6.3. Autotools

6.3.1. Egyszerű autotools környezet kialakítása

A korábban bemutatott GNU Makefile-ok tökéletes eszközt nyújtanak a források menedzselésére, azonban létrehozásuk még néhány fájlból álló könyvtár esetén is nehézkes, későbbi módosítása pedig

igazi agyrém. Az autotools eszközök arra használhatóak fel, hogy néhány magasabb szintű konfigurációs állomány segítségével és az autotools eszközökkel GNU Makefile-okat generáljunk, amelyet a korábban megismert make eszközzel használhatunk fordításra.

Az automake eszközrendszer nem a legegyszerűbb, de jelenleg talán a legelterjedtebb módja alkalmazások és könyvtárak forrásainak menedzselésére, Linux rendszereken a leggyakrabban használt eszközrendszer. A rendszer összetett, több konfigurációs állományból és autotools eszközhívással juthatunk el a fordításig, azaz a Makefile-okig létrejöttéig.

Használjuk a korábban létrehozott kódjainkat, azaz a dinamikus és statikus könyvtárat és a négy alkalmazást. Hozzunk létre egy könyvtárat **testLibraries** néven, benne a **srcsl**, **srcdl**, **srcsa**, **srdda**, **srcsda** és **srcoa** könyvtárakat, amelyekbe másoljuk be a korábban használt forrásfájljainkat! Hozzunk létre továbbá egy **m4** nevű könyvtárat a **testLibraries** gyökérkönyvtárban, amelyet segédfájlok tárolására használ majd az autotools rendszer.

6.3.6. forráskód: A könyvtárszerkezet autotools használatához

```
testLibraries
     m4
     srcda
         mainDynamic.c
     srcdl
         testDynamicLibrary.c
         testDynamicLibrary.h
     srcoa
         mainOptional.c
     srcsa
         mainStatic.c
     srcsda
         mainStaticAndDynamic.c
     srcsl
         testStaticLibrary.c
         testStaticLibrary.h
```

Hozzunk létre a **configure.ac** konfigurációs állományt a **testLibraries** gyökérkönyvtárban és **Makefile.am** nevű állományokat a gyökérkönyvtárban és minden alkönyvtárban!

6.3.7. forráskód: Könyvtárszerkezet konfigurációs állományokkal

```
testLibraries
     m4
     srcda
         Makefile.am
         mainDynamic.c
     srcdl
         Makefile.am
         testDynamicLibrary.c
         testDynamicLibrary.h
     srcoa
         Makefile.am
         mainOptiona.c
     srcsa
         Makefile.am
         mainStatic.c
     srcsda
         Makefile.am
         mainStaticAndDynamic.c
     srcsl
         Makefile.am
         testStaticLibrary.c
         testStaticLibrary.h
     Makefile.am
```

```
configure.ac
```

Töltsük fel most a konfigurációs állományokat az alábbi tartalmakkal!

6.3.8. forráskód: configure.ac

```
AC_INIT([testLibraries], [1.0], [gyuriofkovacs@gmail.com])
AC_CONFIG_MACRO_DIR([m4])
AM_INIT_AUTOMAKE([foreign -Wall -Werror])
LT_INIT
AC_PROG_CC
AC_PROG_RANLIB
AC_PROG_LIBTOOL

CFLAGS=""

AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile srcsl/Makefile srcsa/Makefile srcdl/Makefile srcda/Makefile
    srcsda/Makefile srcoa/Makefile])
AC_OUTPUT
```

6.3.9. forráskód: Makefile.am

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

6.3.10. forráskód: srcsl/Makefile.am

```
lib_LIBRARIES = libtestStaticLibrary.a
libtestStaticLibrary_a_SOURCES = testStaticLibrary.c
include_HEADERS = testStaticLibrary.h

AM_CFLAGS = -I.
```

6.3.11. forráskód: srcdl/Makefile.am

```
lib_LTLIBRARIES = libtestDynamicLibrary.la

libtestDynamicLibrary_la_SOURCES = testDynamicLibrary.c
include_HEADERS = testDynamicLibrary.h
AM_CFLAGS = -I.
```

6.3.12. forráskód: srcsa/Makefile.am

```
bin_PROGRAMS = mainStatic
mainStatic_SOURCES = mainStatic.c

AM_CFLAGS = -I. -I$(srcdir)/../srcsl
mainStatic_LDADD = ../srcsl/libtestStaticLibrary.a
```

6.3.13. forráskód: srcda/Makefile.am

```
bin_PROGRAMS = mainDynamic
mainDynamic_SOURCES = mainDynamic.c

AM_CFLAGS = -I. -I$(srcdir)/../srcdl
mainDynamic_LDADD = ../srcdl/libtestDynamicLibrary.la
```

6.3.14. forráskód: srcsda/Makefile.am

```
bin_PROGRAMS = mainStaticAndDynamic
mainStaticAndDynamic_SOURCES = mainStaticAndDynamic.c

AM_CFLAGS = -I. -I$(srcdir)/../srcdl -I$(srcdir)/../srcsl

mainStaticAndDynamic_LDADD = ../srcdl/libtestDynamicLibrary.la \
  ../srcsl/libtestStaticLibrary.a
```

6.3.15. forráskód: srcoa/Makefile.am

```
bin_PROGRAMS = mainOptional
mainOptional_SOURCES = mainOptional.c

AM_CFLAGS = -I. -I$(srcdir)/../srcdl
mainOptional_LDADD = ../srcdl/libtestDynamicLibrary.la
```

Az autotools rendszer feladat kettős: egyrészt menedzselhetjük vele saját fejlesztésű könyvtárunkat, másrészt olyan konfigurációs scripteket állíthatunk elő vele, amelyek más rendszereken, más architektúrájú és operációs rendszert használó számítógépeken beállítják a fordításhoz szükséges kapcsolókat. Autotools eszközökkel menedzselt könyvtárak első lépése a **configure** nevű script előállítása. Ez a script a rendszer ellenőrzést végzi, és előállítja a fordításhoz használható Makefile-okat. A **configure** script a Makefile-okat a Makefile.am konfigurációs fájlokban megadott információk alapján hozza létre. A **configure** script előállítása mindig a forráscsomagok készítőjének a feladata. Az előző fejezetben láthattuk, hogy a forráscsomagokhoz mindig biztosítani kell olyan állományokat, amelyek lehetővé teszik a forráskód halmaz lefordítását, installálását, esetleg tesztelését. Autotools esetén a **configure** script és a **Makefile.am** állományok szolgálják ezt a célt, azaz ha disztribúcióra kerül a sor, a forráskóddal együtt adnunk kell a **configure** script-et és a **Makefile.am** konfigurációs állományokat.

A fordítással kapcsolatos tényleges információkat, mint például azt, hogy egy target-hez mely forrásfájlok tartoznak, a **Makefile.am** fájlokban adjuk meg. A **configure** script a rendszer ellenőrzését végzi, és a rendszernek, illetve a felhasználó által megadott kapcsolóknak megfelelően előállítja a Makefile-okat a **Makefile.am**-ek segítségével. Hogy pontosan milyen eszközök, könyvtárak jelenlétét kell ellenőriznie, illetve milyen kapcsolókra hogyan kell reagálnia a **configure** script-nek azt a **configure.ac** konfigurációs állományban adhatjuk meg. Vegyük sorra a **configure.ac** tartalmát, és nézzük meg, melyik utasítás mit jelent esetünkben!

Az **AC_INIT** utasítással inicializáljuk a projektünket. Az első paramétere a könyvtár nevét adja meg, második paramétere a verziószámot, harmadik paramétere pedig egy email cím, ahová az esetleges hibajelentéseket küldhetik a könyvtár használói.

Az egész autotools rendszer scriptekből áll, amelyek főleg a konfigurációs állományokban elhelyezett sztringek átírásával, helyettesítésével, azaz makrók feldolgozásával állítják elő azokat a parancsokat és végső target neveket, amelyeket a fordítás során használnak. Az **m4** csomag a Linux rendszereken a legelterjedtebb makró processzor. Könyvtárszerkezetünk gyökerében azért hoztuk létre az **m4** könyvtárat, hogy az **m4** csomag ebben elhelyezhesse a létrehozott temporáris fájlokat. Azt, hogy az **m4** ezt a könyvtárat használhatja, az **AC_CONFIG_MACRO_DIR([m4])** utasítással állítjuk be.

A következő utasítás az **AM_INIT_AUTOMAKE([foreign])** azt specifikálja, hogy könyvtárunk nem lesz része a GNU rendszernek, amely nagyon szigorú disztribúciós szabályokkal rendelkezik (pl. kötelező manual page-ek, stb.), így egy teljes GNU által disztribútult csomaghoz képest kevesebb eszközzel is létrehozhatunk csomagot.

Amikor különböző rendszereken fordítunk GNU eszközökkel, a paraméterezés többnyire megegyezik, (a fordító többnyire ugyanolyan kapcsolókkal rendelkezik, a statikus könyvtárak ugyanúgy archívumba rendezett objekt fájlok), azonban a shared object fájlok különböznek. Linux rendszeren például láthattuk, hogy felhasználtuk a **–fPIC** fordító flag-et, positon-independent-code létrehozására, Windows-on erre nincs szükség, más rendszereknek pedig más további igényeik lehetnek. Hogy a shared object könyvtárak, azaz dinamikusan linkelhető könyvtárak létrehozása közötti különbséget elfedjék az autotools eszközei, kidolgozták az úgynevezett **libtool** eszközt. A **libtool** működése meghaladja ezen tárgy kereteit, azonban tudnunk kell, hogy ha dinamikus könyvtárakkal dolgozunk, a **libtool** használatát specifikálnunk kell a konfigurációs állományainkban. Az **LT_INIT** utasítással kérhetjük a **libtool** eszköz inicializálását.

A következő három utasítás mind a rendszerre telepített eszközöket keresi meg, és megfelelő változókban eltárolja, hogy az egyes eszközök közül, pl. az elérhető C fordítók, melyiket fogja használni. **AC_PROG_CC** a C fordító (C Compiler) jelenlétét vizsgálja, az **AC_PROG_RANLIB** ellenőrzést akkor kell kérnünk, ha valamilyen könyvtárat (akkor is, ha statikusat, akkor is, ha dinamikusat) szeretnénk létrehozni a fordítás során, az **AC_PROG_LIBTOOL** a korábban említett **libtool** jelenlétét és működését ellenőrzi.

Akárcsak a GNU Makefile-ok esetén, autotools használatánál is vannak speciális környezeti változók, ilyen például a **CFLAGS**. Ennek a környezeti változónak az értéke be kerül minden egyes fordítási parancsba. Alapértelmezésben ez tartalmazza a **–g –O2** kapcsolókat, ahol a **–g** azt jelzi, hogy a későbbiekben debuggert szeretnénk használni, az **–O2** pedig egy közepes optimalizálást specifikál. Az alapértelmezett kapcsolókat a **CFLAGS=""** utasítással töröljük.

A következő utasításban fel kell sorolnunk azokat a konfigurációs állományokat (esetünkben csak Makefile-okat), amelyeket a **configure** scriptnek elő kell majd állítania. Vigyázzunk, hogy minden felsorolt fájl esetén legyen a megfelelő helyen egy **Makefile.am**, aminek a tartalma alpján, a rendszernek megfelelően állnak majd elő a végső **Makefile**-k, esetünkben 7 ilyen **Makefile** van: a **AC_CONFIG_FILES ()** paramétereként láthatjuk őket.

Az **AC_OUTPUT** állítja elő végül a **AC_CONFIG_FILES**-ban felsorolt fájlokat.

Vegyük most sorra a **Makefile.am** fájlokat és tartalmukat.

A **testLibraries** gyökérkönyvtárban található **Makefile.am** egyedül azt adja meg, hogy mely további alkönyvtárakban vannak feldolgozandó **Makefile**-ok. Esetünkben ez az 6 könyvtár a statikus és dinamikus alkalmazáshoz, valamint a statikusan, dinamikusan és statikusan és dinamikusan linkelt alkalmazások forráskódját tartalmazó könyvtárak.

A statikus könyvtárhoz tartozó **Makefile.am** válotozók értékadásait tartalmazza és target specifikációkat tartalmaz. Autotools **Makefile.am**-ek esetén a konvenció a

```
where_PRIMARY= targets ...
```

változónév és értékadás forma. A **PRIMARY** rész adja meg, hogy a fordítás során a target-ek nevén milyen fájl jöjjön létre. A lehetőségek:

- **_PROGRAMS** esetén bináris futtatható állományok jönnek létre;
- **_LIBRARIES** esetén statikus könyvtárak;
- **_LTLIBRARIES** esetén libtool, azaz dinamikus könyvtárak;
- **_HEADERS** esetén nincs szükség fordításra, azt vezérelhetjük azonban, hogy a megfelelő header-ök installálása hová történjen;

- **_SCRIPTS** esetén scriptek jönnek létre;
- **_DATA** esetén pedig a felsorolt fájlok resource fájlok, fordítás nem történik, azonban azt vezérelhetjük, hogy ezen változókon keresztül, hogy hová insatllálódjanak.

A **where** rész az értékadás bal oldalán azt specifikálja, hogy installáláskor mely könyvtárba kerüljenek a létrejött target-ek. Ajánlott azonban a konvenciók használata: futtatható állományok a **bin** könyvtárba, míg statikus és dinamikus könyvtárak egyaránt a **lib** könyvtárba kerüljenek installálás után. Ennek megfelelően ha megnézzük a statikus könyvtárhoz tartozó **srcsl** /**Makefile.am** állományt, láthatjuk, hogy egyetlen target-et definiáltunk, amely **lib** könyvtárba kerül majd installáláskor, a neve pedig **libtestStaticLibrary . a**. Ezt követően az egyes target-ekhez tartozó források megadása következik, amelyet a

targetNev_SOURCES = forras_fajlok

formában rendelhetünk hozzá az egyes target-ekhez. Esetünkben ez egyetlen forrásfájl. Ügyeljünk arra, hogy a target nevekben szereplő '.' karaktereket, ha a target név a fentihez hasonlóan értékadás bal oldalán szerepel, minden esetben '_' karakterrel kell helyettesítenünk. Ahhoz, hogy egy statikus vagy dinamikus könyvtárat használni tudjunk, minden esetben szükségünk van header-ökre, így nem szabad elfelejtkeznünk arról, hogy az **include** könyvtárba kerülő header-öket is explicit módon megadjuk a **include_HEADERS =** változó beállításával. Egyetlen utasítás maradt, amelyben a fordító kapcsolóit állítjuk be: az **AM_CFLAGS** változó az aktuális Makefile-ból származó fordításokra van csak hatással, szemben a **CFLAGS** változóval, amely értéke a projekt fordítása során minden fordítási parancsba bekerül. Az **AM_CFLAGS** értéke esetünkben egyetlen –**I.** include könyvtár megadás, ugyanis emlékezzünk, hogy a forrásfájljainkban az **include** utasításokban a header-öket **<>**-ek között adtuk meg, ami azt jelenti, hogy a fordító csak a fordítási parancsban –**I** kapcsoló paramétereként megadott helyeken keresi a megfelelő header-öket.

A dinamikus könyvtárhoz tartozó **Makefile.am** szerkezete azonos, egyedül a target név és a target **PRIMARY**-je különbözik, a target név értelemszerűen más (libtool használata esetén a dinamikus könyvtárak kiterjesztése **. la**), a **PRIMARY**-t viszont **LTLIBRARIES**-re módosult, ugyanis dinamikus könyvtárat hozunk létre, a libtool használatával, ennek jele az **LT** a **LIBRARIES** előtt.

Lássuk a statikusan linkelt alkalmazáshoz készült **srcsa**/**Makefile.am**-et. Az előzőek fényében a **mainStatic** target és forrásfájljainak megadása világos. A forrásokon túl azonban ebben az esetben könyvtárat is rendelünk a target-hez

```
targetNev_LDADD = konyvtar
```

formában. A könyvtárat ezúttal statikus elérési úttal adjuk meg, azonban teljesen dinamikussá is tehetnénk a konfigurációs script-eket, a **configure** által beállított változókkal. Erre ebben az egyszerű tutorial-ban nem térünk ki. A fordító kapcsolók a statikus könyvtár header-jének elérési útjában különböznek csak a statikus és dinamikus könyvtárnál használt kapcsolóktól.

A dinamikus könyvtárhoz linkelt alkalmazáshoz tartozó **srcda**/**Makefile.am** szerkezete azonos, az egyetlen különbség, hogy a statikus helyett a dinamikus könyvtár elérési útjait és nevét adjuk meg. Máris látható az autotools használatának előnye, a GNU Makefile-okhoz képest, ugyanis nem kell foglalkoznunk a linkelés módjának beállításával, a statikus könyvtárakat statikusan, a dinamikusakat dinamikusan linkeli az előálló Makefile alapján a linker.

Az statikus és dinamikus könyvtárhoz is linkelt alkalmazás **srcsda**/**Makefile.am**-je megintcsak azonos a másik két alkalmazáséval, az egyetlen különbség, hogy a fordító kapcsolók között mindkét könyvtár elérési útjait megadjuk és a targethez linkelendő könyvtárak felsorolásában is szerepel mind a statikus, mind a dinamikus könyvtár.

Most, hogy értelmeztük a konfigurációs állományokat, lássuk, hogyan fordíthatjuk le a könyvtárakat és az alkalmazásokat!

A létrehozott konfigurációs állományokból első feladatként a már sokszor említett **configure** script-et kell létrehoznunk. Erre több mód is kínálkozik. Korábbi autotools eszközöknél több különböző script hívására volt szükség a **configure** legfinomabb beállításához. Ez a hívási lánc **aclocal** => **autoheader** => **autoconf** => **automake** most is használható, azonban a lépések értelmezése, paraméterezése és magyarázata messzire vezetne, ezért helyette egy kis projekteknél jól használható script-et indítunk, amely a fenti lépéseket megfelelő alapértelmezett paraméterekkel foglalja magában. Adjuk ki tehát a **testLibraries** gyökérkönyvtárban az alábbi parancsot:

```
~/testLibraries$ autoreconf --install
```

A hívás számos fájlt és könyvtárat hoz létre mind a gyökérkönyvtárban, mind az alkönyvtárakban:

```
~/testLibraries$ ls
aclocal.m4      config.h.in  configure.ac  ltmain.sh    Makefile.in  srcdl   srcsl
autom4te.cache  config.sub   depcomp       m4           missing      srcsa
config.guess    configure    install-sh    Makefile.am  srcda        srcsda
```

A létrejött fájlokkal nem kell foglalkoznunk, az autotools automatikusan hozza létre és használja őket. Látható azonban, hogy a futtatható jogokkal rendelkező **configure** script is létrejött. Futtassuk le előbb a --**help** kapcsolóval.

```
~/testLibraries$ ./configure --help
```

Látható, hogy a **configure** script számos kapcsolóval rendelkezik, amelyekkel a **Makefile.am**-ekben használt változóknak adhatunk értéket. A legfontosabb talán a --**prefix** kapcsoló, amely azt a könyvtár prefix-et adja meg, amely a targetek megadásánál a **where** rész elé kerül. Ha tehát a **configure** script-et --**prefix=/home/gykovacs** kapcsolóval hívjuk meg, akkor a későbbi **make install** hatására a bináris programok a /**home/gykovacs/bin**, a könyvtárak a /**home/gykovacs/lib** könyvtárba kerülnek. A --**prefix** alapértelmezése a /**usr** könyvtár, ami általában megfelelő.

Ha most a **configure** script-et a --**help** kapcsoló nélkül indítjuk, lefut a rendszer ellenőrzése, láthatjuk, hogy számos alapvető dolgot ellenőriz a script, mint például a C fordító működése, próbafordítással. A script a **Makefile.am**-ek felhasználásával előállítja a **Makefile**-okat minden olyan helyen, amit felsoroltunk a **configure.ac**-ben.

Kiadva a **make** parancsot a testLibraries gyökérkönyvtárban, statikus és dinamikus könyvtárunk valamint az alkalmazások is lefordulnak. Érdemes megnézni a könyvtárak linkelési parancsát, ami esetünkben egy-egy **libtool** hívás, amelynek a korábban parancssori linkelésnél használt **gcc** hívás is paramétere.

Lássuk, milyen fontos target-ek jöttek létre a Makefile-okban:

- **make** paraméter nélkül ekvivalens a **make all** hívással, azaz minden könyvtárat és alkalmazást elkészít;
- **make clean** letörli a fordítás során létrehozott már nem szükséges .o object fájlokat;
- **make distclean** letörli a **configure** script által létrehozott fájlokat;
- **make install** installálja a lefordított könyvtárakat és alkalmazásokat a **Makefile.am**-ekben megadottaknak megfelelően;
- **make uninstall** letörli az installált fájlokat;

- **make dist** létrehozza a könyvtárak és alkalmazások forráskódként történő disztribútolására alkalmas .tar.gz csomagot;
- **make distcheck** létrehozza a könyvtárak és alkalmazások forráskódként történő disztribútolására alkalmas .tar.gz csomagot, majd kitömöríti, és teszteli, hogy tényleg fordítható-e, azaz minden fájl benne van-e, ami szükséges.

Indítsuk most a **make**-et a **distcheck** targettel. Ha hibátlanul működik minden, előállt a testLibraries-1.0.tar.gz csomag. A csomag neve és verziószáma a **configure.ac**-ban foglaltaknak megfelelően áll elő. Ha belenézünk .tar.gz állományba, kitömörítve, vagy Midnight Commander használatával, megtalálhatjuk a forrásainkat, a konfigurációs állományokat, valamint a **configure** scriptet és a **configure** létrehozása során létrejött fájlokat. Nagyon fontos tehát, hogy autotools eszközökkel történő disztribúció során mindig adnunk kell a **configure** scriptet, azonban ha jól építjük fel a könyvtárunkat menedzselő konfigurációs fájlokat, a végső csomag automatikusan előáll. Ha valaki letölt egy autotools eszközkkel disztribútolt csomagot, akkor találnia kell benne egy **configure** script-et, amelyet aztán a korábbiakhoz hasonlóan esetleges kapcsolókkal futtatva kaphatjuk meg a **Makefile**-okat, amelyeket aztán felhasználhatunk a fordításhoz.

6.3.2. Fordítási opciók

Lássuk most, hogyan adhatunk a **configure** script-hez egy saját kapcsolót, a debug/release fordítás vezérlésére, jelenleg ugyanis minden –**O0** optimalizálással fordult le.

Mivel a **configure** script-hez szeretnénk új opciót adni, azt a **configure.ac**-ban kell specifikálni, hiszen a **configure.ac** alapján áll elő a **configure** script.

Adjuk hozzá az alábbi sorokat a **configure.ac** fájlhoz, a **AC_PROG_LIBTOOL** és a **CFLAGS=""** utasítások közé!

```
AC_ARG_ENABLE([debug],
[  --enable-debug Turn on debugging],
[case "${enableval}" in
  yes) debug=true ;;
  no)  debug=false ;;
  *) AC_MSG_ERROR([bad value ${enableval} for --enable-debug]) ;;
esac])
AM_CONDITIONAL(DEBUG, [test x$debug = xtrue])
```

A fenti sorok tulajdonképpen két utasítás kiadását takarják. Az **AC_ARG_ENABLE** három paramétert vár, az első az opció neve, amely kapcsolónévként a –**enable**– prefix után megjelenik. Esetünkben a "debug" nevű opciót szeretnénk bekapcsolni, így az előálló **configure** script a –**enable**–**debug** kapcsoló jelenléte esetén fogja a megfelelő beállításokat elvégezni. Az **AC_ARG_ENABLE** második paramétere egy sztring, ami a **configure** –**help** hívás esetén megjelenik. A haramdik paraméter egy többirányú elágazás utasítás **bash** szintaktikával. A haramdik paraméter tehát az az utasítás, ami a –**enable**–**elsoParameter** kapcsoló esetén végrehajtódik. A beépített **enableval** változó étéke alapján a **debug** változó értékét **true**-ra vagy **false**-ra állítjuk. Ha az **enableval** értéke nem **yes** vagy **no**, akkor hibaüzenetet írunk ki a konfigurálás során.

A következő utasításban a Linux bash-ban megszokott **test** programmal ellenőrizzük, hogy az őt követő logikai feltétel teljesül-e, azaz a **debug** változó értéke egyenlő-e **true**-val. A **debug** változó értékét a **$**-el érhetjük el, az **x**-re pedig azért van szükség, mert mivel sztringhelyettesítés után hívódik meg a **test** program, ha a **debug** változó üres, akkor **x** nélkül egy szintaktikailag helytelen **test** paraméterezést kapunk. Makró környezetekben jól bevált szokás a sztringek egyenlőségének ilyen formájú ellenőrzése. **x** helyett tetszőleges karaktersorozat állhat az egynelőség jel két oldalán. Ha az **AM_CONDITIONAL** második paramétere igaz, akkor a **DEBUG** változóba is igaz érték kerül, ellenkező esetben hamis

érték. A továbbiakban az így létrejött **DEBUG** változót használjuk fel a fordítási kapcsolók megfelelő megadására.

A létrejövő DEBUG változó az **Makefile**-ok előállítása során használható, hiszen a **configure** script futásakor állítódik be az értéke, így tehát a **Makefile.am**-ekben kell megadnunk olyan feltételes szerkezeteket, amelyekkel az egyes **Makefile**-ok **–O0 –g** vagy **–O3** fordítási flag-ekkel jönnek létre.

Hozzunk létre a **testLibraries** gyökérkönyvtárban egy **common.mk** nevű fájlt az alábbi tartalommal:

6.3.16. forráskód: common.mk

```
if DEBUG
MYCFLAGS = -O0 -g
else
MYCFLAGS = -O3
endif
```

A **common.mk** tartalma könnyen végiggondolható. Ha a **DEBUG** változó tartalma igaz, a **MYCFLAGS** változó értékeként beállítjuk a **–O0** kapcsolót, ellenkező esetben a **MYCFLAGS** tartalma a teljes optimalizálást jelentő **–O3** kapcsoló.

A következő és egyben utolsó lépés, hogy ezt a kódrészletet minden **Makefile.am**-be beletegyük, amelyre a **––enable–debug** kapcsolót értelmezni szeretnénk. Ehhez egyszerűen minden könyvtárban (a gyökérkönyvtárban nem szükséges) a **Makefile.am** elejére írjuk be a **common.mk** "include"-olását megvalósító parancsot, és emellett minden **Makefile.am**-ben módosítsuk az **AM_CFLAGS** beállítását a **MYCFLAGS** értékének hozzáadásával, azaz például a statikus **srcsl** könyvtárhoz tartozó **Makefile.am** a következőt fogja tartalmazni:

6.3.17. forráskód: srcsl/Makefile.am

```
include ../common.mk

lib_LIBRARIES = libtestStaticLibrary.a
libtestStaticLibrary_a_SOURCES = testStaticLibrary.c
include_HEADERS = testStaticLibrary.h

AM_CFLAGS = ${MYCFLAGS} -I.
```

Miután a fenti módosításokat végrehajtottuk, hozzuk létre újra a **configure** script-et a **autoreconf ––install** hívással. Ha most lefuttatjuk a **configure** script-et a **––help** paraméterrel, látnunk kell az új, saját **––enable–debug** kapcsolónkat:

```
~/testLibraries\$ ./configure --help
...
  --disable-dependency-tracking  speeds up one-time build
  --enable-dependency-tracking   do not reject slow dependency extractors
  --disable-libtool-lock  avoid locking (might break parallel builds)
  --enable-debug Turn on debugging
...
```

Ha most a **configure** script-et a **––enable–debug** kapcsolóval hívjuk és utána kiadjuk a **make** utasítást, a fordítási parancsokban megjelenik a **–O0** kapcsoló, ha azonban a **––enable–debug** nélkül hívjuk, a **–O3** kapcsoló jelenik meg a fordítási parancsokban.

Ügyeljünk arra, hogy az autotools eszközökkel is csak akkor történik fordítás, ha szükséges, azaz valamely forrás módosul. Ha ki szeretnénk próbálni a **configure** különböző hívásai közötti különbséget, a fordítás előtt adjuk ki a **make clean** parancsot, hogy a korábbi fordítás eredményeként előállt fájlok törlődjenek. Ezt követően fog a **make** utasítás hatására ténylegesen fordítani a rendszer.

6.3. Feladat: A fenti példa alapján módosítsuk a **configure.ac** és **common.mk** állományokat úgy, hogy lehetőség legyen benne a warning-ok bekapcsolására valamint az opcionálisan a **mainOptional** fájlba fordítandó dinamikus könyvtárbeli függvényhívás bekapcsolására!

6.3. Megoldás:

6.3.18. forráskód: configure.ac

```
AC_INIT([testLibraries], [1.0], [gyuriofkovacs@gmail.com])
AC_CONFIG_MACRO_DIR([m4])
AM_INIT_AUTOMAKE([foreign -Wall -Werror])
LT_INIT
AC_PROG_CC
AC_PROG_RANLIB
AC_PROG_LIBTOOL

AC_ARG_ENABLE([debug],
[  --enable-debug Turn on debugging],
[case "${enableval}" in
  yes) debug=true ;;
  no)  debug=false ;;
  *) AC_MSG_ERROR([bad value ${enableval} for --enable-debug]) ;;
esac],[debug=false])
AM_CONDITIONAL(DEBUG, [test x$debug = xtrue])

AC_ARG_ENABLE([warnings],
[  --enable-warnings Turn on warnings],
[case "${enableval}" in
  yes) warnings=true ;;
  no)  warnings=false ;;
  *) AC_MSG_ERROR([bad value ${enableval} for --enable-warnings]) ;;
esac],[warnings=false])
AM_CONDITIONAL(WARNINGS, [test x$warnings = xtrue])

AC_ARG_ENABLE([optional],
[  --enable-optional Turn on the optional function call in mainOptional],
[case "${enableval}" in
  yes) optional=true ;;
  no)  optional=false ;;
  *) AC_MSG_ERROR([bad value ${enableval} for --enable-optional]) ;;
esac],[optional=false])
AM_CONDITIONAL(OPTIONAL, [test x$optional = xtrue])

CFLAGS=""

AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile srcsl/Makefile srcsa/Makefile srcdl/Makefile srcda/Makefile
    srcsda/Makefile srcoa/Makefile])
AC_OUTPUT
```

6.3.19. forráskód: common.mk

```
if DEBUG
MYCFLAGS = -O0 -g
else
MYCFLAGS = -O3
endif
if WARNINGS
MYCFLAGS += -Wall -Wextra
endif
if OPTIONAL
MYCFLAGS += -DUSE_DYNAMIC_LIBRARY
endif
```

Az autotools magasszintű eszközöket biztosít arra, hogy a **config.h**-n keresztül történő ***#define*** bekapcsolást véghez vigyük. Elsőként távolítsuk el a **common.mk**-ból a –**DUSE_DYNAMIC_LIBRARY** bekapcsolását. Adjuk értékül a **configure.ac** megfelelő opciójában az új **DEFINE_DIRECTIVE** változónak a **config.h**-ba belerakni kívánt direktívát sztringként. A **configure.ac**-ben ki kell még adnunk az **AC_SUBST** parancsot, a **DEFINE_DIRECTIVE** paraméterrel. Ezen parancs azt jelzi az automake környezetnek, hogy a **DEFINE_DIRECTIVE** változót is helyettesíteni kell az értékével a konfigurációs fájlokban. Hozzuk létre a **config.h.in** template-et, amely a **DEFINE_DIRECTIVE** változó értékére hivatkozik, majd adjuk hozzá a **config.h** header-t a létrehozandó konfigurációs fájlok listájához, azaz az alábbi módon módosulnak az állományaink:

6.3.20. forráskód: configure.ac

```
AC_INIT([testLibraries], [1.0], [gyuriofkovacs@gmail.com])
AC_CONFIG_MACRO_DIR([m4])
AM_INIT_AUTOMAKE([foreign -Wall -Werror])
LT_INIT
AC_PROG_CC
AC_PROG_RANLIB
AC_PROG_LIBTOO
L

AC_ARG_ENABLE([debug],
[  --enable-debug Turn on debugging],
[case "${enableval}" in
  yes) debug=true ;;
  no)  debug=false ;;
  *) AC_MSG_ERROR([bad value ${enableval} for --enable-debug]) ;;
esac],[debug=false])
AM_CONDITIONAL(DEBUG, [test x$debug = xtrue])

AC_ARG_ENABLE([warnings],
[  --enable-warnings Turn on warnings],
[case "${enableval}" in
  yes) warnings=true ;;
  no)  warnings=false ;;
  *) AC_MSG_ERROR([bad value ${enableval} for --enable-warnings]) ;;
esac],[warnings=false])
AM_CONDITIONAL(WARNINGS, [test x$warnings = xtrue])

AC_ARG_ENABLE([optional],
[  --enable-optional Turn on the optional function call in mainOptional],
[case "${enableval}" in
  yes) optional=true
  DEFINE_DIRECTIVE="#define USE_DYNAMIC_LIBRARY";;
  no)  optional=false ;;
  *) AC_MSG_ERROR([bad value ${enableval} for --enable-optional]) ;;
esac],[optional=false])
AM_CONDITIONAL(OPTIONAL, [test x$optional = xtrue])

CFLAGS=""

AC_CONFIG_HEADERS([config.h])
AC_SUBST([DEFINE_DIRECTIVE])
AC_CONFIG_FILES([Makefile srcsl/Makefile srcsa/Makefile srcdl/Makefile srcda/Makefile
    srcsda/Makefile srcoa/Makefile srcoa/config.h])
AC_OUTPUT
```

6.3.21. forráskód: common.mk

```
if DEBUG
MYCFLAGS = -O0 -g
else
MYCFLAGS = -O3
```

```
endif
if WARNINGS
MYCFLAGS += -Wall -Wextra
endif
```

6.3.22. forráskód: srcoa/config.h.in

```
@DEFINE_DIRECTIVE@
```

Az **configure** fájl **enable–optional** opcióval való futtatása után létrejön az **srcoa**/**config.h** állomány, amely éppen a megfelelő ***#define*** direktívát tartalmazza.

Összefoglalva tehát az autotools professzionális eszközöket biztosít források menedzselésére, azonban nagyobb projektek esetén a makrók használata miatt nehézkes lehet. Bár nagyon sok disztribúcióban használják, javaslom a későbbiekben ismertetendő **cmake** vagy **qmake** eszközök használatát, amelyek tovább egyszerűsítik a projektek menedzselését. Figyeljük meg, hogy ez utóbbi megoldásnál a **–DUSE_DYNAMIC_LIBRARY** kapcsoló bekerül minden fájl fordításának parancsába. Ez nem jelent problémát a fordítás során.

6.3.3. Külső könyvtárak használata

Külső könyvtárak használatának demonstrálásához adjuk hozzá a korábbi **mainPNG.c** forrásfájlt az aktuális környezetünkhöz, például az **srcpng** könyvtárba, és hozzunk létre hozzá egy megfelelő, egyszerű **Makefile.am**-et.

Egy csomag meglétét ezt követően a **configure.ac**-ba elhelyezett **PKG_CHECK_MODULES** utasítással ellenőrizhetjük. Működését tekintve két kötelező paramétert vár, az első paramétere egy sztring, a második paramétere pedig egy könyvtár (csomag) neve. Esetünkben a sztring **LIBPNG**, a csomagnév **libpng** lesz. Megkeresi a rendszeren (az alapértelmezett, és a **PKG_CONFIG_PATH** változóban megadott elérési utakon) a **libpng.pc** állományt. Amennyiben van ilyen, definiál egy **LIBPNG_CFLAGS** változót, amely a fordítási flag-eket tartalmazza, valamint egy **LIBPNG_LIBS** változót, ami a linker flageket fogja tartalmazni. Ezt követően a megfelelő **Makefile.am**-ekben csak ezen változók értékére kell hivatkoznunk a megfelelő fordítási és linkelési paranacsok összerakásához. A **configure.ac**-ba kerülő utasítás tehát:

```
PKG_CHECK_MODULES([LIBPNG],[libpng])
```

míg a **mainPNG.c** forrás **Makefile.am**-e az alábbi módon alakul:

6.3.23. forráskód: srcpng/Makefile.am

```
include ../common.mk

bin_PROGRAMS = mainPNG
mainPNG_SOURCES = mainPNG.c

AM_CFLAGS = ${MYCFLAGS} -I. -I$(srcdir)/../srcdl -I$(srcdir)/../srcsl ${LIBPNG_CFLAGS}

mainPNG_LDADD = ../srcdl/libtestDynamicLibrary.la \
  ../srcsl/libtestStaticLibrary.a \
  ${LIBPNG_LIBS}
```

6.3.4. Disztribúciók létrehozása

Az **automake** rendszer a **.tar.gz** jellegű, forráskódot és konfigurációs szkriptet tartalmazó disztribúciók létrehozását támogatja. Ahhoz azonban, hogy könyvtárunkat más könyvtárak is használhassák, szükség van arra, hogy a korábban már említett package-config fájlok is előálljanak a könyvtárunkhoz, hiszen más szoftverek ezen .pc fájlokon keresztül találják meg könyvtárunkat a rendszeren.

A package-config fájlok azonban konkrét elérési utakat tartalmaznak, amelyek csak egy adott gép esetén érvényesek, így nem készíthetünk egzakt .pc fájlokat előre. A megoldás hasonló a **config.h** header létrehozásához: készítünk egy .pc template-et, és azt felvesszük a **configure.ac** fájlban a konfiguráció során létrehozandó fájlok listájába. A package-config fájl template például az alábbi lehet teszcsomagjaink esetén:

6.3.24. forráskód: testlibs.pc.in

```
prefix=@prefix@
exec_prefix=@prefix@
libdir=@prefix@/lib
includedir=@prefix@/include

Name: testlibs
Description: Test libraries
Version: 0.0.1
Libs: -L@prefix@/lib -ltestStaticLibrary -ltestDynamicLibrary
Cflags: -I@prefix@/include
```