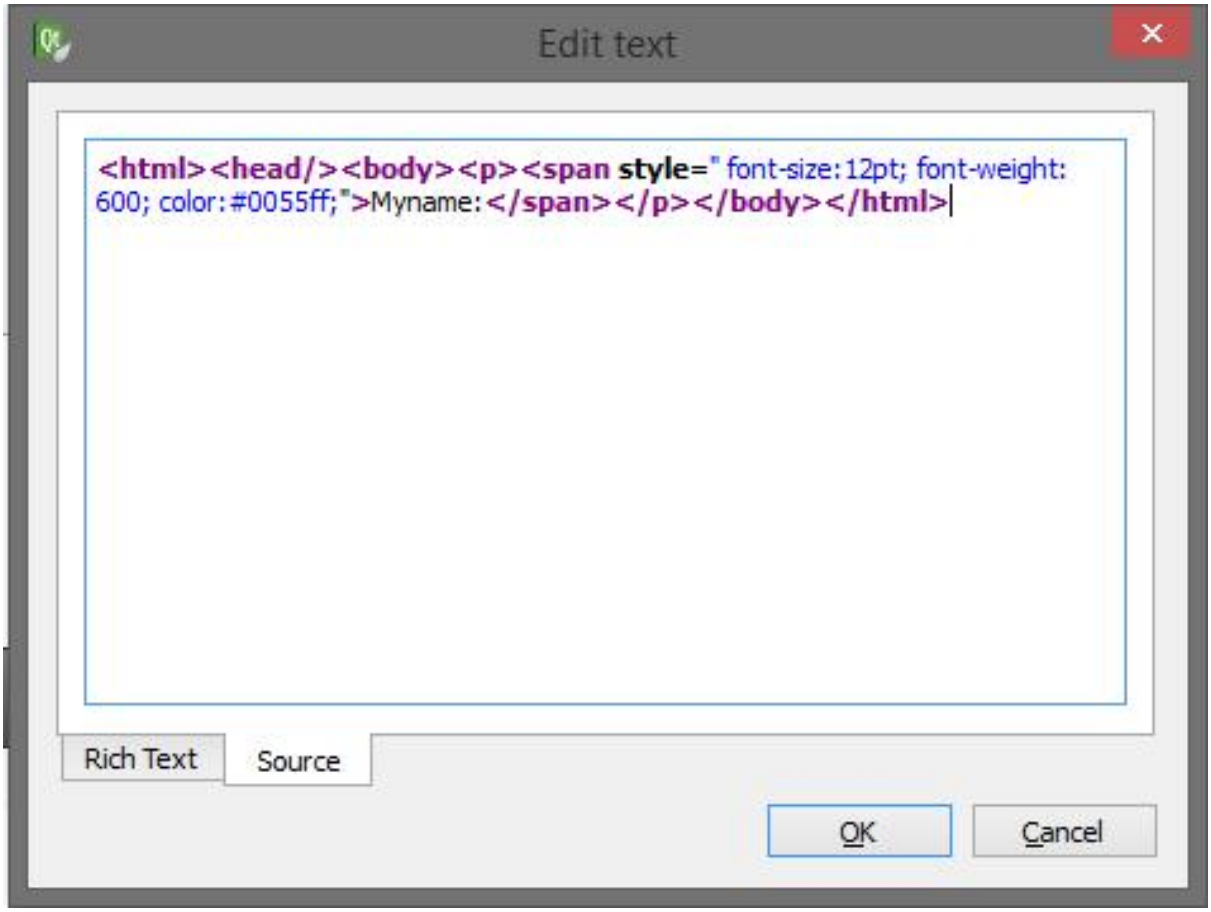
اسم این لیبل به صورت پیشفرض label هست شما میتونید در قسمت objectName اسمش رو تغییر بدین.

من لیبلم رو قرار دادم ولی حالا میخوام متن لیبل رو بر اساس سلیقه تغییر بدم در محیط طراحی شما میتونید با دوباره کلیک روی متن پیشفرض لیبل اون رو تغییر بدین و یا اینکه از قسمت Property خاصیت Text رو تغییر بدین به صورت زیر:



که در این حالت شما میتونید امکان تغییر فونت رو در سریعترین حالت در دست داشته باشید و همچنین سورس HTMLرو در اختیار داشته باشید به صورت زیر:


Edit text
12
Myname:
Rich Text
Source
OK
Cancel


Edit text
<html><head/><body> Myname: </body></html>
Rich Text
Source
OK
Cancel

و یا راه سوم اینه که توسط کد نویسی این کار رو انجام بدین در داخل فایل mainwindow.cpp به صورت زیر:

```
ui->label->setText("<b>Hello</b> , My name is Kambiz");
```

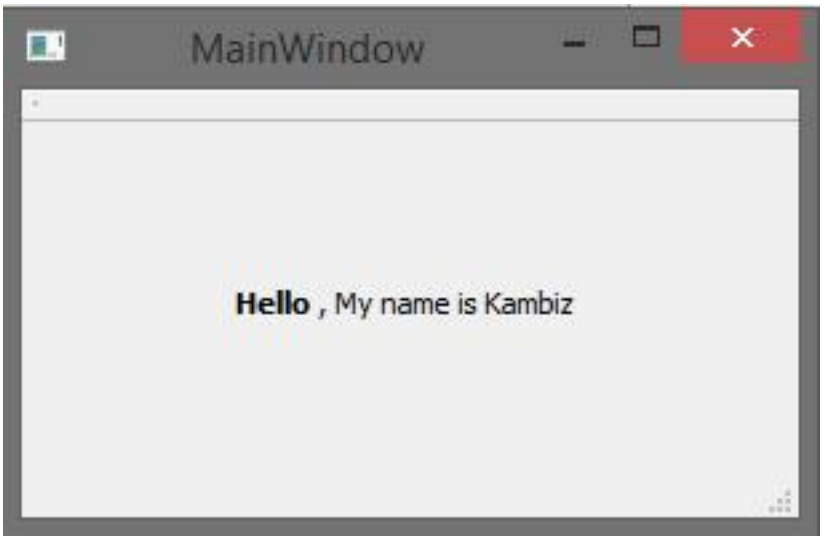
خب در روش کد نویسی اگرچه شما در حالت طراحی تغییراتی دادین و بعد از کد استفاده کردین باید به این توجه کنید که کد نوشته شده در نظر گرفته میشه و هر چیزی که توسط کد نوشتین و استایل دادین در خروجی نمایش داده خواهد شد.

هر چند در حالت دستی قابلیت ویرایش و استایل بندی سریع در اختیار گذاشته میشه ولی برای استایل نویسی همانطور که قبلا گفتم میتونید از قابلیت HTML و CSS استفاده کنید.

کد کلی :

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
QMainWindow(parent),
ui(new Ui::MainWindow)
{
ui->setupUi(this);
ui->label->setText("<b>Hello</b> , My name is Kambiz");

}

MainWindow::~MainWindow()
{
delete ui;
}
```

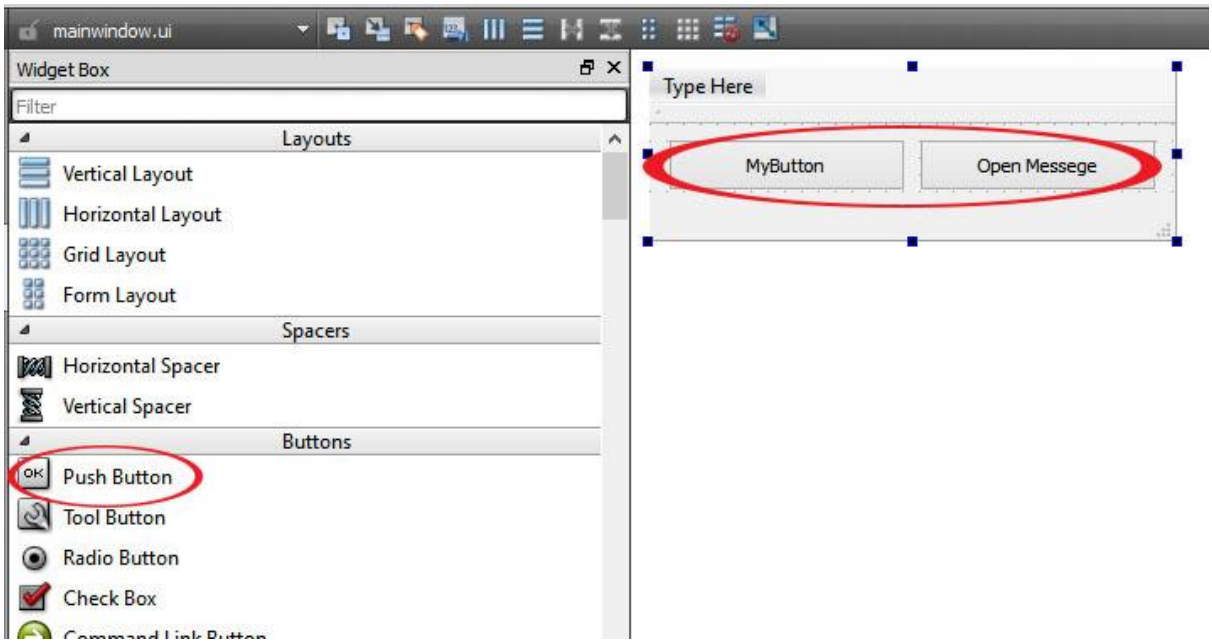
خروجی به صورت زیر:



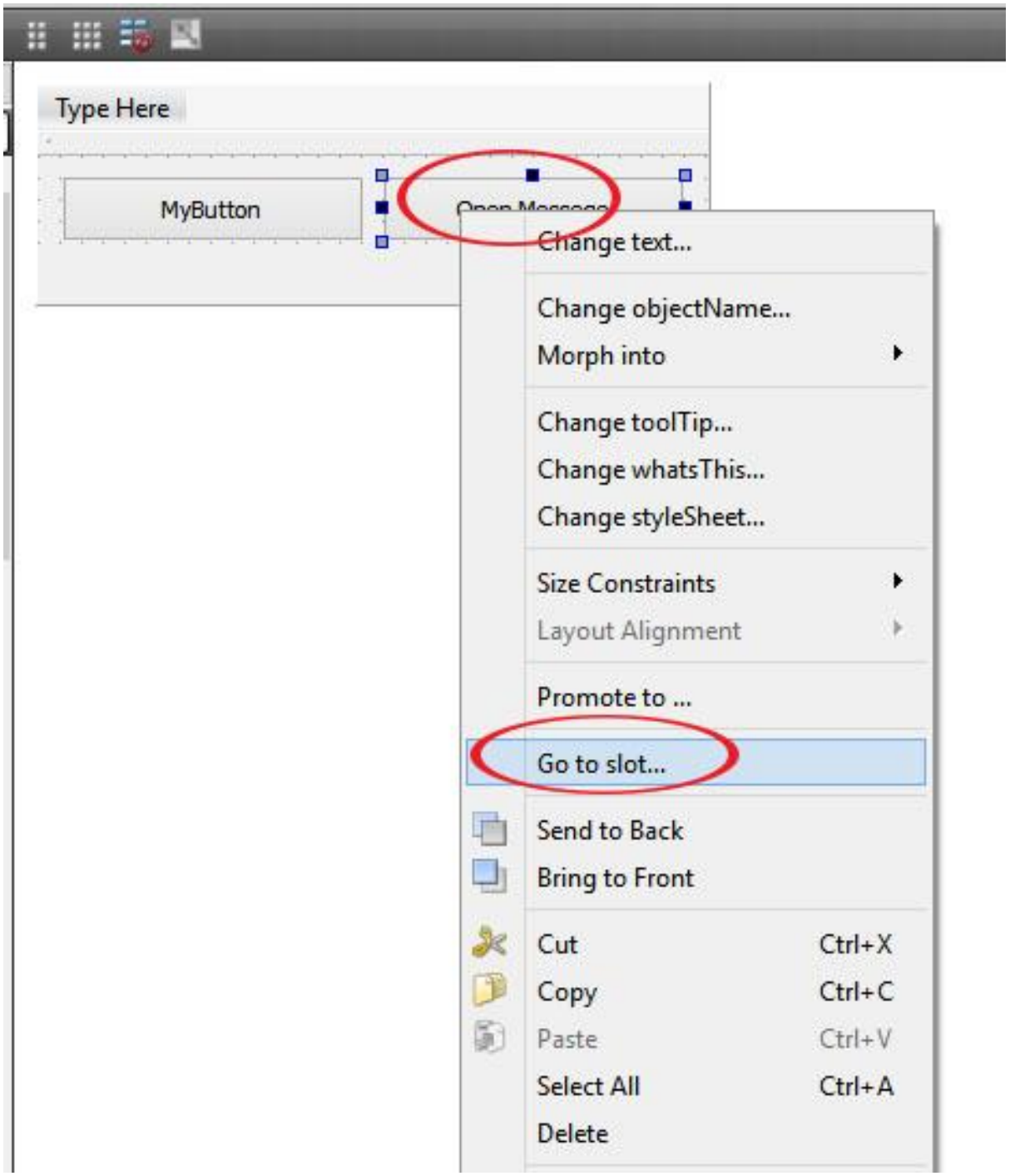
مرحله سیزدهم : معرفی و کار باButton

در رابطه با Button باید بگم یکی از پر کاربرد ترین شیئ هایی هستش که ازش استفاده خواهد شد و مخصوصا رخداد Event در حالت کلیک شدن به صورت زیر پروژه رو ایجاد و از QPushButton استفاده می کنیم.

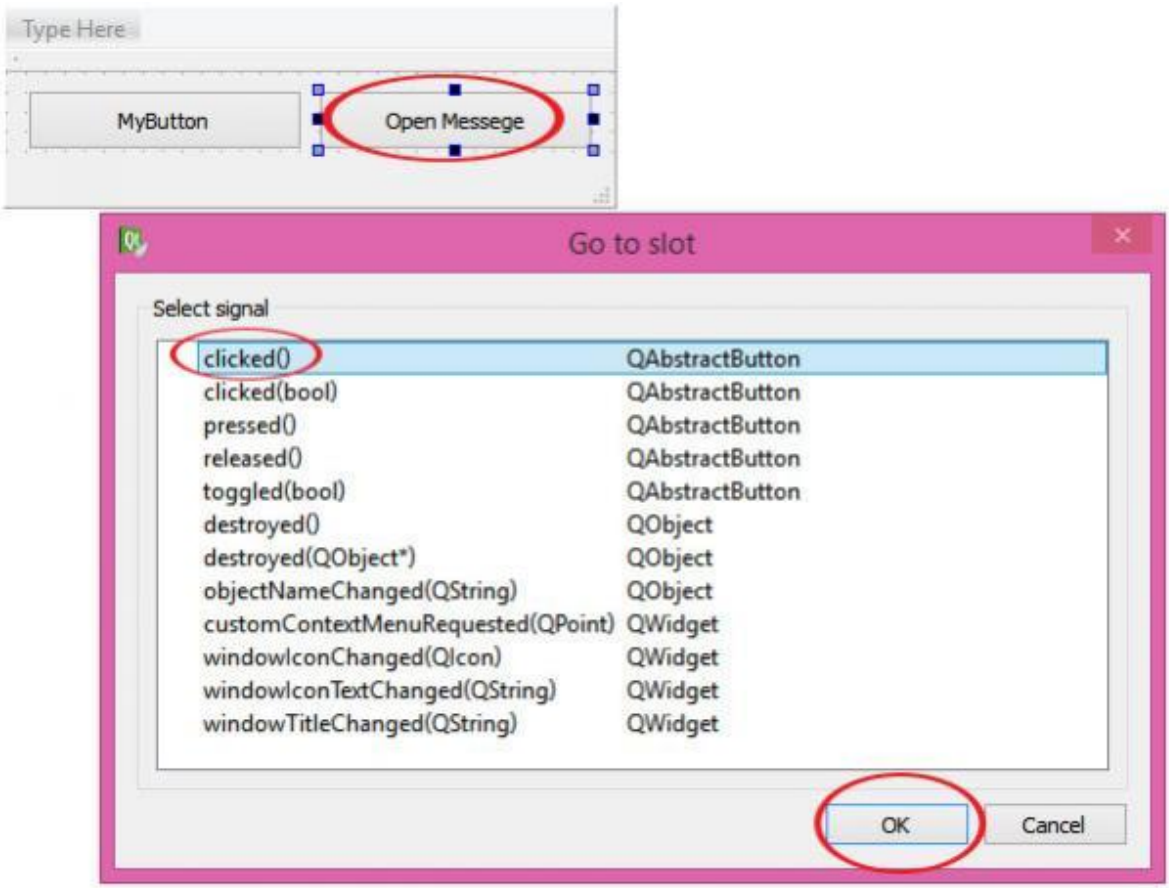
از قسمت WidgetBox دکمه مورد نظر رو انتقال میدیم به صورت زیر:



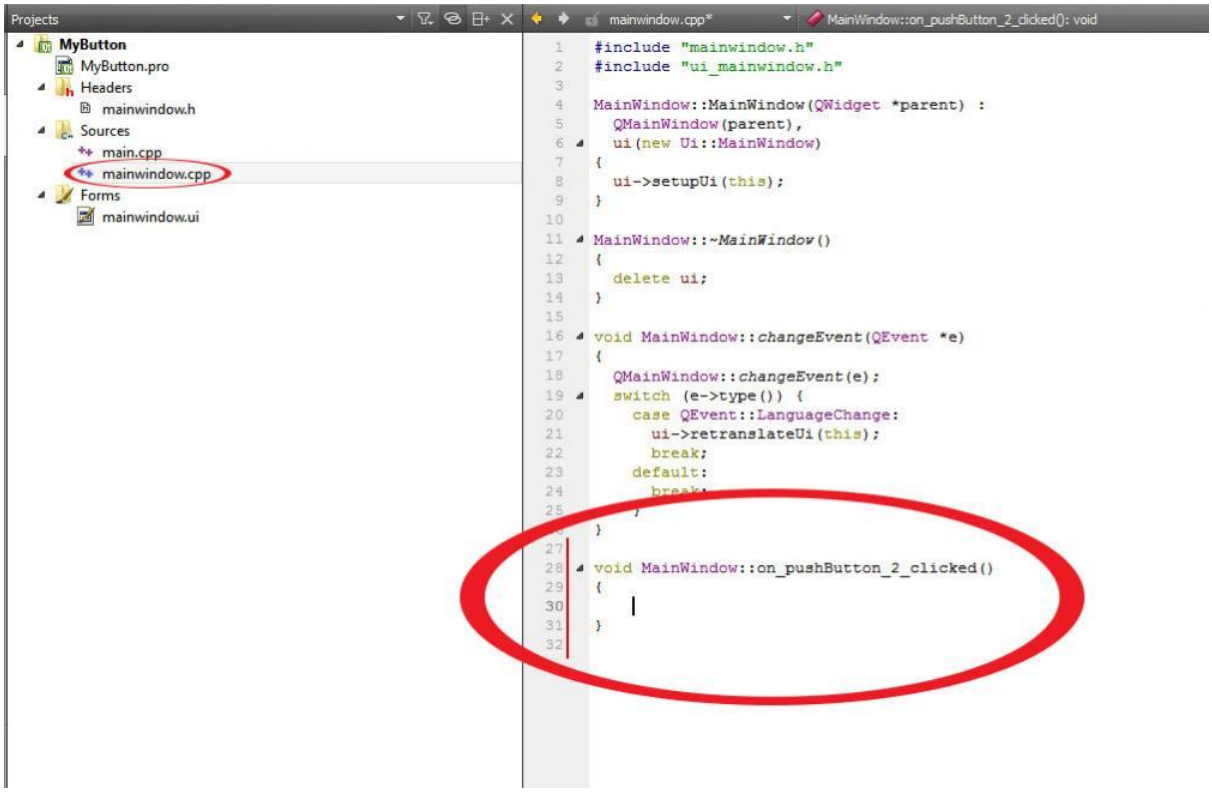
در این مرحله با کلیک راست روی دکمه و انتخاب گزینه مشخص شده Got to slot میتونیم رخداد مورد نظر رو برای دکمه در نظر بگیریم



و در این مرحله میبینید که انواع رخداد ها برای این کنترل وجود داره که من از رخداد کلیک شدن استفاده میکنم.



بعد از انجام این مراحل تابع مربوط به این رخداد برای این شناسه از دکمه ایجاد می شود و شما میتوانید کد مورد نظرتون رو برای این رخداد از کلیک بنویسید.



من کد زیر رو برای نمایش ی پیغام وارد میکنم تا با کلیک شدن روی دکمه پیغام نمایش داده بشه.

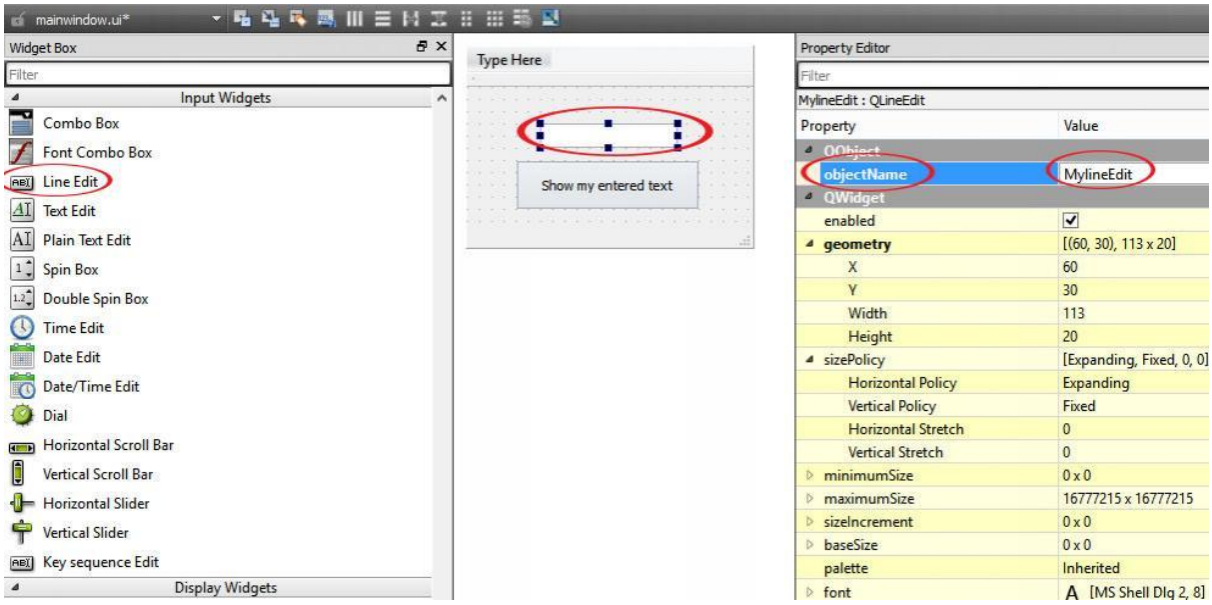
```
QMessageBox msgBox;
msgBox.setText("The document has been modified.");
msgBox.exec();
```

مرحله چهاردهم : معرفی و کار باLineEdit

کنترل LineEditهمان کنترل TextBox معروف هستش که وظیفه این گرفتن اطلاعات از کاربر هست Propertyهای زیادی هم داره و همینطور رویداد های مختلف.

بجکتم رو ایجاد میکنم روی فرم و بعد یه دکمه میزارم و میخوام با کلیک روی دکمه متنی رو که تو LineEditمینویسم نمایش بدم.

قبل از هر چیز هم نام آبجکتم رو عوض میکنم میزارمش MyLineEdit به صورت زیر...



حالا میرم سراغ کد نویسی ... مستقیم میرم تو رویداد کلیک دکمم مینویسم کد زیر رو...

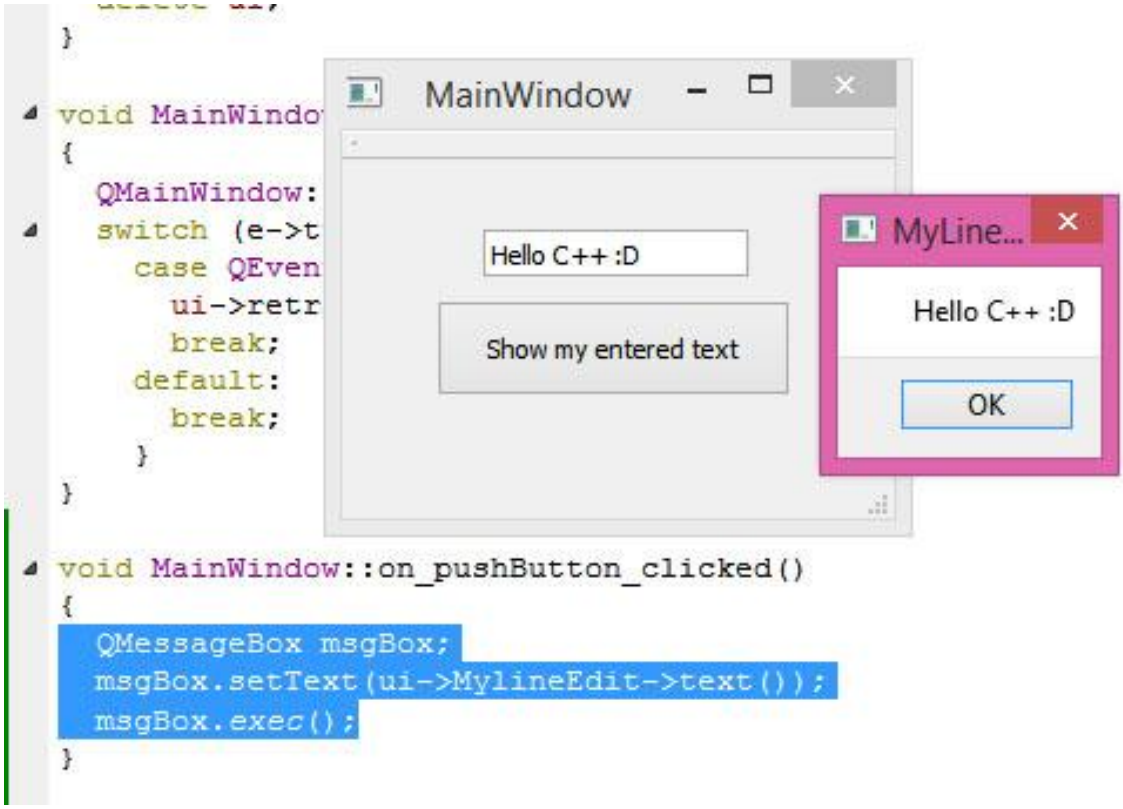
```
QMessageBox msgBox;
msgBox.setText(ui->MylineEdit->text());
msgBox.exec();
```

با این تفاوت که به جای پیغام قبلی خاصیت متن موجود در LineEdit رو فراخوانی میکنم با دستور زیر:

```
ui->MylineEdit->text();
```

این به این معنیه که متن موجود در شیئ کادر ورودی متن من رو بگیر.

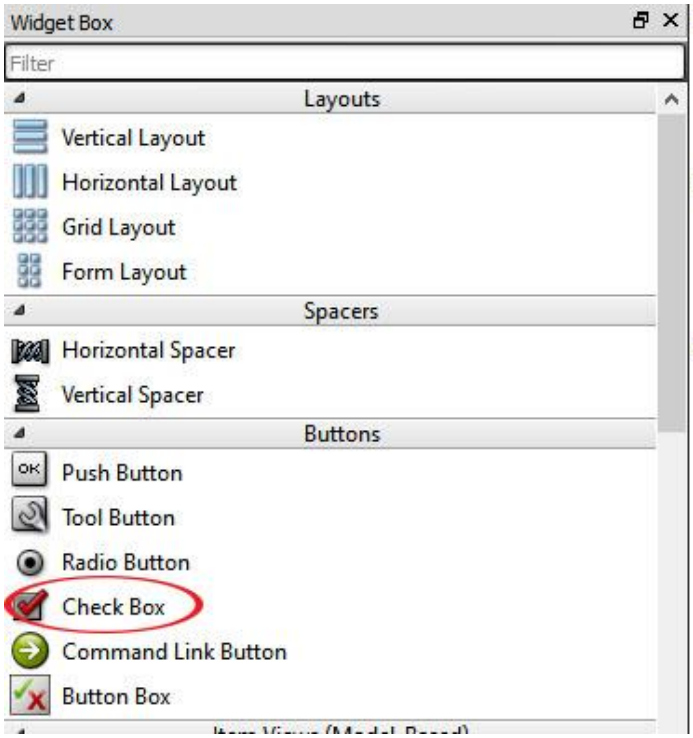
نتیجه شد به صورت زیر:

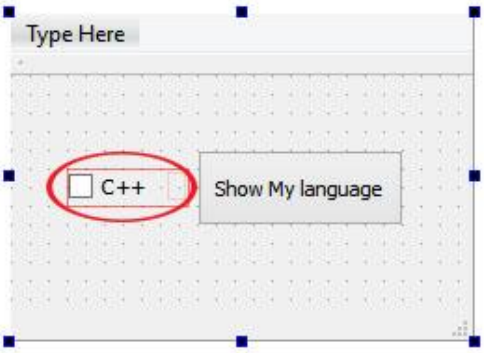


مرحله پانزدهم : معرفی و کار باCheckBox

یکی از کنترل های مهم و کار آمد که من شخصا علاقه خاصی به این کنترل دارم کنترل چک باکس هستش CheckBox

حالا من به صورت زیر چک باکسمو میارم روی فرم و میخوام با کلیک بر روی دکمه اگه CheckBoxمن تیک خورده باشه بهم پیغام بده که تیک خورده یا متن مورد نظرم رو نمایش بده یا اگر تیک نخورده باشه بهم بگه که اقا تیک نخورده انتخابش کن !یا هر چیزی که خوشتون میاد.





و کد زیر در رویداد دکمه من:

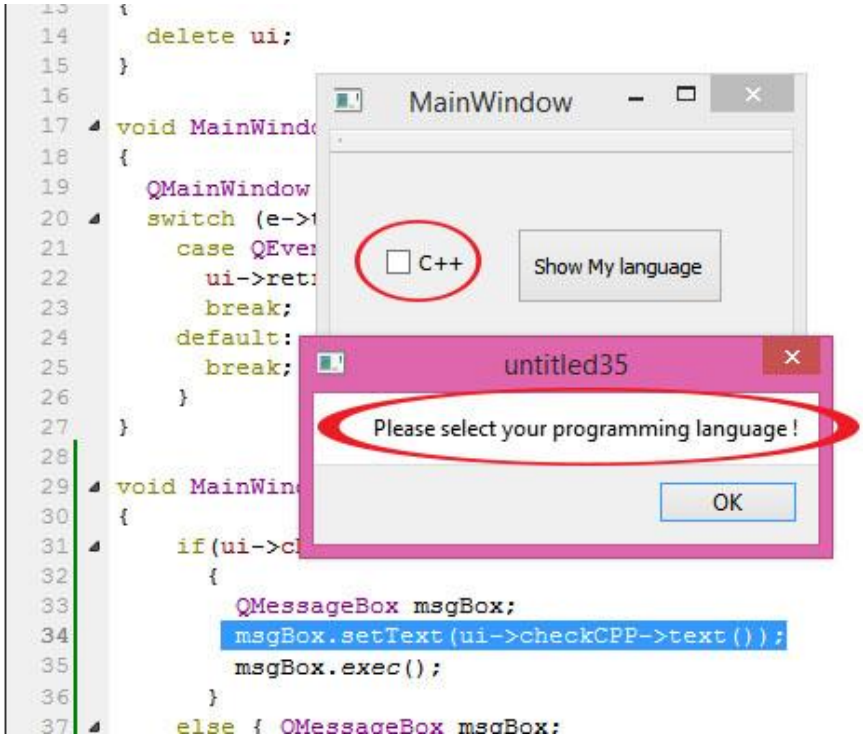
```
if (ui->checkCPP->isChecked())
{
QMessageBox msgBox;
msgBox.setText("C++");
msgBox.exec();
}
else {
QMessageBox msgBox;
msgBox.setText("Please select your programming language !");
msgBox.exec();
}
```

به جای این متن ها میتونید متن خود CheckBox رو هم بگیرید. به صورت زیر:

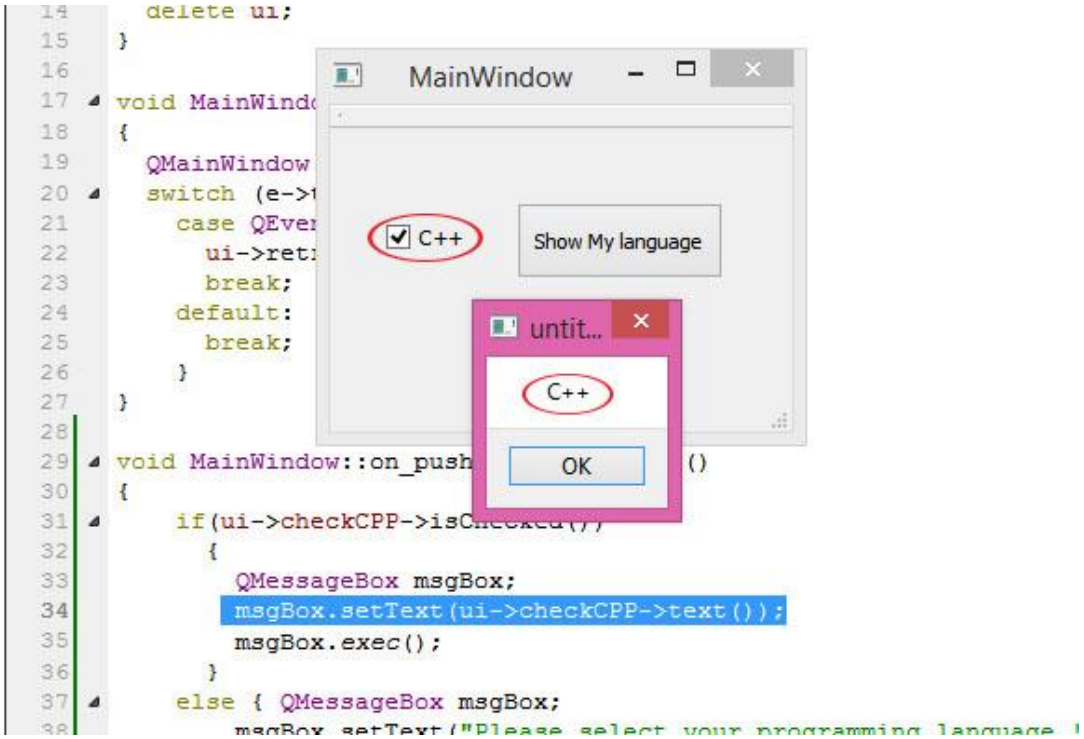
```
msgBox.setText(ui->checkCPP->text());
```

و در نهایت نتیجه به صورت زیر هستش:

در صورتی که تیک نخورده باشه نتیجه زیر:

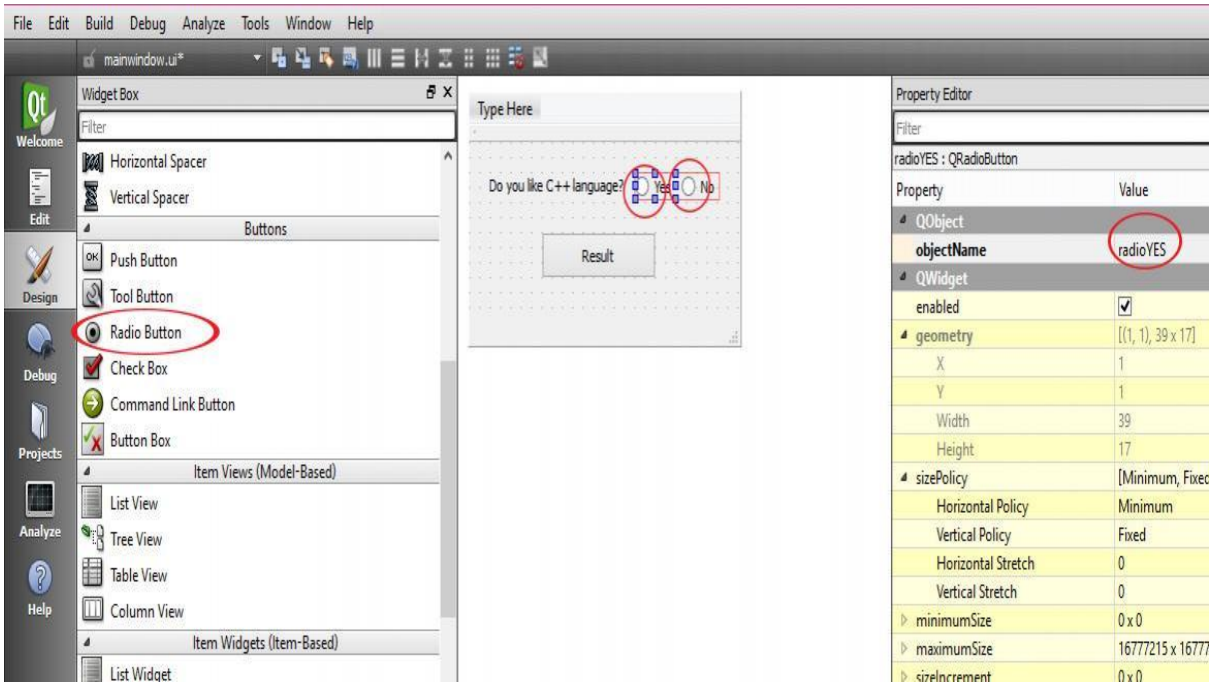


در صورتی که تیک خورده باشه به صورت زیر:



مرحله شانزدهم : معرفی و کار باRadioBox

یکی از کارآمد ترین آبجکت های انتخابی که معمولا برای مشخص کردن یکی از گزینه های پیشنهادی مانند سوالهای چند گزینه ای و ... استفاده از RadioBox هستش که من در این مرحله با دو کنترل رادیو میخوام نتیجه انتخاب شده رو نمتایش بدم تقریبا روشش شبیه به CheckBox هست ولی با کمی تغییرات.



فرمم رو ایجاد و کنترلم رو درج میکنم و برای کنترل Yes و No اسم و مشخصه خاصی میدم.

حالا من میخوام وقتی فرمم بود میشه یکی از این دو کنترل به صورت پیشفرض مقدار دهی بشه که مقدار برابر این کنترل ها همیشه true یا false خواهد بود یعنی در حالت انتخاب true و در غیر اینصورت.false

در قسمت فرم لود من کد زیر رو مینویسم.

```
ui->radioYES->setChecked(true);
```

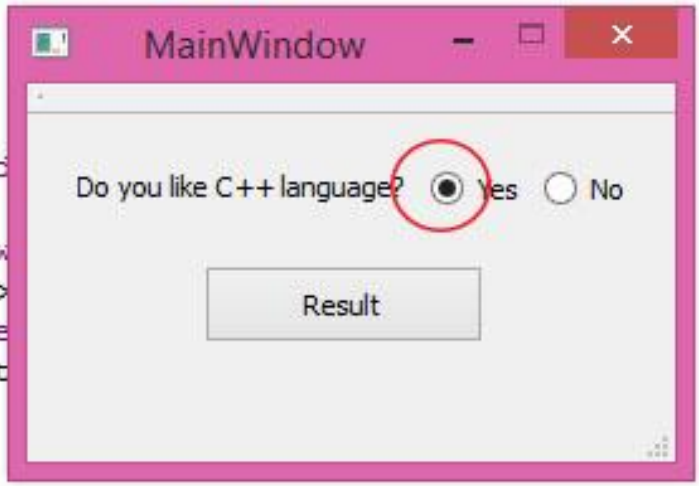
گزینه ui که کلا مربوط به فرم طرحمون هستش یعنی ui و بعد هر شیئی که روی اون قرار داره.

اسم و یا شناسه کنترل رو مینویسم و بعد مقدار true رو براش در نظر میگیرم تا وقتی فرمم لود شد رادیویی که Yes هست در حالت انتخاب قرار بگیره اگه این کارو انجام ندم هیچ یک از این کنترل ها مقدار دهی نخواهند شد یا باید اینجوری توسطز کد مقدار اولیه برای یکی از این دوتا بدیم یا باید از قسمت properties

مشخص کنیم که معمولا هیچوقت هردو باهم انتخاب نخواهند شد !!! فقط یکی از این دو قابل انتخاب هست وقتی دیگری رو انتخاب کنید گزینه بعدیش از حالت true به حالت false مقدار دهی میشه.

نتیجه فرم لود به صورت زیر:

```
{
    ui->setupUi(this);

    //My radio value
    ui->radioYES->setChecked(true);

}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWind
{
    QMainWindow
    switch (e->
      case QEve
        ui->ret
        break;
      default:
        break;
    }
```



و اما الان میخوام در رویداد کلیک دکمه خودم کدی بنویسم که وقتی Yes انتخاب شد یه پیغام و وقتی No انتخاب شد یه پیغام یا هرچیزی که مد نظرمون باشه رو برامون بده.

```
void MainWindow::on_pushButton_clicked()
{


    if (ui->radioYES->isChecked() == true)
    {
        QMessageBox msgBox;
        msgBox.setText("Yes ! I like C++ programming languge! that's
powerfull and faster than ever !");
        msgBox.exec();
    }
    else if (ui->radioNO->isChecked() == true)
    {
```

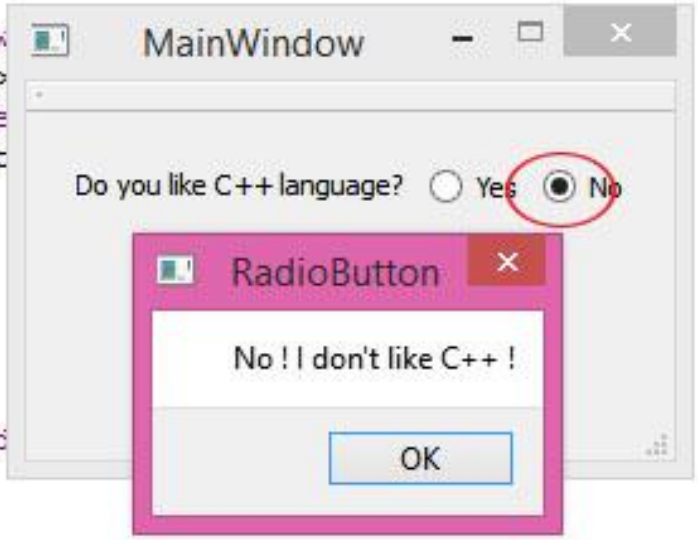
```
        QMessageBox msgBox;
        msgBox.setText("No ! I don't like C++ !");
        msgBox.exec();
    }

}
```

این قسمت من میتونستم مشخص کنم که اگه RadioYes مقدارش true بود فلان کار و اگر مقدارش false بود فلان کار ولی خب خواستم از مقدار موجود در RadioNo هم استفاده کنم در کل میشه از هر دو حالت هم استفاده کرد بنا بر سلیقه خودتون! ...

نتیجه به صورت زیر خواهد بود:

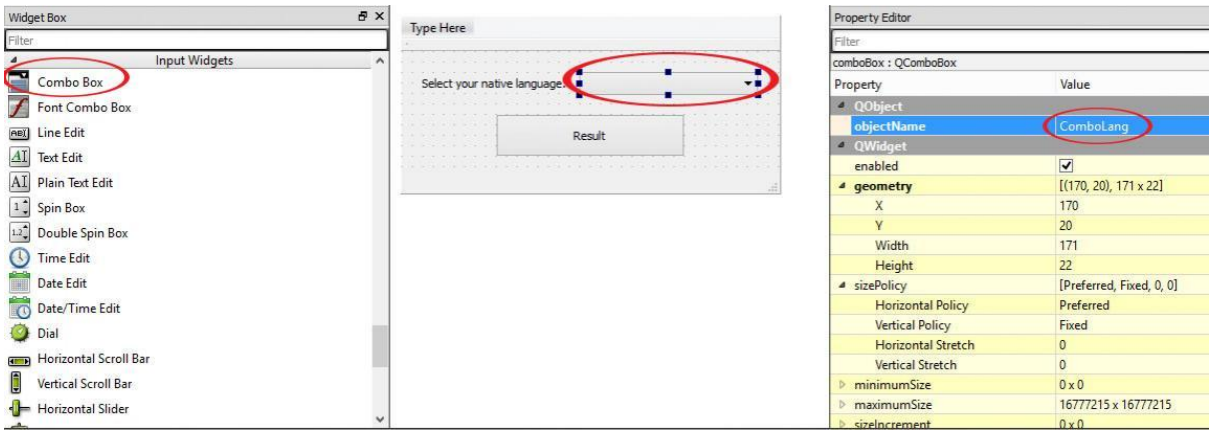

```
        msgBox.setText("Yes ! I like C++ programming languge
        msgBox.exec();
    }
  else if(ui->radioNO->isChecked() == true)
    {
```

```
  delete ui;
}

void MainWindow::changeEvent(QEvent *e)
{
  QMainWindow
  switch (e->
    case QEve
      ui->ret
      break;
    default:
      break;
    }
}

void MainWind
{

  if(ui->radioYES->isChecked() == true)
    {
      QMessageBox msgBox;
        msgBox.setText("Yes ! I like C++ programming languge
        msgBox.exec();
```



مرحله هفدهم : معرفی و کار باComboBox

و اما کار با ComboBox به صورت زیر آبجکت رو انتخاب و نام گذاری میکنم...

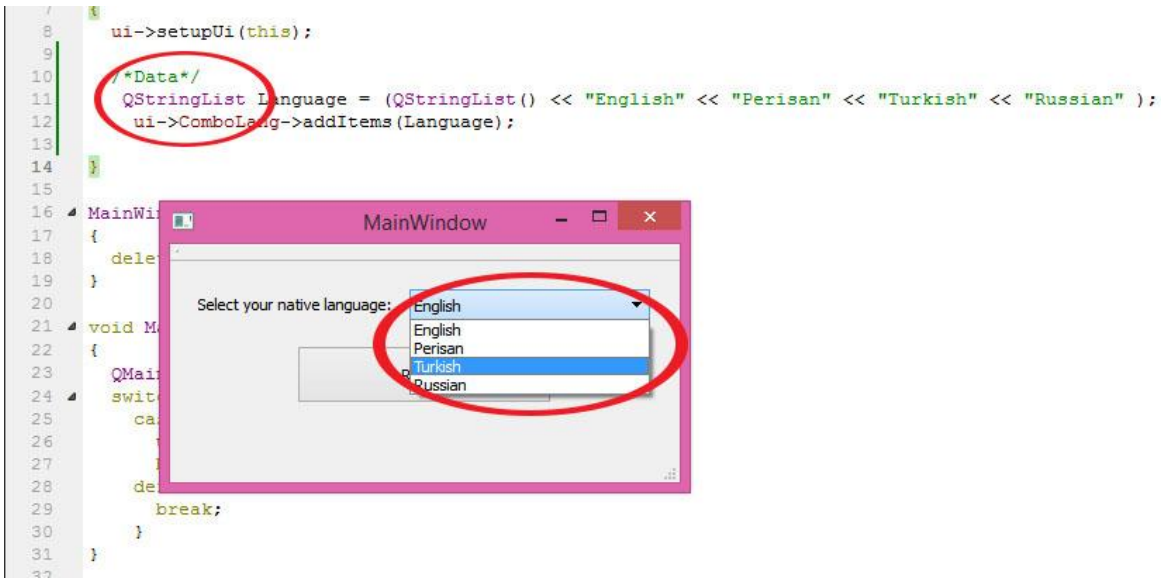


یه توضیحی در رابطه با خصوصیات این کنترل بدم که معمولا در حالت لیست داده های خودش رو میگیره و خروجی رو هم به صورت لیست نمایش میده و در نهایت برای گرفتن مقدار انتخاب شده هم میتونیم از حالت های مختلفی مثل بازگشت دهنده از طریق CurrentText و یا CurrentIndex استفاده کنیم به صورت زیر من ابتدا در فرم لود خودم مقدار هایی رو که میخوام در لیست باکسم نمایش داده بشه تا کاربر بتونی یکیشو انمتخاب کنه رو مینویسم.

```
/*Data*/
QStringList Language = (QStringList() << "English" << "Perisan" <<
"Turkish" << "Russian");
ui->ComboLang->addItems(Language);
```

خب به این قسمت دقت کنید بر اساس خاصیت مقدار دهی شده در این کنترل یک نوع جدید از نوع QStrinListبا نام Language در نظر گرفتم و بعد مقادیر مورد نظرم رو در داخل این لیست توسط عملگر درج << وارد کردم در خط بعدی توسط addItems که نوع آرایه ای یا لیست های خاص رو میگیره توسط Language وارد میکنم.

نتیجه باید به صورت زیر باشه:



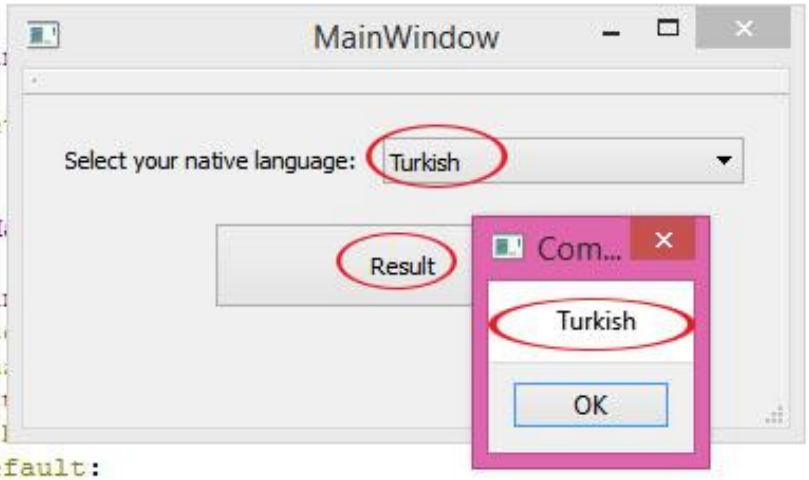
حالا در نظر دارم از خاصیت های انتخابی این کنترل استفاده کنم و مقدار های انتخاب شده رو برگردونم به صورت زیر:

```
QString CurrentVal;
CurrentVal = ui->ComboLang->currentText();
```

```
QMessageBox msgBox;
msgBox.setText(CurrentVal);
msgBox.exec();
```

خب ابتدا چیکار کردم یه نوع متنی تعریف میکنم و بعد چون از نوع متن برگشتی یعنی CurrentText میخوام استفاده کنم نوع رشتم رو معرفی کردم و بعد مقدار این متغیرم رو برابر مقدار متن انتخاب شده COmboBoxخودم قرار میدم و نتیجه میشه به صورت زیر:

```
9    ui->setupUi(this);
10
11   /*Data*/
12    QStringList Language = (QStringList() << "English" << "Perisan
13     ui->ComboLang->addItems(Language);
14
15 }
16
17 MainWi
18 {
19   dele
20 }
21
22 void M
23 {
24   QMai
25   swit
26     ca
27       t
28       b
29     default:
30       break;
31     }
32 }
33
```



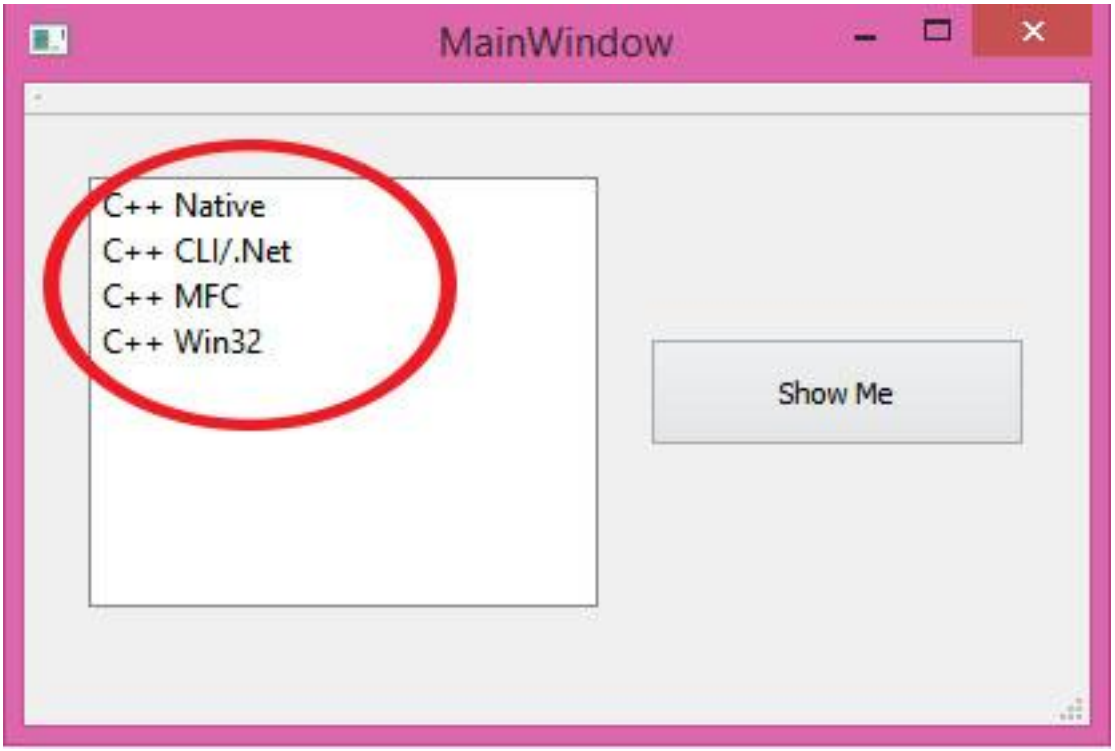
برای مقداری از نوع int هم میتونید متغیر خودتون رو از نوع int تعریف کرده و مقدار CurrentIndex رو بهش اختصاص بدین و کلی کارای دیگه...

مرحله هجدهم : معرفی و کار باListWidget

این کنترل مثل ComboBox عمل میکنه ولی با این تفاوت که محتویات و آیتم های این کنترل به صورت یکجا در دسترس هستن و میشه به صورت یک لیست به آیتم ها دسترسی داشت یه چیزی مثل DataGrid ولی برای آیتم هایی که قراره انتخاب شوند.

به صورت زیر ایجادش میکنیم و در فرم لود همانند ComboBox ایتم های مورد نظر رو بهش میدیم.

```
/*Data*/
QStringList C_Types = (QStringList() << "C++ Native" << "C++ CLI/.Net" <<
"C++ MFC" << "C++ Win32");
ui->listWidget->addItems(C_Types);
```



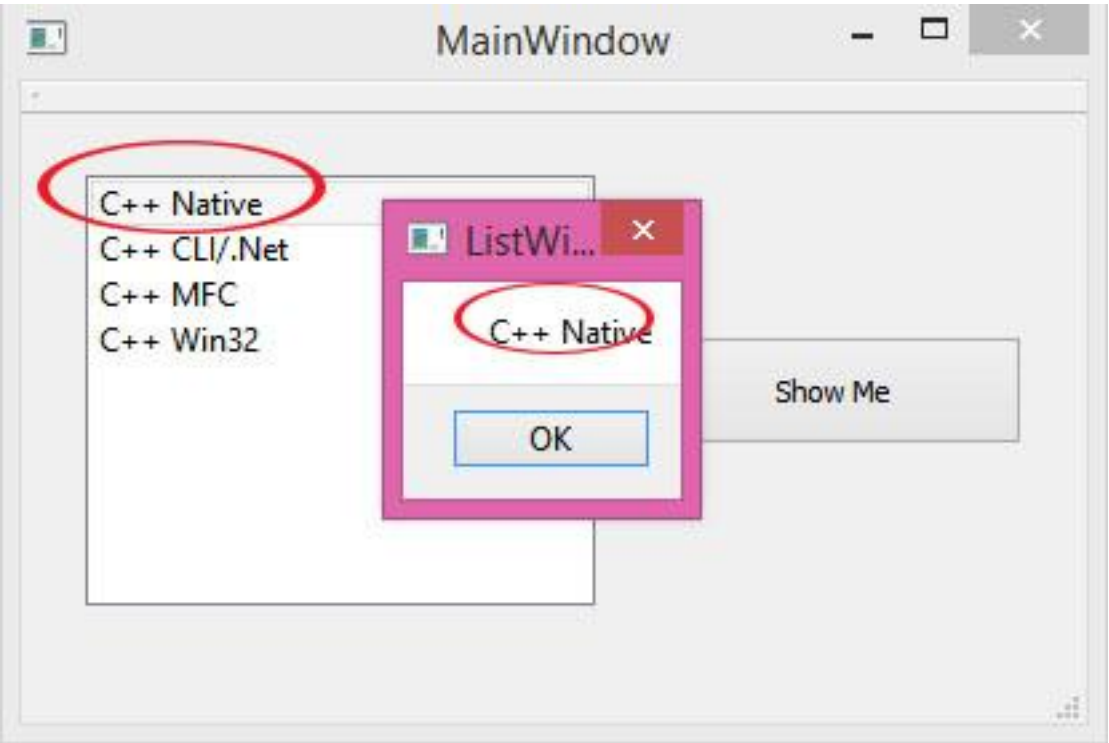
حالا میخوام مقدار انتخاب شده رو نمایش بدم که یکمی متفاوتر از ComboBox خواهد بود به صورت زیر :

```
QString Val;
Val = ui->listWidget->currentItem()->text();
QMessageBox msgBox;
msgBox.setText(Val);
msgBox.exec();
```

خب متغیرم رو در نظر میگیرم و بعد برابرش میکنم با **متن مقدار آیتم انتخاب شده** در لیست باکسم!

و در نهایت مقدار تبدیل شده به رشته رو میفرستم به.MSG

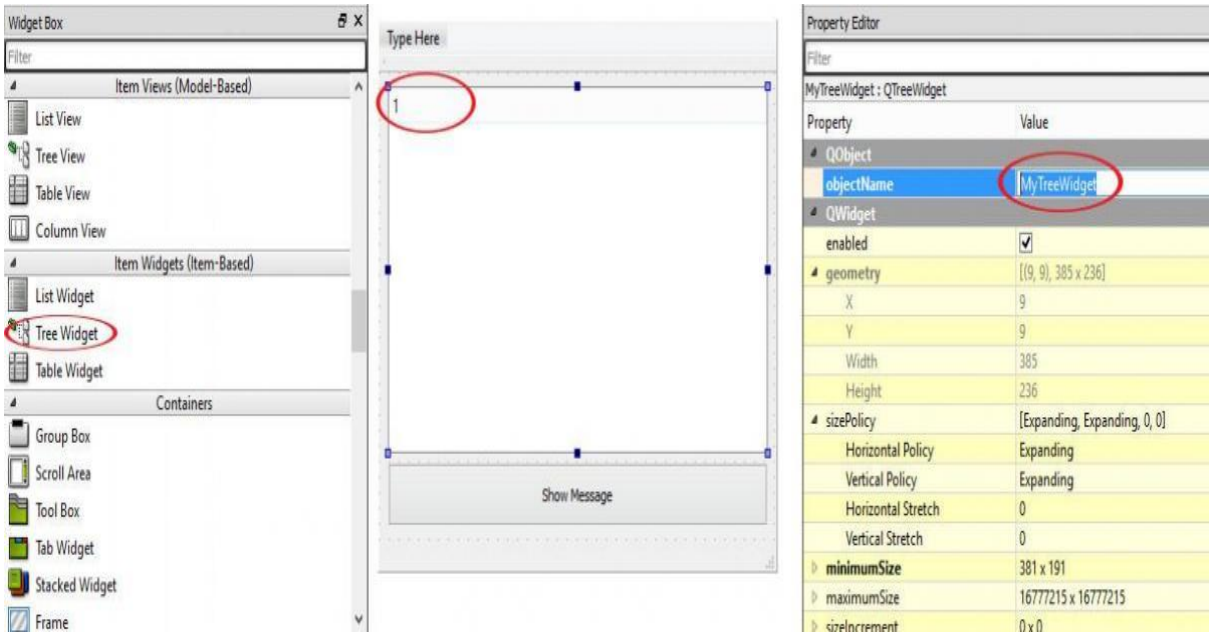
نتیجه به صورت زیر:



مرحله نوزدهم : معرفی و کار باTreeWidget

معرفی و کار با لیست های درختی TreeWidget / به این صورت است که در بسیاری از مواقع که مقادیر قابل انتخاب ما حاوی شاخه ها و زیر شاخه هایی هستند و برنامه نویس باید امکان این رو برای کابر فراهم کنه تا کاربر هم شاخه ها و هم زیر شاخه های بخشی رو که میخواد انتخاب کنه رو ببینه به طور کلی لیست انتخاب به صورت درختی نمایش داده خواهد شد.

به صورت زیر من لیستم رو ایجاد میکنم:



در رابطه با این شیئ باید اینمطور توضیح داد که خاصیت والد و فرزند رو در خودش تخصیص داده است برای مثال من نمونه زیر رو بخوام روی این لیست نمایش بدم به صورت زیر:

کتاب C++ 11

- *فصل اول*
- *فصل دوم*
- و...

کلا یک والد یا زیر شاخه داریم و بقیه موارد زیر شاخه یا همان فرزند هستند در اصل همون Parent و Child

حالا روشی که من میخوام برای این بکار ببرم به صورت زیر خواهد بود ابتدا برای دسترسی و راحتی کار۲ تا تابع تعریف میکنم برای parent وchild

در قسمت mainwindow.h پایینتر ازQ_OBJECT

```
void AddRoot(QString name, QString Discreption);
void AddChild(QTreeWidgetItem * parent, QString name, QString Discreption);
```

خب اینجا من متد ها و همچنین نام و نوع تابع رو مشخص کردم که برای AddRoot یا همان والد نام و توضیحلات رو در نظر میگیرم و برای AddChild یا همان فرزند علاوه بر نام و توضیحات هر یک شناسه تعیین کننده برای شناساندن ایتم مورد نظر برای سر شاخه هستش و اونم از نوع QTreeWidgetItem قرار میدم.

مرحله بعد در فایل اصلیمون mainwindow.cpp توابع رو صدا زده و بدنه توابع رو میسازم به صورت زیر :

```
void MainWindow::AddRoot(QString name, QString Discreption)
{


}

void MainWindow::AddChild(QTreeWidgetItem* parent, QString name, QString
Discreption)
{


}
```

حالا من به کد های زیر در داخل هریک از این دو تابع نیاز دارم به صورت زیر:

```
void MainWindow::AddRoot(QString name, QString Discreption)
{

QTreeWidgetItem * MyItem = new QTreeWidgetItem(ui->MyTreeWidget);


MyItem->setText(0, name);
MyItem->setText(1, Discreption);
ui->MyTreeWidget->addTopLevelItem(MyItem);

AddChild(MyItem, "C++", "Native / Objective");
AddChild(MyItem, "Java", "Objective");

}


void MainWindow::AddChild(QTreeWidgetItem* parent, QString name, QString
Discreption)
{

QTreeWidgetItem * MyItem = new QTreeWidgetItem();
MyItem->setText(0, name);
MyItem->setText(1, Discreption);
parent->addChild(MyItem);

}
```

در تابع اول AddRoot ابتدا یک کپی از QTreeWidgetItem میسازم تا به عنوان آیتم های درختی برام کار کنه و در مرحله بعدی گفتم که در ایندکس اول یعنی ۰ متنی رو که تابع میگیره رو ست کن و در خط بعدیش دقیقا مثل این گفتم توضیحات رو در ایندکس ۱ یعنی دوم این رو ست کن و بعد از این گفتم لیست درختی من این دو گزینه رو از نوع آیتم های سر شاخه قرار بده با کد addTopLevelItem : این کار رو انجام میدم.

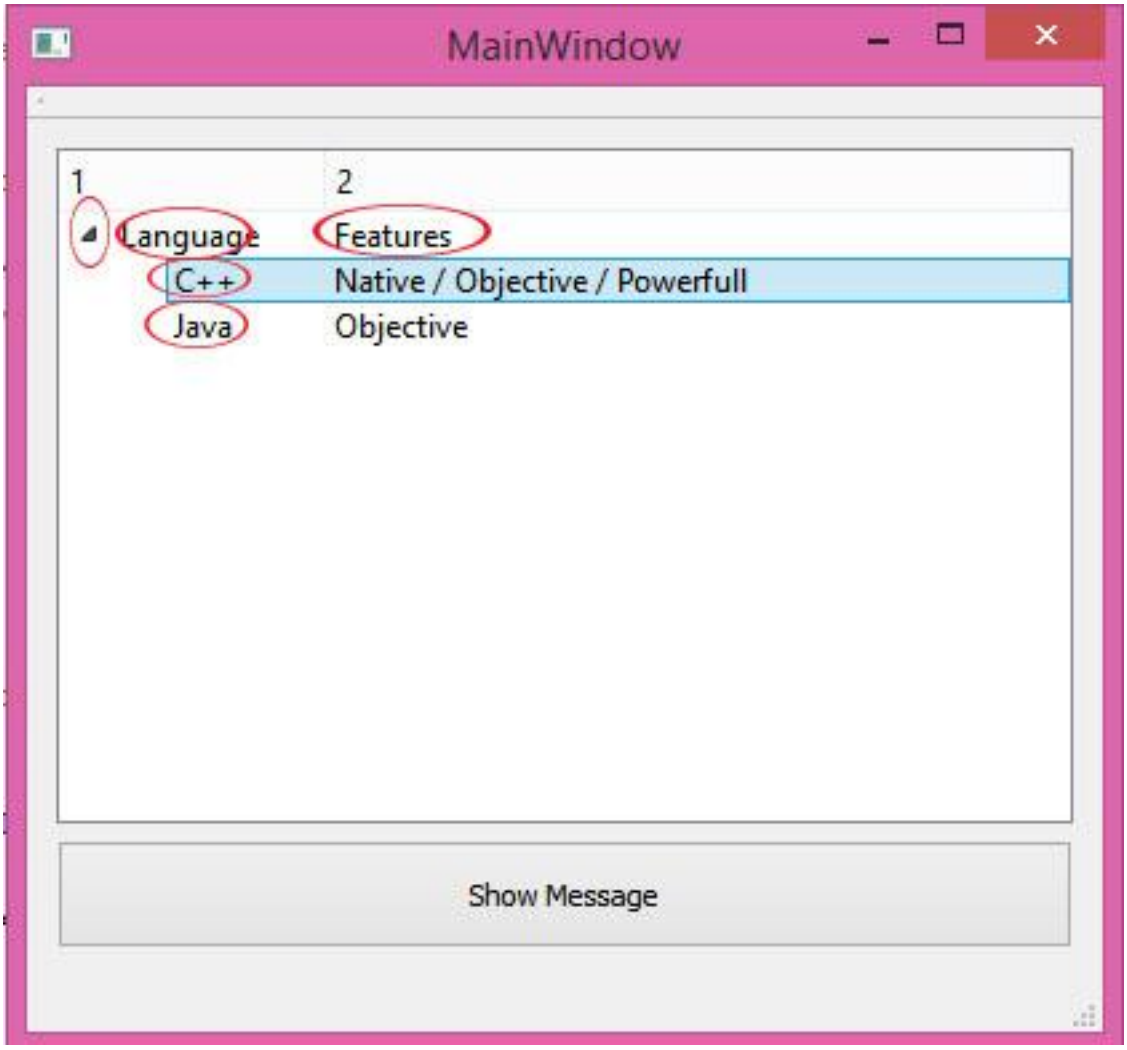
در خط بعدش اومدم گفتم ایتم های AddChild برابر باشه از آیتم های تعریف شده به صورت بالا که از تابع AddChildفراخوانی میشوند به صورت زیر...

سه خط اول در تابع AddChild مثل تابع قبلی برای عنوان و شناسایی اینهاست و در خط آخر گفتم که آقا آیتم های والد من زیر مجموعشون برابر باشه با آیتم های فرزندی که ایجاد میشوند.

در مرحله بعد در قسمت فرم لود لازمه که تعداد ستون ها رو همراه با عناوینش مشخص کنم به صورت زیر:

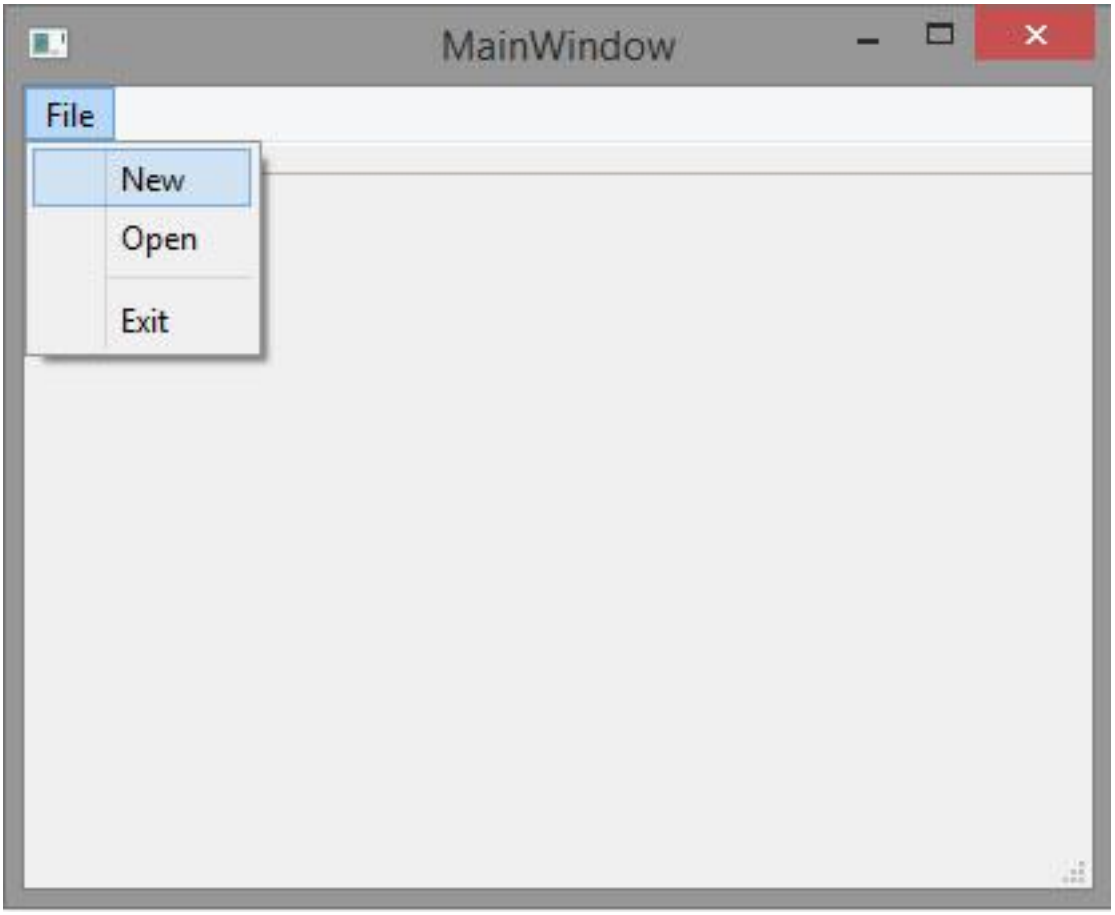
```
ui->MyTreeWidget->setColumnCount(2);
AddRoot("Language", "Features");
```

مشخصه که گفتم دو ستون از نوع زبان و ویژگی های زبان درج کن و بعد از این تابع Addrootفراخوانی شده و تابع AddChild همینطور و نتیجه میشه به صورت زیر...

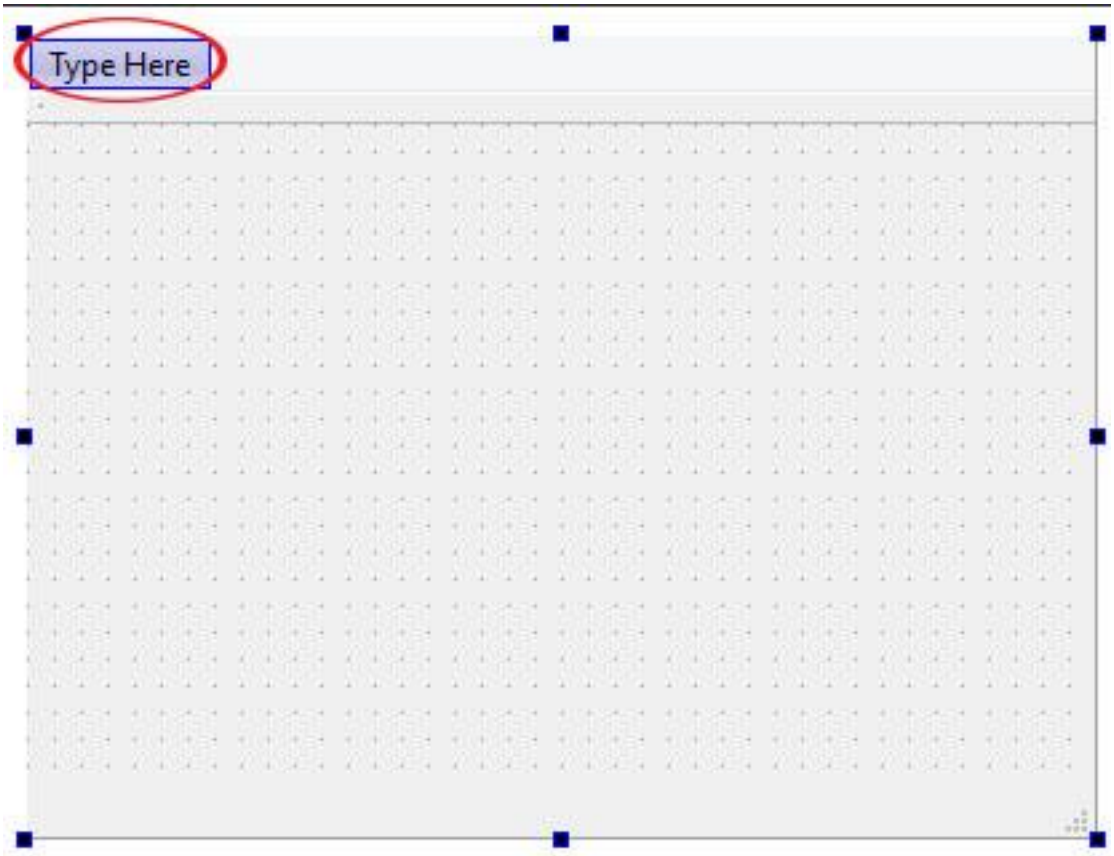


مرحله بیستم : معرفی و کار با Action ها

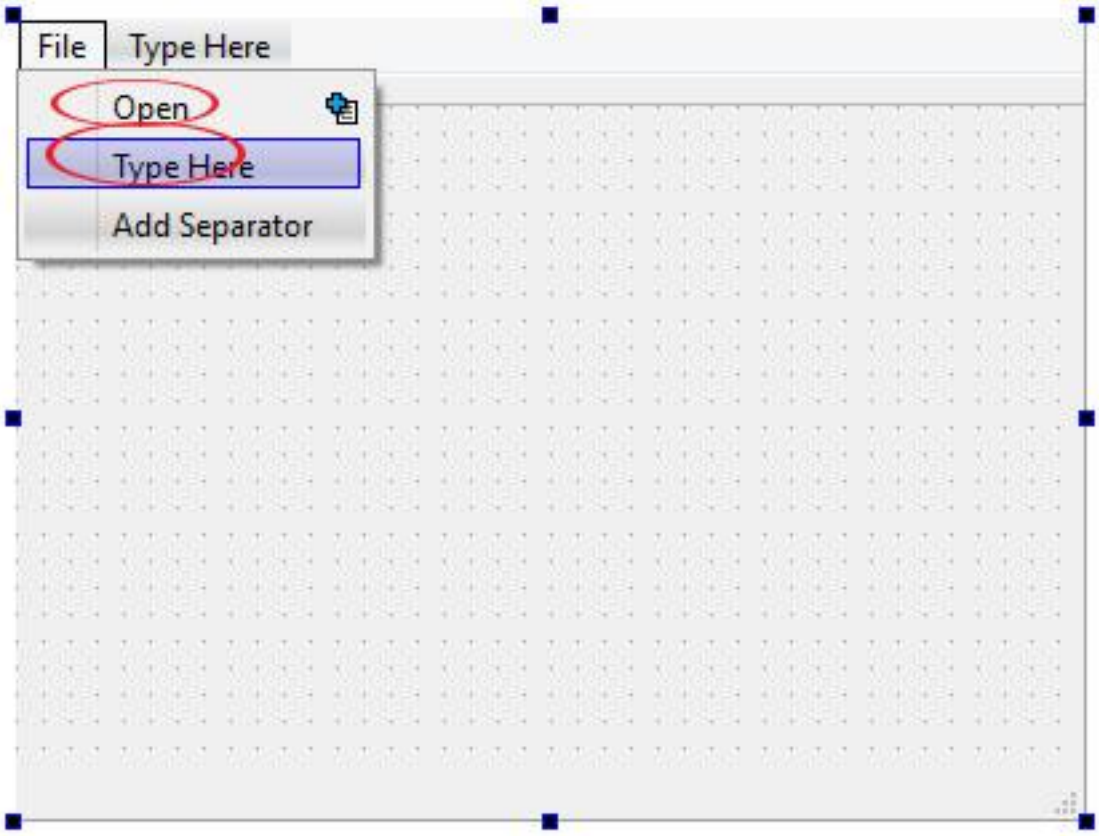
در رابطه با اکشن ها میتونم بگم همون منوهای برنامه هستش که توسط هر اکشن ایجاد شده یک منو با مشخصه خاص تعریف و قابلیت ها و همچنین Event های خاص خودش رو ایجاد میکنه برای مثال من برای اینکه روی فرمم منو های زیر رو ایجاد میکنم به صورت زیر:



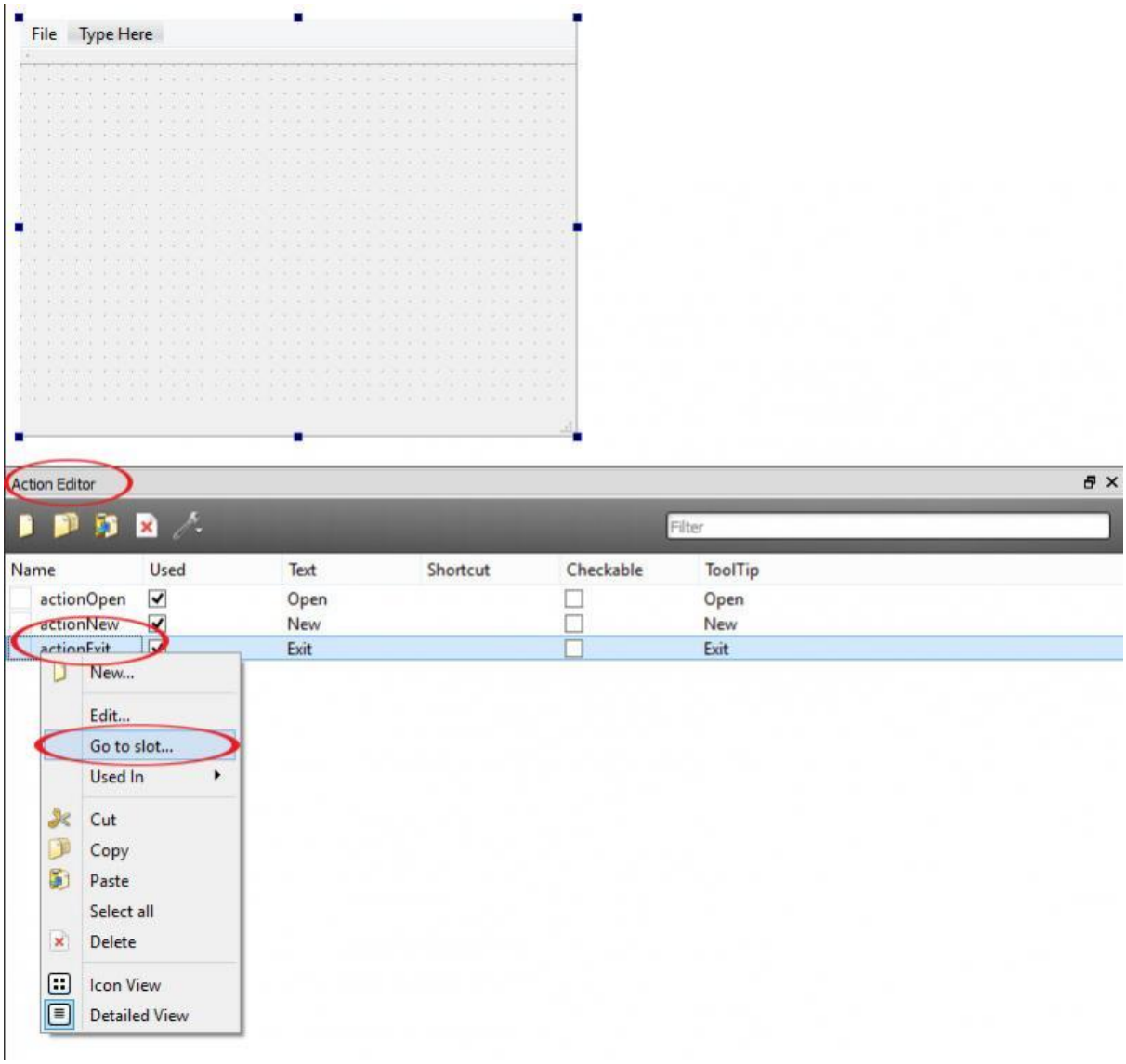
چکار باید انجام بدم ... خیلی راحت روی فرمم قسمت منو روی Type Here کلیک میکنم و اسم منورو مینویسم مثل تصویر زیر...



توجه کنید که هر والد میتونه زیر شاخه های خودش رو داشته باشه همچنین میتونید با کلیک روی Add Separatorخط فاصلخ بین منوها به عنوان جدا کننده ایجاد کنید.



هریک از منو هایی که ایجاد میشه به عنوان یک Action هستش مثل دیگر کنترل های دارای Event ها و Slotهای خاص خودشون هستند. حالا برای اینکه مثلا با کلیک کردن رو دکمه Exit یا همون منوی ساخته شده با نام Exit یه کاری انجام بدیم چکار باید کنیم ؟ خب باید در رخداد کلیک شدن یا انتخاب شدن این Actionکه در اینجا با شناسه actionExit در دسترس هستش وارد بشم و کدهای مورد نظر رو برای انجام کار مورد نظر بنویسیم به صورت زیر قسمت اکش ها و استلات ها مشخص هست:



در ادامه خیلی راحت با وارد شدن به اسلات مورد نظر و ایجاد تابع رخداد triggered مراحل زیر رو ادامه میدیم...

```
void MainWindow::on_actionExit_triggered()
{

}
```

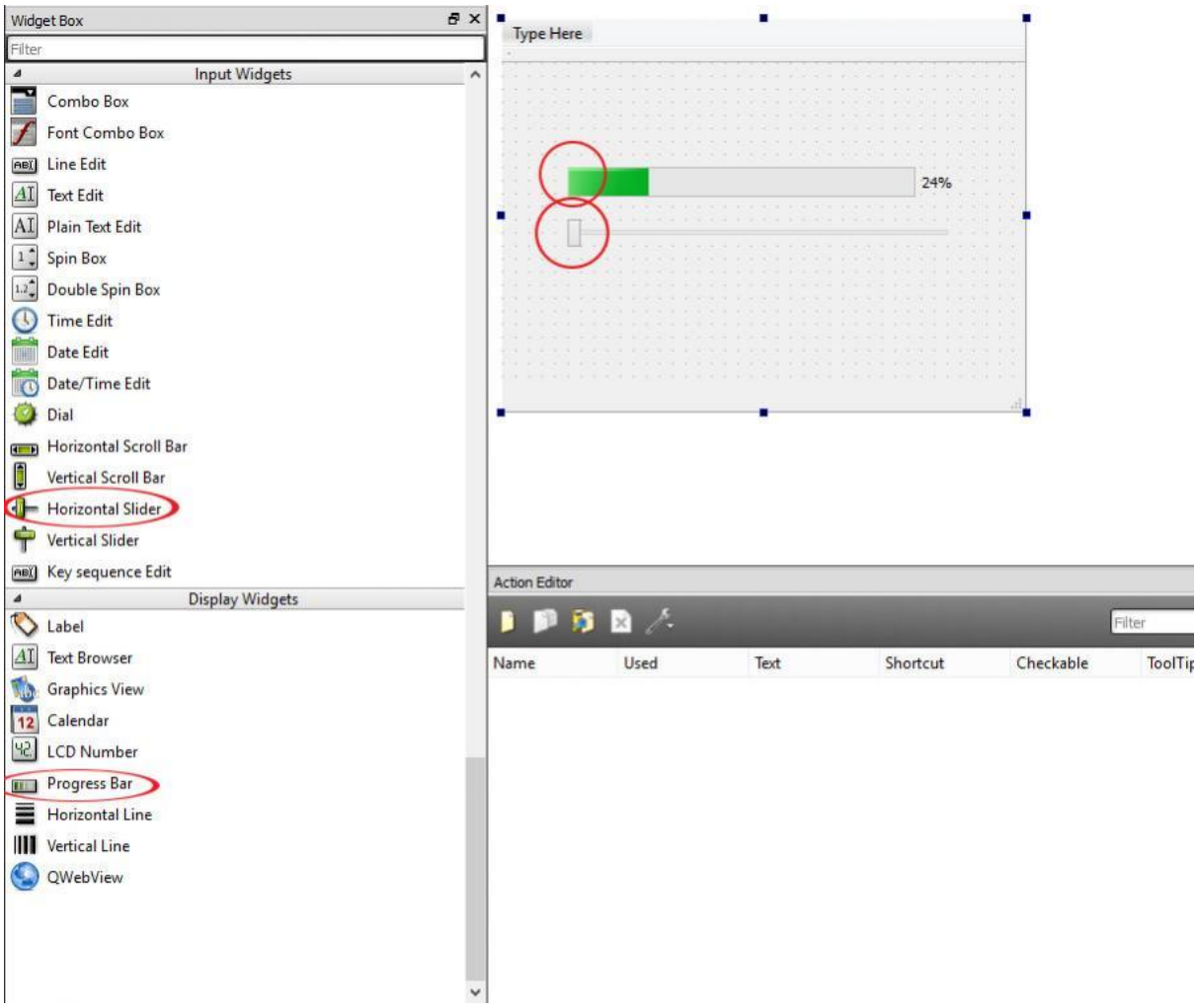
بین تابع شما میتونید هر کدی رو که لازمه بنویسید حالا در این مثال من کد ساده ای برای خروج از برنامه رو مینویسم و تمام ! به صورت زیر...

```
QApplication::exit();
```

و در نهایت با کلیک بر روی منو اکشن Exit برنامه باید از پروسه خارج شود.

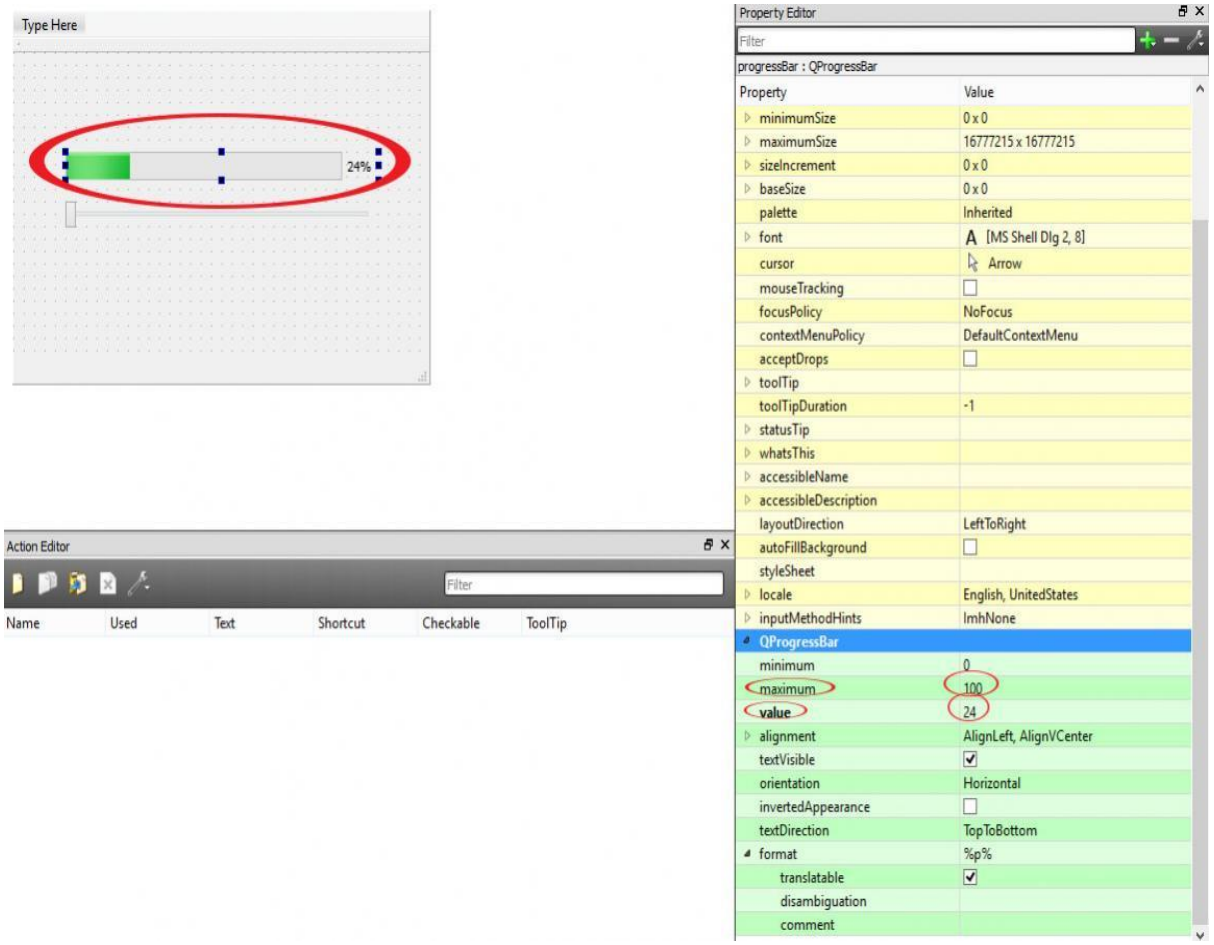
مرحله بیست و یکم : معرفی و کار با Progress و Slider ها

در رابطه با کنترل های progress و Slider چون پراپرتی های تقریبا مشابهی دارند این دو رو باهم ادغام میکنیم به صورت زیر روی فرمم یک Progressbar و یک Slider ایجاد میکنم.



خب قبل از هر چیزی توضیحی در رابطه با این بدم که آبجکت Progressbar همیشه بر اساس پراپرتی valueمقداردهی میشه یعنی اگر ما برای این کنترل Maximum value رو برابر ۱۰۰ قرار بدیم و پراپرتی value این برابر باشه با ۲۴ اونوقت از ۱۰۰ درصد ۲۴ درصد مقدار این کنترل ارزش دهی میشود.

به صورت زیر میتونید تنظیماتش رو اختصاص بدین...



حالا میاییم سراغ کد نویسی... قبل از هر چیز شناسه Progresbar من هستش MyprogressBar و شناسه Slider من هستش MyhorizontalSlider

میخوام دقیقا وقتی مقدار Sliderbar تغییر میکنه مقدار Progressbar هم بر اساس اون تغییر کنه به چه روشی میشه این کار رو انجام داد ؟!

در Qt ما بحث **Signal** و **Slot** هارو داریم که به طور خلاصه بخوام خیلی ساده توضیح بدم این خواهد بود : فرض کنید من یه یک شخصی ایمیل میزنم و در قبال اون ایمیل انتظار دریافت جوابش رو دارم حالا من اینجا به عنوان Signal و شخص دریافت کننده ایمیل من هستش Slotو ارتباط بین این دو مورد توسط چه چیزی انجام پذیره ؟ تابعی داریم به نام **connect** که وظیفه این تابع برقراری ارتباط باین Signal و Slotهستش به صورت زیر عمل خواهیم کرد:

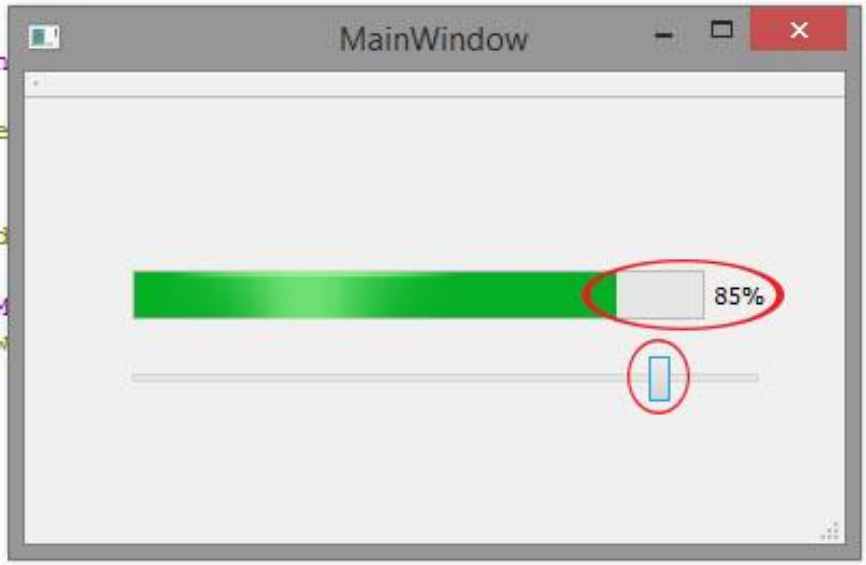
این کد من هست :

```
connect(ui->MyhorizontalSlider, SIGNAL(valueChanged(int)), ui-
>MyprogressBar, SLOT(setValue(int)));
```

در این کد من تابع connect رو فراخوانی کردم این تابع ابتدا Signal رو میگیره ور بعد Slot رو برای سیگنالی که ارسال کرده ستش میکنه خیلی راحت بخواه ساده تر بگم وظیفش همینیه که میبینید سیگنال رو ارسال میکنه و اسلات رو برای سیگنال ارسال شده دریافت میکنه حالا اینجا مقدار ارزشی که ما ارسال میکنیم و دریافت میکنیم از نوع int هستش که در هر دو شیئ ما مقدار value ارسال و دریافت میشه.

حالا بعد از اجرا کردن برنامه هر بار من مقدار Slider رو تغییر بدم مقدار Progressbar هم تغییر خواهد یافت به صورت زیر:

```
3
4   MainWindow::MainWindow(QWidget *parent) :
5     QMainWindow(parent),
6     ui(new Ui::MainWindow)
7   {
8     ui->setupUi(this);
9     connect(ui->MyhorizontalSlider,SIGNAL(valueChanged(int)),ui->My
10
11  }
12
13  Main
14  {
15    de
16  }
17
18  void
19  {
20    QM
21    sw
22
23
24
25
26
27
28  }
29
```

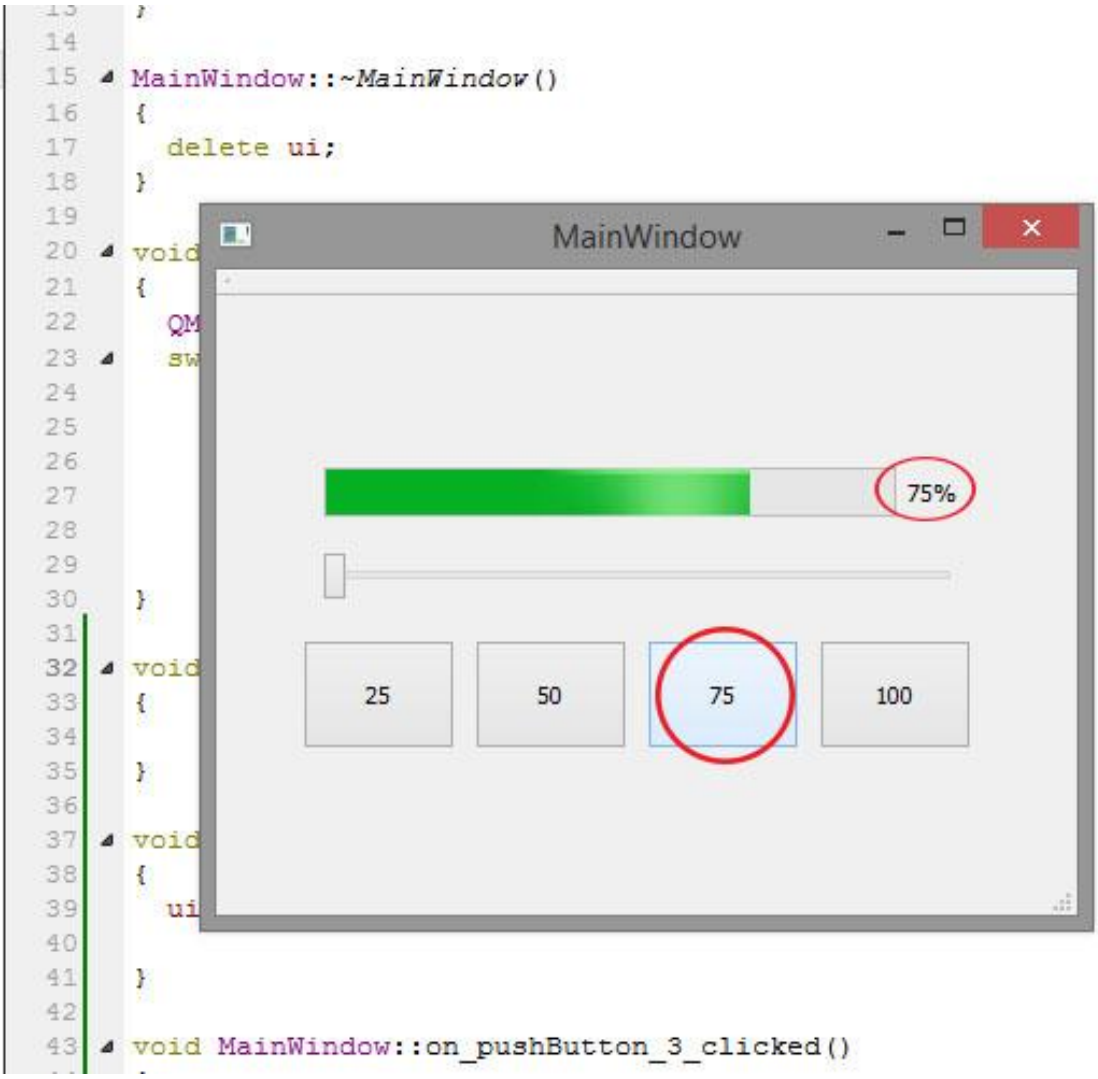


خب این یه خلاصه ای از مفهوم سیگنال و اسلات بین این دو کنترل یکم توضیح بدم در رابطه با ست کردن جداگانه مقدار در کنترلProgressbar

برای این کار از پراپرتی setValue باید استفاده کنیم من برای مثال چهار مقدار متفاورت رو با استفاده از کنترل Button ایجاد و ارسال میکنم به صورت زیر:

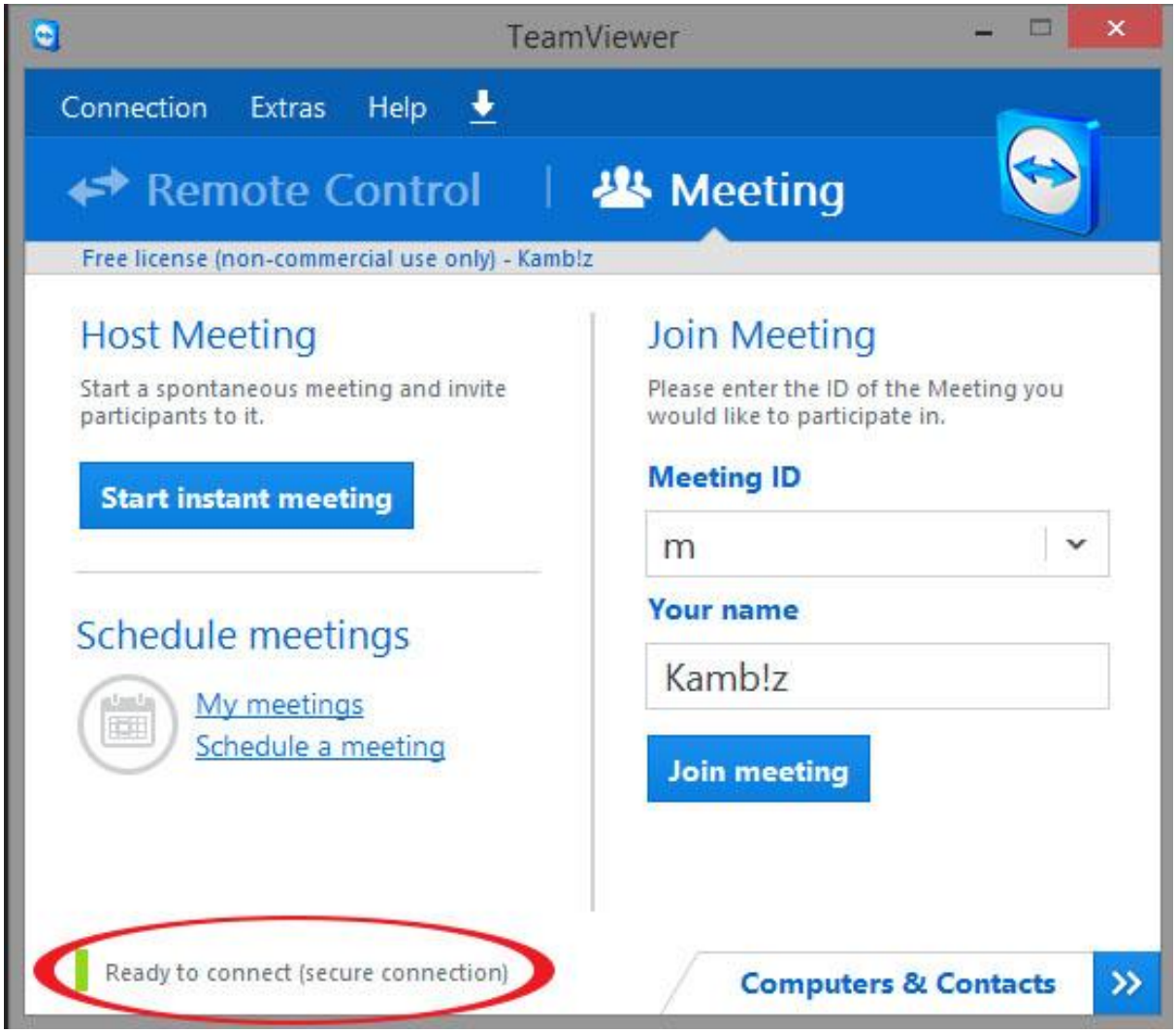
```
[/CPP][CPP]ui->MyprogressBar->setValue(75);
```

این کد مقدار ۷۵ رو از ۱۰۰ درصد ظرفیت progressbar ارسال میکنه.



مرحله بیست و دوم : معرفی و کار با Statusbar ها

یکی از امکانات دیگه در نرم افزار ها که در طراحی ازش میشه استفاده کرد ایجاد نوار وضعیت هستش که خود نوار وضعیت میتونه شامل هر نوع کنترلی باشه برای مثال به صورت زیر نرم افزار TeamViewer رو مثال میزنم زیرش در قسمت پایین که به رنگ قرمز مشخص کردم متن و وضعیت برنامه رو مشخص میکنه که اصولا در این موارد از StatusBar استفاده میکنند.



خب حالا چطور این کار رو میشه انجام داد اینطور باید بگم که خیلی راحتر از کار در دات نت هست ! واقعا راحته اولا خود Qt چهارچوب استانداردی رو داره که اصولا همه Widget ها به صورت پیشفرض از این امکانات پشتیبانی میکنند حالا من میخوام یک نوع برچسب به عنوان خوش آمد گویی روی ویجتم درست کنم به صورت زیر...

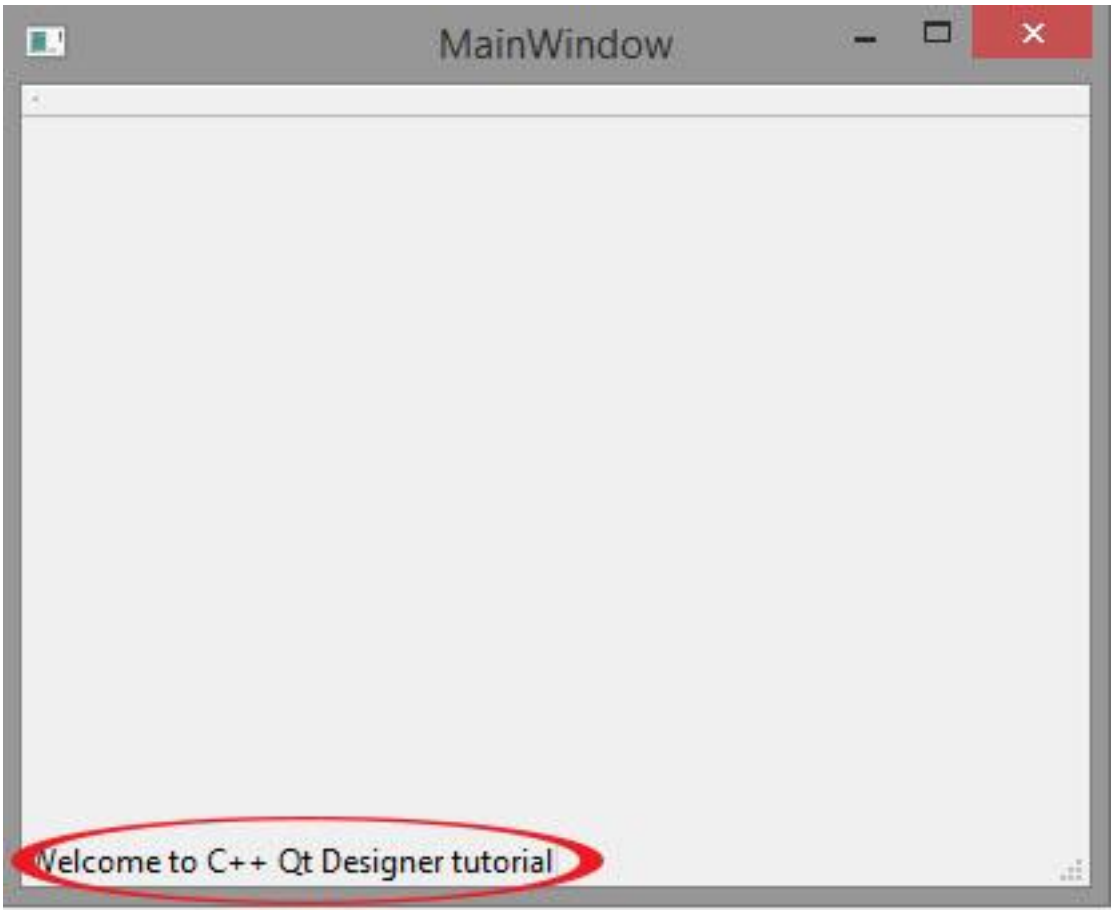
ابتدا یک نوع label میسازم:

```
QLabel *Mylbl = new QLabel();
Mylbl->setText("Welcome to C++ Qt Designer tutorial");
```

خب حالا برای اینکه این برچسب ایجاد شده رو روی Statusbar نمایش بدم باید به صورت زیر عمل کنم :

```
ui->statusBar->addWidget(Mylbl);
```

خیلی ساده و راحت با یه دستور addWidget از Statusbar آبجکت Label رو بهش اضافه میکنم و نتیجه میشه به صورت زیر:



مرحله بیست و سوم: معرفی و کار باMessageBox

خب یکی از مواردی که خیلی پر کاربرد هستش مخصوصا برای نمایش پیغام های خاص مثل (اخطار / اطلاعات / سوال و جواب و ...) استفاده از MessageBox هستش که در C++ در حالت های مختلفی میشه ازش استفاده کرد و من در این آموزش برای Qtاین مثال رو تقریبا سعی میکنم کامل بزنم تا بهتر و مفیدتر باشه.

برای اینکه در برنامتون از MessageBox استفاده کنید ابتدای فایلتون بایت این کلاس رو فراخوانی کنید : QmessageBox

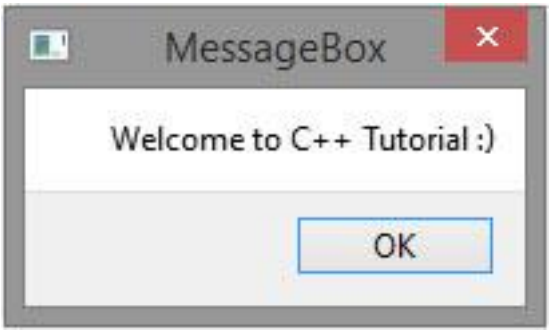
به صورت زیر:

```
#include "QMessageBox"
```

خب حالا من میتونم از این کلاس استفاده کنم ابتدا ازش یه شیئ جدید میسازم:

```
QMessageBox MyMSG;
MyMSG.setText("Welcome to C++ Tutorial :)");
MyMSG.exec();
```

در خط اول از کلاس QMessageBox یک نمونه ساختم و در خط دوم متنی رو درون این آبجکت وارد کردم و در خط بعدی از متد exec برای نمایش Dialog برای پیغام استفاده کردم که نتیجش باید باشه به صورت زیر:



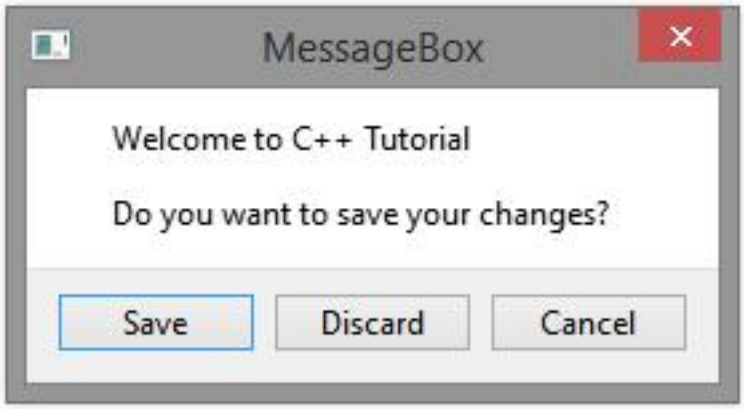
خب حالا من قصد دارم یکمی پیشرفته تر کار کنم با مواردی که میشه در این کلاس ازشون بهرهمند شد آشنا میشیم ... برای مثال من میخوام از این قابلیت برای پرسش سوال استفاده کنم و نتیجش رو با استفاده از Yes یا Noمشخص کنم به صورت زیر کد رو مینویسم:

```
QMessageBox msgBox;
msgBox.setText("Welcome to C++ Tutorial");
msgBox.setInformativeText("Do you want to save your changes?");
msgBox.setStandardButtons(QMessageBox::Save | QMessageBox::Discard |
QMessageBox::Cancel);
msgBox.setDefaultButton(QMessageBox::Save);
int ret = msgBox.exec();
```

خب خط اول و دوم که مشخصه و در خط سوم متنی رو برای نمایش به عنوان سوال پرسی توسط پراپرتی setInformativeTextارسال میکنم در خط بعدیش توسط پراپرتی setStandardButtonsدکمه هایی رو که میخوام کاربر هنگام سوال پرسی ببینه رو مشخص میکنم که در اینجا سه گزینه Save / Cancel / Discardرو من انتخاب میکنم و در خط بعد مشخص کردم که دکمه Save به عنوان دکمه پیشفرض در حالت انتخاب شده قرار بگیره.

و در نهایت پیغام رو نمایش میدم و مقدار بازگشتی حاصل از انتخاب کاربر رو به متغیر ret که از نوع int هستش ارسال میکنم.

به صورت زیر نمایش داده خواهد شد:



خب حالا سوالی این وسط هستش که اگه کاربر هر کلیدی که انتخاب کرد رو و مقدارش رو بازگشت داد بگیره و یه کاری انجام بده مثلا اگه Save رو زد یه کاری و اگه بقیه رو زد کارهای متفاوت دیگری رو انجام بده ... برای این کار به روش زیر عمل میکنم:

```
switch (ret) {
case QMessageBox::Save:
    // Save was clicked
    break;
case QMessageBox::Discard:
    // Don't Save was clicked
    break;
case QMessageBox::Cancel:
    // Cancel was clicked
    break;
default:
    // should never be reached
    break;
}
```

دستور Switch یکی از دستورات معروفی هست که در مواقع پیچیده تر و چند گزینه ای تر بهتر جواب میده.

طرز کارش چطوریه ؟ خب متغیر ret که داشتیم اون بالا هر دکمه ای رو که ما انتخاب کنیم توسط دستور switchارزیابی میشه و نتیجه بر اساس سلیقه مشخص خواهد شد.

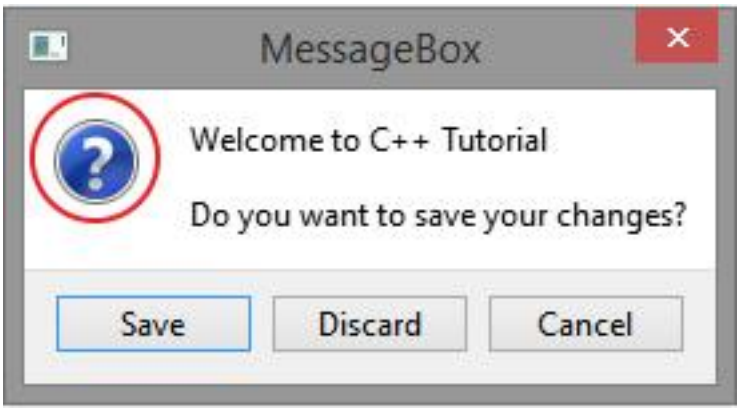
مثلا من در مثالی که زدم به صورت زیر هر انتخابی پیغامش رو ارسال خواهد کرد به یک Label که روی فرمم هست.

```
switch (ret) {
case QMessageBox::Save:
    // Save was clicked
    ui->MyLabel->setText("Save");
    break;
case QMessageBox::Discard:
    // Don't Save was clicked
    ui->MyLabel->setText("Discard");


    break;
case QMessageBox::Cancel:
    // Cancel was clicked
    ui->MyLabel->setText("Cancel");


    break;
default:
    // should never be reached
    break;
}
```

یه Property مهم دیگه ای هم داریم به عنوان ICON که برای این کار هم به صورت زیر عمل کنید :

```
msgBox.setIcon(QMessageBox::Question);
```

نتیجش میشه به صورت زیر:



مرحله بیست و چهارم: معرفی و کار باTimer / Qtimer

در باره Timer هم همونطور که میدونید یکی از مواردی هستش که در ساده ترین حالت میشه ازش ساعت کنونی سیستم رو گرفت و تغییر ساعت / دقیقه / ثانیه رو به صورت Update نمایش داد. برای این کار ما میتونیم کلاس کامل و جامعی رو برای یک Timer اختصاصی در برنامه بنویسیم که بر اساس Signal و Slotهای تعریف شده ازش استفاده کرد.

من در این مرحله نحوه نمایش تغییرات لحظه زمان جاری سیستمم رو نشون میدم که ساعت / دقیقه / ثانیه رو روی فرمم نمایش خواهم داد در حالتی که هر ثانیه به ثانیه عمل Update صورت بگیره.

ابتدا کلاس QTimer رو صدا میزنم:

```
#include <QTimer>
```

در فرم لود کد زیر رو مینویسم :

```
//Start to get System time.


QTimer *timer = new QTimer(this);
timer->setInterval(1000);
timer->start();
connect(timer, SIGNAL(timeout()),
SLOT(updateClock()));
```

در این کد من ابتدا یک نوع کپی از تایمر یعنی **QTimer** ایجاد میکنم در خط بعدی مقدار **1000** میلی ثانیه یا همون ۱ ثانیه رو بهش میدم در خط سوم استارت میزنم و در خط بعدی از تابع **connect** برای ارسال Signalو دریافت جواب از طرف **Slot** رو مینویسم که برای ایجاد اسلات نیاز دارم یه تابعی به نام **updateClock** که زمان سیستم رو برای من بگیره و برگشت بده.

در فایل mainwindow.h اسلات (Slot) مربوط به این تابع رو معرفی میکنم به صورت زیر:

```
//System Clock update
private slots:
void updateClock();
```

نوع تابع رو از نوع تابع با دسترسی **private** به عنوان **Slot** در نظر میگیرم.

حالا بر میگردم فایل mainwindow.cpp و تابع رو مینویسم به صورت زیر:

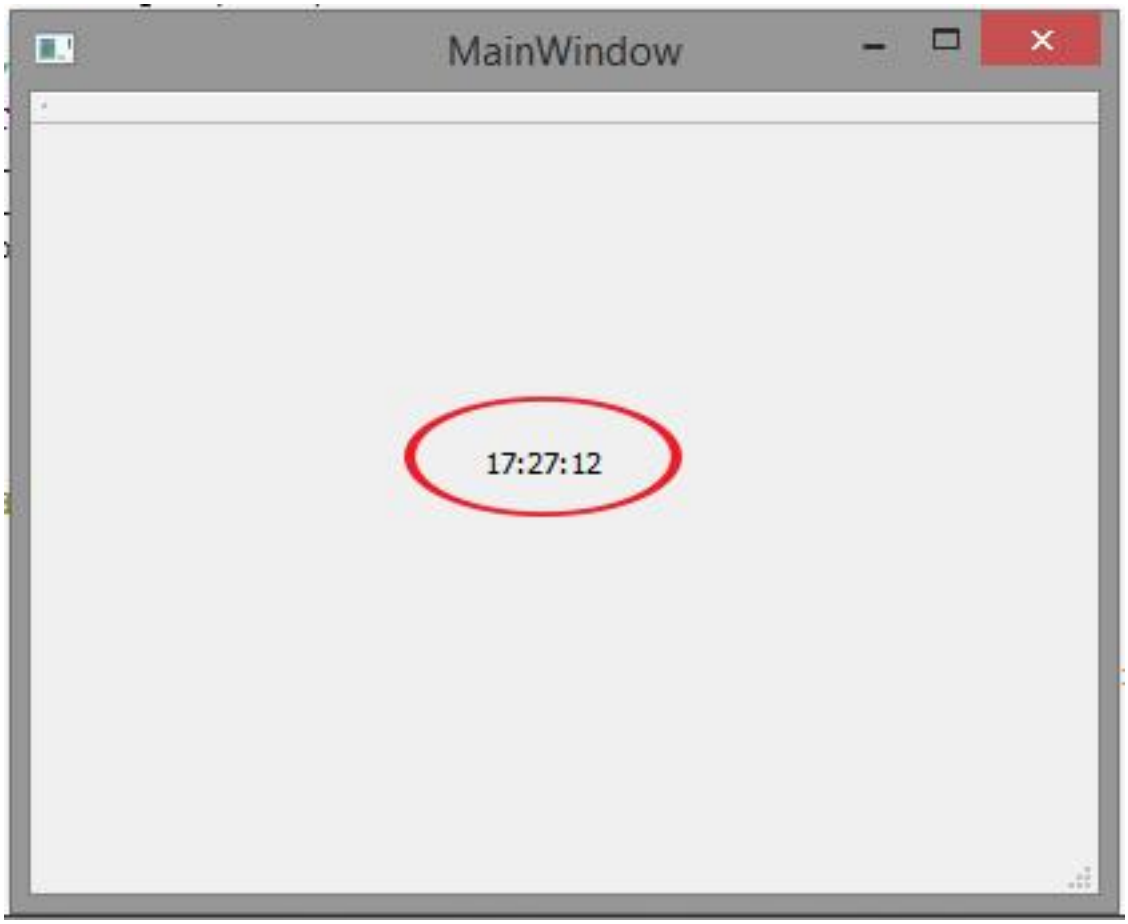
```
void MainWindow::updateClock()
{
QString timeString = QTime::currentTime().toString("hh:mm:ss");
ui->LCurrenTtime->setText(timeString);
}
```

خب اینجا من بدنه تابع رو تشکیل میدم و یک متغیری از نوع رشته QString ایجاد کرده و بعد توسط تابع موجود در کلاس **QTime** دقت کنید این **QTime** هستش با **QTimer** فرق میکنه به این **r** توجه کنید !!! وظیفه این کلاس برگشت دادن زمان جاری سیستم هست که توسط مقدار بازگشتی ثابت (Static) به صورت currentTime رو وارد متغیرم از نوع رشته میکنم البته قبلش تبدیل به رشته میشه و توسط الگوی hh:mm:ss : ساعت : دقیقه :ثانیه ارسال میشه به داخل متغیر.

در خط بعدی هم که مشخصه مقدار متن label موجود روی فرمم رو که اسمش رو **LCurrenTtime** گذاشتم برابر میکنم با متغیری که زمان جاری سیستم در اون ذخیره میشه.

فقط یه نکته که باید برای استفاده از **QTime** هم کلاسش رو وارد فایل مکنید به صورت زیر و نتیجه نهایی...

```
#include "QTime"
```



مرحله بیست و پنجم : معرفی و کار با Thread ها

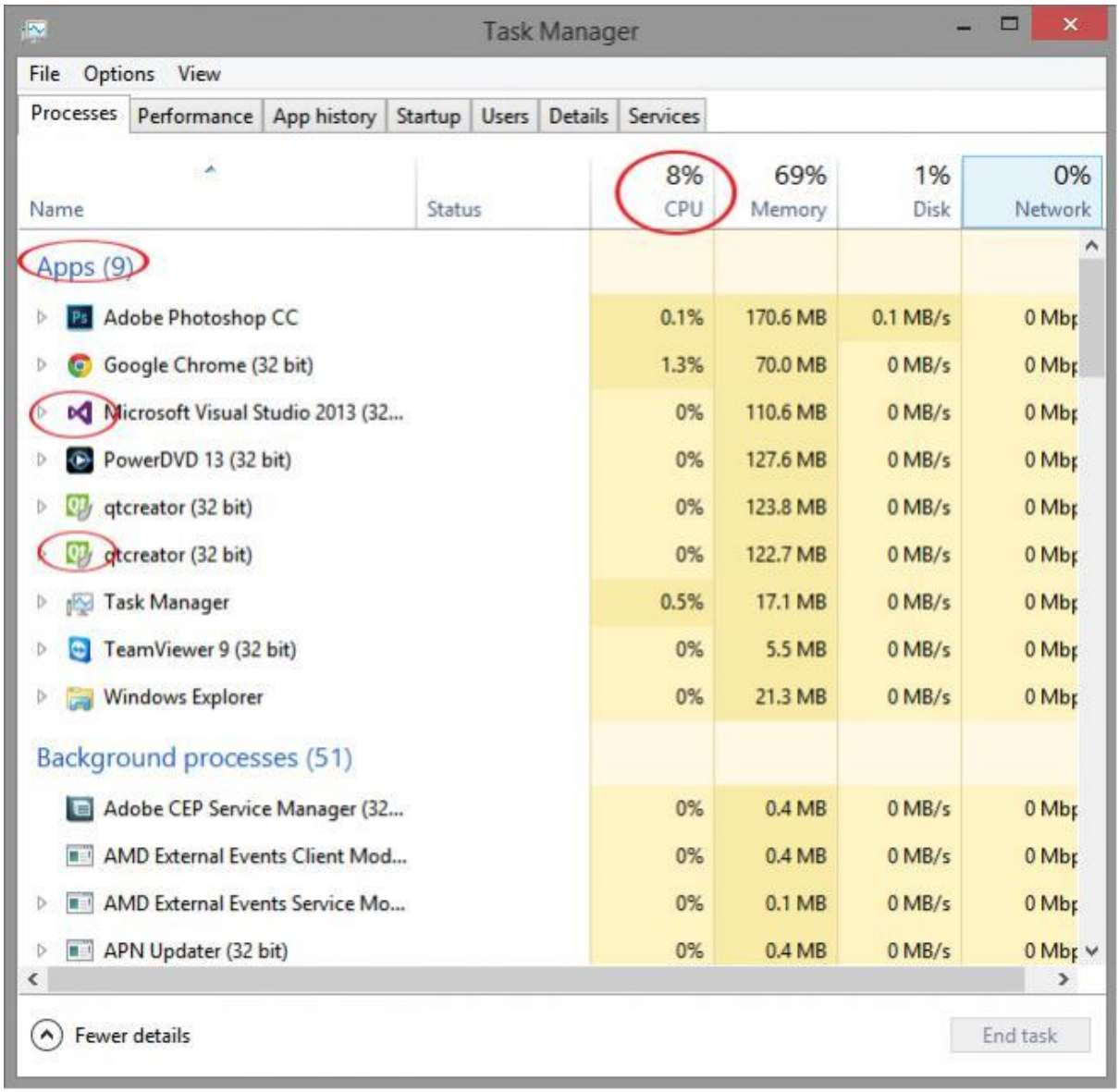
Threadچیست و چه کارهای انجام میده ؟ در کل به چه دردی میخوره و چرا باید ازش استفاده کنیم ؟

توضیح رو خیلی مختصر و خیلی ساده به زبون خودمون میگم تا کسایی که تاحالا این موضوع رو درک نکردن شفافتر و روشنتر درکش کنند.

به طور کلی در نظر بگیرید یک دستگاه های صوتی یا نوار های ویدئویی قدیمی رو که از فناوری Caset استفاده میکنند در این حالت شما وقتی موزیک یا فیلمی رو تماشا میکنید اگه بین فیلم کنونی نیاز داشته باشید تا چند ترک از اون رو به جلو بدین و چند فیلم دیگه رو ببینید مجبور هستید تا اون فیلم رو بیخیال بشید و حالت Seekرو انجام بدین یعنی بکشیدش جلو تا به فیلم مورد نظرتون برسید در این حالت عمل ترتیبی برای رسیدن به هدف صورت گرفته یجور جستجوی ترتیبی برای رسیدن به هدف و اجرای اون هستش حالا ما وقتی پشت کامپیوتر میشینیم و انتظار داریم وارد سیستم عامل که شدیم چندین کار رو هم زمان انجام بدیم برای مثال من مایلم یه طرف برنامه کامپایل کنم یه طرف موزیکمو گوش بدم یه طرفم برنامه های Messenger آنلاین کنم و ... دیگه اینجا هیچ کس حال و حوصله اینو نداره بیاد بشینه دونه دونه کار کنه مثلا اول پیغام بده منتظر بشه تموم شه و بعد بره برنامشو کامپایل کنه و بعد از اون هم تازه بیاد موزیک مورد علاقشو Play کنه و گوش بده ! این ک نشد زندگی عزیز من برای همینم ما بحث Thread رو میاریم وسط که تمام این وظایف رو توسط این مورد حلش میکنیم.

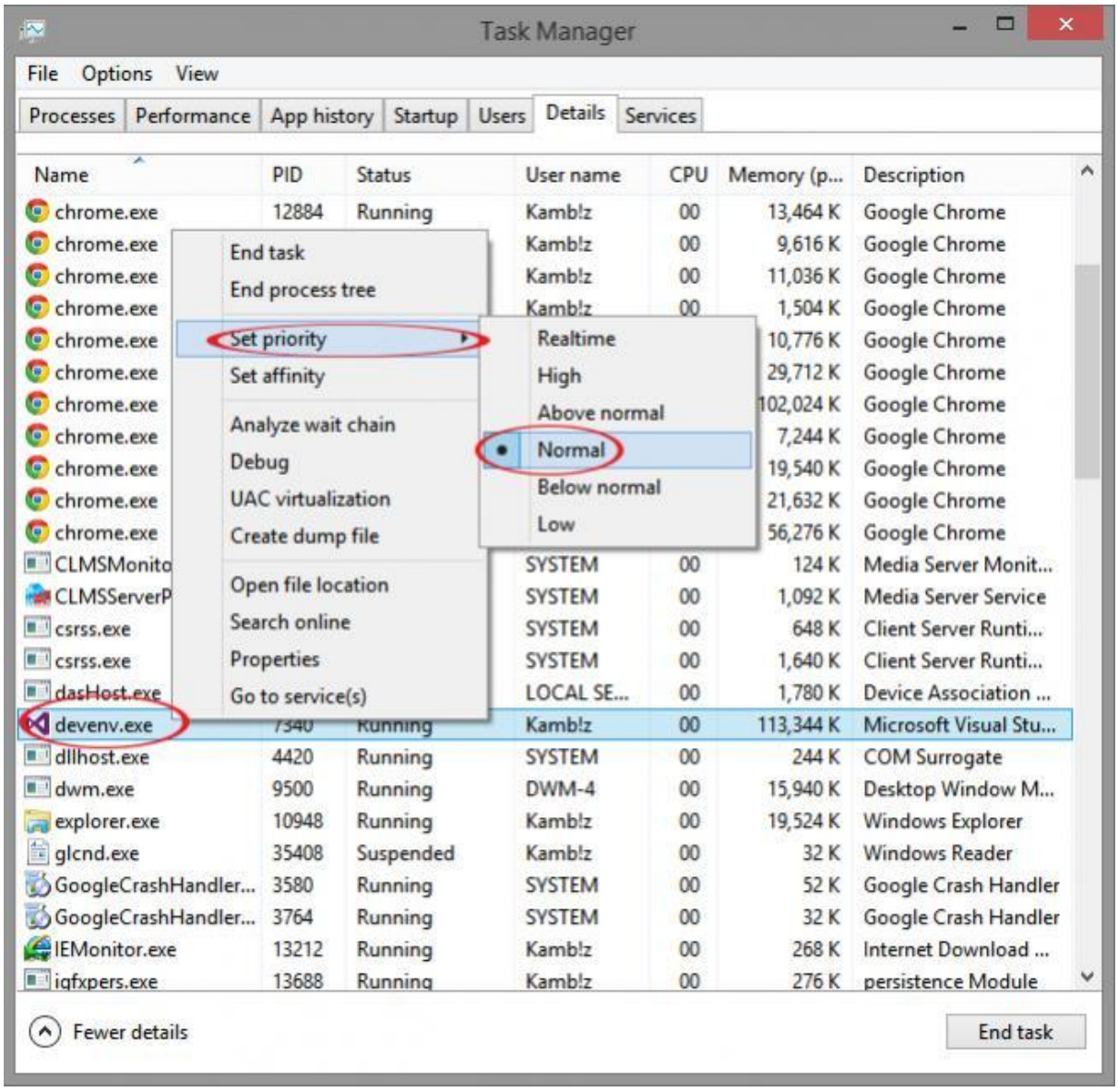
الا برای شروع کار من در مثالی که آموزش خواهم داد دقیقا توضیح میدم که چی به چیه ولی سرعت کامپیوتر ها انقدر زیاده که این پروسه ها در background صورت میگیرند و شما احساس نخواهید کرد که کدوم

Threadکی و چه زمانی اجرا شد ! ولی نتیجه ای که خواهید دید اینه که مواردی که درخواست پردازش فرستادیم به CPUهمشون در حال اجرا هستند چون در کل CPU بر اساس Thread ها برنامه هارو اولویت بندی میکنه برای مثال به پروسه های زیر دقت کنید



طبق تصویر اگه دقت کنید کلی برنامه در حال اجرا هستند که هر کدومشون کلی بندو بساط برای خودشون پهن کردن سر CPU که اگه چنین موارد رو توسط Thread ها مدیریت نکنیم CPU قادر به پردازش هریک از درخواست های متفاوت و مختص همه این برنامه های در حال اجرا به صورت هم زمان رو نخواهد

داشت در حین اجرا پروسه شما برنامتون هنگ خواهد کرد ! بنابراین استفاده از Thread ها در برنامه ها خیلی واجبه در کل یعنی برنامه ای که مینویسید اگه Thread توش بکار برده نشده باشه یک ریال هم ارزش نداره البته این نظر شخصی من هست!

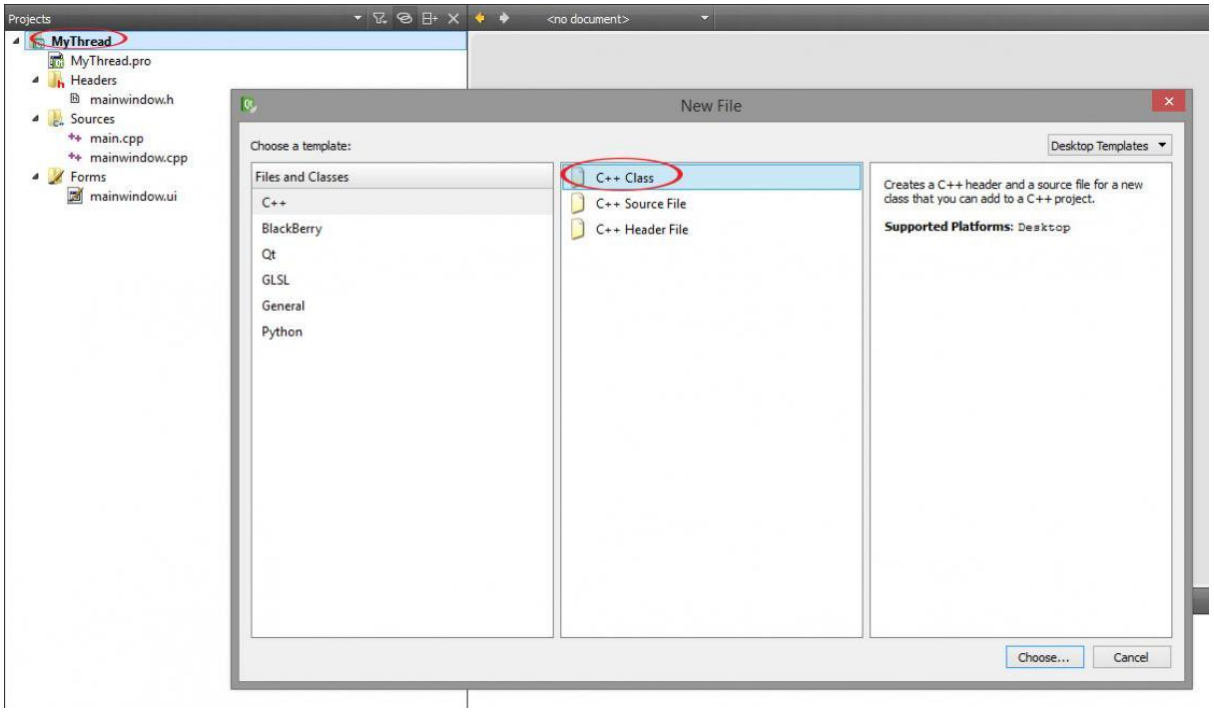


خب حالا یه بحث دیگه برای روشنتر کردن یکی دیگه از فواید Thread ها ... به طور کلی هرکدوم از برنامه های در حال اجرا بر اساس اولویت بندی Thread اون برنامه در اختیار CPU قرار میگیره یعنی اگر اولویت برنامه Visual Studio من بیشتر از برنامه مثلا Qt Creator من باشه و در این صورت اگر همزمان روی

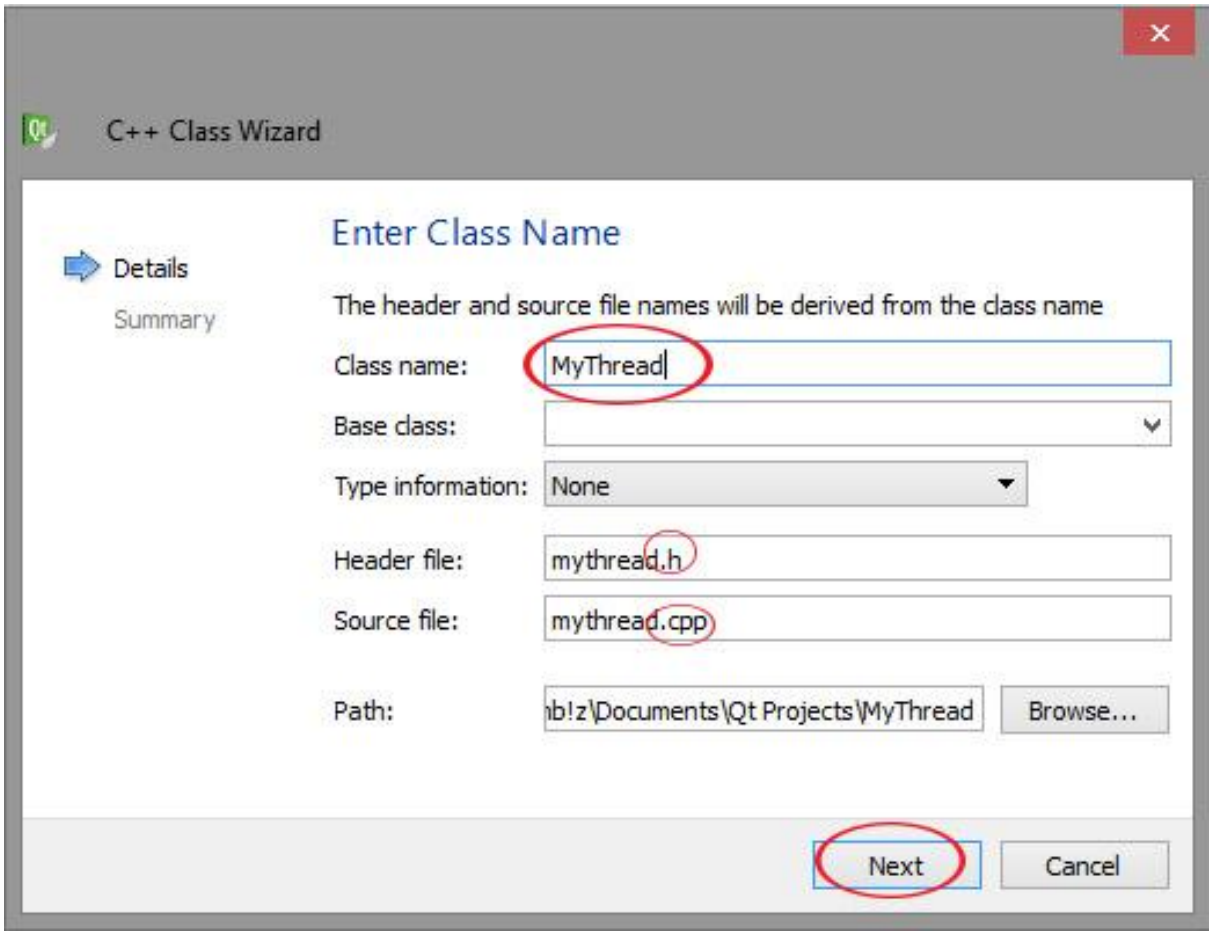
هر دوتای اینها برنامه ای رو کامپایل کنم اونی که اولویت پردازش روش بالاتر هست سریعتر و ابتدا در اختیار پردازش برای CPU ارسال میشود که اینم یکی دیگه از فواید Thread هستش پس ببینید خیلی مهمه در برنامه هایی که پر از شرایط تو در تو و توابع اجرایی دائما در حال اجرا هستن از Thread ها استفاده کنیم.

به طور کلی هر زمانیکه بخواهیم در بیش از یک کار را به طور هم زمان پیش ببریم بهتر است استفاده از **Thread** ها رو فراموش نکنیم.

خب برای شروع این کار بهتر است یک کلاسی ایجاد کنیم ابتدا یک پروژه از نوع QConsole Applicationو بعد به صورت زیر روی پروژه راست کلیک کنید و Add new و بعد قسمت C++ گزینه C++ Class رو انتخاب میکنیم به صورت زیر و ادامه آن....



حالا اطلاعات کلاس رو میدیم به صورت زیر ... نامش رو میزارمMyThread



و ادامه میدیم تا کلاس ما با نام MyThread همراه با فایل های MyThread.h و MyThread.cpp ایجاد بشه.

فایل هیدر من شد به صورت زیر:

```
#ifndef MYTHREAD_H
#define MYTHREAD_H


class MyThread
{
public:
MyThread();
};

#endif // MYTHREAD_H
```

و اینم از فایلcpp.

```
#include "mythread.h"
MyThread::MyThread()
{

}
```

کلاس من اسمش هست MyThread از نوع public ولی من برای شروع کار با کلاس های Threadبه کلاس Thread نیاز دارم باید به صورت زیر فراخوانیش کنم:

```
#include <QThread>
```

و بعد کلاسم رو به طور کلی و عمومی از نوع یک کلاس Thread مشخص کنم به صورت زیر کلاسم رو تغییر میدم:

```
class MyThread : public QThread
{
public:
MyThread();

//Function run
void Run();

};
```

هم نوع کلاس رو از نوع Thread مشخص کردم و همینطور یک تابع برای اجرای Thread نوشتم.

همه این کارها در فایل h. یعنی هیدر کلاس MyThread بود. حالا من بدنه تابع خودم رو در فایل cpp. میسازم به صورت زیر:

```
void MyThread::Run()
{
qDebug() << "My Thread is running !";
}
```

صدا زدن تابع برگرفته شده از کلاس MyThread و ایجاد بدنه تابع با محتویات چاپ این برنامه در حال اجرا می باشد!

البته اگه از qDebug استفاده میکنید بهتره کلاس qDebug رو هم فراخوانی کنید.یا اینکه با std::coutهم میتونید این کار رو انجام بدین.

حالا برای اینکه ببینیم این تابع ساده که کارش عمل چاپ یک متن ساده هست توسط کلاس های Thread کار میکنه یا نه به صورت زیر در فایل main ابتدا فایل mythread.hرو فراخوانی کنید:

```
#include "mythread.h"
```

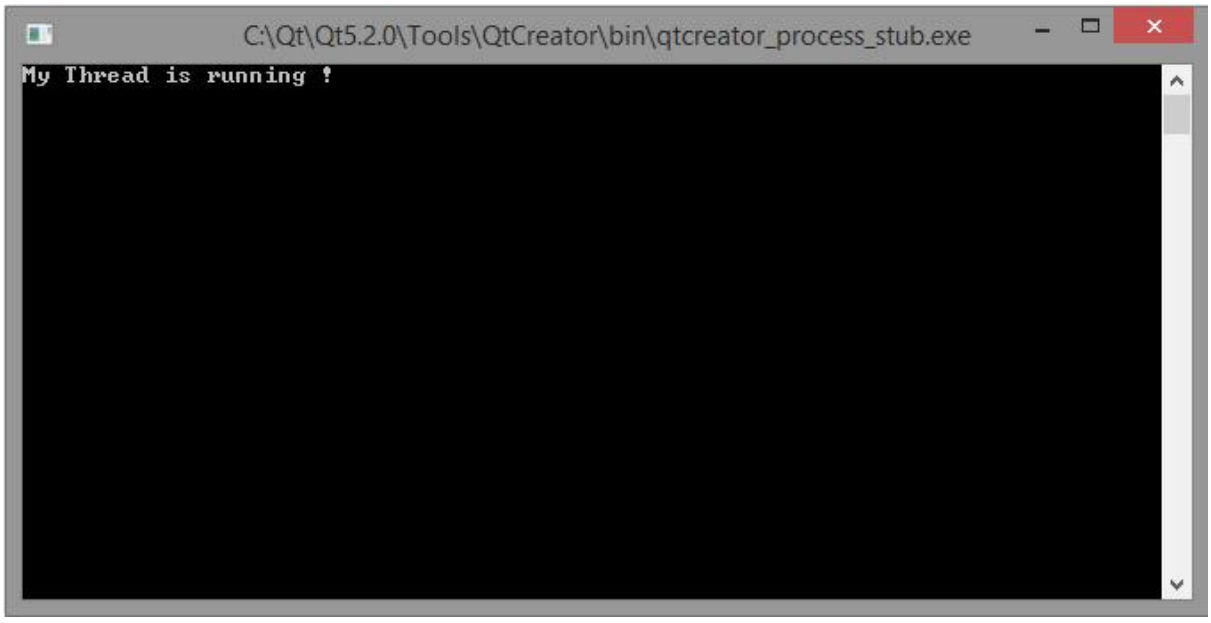
حالا ازش یک کپی میگیرم و تابع Run رو صدا میزنم.

```
MyThread iThread;
iThread.start();
```

کد کلی این بخش :

```
#include <QCoreApplication>
#include "mythread.h"

int main(int argc, char *argv[])
{
QCoreApplication a(argc, argv);

MyThread iThread;
iThread.start();

return a.exec();
}
```

خروجی میشه به صورت زیر:



خب در این مرحله از کار میخوام چند تا پروسه ایجاد کنم که توسط Thread به CPU ارسال بشه چطوری ؟ به صورت زیر ادامه میدم مطلبو...

برای کلاسم در فایل mythread.h یک متغیر از نوع QString تعریف میکنم به صورت زیر:

```
QString Value;
```

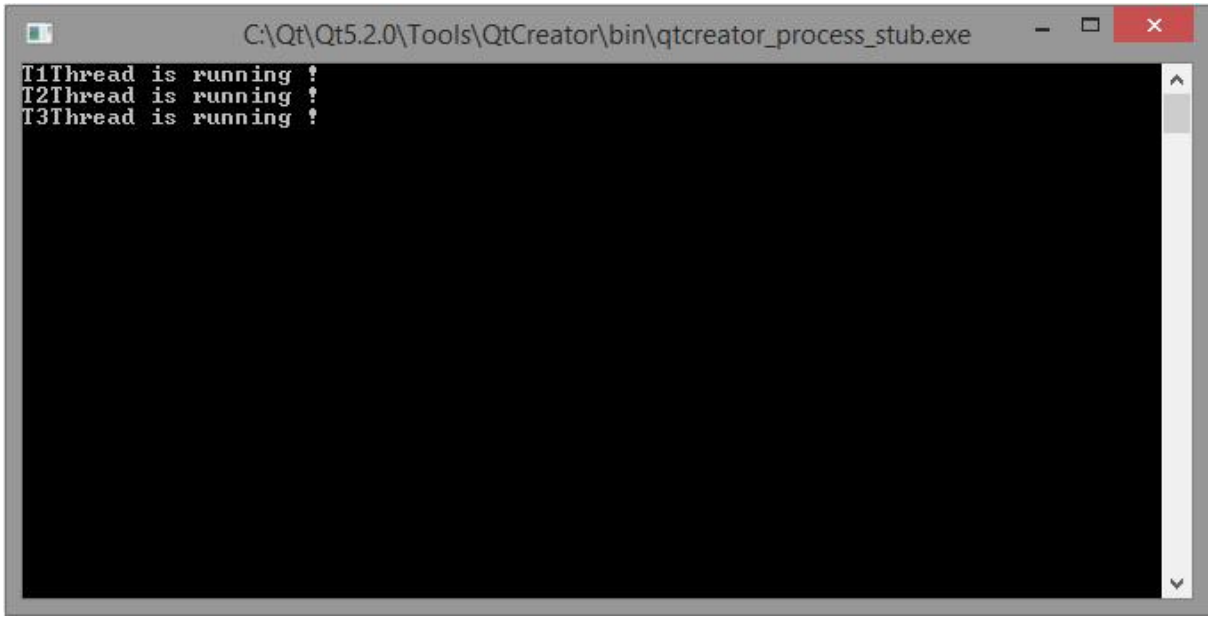
در قسمت بعدی فایل cpp. به صورت زیر دیباگ رو. تغییر میدم:

```
cout << this->Value.toStdString() << "Thread is running !" << endl;
```

و در فایل اصلی به صورت زیر چند بار تابع رو صدا میزنم ولی به کمک Thread این کار رو انجام میدم.

```
MyThread iTH1;
MyThread iTH2;
MyThread iTH3;


iTH1.Value = "T1";
iTH2.Value = "T2";
iTH3.Value = "T3";


iTH1.start();
iTH2.start();
iTH3.start();
```

خب نتیجه میشه به صورت زیر:



خب تقریبا مفهوم Thread ها مشخص شد حالا من میخوام یک تابع رو وارد Thread کنم یه چیز خاص و سنگینی به ذهنم نمیرسه که هنگ کنه ولی فرض کنید اگر من تابع تایمر رو توسط Thread فراخوانی کنم و ازش استفاده کنم چی میشه ؟ فرض کنید هر تابع دیگری که نوشتین و کار اون تابع بارگزاری یا مثلا وصل شدن به یه سرور و آپلود کردن یه چیزی هستش اونوقت ۱۰۰٪ بدون چونو چرا برنامتون هنگام اجرای همون تابع هنگ میکنه برای اینکه از مشکل هنگ کردن خلاص بشیم بازهم چاره کار استفاده از Thread هست ما

با استفاده از این کلاس تابعی که لازمه اجرا بشه رو به داخل Thread ارسال میکنم تا حین اجرا توسط Threadبه CPUبرای پردازش ارسال شود.

به صورت زیر من روی همون پروژه Timer کار میکنم ولی اینبار حرکتش میدم داخل یکThread

اول از همه کلاس Thread رو داخل همون فایل MainWindow.cpp فراخوانی میکنم:

```
#include "QThread"
```

کد زیر در داخل فرم لود در همون پروژه Timer با تغییرات زیر:

```
//Start to get System time.
QTimer *timer = new QTimer(this);
timer->setInterval(1000);
QThread * iTH = new QThread;
timer->moveToThread(iTH);
timer->start();
connect(timer, SIGNAL(timeout()), SLOT(updateClock()));
iTH->start();
```

در خط چهارم یک نمونه از Thread درست کردم:

```
QThread * iTH = new QThread;
```

در خط بعدی توسط کد زیر Timer رو ارسال کردم داخلThread

```
timer->moveToThread(iTH);
```

و در خط آخر Thread رو اجرا کردم با دستور زیر:

```
iTH->start();
```

حالا این برنامه هنگام اجرا وارد **Thread**میشه و مشکل هنگ کردن پیش نخواهد آمد.

به همین سادگی ! در حالی که در دات نت هزارو یک تا کار خود من انجام دادم تا همین تایمر من هنگامی که پینگ میگرفتم به درستی بدون هنگ کردن کار کنه ! ولی اینجا جمعا ۳ خط کد برای خود Thread ننوشتیم.

مرحله بیست و ششم: معرفی و کار باQmap

و اما در رابطه با QMap یک نوع قالب با الگوی خاصی هستش که قابلیت تعریف لیستی متشکل از چندین رشتو کلید های اختصاصی را دارد بر فرض مثال به صورت زیر اگر باشد...

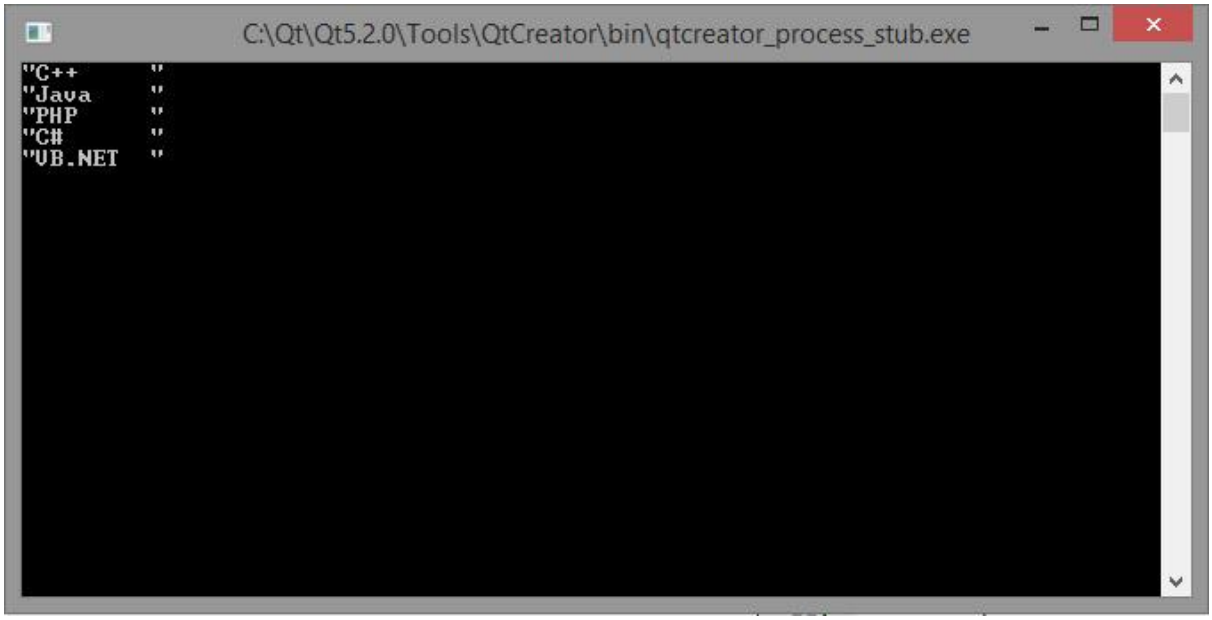
```
QMap<Key, T>
```

برای تعریف کردن این الگو نوع کلاس QMap و بعد کلید و بعد رشته مورد نظر , در کد زیر من عناوینی رو تعریف کردم که هر کدام برابر است با کلید های اختصاصی :

اول از همه کلاس QMap رو فراخوانی میکنم :

```
#include <QMap>



QMap<int, QString> Languages;


Languages.insert(1, "C++ ");
Languages.insert(2, "Java ");
Languages.insert(3, "PHP ");
Languages.insert(4, "C# ");
Languages.insert(5, "VB.NET ");


foreach(int i, Languages.keys())


{
qDebug() << Languages[i];
}
```

حالا در خط اول کلاس QMap رو صدا میزنم و ازش یک کپی میگیرم با شناسه از نوع intو قالب آن از نوع رشته و بعد توسط تابع insert کلید و رشته رو وارد میکنم و در نهایت توسط foreach کلید ها و رشته هارو باهم چاپ میکنم نتیجه میشه به صورت زیر:



خب حالا میریم سراغ فراخوانی کلید ها همراه با رشته های خودشون به صورت زیر کد رو مینویسیم:

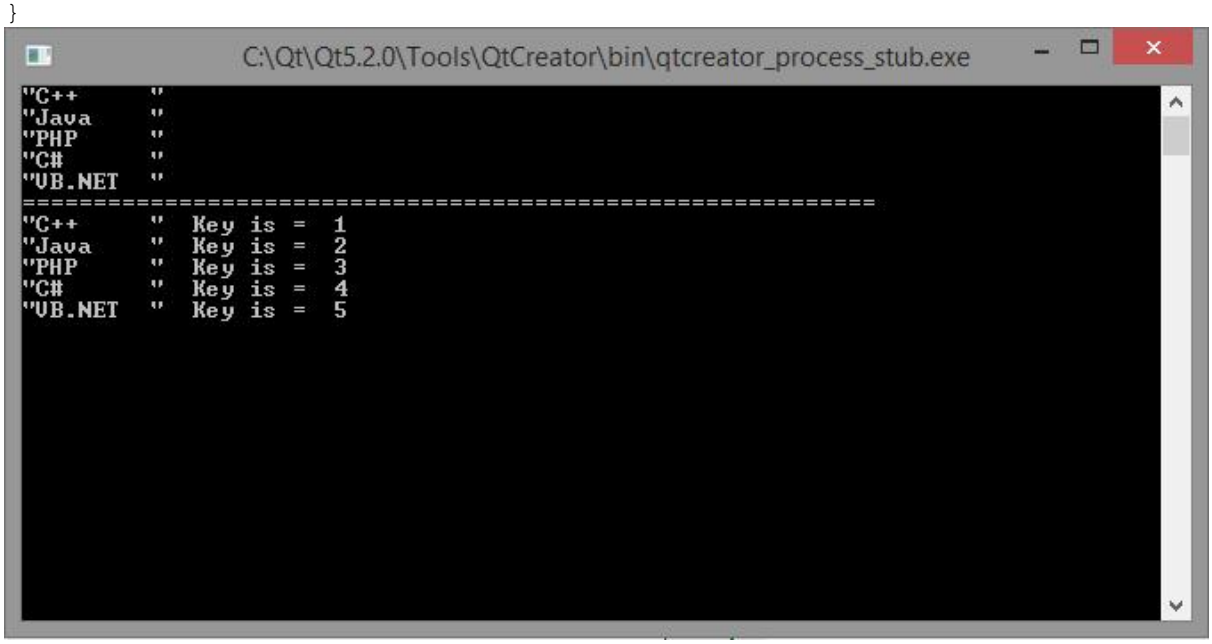
```
QMapIterator<int, QString> ILTH(Languages);
while (ILTH.hasNext())
{
ILTH.next();
qDebug() << ILTH.value() << " Key is = " << ILTH.key();
}
```

اینجا من نیاز به کلاس تکرار کننده یعنی QMapIterator دارم که توسط این مقادیر و کلید های اون رو دریافت و چاپ میکنم و نتیجه به صورت زیر خواهد بود....

کد نهایی :

```
QMap<int, QString> Languages;


Languages.insert(1, "C++ ");
Languages.insert(2, "Java ");
Languages.insert(3, "PHP ");
Languages.insert(4, "C# ");
Languages.insert(5, "VB.NET ");


foreach(int i, Languages.keys())

{
qDebug() << Languages[i];
}

QMapIterator<int, QString> ILTH(Languages);
while (ILTH.hasNext())
{
ILTH.next();
qDebug() << ILTH.value() << " Key is = " << ILTH.key();
```

```
}
```



مرحله بیست و هفتم : معرفی و کار باQhash

تعریف این کلاس دقیقا همانند کلاس QMap هستش با تفاوت های زیر نسبت بهQmap

- قابلیت جستجو و دسترسی بسیار سریعتر را نسبت به QMap دارد.
- در صورتی که لیست تعریف شده حاوی اطلاعات بسیار زیادی باشد در QMap لیست های رشته ای بر اساس کلید های تعریف شده مرتب می شوند ولی در QHash به صورت خودکار تمامی لیست ها بدون قاعده کلیدی مرتب می گردند.
- و تفاوت جزئی در نوع کلید های این دو کلاس که با عملگر های <و == مشخص است.

برای استفاده کلاس QHash رو وارد میکنیم...

```
#include <QHash>
```

بقیه مراحل همانند...QMap

```
QHash<int, QString> Languages;

Languages.insert(1, "C++ ");
Languages.insert(2, "Java ");
Languages.insert(3, "PHP ");
Languages.insert(4, "C# ");
Languages.insert(5, "VB.NET ");

foreach(int i, Languages.keys())
```

```
{
qDebug() << Languages[i];
}

QHashIterator<int, QString> ILTH(Languages);
while (ILTH.hasNext())
{
ILTH.next();
qDebug() << ILTH.value() << " Key is = " << ILTH.key();

}
```

مرحله بیست و هشتم : معرفی و کار باQstringList

این کلاس از رشته ها امکان ایجاد لیستی از رشته هارو در اختیار ما میگذارد برای مثال اگر بخواهیم متغیری تعریف کنیم که حروف الفبا رو برامون به ترتیب نشون بده از این کلاس میتونیم کمک بگیریم.

برای فراخوانی از کلاس لیست رشته ای به صورت زیر عمل میکنیم:

```
#include <QStringList>
```

در کد زیر ابتدا نمونه ای از QStringList میسازم و یک متغیر از نوع رشته که حاوی بخشی از حروف الفبا هستش مشخص میکنم.

```
QStringList MyList;
QString MyLine = "A,B,C,D,E,F,G";
MyList = MyLine.split(",");
foreach(QString Item, MyList)
{
qDebug() << Item;
}
```

در خط بعدی متغیری که از نوع QStringList مشخص کردم رو مقداردهی میکنم و در حین مقدار دهی توسط split محتویات رشته رو جدا گانه در داخل MyList قرار میدم و بعد توسط دستور foreach متغیری رو به عنوان آیتم از نوع رشته در نظر گرفته و چاپ میکنم و نتیجه به صورت زیر خواهد بود...

```
"A"
"B"
"C"
"D"
"E"
"F"
"G"
```

یا مثلا من میخوام مقدار حرف C رو برابر با C++ تغییرش بدم یعنی حرف C رو با C++ جایگزین کنم به راحتی توسط دستور زیر:

```
MyList.replaceInStrings("C", "C++");
```

کد کلی :

```
QStringList MyList;
QString MyLine = "A,B,C,D,E,F,G";
MyList = MyLine.split(",");

MyList.replaceInStrings("C", "C++");

foreach(QString Item, MyList)
{

qDebug() << Item;
}
```

نتیجه :

```
"A"
"B"
"C++"
"D"
"E"
"F"
"G"
```

مرحله بیست و نهم: معرفی و کار با الگوریتم های مرتب سازی

همونطور که میدونید در C++ الگوریتم های مرتب سازی متفاوتی وجود داره که در این میان Qt در کلاس های خودش الگوریتم های مرتب سازی خودش رو ارائه داده... که یکی از ساده ترین اینها تابع **qSort** هستش.

برای مثال من اعداد زیر رو که به صورت زیر اومده میخوام مرتب کنم:

```
4
8
2
1
6
3
```

خب کد رو به صورت زیر مینویسیم...

```
QList<int> MyList;

MyList << 4 << 8 << 2 << 1 << 6 << 3;
qSort(MyList);

foreach(int i, MyList) {

qDebug() << i;
}
```

در خط اول متغیر رو از نوع لیست در نظر گرفتم و در خط بعدی اون مقادیرم رو وارد لیستم کردم و در خط بعدی اون توسط تابع qSort مقادیر رو مرتب میکنم و بعد توسط foreach و qDebug چاپش میکنم.

نتیجه شد به صورت زیر:

```
1
2
3
4
6
8
```

مرحله سی ام : معرفی و کار با الگوریتم های جستجو کننده

یکی از مواردی که به درد میخوره الگوریتم جستجو هست که در اینجا توسط تابع **qFind** میتونیم به راجتی مقدار مورد نظر رو در صورت وجود در لیست داده ها پیدا کنیم.

به صورت زیر کد رو مینویسیم:

```
QList<int> MyList;
MyList << 12 << 0 << 133 << 64 << 512 << 128;
QList<int>::const_iterator MyIterator = qFind(MyList.begin(), MyList.end(),
64);


if (MyIterator != MyList.end())
{
    qDebug() << "Found = " << *MyIterator;
}

else

{
    qDebug() << "Not found!";

}
```

ابتدا متغیرم رو از نوع یک نوع لیست عددی در نظر میگیرم و در خط بعد مقادیر مورد نظرم رو وارد لیست میکنم.

حالا در خط بعدی عمل جستجو یعنی توسط یک لیست جدیدی از اعداد ایجاد کرده و توسط تابع **qFind** مقدار ۶۴ رو از نقطه آغاز مقادیر موجود در لیست تا انتهای اون رو بررسی میکنم.

در خط بعدی یک دستور شرطی برعکس آوردم که در اینجا مشخص کردم اگر مقدار مورد نظر من در لیست موجود بود پیغام Found رو همراه با مقدار یافت شده چاپ کن در غیر اینصورت پیغام Not found رو چاپ خواهد کرد و در نهایت نتیجه به صورت زیر خواهد بود...

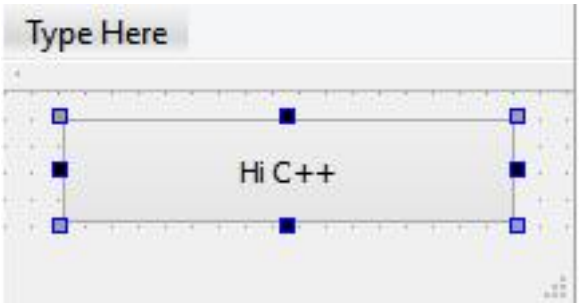
```
Found = 64
```

مرحله سی یکم : معرفی و کارToolTip

ToolTipها چی هستن و به چه دردی میخورن ؟ در کل بیشترین کاربرد این مورد در هنگام Mouse Hoverبر روی اشیاء موجود روی رابط کاربری و فرم های شماست که برای استفاده از این امکانات به روش های زیر عمل خواهیم کرد...

در این بخش من میخوام یک استایل زیبا همراه با متن مورد نظر برای ToolTip کنترلم روی فرم اختصاص بدم توسط دستور setStyleSheet میتونم استالم رو بر اساس کد های CSS به شیئ مورد نظرم اختصاص بدم.

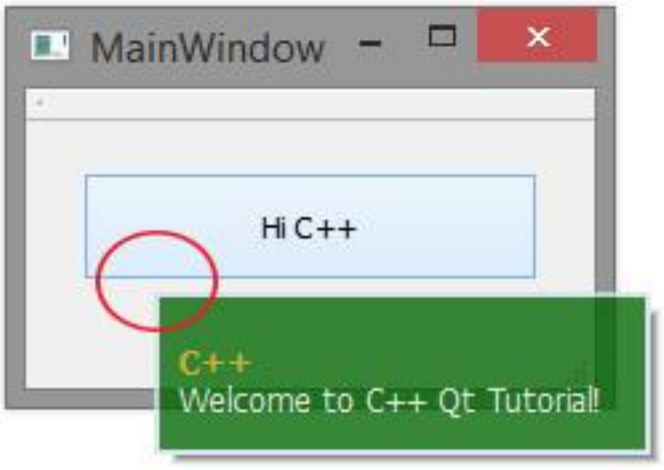
و توسط پراپرتی setToolTip میتونم محتویان ToolTip رو به کنترلم اختصاص بدم به صورت زیر...



داخل فرم لود کد زیر رو مینویسم البته اسم کنترلم رو میزارم.MyButton

```
ui->MyButton->setStyleSheet(" QToolTip{text-align:left;color:
#ffffff;background-color: #006700;font-family: Tahoma ,'PT Sans',
'Helvetica', arial, serif; border: 2px solid #d5def2;padding: 2px;opacity:
200;}");
ui->MyButton->setToolTip("<br><font
color='#C90'><b>C++</b></font><br>Welcome to C++ Qt Tutorial!");
```

نتیجش وقتی ماوس رو روی کنترل میبریم میشه طرح زیر:



اصلا نمیخوام آموزش CSS بدم چون واقعا حالشو ندارم ولی باید به یه نکته مهمی اشاره کنم که وقتی استایلی رو روی کنترل یا فرمتون ست میکنید حتما تخصیصش بدین به هدفی که دارید ! مثلا من در استایل این بخش گفتم QToolTip{} و هر کدی که داخلش بنویسم مختص ToolTip خواهد بود در غیر اینصورت اگر بدون مشخص کردن آبجکت های مورد نظرتون استایلی رو اختصاص بدین در آن صورت همه قسمت های کنترل از استایل شما ارث خواهند برد.

یه توضیحی بدم در رابطه با طرح و استایل این مورد که تمامی اینها مربوطه به کدهای CSS اگه اشخاصی که در طراحی وب و کار با CSS مهارت و تجربه کافی دارن میدونن که چه کارهایی میشه کرد با این!!! CSS

مرحله سی دوم : معرفی و کار با شبکه / استفاده از پروتکل های HTTP وFTP

در اینترنت همانند سایر شبکه های کامپیوتری از پروتکل های متعدد و با اهداف مختلف استفاده می گردد. هر پروتکل از یک ساختار خاص برای ارسال و دریافت اطلاعات (بسته های اطلاعاتی) استفاده نموده و ترافیک مختص به خود را در شبکه ایجاد می نماید (**HTTP** . برگرفته از **Hyper Text Transfer Protocol**)، یکی از متداولترین پروتکل های لایه application است که مسئولیت ارتباط بین سرویس گیرندگان و سرویس دهندگان وب را برعهده دارد.

به طور کلی در این قسمت میرسیم به موارد شبکه و کار با پروتکل ها و...

در **C++** کتابخانه های **Qt** کلاس هایی رو برای شبکه دارند که برای فراخوانی کلاس ها به برنامه ابتدا لازمه در فایل **pro.** موجود در پروژه کد زیر رو اضافه کنیم:

```
QT += network
```

البته میتونید در ادامه همون **QT +=** موجود به این صورت عمل کنید:

```
QT += core network
```

البته دقت کنید باید فاصله بین رفرنس ها وجود داشته باشد.

یک توضیحی بدم که چرا از نماد های **+=** استفاده میکنیم دلیلش اینه و مشخصه که دستور افزودن رو دادیم به **QT** تا رفرنس مورد نظر رو وارد پروژه میکند ولی اگه بخوام رفرنس مورد نظر رو حذف یا از پروژه بیرون کنم چطور ؟ این موقت باید بگم **-=** به مثال زیر توجه کنید:

```
QT += core network
QT -= gui
```

در خط دوم مشخص کردم که رفرنس های مربوط به **gui** که شامل **widget** ها و یا هر رابط کاربری به جز **Console** هستش رو از پروژه حذف کن یعنی نیازی به رفرنس **gui** نداریم.

خب پروژم رو ایجاد میکنم و تحت توضیحات در پست قبلی رفرنس **network** رو به پروژم اضافه میکنم.

حالا روی پروژم راست کلیک میکنم و **Add New** و قسمت **File and Classes** گزینه **C++ Class**رو انتخاب میکنم و اطلاعاتش رو به صورت زیر مشخص میکنم.

ابتدا من نیاز دارم یک کلاس برای پروتکل HTTP ایجاد کنم به صورت زیر ایجاد میکنم و فایل هیدر و اصلی رو آماده میکنم.

```
#ifndef MYHTTP_H
#define MYHTTP_H


#include <QObject>


#include <QtGlobal>
#include <QObject>
#include <QNetworkAccessManager>
#include <QNetworkRequest>
#include <QNetworkReply>
#include <QFile>
#include <QTimer>




class MyHttp : public QObject
{
Q_OBJECT


public:
explicit MyHttp(QObject *parent = 0);
virtual ~MyHttp();


signals:
void addLine(QString qsLine);
void downloadComplete();
void progress(int nPercentage);


public slots:
void download(QUrl url);
void pause();
void resume();
```

```
private slots:
void download();
void finishedHead();
void finished();
void downloadProgress(qint64 bytesReceived, qint64 bytesTotal);
void error(QNetworkReply::NetworkError code);
void timeout();


private:
QUrl _URL;
QString _qsFileName;
QNetworkAccessManager* _pManager;
QNetworkRequest _CurrentRequest;
QNetworkReply* _pCurrentReply;
QFile* _pFile;
int _nDownloadTotal;
bool _bAcceptRanges;
int _nDownloadSize;
int _nDownloadSizeAtPause;
QTimer _Timer;
};


#endif // MYHTTP_H
```

خب در این کلاس من برای HTTP کلاس و توابع مورد نیاز رو تحت کلاس های QNetworkAccessManager / QNetworkRequest / QNetworkReplyایجاد کردم.

که وظیفه هرکدام به صورت زیر است :

کلاس : **QNetworkAccessManager** وظیفه این کلاس این هستش که برای ما قابلیت ارسال اطلاعات و دریافت اطلاعات رو از سرور فراهم میکنه

کلاس : **QNetworkRequest** وظیفه این کلاس که دریافت و نگه داری اطلاعاتی هستش که کلاس **QNetworkAccessManager** به آنها نیاز خواهد داشت.

کلاس : **QNetworkReply** وظیفه این کلاس هم نگه داری داده ها و هیدر هایی هستش که درخواست شده.

کلاس : **QtGlobal** بعضی وقتها لازمه این کلاس رو فراخوانی کنیم زیرا هیدر ها و توابع اساسی Qt توسط اینکلود کردن این کلاس قابل شناسایی هستش که البته روش های دیگری هم هست تا به طور کلی از هر بار اینکلود کردن چنین موارد در قسمت های مختلف پروژه جلوگیری کنیم که بعد ها توضیحاتی خواهیم داد.

کلاس : **QObject** مشخصه که برای ایجاد و شناسایی آبجکت ها استفاده میشود.

کلاس : **QFile** برای کار با فایلها مورد نیاز هستش به عنوان مثال اگه قرار هست فایلی رو دانلود کنیم و عمل مثلا کپی رو روی فایل دانلود شده انجام دهیم به این کلاس نیاز اساسی خواهیم داشت.

کلاس : **QTimer** برا کار با زمان مورد استفاده قرار میگیره که در این مثال ازش برای مشخص سازی زمان دریافت و ارسال استفاده خواهیم کرد.

خب ابتدا من کلاسم رو از نوع عمومی و آبجکت مشخص کردم به صورت زیر:

```
class MyHttp : public QObject
```

برای اینکه تمامی هیدر فایل های آبجکت ما توسط زبان C++ قابل شناسایی باشه لازمه کار اینه از ماکرو مخصوص آبجکت در Qt استفاده کنیم که به صورت زیر در کلاس ازش استفاده خواهد شد.

```
Q_OBJECT
```

در خط بعدی تعریف کلی کلاس مشخص شده و بعد سیگنال / اسلات ها و متغیر های مورد نیاز اعمال گردیده است.

تا اینجا کار ما با فایل MyHttp.h تمونه میریم در پست بعدی کار با فایلMyHttp.cpp

خب فایل cpp ما حاوی کد های زیر خواهد بود که در این فایل ما بدنه توابع خودمون رو کامل کردیم و برای دانلود فایلمون همراه با اطلاعات دقیق ارسال و دریافت لازم داریم ! اشاره کنم که خیلی ساده تر از اینها میشه کد دانلود فایل رو توسط http نوشت ولی نیاز دونستم کمی پروژه پیچیده تر رو انتخاب کنم تا نتیجه حاصل از فایل دانلود شده همراه با پکیج های ارسالی و دریافتی همانند سیستم ارسال و دریافت کننده TeamViewerمشخص بشه که ممکنه خیلی ها به این نیاز داشته باشند.

```
#include "myhttp.h"
#include <QFileInfo>
#include <QDateTime>
#include <QDebug>
```

```
MyHttp::MyHttp(QObject *parent) :
QObject(parent)
, _pManager(NULL)
, _pCurrentReply(NULL)
, _pFile(NULL)
, _nDownloadTotal(0)
, _bAcceptRanges(false)
, _nDownloadSize(0)
, _nDownloadSizeAtPause(0)
{
}
```

```
MyHttp::~MyHttp()
{
if (_pCurrentReply != NULL)
{
pause();
}
}
```

```
void MyHttp::download(QUrl url)
{
qDebug() << "download: URL=" << url.toString();


_URL = url;
{
QFileInfo fileInfo(url.toString());
_qsFileName = fileInfo.fileName();
}
_nDownloadSize = 0;
_nDownloadSizeAtPause = 0;


_pManager = new QNetworkAccessManager(this);
_CurrentRequest = QNetworkRequest(url);


_pCurrentReply = _pManager->head(_CurrentRequest);


_Timer.setInterval(5000);
_Timer.setSingleShot(true);
connect(&_Timer, SIGNAL(timeout()), this, SLOT(timeout()));
_Timer.start();


connect(_pCurrentReply, SIGNAL(finished()), this, SLOT(finishedHead()));
connect(_pCurrentReply, SIGNAL(error(QNetworkReply::NetworkError)), this,
SLOT(error(QNetworkReply::NetworkError)));
}
```

```
void MyHttp::pause()
{
qDebug() << "pause() = " << _nDownloadSize;
if (_pCurrentReply == NULL)
{
return;
}
_Timer.stop();
disconnect(&_Timer, SIGNAL(timeout()), this, SLOT(timeout()));
disconnect(_pCurrentReply, SIGNAL(finished()), this, SLOT(finished()));
disconnect(_pCurrentReply, SIGNAL(downloadProgress(qint64, qint64)), this,
SLOT(downloadProgress(qint64, qint64)));
disconnect(_pCurrentReply, SIGNAL(error(QNetworkReply::NetworkError)),
this, SLOT(error(QNetworkReply::NetworkError)));


_pCurrentReply->abort();
// _pFile->write( _pCurrentReply->readAll());
_pFile->flush();
_pCurrentReply = 0;
_nDownloadSizeAtPause = _nDownloadSize;
_nDownloadSize = 0;
}




void MyHttp::resume()
{
qDebug() << "resume() = " << _nDownloadSizeAtPause;


download();
}




void MyHttp::download()
{
qDebug() << "download()";


if (_bAcceptRanges)
{
QByteArray rangeHeaderValue = "bytes=" +
QByteArray::number(_nDownloadSizeAtPause) + "-";
if (_nDownloadTotal > 0)
{
rangeHeaderValue += QByteArray::number(_nDownloadTotal);
}
_CurrentRequest.setRawHeader("Range", rangeHeaderValue);
}


_pCurrentReply = _pManager->get(_CurrentRequest);


_Timer.setInterval(5000);
_Timer.setSingleShot(true);
connect(&_Timer, SIGNAL(timeout()), this, SLOT(timeout()));
```

```
_Timer.start();


connect(_pCurrentReply, SIGNAL(finished()), this, SLOT(finished()));
connect(_pCurrentReply, SIGNAL(downloadProgress(qint64, qint64)), this,
SLOT(downloadProgress(qint64, qint64)));
connect(_pCurrentReply, SIGNAL(error(QNetworkReply::NetworkError)), this,
SLOT(error(QNetworkReply::NetworkError)));
}




void MyHttp::finishedHead()
{
_Timer.stop();
_bAcceptRanges = false;


QList<QByteArray> list = _pCurrentReply->rawHeaderList();
foreach(QByteArray header, list)
{
QString qsLine = QString(header) + " = " + _pCurrentReply-
>rawHeader(header);
addLine(qsLine);
}


if (_pCurrentReply->hasRawHeader("Accept-Ranges"))
{
QString qstrAcceptRanges = _pCurrentReply->rawHeader("Accept-Ranges");
_bAcceptRanges = (qstrAcceptRanges.compare("bytes", Qt::CaseInsensitive) ==
0);
qDebug() << "Accept-Ranges = " << qstrAcceptRanges << _bAcceptRanges;
}


_nDownloadTotal = _pCurrentReply-
>header(QNetworkRequest::ContentLengthHeader).toInt();


// _CurrentRequest = QNetworkRequest(url);
_CurrentRequest.setRawHeader("Connection", "Keep-Alive");
_CurrentRequest.setAttribute(QNetworkRequest::HttpPipeliningAllowedAttribut
e, true);
_pFile = new QFile(_qsFileName + ".part");
if (!_bAcceptRanges)
{
_pFile->remove();
}
_pFile->open(QIODevice::ReadWrite | QIODevice::Append);


_nDownloadSizeAtPause = _pFile->size();
download();
}




void MyHttp::finished()
```

```
{
qDebug() << __FUNCTION__;


_Timer.stop();
_pFile->close();
QFile::remove(_qsFileName);
_pFile->rename(_qsFileName + ".part", _qsFileName);
_pFile = NULL;
_pCurrentReply = 0;
emit downloadComplete();
}



void MyHttp::downloadProgress(qint64 bytesReceived, qint64 bytesTotal)
{
_Timer.stop();
_nDownloadSize = _nDownloadSizeAtPause + bytesReceived;
qDebug() << "Download Progress: Received=" << _nDownloadSize << ": Total="
<< _nDownloadSizeAtPause + bytesTotal;


_pFile->write(_pCurrentReply->readAll());
int nPercentage =
static_cast<int>((static_cast<float>(_nDownloadSizeAtPause + bytesReceived)
* 100.0) / static_cast<float>(_nDownloadSizeAtPause + bytesTotal));
qDebug() << nPercentage;
emit progress(nPercentage);


_Timer.start(5000);
}



void MyHttp::error(QNetworkReply::NetworkError code)
{
qDebug() << __FUNCTION__ << "(" << code << ")";
}



void MyHttp::timeout()
{
qDebug() << __FUNCTION__;
}
```

در این کلاس تمامی اتفاقانی مثل ارسال / دریافت اطلاعات و تمامی رخداد های موجود در طی دانلود یک فایل رو ایجاد کردیم که هر کدام وظیفه خاص خودش رو خواهد داشت.

خب حالا بیاییم سراغ تجزیه تحلیل این کلاس ببینیم توش چه خبره...

ابتدا موارد زیر رو اینکلود کردیم:

```
#include "myhttp.h"
#include <QFileInfo>
#include <QDateTime>
#include <QDebug>
```

دلیلشون واظحه ! اولیرو حتما باید اینکلود کنیم چون بدون اون دسترسی به کلاس و توابع از پیش تعریف شده در فایل h. امکانپذیر نخواهد بود.

QFileInfoبرای کار با اطلاعات فایل هستش و گزینه QDateTime هم خب برای کار با تاریخو زمانو اینجور چیزا و در نهایت QDebug برای کار QDebug که من خودم شخصا ازش متنفرم چون خیلی ضعیفتر از std::cout عمل میکنه ولی خب طی این آموزش ها از همین روش استفاده میکنیم هرچند هیچ موردی نخواهد داشت اگه شما در پروژه هاتون به جای QDebug از iostream و فراخوانی std::coutبرای چاپ موارد مورد نظرتون استفاده کنید فقط در بعضی از موارد باید عمل تبدیل QString به StdString ها رو رعایت کنید و در رابطه با مقادیر دیگه هم همینطوره...

خب در خط بعدی به صورت زیر...

```
MyHttp::MyHttp(QObject *parent) :
QObject(parent)
, _pManager(NULL)
, _pCurrentReply(NULL)
, _pFile(NULL)
, _nDownloadTotal(0)
, _bAcceptRanges(false)
, _nDownloadSize(0)
, _nDownloadSizeAtPause(0)
{
}
```

اینجا ما کلاسمون رو فراخوانی کردیم همراه با آبجکت هاش و هر یک از کلاس های مشتق شده آن که به نوعی کپی از کلاس ها برای استفاده از آن گرفته شده اند در اینجا آمده است که ابتدا برای هر یک مقادیر پیشفرض بر اساس نوع آنها مشخص شده است.

در خط بعدی به صورت زیر...

```
MyHttp::~MyHttp()
{
if (_pCurrentReply != NULL)
{
pause();
}
}
```

در بدنه کلاس ویرانگر دستور شرطی آمده که اگر پاسخی ارسال شده از طرف سرور به صورت NULL نباشد تابع pause اجرا خواهد گردید.

حالا در مرحله بعدی به صورت زیر....

```
void MyHttp::download(QUrl url)
{
qDebug() << "download: URL=" << url.toString();



_URL = url;
{
QFileInfo fileInfo(url.toString());
_qsFileName = fileInfo.fileName();
}
_nDownloadSize = 0;
_nDownloadSizeAtPause = 0;



_pManager = new QNetworkAccessManager(this);
_CurrentRequest = QNetworkRequest(url);



_pCurrentReply = _pManager->head(_CurrentRequest);



_Timer.setInterval(5000);
_Timer.setSingleShot(true);
connect(&_Timer, SIGNAL(timeout()), this, SLOT(timeout()));
_Timer.start();



connect(_pCurrentReply, SIGNAL(finished()), this, SLOT(finishedHead()));
connect(_pCurrentReply, SIGNAL(error(QNetworkReply::NetworkError)), this,
SLOT(error(QNetworkReply::NetworkError)));
}
```

تابع downloadرو فراخوانی کردیم که حاوی پارامتر Url از نوع QUrl هستش که در حین اجرا ابتدا آدرسی که مختص فایل قابل دانلود هستش رو میگیره و به صورت زیر...

```
qDebug() << "download: URL=" << url.toString();
```

یک کد چاپ برای آدرسی که ارسال کردیم برای دانلود.

```
_URL = url;
{
QFileInfo fileInfo(url.toString());
_qsFileName = fileInfo.fileName();
}
```

اینجا آدرس ارسال شده توسط کاربر دریافت و مشخصات آدرس توسط کلاس QFileInfo بررسی و نام فایل از آدرس گرفته شده و به متغیر از قبل تعریف شده یعنی qsFileName_ارسال میشود.

در خط بعدی کد های زیر...

```
_nDownloadSize = 0;
_nDownloadSizeAtPause = 0;
```

مقدار ۰ رو به متغیر های nDownloadSize_ و همچنین nDownloadSizeAtPause_ اختصاص میدیم.

حالا اینجا کلاس اصلی ما که ازش استفاده خواهیم کرد وار کار خواهد شد به صورت زیر...

```
_pManager = new QNetworkAccessManager(this);
_CurrentRequest = QNetworkRequest(url);
```

اینجا مقدار کلاس مشتق شده pManager_ رو برابر کلاس اصلی QNetworkAccessManager مشخص میکنیم و همچنین کلاس های CurrentRequest_ رو با درخواست کننده آدرس یعنی QNetworkRequestبا پارامتر url اختصاص میدیم.

در این خط به صورت زیر...

```
_pCurrentReply = _pManager->head(_CurrentRequest);
```

مقدار pCurrentReply_ رو برابر میکنیم با هیدر (Header) ای که توسط کلاس CurrentRequest_گرفته شده.

خط های زیر...

```
_Timer.setInterval(5000);
_Timer.setSingleShot(true);
```

خب ایجا ما نیاز داریم به مدت زمانی که باید توسط تایمرمون ایجاد کنیم برای مثال در اینجا میانگین زمانی رو برابر **5000 میلی ثانیه** قرار دادیم و در خط بعدیش مقدار SingleShot یا همان تایمر شات که یکی از پراپرتی های QTimer هستش رو برابر با true قرار دادیم.

و در خط های زیر...

```
connect(&_Timer, SIGNAL(timeout()), this, SLOT(timeout()));
_Timer.start();
```

توسط تابع connect سیگنال ارسالی توسط تایمر رو توسط تابع timeout ارسال و مجددا سیگنال بازگشتی یا همان اسلات رو توسط تابع timeout ارسال میکنیم.

```
connect(_pCurrentReply, SIGNAL(finished()), this, SLOT(finishedHead()));
connect(_pCurrentReply, SIGNAL(error(QNetworkReply::NetworkError)), this,
SLOT(error(QNetworkReply::NetworkError)));
```

این دو کد نهایی عمل اتصال بین سیگنال و اسلات برای ارسال و دریافت جواب به عنوان اتمام کار انجام خواهند داد و در خط بعدی همین کار رو برای زمانی که ارتباط با مشکل برخورده باشه ایجاد میکنیم.

و اما تابع pause به صورت زیر تعریف شده است:

```
void MyHttp::pause()
{
    qDebug() << "pause() = " << _nDownloadSize;
    if (_pCurrentReply == NULL)
    {
        return;
    }
    _Timer.stop();
    disconnect(&_Timer, SIGNAL(timeout()), this, SLOT(timeout()));
    disconnect(_pCurrentReply, SIGNAL(finished()), this, SLOT(finished()));
    disconnect(_pCurrentReply, SIGNAL(downloadProgress(qint64, qint64)),
this, SLOT(downloadProgress(qint64, qint64)));
    disconnect(_pCurrentReply, SIGNAL(error(QNetworkReply::NetworkError)),
this, SLOT(error(QNetworkReply::NetworkError)));


    _pCurrentReply->abort();
    // _pFile->write( _pCurrentReply->readAll());
    _pFile->flush();
    _pCurrentReply = 0;
    _nDownloadSizeAtPause = _nDownloadSize;
    _nDownloadSize = 0;
}
```

موارد ابتداییش مشخص هست میرم سراغ Timert.Stop به این خط که میرسیم از نام تابع هم مشخص وظیفش متوقف کردن عملیات هست که در اینجا توسط تابع **disconnect** تمامی ارتباطات موجود بین سیگنال ها و اسلات ها ازبین خواهند رفت یعنی در این صورت اگه فایلی در حال دانلود باشه یا مثلا وظیفه دانلود به اتمام رسیده باشه توسط این تابع هست که میشه عمل توقف رو انجام داد.

نمیخوام ریز به ریز توضیحات خط به خطی بدم چون واقعا هر کدوم از این کلاس ها چندین سطر توضیحات اختصاصی داره برا همین سعی میکنیم توضیح مختصری در نحوه کار این ها بدم که شاید مفید باشه

در این تابع در کل با استفاده از تابع **disconnect** تمامی ارتباط ها از بین رفته و بعد از آن دستور از بین بردن تمامی درخواست ها توسط تابع abort ارسال می شود.

در خط بعد این ها یک گزینه ای داریم به نام **Flush** به صورت زیر ازش استفاده شده:

```
_pFile->flush();
```

میدونید این چیکار میکنه !؟ اگر در طول دریافت فایل اطلاعات به درستی و کامل بدون هیچگونه خرابی روی فیال به **Buffer** انتقال پیدا کنند در این حالت **Flush** مقدار **true** رو ارسال میکنه و در غیر اینصورت مقدار **false** رو ارسال خواهد کرد در واقع توسط این میتونیم راحت مشخص کنیم که فایلمون با موفقیت و بدون هیچگونه خرابی دانلود شد یا خیر!

خط های بعدی هم که مشخص هست

خب و اما تابع **resume** در باره این تابع هم اینطور توضیح بدم که وظیفش ادامه عملیات دانلود هست درواقع با اجرای این تابع مسلما باید تابع دیگری به نام **download** عملیات خودش رو انجام بده به صوت زیر...

```
void MyHttp::resume()
{
    qDebug() << "resume() = " << _nDownloadSizeAtPause;
    download();
}
```

خیلی راحت تابع فراخوانی شده و در بدنه تابع توسط **qDebug** متنیرو هنگام درخواست ادامه با عنوان مقدار سایز از متغیری که در تابع **pause** مقدار دهی شده بود دریافت میکنه.

و در نهایت ادامه کار توسط اجرا شدن تابع **download** صورت خواهد گرفت.

خب اینجا دوباره ما تابعی نوشتیم با نام download ولی با این تفاوت که دیگه اینجا آدرس فایلی برای دانلود نمیخواد این تابع کی استفاده میشه اگه گفتین !؟! خب همین چند خط قبل من در نهایت تابع downloadرو فراخوانی کردم چون قرار بود ادامه کار رو انجام بده !!! پس نباید از تابع download ای که شامل پارامتر URL بود استفاده می کردیم.

و اما تابع finishedHead که وظیفه این تابع دریافت اطلاعات از هیدر و ارسال اون برای کاربر هستش.

```
void MyHttp::finishedHead()
{
```

```
    _Timer.stop();
    _bAcceptRanges = false;


    QList<QByteArray> list = _pCurrentReply->rawHeaderList();
    foreach(QByteArray header, list)
    {
        QString qsLine = QString(header) + " = " + _pCurrentReply-
>rawHeader(header);
        addLine(qsLine);
    }


    if (_pCurrentReply->hasRawHeader("Accept-Ranges"))
    {
        QString qstrAcceptRanges = _pCurrentReply->rawHeader("Accept-
Ranges");
        _bAcceptRanges = (qstrAcceptRanges.compare("bytes",
Qt::CaseInsensitive) == 0);
        qDebug() << "Accept-Ranges = " << qstrAcceptRanges <<
_bAcceptRanges;
    }


    _nDownloadTotal = _pCurrentReply-
>header(QNetworkRequest::ContentLengthHeader).toInt();


    // _CurrentRequest = QNetworkRequest(url);
    _CurrentRequest.setRawHeader("Connection", "Keep-Alive");

_CurrentRequest.setAttribute(QNetworkRequest::HttpPipeliningAllowedAttribut
e, true);
    _pFile = new QFile(_qsFileName + ".part");
    if (!_bAcceptRanges)
    {
        _pFile->remove();
    }
    _pFile->open(QIODevice::ReadWrite | QIODevice::Append);


    _nDownloadSizeAtPause = _pFile->size();
    download();
}
```

و در صورت کلی تابع اتمام دانلود به صورت زیر هستش...

```
void MyHttp::finished()
{
    qDebug() << __FUNCTION__;


    _Timer.stop();
    _pFile->close();
    QFile::remove(_qsFileName);
    _pFile->rename(_qsFileName + ".part", _qsFileName);
    _pFile = NULL;
    _pCurrentReply = 0;
    emit downloadComplete();
}
```

این جا یک موردی هست که باید یک توضیحی بدم که گمراه کننده نباشه... نوشتیم چاپ کن __FUNCTION__رو **__FUNCTION__** ! نوعی ماکرو (**MACRO**) هستش که به صورت پیش پردازنده عمل میکنند و برای ما اطلاعاتی در رابطه با وضعیت کاری این تابع ارسال خواهد کرد ! یعنی فقط به درد برنامه نویس و توسعه دهنده میخوره یجور LOG میشه بهش گفت که برای کامپایلر های موجود در C++و PHP قابل شناسایی هستش.

موارد زیادی هستش از این ماکروهای استاندارد از پیش تعریف شده که میتونید در رابطه با اینها تحقیق کنید

__FILE__, __LINE__, :و__FUNCTION__

و تابع **downloadProgress** از اسمش هم مشخصه وظیفه این تابع مشخص کردن وضعیت و اطلاعات کلی در رابطه با مقدار اندازه دریافت شده و یا مقدار اندازه باقی مانده همراه با زمانش رو مشخص و نمایش خواهدد داد.

```
void MyHttp::downloadProgress(qint64 bytesReceived, qint64 bytesTotal)
{
    _Timer.stop();
    _nDownloadSize = _nDownloadSizeAtPause + bytesReceived;
    qDebug() << "Download Progress: Received=" << _nDownloadSize << ":
Total=" << _nDownloadSizeAtPause + bytesTotal;


    _pFile->write(_pCurrentReply->readAll());
    int nPercentage =
static_cast<int>((static_cast<float>(_nDownloadSizeAtPause + bytesReceived)
* 100.0) / static_cast<float>(_nDownloadSizeAtPause + bytesTotal));
    qDebug() << nPercentage;
    emit progress(nPercentage);


    _Timer.start(5000);
}
```

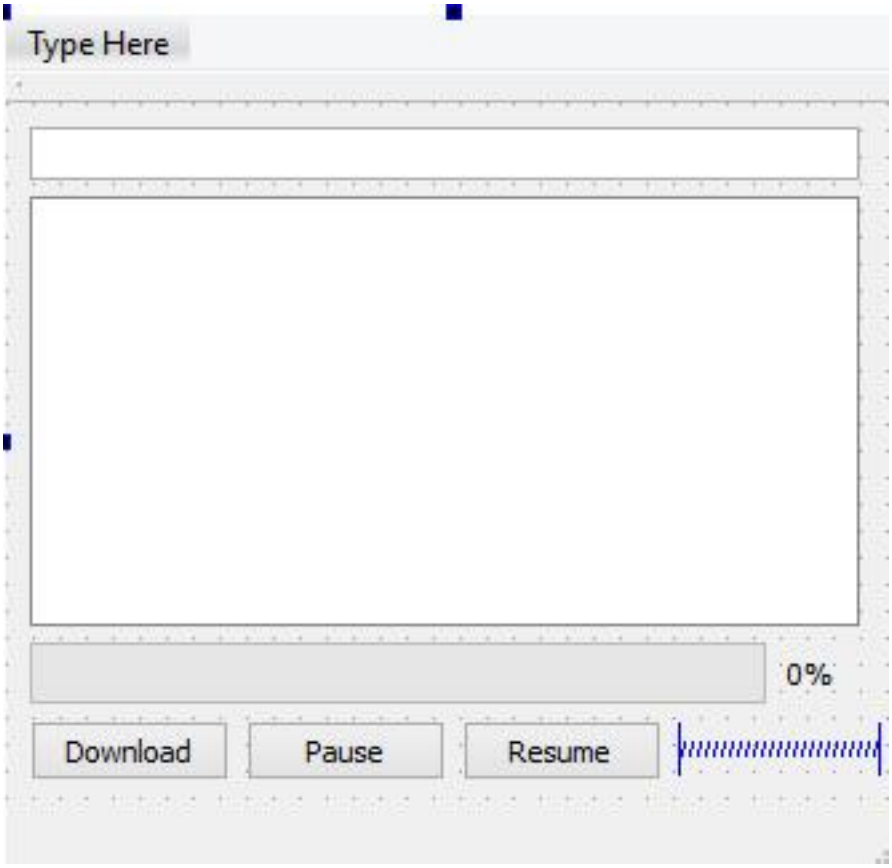
و باز هم یک تابعی برای مشخص سازی مشکلات ممکن..

```
void MyHttp::error(QNetworkReply::NetworkError code)
{
    qDebug() << __FUNCTION__ << "(" << code << ")";
}
```

و تابع آخر هم برای مشکلات ناشی درtimeout

```
void MyHttp::timeout()
{
    qDebug() << __FUNCTION__;
}
```

خب حالا من میخوام در حالت طراحی از این توابع استفاده کنم قبل از هر چیزی فرم زیر رو طراحی میکنیم...



کد های مربوط به فایل هیدر فرمم به صورت زیر خواهد بود که شامل سیگنال ها و متغیر ها و آبجکت های مورد نظرمون هستش...

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H


#if QT_VERSION >= 0x050000
#include <QtWidgets/QMainWindow>
#else
#include <QMainWindow>
#endif


#include "downloadmanager.h"



namespace Ui {
    class MainWindow;
}


class MainWindow : public QMainWindow
{
    Q_OBJECT
```

```
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();


    private slots:
    void addLine(QString qsLine);
    void progress(int nPercentage);
    void finished();


    void on_downloadBtn_clicked();
    void on_pauseBtn_clicked();
    void on_resumeBtn_clicked();




protected:
    void changeEvent(QEvent *e);


private:
    Ui::MainWindow *ui;
    DownloadManager* mManager;

};

#endif // MAINWINDOW_H
```

یه توضیحی در رابطه با کد زیر بدم شاید برای خیلی ها مشخص نشده باشه چیه...

```
#if QT_VERSION >= 0x050000
#include <QtWidgets/QMainWindow>
#else
#include <QMainWindow>
#endif
```

ما در C++ انواع ماکرو هارو داریم که وظیفه اینها قبل از پیش پردازش شدن هست تا اجرا بشن اینا چکار میکنن دقیقا قبل از اینکه کد های ما خونده بشن اجرا میشن شرایط رو بررسی و بر اساس اون شرایط بین کد های ما سوئیچ خواهند شد مثلا در این کد گفتین اگر ماکروی QT_VERSION مقدارش بزرگتر از 0x050000 یعنی کیوت نسخه 5.0 بود برو فلان مسیر فایل فلان رو فراخوانی کن در غیر اینصور از مسیر زیر فایل ها و هیدر هارو برام فراخوانی کن...

این ها قواعد خاصی دارن برای خودشون در رابطه با اینجور چیزا تحقیق کنید مثلا 0x050000 رو میتونید به اینصورت هم بنویسید5.0 :

حالا بریم سراغ فرم لود که لازمه کد های زیر رو بنویسیم...

```
ui->urlEdit->setText("http://qt.digia.com/Static/Images/QtLogo.png");


QStandardItemModel *model = new QStandardItemModel(0, 1, this);
ui->listView->setModel(model);


mManager = new DownloadManager(this);


connect(mManager, SIGNAL(addLine(QString)), this, SLOT(addLine(QString)));
connect(mManager, SIGNAL(downloadComplete()), this, SLOT(finished()));
connect(mManager, SIGNAL(progress(int)), this, SLOT(progress(int)));
```

طبق فرمی که طراحی کردم نام تکست باکسم رو گذاشتم **urlEdit** که ابتدا به صورت پیشفرض آدرس تصویر لوگوی Qt رو بهش اختصاص میدم.

درخط بعدی یک کپی از کلاس **QStandardItemModel** ساختم که وظیفش فراهم کردن یک مدل عمومی برای ذخیره سازی داده های دلخواه هستش.

مدلم رو به شیئ listView که روی فرمم قرار دادم تخصیص میدم و در خط بعد یک کپی از کلاس downloadو سه خط بعدی توسط تابع **connect** سیگنال ها و اسلاتها باهم دیگه ارتباط برقرار می کنند.

روی دکمه Download اسلاتش رو ایجاد و کد زیر رو مینویسم...

```
ui->listView->reset();
QUrl url(ui->urlEdit->text());
mManager->download(url);
ui->downloadBtn->setEnabled(false);
ui->pauseBtn->setEnabled(true);
```

دکمه Pause به صورت زیر...

```
mManager->pause();
ui->pauseBtn->setEnabled(false);
ui->resumeBtn->setEnabled(true);
```

دکمه Resume به صورت زیر...

```
mManager->resume();
ui->pauseBtn->setEnabled(true);
ui->resumeBtn->setEnabled(false);
```

و برای نمایش دادن اطلاعات مربوطه متغیری داشتم به نام AddLine از نوع QStringبود تو فایل h.

طرح فرمم بگردین پیداش میکنید... تابع مربوط به اینرو باید به این صورت بنویسیم...

```
void MainWindow::addLine(QString qsLine)
```

```
{
    int nRow = ui->listView->model()->rowCount();
    ui->listView->model()->insertRow(nRow);
    ui->listView->model()->setData(ui->listView->model()->index(nRow, 0),
qsLine);
}
```

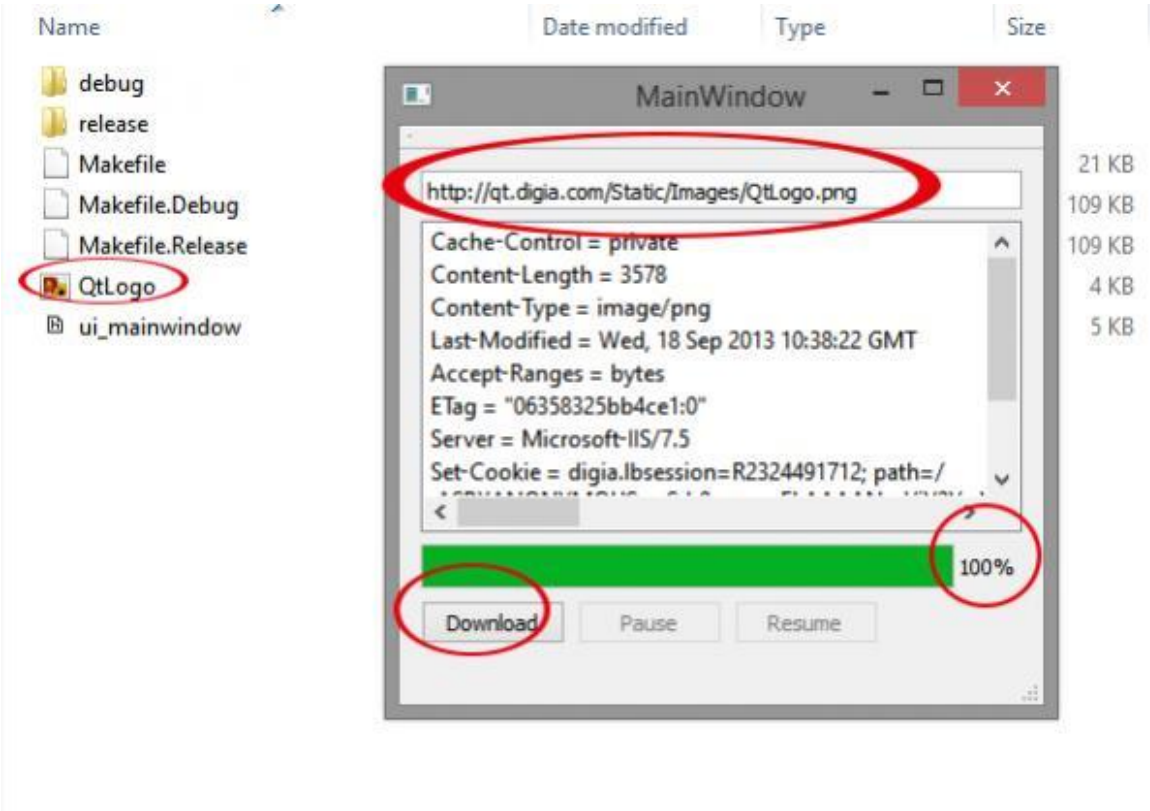
و تابع مربوط به ProgressBar به صورت زیر...

```
void MainWindow::progress(int nPercentage)
{
    ui->progressBar->setValue(nPercentage);
}
```

تابع... Finished

```
void MainWindow::finished()
{
    ui->downloadBtn->setEnabled(true);
    ui->pauseBtn->setEnabled(false);
    ui->resumeBtn->setEnabled(false);
}
```

و نتیجه کار به صورت زیر خواهد بود....



در رابطه با پروتکل FTP هم دقیقا مثل HTTP هستش

ادامه مرحله سی سوم: معرفی و با باینری و سریالیز کردن آبجکن ها

در رابطه با بحث Serialization میتونم اینگونه توضیح دهم که به طور کلی در کامپیوتر و سیستم های ذخیره سازی محتویات فایل ها به صورت های متنی و باینتری ذخیره می شوند.

حالا در حالت عادی یا همان پیشفرض اگر شما طبق مثال زیر رشته ای رو در داخل یک فایل بنویسیم...

```
// writing on a text file
#include <iostream>
#include <fstream>
using namespace std;


int main () {
ofstream myfile ("example.txt");
if (myfile.is_open())
{
myfile << "This is a line.\n";
myfile << "This is another line.\n";
myfile.close();
}
else cout << "Unable to open file";
return 0;
}
```

کاملا به صورت عادی و متنی در فایل مورد نظر ذخیره خواهد شد ولی در حالت باینری تمامی اشیاء به صورت یک حالت منحصر بفرد و خاصی ذخیره می شوند یعنی قبل از ذخیره مستقیم محتویات در فایل اشیاء مورد نظر به صورت **Byte stream** تبدیل شده و سپس در فایل ذخیره می شود.

ما در کل شاید شنیده باشید دو حالت **Serialization** و **DeSerialization** رو در بحث برنامه نویسی و مبحث ذخیره سازی داده ها داریم که به عمل تبدیل آبجکت به صورت باینری همان **Serialization**و به عکس آن عمل **DeSerialization** میگویند

حالا در این مثال من روشی رو برای خواندن و نوشتن اطلاعات در داخل یک فایل رو به صورت**Serialization** و**DeSerialization** به کمک کلاس **QDataStream** توضیح خواهم داد.

یه پروژه از نوع کنسول ایجاد میکنم و میرم سر اصل مطلب...

من میخوام یه متنی رو به صورت زیر وارد کنم...

```
C++ 1
C++ 2
C++ 3
```

و در نهایت نتیجه ذخیره شده در فایل به صورت باینری باشه ! چطور باید عمل کنم به صورت زیر...

```
#include <QCoreApplication>
#include <QFile>
#include <QString>
#include <QDebug>
#include <QMap>



void SaveObject()
{


int iNum = 64;
QMap<int,QString> iMap;
iMap.insert(1,"C++");
iMap.insert(2,"C++ 2");
iMap.insert(3,"C++ 3");


//Save to file.
QFile iFile("c:/Test/itest.txt");


if(!iFile.open(QIODevice::WriteOnly))
{


qDebug() << "Could not open file !";
return;
}


QDataStream iOut (&iFile);
iOut.setVersion(QDataStream::Qt_5_2);


iOut << iNum << iMap;




iFile.flush();
iFile.close();


}


int main(int argc, char *argv[])
{
QCoreApplication a(argc, argv);


SaveObject();
```

```
return a.exec();
}
```

ابتدا هیدر های QString , QFile و QMap رو فراخوانی میکنم چون هدفم نوشتن در داخل یک فایل هستش از مدل **QIODevice::WriteOnly** استفاده کردم.

بعد اومدم یک تابع تعریف کردم که ابتدا به صورت سلیقه ای یک نوع صحیح در نظر گرفتم برای اینکه میخوام همراه با متن مورد نظرم مخلوط بشه و دنبال اون یک نوع رشته ای از نوع Qmap لیست ساختم و یک سری اطلاعات رو به صورت لیست ایجاد کردم.

توسط دستور زیر یک کپی از کلاس QFile گرفتم و مسیر مورد نظرم رو برای ذخیره سازی محتوا در فایل رو دادم...

```
QFile iFile("c:/Test/itest.txt");
```

دستور بعدیش یک شرط تعریف کردم که اقا اگه فایل باز نشد پیغام فلان رو بده د رغیر اینصورت برو ادامه مراحل... البته دقت کنید که چون قصدم نوشتن در فایل هست از اصلی ترین قسمتش **QDataStream** هست که وظیفه این کلاس فراهم کردن امکان انتقال داده های باینری رو به مجرای **QIODevice** میده که اینجا من یک کپی ازش ساختم و گفتم فایلم رو در نظر بگیر و بر اساس نسخه تعیین شده که اینجا من از ۵.۲ خود Qt استفاده کردم رو بهش اختصاص دادم که اصولا از نسخه های اولیه **Qt_1_0** بگیرید برید تا **Qt_5_3** که الان هست در لیستم موجوده میتونم استفاده کنم.

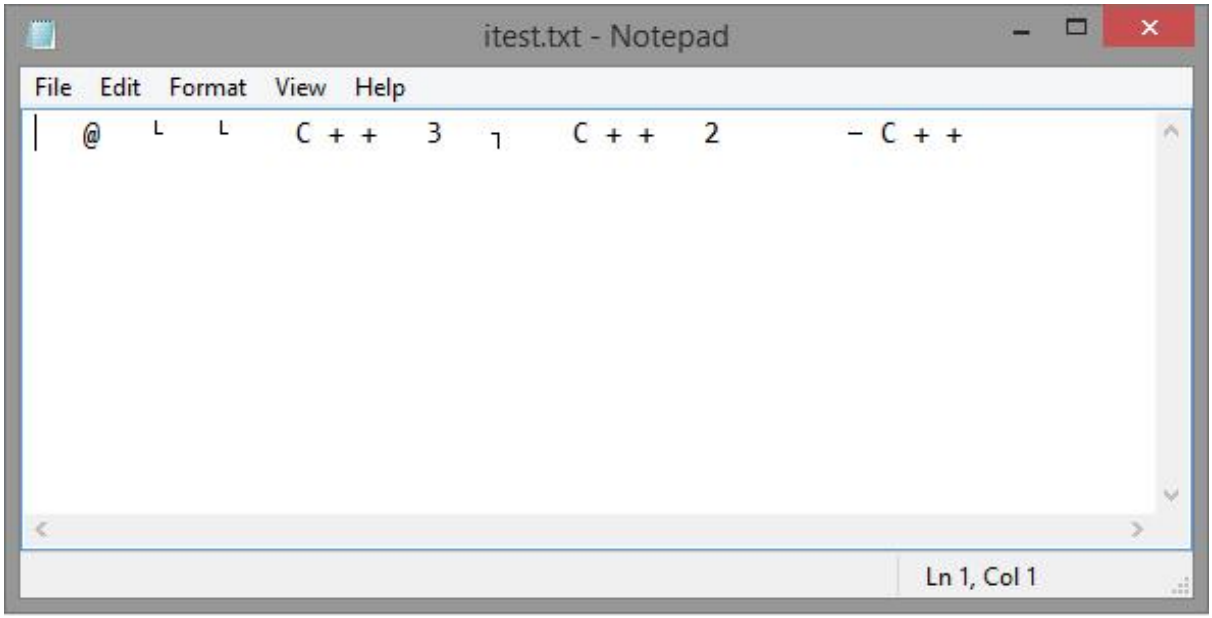
خب در خط بعدیش گفتم iNum رو که از نوع عدد صحیح هستش رو همراه با iMap مخطول و در iOut بریز که به ترتیب اولویت از راست به چپ وارد خواهد شد.

در خط بعدی فلاش زدم که در پست های قبلی توضیح دادم کارش چیه و در نهایت فایل رو بعد از نوشتن و فلاش بستمش.

نتیجه باید در مسیر ذکر شده به صورت باینری ذخیره شده باشه !!! به صورت ضمیمه شده میتونید دریافت و بررسیش کنید فقط مسیر فایل رو طبق شرایط خودتون تغییر بدین.

طبق تصویر مشاهده میکنید که اطلاعات به صورت باینری ذخیره شده است...

نتیجه خروجی حاصل مخلوط شدن متن و اون عددی هستش که اون بالا تعریف کردم یعنی ۶۴.



خب بریم سراغ DeSerialize کردن یا همون خواندن اطلاعات از داخل فایلی که اطلاعاتش باینری شده به صورت زیر...

تابع رو به صورت زیر مینویسم...

```
void ReadObject()
{


int iNum;
QMap<int,QString> iMap;


//Read from file.
QFile iFile("c:/Test/itest.txt");


if(!iFile.open(QIODevice::ReadOnly))
{


qDebug() << "Could not open file !";
return;
}


QDataStream iIn (&iFile);
iIn.setVersion(QDataStream::Qt_5_2);


iIn >> iNum >> iMap;
iFile.close();


qDebug() << "iNum = " << iNum;
```

```
foreach(QString Item , iMap.values())
{
qDebug() << Item;
   }
      }
```

خب حالا اینجا ببینید چه تغییراتی دادیم ابتدا مقدار صحیح رو خالی گذاشتم چون اینجا هدفم خوندن اطلاعات هست پس مقدار از طرف فایل گرفته و برگشت داده میشه به همین متغیر از نوع صحیح (البته منظورم از مقدار همون مقدار ۶۴ هست که به صورت باینری ذخیره شده بود)

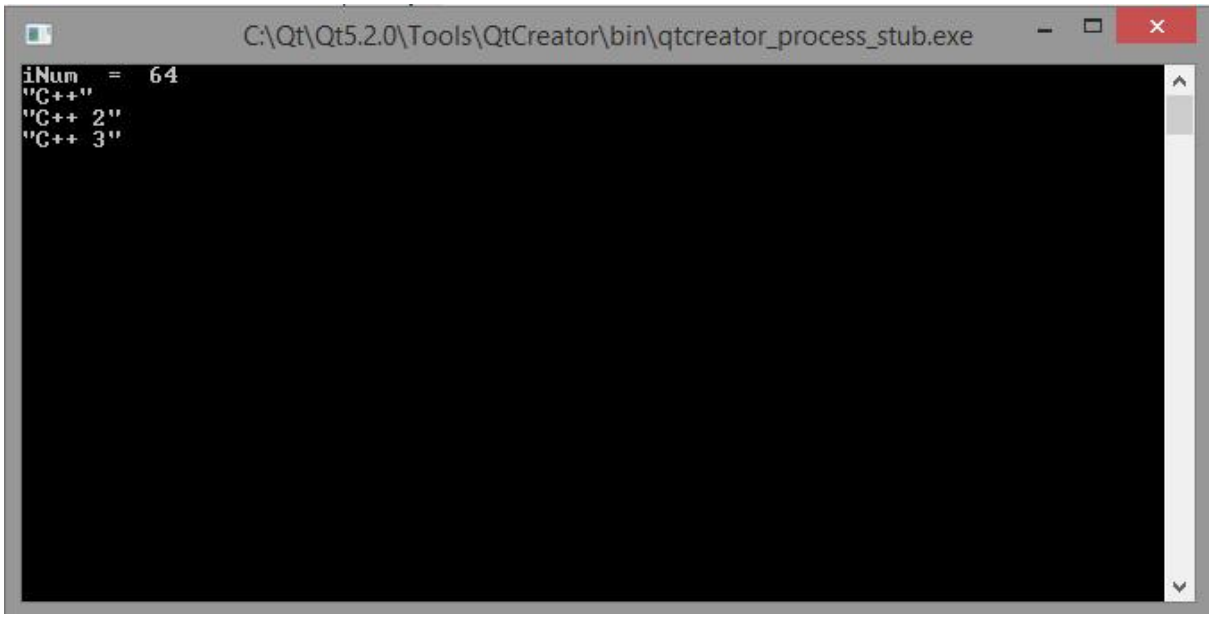
مراحل بعدش دیگه از لیست QMap خبری نیست چون بازم هدف فقط خوندن هست پس میرم سر اصل مطلب...اینبار گفتم QIODevice::ReadOnly و اطلاعات رو به صورت یک رشته به من در کنسول نمایش بده توسط دستور زیر...

ابتدا مقدار بازگشتی عدد رو در متغیر iNum ریخته و چاپش میکنم و برای محتویات رشته ای هم به صورت زیر...

```
foreach(QString Item , iMap.values())
{
qDebug() << Item;
}
```

یه Item از نوع رشته مشخص کردم و بعد مقادیر iMap رو که بازگشت دادم دریافت و در نهایت همرو چاپ کردم.

و نتیجه در نهایت به صورت زیر چاپ خواهد شد...



مرحله سی و چهارم: معرفی و کار با TextStream ها

در رابطه با خواندن نوشتن در رشته ها کلاسی داریم به نام **QTextStream** که امکان نوشتن و خواندن رو برای ما فراهم میکنه... روش کارش مشابه QDataStreamدر پست قبلی هستش با این تفاوت که در اینجا با رشته ها سرو کار داریم...

پروژه ای ایجاد میکنم از نوع کنسول و میرم سراغ کد نویسی که ابتدا تابعی مینویسم برای خواندن با نام **Read** و تابعی مینویسم برای نوشتن با نام **Write** به صورت زیر...

فایل های زیر رو فراخوانی میکنم :

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

کد مربوط بهRead

```
void Read()


{
QFile MyFile("d://MyFile.txt");
if(MyFile.open(QIODevice::ReadOnly | QIODevice::Text))
{
QTextStream MyStream(&MyFile);
QString MyLine;


do


{


MyLine = MyStream.readLine();
qDebug() << MyLine;


}


while(!MyLine.isNull());


}
MyFile.close();
qDebug() << "MyFile Read.";

}
```

و کد مربوط بهWrite

```
void Write()
```

```
{
QFile MyFile("d://MyFile.txt");
if(MyFile.open(QIODevice::WriteOnly | QIODevice::Text))
{
QTextStream MyStream(&MyFile);
MyStream << "Hello \r\n";
MyStream << "World \r\n";



MyStream.flush();
MyFile.close();
qDebug() << "MyFile Written.";

}

}
```

و در نهایت به صورت پشت سرهم هر دو تابع رو به ترتیب اول نوشتن و بعد خواندن فراخوانی میکنم...

```
int main(int argc, char *argv[])
{
QCoreApplication a(argc, argv);


Write(); //First step
Read(); //Second step


return a.exec();
}
```

توضیحات برای این مورد لازم نیست چون همانند QDataStream پیاده سازی شده فقط با این تفاوت که از نوع انحصاری QTextStream استفاده کردیم.

مرحله ۳۵ :معرفی انواع حالت های کامپایل در: Qt

در رابطه با حالت داینامیک (Dynamic) توضیح مختصر:

در این حالت شما در بسیاری از موارد برای توسعه نرم افزار در دو حالت OpenSource و انحصاری قادر خواهید بود.

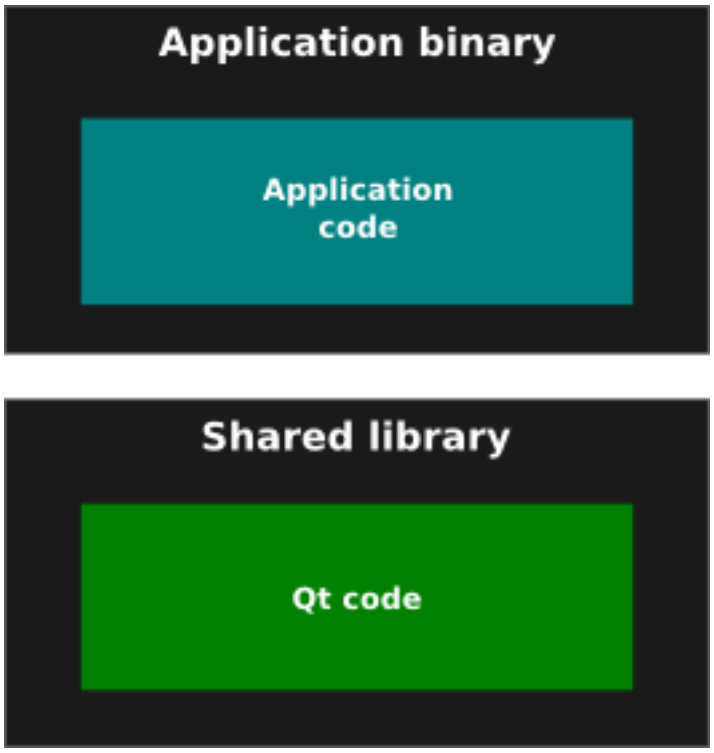
جوانب مثبت و منفی زیر است:

مزایا:

- معمولا برنامه برای کاربر نهایی یا همان End user یک بسته جمع و جور و کامل رو فراهم میکنید و همچنین فایل اجرائی در کم حجمترین و فشرده ترین حالت خارج خواهد شد.
- کتابخانه های Qt رو شما میتونید بدون تغییر و کامپایل مجدد پروژه آن ها را برای توسعه دهندگان بازخورد داده و یا به روز رسانی نمایید و حتی آن ها را تغییر دهید.
- نیاز به منابع سخت افزاری بسیار کمی میباشد برای مثال اشغال حافظه Ram بسیار کمتر از حالت Staticمیباشد.

معایب :

- برای مطمئن شدن از کارائی درست برنامه در هر سیستم و هر سکویی نیاز است بارها و بارها از هر جوانبی برنامه رو نسبت به کتابخانه بررسی نمایید تا وقتی مورد استفاده توسط کاربر یا همان End - Userقرار میگیرد بدون بروز مشکل یا خطایی اجرا شود , معمولا بر روی سیستم های Linux نیز باید کتابخانه های به درستی نصب گردد.
- شما باید مطمئن شوید که تمام کتابخانه های مورد نیاز در سیستم هدف (End User) در دسترس هستند در صورتی که بر روی سیستم مورد نظر در دسترس نیستند باید راه حل مناسبی برای ارائه کتابخانه های مورد نظر فراهم نمایید و خدمات آن را در اختیار کاربر قرار دهید.



در رابطه با حالت استاتیک (Static) توضیح مختصر:

در این حالت شما می توانیدا مطمئن باشید که برنامه شما در هر سیستمی بدون نیاز به پیش نیازی قابل اجرا خواهد بود.

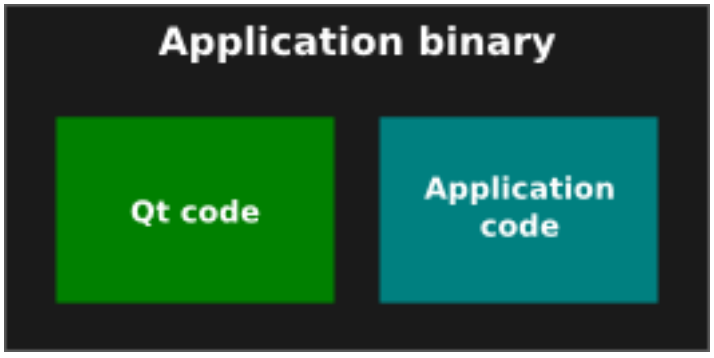
جوانب مثبت و منفی زیر است:

مزایا:

- معمولا برنامه برای کاربر نهایی یا همان End user یک بسته جمع و جور و کامل رو فراهم میکنید.
- برنامه شما میتوایند مستقل از هر نسخه از کتابخانه های موجود بر روی سیستم کاربر برنامه رو اجرا کنید حالا چه Qt4 باشه چه Qt5.3.1 باشه هیچ تداخلی نخواهد داشت.
- کمی سریع نسبت بع حالت داینامیکی اجرا میشود که (در CPU های امروز شما به سختی آن را متوجه خواهید شد)

معایب:

- درخواست های برنامه شما به کتابخانه بسیار زیاد و سنگین خواهد بود زیرا کتابخانه ها نیز به برنامه شما متصل و لینک شده هستند.
- ممکن است برای رفع مشکلات کتابخانه و تغییر / به روز رسانی و ... مجبور به کامپایل مجدد برنامه شوید.
- مصرف منبع Ram در صورت درخواست های پی در پی و چند گانه بسیار زیاد خواهد بود.
- در حالت Runtime شما قادر به اجرای plugins توسط QPluginLoader نخواهید بود.
- کامپایل استاتیک بدون داشتن لیسانس مربوطه از طرف Digia مجاز نیست.



مرحله ۳۶ : نحوه افزودن دیگر کتابخانه های C++ در محیط Qt Creator و استفاده همراه با کتابخانهQt

قصد دارم نحوه افزودن کتابخانه های دیگه رو در محیط Qt توضیح بدم با این روش شما میتونید هر کتابخانه ای رو وارد پروژه کرده و همراه با کتابخانه Qt ازش استفاده کنید برای مثال من خودم به شخصه همیشه رابط کاربری رو با Qt طراحی میکنم و برای برنامه نویسی اصلی از کتابخانه های استاندارد C++ و یا دیگر کتابخانه های اختصاصی مثل Boost و Poco استفاده میکنم.

در این آموزش من نحوه وارد کردن کتابخانه Poco رو براتون توضیح میدم.

زبان C/C++یکی از قابلیت هایی که نسبت به زبان های دیگری مانند C#یا غیره داره نامحدود بودن استفاده از کتابخانه های این زبان هست که به صورت پیشفرض کتابخانه های استاندارد و از قبل تعریف شده در زبان C++ قابل استفاده هستند مانند کلاس های iostream و ... که کاملا پیشفرض روز این زبان ارائه شده است.

الا در رابطه با این موضوع باید توضیحی بدم که اگر برنامه نویس یا توسعه دهنده ای نیازمند استفاده از کتابخانه ای منحصربفرد رو داره میتونه اون رو به زبان C++ اضافه کنه و ازش استفاده کنه برای مثال من برای کار با شبکه و اینترنت کتابخانه POCO رو ترجیح میدم پس برای استفاده از این ابتدا باید پیکربندی و آماده سازی لازم رو انجام بدم به صورت زیر عمل میکنم.

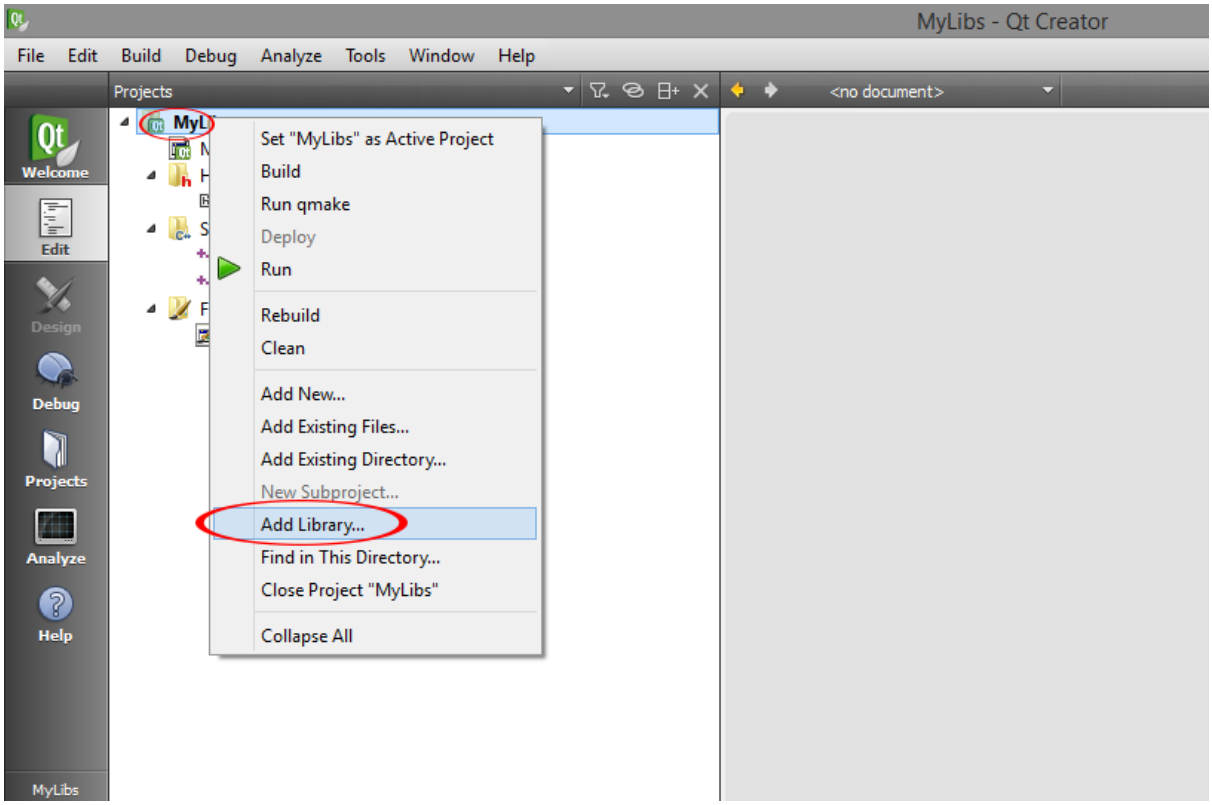
ابتدا کتابخانه رو دانلود میکنم و در مسیر مقابل به صورت دلخواه قرار میدم) C:\poco-1.5.3-all : من امروز در محیط ویندوز هستم برای همین روی ویندوز این رو توضیح میدم توضیحات دقیقا روی محیط Linuxهم صدق میکنه تفاوت چندانی نداره فقط باید کتابخانه رو از قبل کامپایل کرده باشید)

قبل از هر چیزی باید بدونید که هر کتابخانه دارای libs و include هستش هر نوع کتابخانه ای که دانلود کنید این دو تا گزینه رو باید داشته باشه که بعد از کامپایل کتابخانه فایل lib و dll هاش در ویندوز یا Linuxایجاد خواهد شد.

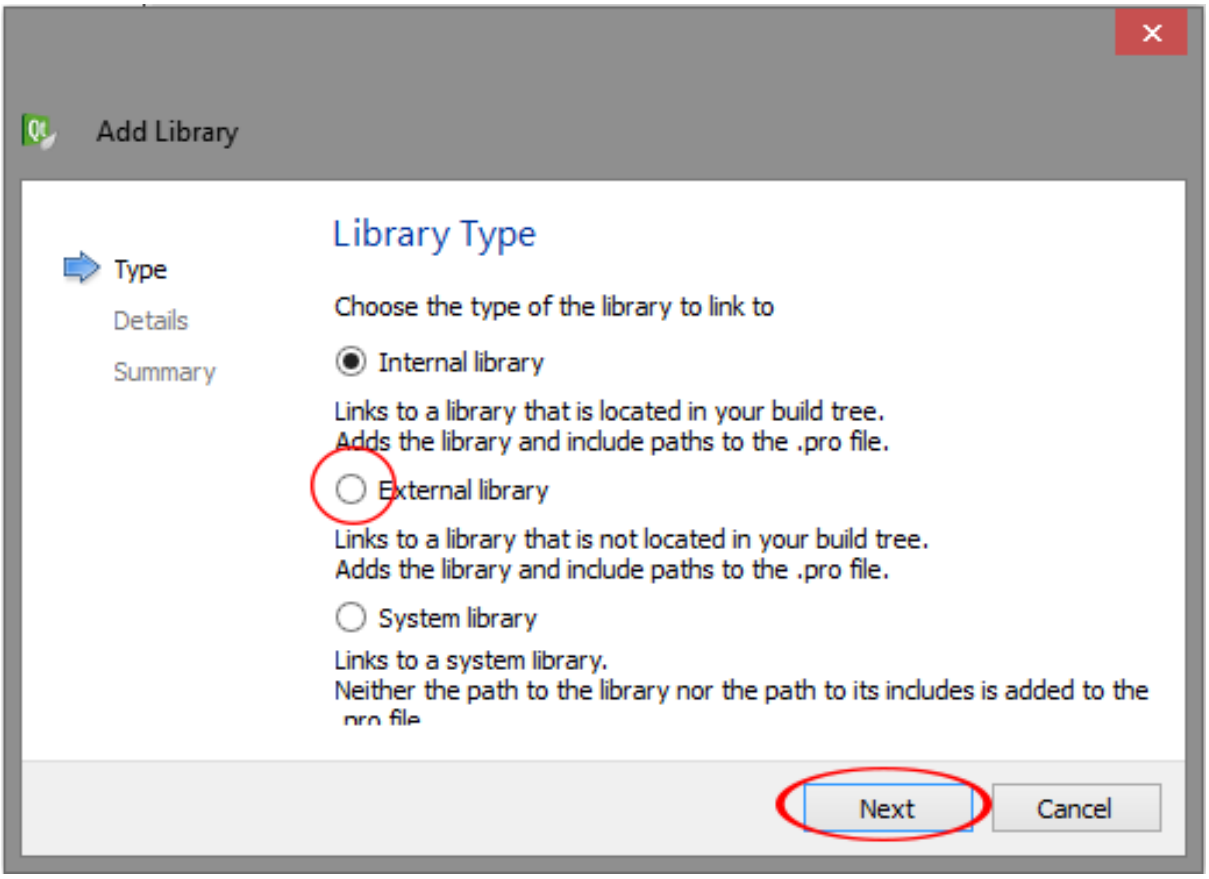
من به دلیل پیچیده بودن کتابخانه) POCO پوکو) رو انتخاب کردم چون شامل کتابخانه های مربوط به Net و ... هستش که برای آموزش هدف من کار با Net هست بنابراین پوشه های include و libs در کنار هم نخواهند بود لذا include مربوط به هر کتابخانه در داخل خودش قرار گرفته.

خب بریم سر اصل مطلب ابتدا یه پروژه ای ایجاد میکنم با نام MyLibs از هر نوعی ایجاد میکنید مهم نیست.

روی پروژه طبق تصویر راست کلیک کنید و گزینه **Add library** رو بزنید.



بعد پنجره زیر نماین میشه که شامل گزینه های زیر هستش:



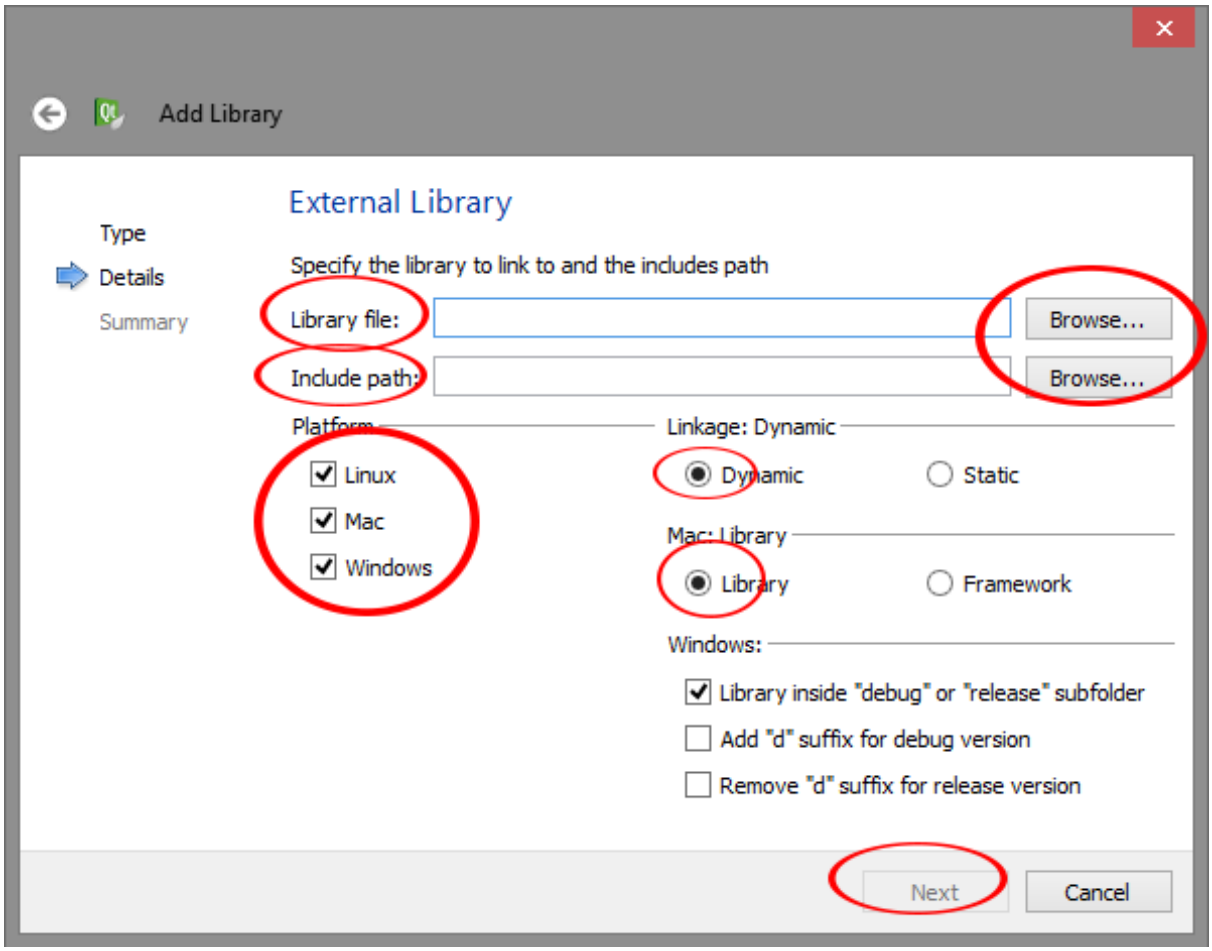
گزینه Internal Library مربوط هست برای زمانی که شما کتابخانه رو در داخل پروژه خودتون ایجاد کردین که معمولا مسیر واقع توسط فایل pro. مشخص خواهد شد به طور کلی کتابخانه های داخلی و غیر Externalرو میتونید از این قسمت شناسایی کنید.

گزینه External library از این گزینه زمانی استفاده میشه که کتابخانه ما در مسیر پروژه ما نیستش یا به طور کلی در داخل پروژه ما قرار نگرفته دقیقا برعکس گزینه اول که میشه همون کتابخانه خارجی یا خارج از پروژه.

گزینه System library هم مربوط میشه به کتابخانه های سیستمی هستش.

و یک گزینه ای هم خواهیم داشت در محیط های Unix که به نام Package Library قابل مشاهده خواهد بود این گزینه هم زمانی مورد استفاده قرار میگیره که شما نیاز دارید کتابخانه رو از طریق سرویس pkg-configدر ایستگاه های Unix که شامل Mac OS X و Linux هستش تنظیم کنید.

حالا ما با این ها کاری نداریم و در استاندارد ترین حالت از External Library استفاده خواهیم کرد بنابراین گزینه External Library رو انتخاب و Next میزنیم.



خب در این قسمت ما گزینه هایی داریم که مربوط به پیکربندی و مشخص کردن مسیر کتابخانه ای هستش که از قبل کامپایل و خروجی های lib و یا dll اون مشخص شده است.

بنابراین گزینه ها وظایف زیر رو دارند:

گزینه Library file مربوط هست به مسیر فایل های مربوط به libs موجود در کتابخانه برای مثال مسیر lib برای کتابخانه Poco در ایستگاه ویندوزی من هستش:-C:\poco

1.5.3-all\lib پس روی گزینه Browse کلیک کرده و این مسیر رو با انتخاب فایل مورد نظرم مثلا PocoNetd.libمشخص میکنم در نهایت مسیر میشه به صورت:-C:\poco

1.5.3-all\lib\PocoNetd.lib

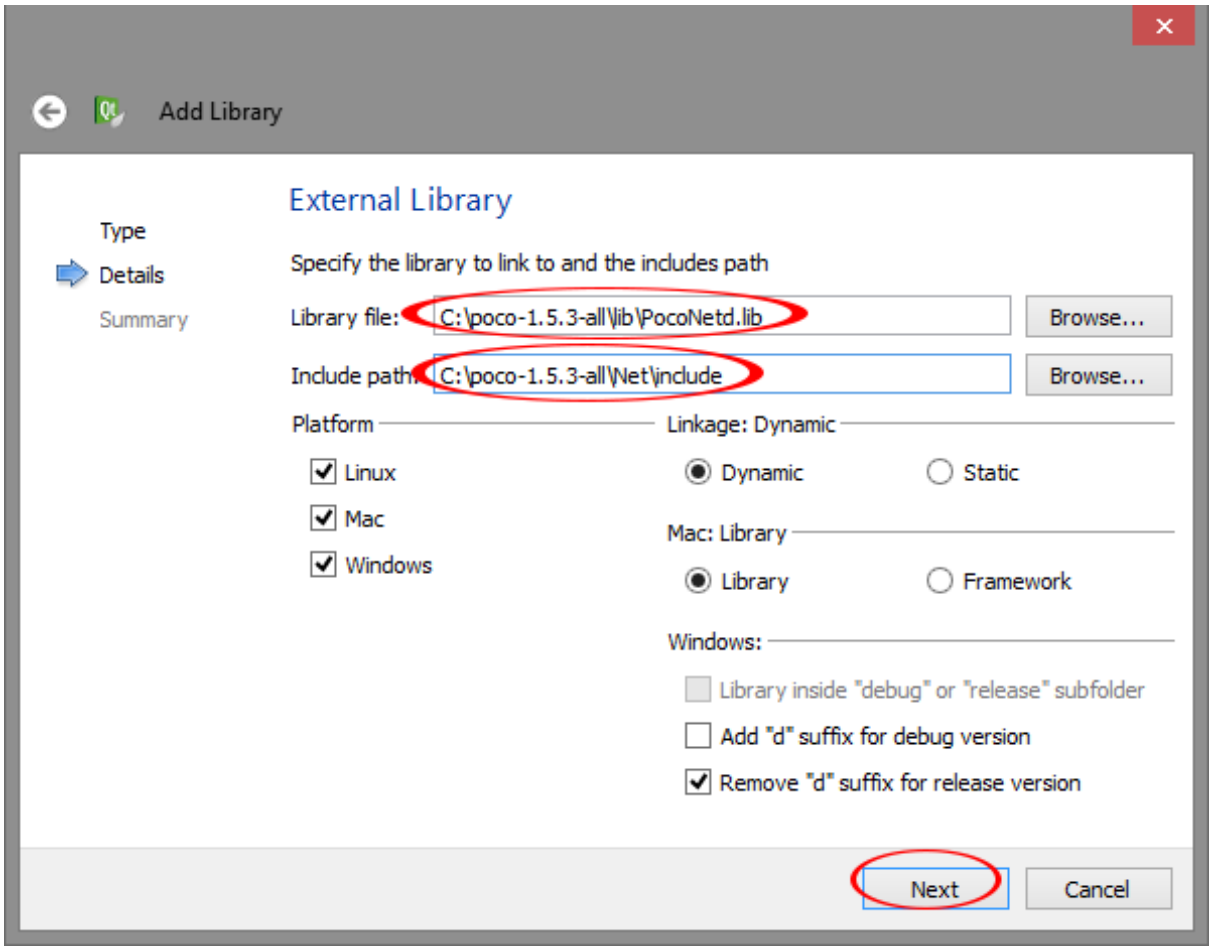
گزینه Include path مربوط هستش به پوشه Include مربوط به کتابخانه که شامل فایل های h. یا همون هدر هاست بنابراین برای اینکه Include مربوط به کلاس های Net رو انتخاب کنم میرم به مسیر : C:\poco-1.5.3-all\Net\includeکه شامل Include های مشخص برای فایل Net هست.

گزینه Platforms مشخص کننده مسیر در فایل pro. برای نوع سیستم عامل هستش.

گزینه Linkage - Dynamic مشخص کننده کتابخانه از نوع Dynamic یا Static و گزینه Frameworkهستش.

و در آخر گزینه ای داریم که مختص ویندوز هست.

تنظیمات رو به صورت زیر انجام میدیم و Next رو میزنیم.



و در نهایت Finished و سپس در داخل فایل pro. کد های زیر رو خواهیم دید.

```
win32:CONFIG(release, debug|release): LIBS += -L$$PWD/../../../../../poco-
1.5.3-all/lib/ -lPocoNet
```

```
else:win32:CONFIG(debug, debug|release): LIBS += -
L$$PWD/../../../../../poco-1.5.3-all/lib/ -lPocoNetd
else:unix: LIBS += -L$$PWD/../../../../../poco-1.5.3-all/lib/ -lPocoNet


INCLUDEPATH += $$PWD/../../../../../poco-1.5.3-all/Net/include
DEPENDPATH += $$PWD/../../../../../poco-1.5.3-all/Net/include
```

کتابخانه من آماده هستش ولی چون در Poco برای استفاده از کلاس مورد نظر پیش نیازی داریم به نام Foundationپس اون گزینه رو هم باید طبق روشی که گفتم اضافه کنیم که در نهایت به صورت زیر خواهد بود:

```
win32:CONFIG(release, debug|release): LIBS += -L$$PWD/../../../../../poco-
1.5.3-all/lib/ -lPocoNet
else:win32:CONFIG(debug, debug|release): LIBS += -
L$$PWD/../../../../../poco-1.5.3-all/lib/ -lPocoNetd
else:unix: LIBS += -L$$PWD/../../../../../poco-1.5.3-all/lib/ -lPocoNet


INCLUDEPATH += $$PWD/../../../../../poco-1.5.3-all/Net/include
DEPENDPATH += $$PWD/../../../../../poco-1.5.3-all/Net/include


win32:CONFIG(release, debug|release): LIBS += -L$$PWD/../../../../../poco-
1.5.3-all/lib/ -lPocoFoundation
else:win32:CONFIG(debug, debug|release): LIBS += -
L$$PWD/../../../../../poco-1.5.3-all/lib/ -lPocoFoundationd
else:unix: LIBS += -L$$PWD/../../../../../poco-1.5.3-all/lib/ -
lPocoFoundation


INCLUDEPATH += $$PWD/../../../../../poco-1.5.3-all/Foundation/include
DEPENDPATH += $$PWD/../../../../../poco-1.5.3-all/Foundation/include
```

خب حالا فایل pro. رو Save کرده و روی گزینه build و بعد Run qmake کلیک میکنیم تا پروژه با تغییراتی که داده شد پیکربندی و آماده شود.

برای آزمایش کد زیر رو مینویسم و کتابخانه بدون مشکل قابل دسترسی و استفاده هستش :

```
#include "mainwindow.h"
#include "ui_mainwindow.h"


#include <Poco/Net/Net.h>
#include <Poco/Net/MailMessage.h>


using namespace Poco::Net;


MainWindow::MainWindow(QWidget *parent) :
QMainWindow(parent),
```

```
ui(new Ui::MainWindow)
{
ui->setupUi(this);


MailMessage mail;
}


MainWindow::~MainWindow()
{
delete ui;
}
```

فقط بازم تاکید میکنم نوع کامپایلری که کتابخانه رو باهاش کامپایل کردین خیلی مهمه مثلا من از قبل این کتابخانه رو با MSVC 2013 نسخه ۳۲ بیت در مد Debug کامپایل کردم پس در Qt هم باید از مد Debugو کامپایلر MSVC 2013 با معماری ۳۲ بیت استفاده کنید.

نه تنها Poco من این مثال رو فقط برای آموزش مراحل تعریف کتابخانه زدم بنابراین از تمامی کتابخانه ها به همین روش میتونید استفاده کنید.