1. O CMakeu słów kilka..

Cmake jest narzędziem do generowania plików do komplilacji dla konkretnego kompilatora, systemu operacyjnego i konfiguracji. Ważne jest to, że on sam w sobie nie kompiluje programu. Przykładowo dla Linuksa tworzony jest plik Makefile.

Jego podstawowym plikiem jest "CmakeLists.txt". W nim zawarte są komnedy napisane specyficznym dla tego narzędzia językiem skryptowym. Ciężko na to patrzeć jak na język proceduralny. Raczej są to umieszczone w odpowiedniej kolejności informacje dotyczące naszego projektu.

2. Przykład pliku CmakeLists.txt wraz z opisem (komentarze zaczynają się od #)

```
# Minimalna wersja programu cmake potrzebna
# do przetworzenia danego projektu.
# Jest to ważne, gdyż niektóre polecenia (makra)
# nie są przetwarzane przez starsze wercje.
cmake_minimum_required (VERSION 2.8.11)

# Nazwa projektu. Tworzona jest zmienna PROJECT_NAME,
# do której możemy się później odwołać poprzez ${PROJECT_NAME}.
# Może być ona użyta do nazwy pliku wykonywalnego.
project(nazwa_projektu)

# Komenda pozwala na nautomatyzowane wyszukiwanie bibliotek.
# Niestety do poprawnego działania wymaga
# plików <nazwa>-config.cmake, przez co ciężko się ją stosuje
# do bibliotek, które tych plików nie dostarczają. Ale Boost,
# Qt i OpenCV takie mają. W przypadku odnalezienia biblioteki
# tworzone są zmienne <nazwa>_INCLUDE_DIRS z plikami nagłówkowymi
# oraz <nazwa>_LIBRARIES ze ścieżką dostępu do biblioteki.
find_package(Boost REQUIRED)

# Komenda set ustawia wartość zmiennej podanej
# jako pierwszy argument na wartość składającą się
# z pozostałych argumentów.
set(SRC_FILES main.cpp dir/another.cpp )

# Polecenie tworzy plik wykonywalny
# z podanych jako argumentu plików źródłowych.
# Można też zamieścić w liście argumentów zmienną,
# która zawiera ścieżki (względne) do plików źródłowych.
# Nie trzeba dodawać do kompilacji headerów.
add_executable (nazwa_pliku_wykonywalnego
      ${SRC_FILES}
      directory/youCanAddAnother.cpp
)

# Komenda służy do podawania ścieżki do folderu,
# w którym znajdują się pliki nagłówkowe.
# Są one szczególnie potrzebne w przypadku
# bibliotek. Ścieżkę do swoich plików nagłówkowych
# jest sens podawać, gdy nie znajdują się one
# razem z plikami źródłowymi, a nie chcemy zawsze
# podawać całej względnej ścieżki dostępu do nich.
target_include_directories( nazwa_pliku_wykonywalnego
      myFolderWithHeaders/include
      ${Boost_INCLUDE_DIRS}
)
```

```
# Służy do podpinania bibliotek do pliku wykonywalnego.
# Może być podana pełna ścieżka do pliku biblioteki
# lub też podana zmienna uzyskana poprzez polecenie
# find_package.
target_link_libraries(nazwa_pliku_wykonywalnego
      /usr/local/lib/libeigen3.so
      ${Boost_LIBRARIES}
)
```

3. (Linux, Linia poleceń) Aby projekt bazujący na cmakeu skompilować najlepiej zacząć od stworzenia osobnego folderu (np. O nazwie build), z którego wywoła się komendę cmake. Pliki pośrednie, wykonywalne, jak i reszta śmieci są tworzone w miejscu wywołania polecenia, więc wywołanie cmake z osobnego folderu pozwoli nam na skuteczne odseparowanie plików projektu, od tego co można swobodnie usunąć. Jako argument podaje się ścieżkę do pliku CmakeLists.txt. Wywołanie cmake spowoduje stworzenie pliku Makefile, który to należy wywołać poleceniem make. W ten oto sposób otrzymujemy ciąg poleceń dla systemu linux:

```
cd ~/ściezka/do/projektu
 mkdir build
 cd build
 cmake ../
 make
 ./executable
```

4. Są programy okienkowe do obsługi cmake, lecz nie korzystałem. W Windows powinno działać podobnie w przypadku korzystania z wiersza poleceń, lecz nie korzystam, więc ciężko mi zapewnić.

5. O Qt słów kilka..

Qt to zestaw bibliotek dla języka C++, które dostarczają wielu użytecznych i dosyć prostych w użyciu klas. Mamy tam klasy reprezentujące wątki (QThread), czasomierze (QTimer), które pozwalają wykonywać pewną czynność co określony czas, klasy potrzebne do tworzenia graficznego interfejsu użytkownika (QMainWindow, QDialog, QLabel, QPushButton) i masę innych. W Qt ważne są dwie rzeczy. Pierwsza z nich to system sygnałów i slotów, a druga to działająca w tle pętla zdarzeń. Zacznę od tego drugiego. Qt proponuje z założenia programowanie oparte na zdarzeniach. Zdarzenia są gromadzone w buforze tworzonym dla każdego wątku i przetwarzane przez obiekty działające w tym wątku. Zdarzenia są powodowane przez naciśnięcie przycisku przez użytkownika w interfejsie, lub też wyemitowanie sygnału przez jakiś obiekt (np. czasomierz). Obsługa pętli zdarzeń jest powstaje w momencie wywołania w kodzie metody "exec()" na jednym z głównych obiektów biblioteki Qt (np. QCoreApplication). Wtedy to dla każdego otrzymanego sygnału zaczynają być wywoływane sloty do nich połączone.

Prosty przykład programu w Qt znajdziecie załączony do tego mini-tutoriala. Przykład ten jednocześnie przedstawia sposób w jaki będzie funkcjonowac nasz projekt na komputerze centralnym.

6. Cmake i Qt

Występują drobne różnice w sposobie pisania skryptu CMakeLists.txt jeśli chodzi o projekt napisany z użyciem Qt oraz projekt bez użycia Qt. Ten drugi jest przedstawiony powyżej. Natomiast sposób podłączania bibliotek Qt wraz z opisem w komentarzach znajdziecie poniżej.

```
cmake_minimum_required(VERSION 2.8.11)
```

```
project(testproject)

# Automatycznie dodaje bieżący folder źródeł
# (zmienna ${CMAKE_CURRENT_SOURCE_DIR} –
# tam gdzie znajduje się plik CmakeLists.txt)
# oraz bieżący folder budowania (zmienna
# ${CMAKE_CURRENT_BINARY_DIR} – tam, skąd
# wywoływany jest program cmake) do ścieżek,
# gdzie szukane są pliki nagłówkowe. Powód dodawania
# tego polecenia wyjaśni się później.
# Jest to ekwiwalent komendy:
# target_include_directories( helloworld
#     ${CMAKE_CURRENT_SOURCE_DIR}
#     ${CMAKE_CURRENT_BINARY_DIR}
# )
set(CMAKE_INCLUDE_CURRENT_DIR ON)

# Automatycznie wykonuje komendę “moc”
# na plikach nagłówkowych, które tego potrzebują.
# Wymaga to dalszego wyjaśnienia. Qt wprowadza
# system sygnałów i slotów, lecz niesie to za soba
# dalsze konsekwencie. Przed kompilacją projektu
# muszą być wykanane dodatkowe przekształcenia kodu,
# które umożliwią zadziałanie wymienionego mechanizmu.
# Aby pliki nagłówkowe (bo o nie głównie chodzi)
# mogły zostać automatycznie wykryte jako wymagające
# przekształcenia muszą zostać podane razem
# z plikami źródłowymi w funkcji "add_executable".
set(CMAKE_AUTOMOC ON)

# Odnajdywanie biblitek. Qt, to nie jest jedna biblioteka.
# Jeżeli chcemy uzyskać dostęp do bibliotek związanych
# z zagadnieniami sieciowymi, musimy dopisać niżej
# drugą komendę find_package(Qt5Network).
# Do pozostałych modułów podobnie.
find_package(Qt5Widgets)

# Nie wymaga wyjasnień
add_executable(helloworld WIN32 main.cpp myQtClass.h myQtClass.cpp)

# Linkowanie bibliotek nie odbywa się już przez
# podanie ścieżki lub przez podanie wartości zmiennej
# Qt5Widget_LIBRARIES, tylko poprzez wpisanie "Qt5::Widgets".
# Nie wiem z czego wynika taki zapis, lecz dzięki niemu
# automatycznie dodają się nagłówki bibliotek (nie musimy
# pisać target_include_directories(...))
target_link_libraries(helloworld Qt5::Widgets)
```