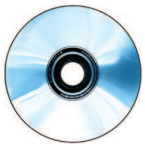

A GNU Fordítógyűjtemény

Miután forráskódunkat egy nekünk tetsző szövegszerkesztőben megírtuk, a processzor számára is érthető gépi kódra kell lefordítanunk, hogy futtatható állományt kapjunk. Ehhez használhatjuk a GNU Fordítógyűjteményt.

Csábító lenne arra gondolni, hogy a GCC (Linking) a GNU C Compiler szavak rövidítése, azaz nem más lehet, mint a GNU C Fordító. Ez néhány éve még igaz volt, de alkotói a megváltozott helyzetnek megfelelően mára más nevet adtak neki: a GCC az angol GNU Compile Collection szavakból alkotott mozaikszó, ami arra utal, hogy a GNU CC nem egyetlen fordítót tartalmaz, hanem különböző programozási nyelveket képes gépi kódra átfordítani, többek között Linux-környezetben. De ha megfelelően van lefordítva, akkor keresztfordítóként (cross compiler, azaz több felületen működő fordítóként) is képes üzemelni, ami azt jelenti, hogy nemcsak a gépünkben lévő, hanem másfajta processzorcsaládokra is programokat tudunk vele fordítani. Például egy Intel processzort tartalmazó x86-os gépen olyan gépi kódot állíthatunk elő, ami Alpha processzorokon is képes lesz futni. Több Linux-terjesztés GCC fordítója alapértelmezetten nem képes ilyen átfordításra, ezért ilyenkor a GCC-csomagot a keresztfordításnak megfelelő beállításokkal újra kell fordítani.
A FreeBSD-hez, az Emacs ***lisp*** könyvtárakhoz és a Linux-rendszermaghoz hasonlóan a GCC is a bazár stílusú fejlesztési modell szerint készül, így nevezte ***Eric S. Raymond*** a nyílt forráskódra alapozott programírást. Hogy egyértelmű legyen a szöveg, a továbbiakban a fájlokat tartalmazó könyvtárakat mappáknak fogom nevezni, hogy megkülönböztessem őket a tárgykódot tartalmazó könyvtáraktól.
Ahogy egy bazárban illik, a GCC készítői a forráskódot mindenki számára olvashatóvá teszik, és írhatóvá varázsolják a fejlesztők számára, akik közül néhány vezető személynek joga van ahhoz, hogy a javasolt változtatásokat elfogadja vagy elutasítsa. Az alkotók szerint a fejlesztés végső célja az, hogy a GCC legyen a világ legjobb fordítója, ezért mindenkitől szívesen fogadják a segítséget, és mindenkit szeretettel várnak a fejlesztők levelezési listáin. Gyakorta adnak ki frissítéseket, amelyek általában nem teljességgel megbízható változatok, hanem úgynevezett pillanatképek (snapshots), és azt mutatják, hogy éppen hol tartanak a kódolási munkák. Elvárják azt is, hogy az esetleges hibákról értesítést kapjanak.

A fordítás

A GCC nem teljes fejlesztőkörnyezet, hanem parancssori fordító. Kezdetnek írjuk be a következő rövid C nyelvű programot egy szövegszerkesztőbe, majd ***linuxvilag.c*** néven mentsük:

```
#include <stdio.h>

int main()
{
  printf("Linuxvilag\n");
  return(0);
}
```

Majd egy parancsértelmezőben adjuk ki a következő parancsot:

```
> gcc linuxvilag.c
```

Most és a továbbiakban a > jel nem része a parancsnak, hanem a parancssorjel, ami a rendszergazda esetében például kettős kereszt (#) lehet. A fordítás eredménye egy ***a.out*** nevű futtatható állomány, ami az alapértelmezett elnevezés. Ha a lefordított fájlt bármilyen szövegnézővel megnézzük, láthatjuk, hogy a fájl első négy betűje a ***.ELF***. Az ELF az Executable and Linkable Format szavak rövidítése, ami futtatható és összeépíthető formátumot jelent. Manapság ez az alapértelmezett fájlformátum, és mind a futtatható, mind a könyvtárállományok a fordítás során elf formátumot kapnak. Nekem az elf szóról mindig a manók jutnak eszembe, hiszen a szónak ez az eredeti angol jelentése.
Ha a bináris állománynak nem az alapértelmezett ***a.out*** elnevezést akarjuk adni, a `-o` kapcsoló használatával ezt tudatnunk kell a fordítóval:

```
> gcc linuxvilag.c -o linuxvilag
```

Most a lefordított állomány neve ***linuxvilag*** lesz. Ennek a kapcsolónak a hatására a GCC a kimenetét a megadott nevű fájlba küldi, függetlenül attól, hogy az futtatható program, bináris állomány vagy assembly kód volt-e. Például írjuk be a következő rövid kódrészletet:

```
struct Quaternion {
  int o;
  int x;
  int y;
  int z;
};

void printQuaternion(struct Quaternion *q)
{
  printf("%d,%d,%d,%d\n",q->o,q->x,q->y,q->z);
}
```

Mint tudjuk, a quaterniók hiperkomplex számok, amelyeket az a+bi+cj+dk alakban írhatunk fel, ahol i2 = -1, j2 = -1, k2 = -1 és ij = k, ji = -k, jk = i, kj = -i, ki = j, ik = -j. A hiperkomplex számok tulajdonképpen a komplex számok kiterjesztései, és gyakoriak a vektoralgebrában, ahol többek között tetszőleges térbeli elforgatások előállítására használhatjuk őket. A mi quaternion szerkezetünk egy ilyen négyes számot jelképez, és rövidke forráskódunk a `printQuaternion()` függvény segítségével megjeleníti azt a parancssoron. Miután beírtuk, és ***quaternion.c*** néven mentettük a forrásfájlt, fordítsuk le a következő paranccsal:

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

A `-c` és `-o` kapcsolók most azt mondják a fordítónak, hogy a ***quaternion.c*** nevű forrásfájl lefordított kódját a ***quaternion.o*** fájlba tegye. Nem meglepő, hogy ez a bináris fájl nem futtatható! Ahhoz, hogy egy C- vagy C++-állomány futtatható legyen, meg kell adnunk valahol egy `main` nevű függvényt, ami minden programindításkor elsőként fut le. Tehát a ***linuxvilag.c*** nevű fő fájlunk így fog kinézni:

```
#include "quaternion.h"

struct Quaternion quaternion = {0,1,2,3};

int main()
{
  printQuaternion(&quaternion);
  return(0);
}
```

A `printQuaternion()` függvény meghatározását a ***quaternion.c*** nevű forrásfájlba tesszük:

```
#include <stdio.h>
#include "quaternion.h"

void printQuaternion(struct Quaternion *q)
{
  printf("%d,%d,%d,%d\n",q->o,q->x,q->y,q->z);
}
```

A quaternion szerkezet és a `printQuaternion()` függvény meghatározásait a ***quaternion.h*** fejállományba tesszük:

```
#ifndef QUATERNION_H_
#define QUATERNION_H_

struct Quaternion {
  int o;
  int x;
  int y;
  int z;
};

void printQuaternion(struct Quaternion *q);

#endif /* QUATERNION_H_ */
```

A fejállományba egy úgynevezett állományőrszemet építettem be, ami megakadályozza, hogy a fordítóprogram ezt a fejállományt többször is feldolgozza, hiszen amikor először találkozik vele, értéket ad a `QUATERNION_H_` makrónak, másodszor vagy harmadszor viszont már mindent figyelmen kívül hagy, ami a `#ifndef` és `#endif` fordítási irányelvek között van.

Most már mindkét fájlt az ismert módon egyenként lefordíthatjuk:

```
> gcc -c quaternion.c -o quaternion.o
> gcc -c linuxvilag.c -o linuxvilag
```

Amikor azonban megpróbáljuk lefuttatni a fordító által létrehozott ***linuxvilag*** bináris állományt, az „Engedély megtagadva” üzenetet kapjuk:

```
bash: ./linuxvilag: Permission denied
```

A linuxvilag fájlnév előtti ./ karakterek a pillanatnyi munkakönyvtárra utalnak. Miután az `ls -l` paranccsal listázzuk a könyvtár tartalmát, látjuk, hogy a ***linuxvilag*** állomány írható és olvasható, de nem futtatható, mert nem látjuk az x betűt:

```
-rw-r r     1 ratio    users          904 FEB
8 23:14 linuxvilag*g*
```

Nem esünk kétségbe, hanem a `chmod` parancs segítségével saját magunk számára futtathatóvá tesszük az állományt:

```
> chmod u+x linuxvilag
```

De ismét csalatkoznunk kell, hiszen most a „Nem tudom futtatni a bináris állományt” üzenetet kapjuk:

```
bash: ./linuxvilag: cannot execute binary file
```

Némi gondolkodás után arra a következtetésre juthatunk, hogy hiába hoztuk létre a ***quaternion.o*** és a ***linuxvilag*** bináris állományokat, semmit sem tudnak sem egymásról, hiszen nem kapcsoltuk össze őket. A GCC az összekapcsolást (linking) is megteszi, de a megfelelő utasításokat kell neki adnunk:

```
> gcc -c quaternion.c -o quaternion.o
> gcc -c linuxvilag.c -o linuxvilag.o
> gcc quaternion.o linuxvilag.o -o linuxvilag
```

Az összekapcsolás az utolsó sorban történik, ahol a GCC a ***quaternion.o*** és ***linuxvilag.o*** bináris állományokat fűzi össze a ***linuxvilag*** nevű futtatható állománnyá. Most már a ***linuxvilag*** program tudni fog mind a ***quaternion.o***, mind pedig a ***linuxvilag.o*** bináris állományok tartalmáról.

Nem feltétlenül kell minden egyes ***.c*** kiterjesztésű forráskódfájlt ***.o*** kiterjesztésű bináris fájllá alakítani. A következő paranccsal a forrásfájlokból közvetlenül kapjuk az összekapcsolt futtatható állományt:

```
> gcc quaternion.c linuxvilag.c -o linuxvilag
```

A fordítás szakaszai

A megfelelő kapcsolók használatával a többlépcsős fordítási folyamatot megszakíthatjuk, és megtekinthetjük a létrehozott állományokat.

1. ***Az előfeldolgozó***

   A futtatható állomány létrehozásának folyamatában nemcsak fordítás és összekapcsolás szerepel, hanem előfeldolgozás is. Valójában nem a fordító kapja meg elsőként a forráskódot, hanem az úgynevezett előfeldolgozó (preprocessor), azaz a cpp program. Írjuk be a következő parancsot:

   ```
   > cpp linuxvilag.c linuxvilag.i
   ```

   Most pedig egy szövegszerkesztőben nézzük meg a létrejövő, előfeldolgozott ***linuxvilag.i*** fájlt. A ***.i*** kiterjesztés nem véletlen, hiszen általában ezt adjuk a feldolgozott állományoknak. Hagyományosan a következő kiterjesztéseket használhatjuk:

.h	fejállomány az előfeldolgozó számára
.c	előfeldolgozásra szoruló C-forráskód
.C	előfeldolgozásra szoruló C++-forráskód
.cc	előfeldolgozásra szoruló C++-forráskód
.cpp	előfeldolgozásra szoruló C++-forráskód
.cxx	előfeldolgozásra szoruló C++-forráskód
.m	Objective-C forráskód
.i	előfeldolgozott C-állomány
.ii	előfeldolgozott C++-állomány
.S	előfeldolgozásra szoruló assembly forráskód
.s	előfeldolgozott assembly forráskód
.o	lefordított tárgykódfájl
.a	lefordított könyvtárállomány
.so	lefordított könyvtárállomány

   Az előfeldolgozó a fejállományokat hozzáfűzi a forráskódhoz, lehetővé téve, hogy feltételes fordítási irányelveket adhassunk meg, és hogy makrókat hozhassunk létre, azaz olyan rövidítéseket, amelyek hosszabb szövegrészek helyett állnak. Ezeket a rövidítéseket az előfeldolgozó kibontja, és önműködően behelyettesíti a makrók helyébe.

   A gcc a `-E` kapcsolóval rávehető arra, hogy megálljon, miután az előfeldolgozó elvégezte a munkáját:

   ```
   > gcc -E linuxvilag.c -o linuxvilag.cpp
   ```

   A példában az előfeldolgozott állomány a ***linuxvilag.cpp*** lesz. Mind a súgó, mind pedig az info oldalakon hangsúlyozzák, hogy az előfeldolgozót csak C vagy azzal együtműködő programozási nyelvekkel használjuk!

2. ***Az assembly forráskód***

   A C nyelvű forráskódot a GCC a következő lépésben assembly forráskóddá alakítja át. Ennél a folyamatnál is megállhatunk a `-S` kapcsoló használatával:

   ```
   > gcc -S linuxvilag.c
   ```

Mivel kimeneti fájlnevet nem adtunk meg, a GCC az assembly kódot alapértelmezetten egy *.s* kiterjesztésű állományba teszi. Ha más nevet akarunk neki adni, a `-o` kapcsolót kell használnunk:

```
> gcc -S linuxvilag.c -o lv.s
```

A fenti, legelső ***linuxvilag.c*** forráskód assembly forráskódja így néz ki:

```
        .file   "linuxvilag.c"
        .version        "01.01"
gcc2_compiled.:
.section        .rodata
.LC0:
        .string         "Linuxvilag\n"
.text
        .align 16
.globl main
        .type    main,@function
main:
        pushl %ebp
        movl %esp,%ebp
        subl $8,%esp
        addl $-12,%esp
        pushl $.LC0
        call printf
        addl $16,%esp
        xorl %eax,%eax
        jmp .L2
        .p2align 4,,7
.L2:
        movl %ebp,%esp
        popl %ebp
        ret
.Lfe1:
        .size    main,.Lfe1-main
        .ident "GCC: (GNU) 2.95.3 20010315
                ➥(SuSE)"
```

3. ***A tárgykód előállítása***
   A harmadik lépésben a GCC az assembly forráskódot tárgykóddá alakítja, aminek alapértelmezetten *.o* a kiterjesztése. A fordítást a `-x` kapcsolóval bárhonnan újrakezdhetjük:
   `> gcc -x c -c linuxvilag.i -o linuxvilag.o`
   A `-x` kapcsoló utáni c betű a C nyelvre utal, és a helyébe a következő nyelvek nevei helyettesíthetők be: `c`, `objective-c`, `c++`, `c-header`, `cpp-output`, `c++-cpp-output`, `assembler` és `assembler-with-cpp`.
   Magát a fordítást az `as` program végzi, ami nem más mint a GNU assembler. Ezt nekünk nem kell külön lefuttatni, a GCC hívja meg helyettünk.
4. ***Az összekapcsolás***
   Az utolsó lépésben az összekapcsoló (linker) összefűzi a tárgykódfájlokat, elrendezi azok adatait és összeköti a szimbólumtáblákat, majd futtatható állományt készít belőlük. A GNU összekapcsoló program az `ld`. Ha kíváncsiak vagyunk rá, hogy egy tárgykódfájlban milyen szimbólumok vannak, adjuk ki az `nm` parancsot, például:

```
> nm quaternion.o
00000000 t gcc2_compiled.
00000000 T printQuaternion
         U printf
```

Az első sorban a hexadecimális számként megadott érték a tagnak a kezdő címhez viszonyított helyzetét mutatja, a `T` betű arra utal, hogy a hozzá tartozó szimbólumot a tárgykódhoz tartozó forráskódban határozták meg, az `U` betű pedig arra, hogy ezt valahol máshol, egy másik forrásfájlban tették meg. Ha kisbetűket látunk, a szimbólum hatóköre helyi (local), ha nagybetűs, akkor globális.
Az `nm` program olyankor jön jól, amikor kíváncsiak vagyunk arra, hogy egy tárgykódfájlban egy adott nevű függvény megtalálható-e. Látjuk például, hogy a fenti listázás szerint a ***quaternion.o*** bináris állomány a `printQuaternion` és `printf` nevű függvényeket tartalmazza, amit a forráskód ismételt megtekintésével ellenőrizhetünk is.
A GCC a fenti négy lépést általában elrejti előlünk, és a futtatható állományt, ha megfelelő utasításokat adunk neki, egy lépésben állítja elő.

A fejállományok elérhetősége

Ha a Linuxvilág CD-mellékletén a Magazin/gcc/code/example04 és Magazin/gcc/code/example05 könyvtárak tartalmát összehasonlítjuk, azt látjuk, hogy az ***example05*** példánál a ***quaternion.c*** forrásfájlt és a ***quaternion.h*** fejállományt változtatás nélkül átmásoltam a ***quaternion*** alkönyvtárba. Ha most az ***example05*** mappában a ***linuxvilag.c*** állományt a szokott módon le akarom fordítani, a következő hibaüzenetet kapom:

```
gcc: quaternion.c: No such file or directory
linuxvilag.c:2: quaternion.h: No such file or
➥directory
```

Nem különösebben meglepő, hogy a fordítóprogram az elkülönített fájlokat nem találja, hiszen azok nincsenek ugyanabban a munkamappában, mint ahonnan megkíséreltem a fordítást. Hiába keresi tehát őket – végül kiírja a „Nincs ilyen fájl vagy mappa” hibaüzenetet. Még azt is megtudjuk, hogy a keresett ***quaternion.h*** fejállományra a ***linuxvilag.c*** forrásfájl második sorában hivatkozunk.
Nincs mit tennünk tehát, meg kell adnunk a keresett fájlok elérési útját:

```
> gcc ./quaternion/quaternion.c linuxvilag.c
➥-I./quaternion -o linuxvilag
```

A fordítónak a `-I` kapcsolóval a fejállományok helyét adhatjuk meg. Felmerülhet bennünk a kérdés, hogy a fordító miért nem keresi a szintén ismeretlen helyen lévő ***stdio.h*** fejállományt. Nos, ez a fejállomány része a szabványos C be- és kiviteli (I/O) függvényeket tartalmazó könyvtárnak, ezért a fordító alapból eléri. Próbaképpen töröljük ki a `#include <stdio.h>` sort a ***linuxvilag.c*** forráskódból, majd fordítsuk újra a programot!

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.

Kapcsolódó címek

A GCC Fordítógyűjtemény honlapja ➲ http://gcc.gnu.org
A GCC programhibáinak bejelentése
➲ http://gcc-bugs@gcc.gnu.org
A bazár stílusú fejlesztési modellről lásd a
➲ http://tuxedo.org/~esr/writings/cathedral-paper.html
➲ http://tuxedo.org/~esr/writings/cathedral-bazaar/
oldalakat és a CD-mellékletet!