6.5. qmake

6.5.1. Egyszerű qmake környezet kialakítása

A **qmake** a cmake-hez hasonlóan magas szintű eszközrendszer források menedzselésére. A **qmake** a **Qt SDK** része, amelyet a Trolltech cég fejleszt a NOKIA részeként. A **qmake** használatához installálnunk kell a **Qt SDK** csomagot. A következőkben bemutatott konfigurációs fájlok a **qmake** 2.0-ás verziójától használhatók.

A qmake fejlesztői észrevették, hogy a konfigurációs fájlok néhány fő típusba sorolhatók a legtöbb esetben: rekurzívan további könyvtárakat adunk meg, amelyek konfigurációs állományokat tartalmaznak, statikus vagy dinamikus könyvtárakat vagy alkalmazásokat specifikálnak. Ennek megfelelően a qmake fő erőssége, hogy az egyes targetek általános beállításait elrejtik, és egy speciális **TEMPLATE** változóval határozhatjuk meg, hogy a konfigurációs állomány egyes utasításait melyik lehetséges minta konkreiztálásaként értelmezze.

Hozzunk létre ismét egy, az előzőekhez hasonló könyvtárszerkezetet:

6.5.35. forráskód: Könyvtárszerkezet és konfigurációs állományok qmake esetén

```
qmaketest
     srcda
         srcda.pro
         mainDynamic.c
     srcdl
         srcdl.pro
         testDynamicLibrary.c
         testDynamicLibrary.h
     srcsa
         srcsa.pro
         mainStatic.c
     srcsda
         srcsda.pro
         mainStaticAndDynamic.c
     srcsl
         srcsl.pro
         testStaticLibrary.c
         testStaticLibrary.h
     qmaketest.pro
```

Lássuk az egyes projekt fájlok (.pro) tartalmát:

6.5.36. forráskód: qmaketest.pro

```
TEMPLATE = subdirs

SUBDIRS += srcsl
SUBDIRS += srcdl
SUBDIRS += srcsa
SUBDIRS += srcda
SUBDIRS += srcsda
```

A gyökérkönyvtárban található projekt fájl a **subdirs** mintát követi, azaz további könyvtárakat ad csak meg, melyek konfigurációs állományokat tartalmaznak. Ezeket a további könyvtárakat a speciális **SUBDIRS** változóhoz kell hozzáadnunk a **+=** operátorral.

6.5.37. forráskód: srcsl/srcsl.pro

```
TEMPLATE = lib
TARGET = testStaticLibrary
QT -= core
QT -= gui
CONFIG -= qt
CONFIG += static

HEADERS += testStaticLibrary.h

SOURCES += testStaticLibrary.c
```

Az első értékadással a library template-et, azaz mintát adjuk meg, majd target-ként a **testStaticLibrary** nevet specifikáljuk. Mivel a **qmake** alapjában véve Qt-t használó projektek konfigurálására készült, néhány Qt könyvtár linkelését alapértelmezetten hozzáadja a projekthez. Ezek eltávolítására a **–=** operátorral kivesszük a speciális **QT** változóból a **core** és **qui** modulokat, valamint a szintén speciális **CONFIG** változóból a **qt** sztringet, ami további Qt specifikus eszközöket ad a fordítási és linkelési parancsokhoz. A **CONFIG** változóhoz hozzáadott **static** sztring definiálja, hogy statikus könyvtárat szeretnénk létrehozni. Ezt követően a könyvtárhoz tartozó header fájlokat hozzáadjuk a **HEADERS** változóhoz, a forrásfájlokat pedig a **SOURCES** változóhoz.

6.5.38. forráskód: srcdl/srcdl.pro

```
TEMPLATE = lib
TARGET = testDynamicLibrary
QT -= core
QT -= gui
CONFIG -= qt
CONFIG += shared

HEADERS += testDynamicLibrary.h

SOURCES += testDynamicLibrary.c
```

A dinamikus könyvtárhoz tartozó konfigurációs állomány értelemszerűen a target és fájlnevekben tér el, valamint a **CONFIG** változóhoz a **static** helyett hozzáadott **shared** sztringben, amely shared object fájl, azaz dinamikus könyvtár létrehozását specifikálja.

6.5.39. forráskód: srcsa/srcsa.pro

```
TEMPLATE = app
TARGET = mainStatic
QT -= core
QT -= gui
CONFIG -= qt
```

```
INCLUDEPATH += ../srcsl

SOURCES += mainStatic.c

LIBS += ../srcsl/libtestStaticLibrary.a
```

A statikusan linkelendő alkalmazás projekt fájlját az **app** template specifikálásával kezdjük. **TARGET** névként beállítjuk a **mainStatic** sztringet, ami az alkalmazás neve lesz, majd a megfelelő Qt specifikus konfigurációs paraméterek eltávolítása után hozzáadjuk az **INCLUDEPATH** változóhoz a statikus könyvtár elérési útját, a **SOURCES** változóhoz a lefordítandó forrásfájlokat, és a **LIBS** változóhoz a linkelendő statikus könyvtárakat.

6.5.40. forráskód: srcda/srcda.pro

```
TEMPLATE = app
TARGET = mainDynamic
QT -= core
QT -= gui
CONFIG -= qt

INCLUDEPATH += ../srcdl

SOURCES += mainDynamic.c

SHARED_LIBS += ../srcdl/libtestDynamicLibrary.so

LIBS += ../srcdl/libtestDynamicLibrary.so
```

A dinamikusan linkelendő alkalmazás konfigurációs állománya hasonló a statikusan linkelendő alkalmazáséhoz, a különbség, hogy a dinamikus könyvtár elérési útját adjuk hozzá az **INCLUDEPATH** változóhoz, és magát a könyvtárat a **LIBS** változóhoz.

6.5.41. forráskód: /srcsda/srcsda.pro

```
TEMPLATE = app
TARGET = mainStaticAndDynamic
QT -= core
QT -= gui
CONFIG -= qt

INCLUDEPATH += ../srcdl
INCLUDEPATH += ../srcsl

SOURCES += mainStaticAndDynamic.c

SHARED_LIBS += ../srcdl/libtestDynamicLibrary.so

LIBS += ../srcsl/libtestStaticLibrary.a
LIBS += ../srcdl/libtestDynamicLibrary.so
```

A statikusan és dinamikusan is linkelendő alkalmazás projektfájlja szintén hasonlóan alakul, a fő különbség, hogy a statikus és dinamikus könyvtárak header-jeinek elérési útját és magukat a statikus és dinamikus könyvtárakat is hozzáadjuk a megfelelő változókhoz.

A konfigurációs állományok alapján gyökérkönyvtárban kiadott

```
qmake -r .
```

paranccsal állíthatjuk elő a GNU Makefile-okat, amelyeket a fordítás során felhasználhatunk. A . a feldolgozandó könyvtár elérési útja, a -r a rekurzivitásra utal, azaz ha az elérési úton található projekt

fájlok valamelyike a **subdirs** template-et valósítja meg, akkor az abban megadott alkönyvtárakra is végrehajtja a qmake a konfigurációt. Az előállt **Makefile**-okat felhasználva a **make** utasítással fordíthatjuk le és linkelhetjük könyvtárainkat és alkalmazásainkat.

6.5.2. Fordítási opciók

Lássuk, hogyan bővíthetjük ki a konfigurációt úgy, hogy a debug/release fordítást parancssorból szabályozhassuk! Hozzunk létre a gyökérkönyvtrában egy **conf.pri** nevű állományt az alábbi tartalommal:

6.5.42. forráskód: conf.pri

```
QMAKE_CFLAGS_RELEASE -= -O2
QMAKE_CFLAGS_RELEASE += -O3
QMAKE_CFLAGS_DEBUG += -O0
QMAKE_CFLAGS_DEBUG += -g
```

Az első sorban kivesszük a speciális **QMAKE_CFLAGS_RELEASE** változóból az **-O2** kapcsolót, majd hozzáadjuk a **-O3** kapcsolót. A **QMAKE_CFLAGS_DEBUG** változóhoz hozzáadjuk a **-O0** és a **-g** kapcsolókat. Ezt a kódrészletet include-oljuk minden forrásfájlt tartalmazó könyvtár projekt fájljába a projekt fájlban elhelyezett alábbi utasítással:

```
include(../conf.pri)
```

Azaz például a statikusan és dinamikusan linkelt alkalmazás konfigurációs állománya a következőre módosul:

6.5.43. forráskód: srcsda/srcsda.pro

```
TEMPLATE = app
TARGET = mainStaticAndDynamic
QT -= core
QT -= gui
CONFIG -= qt

include(../conf.pri)

INCLUDEPATH += ../srcdl
INCLUDEPATH += ../srcsl

SOURCES += mainStaticAndDynamic.c

LIBS += ../srcsl/libtestStaticLibrary.a
LIBS += ../srcdl/libtestDynamicLibrary.so
```

Ezt követően a konfigurációs parancsban a **CONFIG** változóhoz hozzáadott **debug** vagy **release** sztringgel specifikálhatjuk a debug vagy release fordítást:

```
qmake -r CONFIG+=debug .
```

A fenti példában a beépített **QMAKE_CFLAGS_RELEASE** és **QMAKE_CFLAGS_DEBUG** változókat használtuk. Emlékezzünk vissza, hogy a CMake setén is rendelkezésre álltak hasonló, beépített változók ugyenezen célra. Lássuk most, hogyan oldhatjuk meg a debug/release fordítást általánosabb eszközökkel!

A qmake legfontosabb konfigurációs eszközeit a korábbi eszközrendszerekhez hasonlóan a változók és feltételes szerkezetek adják. A koncepció a következő: változókhoz opciókat, mint sztringeket

adunk hozzá és távolítuk el a **+=** és **-=** operátorok felhasználásával, és ezt követően a feltételes szerkezetek ezen sztringek jelenlétének alapján állítják be a megfelelő kapcsolókat, flag-eket. Az általánosan használható konfigurációs változó a **CONFIG**, amelyet már korábban is használtunk. A feltételes szerkezetek az alábbi szintaktikát használják:

```
feltétel{
}
```

Ezen szintaktika miatt a feltételes szerkezetet hatáskörnek is nevezik a qmake terminológiájában. Ezen feltételes szerkezetek tetszőleges mélységben egymásba ágyazhatóak. A feltételek lehetnek a **CONFIG** változóban szereplő sztringek, ekkor csak a megfelelő sztringet használhatjuk önmagában feltételként. A feltétel igaz, ha a sztring jelen van a **CONFIG** változóban. Ha más változókat szeretnénk használni, néhány előre megírt tesztfüggvény közül választhatunk, amelyek a változók tartalmát tesztelik:

- **contains(valtozonev, ertek)** - a feltétel igaz, ha a változó tartalmazza az értéket,
- **count(valtozonev, ertek)** - a feltétel igaz, ha a változó éppen **ertek** darab sztringet tartalmaz,
- **exists(fajlnev)** - a feltétel igaz, ha a fájl létezik,
- **isEmpty(valtozonev)** - a feltétel igaz, ha a változó üres,
- **equals(valtozonev, ertek)** - a feltétel igaz, ha a változó értéke éppen **ertek**.

A fordító és linker flag-eket a **QMAKE_CFLAGS** és **QMAKE_LFLAGS** változókban állíthatjuk be. Az előbbi feltételes szerkezet alapján, így a **config.pri** fájlban elhelyezve a

```
debug{
  QMAKE_CFLAGS+= -O0 -g
}
release{
  QMAKE_CFLAGS+= -O3
}
```

feltételes szerkezetekkel éppen úgy bekerülnek a debug/release fordításhoz szükséges kapcsolók a fordítási parancsba, mint a speciális **QMAKE_CFLAGS_DEBUG** és **QMAKE_CFLAGS_RELEASE** változók használata esetén. Ügyeljünk azonban arra, hogy a **debug** és **release** opciók alapértelmezésként szerepelnek a **CONFIG** változóban.

6.5. Feladat: Adjon hozzá konfigurációs lehetőségeket a qmake környezethez úgy, hogy a **warnings** sztring esetén a **-Wall -Wextra** kapcsolók szerepeljenek a fordítási sorban, míg a **usedynlib** sztring esetén a dinamikus könyvtár használatát bekapcsoló **-DUSE_DYNAMIC_LIBRARY** kapcsoló legyen benne a fordítási sorban!

6.5. Megoldás: A **conf.pri** fájlban az alábbi feltételes szerkezeteket kell elhelyeznünk:

```
warnings{
  QMAKE_CFLAGS+= -Wall -Wextra
}
usedynlib{
  QMAKE_CFLAGS+= -DUSE_DYNAMIC_LIBRARY
}
```

A **mainOptional** alkalmazásban a dinamikus könyvtár használatának **config.h** header fájlon keresztül történő bekapcsolása már problémás kérdés, ugyanis a qmake nem tartalmaz eszközöket változó fájlokba történő helyettesítésére, úgy, mint az automake vagy a cmake. A megoldás egy egyszerű **system** függvényhívás lehet, amelyben az operációs rendszernek kiadott **echo** parancsot a megfelelő fájlba irányítva létrehozhatunk akár header fájlokat is:

```
usedynlib{
  system(echo "#define USE_DYNAMIC_LIBRARY" > config.h)
}
```

6.5.3. Külső könyvtárak használata

A külső könyvtárak használatára qmake-ben nem állnak rendelkezésünkre olyan kifinomult eszközök, mint automake-ben vagy cmake-ben. Legkevésbé hordozható megoldás a felhasználandó könyvtárak fordító és linker flag-jeinek rögzítése a konfigurációs állományokban. Ez linux rendszerek esetén többé-kevésbé működőképes, hiszen a különböző forráskód csomagok mind a /**usr** könyvtár különböző alkönyvtáraiba installálják a header-öket és a library-kat is. Windows környezetben ajánlott a külső csomagokat is elhelyezni a fordítási környeztünkben, annak részeként, és ekkor a megfelelő elérési utak rögzítése nem jelent problémát a fordítás és linkelés szempontjából.

Mindazonáltal az **exist**, **for** és **defineTest** makrók felhasználásával bizonyos rugalmasságot vihetünk a build rendszerbe, a leggyakoribb helyeket ellenőrizve, ahol a szükséges csomag előfordulhat.

Beillesztve a **mainPNG.c** forrást jelenlegi környezetünkbe, Linux rendszeren az alábbi sorokkal érhetjük el, hogy a megfelelő flag-ek bekerüljenek a fordítási és linkelési parancsba:

```
exists(/usr/include/png.h){
  INCLUDEPATH+= /usr/include
}
exists(/usr/local/include/png.h){
  INCLUDEPATH+= /usr/local/include
}
exists(/usr/lib/libpng.so){
  LIBS+= -L/usr/lib -lpng
}
exists(/usr/local/lib/libpng.so){
  LIBS+= -L/usr/local/lib -lpng
}
```

6.5.4. Disztribúció létrehozása

Az eddig létrehozott **Makefile**-ok mind tartalmaznak **install** target-et. Nincs ez másképp a qmake által létrehozott **Makefile**-lal sem, azonban ez alapértelmezésben üres, azaz nincs hatása. Ahhoz, hogy környezetünk bizonyos részeit (header-ök, targetek, stb.) installálni tudjuk a **make install** paranccsal, hozzá kell adnunk a megfelelő sztringeket az **INSTALLS** változóhoz. Alkalmazások és könyvtárak esetén az alábbi formában specifikálhatjuk az installálás helyét:

```
target.path=konyvtarnev
INSTALLS += target
```

ahol a **target** nem az aktuális target neve, hanem maga a **target** sztring. Tehát ha például a statikus könyvtárat szeretnénk insallálni a /**usr**/**lib** könyvtárba, akkor az alábbi sorokat kell elhelyeznünk a statikus könyvtárhoz tartozó .pro fájlban:

```
target.path=/usr/lib
INSTALLS += target
```

Újrakonfigurálás után már valóban installál a **make install** parancs.

A qmake forráscsomagok disztribúcióját támogatja, olyan .tar.gz csomagokat hoz létre, amelyek tartalmazzák a forrásfájljainkat valamint a konfigurációs .pro fájlt. Forráscsomag létrehozásánál a qmake a csomagba elhelyezi még a .pro fájlunkban használt beépített funkciók definíciós fájljait, tehát sok néhány soros qmake .pro fájl is bekerül a csomagba, amely függetlenné teszi azt a qmake célgépen használt verziójától. A csomagba bekerülnek a **SOURCES** változóban szereplő fájlok, valamint a **DISTFILES** változónak értékül adott fájlok. Ahhoz tehát, hogy a header fájljok is bekerüljenek a készülő forráscsomagba, hozzá kell adnunk őket a **DISTFILES** változóhoz, azaz a statikus könyvtár esetén például az alábbi módon néz ki a végső projekt fájl:

6.5.44. forráskód: srcsl/srcsl.pro

```
TEMPLATE = lib
TARGET = testStaticLibrary
QT -= core
QT -= gui
CONFIG -= qt
CONFIG += static

include(../conf.pri)

HEADERS += testStaticLibrary.h
DISTFILES += testStaticLibrary.h

SOURCES += testStaticLibrary.c

target.path=/usr/lib
INSTALLS += target
```

Kiadva a **make dist** parancsot, előáll a forráscsomag.