6.4. CMake

6.4.1. Egyszerű CMake környezet kialakítása

A **cmake** eszközrendszer segítségével magas szintű konfigurációs állományokból kiindulva szinte tetszőleges IDE-hez generálhatunk projektfájlokat, beleértve a GNU Makefile-okat is. A **cmake** használatát is az eddigi egyszerű statikus és dinamikus könyvtárakon és alkalmazásokon keresztül szemléltetjük.

Hozzuk létre az alábbi könyvtárszerkezetet, üres **CMakeLists.txt** állományokkal:

```
cmaketest
     srcda
         CMakeLists.txt
         mainDynamic.c
     srcdl
         CMakeLists.txt
         testDynamicLibrary.c
         testDynamicLibrary.h
     srcsa
         CMakeLists.txt
         mainStatic.c
     srcsda
         CMakeLists.txt
         mainStaticAndDynamic.c
     srcsl
         CMakeLists.txt
         testStaticLibrary.c
         testStaticLibrary.h
     CMakeLists.txt
```

Az egyes **CMakeLists.txt** állományok tartalma a következő:

6.4.25. forráskód: CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)

SET(CMAKE_C_COMPILER gcc)

PROJECT(cmaketest C)

SET(PACKAGE_NAME "cmaketest")
SET(MAJOR_VERSION "0")
SET(MINOR_VERSION "0")
SET(PATCH_VERSION "1")

SET(PACKAGE_VERSION    ${MAJOR_VERSION}.${MINOR_VERSION}.${PATCH_VERSION})

SUBDIRS(srcsl srcdl srcsa srcda srcsda)

SET(CMAKE_VERBOSE_MAKEFILE on)
```

A gyökérkönyvtárban található **CMakeLists.txt** állomány szerepe hasonló az autotools **Makefile.am**-jéhez, azaz elsősorban azokat az alkönyvtárakat sorolja fel, amelyekben további **CMakeLists.txt** állományok találhatók, és amelyeket a fordítás során fel kell dolgozni.

Az első utasításban a minimális cmake verziót adjuk meg, amellyel a konfigurációs állomány használandó. A cmake 2.6 és 2.8 között számos különbség van, esetünkben ragaszkodunk a legalább 2.8-as verziószámú cmake-hez. A második utasítással a C fordítót adjuk meg explicit módon, ugyanis több különböző C fordító is lehet installálva. A haramdik utasítás a projekt definiálása, a **PROJECT** függvény első paramétere a projekt neve, második paramétere a projekt nyelve. C++ esetén a projekt nyelvét **CXX**-ként kell megadni. Ezt követően négy változó beállítása következik. Cmake-ben a változókhoz értéket a **SET** függvénnyel rendelhetünk, amelynek első paramétere a változónév, második paramétere pedig a változó értéke. A négy változó a csomag nevét és verziószámát hordozza. A következő utasításban a **PACKAGE_VERSION** változó értékeként összerakjuk a fő-, al- és patch-verziószámokat. Ezt követően a **SUBDIRS** függvény paramétereiként felsoroljuk a további feldolgozandó könyvtárakat. Utolsó utasításként bekapcsoljuk a beszédes Makefile-ok funkciót, azaz a **CMAKE_lstinlineOSE_MAKEFILE** változó értékét **on**-ra állítjuk.

6.4.26. forráskód: srcsl/CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)

SET(PACKAGE_NAME testStaticLibrary)
SET(MAJOR_VERSION 0)
SET(MINOR_VERSION 0)
SET(PATCH_VERSION 2)
SET(PACKAGE_VERSION    ${MAJOR_VERSION}.${MINOR_VERSION}.${PATCH_VERSION})

SET(CMAKE_C_COMPILER gcc)

PROJECT(testStaticLibrary C)

AUX_SOURCE_DIRECTORY(. SRCVAR)

ADD_LIBRARY(${PACKAGE_NAME} ${SRCVAR})

INCLUDE_DIRECTORIES(.)

SET_TARGET_PROPERTIES(${PACKAGE_NAME} PROPERTIES LINKER_LANGUAGE C)

SET(CMAKE_VERBOSE_MAKEFILE on)
```

A statikus könyvtár **CMakeLists.txt** állománya sokban hasonlít a gyökérkönyvtárban létrehozott konfigurációs állomány tartalmához. A cmake minimális verziószámának megadásával kezdődik, majd definálju a c nyelvű projektet, ezt követően beállítjuk a csomag nevét és verzióját tartalmazó változókat, és a fordítót. Az **AUX_SOURCE_DIRECTOR** függvény két paraméterrel rendelkezik. Az első paramétere egy könyvtárnév, a második egy változó neve. Összegyűjti az első paraméterként megadott könyvtárban található a projekt nyelvének megfelelő forrásfájlok neveit (esetünkben .c forrásfájlokat), és berakja nevüket az **SRCVAR** változóba. Az **SRCVAR** változó tartalmazza tehát azokat a forrásfájlokat, amelyeket le akarunk fordítani. Az **ADD_LIBRARY** függvénnyel adhatunk hozzá library target-et a projekthez. A függvény első paramétere a könyvtár neve, ami esetünkben a csomagnév, további paraméterei pedig a könyvtárhoz lefordítandó forrásfájlok nevei, amelyet az **SRC_VAR** változóba már összegyűjtöttünk. Az **INCLUDE_DIRECTORIES** függvényhívás paraméterei azok a könyvtárak, amelyek -I kapcsolóval bekerülnek a fordítási sorba. Ezt követően tulajdonságokat beállítjuk a **PUBLIC_HEADER** tulajdonságot a **PACKAGE_NAME** változó értéke targethez. A **SET_PROPERTY** paraméterezése a következő: az első paramétere annak az eszköznek a típusa, amely valamely tulajdonságát be szeretnénk állítani. Második paramétere az eszköz konkrét neve, ezt követi a **PROPERTY** kulcsszó, majd a beállítandó tulajdonság neve, és ezt követően a tulajdonság értéke. Esetünkben a **PUBLIC_HEADER** tulajdonság kapja meg a **testStaticLibrary.h** értéket, azaz a könyvtárhoz tartozó publikus header fájlt. Ez a fájl bele kell, hogy kerüljön az elkészülő csomagba ahhoz, hogy a könyvtárat használni lehessen. Az **INSTALL** függvénnyel állítjuk be az installálás paramétereit. Első paramétere az eszközök neve, amelyek installálását be szeretnénk állítani, esetünkben **TARGETS**, ezt követik a beállítandó target-ek nevei, esetünkben ez a könyvtárnév, amely neve a csomag név változóban érhető el. Ezt követően az egyes target típusokhoz állíthatjuk be **TARGET_TIPUS DESTINATION konyvtarnev** módon, hogy az installálás fő könyvtárához relatívan hová kerüljenek. A Linux rendszerek konvenciójának megfelelően a könyvtárak az installálás **lib** alkönyvtárába, a header-ök az **include** alkönyvtárba, míg az archívumok a szintén a **lib** alkönyvtárba kerülnek. Utolsó utasításként a **Makefile**-okat beszédesre állítjuk.

6.4.27. forráskód: srcdl/CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)

SET(PACKAGE_NAME testDynamicLibrary)
SET(MAJOR_VERSION 0)
SET(MINOR_VERSION 0)
SET(PATCH_VERSION 1)
SET(PACKAGE_VERSION    ${MAJOR_VERSION}.${MINOR_VERSION}.${PATCH_VERSION})

SET(CMAKE_C_COMPILER gcc)

PROJECT(testDynamicLibrary C)

AUX_SOURCE_DIRECTORY(. SRCVAR)

ADD_LIBRARY(${PACKAGE_NAME} ${SRCVAR})

INCLUDE_DIRECTORIES(.)

SET_TARGET_PROPERTIES(${PACKAGE_NAME} PROPERTIES LINKER_LANGUAGE C)

SET(CMAKE_VERBOSE_MAKEFILE on)
```

A dinamikus könyvtárhoz tartozó konfigurációs állomány minimálisan tér el a statikus könyvtárétól. A "statikus" szó értelemszerű "dinamikusra" történő átírása mellett az **ADD_LIBRARY** függvényben a **SHARED** kulcsszóval specifikáljuk, hogy dinamikus könyvtárat hozunk létre, minden más utasítás és célja megegyezik a statikus könyvtárral.

6.4.28. forráskód: srcsa/CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)

SET(PACKAGE_NAME mainStatic)
SET(MAJOR_VERSION 0)
SET(MINOR_VERSION 0)
SET(PATCH_VERSION 1)
SET(PACKAGE_VERSION    ${MAJOR_VERSION}.${MINOR_VERSION}.${PATCH_VERSION})

#SET(CMAKE_C_COMPILER gcc)

PROJECT(${PACKAGE_NAME} C)

LINK_DIRECTORIES(../srcsl)

INCLUDE_DIRECTORIES(. ../srcsl)

AUX_SOURCE_DIRECTORY(. SRCVAR)
ADD_EXECUTABLE(${PACKAGE_NAME} ${SRCVAR})

TARGET_LINK_LIBRARIES(${PACKAGE_NAME} testStaticLibrary)

SET_TARGET_PROPERTIES(${PACKAGE_NAME} PROPERTIES LINKER_LANGUAGE C)

SET(CMAKE_VERBOSE_MAKEFILE on)
```

A statikusan linkelendő alkalmazás **CMakeLists.txt** állománya is hasonlóan épül fel a könyvtárakéhoz. A különbség a **LINK_DIRECTORIES** függvény, amely paraméterei a fordítási sorba -L kapcsolóval kerülnek be, azaz itt keresi a fordító a -l kapcsolóval megadott könyvtárakat. Az **INCLUDE_DIRECTORIES** függvény parméterei közé bekerül most a statikus könyvtár elérési útja is, hiszen itt találjuk azt a header-t, amellyel a statikus könyvtárat használni tudjuk. A futtatható állomány target-eket az **ADD_EXECUTABLE** függvénnyel adhatjuk meg, amely paraméterezése azonos az **ADD_LIBRARY** függvény paraméterezésével. Ezt követően a **TARGET_LINK_LIBRARIES** függvénnyel adhatjuk meg azokat a könyvtárakat, amelyek az első paraméterként megadott target linkelési utasításába a -l kapcsolóval kerülnek be. Mivel a target most csak egy futtatható állományból áll, az **INSTALL** függvényben csak a futtatható azaz **RUNTIME** típusú target-ekhez kell megadnunk az installálás célkönyvtárát.

6.4.29. forráskód: srcda/CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)

SET(PACKAGE_NAME mainDynamic)
SET(MAJOR_VERSION 0)
SET(MINOR_VERSION 0)
SET(PATCH_VERSION 2)
SET(PACKAGE_VERSION    ${MAJOR_VERSION}.${MINOR_VERSION}.${PATCH_VERSION})

SET(CMAKE_C_FLAGS "-fopenmp -Wall -Wextra -fPIC")
SET(CMAKE_C_FLAGS_DEBUG "-g -O0")
SET(CMAKE_C_FLAGS_RELEASE "-O2")

PROJECT(${PACKAGE_NAME} C)

LINK_DIRECTORIES(../srcdl)

INCLUDE_DIRECTORIES(. ../srcdl)

AUX_SOURCE_DIRECTORY(. SRCVAR)
ADD_EXECUTABLE(${PACKAGE_NAME} ${SRCVAR})

TARGET_LINK_LIBRARIES(${PACKAGE_NAME} testDynamicLibrary)
```

```
SET_TARGET_PROPERTIES(${PACKAGE_NAME} PROPERTIES LINKER_LANGUAGE C)

SET(CMAKE_VERBOSE_MAKEFILE on)
```

A dinamikus alkalmazás szerkezete teljesen azonos, a "statikus" és "dinamikus" szavak értelemszerű cseréjével, a statikus és dinamikus linkelés közötti különbséget a cmake eszközök teljesen elfedik.

6.4.30. forráskód: srcsda/CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)

SET(PACKAGE_NAME mainStaticAndDynamic)
SET(MAJOR_VERSION 0)
SET(MINOR_VERSION 0)
SET(PATCH_VERSION 2)
SET(PACKAGE_VERSION    ${MAJOR_VERSION}.${MINOR_VERSION}.${PATCH_VERSION})

SET(CMAKE_C_COMPILER gcc)

PROJECT(${PACKAGE_NAME} C)

LINK_DIRECTORIES(../srcsl ../srcdl)

INCLUDE_DIRECTORIES(. ../srcsl ../srcdl)

AUX_SOURCE_DIRECTORY(. SRCVAR)
ADD_EXECUTABLE(${PACKAGE_NAME} ${SRCVAR})

TARGET_LINK_LIBRARIES(${PACKAGE_NAME} testStaticLibrary testDynamicLibrary)

SET_TARGET_PROPERTIES(${PACKAGE_NAME} PROPERTIES LINKER_LANGUAGE C)

SET(CMAKE_VERBOSE_MAKEFILE on)
```

A statikusan és dinamikusan linkelt alkalmazás konfigkurációs állományának szerkezete ugyanaz, mint az előző két alkalmazásé, a különbség, hogy a linkelendő library-k könyvtárai és az include könyvtárak között felsoroljuk mind a statikus, mind a dinamikus könyvtár elérési útját, valamint a linkelendő library-k listája most mind a statikus, mind a dinamikus könyvtárat tartalmazza.

Lássuk, hogyan használhatjuk a konfigurációs állományainkat.

A **cmake** program első parancssori argumentuma azon könyvtár elérési útja, amely a konfigurálandó **CMakeLists.txt** állományt tartalmazza. A **cmaketest** gyökérkönyvtárban állva tehát ha kiadjuk a **cmake .** utasítást, a gyökérkönyvtárban található **CMakeLists.txt** alapján elkészül a GNU szabványú **Makefile**, valamint mivel a gyökérkönyvtár konfigurációs állományában további könyvtárakat adtunk meg a **SUBDIRS** függvény paramétereként, azokban is végbemegy a konfiguráció, azaz minden könyvtárban előáll egy **Makefile** állomány. A gyökérkönyvtárban kiadva most a **make** utasítást, a könyvtáraink és alkalmazásaink lefordulnak és linkelődnek.

A **cmake** **-h** utasítást kiadva áttekinthetjük, milyen opciói vannak a **cmake** alkalmazásnak. Az áttekintés utolsó szekciója azt sorolja fel, milyen IDE-khez készíthetünk projekt fájlokat. Linux rendszeren a

```
  Unix Makefiles               = Generates standard UNIX makefiles.
  CodeBlocks - Unix Makefiles = Generates CodeBlocks project files.
  Eclipse CDT4 - Unix Makefiles
                               = Generates Eclipse CDT 4.0 project files.
  KDevelop3                    = Generates KDevelop 3 project files.
  KDevelop3 - Unix Makefiles  = Generates KDevelop 3 project files.
```

generátorok közül válogathatunk, azaz például KDevelop3 fejlesztőkörnyezethez létrehozhatunk projekt fájlokat a

```
cmake -G "KDevelop3" .
```

utasítással.

6.4.2. Fordítási opciók

A **CMakeLists.txt** fájlokban számos utasítás és vezérlési szerkezet áll rendelkezésünkre, hogy a konfigurációt megfelelően végrehajthassuk. A cmake paraméterezése parancssoron keresztül történik, a **cmake** utasítást követően **-D** kapcsolóval egybeírva tetszőleges változónevet megadhatunk és annak az **=** operátorral tetszőleges értéket be is állíthatunk. Ezt követően a **CMakeLists.txt** állományok úgy tekinthetők, mint egyfajta imperatív szkript, amelyekben a parancssorban megadott változók használhatóak. Ha nem adunk meg egy változónevet és mégis használjuk a szkriptben, akkor annak értéke üres lesz.

Egészítsük ki most konfigurációs állományainkat úgy, hogy a debug/release konfigurációt a **cmake** utasítás kapcsolóival szabályozni tudjuk. Ehhez minden forrást tartalmazó könyvtár **CMakeLists.txt** állományát egészítsük ki a következő sorokkal:

```
SET(CMAKE_C_FLAGS_DEBUG "-g -O0")
SET(CMAKE_C_FLAGS_RELEASE "-O3")
```

A **CMAKE_C_FLAGS_DEBUG** egy speciális változó, amely a debug fordítás kapcsolóit tartalmazza, míg a **CMAKE_C_FLAGS_RELEASE** változó a release fordítás kapcsolóit. Az egyes fordítási módokat a **cmake** parancsban definálható **CMAKE_BUILD_TYPE** változó **debug** vagy **release** értékével állíthatjuk be. Ha a konfigurációs parancsban definálunk változókat, azokat értékükkel együtt a -D kapcsolóval egybeírva kell megadni, azaz debug fordításhoz az alábbi módon kell konfigurálnunk a projektünket:

```
cmake -DCMAKE_BUILD_TYPE=debug .
```

További, bonyolultabb opciók is megadhatók parancssori változók definiálásával és a **CMakeLists.txt** állományokban elhelyezhető **if** szerkezetek segítségével: CMake-ben az elágaztató utasítások szerkezete

```
if(kifejezes)
  [utasitasok]...
[elseif(kifejezes2)
  [utasitasok]...]...
[else(kifejezes)
  [utasitasok]...]
endif(kifejezes)
```

A fenti szintaktikában a **kifejezes** mindhárom előfordulása ugyanazt a kifejezést jelenti. A kifejezés lehet egy egyszerű változó is. Ebben a kifejezés hamis, ha a változó értéke *üres*, **0**, **N**, **NO**, **OFF**, **FALSE** , **NOTFOUND** vagy <valtozonev>-**NOTFOUND**. Minden más esetben a kifejezés igaz lesz. Lássuk, hogyan állíthatjuk be parancssorból a debug/release fordítást a beépített **CMAKE_C_FLAGS_DEBUG** és **CMAKE_C_FLAGS_RELEASE** változók nélkül. Legyen a parancssori argumentum neve **ENABLE_DEBUG** . Ekkor az alábbi feltételes szerekezetet elhelyezve a **CMakeLists.txt** állományokban, éppen a kívánt mű ködést kapjuk:

```
IF ( ENABLE_DEBUG )
  SET(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -g -O0")
ELSE (ENABLE_DEBUG )
  SET(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -O3")
ENDIF( ENABLE_DEBUG )
```

A kódot elhelyezve azokban a **CMakeLists.txt** állományokban, amelyekben opcionálissá szeretnénk tenni a debug fordítást, a **cmake** parancsban a **-DENABLE_DEBUG=on** utasítással kapcsolhatjuk be azt.

6.4. Feladat: Adjon hozzá a **CMakeLists.txt** fájlokhoz olyan opciókat, amelyek a warningok és az opcionálisan linkelendő dinamikus könyvtár használatát kapcsolják be!

6.4. Megoldás: A warning-ok bekapcsolása a debug/release fordításhoz hasonlóan történik:

```
IF ( ENABLE_WARNINGS )
  SET(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -Wall -Wextra")
ENDIF( ENABLE_WARNINGS )
```

A **mainOptional** alkalmazásba opcionálisan belefordítandó dinamikus könyvtárbeli függvényhívást az **ENABLE_OPTIONAL** változóval kapcsolhatjuk be:

```
IF ( ENABLE_OPTIONAL )
  SET(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -DUSE_DYNAMIC_LIBRARY")
ENDIF ( ENABLE_OPTIONAL )
```

A fenti feltételes szerkezeteket természetesen azon **CMakeLists.txt** állományokba kell beírnunk, amelyekben szeretnénk, hogy a kapcsolók hatása érvényesüljön. Ha a gyökérkönyvtárban lévő konfigurációs állományba írjuk be, és nem szerepel az alkönyvtárak konfigurációs állományaiban a **CMAKE_C_FLAGS** teljes felülírása értékadással, akkor a megfelelő kapcsolók minden alkönyvtár forrásfájljainak fordítási parancsába bekerülnek. Lássuk most, hogyan kapcsolható be a dinamikus könyvtár használata a **config.h** header-ön keresztül! Az **automake**-es megoldáshoz hasonlóan, itt is létre kell hoznunk egy template-et a konfigurációs állományhoz, azaz az alábbi **config.h.in** állományt, és include-oljuk a **config.h**-t a **mainOptional.c** forráskódjának elsősorában.

6.4.31. forráskód: config.h.in

```
@DEFINE_DIRECTIVE@
```

A konfigurációs állományok konkretizálását a **CONFIGURE_FILE** utasítással végezhetjük el. Első kötelező paramétere egy template fájlnév (esetünkben a fenti **configure.h.in** állomány), második paramétere pedig a template konfigurációs állományból előálló állomány neve. A megfelelő **CMakeLists.txt** fájlban (például abban, amely az **srcoa** könyvtárban van), az alábbi kódrészletet kell elhelyezni:

```
CONFIGURE_FILE(config.h.in config.h)
```

Ezt parancssorban a

```
cmake -DUSE_DYNAMIC_LIBRARY=yes .
```

utasítással kapcsolhatjuk be a konfigurációs állomány megfelelő tartalommal való feltöltését.

6.4.32. forráskód: srcoa/CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)

SET(PACKAGE_NAME mainOptional)
SET(MAJOR_VERSION 0)
```

```
SET(MINOR_VERSION 0)
SET(PATCH_VERSION 2)
SET(PACKAGE_VERSION    ${MAJOR_VERSION}.${MINOR_VERSION}.${PATCH_VERSION})

SET(CMAKE_C_FLAGS "-fPIC")

PROJECT(${PACKAGE_NAME} C)

LINK_DIRECTORIES(../srcdl)

INCLUDE_DIRECTORIES(. ../srcdl)

IF ( USE_DYNAMIC_LIBRARY )
    SET(DEFINE_DIRECTIVE "#define USE_DYNAMIC_LIBRARY")
ENDIF ( USE_DYNAMIC_LIBRARY )

CONFIGURE_FILE(config.h.in config.h)

AUX_SOURCE_DIRECTORY(. SRCVAR)
ADD_EXECUTABLE(${PACKAGE_NAME} ${SRCVAR})

TARGET_LINK_LIBRARIES(${PACKAGE_NAME} testDynamicLibrary)

SET_TARGET_PROPERTIES(${PACKAGE_NAME} PROPERTIES LINKER_LANGUAGE C)

SET(CMAKE_VERBOSE_MAKEFILE on)
```

A kapcsoló egy **IF**-ben van lekezelve, amennyiben parancssori kapcsoló igaz értéket hordoz, akkor a **DEFINE_DIRECTIVE** értékül kapja a tényleges megfelelő ***#define*** direktívát. Ezt követően a **CONFIGURE_FILE** utasítás megnyitja az első paraméterként kapott fájlt és abban minden változó helyére beírja a Cmake változó nevét, esetünkben a **DEFINE_DIRECTIVE** változó tartalmát, és a megváltozott fájlt kiírja a második paraméterként.

6.4.3. Külső könyvtárak használata

Adjuk hozzá build környezetünkhoz a korábban is használt **mainPNG.c** forrásfájlt és készítsünk hozzá egy egyszerű **CMakeLists.txt** állományt.

A CMake egyik nagy előnye, hogy a külső csomagok használatával járó gondokat (installált csomag megtalálása, flag-ek beállítása, stb.), magas szinten, interoperábilis módon oldja meg. A koncepció hasonló a **pkg−config** működési elvéhez, azonban cmake esetén a leíró állományokról nem a forráskód csomagok gondoskodnak, hanem a CMake készítői.

A makró, amelyet használnunk kell, a **FIND_PACKAGE**. Első paramétere a csomag neve, amelyet szeretnénk megtalálni, második paramétere pedig egy opcionális **REQUIRED** kulcsszó. Működését tekintve ha a CMake megtalálja a <PACKAGE> nevű csomagot, akkor az alábbi változókat definiálja:

- **<PACKAGE>_FOUND** - azaz csomag installálva van a rendszerre,
- **<PACKAGE>_INCLUDE_DIRS** vagy **<PACKAGE>_INCLUDES** - tartalma a csomag használatához szükséges header fájlok elérési útja,
- **<PACKAGE>_LIBRARIES** vagy **<PACKAGE>_LIBS** - tartalma a csomag library-k elérési útja és neve,
- **<PACKAGE>_DEFINITIONS** - esetleges további definíciók.

Egy megtalált csomag esetén tehát létrejönnek a fenti változók, amelyeket felhasználhatunk a csomag jelenlétének ellenőrzésére, és arra, hogy beállítsuk a megfelelő fordítási/linkelési kapcsolókat.

A csomagok megtalálásának mechanizmusa a package-config-gal ellentétben itt nem .pc fájlokkal,

hanem úgynevezett modulokkal operál. Minden modul egy-egy csomaghoz tartozik, (például a **libpng** csomaghoz tartozó modul neve **PNG**, és minden modulhoz tartozik egy fájl, mely nevének szerkezete: **Find<PACKAGE>.cmake**. Ezen fájlban lévő utasítások találják meg a tényleges header fájlokat és könyvtárakat a rendszeren, és végzik a korábban említett négy változó értékének beállítását. A leggyakrabban használt külső csomagokhoz a cmake fejlesztői előre megírták ezeket a modulokat, és azokat a **CMAKE_MODULE_PATH** környezeti változóban felsorolt elérési utakon keresi a cmake.

A **libpng** csomag használatához tehát a **mainPNG.c** alkalmazás konfigurációs fájljában az alábbi utasítást kell elhelyeznünk:

```
FIND_PACKAGE(PNG)
INCLUDE_DIRECTORIES(${PNG_INCLUDE_DIR})
TARGET_LINK_LIBRARIES(${PACKAGE_NAME} ${PNG_LIBRARIES})
```

konfigurálva a **cmake** paranccsal, látható, hogy a cmake keresi és meg is találja a **libpng** könyvtárat, valamint annak előfeltételét, a **zlib** könyvtárat is, majd a **make** parancs kiadása után a fordítási és linkelési parancsban jól látható a megfelelő **–I** és **–l** kapcsolók megjelenése. Jegyezzük meg, hogy itt a linkelendő könyvtárak teljes elérési útja jelenik meg a **–l** kapcsolók mögött, így a **–L** kapcsolókra nincs szükség.

6.4.4. Disztribúció létrehozása

Lássuk, hogyan hozhatunk létre disztribútolható csomagokat a **cmake** segítségével! Több különböző típusú csomag létrehozására van lehetőségünk a **cpack** programmal, azonban előbb néhány, a disztribúcióval kapcsolatos adatot meg kell adnunk változókban a **CMakeLists.txt** állományokban. Helyezzük el a következő sorokat minden forrásfájlt tartalmazó könyvtár **CMakeLists.txt** állományának végén, megfelelő változó értékekkel:

```
SET(CPACK_PACKAGE_DESCRIPTION_SUMMARY "testStaticLibrary")
SET(CPACK_PACKAGE_VENDOR "Gyorgy Kovacs")
SET(CPACK_PACKAGE_CONTACT "gyuriofkovacs@gmail.com")
SET(CPACK_DEBIAN_PACKAGE_MAINTAINER    "gyuriofkovacs@gmail.com")
SET(CPACK_PACKAGE_DESCRIPTION_SUMMARY "cmaketest$")
SET(CPACK_PACKAGE_VERSION_MAJOR "${MAJOR_VERSION}")
SET(CPACK_PACKAGE_VERSION_MINOR "${MINOR_VERSION}")
SET(CPACK_PACKAGE_VERSION_PATCH "${PATCH_VERSION}")

INCLUDE(CPack)
```

Mindemellett azt is meg kell határoznunk, hogy az egyes fájlok a tepelítés során hová kerüljenek. Linux környezetben a header fájlok általában egy **include** nevű könyvtárba kerülnek, a futtatható állományok valamely **bin** könyvtárba, míg a library-k egy **lib** nevű könyvtárba. Windows-on már más a helyzet, ugyanis a futtatható állományokat és a dinamikus könyvtárakat célszerű egymás mellé telepíteni. Ezen különbségek elfedésére és maguknak a célkönyvtáraknak a megadására használható az **INSTALL** makró:

```
INSTALL(TARGETS ${PACKAGE_NAME} RUNTIME DESTINATION bin LIBRARY DESTINATION lib ARCHIVE
    DESTINATION lib)
FILE(GLOB HEADERS "*.h")
INSTALL(FILES ${HEADERS} DESTINATION include)
```

Az első sor jelentése egyrészt, hogy a **${PACKAGE_NAME}** nevű target-et installálja **make install** hatására, másrészt ugyanazon makró hívással specifikáljuk, hogy a futásidőben használandó eszközök (alkalmazások/dinamikus könyvtárak) a **bin** könyvtárba kerüljenek, a library-k (dinamikus könyvtárak) a **lib** könyvtárba és az archivumok is a **lib** könyvtárba. Kicsit zavaró ugyan, de a fenti utasítás

elfedi a Windows/Linux környezetek különbözőségét, ugyanis linux-on csak az alkalmazások kerülnek a **RUNTIME** eszközök osztályába, a dinamikus könyvtárak pedig a **LIBRARY** kulcsszó után kijelölt helyre települnek, míg Windows-on mind az alkalmazások, mind a dinamikus könyvtárak a **RUNTIME** kulcsszó után szereplő célkönyvtárba kerülnek. A **FILE** makró első paramétere (**GLOB**) azt jelzi, hogy a forráskönyvtárban lévő összes alkönyvtárral is dolgozzon, miközben a második paraméterként megadott **HEADERS** változóba kigyűjti a harmadik paraméterként kapott reguláris kifejezésre illeszkedő fájlneveket. A harmadik sorban egy újabb **INSTALL** makró hívással a **FILES** kulcsszó használatával tetszőleges fájlokra megadhatjuk, hogy hová kerüljenek installás után. A header fájlok azért tartoznak az egyéb kategóriába, mert csak a fejlesztői csomagokhoz van rájuk szükség. Az **INSTALL** makrókban megadott könyvtárak relatív könyvtárak a **CMAKE_INSTALL_PREFIX** beépített változó értékéhez képest, amely alapértelmezésben Linux rendszereken a /**usr** értéket tartalmazza. Windows rendszeren létrehozott grafikus NSIS installer esetén az felhasználó által kiválasztott telepítési célkönyvtárhoz képest lesznek relatívak.

A statikusan és dinamikusan linkelt alkalmazás **CMakeLists.txt** állománya tehát a fenti bővítések után az alábbi módon fog kinézni:

6.4.33. forráskód: srcsda/CMakeLists.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)

SET(PACKAGE_NAME mainStaticAndDynamic)
SET(MAJOR_VERSION 0)
SET(MINOR_VERSION 0)
SET(PATCH_VERSION 2)
SET(PACKAGE_VERSION    ${MAJOR_VERSION}.${MINOR_VERSION}.${PATCH_VERSION})

SET(CMAKE_C_COMPILER gcc)
SET(CMAKE_C_FLAGS_DEBUG "-g -O0")
SET(CMAKE_C_FLAGS_RELEASE "-O2")

PROJECT(${PACKAGE_NAME} C)

LINK_DIRECTORIES(../srcsl ../srcdl)

INCLUDE_DIRECTORIES(. ../srcsl ../srcdl)

AUX_SOURCE_DIRECTORY(. SRCVAR)
ADD_EXECUTABLE(${PACKAGE_NAME} ${SRCVAR})

TARGET_LINK_LIBRARIES(${PACKAGE_NAME} testStaticLibrary testDynamicLibrary)

SET_TARGET_PROPERTIES(${PACKAGE_NAME} PROPERTIES LINKER_LANGUAGE C)

INSTALL(TARGETS ${PACKAGE_NAME} RUNTIME DESTINATION bin LIBRARY DESTINATION lib ARCHIVE
    DESTINATION lib)
FILE(GLOB HEADERS "*.h")
INSTALL(FILES ${HEADERS} DESTINATION include)

SET(CPACK_PACKAGE_DESCRIPTION_SUMMARY "testApp")
SET(CPACK_PACKAGE_VENDOR "Gyorgy Kovacs")
SET(CPACK_PACKAGE_CONTACT "gyuriofkovacs@gmail.com")
SET(CPACK_DEBIAN_PACKAGE_MAINTAINER    "gyuriofkovacs@gmail.com")
SET(CPACK_PACKAGE_DESCRIPTION_SUMMARYY "openip drscreen application")
SET(CPACK_PACKAGE_VERSION_MAJOR "${MAJOR_VERSION}")
SET(CPACK_PACKAGE_VERSION_MINOR "${MINOR_VERSION}")
SET(CPACK_PACKAGE_VERSION_PATCH "${PATCH_VERSION}")

INCLUDE(CPack)

SET(CMAKE_VERBOSE_MAKEFILE on)
```

A fenti sorok **CMakeLists.txt** fájlokba történő beírása után tetszőleges alprojekthez, vagy akár az egész projekthez készíthetünk disztribúcióra kész csomagokat az alábbi módon. A **cpack** --**help** utasítással nézhetjük meg, milyen disztribúció generátorok érhetőek el:

```
DEB                            = Debian packages
NSIS                           = Null Soft Installer
RPM                            = RPM packages
STGZ                           = Self extracting Tar GZip compression
TBZ2                           = Tar BZip2 compression
TGZ                            = Tar GZip compression
TZ                             = Tar Compress compression
ZIP                            = ZIP file format
```

Disztribúciót ezt követően a **cpack** -**G generator_azonosito .** paranccsal hozhatunk létre. A **STGZ**, **TBZ2**, **TGZ**, **TZ**, **ZIP** disztribúciók egyszerűen tömörítik a könyvtárakat, a **DEB** szabványos Debian csomagot, a **RPM** Red Hat Linux-ra épülő disztribúciók által használt csomagot, az **NSIS** pedig grafikus Windows telepítőt hoz létre. Természetesen az **NSIS** generátort csak windows rendszeren használhatjuk, használata előtt fel kell telepítenünk az ingyenes **NSIS** szoftvert, amely a következő linkről tölthető le: http://nsis.sourceforge.net/Main_Page.

Jegyezzük meg, hogy a CMake beépített módon csak a bináris (alkalmazások + dinamikus könyvtárak) és bináris fejlesztői (alkalmazások + dinamikus, statikus könyvtárak + header-ök) csomagok létrehozását támogatja. Szemben az **automake** eszközökkel, amelyek forráscsomag disztribúciót hoztak létre build script-tel. Ahhoz, hogy CMake-kel készítsünk forrás disztribúciót, a forráskódokat és a **CMakeLists.txt** fájlokat is meg kell adnunk megfelelő **INSTALL** makró hívásban.

A teljesség kedvéért lássuk, hogyan hozhatjuk létre a **FindTESTLIBS.cmake** modult, amelyet disztribútolva segíthetjük könyvtáraink megtalálását és felhasználását más rendszereken. A modul neve tehát rögzített, ha a csomagunkat testlibs-nek hívjuk, akkor a korábban említett nevű modult kell létrehoznunk, tartalmát tekintve pedig négy változót kell definiálnunk:

- **TESTLIBS_FOUND** abban az esetben, ha a rendszeren fenn van a könyvtárunk,
- **TESTLIBS_INCLUDE_DIR** ha a rendszerre telepítve van a könyvtárunk, akkor a header fájlok elérési útjai,
- **TESTLIBS_LIBRARIES** ha a rendszerre telepítve van a könyvtárunk, akkor a library-k nevei.

A megfelelő modul az alábbi lesz:

6.4.34. forráskód: FindTESTLIBS.cmake

```
# - Find TESTLIBS library
# Find the native TESTLIBS includes and library
# This module defines
#  TESTLIBS_INCLUDE_DIR, where to find testStaticLibrary.h, etc.
#  TESTLIBS_LIBRARIES, libraries to link against to use TESTLIBS.
#  TESTLIBS_FOUND, If false, do not try to use TESTLIBS.

#=============================================================================
# Copyright 2002-2009 Kitware, Inc.
#
# Distributed under the OSI-approved BSD License (the "License");
# see accompanying file Copyright.txt for details.
#
# This software is distributed WITHOUT ANY WARRANTY; without even the
# implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
# See the License for more information.
#=============================================================================
# (To distributed this file outside of CMake, substitute the full
#  License text for the above reference.)
```

```
FIND_PATH(TESTLIBS_INCLUDE_DIR testStaticLibrary.h)

SET(TESTLIBS_NAMES ${TESTLIBS_NAMES} testStaticLibrary testDynamicLibrary)
FIND_LIBRARY(TESTLIBS_LIBRARIES NAMES ${TESTLIBS_NAMES} )

# handle the QUIETLY and REQUIRED arguments and set TESTLIBS_FOUND to TRUE if
# all listed variables are TRUE
INCLUDE(FindPackageHandleStandardArgs)
FIND_PACKAGE_HANDLE_STANDARD_ARGS(TESTLIBS  DEFAULT_MSG  TESTLIBS_LIBRARIES
    TESTLIBS_INCLUDE_DIR)
```

Az első makró hívással a **testStaticLibrary.h** header elérési útját keressük meg, a második makróval beállítjuk a **TESTLIBS_NAMES** temporális változó értékét azon library-ke nevére, amelyeket szükségesek ahhoz, hogy a csomaggal fordítani tudjunk, majd megkeressük ezeket a library-ket a **FIND_LIBRARY** makró felhasználásával. Ezt követően a beépített **FindPackageHandleStandardArgs** csomag include-olása után a hasonló nevű makró létrehozza és beállítja az első paramétereként kapott csomagnévhez tartozó **TESTLIBS_FOUND** változó értékét annak megfelelően, hogy a harmadik és negyedik paraméterei értelmes library-ket és include könyvtárakat határoznak meg. Második paraméter a konfiguráció során a konzolon megjelenő, beépített, standard üzenet használatára vonatkozik.