# A GNU Fordítógyűjtemény (2. rész)

Ebben a részben a GCC kapcsolóival és használatuk rejtelmeivel ismerkedhetünk meg.

Amennyiben már sok függvényt írtunk meg, és azok eléggé általánosak ahhoz, hogy más programokban is felhasználjuk őket, célszerű könyvtárakat kialakítani belőlük. A programfejlesztésben a könyvtár (library) szó újra felhasználható, lefordított programrészeket jelent, amelyeket egy állományba gyűjtünk össze.
Gyakorlásképpen készítsünk könyvtárat sorozatunk előző részében használt ***quaternion.c*** forrásfájlból, ami egyetlenegy saját függvényt tartalmaz, a `printQuaternion()` nevűt, ez hívja meg a standard C könyvtár `printf()` függvényét. Első lépésben az ismert módon tárgykódot kell készítenünk a forráskódból, majd azt az `ar` program segítségével archív állománnyá kell alakítanunk:

```
> gcc -c quaternion.c -o liblinux.o
> ar rcs liblinux.a liblinux.o
```

Az `ar` program utáni `rcs` kapcsolók jelentése a következő:

- `r`: beszúrja a felsorolt fájlokat az archív állományba, törölve az ott már jelenlévő, azonos nevűeket (az r rövidítés a replace, azaz helyettesít szóból származik).
- `c`: létrehozza az archív fájlt, ha még nem létezik (a c a create szó első betűjére utal).
- `s`: tárgykód fájlindexet fűz az archív állományhoz vagy frissíti már létező fájl esetében.

A GNU `ar` program utáni első fájlnév a létrehozandó archív neve lesz, ezután kell felsorolni azokat a fájlokat, amelyeket az archívban akarunk tárolni. Az archív fájlok szerkezete olyan, hogy több tárgykódfájlt képesek összefogni, miközben az egyes fájlok és azok tartalma számunkra elérhető marad. Az archívban lévő fájlokat tagoknak (members) hívjuk, és az `ar` programmal módosíthatjuk őket. Az archív könyvtárfájlok kiterjesztése alapértelmezetten ***.a***.
Ne felejtsük el, hogy új könyvtárak létrehozásakor fejállományokat is kell készítenünk, és azokat elérhetővé kell tennünk a fordító számára!
Ha most a ***liblinuxvilag.a*** könyvtárat bemásoljuk egy munkamappába a a ***41-es CD Magazin/GCC/lib/static/01*** mappájában található fájlokkal együtt, akkor lefordíthatóvá válik picurka, de nagy tudású programunk:

```
> gcc linuxvilag.c -o linuxvilag -I. -L.
➥-llinux
```

A `-L` kapcsoló a könyvtár helyét mutatja, ami a ***01*** példamappában a munkamappa, ezt egy pont jelzi. Mivel egy mappában több más programkönyvtár is lehet, a GCC fordítónak meg kell mondanunk , hogy programunkat melyikkel akarjuk összekapcsolni. Ehhez a `-l` kapcsolót használjuk, ami után a használni kívánt könyvtár neve következik.
Az előző CD-melléklet ***Magazin/GCC/lib/static/02*** mappájában ugyanezek a fájlok vannak elhelyezve, de itt két almappát is kialakítottam, ahová a fejállományt és a linuxvilag könyvtárat tettem. Ebből az okból kifolyólag most a következő fordítási utasítást kell kiadnunk:

```
> gcc linuxvilag.c -o linuxvilag -I./header
➥-L./library -llinuxvilag
```

A fordítás után a linuxvilag program futtatható lesz. Míg a ***02*** almappában a pillanatnyi munkakönyvtárhoz viszonyított elérési utat adtam meg, a ***03*** almappában teljesen relatív útmegadást alkalmaztam:

```
> gcc linuxvilag.c -o linuxvilag -Iheader
➥-Llibrary -llinux
```

Természetesen abszolút elérési út megadása is lehetséges lett volna:

```
> gcc linuxvilag.c -o linuxvilag
➥-I/home/linuxvilag/gcc/lib/static/04/header
➥-L/home/linuxvilag/gcc/lib/static/04/library
➥-llinux
```

Munkánk végeztével elégedetten nyugtázhatjuk, hogy a linuxvilag nevű statikus könyvtárral összekapcsolt programunk lefut.

## Osztott könyvtárak készítése

A statikus könyvtárak (static libraries) újra felhasználhatók, de minden egyes újonnan megírt és lefordított programhoz ismételten hozzá kell kapcsolnunk őket. A dinamikus vagy más néven osztott könyvtárak (shared libraries) viszont csak egyszer kerülnek be a memóriába, és ott egyszerre több program is el tudja érni őket.
A statikus könyvtárakat fordításidőben kapcsoljuk a programokhoz, az osztott könyvtárakat pedig futásidőben.
Ha osztott könyvtárat akarunk készíteni, akkor először a forráskódból kell tárgykódot készítenünk az ismert módon, de most használnunk kell a `-fPIC` kapcsolót, hogy a fordító helyzettől független kódot (Position Independent Code) hozzon létre, ami azt jelenti, hogy a behívott programrészlet bármilyen memóriacímre képes betöltődni:

```
> gcc -fPIC -c quaternion.c -o liblinux.o
```

A Unix-hagyományok szerint a könyvtárak nevei a ***lib*** rövidítéssel kezdődnek, ezért nem kell teljes névvel hivatkozni rájuk. Például a fenti ***liblinux.a*** statikus könyvtárra elegendő volt a `-llinux` névvel hivatkozni. A megosztott könyvtárak nevei szintén a library szóból származó lib szócskával kezdődnek, de a név után egy pont és a so kiterjesztés következik. Ezt a formát so-névnek (soname) hívják, például a ***vga*** könyvtár esetében ez ***libvga.so***. A so név után a fő- és az alváltozatszám következik, majd a kiadás azonosítója.
Miután létrehoztuk a ***liblinux.o*** tárgykódfájlt, a következő paranccsal el kell készítenünk a megosztott könyvtárat:

```
> gcc -shared -Wl,-soname,liblinux.so
➥-o liblinux.so.1.0.0 liblinux.o -lc
```

A következő kapcsolókat láthatjuk a parancs után:

- `-shared`: olyan tárgykódot hoz létre, ami később futásidőben is hozzákapcsolható más programokhoz.
- `-Wl`: az összekapcsoló program (linker) számára ad át kapcsolókat. A vesszők teszik lehetővé, hogy egyszerre többet is megadjunk.
- `-soname`: a fentebb említett so névre utal.
- `-liblinux.so`: az adott könyvtár tényleges so neve.
- `-liblinux.so.1.0.0`: a teljes so név a fő- és alváltozatszámmal, valamint a kiadást jelölő számmal együtt.

- `-liblinux.o`: az a tárgykódfájl, amit korábban a `-fPIC` kapcsolóval fordítottunk le.
- `-lc`: a gépünkben lévő C könyvtárhoz kapcsolódunk, hogy lehetetlenné tegyük egy olyan C-változat használatát, ami miatt a program futása megbízhatatlanná válna. Ha ezt a kapcsolót használjuk, egy inkompatibilis C könyvtár esetén le sem fogjuk tudni fordítani a programot (lásd még az `info ldconfig` súgóoldalt).

Ezután létre kell hoznunk a közvetett hivatkozásokat:

```
ln -s liblinux.so.1.0.0 liblinuxvilag.so.1
ln -s liblinux.so.1.0.0 liblinuxvilag.so
```

Ezzel el is készítettünk megosztott könyvtárunkat, amire elegendő a linuxvilag szóval hivatkozni. Az utolsó előtti lépésben lefordítjuk programunkat:

```
> gcc linux.c -o linuxvilag -Iheader
➥-Llibrary -llinuxvilag
```

Ha nem vagyunk biztosak abban, hogy programunk valóban egy osztott könyvtárhoz kapcsolódott-e, használjuk az `ldd` programot, például:

```
> ldd static
```

Az `ldd` a Library Dependency Display szavak rövidítése, amit könyvtárfüggőség-kijelzőként fordíthatnánk magyarra. Ez az eszköz megmutatja nekünk, hogy egy adott program milyen megosztott könyvtárakkal van összekapcsolva.
Ha a `static` nevű, futtatható állományhoz (lásd az ***executable*** almappát az előző CD-mellékleten) a statikus linuxvilag könyvtárat kapcsoltuk, az `ldd` program a következőt írja ki a standard kimenetre:

```
libc.so.6 => /lib/libc.so.6 (0x4002c000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2
(0x40000000)
```

Ez azt jelenti, hogy programunk a standard C könyvtár mellett még az ***ld.so*** betöltőprogramot is használja.

## A GCC kapcsolói

Esély sincs arra, hogy ebben a rövid írásban felsoroljam a GCC összes kapcsolóját, de néhány fontosabbat megemlítenék. Ezek a kód egyszerűsítésével, a hibakereséssel és a GCC üzeneteivel kapcsolatosak. Aki részletesebb ismertetésre vágyik, az nézze meg az `info gcc` súgólapokat.

- ***Egyszerűsítés***
  Az optimalizálás növeli a forráskód lefordításához szükséges időt.
  `-O` – a fordító megpróbálja csökkenteni a tárgykód méretét és gyorsítani a lefordítandó programok végrehajtási idejét.
  `-O0` – nincs semmiféle egyszerűsítés.
  `-O1` – ugyanaz, mint a `-O` kapcsoló.
  `-O2` – magában foglalja az összes `O1` szintű egyszerűsítést, amit még olyan utasításokkal egészít ki, amik növelik a processzor munkájának a hatékonyságát. Az `O2` szintű egyszerűsítés általában kellő hatékonyságú.
  `-O3` – magában foglalja az összes `O2` szintű egyszerűsítést, de cikluskibontással további gyorsítást próbál elérni, azaz ha egy ciklusnak a lépésszáma már a fordítás idején ismert, akkor a fordító megszünteti a ciklust, és minden egyes lépést egymás után következő programrésszé alakít át, megnövelve így a program méretét, de felgyorsítva annak futását.
  `-Os` – a program méretének csökkentése az elsődleges cél. Magába foglal minden olyan `O2` szintű egyszerűsítést, ami nem növeli nagymértékben a program méretét.
- ***Hibakeresés***
  A hibakeresést megkönnyítő adatok kerülnek bele a tárgykódállományokba, ami lehetővé teszi a nyomkövetést, de nagymértékben megnöveli a bináris állományok méretét. Ez a növekedés akár nagyságrendi is lehet. Logikusnak tűnhet, hogy könyvtárainkat hibakeresés nélkül fordítsuk le, hiszen így a méretük kisebb lesz. De ha arra gondolunk, hogy más fejlesztők is használni fogják azokat, már nem feltétlenül olyan jó ötlet a hibakeresés lehetetlenné tétele.
  Itt is több szint létezik:
  `-g` – az alapértelmezett szint.
  `-g1` – a legalacsonyabb szint. A legkevesebb hibakereséshez felhasználható adatot tartalmazza.
  `-g2` – megegyezik a `-g` kapcsolóval, tehát ez az alapértelmezett szint. Szimbólumtáblákat épít bele a tárgykódállományba, bent hagyja a sorok számozását és további adatot kapunk a helyi és külső változókról.
  `-g3` – tartalmazza a `g2` szint minden hibakeresést segítő adatát a makró-meghatározásokkal együtt.
  `-ggdb` – olyan adatokat épít be a tárgykódállományba, amiket a GNU hibakereső is tud használni. Ez a program a GDB, amiről egy későbbi írásomban még részletesen szólni kívánok. Ennek is három szintje van, például a `-ggdb3` alakot is beírhatjuk.

Általában nem célszerű a hibakereső adatok beépítése az egyszerűsítéssel együtt, annak ellenére, hogy a GCC ezt megengedi. Nem várt hibák léphetnek fel, mivel az egyszerűsítés következtében eltűnhetnek olyan változók, amikről biztosan tudjuk, hogy előzőleg megadtuk őket; programunk vezérlése futás közben olyan helyre ugorhat, amire egyáltalán nem számítunk; vannak olyan függvények, amelyekre várakozásainkkal ellentétben nem kerül rá a vezérlés, mert azok eredményét állandónak tekinti az egyszerűsítő program; más függvények pedig egészen máshová kerülnek, mint ahová mi eredetileg megírtuk őket, mert kikerültek a ciklusokból. Általános szabálynak tekinthetjük, hogy elsőként mindig a hibákat keressük meg, és csak azután egyszerűsítjük a tárgykódállományt, miután úgy gondoljuk, hogy az hibátlan.

- ***Hibák és üzenetek***
  A GCC megkülönbözteti az olyan hibákat, amelyek valóban végzetes hibák és lefordíthatatlanná teszik a programunkat, és az olyan hibákat, amelyek valamilyen programozói odanemfigyelésből erednek, mint például egy változó megadása, de fel nem használása. Ezzel a témakörrel kapcsolatban két kapcsolót említek meg, amelyeknek a feladatát a `-W` után következő szó határozza meg:
  `-Wall` – minden üzenetet kiír fordítás közben.
  `-Werror` – minden üzenetet hibaként értelmez.

## A GCC Fordítógyűjtemény

A fenti példákban C-forráskódot alakítottunk futtatható állománnyá a GCC segítségével. Az utolsó példában azonban már egy C++-fájlt fordítottunk le a következő utasítással (lásd az előző CD-melléklet ***gcc/code/example09*** mappáját):

```
gcc quaternion.cc linuxvilag.cc -o linuxvilag
```

A `gcc` minden nehézség nélkül lefordítja a C++, Java, Fortran, Pascal programokat.

*Szaló István*
(ratiosoft@freemail.hu) tanár, immár több mint másfél évtizede foglalkozik programozással, de csak a Java és a Linux megismerése után tudta meg, hogy mi is az igazi programozás. Több írása megjelent már a hazai számítástechnikai lapokban.