Bucureşti, Str. Smochinului nr. 8, sector 1,
cod poştal 031310, România

Difuzare / distribuţie carte: tel/fax: 021 348 14 74
mobil: 0733 131 145, 0728 084 802
e-mail: difuzare@edituraeikon.ro

Redacţia: tel: 021 348 14 74
mobil: 0728 084 802, 0733 131 145
e-mail: contact@edituraeikon.ro
web: www.edituraeikon.ro

Editura Eikon este acreditată de Consiliul Naţional
al Cercetării Ştiinţifice din Învăţământul Superior (CNCSIS)

Descrierea CIP a Bibliotecii Naţionale a României
POPA, DAN INFORMATICIAN
Introducere în limbajul C++ / Dan Popa. - Bucureşti : Eikon, 2018
ISBN 978-606-711-880-3

004

Tehnoredactor: Mihăiţă Stroe

Editor: Valentin Ajder

Dan Popa

Introducere în limbajul C++ extins folosind Qt Creator

EIKON
Bucureşti, 2018

Cuprins

C1

Legenda Qt-ului

A fost odată ca niciodată că de n-ar fi nu s-ar povesti. Pe vremea când lumea era împărțită în mici regate care se numeau Windows, Linux, QNX, Unix și așa mai departe. Pe vremea când războaiele zguduiau lumea și oamenii trebuiau să trăiască cu armele lângă ei. Se povestește că într-o țară nordică, necopleșită de dușmani, oamenii prinseseră niște ani de pace după marele război și după cel care l-a urmat, cel de-al doilea mare război și încercau să-și reia cursul vieții. Unii mai erau betegi, alții ajungeau la doctori de voie de nevoie, o parte îmbătrâniseră așa că spitalele erau pline. Cu timpul, pacea părând a fi statornică, cel puțin așa o vedeau cei din acele vremuri, mai marii țării au înlocuit serviciul militar obligatoriu cu un fel de serviciu civil obligatoriu, astfel încât tinerii să poată să-și folosească și ei priceperea în folosul comunității. Așa au ajuns doi tineri, cu ceva experiență în ale calculatoarelor, la un spital, unde au găsit o mulțime de computere, incapabile să se înțeleagă între ele și mai ales incapabile a-i vorbi omului pe aceeași limbă. Fiindcă ele proveneau din acele regate diferite de care v-am pomenit. Cei doi aveau de scris o aplicație comună tuturor acelor calculatoare. Dar inventariind bibliotecile de obiecte de la acea vreme au găsit că nu exista nici una care să fie portabilă și să poată fi folosită pe toate sistemele de operare. Și aceasta cu toate că, anii de început ai P.O.O.-ului fiind deja trecuți, toți știau că, în teorie cel puțin, se pot scrie biblioteci de obiecte dotate cu încapsulare (deci ascunzând detaliile implementării), moștenire, (putând fi dezvoltate prin adăugiri repetate, derivări și (re)programare a diferențelor) și polimorfism (putând să se comporte

diferit, după nevoi). Dar cu toate acestea asemenea biblioteci portabile nu existau. Și aceasta a fost prima lor concluzie: aveau nevoie de o bibliotecă comună portabilă înzestrată cu mecanisme ascunse, încapsulate, de acces (unitar și transparent pentru programator) la detaliile specifice ale platformelor de bază. Așa se spune – într-o legendă modernă - că ar fi început povestea Qt-ului. Cu o bibliotecă de butoane și Widget-uri, elemente grafice portabile.

Apăruse totuși o problemă: dacă un programator a fi scris un cod independent de sistemul de operare, cum ar fi știut calculatorul să producă după link-editare un binar pentru o platformă, un sistem de operare sau pentru altul? Era clar că sub codul acela comun trebuia să mai fie un vrăjitor, un fel de preprocesor care să facă cumva să adapteze obiectele generale la detaliile sistemului de operare de bază. De aici a apărut utilitarul QMake.

Problema adaptării codului la diferite sisteme de operare nu era chiar ușoară, sistemele de operare erau distincte, bucla de mesaje Windows se dovedise o adevărată pată neagră în istoria informaticii – era o soluție de compromis moștenită de la interfețele Windows de pe DOS – vulnerabilă și rigidă (toate mesajele trebuiau să treacă prin gâtuitura ei) iar concurența din lumea Unix inventase ceva mult mai bun, serverul grafic X – un server adevărat, cu autentificarea aplicațiilor și o cu totul altă filosofie.

Cum să le împaci și să faci un mecanism care să se potrivească pe toate, să fie abstract și flexibil și să se lase transcris apoi în celelalte feluri de cod: cu bucla de mesaje, cu server X și care mai erau pe acolo?

Generalitatea cerea ca orice obiect, orice buton de exemplu, oriunde ar fi pus în aplicație să poată lansa o altă metodă a altui obiect – așa cum orice lampă de birou poate fi alimentată din orice priză – cu condiția să-i întinzi un prelungitor până acolo.

Problema era foarte importantă, un concept lăudat în POO, utilizat în C, C++, C#, Visual Studio și așa mai departe, conceptul de obiecte

incluse în alte obiecte, își arătase deja limitele și programatorii erau chinuiți de probleme practice cum este cea din următoarea istorioară:

Să presupunem că ai scris pentru un client o aplicație. Ea avea o fereastră mare, principală, colorată și înăuntrul ei, ca niște obiecte incluse, alte chenare și butoane. Când apăsai pe butoane se întâmplau diverse lucruri. În spatele fiecărui buton se ascundea câte o bucată de cod, o subrutină care la apăsarea pe buton (On Click ar zice unii) procesa date și actualiza variabile globale (care variabile globale sunt altă pacoste pentru stabilitatea programelor).

În ziua următoare se întorcea clientul cu aplicația și adăuga o cerință banală: Te rog, pune lângă butonul acela, din chenarul cutare, un buton sau o listă derulantă sau altceva ca să schimb culoarea ferestrei principale, că monitorul meu – zicea el – este cam întunecat.

Această mică și aparent inofensivă cerință nu se putea rezolva direct cu conceptele Programării Orientate Obiect!! Din nefericire, în programarea orientată pe obiecte clasice obiectele exterioare au acces la câmpurile obiectelor interioare dar nu și invers! Obiectele interioare, cum era butonul din poveste – care erau ca un om într-o vale între munți - nu puteau vedea dincolo de versanții înconjurători. Ori exact așa ceva le trebuia, să transmită un mesaj dincolo de munți. În termeni de specialitate, spunem că nu puteau sparge încapsularea, decât cu niște trucuri cam nedemne de P.O.O. Firește se putea recurge la bucla de mesaje sau la un pointer la obiectul aplicație, pus într-o variabilă globală (periculos) sau transferat din metodă în metodă până la butonul buclucaș care ar fi știut ce să facă cu el: să-l dereferențieze și să modifice culoarea ferestrei.

În fond ce le trebuia unor asediați – ce le-ar fi trebuit cavalerilor dintr-o cetate asediată? O legătură cu exteriorul, un tunel sau altceva de felul acesta, care să-l scoată pe un mesager dincolo de încercuire, acolo unde avea(u) nevoie. Sau dacă erau mai mulți să-i trimită în mai multe locuri.

În Qt un astfel de tunel se numește *conexiune*. Are o intrare și o ieșire numite *semnal* – cel care pleacă – și *slot* – locul undeva ajunge

semnalul. Vi le puteți imagina cam ca niște capete ale unui prelungitor electric.

Bineînțeles că puriștii P.O.O.-ului au sărit în sus (deoarece se încălca încapsularea). Ca întotdeauna în istoria științei când o nouă descoperire, invenție sau inovație este gata să amenințe bulgărele de aur al celor deja adunate. Cu siguranță cunoașteți asemenea exemple. S-a spus: Aparatul de zbor mai greu decât aerul nu va zbura niciodată! Spațiul și timpul sunt unice, nemodificabile, fixe. – Și a apărut teoria relativității! Apa este incompresibilă! Și G. Constantinescu a inventat teoria sonicității și motorul sonic – cel care transmitea energia prin vibrațiile și compresiile apei. În Academia Franceză s-ar fi strigat: Nici un nenorocit de metal nu poate imita nobilele organe ale graiului omenesc! Iar trimisul lui Edison prezenta onoratei adunări ... fonograful. Astăzi facem sinteză vocală pe calculatoare și colecționăm pattern-uri de voci.

Conexiunile care definesc în mod abstract legăturile dintre obiecte sunt apoi procesate de preprocesorul versiunii de Qt destinat platformei care ne interesează și sunt generate fișiere de cod adecvate sistemului de operare pentru care (nu *pe care*) dezvoltăm aplicația. Acestea sunt compilate în codul mașină nativ al acelui sistem de operare sau procesor. Sau în bytecode, ca în cazul Qt-ul pentru Android.

Ulterior Qt-ul a evoluat. S-au adăugat și biblioteci de obiecte mai puțin grafice, obiecte pentru rețea, obiecte pentru multimedia, obiecte pentru grafică în OpenGL șamd. Era clar că într-un proiect nu pot fi incluse obligatoriu toate și așa a apărut ideea unui fișier proiect – util atunci când lucrați la un proiect cu IDE-ul - Qt Creator se numește – cum altfel?

De fapt sunt mai multe IDE-uri, medii de lucru care pot folosi Qt-ul.

1) KDevelop este unul – cu el s-a lucrat în cadrul proiectului KDE – interfața grafică Linux care a dat look-ul Windows-ului 7 și 8. (Apropos, știați că exista un KDE pentru Windows care poate înlocui Explorer-ul?)

2) Cu CodeBlocks puteți de asemenea dezvolta mici proiecte în Qt.

3) Nokia a mizat pe mediul de dezvoltare Qt Creator iar Ubuntu a oferit o versiune cu adăugiri numită Ubuntu-SDK.

4) Compilatoare ca Watcom C pot fi folosite împreună cu Ide-urile asociate la a dezvolta proiecte cu biblioteci Qt.

5) Designerul vizual Qt Designer ajuta și el la urgentarea scrierii aplicațiilor, permițând generarea vizuală a obiectelor – de un tip aparte, (UI), ale interfeței grafice. Acestea pot fi folosite de-a gata în programe, tot ce trebuie este să știm că fiecare buton e un câmp în obiectul compus, să nu le încurcăm între ele. Apoi este simplu: tragem cabluri de legătură – pardon conexiuni! - de la aceste elemente ale interfeței la funcțiile obiectului nostru aplicație. Și gata. Aplicația va răspunde la comenzile interfeței.

Primul fișier proiect

Pentru a putea începe să lucrați cu ediția a 5.a Qt Creator-ului aveți nevoie la fiecare lucrarea de un fișier *proiect*. De unde îl luați? Dacă faceți o lucrare care continuă sau dezvoltă un exemplu existent puteți pleca de la fișierul proiect al acelui exemplu. Dacă este un nou proiect care pornește conform unuia dintre șabloanele prestabilite, Qt Creator-ul va crea el un proiect pentru dumneavoastră. Mai este și varianta să deschideți un Empty Project, un proiect minimal. La versiunea Qt Creator 4.6 (de la Nokia) acest fișier indică doar locațiile surselor și headerelor necesare proiectului. De când Qt 5-ul s-a dezvoltat și diversificat, s-a complicat nițel și sintaxa fișierelor proiect. Acum pot să mai conțină pe lângă surse .h sau .cpp sau de alt fel și *resurse* (bitmap-uri, png-uri etc) și alte feluri de indicații: Tipul de proiect, *aplicație* sau set de demo-uri în subdirectoare sau alte feluri (lista va rămâne deschisă, deja s-a adăugat QML-ul...), bibliotecile incluse pe lângă baza, *core,* a Qt-ului. Căile către diferite directoare etc. Partea bună este că în documentațiile claselor veți găsi și câte o indicație despre ce se mai adaugă la *core*-ul Qt. În exemplele care vor urmă vom folosi mai întâi elemente de *gui – Graphic User Interface – și Widgets* – acele figurine care

apar pe interfețele grafice și pot fi din colecțiile predefinite (Butoane, Check Box-uri etc) sau pot fi create (pe gustul clientului sau conform nevoilor aplicației) de programator. În fișierul proiect, cu semnul # încep rândurile care conțin comentarii. Numele care apare în dreptul directivei TARGET va fi prezent și în bara aplicației, în cazul în care nu-l modifică programul.

```
TEMPLATE = app
TARGET = QtHelloWorld
QT += core gui
QT += widgets
#INCLUDEPATH += .
SOURCES += exemplul1.cpp
```

Scrieți cu editorul de text favorit textul de mai sus și salvați-l în directorul viitorului proiect, cu extensia .pro.

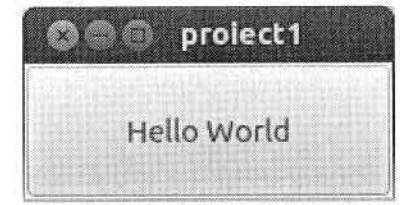
Dacă ați mai folosit deja Qt Creator-ul puteți începe un nou proiect și alegeți din rubrica Other a colecției de șabloane de proiecte proiectul de tip Empty Project – proiect deschis. În cazul Qt-ului 4.6.4 n-ar mai fi trebuit să editați nimic la el. În cazul Qt-ului din seria 5 adăugați textul de mai sus dar fără numele fișierului sursă C++. Qt Creatorul adaugă automat în proiect numele fișierelor cu care lucrăm, pe măsură ce, dând un click dreapta pe proiect și alegând "Add (new) file", le adăugăm la proiect.

Nota: Atenție, dacă Qt Creator-ul semnalează faptul că nu găsește nici o regulă pentru a obține, de exemplu main.o sau alt .obj, trebuie să verificați fișierul proiect. Este posibil ca numele sursei .cpp din care el va proveni să fie *de două ori* scris acolo. Ștergeți numele care se repetă, iar eroarea va dispărea. Rândurile succesive se separă prin semne "\" - acest caracter înseamnă "continuarea este pe rândul următor."

Primul program Qt

Am întâlnit prima oară acest exemplu pe CD-ul unei distribuții Linux numită Dragon Linux. Ceea ce însemna că era oferit sub licența generala GNU GPL, întregii comunități. Ulterior am găsit cam același exemplu și în documentațiile on-line distribuite de firma Trolltech. El este un fel de Hello World al lumii Qt-iști-lor. Programul conține practic minimul de cod din care se poate obține o aplicație. Scrieți deocamdată acest program în editorul dumneavoastră favorit și salvați-l sub numele *main.cpp* sau *exemplul1.cpp*. (Presupunem că abia de acum încolo învățați să folosiți Qt Creatorul și până acum nu ați lucrat deloc cu el.) Dacă ați folosit *main.cpp* treceți-l și în fișierul proiectului, printre surse. Cele două fișiere, un .pro și un .cpp se vor afla în același director, să-i zicem Qt1.

```
#include <qapplication.h>
#include <qpushbutton.h>
int main (int argc, char **argv)
{
  QApplication a(argc, argv);
  QPushButton h("Hello World");
  h.resize(100,30);
  h.show();
  return a.exec();
}
```



În imagine: Proiectul în execuție.

Autorul programului a avut probabil în minte așa ceva: Și voi va trebui să vă obișnuiți, ca în filmul Matrix, să „vedeți" imaginea obiectelor ce vor rezulta direct din cod.

Primul program Qt, comentat

Nu prea sunt multe moduri de a scrie un program Qt pentru prima lecție. Cu siguranță acesta trebuie să creeze o aplicație, adică, în Qt, un obiect aplicație. Pentru a folosi clasa obiectelor aplicație trebuie importat fișierul header <qapplication.h>. Mai nou se folosește

o specificație a clasei care aduce mai mult cu numele acesteia din Qt: <QApplication>.

Cu această notație, headerele se scriu exact ca și denumirile claselor Qt și fără acel .h pe care îl foloseam de fiecare dată în C. Dat fiind că actualmente Qt are în jur de 900-1000 de clase – în funcție de versiune – și numărul lor crește zi de zi, o asemenea convenție este binevenită. În plus, ea adaugă un nivel intermediar de indirectare, permițând unui cod care folosește o clasă de obiecte, de exemplu de butoane, să folosească o altă clasă, eventual pe altă platformă, fără a modifica nimic din codul sursă al aplicației.

```
#include <QApplication>
#include <QPushButton>

int main (int argc, char **argv)
{
  QApplication a(argc, argv);
  QPushButton h("Hello World");
  h.resize(100,30);
  h.show();
  return a.exec();
}
```

În imagine: Proiectul în execuție.

Ca urmare este recomandat să scrieți în noul stil, fără .h.

Funcția *main* este funcția principală a programului. Observați că ea primește parametrii *argc* – numărul de argumente din linia de comandă și tabela de stringuri *argv* – care conține aceste argumente. (Primul este chiar numele programului). Nu renunțați la ele, sunt necesare obiectului aplicație pentru niște inițializări de care depinde aspectul acestuia. Se folosesc argumente suplimentare în linia de comandă pentru a configura obiectul aplicație și el trebuie să le primească.

```
int main (int argc, char **argv)
{
  QApplication a(argc, argv);
  QPushButton h("Hello World");
```



```
  h.resize(100,30);
  h.show();
  return a.exec();
}
```

Corpul, blocul de cod al funcției main, conține diverse comenzi date obiectelor. Totul urmează clasica filosofie a lumii obiectuale: Obiectele se nasc (le inițializează un constructor), trăiesc (prestează serviciile care le sunt cerute) și mor (sunt dealocate). Obiectul aplicație are sarcina de a pune în funcțiune tot mecanismul acelor semnale și sloturi fără de care aplicația nu ar putea funcționa și ar rămâne doar o fereastră, frumos desenată, dar inactivă. Apropos de desenare și acest lucru trebuie cerut explicit ferestrei principale. Unii programatori separă astfel codul pe mai multe linii, evidențiind grupurile de servicii succesiv cerute unui anumit obiect.

```
QApplication a(argc, argv);
```

Am cerut crearea obiectului aplicație de către constructorul clasei sale. Urmează să cerem crearea obiectului de tip QPushButton, pe nume .h, și să-l punem la treabă.

```
QPushButton h("Hello World");
h.resize(100,30);
h.show();
```

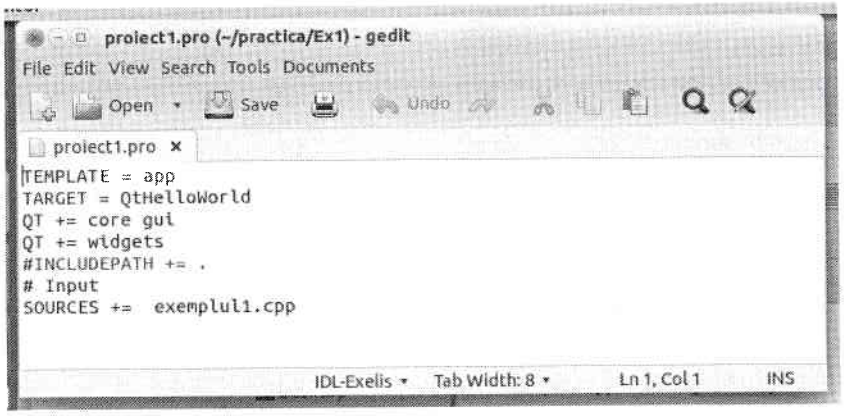
Am cerut crearea obiectului de tip QPushButton și l-am pus să se redimensioneze, căpătând anumite dimensiuni apoi să se afișeze pe ecran (de fapt în fereastra aplicației) cu dimensiunile acelea, proaspăt căpătate. Urmează să-l activăm, ca să poată fi apăsat. Facem acest lucru cerând obiectului *a*, adică aplicației să se *execute*. Apelăm metoda *exec()* a obiectului *a* scriind întotdeauna *a.exec()* la finalul programului. Acest lucru pornește tot mecanismul procesării evenimentelor, click-urilor și așa mai departe. Fereastra aplicației devine activă, puteți apăsa pe butoane, observați cum i se schimbă aspectul, chiar dacă deocamdată apăsarea pe el nu declanșează nici o altă acțiune suplimentară.

return a.exec();

Unii autori descriu apelurile metodelor *resize()* și *show()* ale obiectului buton de mai sus în termeni de procesare a evenimentelor (ca și cum toate platformele software ar avea buclă de evenimente ca Windows-ul. Ei spun că prima lansează un eveniment *resize()* iar cealaltă un eveniment *show()*. Dar să nu rămâneți ancorați de această explicație. Probabil la Qt-ul pentru Windows se ajunge într-adevăr la așa ceva până la urmă dar nu este exclus să apară și alte implementări ale Qt-lui, pe alte platforme, care să funcționeze altfel. (Cum se întâmplă la serverul X .)

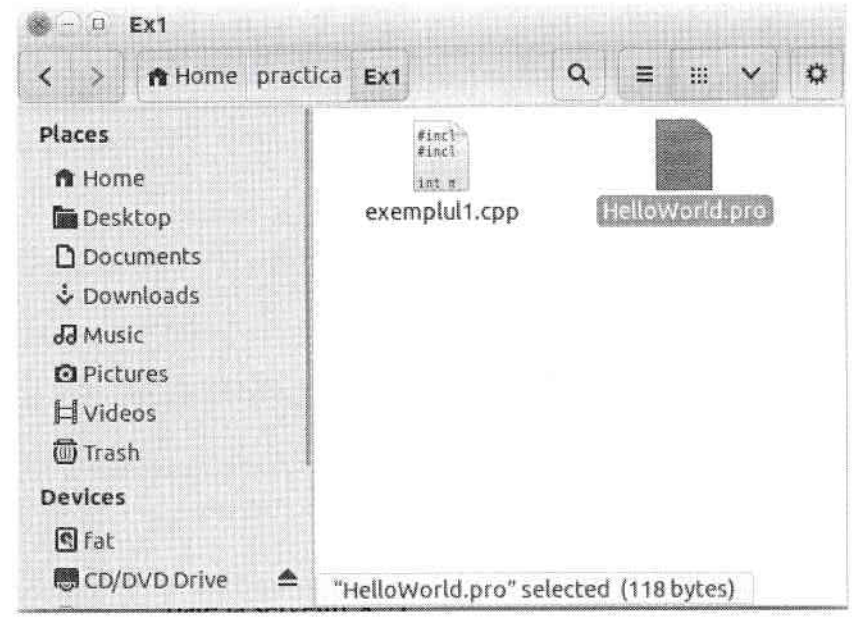
Primul program Qt, realizarea practică

Dacă nu ați folosit Qt Creator-ul niciodată, pur și simplu editați cele două fișiere de la începutul capitolului în editorul favorit și salvați-le în viitorul dosar al proiectului.



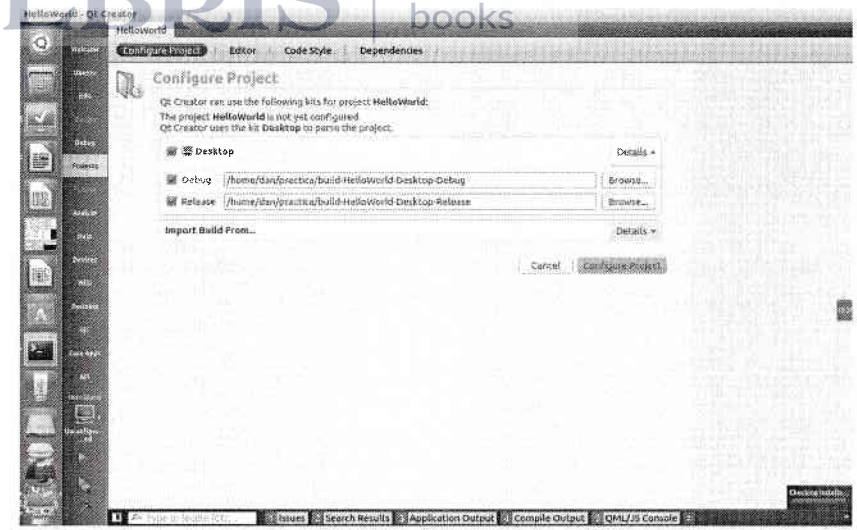
În imagine: Fișierul .pro al proiectului, editat cu editorul gedit

După editarea celui de-al doilea fișier veți avea un dosar cu două fișiere, laolaltă:



În imagine:Cele două fișiere în dosarul proiectului.

Numele fișierului .pro poate fi ales cum doriți, sau poate fi schimbat dar numele pus fișierului .cpp trebuie să figureze corect în fișierul .pro. Exact așa cum este el numit. Nu este obligatorie folosirea numelui *main.cpp*. Acum deschideți Qt Creatorul dând un click pe fișierul .pro (sau eventual alegând din meniul dat de tasta dreaptă a mouse-ului opțiunea Open With Qt Creator). Se va deschide Qt Creator-ul și vi se va solicita să configurați proiectul. Dacă nu vă interesează detaliile (și nici nu doriți să dezvoltați pentru alte platforme, folosind alte *kit*-uri de dezvoltare) dați un simplu click pe butonul de configurare. (Pe acest Ubuntu 14.04 cu Qt Creator-ul numit Ubuntu-SDK configurarea cu un click a funcționat perfect, dar nu mă criticați dacă lucrurile se vor schimba în viitor.)



În imagine: Configurarea proiectului în Qt Creator

Apoi, pe pagina următoare puteți admira proiectul în mediul Qt Creator, gata de lucru. Veți recunoaște cu ușurință:

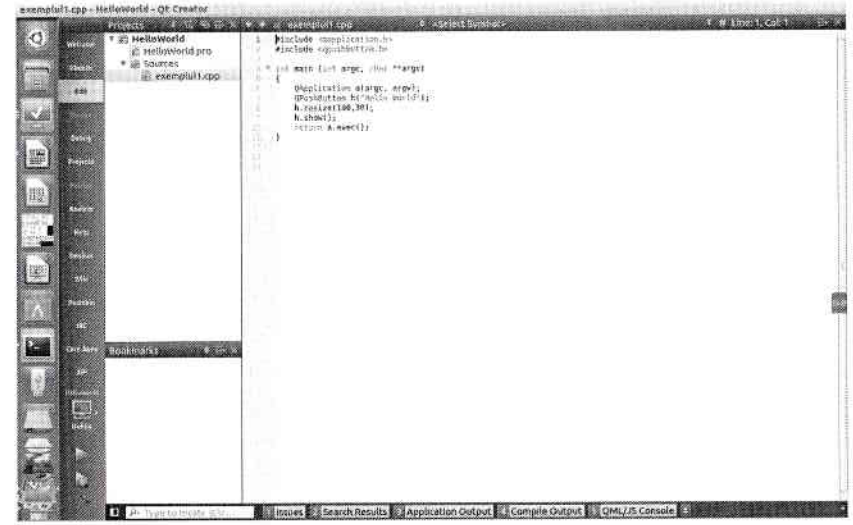
a) În stânga sus se vede proiectul sub forma unei reprezentări arborescente. Țineți minte că un click dreapta pe proiect va scoate la vedere un meniu ascuns, care este de folos atunci când nu sunt vizibile meniurile Qt Creatorului – cazul din imagine. Folosind acest meniu veți gestiona proiectul, fie că veți face curățenie, fie că îl veți da lui QMake, sau îl veți recompila – cu Rebuild, fie veți adauga fișiere sau fișier de resurse (.qrc). Meniul conține și alte comenzi, puteți să adăugați sau să eliminați alte fișiere din proiect și să precizați dacă vor fi păstrate sau șterse după eliminare.

b) În dreapta sus vor apare ferestrele de editare, iar lângă fereastra curentă de editare, pe marginea dreaptă, va apare și Help-ul, documentația, ca atunci când, de exemplu veți pune cursorul mouse-ului pe un nume de clasă și veți apăsa tasta F1. Reținenți manevra: Puneți cursorul de mouse pe un nume de clasă Qt și apăsați F1 (sau Fn și F1).

c) În stânga jos este o mică listă a ferestrelor cu fișiere deschise, fie că fac parte din proiect fie că nu.

d) În dreapta jos vor apare ferestrele cu mesajele provenite din desfășurarea operațiilor, de obicei *compilarea* dar și cele provenite de la depanator.

Tot în stânga jos, primul buton triunghiular verde este Run iar al doilea, cu un semn suplimentar, este Debug. Bara (cafenie aici) din extrema stângă este aceea a utilitarului Unity din distribuția Ubuntu și nu face parte din software-ul QtCreator.



În imagine: Aplicația în curs de editare, gata de a fi compilată și rulată.

Acum mai rămâne de apăsat Run, triunghiul verde din stânga, pentru a porni aplicația!

Exercițiu: Măriți dimensiunea butonului la 300x300 astfel încât întreg titlul QtHelloWorld – sau un altul lung, de pe bara ferestrei - să fie vizibil.