2012. December

Informatikai Navigátor

Érdekességek programozóknak

Gondolatok a szoftverek használatáról és fejlesztéséről

7. szám

Tartalomjegyzék

Főszerkesztő: Nyiri Imre (imre.nyiri@gmail.com)

1. A programozói könyvtárak használata Linux környezetben

Ebben a kiadványban a programozók által használt eszközök áttekintése kiemelt figyelmet kap, így ebben a cikkben a legalapvetőbb eszköz, a kód ismételt felhasználását támogató programozói könyvtárak használata kerül áttekintésre. Az itt megszerzett ismeretek szükségesek lesznek a további cikkek alaposabb megértéséhez is.

A programozói könyvtár fogalma

A programok forráskódja szinte mindig több fájlból áll, amiket külön-külön le kell fordítani, majd az így kapott bináris fájlokat (ezek az *object fájl*, aminek a kiterjesztése Linuxon *.o*) össze kell kapcsolni (linkelni) futtatható programmá. A több forrásfájl használata 2 szempontból is szükséges:

1. A program kezelhető méretű darabokra való szétszedése
2. A már máshol használt függvények könnyű újrahasznosíthatósága

Nézzünk erre mindjárt itt az elején egy példát! Az 1. és 2. Programlisták *C* függvényeket tartalmaznak, a 2 forrásfájlban összesen 5 darabot.

```
// 1. Programlista (rutinok-1.c)

#include <stdio.h>

void fv1(char *s)
{
    printf( "\nCalled fv1: %s", s );
}

void fv2(char *s)
{
    printf( "\nCalled fv2: %s", s );
}

void fv3(char *s)
{
    printf( "\nCalled fv3: %s", s );
}
```

```
// 2. Programlista (rutinok-2.c)

#include <stdio.h>

void gv1(char *s)
{
    printf( "\nCalled gv1: %s", s );
}

void gv2(char *s)
{
    printf( "\nCalled gv2: %s", s );
}
```

A 3. Programlista egy főprogram, ahol az 5 függvény közül 3-at használunk is.

```
// 3. Programlista (main.c)

#include <stdio.h>
#include <stdlib.h>

#include "rutinok.h"

int main(int argc, char** argv)
{
   fv1("aaaaaa");
   fv2("aaaaaa");
   gv2("aaaaaa");
   return (EXIT_SUCCESS);
}
```

A következő feladat a futtatható program előállítása, amihez itt most a *gcc*-t használjuk. Adjuk ki ezt a parancsot:

```
gcc main.c rutinok-1.c rutinok-2.c -o test
```

Ekkor létrejön egy *test* nevű futtatható program, amit a *./test* paranccsal futtatva ezt kapjuk:

```
Called fv1: aaaaaa
Called fv2: aaaaaa
Called gv2: aaaaaa
```

Itt mindent a kiinduló forrásfájlokból állítottunk elő, azonban ez több szempontból sem a legjobb megoldás, ugyanis, ha ezek közül valamelyik nem változott, akkor fölösleges ismételten újrafordítani. A következő parancsok emiatt a *rutinok-1.c* és *rutinok-2.c* fájlokat egyenként fordítják le (a *-c* kapcsoló csak fordítást kér), aminek az eredménye a *rutinok-1.o* és *rutinok-2.o* object fájlok keletkeznek:

```
gcc -c rutinok-1.c
gcc -c rutinok-2.c
```

Ezután a program összeépítése már így is lehetséges:

```
gcc main.c rutinok-1.o rutinok-2.o -o test2
```

A *./test2* program futása természetesen ugyanazt csinálja, mint a *./test*. Mi ennek az előnye? A 4. Programlista pirossal kiemelt sora új, azonban a másik 2 forrásfájl változatlan, így láthatjuk, hogy csak a *main.c* fordítása szükséges, majd egyből építhetjük a binárisokból a programot:

```
// 4. Programlista (main.c)

#include <stdio.h>
#include <stdlib.h>

#include "rutinok.h"

int main(int argc, char** argv)
{
    fv1("aaaaaa");
    fv2("aaaaaa");
    gv2("aaaaaa");
    fv3("333333");
    return (EXIT_SUCCESS);
}
```

Fordítsuk le újra:

```
gcc main.c rutinok-1.o rutinok-2.o -o test2
```

A futási kép a következő lesz, azaz a megváltoztatott program előállt:

```
Called fv1: aaaaaa
Called fv2: aaaaaa
Called gv2: aaaaaa
Called fv3: 333333
```

A statikus library

A fent bemutatott módszer remekül működik, azonban a keletkezett object fájlok nagy száma és külön-külön kezelése egy idő után nagy terhet jelentene a fejlesztők számára. Jó lenne tematikusan, valamilyen rendezőelvek betartása mentén az így összetartozó object fájlokat is becsomagolni egy-egy tároló fájlba. Ennek a megvalósítását hívjuk programozói könyvtárnak (*library*). Történelmileg úgy alakult, hogy erre a Unix *ar* (archive) parancsát használjuk, amiről részletesen itt olvashatunk: `http://linux.about.com/library/cmd/blcmdl1_ar.htm`. Ennyit azonban nem kell tudni róla, mert mindig a következő példában mutatott módon használjuk:

```
ar rcs librutinok.a rutinok-1.o rutinok-2.o
```

Ezzel létrejön egy *librutinok.a* (statikus használatra tervezett) könyvtár. Nézzük meg a Unix *nm* parancsával (`http://linux.about.com/library/cmd/blcmdl1_nm.htm`) a kivonatolt tartalmát!

```
nm librutinok.a

rutinok-1.o:
00000000 T fv1
0000001c T fv2
00000038 T fv3
         U printf

rutinok-2.o:
00000000 T gv1
0000001c T gv2
         U printf
```

Látható, hogy mindkét object fájl ott van, sőt még az elérhető globális rutinok (függvények) neveit is láthatjuk. A *T* azt jelenti, hogy ez egy olyan név, aminek a kódja meg is van az object file-ban. Az *U* a *printf* függvényre való undefined, azaz fel nem oldott külső hivatkozást jelenti. A *gcc* a linkelés (azaz a kód darabkák összeépítése során) a *printf* függvényt máshol fogja megkeresni, azaz nem itt van megadva, de innen használják. Ezt persze mi is tudjuk, hiszen mi írtuk a forráskódot. Rendelkezünk egy

bináris *librutinok.a* könyvtárral, építsük össze a programot most ennek a használatával!

```
gcc main.c -L. -lrutinok -o test3
```

A *-L* kapcsoló után egy fájlrendszerbeli keresési utakat adhatunk meg, az összekapcsolásnál innen próbálja meg a linker megtalálni a használt külső, globális függvényeket. A *-l* a *librutinok.a* könyvtárat jelöli ki, itt az a névkonvenció, hogy a kezdő *lib* és *.a* kiterjesztés mindig elmarad. A *-o* pedig a már megszokott módon azt mondja meg, hogy a legyártandó bináris programunk neve *test3* legyen. A *test3* persze bitre pontosan ugyanaz, mint a korábbi *test2*. A statikus könyvtárakat úgy kell kezelni, mint általában minden archívumot, azaz kiegészíthetjük, törölhetünk belőle. Az 5. Programlista egy új object fájl forrása.

```
// 5. Programlista (matek-rutinok.c)

int osszeg(int a, int b)
{
    return a + b;
}

int kulonbseg(int a, int b)
{
    return a - b;
}
```

Fordítsuk le:

```
gcc -c matek-rutinok.c
```

A keletkezett *matek-rutinok.o* fájlt tegyük be a *librutinok.a* könyvtárunkba:

```
ar r librutinok.a matek-rutinok.o
```

Az új főprogramunk a következő, ahol pirossal jelöltük a változást:

```
// 6. Programlista (matek-rutinok.c)

#include <stdio.h>
#include <stdlib.h>

#include "rutinok.h"

int main(int argc, char** argv)
{
   fv1("aaaaaa");
   fv2("aaaaaa");
   gv2("aaaaaa");
   fv3("333333");
   int e = kulonbseg(12, 4);
   printf("\nA %i és %i különbsége: %i", 12, 4, e);
   return (EXIT_SUCCESS);
}
```

Készítsük el az új *test4* bináris programot:

```
gcc main.c -L. -lrutinok -o test4
```

A *./test4* futási eredménye a következő lesz:

```
Called fv1: aaaaaa
Called fv2: aaaaaa
Called gv2: aaaaaa
Called fv3: 333333
A 12 és 4 különbsége: 8
```

Most az *nm* paranccsal nézzünk bele az új könyvtárba:

```
nm librutinok.a

rutinok-1.o:
00000000 T fv1
0000001c T fv2
00000038 T fv3
         U printf

rutinok-2.o:
00000000 T gv1
0000001c T gv2
         U printf

matek-rutinok.o:
0000000d T kulonbseg
00000000 T osszeg
```

Befejezésül nézzük meg azt is, amikor több könyvtárból szedi össze a *gcc* az összeépítendő programot. Ehhez a *matek-rutinok.c* forrást egészítsük ki a következő függvénnyel:

```
#include <math.h>

#define PI 3.14159265
...
double sinWithDeg(double x)
{
    return sin( PI*x/180 );
}
```

Itt a *C* standard matematikai csomagját is használjuk, mert a *sin(x)* a *libm.a* könyvtárban található. A mi rutinunk a fokban való számítást valósítja meg az eredeti radián helyett. Készítsük el az új object fájlt:

```
gcc -c matek-rutinok.c
```

Cseréljük erre a *librutinok.a* fájlban lévő változatot:

```
ar r librutinok.a matek-rutinok.o
```

Építsük össze az új *test5* programot, ami futáskor kiegészül a *0.500000* érték megjelenítésével.

```
gcc main.c -L. -lrutinok -lm -o test5
```

Felhívjuk a figyelmet, hogy itt minden jól működik, a kívánt sinus érték jelenik meg és több könyvtárat (a matematikai és a saját rutinok) is használtunk. Azonban valamit elrontottunk! A Unix *ldd* parancs (`http://linux.about.com/library/cmd/blcmdl1_ldd.htm`) kiírja egy futtatható programról, hogy milyen dinamikus könyvtárakat használ, nézzük meg!

```
ldd test5
        linux-gate.so.1 =>  (0x00110000)
        libm.so.6 => /lib/i386-linux-gnu/libm.➥
            so.6 (0x002f1000)
        libc.so.6 => /lib/i386-linux-gnu/libc.➥
            so.6 (0x00111000)
        /lib/ld-linux.so.2 (0x00922000)
```

Ez pedig nem jó, mert mi a statikus változatot szerettük volna a példában linkelni. Ehhez a *static* kapcsoló használatát felejtettük el kiadni, azaz a fordítás és linkelés helyes parancsa ez lett volna:

```
gcc -static main.c -L. -lrutinok -lm -o test5
```

Most nézzük meg az előzőeket:

```
ldd test5
        not a dynamic executable
```

Ez már rendben van, a futás is sikeres. A bináris könyvtárak legtöbbször statikus (*.a*) és dinamikus (*.so=shared object*) változatban is elérhetőek, így jogos, hogy választanunk kell közöttük. Az osztott változat használata az alapértelmezett, általában annak külön oka van, ha mégis a futtatható programhoz akarunk statikusan linkelni. Ez ma már egy ritkábban használt, régebbi módszer.

Az osztott használatú library

A dinamikus, futás során történő linkelésnek több előnye is van, amiből általában hármat mindig meg szoktak említeni:

1. Több bináris programhoz is ugyanazt az egy könyvtárat használjuk, így nem kell mindegyikhez külön hozzászerkeszteni. Ezzel operatív memóriát (persze lemez tárhelyet is) takarítunk meg, ugyanis ugyanaz a kód már nem töltődik be többször, míg a statikus szerkesztésű programoknál erre nincs ráhatásunk.

2. Verziókezelés. A program egyes komponensei fizikailag is több bináris fájlban jelennek meg, így az egyes könyvtárak cseréje is elég, amikor a programot frissítjük.

3. Unix alatt használhatunk szimbolikus linkeket a fájlokra, ami azok alias nevei. Amennyiben N darab program használja a *libKonvytar.so* library-t és annak elkészítjük egy új változatát, akkor megtehetjük, hogy azt így hívjuk: *libKonyvtarMutans.so*. Persze a főprogram továbbra is a *libKonvytar.so* névhez ragaszkodik, de létrehozhatunk egy ilyen nevű linket a mutáns változatra. A dinamikus linkelés keresési sorrendjével pedig biztosítható, hogy ez a link kerüljön először megtalálásra. Mit nyertünk? Képesek lettünk arra, hogy az új környezetben már csak N-1 darab program használja az eredeti könyvtárat, míg az N. már a mutáns változat szerint működik.

A továbbiakban a 3 forráskönyvtár tartalma alapján készítsünk egy *.so* könyvtárat! Ehhez az alábbi módon újra el kell készíteni az object fájlokat, ugyanis ilyenkor az *fPIC* (position independent code) kapcsolót meg kell adni:

```
gcc -fPIC -c rutinok-1.c
gcc -fPIC -c rutinok-2.c
gcc -fPIC -c matek-rutinok.c
```

A lefordított object fájlokból így tudjuk elkészíteni a *librutinok.so.1.0.1* fizikai bináris fájlt, amit valódi vagy fizikai so névnek is nevezünk.

```
gcc -shared -Wl,-soname,librutinok.so.1 -o ➥
    librutinok.so.1.0.1 rutinok-1.o rutinok-2.➥
    o matek-rutinok.o
```

Ezt követően kiadunk 2 szimbolikus link készítő parancsot. A *librutinok.so.1* alias név az un. so név. A futó program ténylegesen ezt fogja meghivatkozni. A számot (esetünkben most 1) a főverziónak nevezzük. A hivatkozott fizikai név változhat alatta, emiatt abban a további 2 szám az alverzió és a release szám. Mindez persze nem kötelező, de ajánlott:

```
ln -s librutinok.so.1.0.1 librutinok.so.1
```

A 2. szimbolikus link a *so* névre mutat és csak az a feladata, hogy a futtatható programban a *-l* kapcsoló után legyen egy olyan név, amit *lib* előtaggal és *.so* kiterjesztéssel fájlnévre lehet asszociálni, azaz esetünkben ez írható: *-lrutinok*.

```
ln -s librutinok.so.1 librutinok.so
```

Miután legyártottuk a könyvtárat és a kulturált és rugalmas használatot biztosító 2 linket, felmerül a következő kérdés: Hogyan telepítsük? A helyes telepítés mindig a következő 2 igény megfelelő kielégítését jelenti:

1. A program összeépítése során a linker megtalálja a hivatkozott osztott könyvtárakat
2. A program futtatása során a linker megtalálja és betölthesse a hivatkozott osztott könyvtárakat

Látható, hogy itt nagy hangsúly van a keresésen, aminek sorrendje az alábbi szabályok szerint alakul:

1. Az *LD_LIBRARY_PATH* környezeti változóban felsorolt könyvtárak
2. A */etc/ld.so.cashe* fájlba felvett könyvtárakban
3. */usr/lib*
4. */lib*

Amennyiben a 3. vagy 4. helyre másoljuk a 3 *so* fájlunkat, úgy a linkelés és futtatás is megtalálja automatikusan a könyvtárat. Ez azt jelenti, hogy ilyenkor így állíthatjuk elő a programot:

```
gcc main.c -lrutinok -lm -o test6
```

A többi esetben a program előállításához mindig meg kell adni azt a könyvtárat, ahol az *so* fájljaink vannak. Mi a példában ide másoltuk a fájlt és a 2 linkjét: */usr/local/lib/creedsoft-org/*. Ekkor a fordítás:

```
gcc main.c -L/usr/local/lib/creedsoft-org -➥
    lrutinok -lm -o test6
```

A lefordított program futtatásához 1. vagy 2. keresést kell alkalmazni. A 2. esetben a */etc/ld.so.conf* fájlba fel kell venni a */usr/local/lib/creedsoft-org/* könyvtárat, majd futtatni ezt a parancsot, ami frissíti az *ld.so.cashe* file-t, azaz bejegyzi a mi osztott könyvtárunk elérhetőségét is:

```
sudo ldconfig -n /usr/local/lib/creedsoft-org
```

Dinamikus könyvtárhasználat

A fentiek kiegészítésképpen szeretnénk röviden bemutatni azt a lehetőséget, amikor a fordítás során nem kapcsoljuk a programhoz az osztott library-t, azonban futás közben azt mégis használjuk. Erre mutat egy példát a 7. Programlista.

```
// 7. Programlista (main.c)

#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>

int main(int argc, char** argv)
{
    void *handle;
    double (*sinus)(double);

    char *error;
    handle = dlopen("/lib/i386-linux-gnu/libm.➥
        so.6", RTLD_LAZY);
    if (!handle) {
        fputs(dlerror(), stderr);
        exit(1);
    }
    sinus = dlsym(handle, "sin");
    if ((error = dlerror()) != NULL) {
        fputs(error, stderr);
        exit(1);
    }
    printf("%f\n", (*sinus)(2.0));
    dlclose(handle);
}
```

A programot egyszerűen csak így kell lefordítani, miután a *test8* binárist kapjuk:

```
gcc -o test8 main.c -ldl
```

A *dlfcn.h* deklaráción keresztül használhatjuk a *dlopen()* függvényt, ami betölti azt az *so* tartalmat, ami a *sin* könyvtári függvényt tartalmazza. A *sinus* név pedig egy függvényre mutató pointer. A C nyelv ismeretében ennyi már elég is a megértéshez.

Mikor érdemes a dinamikus betöltést használni? A pluginok és modulok esetén. Képzeljünk el egy másik könyvtárat is, ahol a *sin* szintén meg van valósítva, persze ugyanazzal a felülettel. A paramétere 1 double és azt is ad vissza. Akkor viszont csak 1 string annak a paramétere, hogy melyik *so*-t töltjük be. Ezzel pontosan olyan környezetet tudunk használni, amit az ilyen dinamikusan bepattintható program darabkák szeretnek. A Linux sok alrendszere így van kialakítva, például a PAM (Pluggable Authentication Module) is.

A header fájl szerepe

A kódkönyvtárak kialakításához szükséges, hogy azokhoz 1 vagy több C header fájlt is készítsünk. A mi *librutinok.so* könyvtárunkhoz például ezt használhatjuk:

```
// 8. Programlista (rutinok.h)

#ifndef RUTINOK_H
#define RUTINOK_H

#ifdef __cplusplus
extern "C" {
#endif

    void fv1(char *s);
    void fv2(char *s);
    void fv3(char *s);

    void gv1(char *s);
    void gv2(char *s);

    int osszeg(int a, int b);
    int kulonbseg(int a, int b);
    double sinWithDeg(double x);

#ifdef __cplusplus
}
#endif
#endif  /* RUTINOK_H */
```

A C nyelvben fontos, hogy a kódgenerálás során legyen elég ismeretünk a használt dolgokról, még akkor is, ha nem ott implementáljuk. Amikor meghívjuk a *sinWithDeg()* függvényt, tudnunk kell, hogy az milyen paraméterezésű, mi neve és a visszatérési értéke. Ezt deklarálni kell a használat előtt, erre szolgálnak a header (vagy fejléc) fájlok, hogy ne kelljen mindig, minden esetben ezt a programozónak végeznie. Érdemes a fejlécfájlokat a 8. Programlistán látható módon formalizálni, amivel a következő 2 előnyt érjük el:

1. A header file csak 1 alkalommal lesz beemelve a *gcc* részére, mert másodszor a RUTINOK_H beállítása miatt már kimarad.

2. A C++ fordító biztosít egy *__cplusplus* flag-et, így abban az esetben az *extern "C"* megakadályozza a nevek kódolását, ami a C++ alapértelmezése, de ekkor nem találná meg a linker a C könyvtárban a keresett objektumot, hiszen ott kódolatlanul lett regisztrálva. Erre mindig tekintettel kell lennünk, amikor C és C++ kódot keverve használunk.

A könyvtár kompatibilitásának elvesztése

Csak röviden megemlítjük azokat az okokat, ami miatt a linker vagy a futtató nem tud már használni egy könyvtárat:

1. Az üzleti logika megváltozott

2. Globális (exportált) függvények törlődtek

3. Megváltozott egy vagy több függvény felülete

A fenti esetekben kötelező a főverzió növelése.

2. CMAKE – Az alkalmazások összeépítése

Tisztán emlékszünk a CMake eszközzel való első találkozásunkra. A Kdevelop 4-es sorozattal kezdtünk dolgozni, amikor a fájlok között feltűnt egy puritán kinézetű CMakeLists.txt nevű. Elkezdtük nézgetni és rájöttünk, hogy a K Desktop Environment és a Linux egyik legjobb fejlesztői környezete miért is használja. A CMake nagyon sokat tud, a fejlesztési életciklus build, teszt és csomagolás (telepítés) munkáit egyaránt hatékonyan támogatja.

A CMake egy *Cross-Platform Makefile Generator*, azaz egy egyszerű leíró szövegfájl (ennek neve általában *CMakeLists.txt*) alapján el tudja készíteni a program összeépítéséhez szükséges *Makefile*-t (ismeri ezenkívül az MS Visual Studio, Apple Xcode formátumot is).

A CMake használata

A továbbiakban bemutatjuk, hogy az első cikk forráskódjait használva miképpen foghatjuk munkára a *CMake*-et.

Készítettünk egy ilyen *CMakeLists.txt* fájlt:

```
1 cmake_minimum_required (VERSION 2.6)
2 project ( cmake-test )
3
4 add_executable( cmake-test
5  main.c rutinok-1.c rutinok-2.c
6  matek-rutinok.c
7 )
8
9 target_link_libraries(cmake-test m dl)
```

Az 1. sort minden esetben érdemes kitenni. Az érdemi rész a 2. sornál kezdődik, ahol a *project* paranccsal megmondjuk, hogy a mi projektünk most *cmake-test* névre hallgasson. A 4-7. sorok között az *add_executable* kulcsszóval rendelkezünk arról, hogy milyen forrásfájlokat akarunk a projektbe építeni. Itt az 1. cikkből ismert forrásokat látjuk természetesen, hiszen a példánk erről szól. A 9. sor *target_link_libraries* parancsával a projektünkhöz (azaz a *cmake-test*-hez) hozzátesszük a használt dinamikus könyvtárakat (látható, hogy a könyvtárak neveit ugyanúgy kell megadnunk, ahogy azt a *gcc -l* paraméterénél tettük):

- m → libm.so (matematikai könyvtár)
- dl→ libdl.so (a dinamikus betöltéshez szükséges könyvtár)

A fenti *CMakeLists.txt* fájlra futtassuk le a *cmake* parancsot:

```
cmake CMakeLists.txt

Ez a parancs képernyős kimenete:
-- Configuring done
-- Generating done
-- Build files have been written to: /home/➥
    tanulas/cpp/InfNavCProj
```

Ennek eredményeképpen létrejön a *Makefile*, amit így nem kell kézzel elkészítenünk. Futtassuk is le!

```
make

Ez a parancs képernyős kimenete:
[100%] Built target cmake-test
```

Csak emlékeztetőül, nézzük meg a *main.c* forrás utolsó változatát:

```
// 1. Programlista (main.c)

#include <stdio.h>
#include <stdlib.h>

#include <dlfcn.h>

#include "rutinok.h"

/*
 *  Test Program
 */
int main(int argc, char** argv)
{
    void *handle;
    double (*sinus)(double);

    char *error;
    handle = dlopen("/lib/i386-linux-gnu/libm.➥
        so.6", RTLD_LAZY);
```

```
    if (!handle) {
        fputs(dlerror(), stderr);
        exit(1);
    }
    sinus = dlsym(handle, "sin");
    if ((error = dlerror()) != NULL) {
        fputs(error, stderr);
        exit(1);
    }
    printf("Dinamikus sinus: %f\n", (*sinus)(2.0));
    dlclose(handle);

    fv1("aaaaaa");
    fv2("aaaaaa");
    gv2("aaaaaa");
    fv3("333333");

    int e = kulonbseg(12, 4);
    printf("\nA %i és %i különbsége: %i", 12, 4, e);

    double d = sinWithDeg( 30 );
    printf("\n %f\n", d );

    return (EXIT_SUCCESS);
}
```

A *make* parancs a projekt nevével megegyező *cmake-test* futtatható bináris programot hozta létre, futtassuk is le!

```
./cmake-test

Ez a parancs képernyős kimenete:
Dinamikus sinus: 0.909297

Called fv1: aaaaaa
Called fv2: aaaaaa
Called gv2: aaaaaa
Called fv3: 333333
A 12 és 4 különbsége: 8
 0.500000
```

Minden rendben van, ezt vártuk.

Használjunk könyvtárakat is!

A fenti kód építését most csináljuk meg úgy, hogy építünk egy *librutinok.so* könyvtárat és a *main.c* ebből használja majd a meghívandó függvényeket. Mindehhez egy kicsit átírtuk a *CMakeLists.txt* fájlt:

```
cmake_minimum_required (VERSION 2.6)
project ( cmake-test )

add_library(rutinok SHARED
        rutinok-1.c
        rutinok-2.c
        matek-rutinok.c
)

add_executable( cmake-test main.c )

target_link_libraries(rutinok m)
target_link_libraries(cmake-test dl rutinok)
```

Az igazi újdonság a 4-7. sorok között van, ahol az *add_library* paranccsal egy új *rutinok* nevű *so* könyvtár felépítését kérjük (aminek a fizikai neve persze *librutinok.so* lesz ezzel) a megadott 3 darab *C* forrásból. A 12. sor *target_link_libraries* utasítása arról rendelkezik, hogy a célja (azaz most a *rutinok)* felépítéséhez a *libm* matematikai könyvtár is szükséges. A 13. sor a futtatható *cmake-test* előállításához szükséges osztott könyvtárakat jelöli ki:

- A *dl* azért kell, mert a main.c használ belőle.
- A *rutinok* pedig a *librutinok.so* használatát definiálja, persze azért mert most már ott vannak a meghívott függvények.

A forráskód konfigurálása

A következőkben megtanuljuk, hogy a forrásprogram egyes header fájljait hogyan lehet dinamikusan konfigurálni, illetve erre való példaként megnézzük a verziókezelés egy megvalósítási lehetőségét. Előtte azonban ismerjünk meg néhány CMake elemet, ami ehhez szükséges!

CMake változók

A CMake sok előre beállított (környezeti) változóval rendelkezik, amelyekre a *CMakeList.txt* fájlban hivatkozni tudunk, ugyanakkor ilyeneket magunk is létrehozhatunk a *set* paranccsal, aminek felépítése ilyen:

set(<variable> <value-list>). Példa:

```
set(MyVar alma)
```

vagy egy lista, aminek elemei ';'-vel kerülnek elválasztásra:

```
set(MyVar alma;körte;szilva)
```

A változókra *${MyVar}* szintaxissal tudunk a *CMakeList.txt* file egyéb helyein hivatkozni. A példánkban használni fogunk 2 előre definiált CMake változót, emiatt itt megadjuk a jelentésüket:

- *PROJECT_SOURCE_DIR*: Az a mappa, ami a forráskód gyökere (top level source mappa).
- *PROJECT_BINARY_DIR*: Az a mappa, ami a bináris építés gyökere (top level build mappa).

A fentiek alapján például a *PROJECT_BINARY_DIR* változóra így tudunk hivatkozni: *${PROJECT_BINARY_DIR}*.

A configure_file parancs

A *configure_file* command egy fájlt másol, miközben megváltoztatja annak a tartalmát. Az alapvető szintaxisa így néz ki:

```
configure_file(<input> <output>)
```

Esetünkben majd a *rutinok.h* fájl végleges, verziózott alakját fogjuk így előállítani, így arra így fog kinézni a használata:

```
configure_file
(
        "${PROJECT_SOURCE_DIR}/rutinok.h.in"
        "${PROJECT_BINARY_DIR}/rutinok.h"
)
```

Az egy elfogadott konvenció, hogy egy *file.ext* output esetén a parancs input-ja *file.ext.in* legyen. Az *in* fájl készítése során a környezeti változókra *'@'* jelek között hivatkozhatunk, amire nemsokára látunk példát.

Az include_directories parancs

A make file előállításakor be kell tudni állítani, hogy a forráskód fordítása során milyen könyvtárakban keressen include fájlokat a fordító. Ez a *gcc -I* kapcsolójának felel meg. Esetünkben majd a *rutinok.h* lesz használva, aminek mappáját (pontosabban szólva szóközökkel elválasztott könyvtárlistát sorolhatunk fel) így adhatjuk meg a CMake leíróban:

```
include_directories("${PROJECT_BINARY_DIR}")
```

A példa - verzió kiírás

Ennyi előzetes ismeret már elegendő lesz, hogy a verziókezelés példát megértsük. A 2. Programlista mutatja a *rutinok.h.in* fájl tartalmát, ahol definiáltuk a *MAIN_VERSION* és *SUB_VERSION* konstansokat, amelyek értékét a CMake fogja végül megadni, amikor előállítja a fordításhoz szükséges *rutinok.h* header fájlt.

```
// 2. Programlista (rutinok.h.in)

#ifndef RUTINOK_H
#define RUTINOK_H

#ifdef __cplusplus
extern "C" {
#endif

#define MAIN_VERSION @MAIN_VERSION@
#define SUB_VERSION @SUB_VERSION@

    void fv1(char *s);
    void fv2(char *s);
    void fv3(char *s);

    void gv1(char *s);
    void gv2(char *s);

    int osszeg(int a, int b);
    int kulonbseg(int a, int b);
    double sinWithDeg(double x);

#ifdef __cplusplus
}
#endif

#endif /* RUTINOK_H */
```

A 3. Programlista a teljes és új *CMakeLists.txt* fájlt mutatja. Az előző változathoz képest már megjegyzéseket is tartalmaz, amit '#' karakterrel kezdve tudunk elhelyezni. A *Variables* résznél láthatjuk a 2 változónkat, amiket *set* paranccsal hoztunk létre. A *configure_file* előállítja a *rutinok.h.in* → *rutinok.h* transzformációt. Az *include_directories* rögzíti, hogy a megadott könyvtárban is keressen header fájlt a

fordító rendszer. A fájl többi része nem változott.

```
# 3. Programlista (CMakeLists.txt)

cmake_minimum_required (VERSION 2.6)
project ( cmake-test )

# Variables
set(MAIN_VERSION 1)
set(SUB_VERSION 0)

# Configure files
configure_file (
    "${PROJECT_SOURCE_DIR}/rutinok.h.in"
    "${PROJECT_BINARY_DIR}/rutinok.h" )

# Directories for include files
include_directories("${PROJECT_BINARY_DIR}")

# To make librutinok.so
add_library(rutinok SHARED rutinok-1.c rutinok➥
    -2.c matek-rutinok.c)

# To make an executable program
add_executable( cmake-test main.c )
target_link_libraries(rutinok m)
target_link_libraries(cmake-test dl rutinok)
```

Az alábbi főprogram csak a verzió kiírásával egészült ki:

```
int main(int argc, char** argv)
{
    printf("\nThe program version is %d.%d\n",➥
        MAIN_VERSION, SUB_VERSION);

    void *handle;
        ...
}
```

A következő képernyő szekvencia a cmake → make → futtatás parancssorozatot mutatja. A *./cmake-test* futási képét is idemásoltuk a jobb megértés kedvéért, láthatjuk a megjelenített *The program version is 1.0* szöveget.

```
cmake CMakeLists.txt
make

./cmake-test

The program version is 1.0
Dinamikus sinus: 0.909297

Called fv1: aaaaaa
Called fv2: aaaaaa
Called gv2: aaaaaa
Called fv3: 333333
A 12 és 4 különbsége: 8
 0.500000
```

Az so fájlok az alkönyvtárakban

Nagy projekt esetén érdemes annak részeit több mappába szétosztani. Az ilyen alkönyvtárakat az *add_subdirectory* paranccsal lehet megadni a CMake számára. A *librutinok.so* fájt most helyezzük a *rutinok* nevű mappába. Ehhez a projekt gyökér alatt hozzunk létre egy *rutinok* nevű mappát:

```
mkdir rutinok
```

Ezután a *CMakeList.txt* fájlt változtassuk ilyenre:

```
# 4. Programlista (CMakeLists.txt)

cmake_minimum_required (VERSION 2.6)
project ( cmake-test )

# Variables
set(MAIN_VERSION 1)
set(SUB_VERSION 0)

# Configure files
configure_file (
    "${PROJECT_SOURCE_DIR}/rutinok.h.in"
    "${PROJECT_BINARY_DIR}/rutinok.h" )

# Directories for include files
include_directories("${PROJECT_BINARY_DIR}")

# To make librutinok.so
#add_library(rutinok SHARED rutinok-1.c ➥
    rutinok-2.c matek-rutinok.c)
add_subdirectory( rutinok )

# To make an executable program
add_executable( cmake-test main.c )
#target_link_libraries(rutinok m)
target_link_libraries(cmake-test dl rutinok)
```

Az *add_library(rutinok SHARED ...)* sor megjegyzésbe került, ehelyett azt deklaráltuk az *add_subdirectory(rutinok)* sorral, hogy a *rutinok* könyvtárban is van még „teendő”, azaz egy ott elhelyezett *CMakeList.txt* fájl. A *target_link_libraries(rutinok m)* sor is kikerült a végrehajtandók közül, de nézzük csak meg a rutinok mappa alatt lévő *CMakeList.txt* fájt ugyanis átkerült oda:

```
# 5. Programlista (CMakeLists.txt)
# A rutinok mappa alatt

add_library(rutinok SHARED rutinok-1.c rutinok➥
    -2.c matek-rutinok.c)
target_link_libraries(rutinok m)
```

Természetesen a rutinok alkönyvtárba átmozgattuk a 3 darab *C* forrásfájlt is. A *cmake CMakeLists.txt* és *make* futtatása után a *librutinok.so* a rutinok, míg a *cmake-test* pedig a gyökérkönyvtárba jött létre és ugyanaz a futási eredménye, mint eddig.

Opcionális részletek a forráskódban

Lehetőségünk van az *in* fájlokban egy *#cmakedefine* utasítással megadni azt a lehetőséget, hogy a generált header fájlba egy *#define* bekerüljön vagy sem. A mi esetünkbe a *rutinok.h.in* fájlba betettük ezt:

```
#cmakedefine USE_FV1
```

Ennek hatására a generált *rutinok.h* fájlba ez a sor is létrejött:

```
#define USE_FV1
```

Amennyiben ezzel azt szeretnénk szabályozni, hogy fv1() vagy fv2() függvény hívódjon meg, így kell megváltoztatni a main.c kódot:

```
int main(int argc, char** argv)
{
    printf("\nThe program version is %d.%d\n",➥
        MAIN_VERSION, SUB_VERSION);
    ...
#ifdef USE_FV1
    fv1("aaaaaa");
#else
    fv2("aaaaaa");
#endif
        ...
}
```

A CMakeList.txt opciók

Lehetőségünk van arra, hogy a *CMakeList.txt* munkautasításait is opcionálisan választhatónak adjuk meg. Erre szolgál az *option* parancs, melynek használatára egy példa:

```
option(CMAKE_A_CASE "CMake A or B Cases" ON)
```

Itt a *CMAKE_A_CASE* az opció neve, ami *ON* vagy *OFF* állapotú lehet. Ezután a *CMAKE_A_CASE* így használható a *CMakeList.txt* fájlban:

```
if ( CMAKE_A_CASE )
        add_library(rutinok SHARED
                rutinok-1.c
                rutinok-2.c
                matek-rutinok.c)
        target_link_libraries(rutinok m)
endif ( CMAKE_A_CASE )
```

Telepítési támogatás

Általában a következő fájltípusok telepítése merül fel:

- statikus és dinamikus könyvtárak
- futtatható programok
- header fájlok
- konfigurációs fájlok

Tegyük be a *CMakeList.txt* fájlba a következő sort, majd generáljuk le a *Makefile*-t és buildeljünk a *make* paranccsal.

```
install (TARGETS cmake-test DESTINATION /home/➥
    tanulas/cpp/InfNavCProj/bin)
```

A fenti parancs lehetővé teszi a *make install* parancs kiadását, ahol a *TARGETS* mögött a telepítendő bináris, a *DESTINATION* után pedig a telepítési célkönyvtár található. Futása után a megadott könyvtárban fogjuk találni a *cmake-test* programot. Ugyanezen parancs természetesen a bináris könyvtárakra is működik. A header és konfigurációs fájlok másolására (telepítésére) egyaránt alkalmas az *install* command következő alakja:

```
install (FILES rutinok.h DESTINATION /home/➥
    tanulas/cpp/InfNavCProj/include)
```

Használhatjuk a *CMAKE_INSTALL_PREFIX* környezeti változót telepítési root helyre, ami esetünkben például */home/tanulas/cpp/InfNavCProj/* könyvtár. Ekkor elég lett volna a *bin, include, ...* megadása is destinationnak.

Tesztelési támogatás

Természetesen most is a *CMakeList.txt* fájlban kell meghatároznunk a teszteseteket. A fájl végéhez tegyük hozzá ezt a sort:

```
enable_testing()
```

Ekkor a fordítás után a *make test* parancs is használható lesz. Az egyes teszteseteket a következő sorokkal vehetjük fel:

```
add_test (Run1 cmake-test 25)
```

Itt a *Run1* a teszteset neve, a *cmake-test,* a target, amit tesztelünk. A *25* helyén egy *arg1, arg2, ...* parancssori paraméterlista lehet. A *CMake* macro lehetőséget ad a fentiekhez hasonló tesztesetek generálására is. Most futtassuk le a teszteket!

```
make test

Running tests...
Test project /home/tanulas/cpp/InfNavCProj
    Start 1: Run1
1/1 Test #1: Run1 ➥
    ..............................   Passed    ➥
    0.00 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =   0.05 sec
```

Telepítőkészlet építés

Egyszer elérkezik az az idő, amikor az alkalmazásunkat telepíthető csomag formájában terjeszteni szeretnénk. Erre Linux alatt mindenképpen valamilyen csomagformátumot érdemes választani: debian (*.deb*), Red Hat (*.rpm*). Az alábbiakban a debian csomag elkészítését mutatjuk be röviden. A *CMakeList.txt* fájl végére tegyük a következő sorokat:

```
...
# build a CPack driven installer package
include (InstallRequiredSystemLibraries)
set (CPACK_RESOURCE_FILE_LICENSE "${➥
    CMAKE_CURRENT_SOURCE_DIR}/License.txt")
set (CPACK_PACKAGE_VERSION_MAJOR "${➥
    MAIN_VERSION}")
set (CPACK_PACKAGE_VERSION_MINOR "${➥
    SUB_VERSION}")
set(CPACK_PACKAGE_CONTACT "private")
include (CPack)
```

Ez a *cmake* futtatásakor létrehozza a következő 2 fájlt:

- *CPackConfig.cmake* → bináris csomag előállítására
- *CPackSourceConfig.cmake* → source csomag előállítására

Példánkban állítsunk elő egy debian bináris csomagot! A generált *CPackConfig.cmake* file-ban tegyünk 3 változtatást, mely után így néznek majd ki ezek a sorok:

- *SET(CPACK_BINARY_BUNDLE "ON")*
- *SET(CPACK_BINARY_DEB "ON")*
- *SET(CPACK_GENERATOR "DEB")*

Ezután a már megismert 2 parancs használata szükséges:

```
make
make install
```

Végül futtassuk le a telepítő generátort, ami az installált tartalom alapján fog dolgozni, így mások gépén is ilyen fájl elrendezés fog létrejönni. Ezt kétféleképpen is megtehetjük:

```
cpack -C CPackConfig.cmake
```

Ez a *Makefile* része is, ezért így is előállíthatjuk:

```
make package
```

A fájlrendszerbe létrejött a következő debian csomagfájl: *cmake-test-1.0.1-Linux.deb.*

Kukkantsunk bele:

```
new debian package, version 2.0.
 size 3934 bytes: control archive=316 bytes.
     169 bytes,     9 lines      control
      78 bytes,     1 lines      md5sums
 Package: cmake-test
 Version: 1.0.1
 Section: devel
 Priority: optional
 Architecture: i386
 Installed-Size: 8
 Maintainer: private
 Description: cmake-test built using CMake

drwxrwxr-x root/root         0 2012-10-20 20:26 ./home➥
    /
drwxrwxr-x root/root         0 2012-10-20 20:26 ./home➥
    /tanulas/
drwxrwxr-x root/root         0 2012-10-20 20:26 ./home➥
    /tanulas/cpp/
drwxrwxr-x root/root         0 2012-10-20 20:26 ./home➥
    /tanulas/cpp/InfNavCProj/
drwxrwxr-x root/root         0 2012-10-20 20:26 ./home➥
    /tanulas/cpp/InfNavCProj/bin/
-rwxr-xr-x root/root      7759 2012-10-20 20:11 ./home➥
    /tanulas/cpp/InfNavCProj/bin/cmake-test
```

A tartalma jelenleg csak a *cmake-test* program, mert az install-t is csak erre definiáltuk a példánkban.

CMake részletesebben

A CMake egy összetett rendszer, amit eddig bemutattunk arra nagyjából mindig szükség van. Fontosságát az is mutatja, hogy a *KDevelop* környezet néhány éve már ezt a build system-et használja maga alatt. A kiterjedtségét mutatja, hogy csak az előre definiált változó kategóriáiból az alábbi 5 létezik:

- A viselkedést megváltoztató változók
- A rendszert leíró változók
- A nyelvek használatát segítő változók
- A buildelés vezérlését befolyásoló változók
- Különféle információkat szolgáltató változók

Persze a fentiek kifinomult használatára ritkán van szükség. Több környezetet is támogat, például csak pillantsunk rá a Java használatára. Legyen egy *A* class:

```
class A
{
  public A()
    {
    }

  public void printName()
    {
      System.out.println("A");
    }
}
```

És egy *HelloWorld* osztály:

```
class HelloWorld
{
    public static void main(String args[])
    {
        A a;
        a = new A();
        a.printName();
        System.out.println("Hello World!");
    }
}
```

Ehhez a következő *CMakeList.txt* készíthető:

```
project(hello Java)

cmake_minimum_required (VERSION 2.6)
set(CMAKE_VERBOSE_MAKEFILE 1)

find_package(Java COMPONENTS Development)
include (UseJava)

add_jar(hello A.java HelloWorld.java)
```

Futtassuk a *cmake* parancsot!

```
cmake CMakeLists.txt
```

A létrejött *Makefile* az operációs rendszerre előzőleg telepített Java JDK *javac* és *jar* parancsai segítségével fordít és végeredményként keletkezik egy *hello.jar* nevű java könyvtár, amibe belenézve ezt láthatjuk:

```
Archive:  /media/sda3/documents/cmake/cmake-2.8.8/➥
    Tests/Java/hello.jar

       0  Defl:N        2   0% 2012-10-20 19:20 ➥
         00000000  META-INF/
      68  Defl:N       67   2% 2012-10-20 19:20 31533➥
         a3c  META-INF/MANIFEST.MF
     480  Defl:N      325  32% 2012-10-20 19:20 ➥
         4814829a  HelloWorld.class
     377  Defl:N      264  30% 2012-10-20 19:20 ➥
         bf405ede  A.class
```

***

Befejezésül 2 fontos információ forrást szeretnénk kiemelni:

- A CMake project weboldala: `http://www.cmake.org/`.
- Mastering CMake könyv (ISBN-10: 1930934114)

3. Object Pascal – C/C++ könyvtárak használata

A Delphi fejlesztői környezetet sokan ismerik és szeretik, aminek létezik egy kiváló linuxos implementációja a Lazarus. Ez az IDE a Free Pascalt használja, ami természetesen pontosan ugyanazt az Object Pascalt valósítja meg, amit a Delphi is használ. A Lazarus ma már egy kiérlelt, stabil környezet, amivel bármilyen produktív alkalmazásfejlesztés megvalósítható. Fejlesztése 1999-ben indult, így ma már 13 éves múltra tekinthet vissza. Ebben a cikkben azt mutatjuk be, hogyan lehet ehhez a környezethez C/C++ részleteket illeszteni.

A librutinok.so elérése

Az első 2 cikkben megismertük a meglehetősen egyszerű, de a céljainknak eddig tökéletesen megfelelő *librutinok.so* könyvtárat. Időközben kiegészült 2 új, hasonlóan kicsi függvénnyel (*echo()* és *concatByC()*), így tekintsük meg a használatához szükséges aktuális header fájlt (3-1. Programlista).

```
// 3-1. Programlista: A rutinok.h

#ifndef RUTINOK_H
#define RUTINOK_H

#ifdef __cplusplus
extern "C" {
#endif

#define MAIN_VERSION 1
#define SUB_VERSION 0

//#define USE_FV1

    void fv1(char *s);
    void fv2(char *s);
    void fv3(char *s);
    void gv1(char *s);
    void gv2(char *s);
    int osszeg(int a, int b);
    int kulonbseg(int a, int b);
    double sinWithDeg(double x);
    char *echo(char *s);
    void concatByC(char d[], char *s);

#ifdef __cplusplus
}
#endif

#endif /* RUTINOK_H */
```

A 3-2. Programlista a 2 új függvény implementációját mutatja. Az *echo()* egyszerűen visszaadja a kapott karaktersorozatot. A *concatByC()* kap 2 karaktersorozatot és az elsőben visszaadja a konkatenációját. Ehhez az *sb[]* tömböt is használja.

```
// 3-2. Programlista: A rutinok-1.c 2 új függvénye

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char sb[100];
...
char *echo(char *s)
{
    return s;
}

void concatByC(char d[], char *s)
{
    strcat(d, s);
}
```

Van tehát egy osztott könyvtárunk és egy header fájl, ami az eléréséhez, használatához szükséges deklarációkat tartalmazza. A kérdés az, hogy ezt mi módon tudnánk használni Free Pascal-ból? Szerencsénk van! A rendszerhez tartozik egy *h2pas* utility, ami képes a header fájlból az Object Pascal unit-ot automatikusan legenerálni. Erre a következő parancs megfelelő:

```
h2pas -d -D -l librutinok.so -o ./rutinok.pp ../cpp/InfNavCProj/rutinok.h
```

Vegyük sorra az egyes paraméterek jelentését!

- *-d* → az *external* használat előírása, ugyanis minden függvényt a külső *librutinok.so* fájl implementál majd.
- *-D* → az external függvény neve legyen az, ami az *so*-ban van.

- *-l librutinok.so* → Ezt az *so* könyvtárat használjuk az implementációhoz.
- *-o ./rutinok.pp* → A generált Object Pascal unitnak ez lesz a neve.
- *../cpp/InfNavCProj/rutinok.h* → A generálás alapja ez a header fájl lesz.

A futtatás eredményeképpen keletkező *rutinok* unit forráskódját a 3-3. Programlista tartalmazza. Ez egy teljesen automatikusan generált kód, aminek néhány sorához szeretnénk egy kis magyarázatot fűzni. A 19. sorban elnevezte a generátor *External_library*-nak a *librutinok.so* stringet, ami a könyvtárunk fizikai neve. A 37-46 sorok között minden *procedure* és *function* (aminek a neveit felismerhetjük a header fájl tartalmából származtatva) ezzel a névvel linkeli be a külső könyvtárat. A következő 2 kulcsszó itt mindig előfordul:

- *cdecl* → Ez egy technikai különbségre utal, ami a C/C++ és Pascal eljáráshívás különbségéből adódik. Híváskor a paraméterek mindig a program STACK területére kerülnek, az eljárás onnan tudja kiolvasni azokat. A Pascal a paramétereket balról-jobbra, míg a C/C++ jobbról-balra pakolja ide. Amennyiben C/C++ rutint hívunk, akkor követnünk kell Pascalban is a jobbról-balra konvenciót, amit ez a kulcsszó jelöl meg. Ez fontos, hiszen az alprogram erre a sorrendre számít miközben futása során eléri a paraméterek értékeit.
- *external* → Ezzel lehet megadni, hogy az adott Pascal eljárás implementációja nem Pascalban, hanem egy külső helyen lesz elérhető.

Tekintettel arra, hogy a *rutinok* unit összes eljárása *external*, így a unit implementation része üres maradt.

```
1  // 3-3. Programlista: A generált rutinok unit
2
3  unit rutinok;
4  interface
5
6  {
7    Automatically converted by H2Pas 1.0.0 from ../cpp/InfNavCProj/rutinok.h
8    The following command line parameters were used:
9      -d
10     -D
11     -l
12     librutinok.so
13     -o
14     ./rutinok.pp
15     ../cpp/InfNavCProj/rutinok.h
16 }
17
18   const
19     External_library='librutinok.so'; {Setup as you need}
20
21   Type
22   Pchar  = ^char;
23 {$IFDEF FPC}
24 {$PACKRECORDS C}
25 {$ENDIF}
```

```
26
27
28  {$ifndef RUTINOK_H}
29  {$define RUTINOK_H}
30  { C++ extern C conditionnal removed }
31
32    const
33      MAIN_VERSION = 1;
34      SUB_VERSION = 0;
35    {#define USE_FV1 }
36
37    procedure fv1(s:Pchar);cdecl;external External_library name 'fv1';
38    procedure fv2(s:Pchar);cdecl;external External_library name 'fv2';
39    procedure fv3(s:Pchar);cdecl;external External_library name 'fv3';
40    procedure gv1(s:Pchar);cdecl;external External_library name 'gv1';
41    procedure gv2(s:Pchar);cdecl;external External_library name 'gv2';
42    function osszeg(a:longint; b:longint):longint;cdecl;external External_library ➥
          name 'osszeg';
43    function kulonbseg(a:longint; b:longint):longint;cdecl;external ➥
          External_library name 'kulonbseg';
44    function sinWithDeg(x:double):double;cdecl;external External_library name '➥
          sinWithDeg';
45    function echo(s:Pchar):Pchar;cdecl;external External_library name 'echo';
46    procedure concatByC(d:Pchar; s:Pchar);cdecl;external External_library name '➥
          concatByC';
47
48  { C++ end of extern C conditionnal removed }
49  {$endif}
50    { RUTINOK_H  }
51
52  implementation
53
54  end.
```

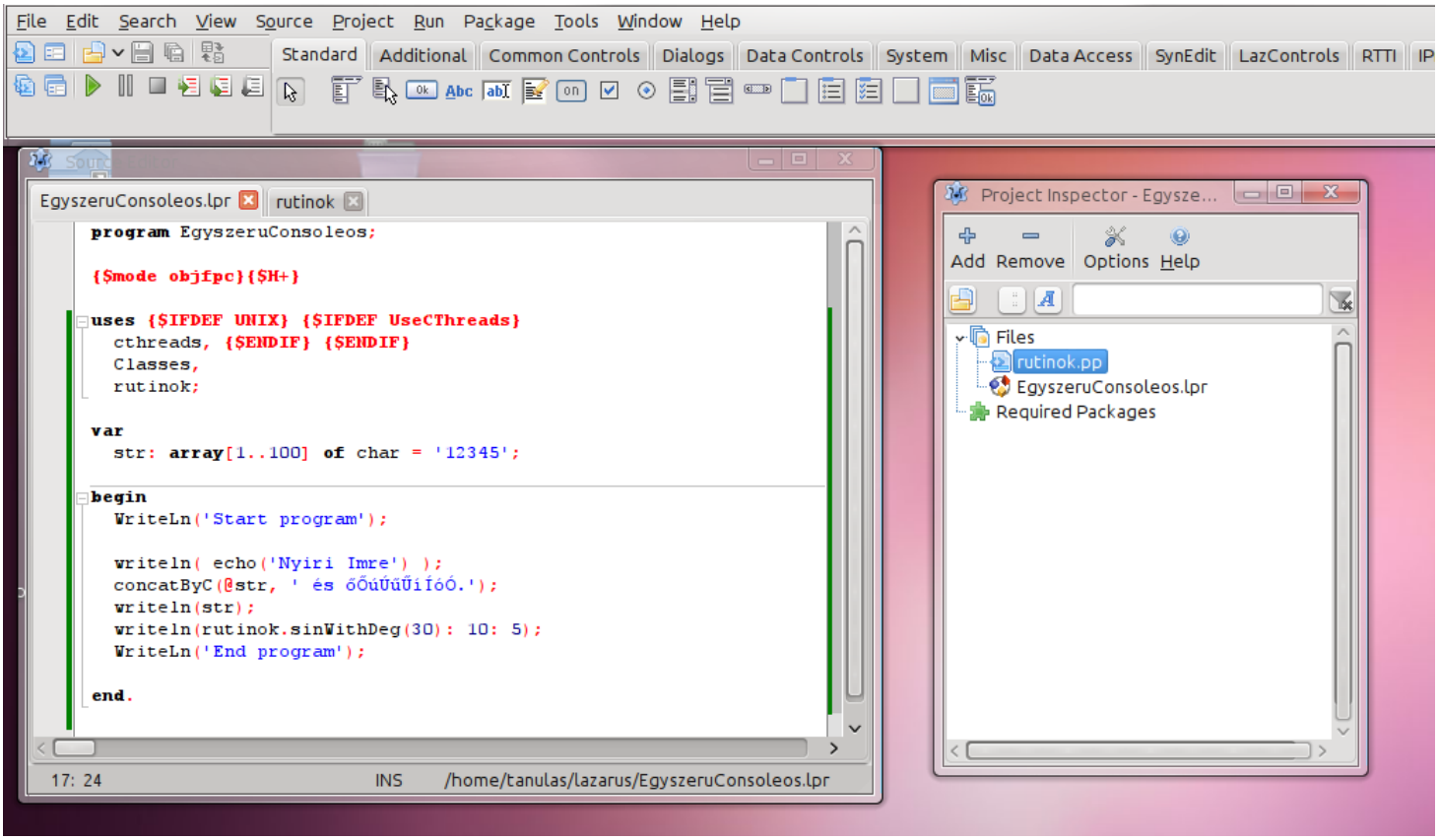
A unit kipróbálását egy egyszerű konzol programmal fogjuk elvégezni, amit a 3.1. ábra bal oldali editor ablaka mutat. Az ábra a Lazarus IDE-t jeleníti meg, ahogy a *Project Inspector* segítségével (a + gombra kattintva) beimportáltuk a *rutinok.pp* fájlt, így elérhetővé vált a unit is. Az *EgyszeruConsoleos* program nem túl bonyolult. A már ismert *sinWithDeg()* mellett az *echo()* és *concatByC()* használatát teszteli le, aminek a futási képe így néz ki:

```
./EgyszeruConsoleos

Start program
Nyiri Imre
12345 és őÕúÚűŰÜíÍóÓ.
   0.50000
End program
```

Az *echo()* tényleg visszaadja a stringet. A *concatByC()* a

```
str: array[1..100] of char = '12345';
```

helyen megadott buffert használja, majd a következő sor tényleg azt írja ki, amit vártunk. A *0.50000* érték pedig a sin(30) fok. Szeretnénk kiemelni még a program elejét is, ahol a *uses* szekciónál láthatjuk a *rutinok* unit használatának igénylését.



3.1. ábra: Lazarus - A rutinok unit importálása

Fejlettebb C szerkezetek használata

Célunk az, hogy összetettebb C nyelvi struktúrákat is kipróbáljunk Pascal program-elemként. Ehhez most nézzük meg az erre a célra elkészített header fájlt:

```
// 3-4. Programlista: pascal-rutins.h

#define TRUE 1
#define FALSE 0

typedef int Salary;
typedef int bool;

typedef struct
{
    unsigned experienced : 1;
    unsigned priority    : 1;
    unsigned candidate   : 1;
    unsigned trained     : 1;
} EmployeeFlags;

typedef struct
{
    char* name;
    int   category;
    bool  expert;
    long  salary;
    EmployeeFlags flags;
    double percent;
} Worker;

typedef struct
{
    char* name;
    int   category;
    bool  manager;
    long  salary;
    long  bonus;
    EmployeeFlags flags;
    double percent;
} Boss;

typedef union
{
    Worker w;
    Boss   b;
} Employee;
```

```
typedef enum { Spring, Summer, Autumn, Winter } Season;

typedef int (*MyFunction)(double x1, double x2);
typedef int (*PFunction)(int a, int b);
typedef int (*PFunctionArray[5])(int a, int b);
typedef int* PInt;
typedef int* IntPointerArray[7];

typedef int IntArray[7];
typedef IntArray *PIntArray;

extern int increaseWorkerSalary(Worker* w, int delta);
extern int increaseBossSalary(Boss* b, int delta);
extern int increaseSalary(Employee* e, int delta, bool isBoss);
extern bool isGoodSalary(int salary);
extern void setEmployeeFlags(Employee* e, bool isBoss, EmployeeFlags flags);
extern bool getTrained(Employee* e, bool isBoss);
extern void anOperation(Worker *w, PFunction op, int delta);
```

Aki ismeri az alap C nyelvet, annak a fenti kód úgy tűnhet, hogy itt összehordtuk a C nyelv csaknem összes elemét. Az egyszerű típusnév definíció mellett (példa: *typedef int Salary;*) láthatunk struktúra megadást: *Worker*, *Boss*. Ezek a pascal *record*-nak felelnek meg. Deklaráltunk egy *EmployeeFlags* szerkezetet, ami biteken tárol logikai értékeket. A *union* (*Employee*) a változó record. A következő sorokban többféle *pointer* típust is megalkottunk. A header fájl 7 függvény deklarációjával zárul, aminek implementációját a következő 3-5. Programlista mutatja. Nem túl bonyolúltak, ezért nem is fűzünk további magyarázatot hozzá.

```
// 3-5. Programlista: pascal-rutins.c

#include "pascal-rutins.h"

int increaseWorkerSalary(Worker* w, int delta)
{
    w->salary += delta;
    return w->salary;
}

int increaseBossSalary(Boss* b, int delta)
{
    b->salary += delta;
    return b->salary;
}

int increaseSalary(Employee* e, int delta, bool isBoss)
{
    if ( isBoss )
    {
        e->b.salary += delta;
        return e->b.salary;
    }
    {
        e->w.salary += delta;
        return e->w.salary;
    }
}

bool isGoodSalary(int salary)
{
    if (salary > 1000000 ) return TRUE;
    else return FALSE;
}

void setEmployeeFlags(Employee* e, bool isBoss, EmployeeFlags flags)
{
    if ( isBoss )
    {
        e->b.flags = flags;
    }
    {
        e->w.flags = flags;
    }
}

bool getTrained(Employee* e, bool isBoss)
{
    if ( isBoss )
    {
        return ( e->b.flags.trained != 0 );
    }
    {
        return ( e->w.flags.trained != 0 );
    }
}

void anOperation(Worker *w, PFunction op, int delta)
{
    w->salary = op(w->salary, delta);
}
```

A 3-6. Programlista az előzőekben megalkotott kód C nyelven való tesztelésére szolgál. Láthatjuk, hogy ezeket a nyelvi elemeket hogyan használjuk, de még maradunk a C nyelv keretein belül. Ez hasznos lesz, amikor mindezt már pascal nyelven fogjuk használni, ugyanis ismerős kódot kell majd ott megérteni és összehasonlíthatjuk a C-beli alkalmazással.

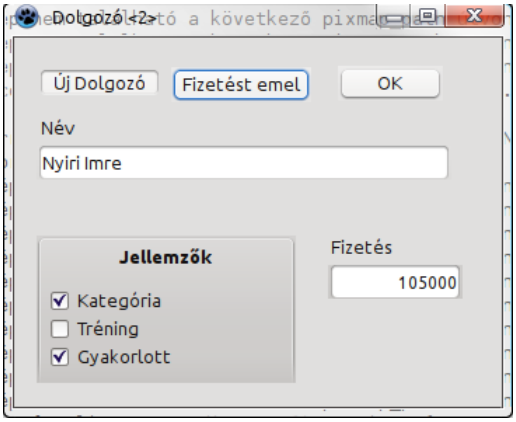
```
// 3-6. Programlista: test.c

#include "pascal-rutins.h"
```

```
int add(int a, int b)
{
    return a+b;
}

int main()
{
    printf("Hello World! Start test...\n");

    EmployeeFlags flags;
    flags.candidate=1;
    flags.experienced=0;
    flags.priority=1;
    flags.trained=1;

    Worker w;
    w.salary=10000;
    w.name="Nyiri Imre";
    w.category=1;
    w.expert=1;
    w.percent=0.56;
    w.flags=flags;

    Boss b;

    Employee e;
    e.w.category=w.category;
    e.w.expert=w.expert;
    e.w.flags=w.flags;
    e.w.name=w.name;
    e.w.percent=w.percent;
    e.w.salary=w.salary;

    printf("%i\n", isGoodSalary(1000000000));

    printf("Fizetés: %i Ft\n", w.salary);
    increaseWorkerSalary(&w, 5000);
    printf("Fizetés: %i Ft\n", w.salary);

    printf("Fizetés: %i Ft\n", e.w.salary);
    increaseSalary(&e, 3000, 0);
    printf("Fizetés: %i Ft\n", e.w.salary);

    printf("Trained: %i\n", getTrained(&e, 0))
        ;

    EmployeeFlags flags2;
    flags.candidate=1;
    flags.experienced=0;
    flags.priority=1;
    flags.trained=0;

    setEmployeeFlags(&e, 0, flags2);
    printf("Trained: %i\n", getTrained(&e, 0))
        ;

    printf( w.name );
    w.name="Másik név";
    printf( w.name );

    printf(e.w.name );

    PFunction pfunc = add;

    anOperation(&w, add, 7000);
    printf("Fizetés: %i Ft\n", w.salary);

    return 1;
}
```

A *pascal-rutins.c* kódot azért készítettük, hogy egy osztott könyvtár legyen belőle, amit most így hoztunk létre (eredménye: *libpasruts.so.1.0.1*):

```
gcc -fPIC -c pascal-rutins.c
gcc -shared -Wl,-soname,libpasruts.so.1 -o ➥
    libpasruts.so.1.0.1 pascal-rutins.o
```

A 3.2. ábra mutatja a futási képét annak a Lazarus (Object Pascal) kódnak, amit erre a könyvtárra építve készítettünk. A soron következő részekben ennek a forráskódját láthatjuk.



3.2. ábra: A TestGUIForC program

Az első és példánk szempontjából legfontosabb a *h2pas* által generált (a *pascal-rutins.h* alapján) pascal unit (3-7. Programlista). A unit-ot kézzel egy kicsit javítottuk, de alapvetően automatikusan generálódott és hasonlít az előző példában megismert szerkezethez. A 22-68 sorok között láthatjuk a C header által generált megfelelő pascal típusokat. A 75-76 sorokban eredetileg TRUE és FALSE szerepelt, de ezt át kellett nevezni, hiszen ezek foglalt szavak. A 79-86 sorok az *EmployeeFlags* kezeléséhez szükségesek. Az *implementation* rész 119-164 sorai pedig ezt valósítják meg, mert a pascal nem rendelkezik hasonló bitenkénti mező megadási lehetőséggel. Ez az oka, hogy most ebbe a szekcióba is

generálódott kód, hiszen ezt nem fogjuk megtalálni a könyvtári *so* fájlban sem. A 7 külső függvény (97-116 sorok) csak az interface megadása miatt szükséges, azaz azt kell itt megadni, hogy pascal-ból milyen felületen hívjuk a *libpasruts.so* rutinjait.

```
1  // 3-7. Programlista: A pascal_rutins unit
2
3  unit pascal_rutins;
4
5  interface
6
7  {
8    Automatically converted by H2Pas 1.0.0 from pascal-rutins.h
9    The following command line parameters were used:
10     -d
11     -D
12     -l
13     libpasruts.so.1.0.1
14     -o
15     ./pascal-rutins.pp
16     pascal-rutins.h
17 }
18
19 const
20   External_library = 'libpasruts.so.1.0.1'; {Setup as you need}
21
22 type
23
24   PInt = ^longint;
25   IntPointerArray = array[0..6] of ^longint;
26   IntArray = array[0..6] of longint;
27   PIntArray = ^IntArray;
28   Season = (Spring, Summer, Autumn, Winter);
29   Salary = longint;
30   bool = longint;
31
32   EmployeeFlags = record
33     flag0: word;
34   end;
35
36   Worker = record
37     Name: ^char;
38     category: longint;
39     expert: bool;
40     salary: longint;
41     flags: EmployeeFlags;
42     percent: double;
43   end;
44
45   Boss = record
46     Name: ^char;
47     category: longint;
```

```
48      manager: bool;
49      salary: longint;
50      bonus: longint;
51      flags: EmployeeFlags;
52      percent: double;
53    end;
54
55    Employee = record
56      case longint of
57        0: (w: Worker);
58        1: (b: Boss);
59    end;
60
61    PBoss = ^Boss;
62    PEmployee = ^Employee;
63    PWorker = ^Worker;
64
65    MyFunction = function(x1: double; x2: double): longint; cdecl;
66    PFunction = function(a: longint; b: longint): longint; cdecl;
67  //PFunctionArray = array[0..4] of function (a:longint; b:longint):longint;cdecl;
68  //PFunctionArray = array[0..4] of PFunction;cdecl;
69
70  {$IFDEF FPC}
71  {$PACKRECORDS C}
72  {$ENDIF}
73
74  const
75    XTRUE = 1;
76    XFALSE = 0;
77
78  const
79    bm_EmployeeFlags_experienced = $1;
80    bp_EmployeeFlags_experienced = 0;
81    bm_EmployeeFlags_priority = $2;
82    bp_EmployeeFlags_priority = 1;
83    bm_EmployeeFlags_candidate = $4;
84    bp_EmployeeFlags_candidate = 2;
85    bm_EmployeeFlags_trained = $8;
86    bp_EmployeeFlags_trained = 3;
87
88  function experienced(var a: EmployeeFlags): dword;
89  procedure set_experienced(var a: EmployeeFlags; __experienced: dword);
90  function priority(var a: EmployeeFlags): dword;
91  procedure set_priority(var a: EmployeeFlags; __priority: dword);
92  function candidate(var a: EmployeeFlags): dword;
93  procedure set_candidate(var a: EmployeeFlags; __candidate: dword);
94  function trained(var a: EmployeeFlags): dword;
95  procedure set_trained(var a: EmployeeFlags; __trained: dword);
96
97  function increaseWorkerSalary(w: PWorker; delta: longint): longint;
98    cdecl; external External_library Name 'increaseWorkerSalary';
99
```

```
100 function increaseBossSalary(b: PBoss; delta: longint): longint;
101   cdecl; external External_library Name 'increaseBossSalary';
102
103 function increaseSalary(e: PEmployee; delta: longint; isBoss: bool): longint;
104   cdecl; external External_library Name 'increaseSalary';
105
106 function isGoodSalary(salary: longint): bool; cdecl;
107   external External_library Name 'isGoodSalary';
108
109 procedure setEmployeeFlags(e: PEmployee; isBoss: bool; flags: EmployeeFlags);
110   cdecl; external External_library Name 'setEmployeeFlags';
111
112 function getTrained(e: PEmployee; isBoss: bool): bool; cdecl;
113   external External_library Name 'getTrained';
114
115 procedure anOperation(w: PWorker; op: PFunction; delta: longint); cdecl;
116   external External_library Name 'anOperation';
117
118
119 implementation
120
121 function experienced(var a: EmployeeFlags): dword;
122 begin
123   experienced := (a.flag0 and bm_EmployeeFlags_experienced) shr
124     bp_EmployeeFlags_experienced;
125 end;
126
127 procedure set_experienced(var a: EmployeeFlags; __experienced: dword);
128 begin
129   a.flag0 := a.flag0 or ((__experienced shl bp_EmployeeFlags_experienced) and
130     bm_EmployeeFlags_experienced);
131 end;
132
133 function priority(var a: EmployeeFlags): dword;
134 begin
135   priority := (a.flag0 and bm_EmployeeFlags_priority) shr ➥
        bp_EmployeeFlags_priority;
136 end;
137
138 procedure set_priority(var a: EmployeeFlags; __priority: dword);
139 begin
140   a.flag0 := a.flag0 or ((__priority shl bp_EmployeeFlags_priority) and
141     bm_EmployeeFlags_priority);
142 end;
143
144 function candidate(var a: EmployeeFlags): dword;
145 begin
146   candidate := (a.flag0 and bm_EmployeeFlags_candidate) shr ➥
        bp_EmployeeFlags_candidate;
147 end;
148
149 procedure set_candidate(var a: EmployeeFlags; __candidate: dword);
```

```
150 begin
151   a.flag0 := a.flag0 or ((__candidate shl bp_EmployeeFlags_candidate) and
152     bm_EmployeeFlags_candidate);
153 end;
154
155 function trained(var a: EmployeeFlags): dword;
156 begin
157   trained := (a.flag0 and bm_EmployeeFlags_trained) shr bp_EmployeeFlags_trained;
158 end;
159
160 procedure set_trained(var a: EmployeeFlags; __trained: dword);
161 begin
162   a.flag0 := a.flag0 or ((__trained shl bp_EmployeeFlags_trained) and
163     bm_EmployeeFlags_trained);
164 end;
165
166 end.
```

```
1  // 3-8. Programlista: A TestForm unit
2
3  unit TestForm;
4
5  {$mode objfpc}{$H+}
6
7  interface
8
9  uses
10   Classes, SysUtils, FileUtil, Forms, Controls, Graphics, Dialogs, StdCtrls,
11   ExtCtrls, pascal_rutins;
12
13 type
14
15   { TForm1 }
16
17   TForm1 = class(TForm)
18     btnOK: TButton;
19     Button1: TButton;
20     checkGroup: TCheckGroup;
21     edtFizetes: TEdit;
22     edtText: TEdit;
23     btnInitWorker: TToggleBox;
24     Label1: TLabel;
25     Label2: TLabel;
26     procedure btnInitWorkerChange(Sender: TObject);
27     procedure btnOKClick(Sender: TObject);
28     procedure Button1Click(Sender: TObject);
29     procedure checkGroupClick(Sender: TObject);
30   private
31     { private declarations }
32   public
33     { public declarations }
```

```
34    end;
35
36  var
37    Form1: TForm1;
38
39    dolgozo : Worker;
40    flags    :EmployeeFlags;
41
42  implementation
43
44  {$R *.lfm}
45
46  { TForm1 }
47
48  procedure TForm1.btnOKClick(Sender: TObject);
49  begin
50    edtText.Text := 'Alma';
51  end;
52
53  procedure TForm1.Button1Click(Sender: TObject);
54  var
55        s : string;
56  begin
57    increaseWorkerSalary(@dolgozo, 5000);
58    str(dolgozo.salary, s);
59    edtFizetes.Text := s;
60  end;
61
62  procedure TForm1.checkGroupClick(Sender: TObject);
63  begin
64
65  end;
66
67  procedure TForm1.btnInitWorkerChange(Sender: TObject);
68  var
69        s : string;
70  begin
71        dolgozo.category:=1;
72        dolgozo.name:='Nyiri Imre';
73        dolgozo.salary:=100000;
74        dolgozo.expert:=XTRUE;
75        dolgozo.percent:=56.5;
76        set_candidate(flags, 1);
77        set_experienced(flags, 1);
78        set_priority(flags, 0);
79        set_trained(flags, 0);
80        dolgozo.flags:=flags;
81        edtText.Text := dolgozo.name;
82        str(dolgozo.salary, s);
83        edtFizetes.Text := s;
84        checkGroup.Checked[0]:= (dolgozo.category=1);
85        checkGroup.Checked[1]:= trained(flags)=1;
```

```
86        checkGroup.Checked[2]:= experienced(flags)=1;
87  end;
88
89  end.
```

A 3.2. ábra formját a 3-8. Programlista tartalmazza, a 11. sorban láthatjuk a *pascal_rutins* unit használatát is. A figyelmet a *btnInitWorker* és a *Button1* gomb eseménykezelőire szeretnénk felhívni. Az első a 67-86 sorok között van kifejtve, a program futása során ezt a gombot nyomtuk meg először, ami feltölt egy *dolgozo* record-ot (a 39. sorban definiáltuk). A futás során a 2. lépés a *Button1* gombon való click volt, a kódját az 53-60 közötti soroknál láthatjuk, azaz a dolgozó fizetését 5000 egységgel emeltük meg.

C++ osztályok használata

A C++ közvetlen illesztése nem triviális, ugyanis az objektum nevek átkódoltan kerülnek be az object fájlba. Ennek oka természetesen következik a C++ lehetőségeiből, de itt ezt most nem részletezzük.

```
// 3-9. Programlista: Test.h

class Test
{
public:
    int egyop(int a, int b);
};
```

```
// 3-10. Programlista: Test.cpp

#include "Test.h"

int Test::egyop(int a, int b)
{
  return a+b;
}
```

A 3-9. és 3-10. listák egy *Test* nevű osztályt adnak meg. A 3-11. és 3-12. listák pedig egy segéd C++ modult, ahol *extern "C"* utasítással kértük, hogy a *burkolo* név ne változzon meg az object fájlban, így a pascal is könnyen össze tud majd ezzel szerkesztődni. A *libcs-lib.so* a *Test.cpp* és *burkolo.cpp* fájlok fordításával keletkezett.

```
// 3-11. Programlista: burkolo.h

extern "C"
{
    int burkolo(int a, int b);
}
```

```
// 3-12. Programlista: burkolo.cpp

#include "Test.h"
#include "burkolo.h"

extern "C"
{
    int burkolo(int a, int b)
    {
        Test test;
        return test.egyop(a, b);
    }
}
```

Végül a 3-13. Programlista a generált pascal unitot mutatja be, ami modulként tud kapcsolódni a pascal forráskód más elemeihez.

```
// 3-13. Programlista: A burkolo unit

unit burkolo;
interface

  const
    External_library='libcs-lib.so'; {Setup as you need}

{$IFDEF FPC}
{$PACKRECORDS C}
{$ENDIF}

  function burkolo(a:longint; b:longint):longint;cdecl;external
      External_library name 'burkolo';

implementation
end.
```

4. C# – C/C++ könyvtárak használata

A *C#* nyelv és a .NET környezet a Microsoft fejlesztése, de létezik hozzá jó minőségű linuxos implementáció, a Mono. Az eddigiek folytatásaként most azt vizsgáljuk meg, hogy a *C#* programjaink számára miként tehetjük elérhetővé a C/C++ nyelven készült kódokat. Ehhez egy *SWIG* (*Simplified Wrapper and Interface Generator*) nevű eszközt fogunk használni ami egy igazi svájci bicska az ilyen típusú feladatok megoldásához. Webhelye: `http://www.swig.org/`. Linux, OS-X/Darwin és MS Windows környezetek alatt egyaránt használható.

Példáinkban a C/C++ nyelven megírt programrészek más számítógépes nyelvekhez, platformokhoz való integrálását mutatjuk be, eközben így azt is megtanuljuk, hogy egy létező bináris könyvtárat hogyan lehet illeszteni a célkörnyezethez. Ebben a cikkben a C# nyelvhez való illesztés példáin keresztül adunk erre ízelőt. A *swig* 1996 óta fejlődik, magas tudású szoftver, amit – jellegénél fogva – elsősorban rendszerintegrációs és/vagy prototípus készítési feladatokhoz lehet hatékonyan használni. Még ma is intenzíven fejlesztik, amit innen is áttekinthetünk: `http://www.swig.org/history.html`. Ismeri az ANSI C/C++ szintaxist, így használata meglehetősén széles skálán lehetséges.

A librutinok.so használata

Az eddigi hagyományainknak megfelelően most is nézzük meg az 1. cikkben elkészített *librutinok.so* használatát, persze most C# környezetből. Mindehhez a *swig* eszközt hívjuk segítségül, aminek a használata alapszinten könnyű, bár igen kifinomult megoldásokat is lehetővé tesz. Ennek első lépése egy *swig* konfigurációs fájl (aminek a kiterjesztése a konvenció szerint *.swg* vagy *.i* szokott lenni) készítése:

```
/* rutinok.swg */

%module Rutinok
%{
    #include "rutinok.h"
%}

int osszeg(int a, int b);
int kulonbseg(int a, int b);
double sinWithDeg(double x);
char *echo(char *s);
```

Láthatjuk, hogy ez egyszerű. A module neve *Rutinok* lesz, ami egyben a C#-ból használt osztály neveként is feltűnik majd. Van egy automatizmus, ami ilyenkor a hozzá tartozó *.so* fájlt is ilyen néven keresi, ezért az eredeti *librutinok.so*-ra egy *libRutinok.so* linket is tettünk (az is lehetett volna, hogy a module neve rutinok lesz, de a kis betűs class név nem szép). A *%{...%}* között a használt header fájl nevét kell megadnunk, majd felsorolni azokat a függvényeket, amire illesztő felületet kérünk és ezzel bekerül a generálandó *Rutinok.cs* C# fájlba. A fentiekben látható, hogy itt most 4 ilyen meghívható nevet jegyeztünk be. Ennyit kell mindösszesen tennünk és már a következő paranccsal hívhatjuk is a *swig*-et:

```
swig -csharp rutinok.swg
```

Az *swg* fájl független a generálandó célnyelvtől, ami egy nagyon kiváló tulajdonság. Ugyanazt a leírót használhatjuk C#, Java, PHP, Python, Perl, R, Ruby, Tcl/Tk esetén is. A *-csharp* kapcsolóval természetesen most C# illesztőt kérünk, ami esetünkben az alábbi 3 fájl legenerálását eredményezte:

- *rutinok_wrap.c*
- *RutinokPINVOKE.cs*
- *Rutinok.cs*

A *Rutinok.cs* tartalmazza azt a felületet, amit közvetlenül használni fogunk, a 4-1. Programlista mutatja a tartalmát. A *rutinok_wrap.c* és *RutinokPINVOKE.cs* fájlok belsejével nem kell törődnünk, csak használni kell őket. Az elsőt le kell fordítanunk és be kell tenni az *.so* lib-be. Amennyiben nem lenne *libRutinok.so* könyvtárunk, akkor esetünkben ezt a 2 lépést kell csinálni:

```
gcc -c -fPIC rutinok-1.c rutinok-2.c matek-➥
    rutinok.c rutinok_wrap.c
gcc -shared rutinok-1.o rutinok-2.o matek-➥
    rutinok.o rutinok_wrap.o -o libRutinok.so
```

```
// 4-1. Programlista: Rutinok.cs

/* ----------------------------------------
 * This file was automatically generated
 *       by SWIG (http://www.swig.org).
 * Version 2.0.7
 *
 * Do not make changes to this file unless
 * you know what you are doing--modify
 * the SWIG interface file instead.
 * ---------------------------------------- */

using System;
using System.Runtime.InteropServices;

public class Rutinok {
  public static int osszeg(int a, int b) {
    int ret = RutinokPINVOKE.osszeg(a, b);
    return ret;
  }

  public static int kulonbseg(int a, int b) {
    int ret = RutinokPINVOKE.kulonbseg(a, b);
    return ret;
  }

  public static double sinWithDeg(double x) {
    double ret = RutinokPINVOKE.sinWithDeg(x);
    return ret;
  }

  public static string echo(string s) {
    string ret = RutinokPINVOKE.echo(s);
    return ret;
  }
}
```

Felhívjuk a figyelmet arra, hogy a *Rutinok* class valóban csak a kért 4 darab metódust tartalmazza. Abban az esetben, ha egy header fájl összes függvényére szeretnénk illesztést kérni, akkor azok felsorolása helyett a lenti 2. változatú *swg* fájl is használható, azaz a felsorolás helyén csak megadjuk újra az include fájl nevét.

```
/* rutinok.swg - 2. változat */

%module Rutinok
%{
    #include "rutinok.h"
%}

%include "rutinok.h"
```

Kész vagyunk az illesztéssel, ezért írjunk egy C# tesztprogramot, ami használja is azt. Ennek a neve *runme.cs* lesz (4-2. Programlista).

```
// 4-2. Programlista: runme.cs

using System;

public class runme {

static void Main() {
 Console.WriteLine( Rutinok.osszeg(34, 56) );
 Console.WriteLine( Rutinok.sinWithDeg(30) );
 Console.WriteLine( Rutinok.kulonbseg(34, 56) ➥
     );
 Console.WriteLine( Rutinok.echo("Hello echo!"➥
     ) );
}
}
```

A teszt forráskódja triviális, további megjegyzéseket nem fűzünk hozzá. A következő lépés a fordítás és futtatás. Az alábbi utasítás a *mono* (egyik) *C#* fordítójával lefordít minden *cs* kiterjesztésű forrást az aktuális könyvtárban, majd a generált kódot a *runme.exe* bináris fájlba helyezi el.

```
dmcs -out:runme.exe *.cs
```

A futtatási kép így néz ki:

```
./runme.exe

90
0,499999999481858
-22
Hello echo!
```

Láthatjuk, hogy a programot egyszerűen csak a parancssorba írva futtathatjuk. A *sinWithDeg(30)* eredményével kapcsolatosan 2 fontos dolgot szeretnék kiemelni:

- Nem 0.5000 a kiírt érték, mert most semmit sem tettünk a *double* érték kerekítése ügyében.

- A metódus – remélhetőleg még emlékszünk erre – a *libm.so* matematikai könyvtárat is használja, azaz láthatóan ez is feloldásra került a futás során.

Jelen példánkat azzal szeretnénk zárni, hogy a *rutinok.swg* 2. változata helyett közvetlenül is használhatjuk a *rutinok.h* header fájlt, azaz teljes import esetén valójában nem is kell *swig* konfigurációs leírást készítenünk.

```
swig -csharp -module Rutinok rutinok.h
```

A C és C++ együtt

A továbbiakhoz jól fog jönni annak a megértése, hogy a C és C++ nyelveket hogyan tudjuk együtt használni. Ehhez tekintsük az alábbi kis C programot.

```
// elso.c

int  osszead(int a, int b)
{
    return a+b;
}
```

Majd nézzük meg az őt használó C++ forrásfájlt is:

```
// hello.cpp

#include <iostream>

extern int osszead(int, int);

int main()
{
  std::cout << "Hello World!" << std::endl;
  std::cout << osszead(34, 6);
  return 0;
}
```

A *hello.cpp* C++ kód használja az *osszead()* függvényt, emiatt *extern* módosítóval be kell őt mutatni (deklarálni kell, amit általában header fájlban teszünk, de most eltekintünk ettől). Most fordítsuk le mindkettőt!

```
gcc -c elso.c
g++ -c hello.cpp
```

Kaptunk 2 object fájlt, szerkesszük össze őket egy *progi.exe* futtatható programmá!

```
g++ hello.o elso.o -o progi.exe

hello.o: In function `main':
hello.cpp:(.text+0x3d): undefined reference to
    `osszead(int, int)'
collect2: error: ld returned 1 exit status
```

Hoppá! Ez nem ment, nem találja az *osszead(int, int)* függvényt a *hello.cpp*. Az *elso.c* egy C program C fordítóval (*gcc*), míg a *hello.cpp* egy C++ kód C++ fordítóval (*g++*) fordítva. Nézzük meg az *elso.o* fájl belsejét:

```
nm elso.o

00000000 T osszead
```

Majd a *hello.o* fájlból egy részletet is:

```
U osszead(int, int)
U std::ostream::operator<<(int)
U std::ostream::operator<<(std::ostream& (*)(
    std::ostream&))
U std::ios_base::Init::Init()
U std::ios_base::Init::~Init()
U std::cout
```

Valóban láthatjuk, hogy C++ esetén az object fájlban az *osszead()* neve kódolt abban az értelemben, hogy a paraméter típusok nevei is belekerülnek abban a névbe, ahogy azt elnevezi a szimbólumtáblában. Ekkor persze hiába mondtunk *extern*-t, ilyen nevűt nem fog találni az *elso.o* fájlban, ugyanis azt C fordítóval csináltuk, ami ilyesmi kódolással nem foglalkozik. A C++ azért kódolta el a függvény nevét, hogy a forráskódban több ilyen nevű is lehessen, amennyiben azok paraméterei eltérnek. Ugyanakkor ez a többértelműség az object fájlban már nem lesz meg, ahogy láthattuk. Mi akkor a megoldás erre az esetre? A C++ *extern* kulcsszónak van egy módosult alakja: *extern „C”*. Használjuk ezt, ahogy a javított *hello.cpp* tartalmazza:

```
// hello.cpp

#include <iostream>

extern "C" int osszead(int, int);

int main()
{
  std::cout << "Hello World!" << std::endl;
  std::cout << osszead(34, 6);
  return 0;
}
```

Nézzük meg a keletkezett *hello.o* egy részletét, láthatjuk, hogy az *osszead*-ból kikerült minden típusinformáció:

```
U osszead
U std::ostream::operator<<(int)
U std::ostream::operator<<(std::ostream& (*)(➥
    std::ostream&))
U std::ios_base::Init::Init()
U std::ios_base::Init::~Init()
U std::cout
```

Próbáljuk meg újra a szerkesztést, sikerülni fog! A futási eredményt a következő:

```
./progi.exe

Hello World!
```

A curl használata

A *cURL* több protokollon keresztüli fájlelérést biztosít egy parancssori eszköz és egy könyvtár segítségével. A *libcurl* egy ingyenes kliensoldali URL-transzfer könyvtár, amely támogatja az FTP, FTPS, Gopher, HTTP, HTTPS, SCP, SFTP, TFTP, Telnet, DICT, fájl URI-séma, LDAP, LDAPS, IMAP, POP3, SMTP és RTSP protokollokat. A könyvtár kezeli a HTTPS tanúsítványokat, a HTTP POST-ot, a HTTP PUT-ot, az FTP-feltöltést, Kerberost, a HTTP űrlap alapú feltöltést, a proxykat, a sütiket, a felhasználónév-jelszóval történő autentikációt, a fájltranszfer-folytatást és a HTTP proxyt. A következő példában bemutatjuk a használatát, először egy tiszta C++ programon keresztül. A 4-3. Programlista a fejlesztendő *CurlHelper* osztály deklarációját tartalmazza, egyetlen hasznos metódussal, a *getHttp()* hívás lehetőségével. A metódus a paraméterül kapott URL alapján letölt egy html lapot.

```
// 4-3. Programlista: CurlHelper.h

#ifndef CURLHELPER_H
#define CURLHELPER_H

#include <iostream>
#include <string>
#include <curl/curl.h>

using namespace std;

class CurlHelper {
public:
    CurlHelper();
    CurlHelper(const CurlHelper& orig);
    virtual ~CurlHelper();

    char *getHttp(char *url);
    string getHttp(const string url);

private:
};

#endif  /* CURLHELPER_H */
```

A 4-4. Programlista az osztály implementációját tartalmazza, a hívott függvények a *lubcurl* függvényei.

```
1  // 4-4. Programlista: CurlHelper.cpp
2
3  #include "CurlHelper.h"
4
5  int writer(char *data, size_t size, size_t nmemb, string *buffer);
6
7  CurlHelper::CurlHelper()
8  {
9  }
10
11 CurlHelper::CurlHelper(const CurlHelper& orig)
12 {
13 }
14
15 CurlHelper::~CurlHelper()
16 {
17 }
```

```
18
19 char *CurlHelper::getHttp(char *url)
20 {
21     string s = getHttp( string(url) );
22     return (char *)s.c_str();
23 }
24
25 string CurlHelper::getHttp(const string url)
26 {
27     string buffer;
28
29     CURL *curl;
30     CURLcode result;
31
32
33     curl = curl_easy_init();
34
35     if (curl) {
36
37         curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
38         curl_easy_setopt(curl, CURLOPT_HEADER, 0);
39         curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, writer);
40         curl_easy_setopt(curl, CURLOPT_WRITEDATA, &buffer);
41
42         result = curl_easy_perform(curl); //http get performed
43
44         curl_easy_cleanup(curl); //must cleanup
45
46         //error codes: http://curl.haxx.se/libcurl/c/libcurl-errors.html
47         if (result == CURLE_OK)
48             return buffer;
49         //curl_easy_strerror was added in libcurl 7.12.0
50         cerr << "error: " << result << " " << curl_easy_strerror(result) << endl;
51         return "";
52     }
53
54
55     cerr << "error: could not initalize curl" << endl;
56     return "";
57 }
58
59 int writer(char *data, size_t size, size_t nmemb, string *buffer)
60 {
61     int result = 0;
62     if (buffer != NULL) {
63         buffer->append(data, size * nmemb);
64         result = size * nmemb;
65     }
66     return result;
67 }
```

Az osztály használatát a 4-5. Programlista (*test-curl.cpp*) mutatja be, ahol a Google induló

lapját kérjük le és írjuk ki a képernyőre.

```
// 4-5. Programlista: test-curl.cpp

#include <cstdlib>
#include <iostream>
#include "CurlHelper.h"
using namespace std;

int main(int argc, char** argv)
{
    string url = "http://www.google.com";
    CurlHelper ch;
    string s = ch.getHttp( url );
    cout << s;
    return 0;
}
```

A fordítást az alábbi 3 paranccsal végeztük:

```
g++ -c CurlHelper.cpp
g++ -c test-curl.cpp
g++ curl-test.o CurlHelper.o -lcurl -o ➥
    curltest.exe
```

Megjegyezzük, hogy ebben a kisebb példában persze ez az 1 parancs is hatékonyan használható lett volna:

```
g++ test-curl.cpp CurlHelper.cpp -lcurl -o ➥
    curltest.exe
```

A program futási képernyője:

```
./curltest.exe

<HTML><HEAD><meta http-equiv="content-type" content="➥
    text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.hu/">here</A>.
</BODY></HTML>
```

A következő lépésekben a *CurlHelper* osztályt C# kódból fogjuk használni. Ehhez generáljuk le az illesztő forráskódot! A swig konfigurációs file (*CurlHelperModule.swg*) tartalma ez lett, azaz semmi különlegeset nem tartalmaz, egyszerűen használja a meglévő header fájlunkat:

```
%module
%{
   #include "CurlHelper.h"
%}

%include "CurlHelper.h"
```

A proxy forrásokat ezután így lehet előállítani a *CurlHelperModule* modulra:

```
swig -c++ -csharp CurlHelperModule.swg
```

C++ kódot kértünk a wrapper-hez, az illesztő module neve pedig *CurlHelperModule* lett. A következő fájlok jöttek létre:

- *CurlHelperModule_wrap.cxx*
- *CurlHelper.cs*
- *CurlHelperModule.cs*
- *CurlHelperModulePINVOKE.cs*
- *SWIGTYPE_p_string.cs*

Tekintettel arra, hogy a C++ string osztály illesztését most még nem ismerjük, ezért a *CurlHelper* osztály *char ** változatú metódusát fogjuk használni, ami a C# string-re mappelődik (ez egyébként a másik, *string*-es metódushoz delegálja a kérést):

```
char *CurlHelper::getHttp(char *url);
```

A *SWIGTYPE_p_string.cs* a C++ *std::string* reprezentációja, ennek használatáról lentebb írunk. Fordítsuk le a C++ fájlokat, majd építsük meg az *so* fájlt:

```
g++ -c -fPIC CurlHelperModule_wrap.cxx
g++ -c -fPIC CurlHelper.cpp

g++ -shared CurlHelper.o CurlHelperModule_wrap➥
    .o -lcurl -o libCurlHelperModule.so
```

A generált *CurlHelperModule.cs* most gyakorlatilag üres, mert csak a *CurlHelper* class van alapvetően exportálva, az *swg* fájl ezenfelül mást nem tartalmaz:

```
// 4-6. Programlista: CurlHelperModule.cs

using System;
using System.Runtime.InteropServices;

public class CurlHelperModule {
}
```

A *CurlHelper* class számára viszont egy külön C# forrás jött létre, itt (4-7. Programlista) csak az általunk használt részletet mutatjuk meg:

```
// 4-7. Programlista: CurlHelper.cs

using System;
using System.Runtime.InteropServices;

public class CurlHelper : IDisposable {
...
  public string getHttp(string url) {
    string ret = CurlHelperModulePINVOKE.➥
        CurlHelper_getHttp__SWIG_0(swigCPtr, ➥
        url);
    return ret;
  }
...
}
```

A főprogram végül az alábbi lett:

```
// 4-8. Programlista: runme.cs

using System;

public class runme {
  static void Main()
  {
    CurlHelper ch = new CurlHelper();
    string s = ch.getHttp("http://www.index.hu➥
        ");
    Console.WriteLine( s );
  }
}
```

Fordítsuk le a C# fájlokat, a futtatható program neve *runme.exe* lesz ismét:

```
dmcs -out:runme.exe *.cs
```

Végül nézzük meg a futási eredményt! Láthatjuk, hogy működik a *curl* könyvtár C# alatt. Csak gondoljunk egy kicsit bele! A *curl* egy C nyelven írt könyvtár, amire C++ felületet húztunk és végül C# (mono) környezetből használtuk.

```
./runme.exe
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN"> ➥

    <html><head> ➥

    <title>301 Moved Permanently</title> ➥

    </head><body> ➥

    <h1>Moved Permanently</h1> ➥

    <p>The document has moved
<a href="http://index.hu/">here</a>.</p><hr> ➥

    <address>Apache/2.2.9 (Debian) mod_ssl/2.2.9 ➥
    OpenSSL/0.9.8g Server at www.index.hu Port 80</➥
    address>            </body></html>
```

C++ illesztés

Ez egy olyan nagy témakör, amit itt nem lehet részletesen bemutatni, ehelyett ajánljuk a *swig* dokumentáció tanulmányozását. Itt most arra vállalkozunk, hogy bemutatjuk az *std::string* használatát, ahogy azt már korábban ígértük. Az ilyen és ehhez hasonló C++/STL (*Standard Template Library*) típusok leképzését ismeri a swig, amiket include fájlokban határoz meg. Írjuk be ezt a parancsot:

```
swig -swiglib
/usr/share/swig2.0
```

A válasz azt a könyvtárat adja meg, ahol ezek a típus mappingek megtalálhatóak (ezek a swig program részei). A *string* használatához mi most a */usr/share/swig2.0/csharp/std_string.i* include fájlt fogjuk használni. Ennek megfelelően tekintsük meg most a módosított *CurlHelperModule.swg* állományt, ahova egy *std_string.i* include is bekerült!

```
%module CurlHelperModule
%{
   #include "CurlHelper.h"
%}

%include "std_string.i"

%include "CurlHelper.h"
```

A *CurlHelper.h* header-ben csak a *string*-es metódus maradt meg, azaz megjegyzésbe tettük az előző példában használt *char ** típust használó alakot:

```
// 4-9. Programlista: CurlHelper.h

#ifndef CURLHELPER_H
#define CURLHELPER_H

#include <iostream>
#include <string>
```

```
#include <curl/curl.h>

using namespace std;

class CurlHelper {
public:
    CurlHelper();
    CurlHelper(const CurlHelper& orig);
    virtual ~CurlHelper();

    //char *getHttp(char *url);
    string getHttp(string url);
private:

};

#endif  /* CURLHELPER_H */
```

A következő parancs futása után már nem keletkezik meg a *SWIGTYPE_p_string.cs:*

```
swig -c++ -csharp CurlHelperModule.swg
```

Ugyanakkor viszont látható az alábbi kódrészletben, hogy a C# *CurlHelper* class metódusa immár C# *string* típuson keresztül érhető el.

```
using System;
using System.Runtime.InteropServices;

public class CurlHelper : IDisposable {
  private HandleRef swigCPtr;
  protected bool swigCMemOwn;
...
public string getHttp(string url) {
    string ret = CurlHelperModulePINVOKE.➥
        CurlHelper_getHttp(swigCPtr, url);
    if (CurlHelperModulePINVOKE.➥
        SWIGPendingException.Pending) throw ➥
        CurlHelperModulePINVOKE.➥
        SWIGPendingException.Retrieve();
    return ret;
  }
...
```

Végül adjuk ki a következő parancsokat újra:

```
g++ -c -fPIC CurlHelperModule_wrap.cxx
g++ -c -fPIC CurlHelper.cpp
g++ -shared CurlHelper.o CurlHelperModule_wrap➥
    .o -lcurl -o libCurlHelperModule.so
dmcs -out:runme.exe *.cs
```

Majd futtassuk le a C# programot!

```
./runme.exe

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//➥
    EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="http://➥
    index.hu/">here</a>.</p>
<hr>
<address>Apache/2.2.9 (Debian) mod_ssl/2.2.9 ➥
    OpenSSL/0.9.8g Server at www.index.hu Port➥
     80</address>
</body></html>
```

Minden a helyén van, a futási eredmény is a kívánt kimenetet adta, miközben már a C++ sokat tudó *string* osztályát használtuk közvetlenül. A swig sok mintapéldával van összecsomagolva, így valószínű, hogy a legtöbb problémánkra már ott van egy lehetséges megoldás. Befejezésül megadjuk a fontosabb C++ és C# kapcsolatú include fájlok listáját, amiben a mi *string* típusunk is ott van:

- std_common.i
- std_deque.i
- std_except.i
- std_map.i
- std_pair.i
- std_shared_ptr.i
- std_string.i
- std_vector.i
- std_wstring.i stl.i
- typemaps.i
- wchar.i

Makrókat és saját include fájlokat természetesen magunk is fejleszthetünk, így az illesztési feladatokat – igény esetén – teljesen testre szabva végezhetjük el.

5. Java – C/C++ rutinok és könyvtárak használata

Az előző írás folytatásaként nézzük meg a Java és a külső könyvtárak és C/C++ kódrészletek illesztését is. Természetesen ezt is a *swig* utility segítségével fogjuk elvégezni és bemutatni. Java oldalról az ismert JNI (*Java Native Interface*) felülethez kell kötnünk mindezt. A példák megoldásainak szépségét az adja, hogy a már ismert swig swg konfigurációs fájlokat fogjuk használni, demonstrálni azok nyelvfüggetlen voltát is.

A szokásos „rutinok” példánk

Ezzel a cikkel nem csupán annyi a célunk, hogy Java esetére is bemutassuk a C/C++ illesztést, hanem a swig eszközről további tudnivalókat is szeretnénk átadni. Most azonban vegyük az ismerős *libRutinok.so* használatára való példát! A *rutinok.swg* fájlon a már említettek miatt nem kell változtatni, ezzel a paranccsal lehet legenerálni a Java illesztés kódját:

```
swig -java -package org.cs.swig.test rutinok.➥
    swg
```

Itt most megadtuk azt is, hogy a generált Java forráskód milyen csomagba kerüljön: *org.cs.swig.test*. C# esetén a *package* opció helyett *namespace* van, ezt nem használtuk az előzőekben, de ezzel a swig paranccsal tehettük volna:

```
swig -csharp -namespace org.cs.swig.test ➥
    rutinok.swg
```

A következő fájlok jöttek létre:

- *Rutinok.java* → Ez az az osztály (5-1. Programlista), amit a Java programok használhatnak. Neve a *module* nevével egyezik.
- *RutinokJNI.java* → A Java JNI illesztés rétege (5-2. Programlista). Ez egyszerűbb szerkezetű, mint a C# kód, ugyanis a Java egy alaposan átgondolt native interfésszel rendelkezik. Látható, hogy az egyes metódusok csak deklarálva lettek, mert a *native* kulcsszó jelzi, hogy az implementáció más helyen található (a *libRutinok.so* fájlban).
- *rutinok_wrap.c* → A kapcsolat másik, C/C++-os oldala, tartalmát nem szükséges megérteni.

```
// 5-1. Programlista: Rutinok.java

package org.cs.swig.test;

public class Rutinok {
  public static int osszeg(int a, int b) {
    return RutinokJNI.osszeg(a, b);
  }

  public static int kulonbseg(int a, int b) {
    return RutinokJNI.kulonbseg(a, b);
  }

  public static double sinWithDeg(double x) {
    return RutinokJNI.sinWithDeg(x);
  }

  public static String echo(String s) {
    return RutinokJNI.echo(s);
  }
}
```

```
// 5-2. Programlista: RutinokJNI.java

package org.cs.swig.test;
public class RutinokJNI {
  public final static native int osszeg(int jarg1, int jarg2);
  public final static native int kulonbseg(int jarg1, int jarg2);
  public final static native double sinWithDeg(double jarg1);
  public final static native String echo(String jarg1);
}
```

Végül nézzük meg a teszteléshez használt indító osztályt:

```
// 5-3. Programlista: Runme.java

package org.cs.swig.test;

public class Runme {

  static
  {
    try
    {
        System.loadLibrary("Rutinok");
    } catch (UnsatisfiedLinkError e) {
      System.err.println("Native code library failed to load.\n" + e);
      System.exit(1);
    }
  }

  public static void main(String argv[])
  {
    int o = Rutinok.osszeg(32, 18);
    System.out.println("Összeg = " + o);
    System.out.println("sin(30)=" + Rutinok.sinWithDeg( 30 ));
    System.out.println("Ezt küldtem=" + Rutinok.echo("Árvítűrő tükörfúrógép"));
  }
}
```

A *Rutinok* class betöltésekor, annak inicializációja során a *System.loadLibrary()* is lefut, ami dinamikusan betölti a *libRutinok.so* osztott könyvtárat, így a külső native implementáció rendelkezésre fog állni. Egy olyan *libRutinok.so* fájl szükséges, amiben a *rutinok_wrap.c* is benne van:

```
gcc -I/usr/lib/jvm/default-java/include/ -c -fPIC rutinok-1.c rutinok-2.c matek-rutinok.c rutinok_wrap.c
gcc -shared *.o -o libRutinok.so
```

A *-I* include kapcsolót azért adtuk meg, mert a Java JNI illesztés header fájljait el kell érni fordításkor, emiatt meg kell adnunk, hogy hol van. A Java forrásokat a következő utasítással lehet fordítani:

```
javac -d . Rutinok.java RutinokJNI.java Runme.➥
    java
```

A *-d* kapcsoló a csomagnak megfelelő könyvtár szerkezetet is létrehozza az aktuális mappa alatt. Utolsó lépésként már futtathatjuk a Java kódot, minden rendben működik:

```
java org.cs.swig.test.Runme
Összeg = 50
sin(30)=0.499999999481858
Ezt küldtem=Árvítűrő tükörfúrógép
```

Konstansok használata

A swig konfigurációs fájlokban konstansok is definiálhatók, amit a generált forráskód tartalmazni fog. A *rutinok.swg* állományt például most így egészítettük ki, azaz különféle módon állandó értékeket határoztunk meg:

```
%module Rutinok
%{
    #include "rutinok.h"
%}

/* Force the generated Java code to use the C
constant values rather than making a
JNI call */

%javaconst(1);

/* A few preprocessor macros */

#define     ICONST      42
#define     FCONST      2.1828
#define     CCONST      'x'
#define     CCONST2     '\n'
#define     SCONST      "Hello World"
#define     SCONST2     "\"Hello World\""

/* This should work just fine */
#define     EXPR        ICONST + 3*(FCONST)

/* This shouldn't do anything */
#define     EXTERN      extern

/* Neither should this (BAR isn't defined) */
#define     FOO         (ICONST + BAR)

/* The following directives
also produce constants */

%constant int iconst = 37;
%constant double fconst = 3.14;

int osszeg(int a, int b);
int kulonbseg(int a, int b);
double sinWithDeg(double x);
char *echo(char *s);
```

A fentiek egyértelműek, de a *%javaconst(1)* direktíva magyarázatra szorul. Amennyiben nem adjuk meg, úgy a konstansok a wrapper C/C++ forrásba kerülnek és csak a JNI felületről lesznek elérhetők. Ez azonban nem szükséges mindig, hiszen ezeket az állandókat a célnyelvben (most Java) is felvehetjük és ekkor az alábbi extra *RutinokConstants.java* Java forrás generálódik:

```
// 5-4. Programlista: RutinokConstants.java

package org.cs.swig.test;

public interface RutinokConstants {
  public final static int ICONST = 42;
  public final static double FCONST = 2.1828;
  public final static char CCONST = 'x';
  public final static char CCONST2 = '\n';
  public final static String SCONST = "Hello␣➥
      World";
  public final static String SCONST2 = "\"➥
      Hello␣World\"";
  public final static double EXPR = ➥
      42+3*(2.1828);
  public final static int iconst = 37;
  public final static double fconst = 3.14;
}
```

Változók használata

Ebben a pontban arra látunk példát, hogy a külső C/C++ kódban hogyan definiálunk változókat, illetve azokat milyen módon érjük el Javaból. A *rutinok.swg* fájlunkba most betettünk több *extern* deklarációt különféle C/C++ típusú változóra. Újdonság még, hogy meghatároztuk 5 új függvény exportálását is, amit majd természetesen a Java kódból szeretnénk használni. Ezek implementációja és a változók definíciója (azaz tényleges létrehozása) is a *rutinok-2.c* forrásba (5-5. Programlista) került.

```
%module Rutinok
%{
    #include "rutinok.h"
%}

/* Some global variable declarations */
%inline %{
extern int              ivar;
extern short            svar;
extern long             lvar;
extern unsigned int     uivar;
extern unsigned short   usvar;
extern unsigned long    ulvar;
extern signed char      scvar;
extern unsigned char    ucvar;
extern char             cvar;
extern float            fvar;
extern double           dvar;
extern char            *strvar;
extern const char       cstrvar[];
extern int             *iptrvar;
extern char             name[256];

extern Point           *ptptr;
extern Point            pt;
%}

/* Some read-only variables */

%immutable;

%inline %{
extern int  status;
extern char path[256];
%}
```

```
%mutable;

/* Some helper functions to make it
easier to test */

%inline %{
extern void  print_vars();
extern int  *new_int(int value);
extern Point *new_Point(int x, int y);
extern char  *Point_print(Point *p);
extern void  pt_print();
%}
```

```
/* Original functions */
int osszeg(int a, int b);
int kulonbseg(int a, int b);
double sinWithDeg(double x);
char *echo(char *s);
```

Amikor lefuttatjuk a *swig* generátort, akkor persze ezek az deklarációk benne lesznek a generált *rutinok_wrap.c* kódban.

```
// 5-5. Programlista: rutinok-2.c

#include <stdio.h>
#include <stdlib.h>

#include "rutinok.h"

int             ivar = 0;
short           svar = 0;
long            lvar = 0;
unsigned int    uivar = 0;
unsigned short  usvar = 0;
unsigned long   ulvar = 0;
signed char     scvar = 0;
unsigned char   ucvar = 0;
char            cvar = 0;
float           fvar = 0;
double          dvar = 0;
char           *strvar = 0;
const char      cstrvar[] = "Goodbye";
int            *iptrvar = 0;
char            name[256] = "Dave";
char            path[256] = "/home/beazley";


/* Global variables involving a structure */
Point          *ptptr = 0;
Point           pt = { 10, 20 };

/* A variable that we will make read-only in the interface */
int             status = 1;

//--- Variables
void print_vars() {
   printf("ivar      = %d\n", ivar);
   printf("svar      = %d\n", svar);
   printf("lvar      = %ld\n", lvar);
   printf("uivar     = %u\n", uivar);
   printf("usvar     = %u\n", usvar);
   printf("ulvar     = %lu\n", ulvar);
   printf("scvar     = %d\n", scvar);
```

```
  printf("ucvar      = %u\n", ucvar);
  printf("fvar       = %g\n", fvar);
  printf("dvar       = %g\n", dvar);
  printf("cvar       = %c\n", cvar);
  printf("strvar     = %s\n", strvar ? strvar : "(null)");
  printf("cstrvar    = %s\n", cstrvar ? cstrvar : "(null)");
  printf("iptrvar    = %p\n", iptrvar);
  printf("name       = %s\n", name);
  printf("ptptr      = %p (%d, %d)\n", ptptr, ptptr ? ptptr->x : 0, ptptr ? ptptr➥
     ->y : 0);
  printf("pt         = (%d, %d)\n", pt.x, pt.y);
  printf("status     = %d\n", status);
}

/* A function to create an integer (to test iptrvar) */

int *new_int(int value) {
  int *ip = (int *) malloc(sizeof(int));
  *ip = value;
  return ip;
}

/* A function to create a point */

Point *new_Point(int x, int y) {
  Point *p = (Point *) malloc(sizeof(Point));
  p->x = x;
  p->y = y;
  return p;
}

char * Point_print(Point *p) {
  static char buffer[256];
  if (p) {
    sprintf(buffer,"(%d,%d)", p->x,p->y);
  } else {
    sprintf(buffer,"null");
  }
  return buffer;
}

void pt_print() {
  printf("(%d, %d)\n", pt.x, pt.y);
}

//--- Variables


void gv1(char *s)
{
    printf( "\nCalled gv1: %s", s );
}
```

```
void gv2(char *s)
{
    printf( "\nCalled gv2: %s", s );
}
```

Az 5-6. Programlista az új tesztprogramot mutatja. Az érdekessége az, hogy a változók elérését az illesztőfüggvényen és a segédfüggvényen (*print_vars()*) keresztül is bemutatja. Az is látható, hogy az 5 db új függvény is elérhető a Java *Rutinok* osztályon keresztül (példa a kódból: *Rutinok.new_int(37)*).

```
// 5-6. Programlista: Runme.java

package org.cs.swig.test;

public class Runme {

  static
  {
    try
    {
        System.loadLibrary("Rutinok");
    } catch (UnsatisfiedLinkError e) {
      System.err.println("Native code library failed to load.\n" + e);
      System.exit(1);
    }
  }

  public static void main(String argv[])
  {
    // Try to set the values of some global variables
    Rutinok.setIvar(42);
    Rutinok.setSvar((short)-31000);
    Rutinok.setLvar(65537);
    Rutinok.setUivar(123456);
    Rutinok.setUsvar(61000);
    Rutinok.setUlvar(654321);
    Rutinok.setScvar((byte)-13);
    Rutinok.setUcvar((short)251);
    Rutinok.setCvar('S');
    Rutinok.setFvar((float)3.14159);
    Rutinok.setDvar(2.1828);
    Rutinok.setStrvar("Hello World");
    Rutinok.setIptrvar(Rutinok.new_int(37));
    Rutinok.setPtptr(Rutinok.new_Point(37,42));
    Rutinok.setName("Bill");

   // Now print out the values of the variables
    System.out.println( "Variables (values printed from Java)" );
    System.out.println( "ivar      =" + Rutinok.getIvar() );
```

```
    System.out.println( "svar      =" + Rutinok.getSvar() );
    System.out.println( "lvar      =" + Rutinok.getLvar() );
    System.out.println( "uivar     =" + Rutinok.getUivar() );
    System.out.println( "usvar     =" + Rutinok.getUsvar() );
    System.out.println( "ulvar     =" + Rutinok.getUlvar() );
    System.out.println( "scvar     =" + Rutinok.getScvar() );
    System.out.println( "ucvar     =" + Rutinok.getUcvar() );
    System.out.println( "fvar      =" + Rutinok.getFvar() );
    System.out.println( "dvar      =" + Rutinok.getDvar() );
    System.out.println( "cvar      =" + (char)Rutinok.getCvar() );
    System.out.println( "strvar    =" + Rutinok.getStrvar() );
    System.out.println( "cstrvar   =" + Rutinok.getCstrvar() );
    System.out.println( "name      =" + Rutinok.getName() );

    System.out.println( "\nVariables (values printed from C)" );
    Rutinok.print_vars();
    System.out.println( "\nNow I'm going to try and modify some read only ➥
        variables" );

    System.out.println( "     Trying to set 'path'" );
  }
}
```

A *Runme* osztály futási eredménye a következő lett:

```
java org.cs.swig.test.Runme

Variables (values printed from Java)
ivar      =42
svar      =-31000
lvar      =65537
uivar     =123456
usvar     =61000
ulvar     =654321
scvar     =-13
ucvar     =251
fvar      =3.14159
dvar      =2.1828
cvar      =S
strvar    =Hello World
cstrvar   =Goodbye
iptrvar   =7cae6d10
name      =Bill
ptptr     =7caf4570(37,42)
pt        =7caf8c68(10,20)

Variables (values printed from C)
ivar      = 42
svar      = -31000
lvar      = 65537
uivar     = 123456
usvar     = 61000
ulvar     = 654321
scvar     = -13
ucvar     = 251
fvar      = 3,14159
dvar      = 2,1828
cvar      = S
strvar    = Hello World
cstrvar   = Goodbye
iptrvar   = 0x7cae6d10
name      = Bill
ptptr     = 0x7caf4570 (37, 42)
pt        = (10, 20)
status    = 1
```

A C++ enum használata

A C++ felsorolás típust itt nem mutatjuk be, de az alábbiakban bemutatjuk annak a Java oldali használati módját, amihez készítettünk egy *enum-test.cpp* forrást. Előtte azonban az *enum-test.h* header állományba vegyünk fel egy egyszerű C++ felsorolást (5-7. Programlista)! Ez a *Color* típus, amit az *enum_test()* függvényen keresztül fogunk most használni.

```
// 5-7. Programlista: enum-test.h

enum Color { RED, BLUE, GREEN };

void enum_test( Color c );
```

Erre írjuk egy C++ függvényt, amit az 5-8. Programlista mutat.

```
// 5-8. Programlista: enum-test.cpp

#include "enum-test.h"
#include <stdio.h>

void enum_test(Color c)
{
  if (c == RED) {
    printf("color = RED, ");
  } else if (c == BLUE) {
    printf("color = BLUE, ");
  } else if (c == GREEN) {
    printf("color = GREEN, ");
  } else {
    printf("color = Unknown color!, ");
  }
}
```

Mondjuk meg a swig számára, hogy miből kell főzni:

```
%module EnumTest

%{
    #include "enum-test.h"
%}

/* Force the generated Java code to use
the C enum values rather than
making a JNI call */

%javaconst(1);

%include "enum-test.h"
```

Adjuk ki a következő 3 parancsot, ami legenerálja a swig illesztést, fordít és könyvtárat készít *libEnumTest.so* néven.

```
swig -c++ -java -package org.cs.swig.test enum-test.swg
g++ -I/usr/lib/jvm/default-java/include/ -c -fPIC enum-test.cpp enum-test_wrap.cxx
g++ -shared enum-test.o enum-test_wrap.o -o libEnumTest.so
```

Ennyi csupán, amit tennünk kellett az enum használatához, amit az 5-9. Programlista mutat.

```
// 5-9. Programlista: Runme.java

package org.cs.swig.test;

public class Runme {
  static
  {
    try
    { System.loadLibrary("EnumTest");}
    catch (UnsatisfiedLinkError e) {
      System.err.println("Native code library ➥
          failed to load.\n" + e);
      System.exit(1);
    }
  }

  public static void main(String argv[])
  {
     System.out.println("    " + Color.RED + "➥
         = " + Color.RED.swigValue());

     EnumTest.enum_test( Color.RED );
  }
}
```

Fordítsuk le és futtassuk a Java tesztprogramunkat:

```
javac -d . *.java
java org.cs.swig.test.Runme

    RED = 0
color = RED
```

A curl könyvtár használata

Most a már ismert *curl* illesztést elvégezzük Java-ra is. A *CurlHelperModule.swg* fájl természetesen most is ugyanaz marad, de ezúttal Java nyelvre generáljuk az illesztést:

```
swig -c++ -java -package org.cs.swig.test ➥
    CurlHelperModule.swg
```

A C++ rész fordítása a következő parancsokkal lehetséges:

```
g++ -I/usr/lib/jvm/default-java/include/ -c -fPIC CurlHelperModule_wrap.cxx
g++ -c -fPIC CurlHelperModule_wrap.cxx
g++ -shared CurlHelper.o CurlHelperModule_wrap.o -lcurl -o libCurlHelperModule.so
```

Java tesztprogramként ezt használtuk (5-10. Programlista):

```
// 5-10. Programlista: Runme.java

package org.cs.swig.test;

public class Runme {

  static
  {
    try
    {
      System.loadLibrary("CurlHelperModule");
    } catch (UnsatisfiedLinkError e) {
      System.err.println("Native code library ➥
          failed to load.\n" + e);
      System.exit(1);
    }
  }

  public static void main(String argv[])
  {
      CurlHelper ch = new CurlHelper();
      String s = ch.getHttp("http://www.index. ➥
          hu");
      System.out.println( s );
  }
}
```

Végül a Java források fordítása és a tesztfuttatás így alakul:

```
javac -d . *.java

java org.cs.swig.test.Runme

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//➥
    EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="http://➥
    index.hu/">here</a>.</p>
<hr>
<address>Apache/2.2.9 (Debian) mod_ssl/2.2.9 ➥
    OpenSSL/0.9.8g Server at www.index.hu Port➥
     80</address>
</body></html>
```

C++ referencia típus

A C++ referencia változó egy olyan változó, amely egy másiknak az alias neve. Deklaráláskor erre az *&* jellel utalunk. Gondolhatunk rá, mint egy automatikus működésű pointerre, aminek azonban csak egyszer állíthatjuk be azt a címet, amire mutat. Az 5-11. Programlista (*referencia-test.h*) *operator+()* metódusa 2 *Vector*-ra vonatkozó referencia paraméterrel rendelkezik: *a* és *b*. A visszatérési érték pedig egy értékként kezelt *Vector*, ami megfelel az összeadás megszokott szemantikájának. A *VectorArray* class *operator[]()* metódusa (indexelő operátor) pedig visszatérésként egy *Vector* referenciát ad.

```
// 5-11. Programlista: referencia-test.h

class Vector {
private:
  double x,y,z;
public:
  Vector() : x(0), y(0), z(0) { };
  Vector(double x, double y, double z) : x(x),➥
       y(y), z(z) { };
  friend Vector operator+(const Vector &a, ➥
      const Vector &b);
  char *print();
};

class VectorArray {
private:
  Vector *items;
  int     maxsize;
public:
  VectorArray(int maxsize);
  ~VectorArray();
  Vector &operator[](int);
  int size();
};
```

Balérték, Jobbérték. Egy C/C++ változó állhat az értékadás jobb vagy bal oldalán. Amikor jobb oldalon áll, akkor a változó mögötti értéket veszi a fordító. Amikor a bal oldalon, akkor a címét, hogy ahhoz a memória rekeszhez egy értéket rendeljen. □

Az osztályok implementációját a 5-12. Programlista mutatja. Az *operator+()* 2 vektor összegét adja vissza értékként. A *print()* metódus a Vector string alakját adja vissza. A *VectorArray* indexelő operátora visszaadja az *i.* elemet, ami egy *Vector*-ra mutató referencia, így balértékként is használható. Ez természetes viselkedés, mert a tömböknél megszoktuk, hogy egy *v[i]* lehet az értékadás bal és jobb oldalán is.

```
// 5-12. Programlista: referencia-test.cpp

#include "referencia-test.h"
```

```
#include <stdio.h>
#include <stdlib.h>

Vector operator+(const Vector &a, const Vector
    &b) {
  Vector r;
  r.x = a.x + b.x;
  r.y = a.y + b.y;
  r.z = a.z + b.z;
  return r;
}

char *Vector::print() {
  static char temp[512];
  sprintf(temp,"Vector %p (%g,%g,%g)", this, x
      ,y,z);
  return temp;
}

VectorArray::VectorArray(int size) {
  items = new Vector[size];
  maxsize = size;
}

VectorArray::~VectorArray() {
  delete [] items;
}

Vector &VectorArray::operator[](int index) {
  if ((index < 0) || (index >= maxsize)) {
    printf("Panic! Array index out of bounds.\
        n");
    exit(1);
  }
  return items[index];
}

int VectorArray::size() {
  return maxsize;
}
```

A swig konfigurációt az alábbi *referencia-test.swg* fájl mutatja. Mit látunk belőle?

1. Exportálja a wrapper C++ kód felé a *Vector* class-t
2. Kéri, hogy szó szerint bekerüljön a wrapper-be az *addv()* segédfüggvény, mert Javából még nem lehet *operator overloading* mechanizmust használni.
3. Exportálja a *VectorArray* class-t, de az osztályhoz az indexelő operátor helyett bevezeti a *get()* és *set()* metódusokat.

```
%module ReferenciaTest

%{
#include "referencia-test.h"
%}

class Vector {
public:
    Vector(double x, double y, double z);
   ~Vector();
    char *print();
};

/* This helper function calls
an overloaded operator */

%inline %{
Vector addv(Vector &a, Vector &b) {
  return a+b;
}
%}

/* Wrapper around an
array of vectors class */

class VectorArray {
public:
  VectorArray(int maxsize);
  ~VectorArray();
  int size();

  /* This wrapper provides an alternative to
      the [] operator */
  %extend {
    Vector &get(int index) {
      return (*$self)[index];
    }
    void set(int index, Vector &a) {
      (*$self)[index] = a;
    }
  }
};
```

Generáljuk le a Java illesztést forrásait:

```
swig -c++ -java -package org.cs.swig.test
    referencia-test.swg
```

A következő lépésben a C++ oldalt fordítsuk le a szokásos módon, aminek az eredménye a *libReferenciaTest.so* fájl lesz.

```
g++ -I/usr/lib/jvm/default-java/include/ -c -fPIC referencia-test_wrap.cxx referencia-test.cpp
g++ -shared referencia-test_wrap.o referencia-test.o -o libReferenciaTest.so
```

A teszteléshez írt kódot az 5-13. Programlista tartalmazza. Láthatjuk, hogy most a *libReferenciaTest.so* könyvtárat töltöttük be dinamikusan. Tanulmányozzuk a kódot! Nem bonyolult, amiatt nem is fűzünk hozzá külön megjegyzéseket.

```
// 5-13. Programlista: Runme.java

package org.cs.swig.test;

public class Runme {
  static
  {
    try
    {
      System.loadLibrary("ReferenciaTest");
    } catch (UnsatisfiedLinkError e) {
      System.err.println("Native code library failed to load.\n" + e);
      System.exit(1);
    }
  }

  public static void main(String argv[])
  {
    System.out.println( "Creating 2 Vector objects:" );
    Vector a = new Vector(3,4,5);
    Vector b = new Vector(10,11,12);
    System.out.println( "    Created " + a.print() );
    System.out.println( "    Created " + b.print() );
    //
    System.out.println( "Adding a+b" );
    Vector c = ReferenciaTest.addv(a,b);
    System.out.println( "    a+b = " + c.print() );
    c.delete();
    c = null;

    // ----- Create a vector array -----
    System.out.println( "Creating an array of vectors" );
    VectorArray va = new VectorArray(10);
    System.out.println( "    va = " + va.toString() );
    va.set(0,a);
    va.set(1,b);
    va.set(2,ReferenciaTest.addv(a,b));

    // Get some values from the array
    System.out.println( "Getting some array values" );
    for (int i=0; i<5; i++)
    {
        System.out.println( "    va(" + i + ") = " + va.get(i).print() );
    }

   // Watch under resource meter to check on this
    System.out.println( "Making sure we don't leak memory." );
    for (int i=0; i<1000000; i++) c = va.get(i%10);

    // ----- Clean up -----
    System.out.println( "Cleaning up" );
    va.delete();
    a.delete();
    b.delete();
  }
}
```

Miután átnéztük a tesztprogram kódját, fordításuk le és futtassuk! Mindezt a következő utasításokkal tehetjük meg:

```
javac -d . *.java

java org.cs.swig.test.Runme

Creating 2 Vector objects:
    Created Vector 0x7c8f4d98 (3,4,5)
    Created Vector 0x7c8f4db8 (10,11,12)
Adding a+b
    a+b = Vector 0x7c8f4e48 (13,15,17)
Creating an array of vectors
    va = org.cs.swig.test.VectorArray@16218f9
Getting some array values
    va(0) = Vector 0x7c8f4e90 (3,4,5)
    va(1) = Vector 0x7c8f4ea8 (10,11,12)
    va(2) = Vector 0x7c8f4ec0 (13,15,17)
    va(3) = Vector 0x7c8f4ed8 (0,0,0)
    va(4) = Vector 0x7c8f4ef0 (0,0,0)
Making sure we don't leak memory.
Cleaning up
```

C++ template

A C++ template a generikus programozás és kódgenerálás eszköze. A következő 5-14. Programlista egy template függvényt és egy template class deklarációját tartalmazza.

```
// 5-14. Programlista: template-sample.h

template<class T> T max(T a, T b) { return  a>➥
    b ? a : b; }

template<class T> class vector {
  T *v;
  int sz;
 public:
  vector(int _sz) {
    v = new T[_sz];
    sz = _sz;
  }
  T &get(int index) {
    return v[index];
  }
  void set(int index, T &val) {
    v[index] = val;
  }
#ifdef SWIG
  %extend {
    T getitem(int index) {
      return $self->get(index);
    }
    void setitem(int index, T val) {
      $self->set(index,val);
    }
  }
#endif
};
```

Az ehhez tartozó swig konfiguráció a következő:

```
%module TemplateTest

%{
#include "template-sample.h"
%}

/* Let's just grab the original
header file here */

%include "template-sample.h"

/* Now instantiate some specific
template declarations */

%template(maxint) max<int>;
%template(maxdouble) max<double>;
%template(vecint) vector<int>;
%template(vecdouble) vector<double>;
```

A *%template* kulcsszó egy konkrét, példányosítás utáni nevet ad a Java kód számára. Állítsuk elő az illesztést:

```
swig -c++ -java -package org.cs.swig.test ➥
    template-sample.swg
```

A létrejött fájlok:

- *template-sample_wrap.cxx*
- *TemplateTest.java*
- *TemplateTestJNI.java*
- *vecdouble.java* → a *vector<double>* típusra
- *vecint.java* → a *vector<int>* típusra

A teszteléshez használt következő Java forrás mutatja, hogy az előzőleg *%template*-tel adott nevet (példa: *maxint()* a függvényre vagy *vecint* a *vector<int>* típusra) milyen névvel használjuk. Egyébként a *Runme* teszt osztály egyértelmű. A *TemplateTest* class a module osztály, ezen keresztül hívjuk a függvényket. A *vecdouble* és *vecint* saját típusként jelenik meg a Java forráskódban is.

```
// 5-15. Programlista: Runme.java

package org.cs.swig.test;

public class Runme {

  static
  {
    try
    {
      System.loadLibrary("TemplateTest");
    } catch (UnsatisfiedLinkError e) {
      System.err.println("Native code library failed to load.\n" + e);
      System.exit(1);
    }
  }

  public static void main(String argv[])
  {
      // Call some templated functions
    System.out.println(TemplateTest.maxint(3,7));
    System.out.println(TemplateTest.maxdouble(3.14,2.18));

    // Create some class

    vecint iv = new vecint(100);
    vecdouble dv = new vecdouble(1000);

    for (int i=0; i<100; i++)
        iv.setitem(i,2*i);

    for (int i=0; i<1000; i++)
          dv.setitem(i, 1.0/(i+1));

    {
    int sum = 0;
    for (int i=0; i<100; i++)
          sum = sum + iv.getitem(i);

    System.out.println(sum);
    }

    {
    double sum = 0.0;
    for (int i=0; i<1000; i++)
          sum = sum + dv.getitem(i);
    System.out.println(sum);
    }
  }
}
```

A fordítást és futtatási eredményt itt láthatjuk:

```
javac -d . *.java

java org.cs.swig.test.Runme
```

```
7
3.14
9900
7.485470860550343
```

6. Python – C/C++ rutinok és könyvtárak használata

Az előző 2 írás folytatásaként most röviden áttekintjük a native kódrészletek és könyvtárak Python nyelvből való használatát. Ez a script nyelv napjaink egyik legnépszerűbb eszköze, így fontos tudni azt is, hogy a hiányzó külső funkcionalitások milyen módon integrálhatóak a nyelvbe.

A továbbiakban 2 példán keresztül mutatjuk be a külső könyvtárak és C/C++ kódrészek használatát. Az első az ismerős „rutinok” példa lesz, a másik pedig egy C++ class illesztéséről fog szólni.

A szokásos „rutinok” példa

A C kód (a *rutinok.h* header és *C* forrás) és a swig *rutinok.swg* fájlok az előző cikkekből megmaradva már készen vannak, most azonban a python illesztő parancsot kell lefuttatni:

```
swig -python rutinok.swg
```

A generált *Rutinok.py* illesztés egy részletét mutatja a 6-1. Programlista. Csak azt a részt mutatjuk itt meg, ami a használat szempontjából lényeges.

```
// 6-1. Programlista: Rutinok.py
...
import _Rutinok
...
def osszeg(*args):
  return _Rutinok.osszeg(*args)
osszeg = _Rutinok.osszeg

def kulonbseg(*args):
  return _Rutinok.kulonbseg(*args)
kulonbseg = _Rutinok.kulonbseg

def sinWithDeg(*args):
  return _Rutinok.sinWithDeg(*args)
sinWithDeg = _Rutinok.sinWithDeg

def echo(*args):
  return _Rutinok.echo(*args)
echo = _Rutinok.echo
```

A python-hoz generált *rutinok_wrap.c* forrással hozzuk létre a python megfelelő *_Rutinok.so* fájlt. Python esetén fontos, hogy az *so* fájl ilyen szerkezű névvel rendelkezzen (aláhúzással kezdődik és nincs *lib* előtag).

```
gcc -I/usr/include/python2.7 -c -fPIC rutinok-1.c rutinok-2.c matek-rutinok.c rutinok_wrap.c
gcc -shared *.o -o _Rutinok.so
```

Ezzel kész vagyunk, próbáljuk ki! Egyszerű párbeszédes üzemmódban indítsuk el a pythont. Az *import Rutinok* sor behozza a most elkészült Rutinok modult. Ezután a szokásos *echo()*, *sinWithDeg()* és *osszeg()* függvényeket futtatjuk interaktív módban.

```
inyiri@csdev1:$ python
Python 2.7.3 (default, Sep 26 2012, 21:53:58)
[GCC 4.7.2] on linux2
Type "help", "copyright", ...
>>> import Rutinok
>>> Rutinok.echo("aaa")
'aaa'
>>> Rutinok.sinWithDeg(30)
0.499999999481858
>>> Rutinok.osszeg(49, 51)
100
>>>
```

A C++ class használata

Vegyük a 6-3. Programlista által mutatott C++ osztályokat (és a hozzá tartozó header fájlt: 6-2. Programlista).

```
// 6-2. Programlista: example.h

class Shape {
public:
  Shape() {
    nshapes++;
  }
  virtual ~Shape() {
    nshapes--;
```

```
  };
  double  x, y;
  void    move(double dx, double dy);
  virtual double area(void) = 0;
  virtual double perimeter(void) = 0;
  static  int nshapes;
};

class Circle : public Shape {
private:
  double radius;
public:
  Circle(double r) : radius(r) { };
  virtual double area(void);
  virtual double perimeter(void);
};

class Square : public Shape {
private:
  double width;
public:
  Square(double w) : width(w) { };
  virtual double area(void);
  virtual double perimeter(void);
};
```

```
// 6-3. Programlista: example.cpp

#include "example.h"
#define M_PI 3.14159265358979323846

/* Move the shape to a new location */
void Shape::move(double dx, double dy) {
  x += dx;
  y += dy;
}

int Shape::nshapes = 0;

double Circle::area(void) {
  return M_PI*radius*radius;
}

double Circle::perimeter(void) {
  return 2*M_PI*radius;
}

double Square::area(void) {
  return width*width;
}

double Square::perimeter(void) {
  return 4*width;
}
```

Az osztály python-hoz való illesztéséhez a következő, meglehetősen egyszerű swig konfigurációs fájlt lehet készíteni:

```
%module example

%{
#include "example.h"
%}

/* Let's just grab the
original header file here */

%include "example.h"
```

A következő parancsok a már jól ismert lépések, aminek az eredményeként az *_example.so* és *example.py* python forrás jön létre.

```
swig -c++ -python example.swg

g++ -I/usr/include/python2.7 -c -fPIC example.cpp example_wrap.cxx
g++ -shared *.o -o _example.so
```

A tesztelést most nem interaktív módon, hanem a 6-4. Programlista által mutatott *runme.py* scripttel végezzük el. Ez a tesztprogram érthető, megértéséhez talán nem szükséges külön magyarázat. Az elején beimportálja a most megalkotott *example* modult és különféle módon használja a *Circle* és *Square* osztályokat, amik C++ nyelven készültek.

```
// 6-4. Programlista: runme.py

import example

# ----- Object creation -----
print "Creating some objects:"
c = example.Circle(10)
print "    Created circle", c
s = example.Square(10)
print "    Created square", s
```

```
# ------ Access a static member ------
print "\nA total of", example.cvar.Shape_nshapes,"shapes were created"

# Set the location of the object
c.x = 20
c.y = 30

s.x = -10
s.y = 5

print "\nHere is their current position:"
print "    Circle = (%f, %f)" % (c.x,c.y)
print "    Square = (%f, %f)" % (s.x,s.y)

# ------ Call some methods ------
print "\nHere are some properties of the shapes:"
for o in [c,s]:
      print "   ", o
      print "        area      =", o.area()
      print "        perimeter =", o.perimeter()

print "\nGuess I'll clean up now"

# Note: this invokes the virtual destructor
del c
del s

s = 3
print example.cvar.Shape_nshapes,"shapes remain"
print "Goodbye"
```

A tesztprogram futási képe a következő:

```
inyiri@csdev1:$ python runme.py
Creating some objects:
    Created circle <example.Circle; proxy of <Swig Object of type 'Circle *' at 0x8fc0b30> >
    Created square <example.Square; proxy of <Swig Object of type 'Square *' at 0x8fc0cf8> >

A total of 2 shapes were created

Here is their current position:
    Circle = (20.000000, 30.000000)
    Square = (-10.000000, 5.000000)

Here are some properties of the shapes:
    <example.Circle; proxy of <Swig Object of type 'Circle *' at 0x8fc0b30> >
        area      =  314.159265359
        perimeter =  62.8318530718
    <example.Square; proxy of <Swig Object of type 'Square *' at 0x8fc0cf8> >
        area      =  100.0
        perimeter =  40.0

Guess I'll clean up now
1 shapes remain
Goodbye
```

Java

A Python és Java illesztés egy érdekes lehetőség, hiszen a Java nagyon kiterjedt osztálykönyvtárral bír. Erre több megoldás is létezik, fontossága miatt most az egyiket mutatjuk meg. Igaz ez nem C/C++ kiegészítés, de hasonlóan fontos és közben a C++ nyelv is használatra került. Most az apache JCC nevű eszközét mutatjuk meg, aminek a webhelye: `http://lucene.`

apache.org/pylucene/jcc/. A JCC bármely linux disztribúcióban állandó csomagként megtalálható. Vegyük a 6-5. Programlista *TestClass.java* osztályát. A feladat az, hogy ezt illeszteni kell a python környezethez, aminek futásakor a Java VM egy példányát el lehet dinamikusan indítani és a Java releváns részeket abban végrehajtatni.

```
// 6-5. Programlista: TestClass.java

package org.cs.test;

public class TestClass
{
    private String msg;

    public TestClass() {
    }

    public void speak(String msg) {
      System.out.println(msg);
    }

    public void setString(String s) {
       msg = s;
    }

    public String getString() {
      return msg;
    }
}
```

Fordítsuk le a *TestClass.java* osztályt és tegyük be a *mytest.jar* fájlba:

```
javac -d . TestClass.java
jar -cf mytest.jar org
```

Ebben a lépésben használjuk a *jcc* programot a következő módon:

```
python -m jcc --jar mytest.jar --python ➥
    testmodul --build
```

A következők jönnek létre:

- *_testmodul.so*
- *testmodul* mappa, ami egy python modul (tartalma: *__init__.py* és *mytest.jar*)

A *jcc* előállít a *jar* alapján C++ forrásokat és header fájlokat, sőt a *build* kapcsoló hatására le is fordította azt (*_testmodul.so*). A később használható python modulnevet pedig a *–python* kapcsoló után adtuk meg. A következő lépés az elkészült *testmodul*-t próbáljuk ki, ezért interaktív módban indítsuk el a python-t:

```
inyiri@csdev1:$ python
Python 2.7.3
[GCC 4.7.2] on linux2
Type "help", "copyright", ...
```

Töltsük be a most elkészült modulunkat:

```
>>> import testmodul
```

Inicializáljuk a JVM-et:

```
>>> testmodul.initVM(classpath=testmodul.➥
    CLASSPATH)
<jcc.JCCEnv object at 0x87061f0>
```

Végül hozzuk létre a Java *TestClass* osztály 1 példányát és hívjuk meg a *speak()* metódusát.

```
>>> t = testmodul.TestClass()
>>> t.speak("Alma")
Alma
>>>
```

Ami kimaradt...

Hasonlóan az előző cikkhez, itt sem tudunk minden részletet ismertetni, ezért továbbra is biztatunk mindenkit a swig dokumentáció önálló tanulmányozására. Sok speciális, de érdekes példa lehetett volna még:

- callback használat
- kivételkezelés
- C függvényre mutató pointerek használata
- változó paraméterek
- C++ template
- C++ referencia
- A haladó C++ lehetőségek illesztése

7. HyperSQL DataBase (HSQLDB)

A HSQLDB (*Hyper Structured Query Language Database*) egy Javaban írt relációs adatbázis-kezelő rendszer. Gyors és kis méretű adatbázis kezelő, amelyben lehetőség van memóriában, illetve lemezen is tárolni a táblákat. Elérhető beágyazott és szerver módban egyaránt. Használják adatbázis-kezelőként és perzisztencia motorként több nyílt forráskódú projektben (*LibreOffice Base*), valamint kereskedelmi szoftverben (*Mathematica*). Elég nagy részhalmazát támogatja az SQL-92 és SQL-2008 szabványoknak. Webhelye: `http://hsqldb.org/`.

A HSQLDB már több, mint 10 éve fejlesztett, stabil, jó minőségű RDBMS[1]. A Java világban sokszor lehet vele találkozni, mert stabilitása mellett van néhány olyan tulajdonsága is, amit a fejlesztők szeretnek. Egyszerűen rendelkezésre álló és teljes funkcionalitású SQL környezet, így a fejlesztés és tesztelés fázisban is gyors, hatékony segítséget ad.

Alapvető ismeretek

A *HSQLDB* a projekt webhelyéről tölthető le és egyszerűen csak ki kell csomagolni a tartalmát egy mappába. A *lib* könyvtár alatt lévő *hsqldb.jar* fájl mindent tartalmaz:

- Az adatbázis motort
- Az kliens adminisztrációs eszközöket és a
- Java JDBC drivert

A hatékonyabb munkához érdemes elkészíteni a következő 5 parancsindító scripet. A *runManagerSwing.sh* egy grafikus frontend vastag programot indít el, aminek a képét a 7.1. ábra mutatja.

```
# runManagerSwing.sh
java -classpath hsqldb/lib/hsqldb.jar org.hsqldb.util.DatabaseManagerSwing $1 $2 $3 $4 $5 $6 $7 ➥
    $8 $9
```

A *runManager.sh* egy hasonló, más stílusú SQL felületet indít el.

```
# runManager.sh
java -classpath hsqldb/lib/hsqldb.jar org.hsqldb.util.DatabaseManager $1 $2 $3 $4 $5 $6 $7 $8 $9
```

A *runServer.sh* a HSQLDB szerver módját indítja el, így ebben az esetben az éppen megadott adatbázis hálózaton is elérhető lesz. Ez a működés a produktív mód, de fejleszteni is így érdemes.

```
# runServer.sh
java -classpath hsqldb/lib/hsqldb.jar org.hsqldb.util.DatabaseManager $1 $2 $3 $4 $5 $6 $7 $8 $9
```

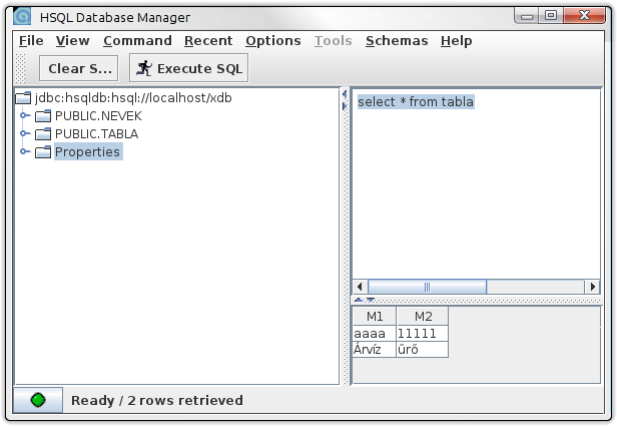
Az adatbázis a *http(s)* felületen is elérhető ezzel a szolgáltatással (*runWebServer.sh*).

```
# runWebServer.sh
java -classpath hsqldb/lib/hsqldb.jar org.hsqldb.server.WebServer $1 $2 $3 $4 $5 $6 $7 $8 $9
```

A *runUtil.sh* utility osztályok indítását segíti.

```
# runUtil.sh
java -classpath hsqldb/lib/hsqldb.jar org.hsqldb.util.$1 $2 $3 $4 $5 $6 $7 $8 $9
```

[1]RDBMS=Relation DataBase Management System



7.1. ábra: Database Manager (Swing)

A *HSQL Database Manager* segítségével létrehoztunk egy *alma.db* nevű adatbázis. Ezt úgy kell megtenni, hogy a *Connect* során ezt a JDBC stringet adtuk meg: *jdbc:hsqldb:file://home/tanulas/hsqldb/alma.db*. Látható, hogy az adatbázis fájlok esetünkben a */home/tanulas/hsqldb* mappában fognak megkeletkezni. Belépés után a következő 2 táblát hoztuk létre:

```
CREATE TABLE PUBLIC.TABLA(M1 VARCHAR(10),M2 ➥
    VARCHAR(20))
CREATE TABLE PUBLIC.NEVEK(FIRS VARCHAR(50),➥
    LAST VARCHAR(50))
```

Nézzük meg, hogy milyen fájlokból áll egy adatbázis, esetünkben az *alma.db*.

- *alma.db.script* → A teljes adatbázis nulláról való létrehozását tartalmazó SQL parancsok sorozata. A HSQLDB képes az adatbázis fájlban és memóriában is tárolni. A memóriában való tárolás nem biztosít perzisztenciát, azaz az adatok elvesznek. Ekkor a JDBC connect string alakja: *jdbc:hsqldb:mem:mymemdb*. Lehetőség van az adatbázist fájlba perzisztálni, ekkor használatos ez a script. Ilyenkor így néz ki a JDBC string:

```
jdbc:hsqldb:file://home/tanulas/hsqldb/➥
    alma.db
```

- *alma.db.properties* → Az adatbázisra vonatkozó property-k.
- *alma.db.log* → Ide történik a naplózás, de adatbázis shutdown esetén ürül.
- *alma.db.lck* → Amikor *in-process* módon használjuk az adatbázis-kezelőt, akkor azt csak 1 user érheti el egyidőben, mert lockolódik. Ez a korlátozás *server* módban természetesen nem áll fenn, akárhány user párhuzamosan dolgozhat az adatbázison.

Végül már most az elején szeretnénk egy lényeges dolgot kiemelni. A perzisztens (fájl alapú) adattáblák 2 módon kezelődnek:

- *memory* → Ez az alapértelmezés. Ekkor az adatbázis tábla mindig a script fájlból jön létre, emiatt ez csak teszteléshez ajánlott.
- *cached* → Az adatok cached módon egy külső fájlban vannak, így produktív mód esetén ezt kell használni, mert nagyméretű táblák kezelésére alkalmas.

A következő paranccsal beállítottuk, hogy a *nevek* tábla *cached* legyen (ezt már a *create table* parancsban is megtehettük volna):

```
SET TABLE nevek TYPE CACHED
```

Ekkor az eddigi fájlok mellett létrejön az adatok tárolója is:

- *alma.db.data*

Ennyi előzmény után már meg is írhatjuk a 7-1. Programlista programját, amit először az 1 felhasználós (nem szerver módú) környezetben próbáltunk ki a *jdbc:hsqldb:file://home/tanulas/hsqldb/alma.db* connection stringgel. A Database Manager futott és fogta a teljes adatbázis, így a program első futása kivételre ugrott. Ezután kilépve az admin programból, a mi tesztprogramunk is hibátlanul lefutott, bizonyítva ennek a módszernek az 1 felhasználós voltát. Elkerülendő, hogy

ez többet ne forduljon elő, illetve a manager és sok más kliens párhuzamosan is használhassa az adatbázist, indítsuk azt el server/hálózatos módban:

```
./runServer.sh --database.0 file:➥
   alma.db --dbname.0 xdb
```

Az egyből látható, hogy az *alma.db* lett felmountolva, az *xdb* pedig az adatbázis alias neve, ez látszik kívülről, hogy ne lehessen kitalálni az eredeti AB nevet. A 7.1. ábra már a hálózatos kapcsolódást mutatja, illetve a 7-1. Programlista éppen aktuális, nem megjegyzésbe tett connect stringje is ez:

```
jdbc:hsqldb:hsql://localhost/xdb
```

A *hsql* a server mód protokollja, létezik a *hsqls* változata is a titkosított csatornához.

```
1  // 7-1. Programlista: TestHsqlDbConnection.java
2
3  package org.cs.test;
4
5  import java.sql.Connection;
6  import java.sql.DriverManager;
7  import java.sql.ResultSet;
8  import java.sql.SQLException;
9  import java.sql.Statement;
10
11 public class TestHsqlDbConnection
12 {
13     public static void main(String[] args) throws SQLException
14     {
15         // jdbc:hsqldb:hsql://localhost/xdb
16         // jdbc:hsqldb:file://home/tanulas/hsqldb/alma.db
17         Connection conn = DriverManager.getConnection("jdbc:hsqldb:hsql://➥
               localhost/xdb", "SA", "");
18         Statement st = conn.createStatement();
19         ResultSet rs = st.executeQuery("select * from tabla");
20
21         for (;rs.next();)
22         {
23             System.out.println( rs.getString(1) );
24             System.out.println( rs.getString(2) );
25         }
26
27         st.close();
28         rs.close();
29         conn.close();
30     }
31 }
```

A *runWebServer.sh* paranccsal történő server mód indítás a *https(s)* kapcsolat felett publikálja ki az AB szervert. Ekkor a használható connect string:

```
Connection c = DriverManager.getConnection("jdbc:hsqldb:http://localhost/xdb", "SA", "");
```

Adatbázis ezzel a Java sorral állítható le:

```
Connection = DriverManager.getConnection("jdbc:hsqldb:file://home/tanulas/hsqldb/alma.db;➥
    shutdown=true", "SA", "");
```

A *CACHED* vagy *MEMORY* tárolás alapértelmezett módját ezzel az SQL paranccsal lehet beállítani:

```
SET DATABASE DEFAULT TABLE TYPE { CACHED | ➥
    MEMORY };
```

A környezet támogatja az átmeneti (temporary) táblákat, amik csak a connect után jönnek létre, üresen. Disconnect esetén pedig megszűnnek. Az ilyen táblák mindig egy session-höz keletkeznek és rendelődnek.

Az SQL nyelv támogatása

A HSQLDB törekszik a legfrissebb SQL szabványok implementálására az adatbázis használat mindhárom területén:

- DDL (*Data Definition Language*)
- DML (*Data Manipulation Language*)
- DQL (*Data Query Language*)

A szabványos adattípusokat támogatja, itt most nem térünk ki erre részletesen:

- Numeric típusok (Integral Types, DECIMAL, DOUBLE)
- Boolean típus
- Character String típusok (CHARACTER, CHAR, VARCHAR, CLOB)
- Binary String típusok (BINARY, VARBINARY, BINARY VARYING, BLOB, LONGVARBINARY)
- Bit String típusok (BIT, BIT VARYING)
- Datetime típusok (DATE, TIME(6), TIMESTAMP(2) WITH TIME ZONE)
- Interval típusok (INTERVAL YEAR TO MONTH, INTERVAL SECOND(4,6))
- Tömbök (Arrays)

Kipróbáltunk sok SQL parancsot, mindegyik tökéletesen működött. A lentiek csak ebből egy részletet tartalmaznak, hiszen a jelen cikk nem az SQL nyelvet szeretné ismertetni. Két példa tábla létrehozására:

```
CREATE TABLE nevnap (
  nev1 varchar(25) NOT NULL,
  nev2 varchar(25) NOT NULL,
  ho   integer NOT NULL,
  nap  integer NOT NULL
)

CREATE TABLE orszagok
(
  orszag varchar(27) ,
  fovaros varchar(19) ,
  foldr_hely varchar(37) ,
  terulet decimal(11,2) default 0.00 NOT NULL,
  allamforma varchar(30) ,
  nepesseg int default 0 NOT NULL ,
  nep_fovaros int default '0' NOT NULL ,
  autojel char(3) ,
  country varchar(31) ,
  capital varchar(19),
  penznem varchar(20),
  penzjel char(3) ,
  valtopenz varchar(18),
  telefon int default '0' NOT NULL ,
  gdp int default '0' NOT NULL ,
  kat int  default '0' NOT NULL
)
```

Egy új index legenerálása:

```
create index idx_nevnap_datum on nevnap(ho, ➥
    nap)
```

Domain létrehozása. Ez egy fontos SQL lehetőség, amit a HSQLDB is támogat. Azért jelentős, mert az egyes oszlopoknak megadható típusok helyett korábban megalkotott domain nevek

is alkalmazhatóak, amik előre preparált konkrétabb típusok és esetleg már a megszorítások is hozzá vannak rendelve.

```
CREATE DOMAIN postal_code AS varchar(20)
CHECK( value IS NOT NULL AND CHARACTER_LENGTH(➥
    value) > 2);
```

Az ilyen táblák adnak információt az adatbázis metaadatairól:

```
SELECT * FROM   INFORMATION_SCHEMA.TABLES
```

Szekvencia létrehozása és a következő érték lekérése:

```
CREATE SEQUENCE my_sorszam
SELECT next value for my_sorszam  from nevek
```

A Session és Tranzakció

Minden SQL művelet a felhasználói bejelentkezés után kialakult kapcsolathoz (*session*) rendelve hajtódik végre. A session néhány előre beállított jellemzővel rendelkezik, amiket persze a használata során át is lehet állítani. Itt most nem adjuk meg az összes ilyen kapcsolathoz rendelt változó és attribútum beállítási lehetőséget, de bemutatunk párat. A session a tranzakciókezelés kerete is, amit a HSQLDB teljes körűen ismer (*COMMIT*, *ROLLBACK*).

Session változók

A bejelentkezés után létrejön néhány beállított, de már nem megváltoztatható érték. Példák:

- CURRENT_USER (a bejelentkezett felhasználó login neve)
- CURRENT_SCHEMA (az éppen használt séma neve)

Ezzel együtt képesek vagyunk új változókat is létrehozni, melyre egy példa:

```
DECLARE counter INTEGER DEFAULT 3;
DECLARE result VARCHAR(20) DEFAULT NULL;
SET counter=15;
CALL myroutine(counter, result)
```

Jelenleg ezek a változók csak a tárolt eljárások IN, OUT vagy INOUT paraméterei lehetnek, azaz egy SQL műveletben nem használhatóak fel.

Session Táblák

Ezek a táblák a session alatt hozhatók létre, azok lezárásával meg is szűnnek.

```
DECLARE LOCAL TEMPORARY TABLE buffer (id ➥
    INTEGER PRIMARY KEY, textdata VARCHAR(100)➥
    ) ON COMMIT PRESERVE ROWS
```

Tranzakciók

A HSQLDB támogat különféle transaction isolation modellt:

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE

Ezek az összes konkurenciavezérlési típus alatt elérhetőek, mely az alábbi paranccsal állítható be:

```
SET DATABASE TRANSACTION CONTROL { LOCKS | ➥
    MVLOCKS | MVCC }
```

A *two-phase locking modell* az alapértelmezett mód, amelyben a HSQLDB az elosztott tranzakciók részévé tud válni. A session-ök az SQL környezetek ismert *alter session* parancsával állíthatók. Az auto commit ezzel a *set* utasítással manipulálható:

```
SET AUTOCOMMIT { TRUE | FALSE }
```

Jogosultságkezelés

A HSQLDB – ahogy azt a komolyabb adatbáziskezelők esetén megszoktuk – egy teljes körű hitelesítési és jogosultságkezelési rendszerrel rendelkezik, így a más rendszerekből ismert adminisztrációs parancsok itt is használhatóak. Példaként hozzunk létre egy *inyiri* nevű user-t (a parancsban 1 lépéssel admin lett):

```
CREATE USER inyiri PASSWORD pppppp ADMIN
```

A repository (system tables) tartalmazza az adatbázis összes felhasználóját, így ezzel a paranccsal már ez az új user is megjelenik:

```
SELECT * FROM INFORMATION_SCHEMA.SYSTEM_USERS
```

A szokásos *alter* és *drop user* parancsok is elérhetőek. Fontos annak a megadása, hogy bejelentkezés után melyik sémába kerül a felhasználó. Ez így lehetséges:

```
ALTER USER <user name> SET INITIAL SCHEMA <➥
    schema name>
```

Persze ehhez sémákat is létre lehet hozni, a későbbiekben látjuk majd, hogy ezt a HSQLDB támogatja. Egy új adatbázis létrehozásakor létrejön 2 séma:

- PUBLIC → Amennyiben nem rendeltük máshova a felhasználót, ez lesz a sémája.
- INFORMATION_SCHEMA → Az adatbázis repository sémája.

A *user* objektumok már az SQL-92 szabványban is léteztek, de a *role* csak az SQL-99-ben jelent meg. Jogosultságokat (*GRANT*=engedményezés) *user* vagy *role* kaphat, amelyek ezekhez hasonlóak lehetnek:

- Tábla select,
- Tábla változtatási műveletek (insert, update, delete),
- Futtatás,
- DDL műveletek

Például a lehetséges tábla privilégiumokat ezzel a select-tel lehet lekérdezni:

```
SELECT * FROM INFORMATION_SCHEMA.➥
    TABLE_PRIVILEGES
```

Nézzük meg egy példán keresztül a user, role és grant használatát a korábban létrehozott *inyiri* user-re! Első lépésben csinálunk egy *approle* nevű szerepkört (role):

```
CREATE ROLE approle
```

Ezután azt mondjuk, hogy *inyiri* felhasználót felhatalmazzuk ennek a szerepkörnek a betöltésére:

```
GRANT approle TO inyiri
```

Innen kezdve az *approle* szerepkörhöz rendelt jogokban *inyiri* is eljárhat. A következő 3 parancsban adunk a szerepkörnek néhány jogot:

```
GRANT SELECT, UPDATE ON TABLE nevnap TO ➥
    approle
GRANT USAGE ON SEQUENCE asequence to approle
GRANT EXECUTE ON ROUTINE aroutine TO approle
```

A fentiek értelmében szelektálhatja és frissítheti a *nevnap* táblát, használhatja az *asequence* szekvenciát, valamint futtathatja az *aroutine*-t. A jogok az SQL-ben megszokott *REVOKE* privilege paranccsal vonhatóak vissza.

A HSQLDB 2 előre létező, beépített felhasználót biztosít: *SA* és *SYS*. Ugyanígy a következő szerepkörök is előredefiniáltak:

- PUBLIC → Mindenkinek szükséges
- _SYSTEM → Ez a role nem rendelhető user-hez, a rendszer használja.
- DBA → Az összes adminisztratív taszkot végre lehet vele hajtani
- CREATE_SCHEMA → Egy sémát hozhat létre, módosíthat vagy törölhet
- CHANGE_AUTHORIZATION → Megváltoztathat jogosultságkezelési beállításokat

A HSQLDB sémák és objektumok

A séma

Egy adatbázis katalógusa tartalmaz sémákat, a sémák pedig az adatbázis objektumok névterei. Minden HSQLDB adatbázis pontosan 1 katalógust tartalmaz, aminek a neve *PUBLIC*, de ezt az *ALTER CATALOG RENAME TO* utasítással át is lehet nevezni. Egy új sémát így hozhatunk létre:

```
CREATE SCHEMA <schema name clause> [ <schema ➥
    character set specification> ] [ <schema ➥
    element>... ]
```

ahol a *schema name clause* alakja így néz ki:

```
<schema name>
   vagy
AUTHORIZATION <authorization identifier>
   vagy
<schema name> AUTHORIZATION <authorization ➥
    identifier>
```

Ez azt jelenti, hogy az egyszerű *CREATE SCHEMA <schema name clause>* esetben az aktuális user a séma tulajdonosa. Amennyiben *authorization identifier* részt is megadunk, úgy az egy *user* vagy *role* lehet. Példa:

```
CREATE SCHEMA ACCOUNTS AUTHORIZATION DBA;
```

Ebben az esetben az *ACCOUNTS* nevű sémát hoztuk létre, amit *DBA* szerepkörben lehet tulajdonolni.

Táblák

Korábban ismertettük a táblákkal kapcsolatos alapvető ismereteket, így most csak annyit szeretnénk megjegyezni, hogy ezek az adatbázisbázis üzleti adatainak tárolói, így a sémák talán legfontosabb objektumai. Az SQL nyelv összes ismert DDM művelete használható. A táblák *ID* oszlopának értékét egy névtelen szekvenciából érdemes automatikusan feltölteni, ahogy a példa is mutatja:

```
CREATE TABLE t
(
 id INTEGER GENERATED ALWAYS AS IDENTITY(START WITH ➥
     100),
 name VARCHAR(20) PRIMARY KEY
)
```

Természetesen külön szekvenciát is használhatunk erre a célra:

```
CREATE TABLE t
(
 id INTEGER GENERATED BY DEFAULT AS SEQUENCE s,
 name VARCHAR(20) PRIMARY KEY
)
```

View

A view egy SQL select, amit névvel is illetünk és úgy kezeljük, mintha egy tábla lenne. A *januar_nevek* egy olyan nézete a *nevnap* táblának, ami csak az 1. hónap sorait tartalmazza:

```
create view januar_nevek as select * from ➥
    nevnap where ho=1
```

A nézetekre *DROP VIEW* és *ALTER VIEW* lehetőségek használhatóak.

Nevek és referenciák

A sémákban minden objektumnak egy sémán belüli egyedi neve van, ezekkel hivatkozunk rá. Ez lehet minősített is, ahogy a következő lehetőség mutatja egy tábla oszlop esetére:

```
<catalog name>.<schema name>.<table name>.<➥
    column name>
```

Ezzel a teljes névhivatkozással más objektumok is megszólíthatóak, például egy szekvencia esetén ez így nézne ki:

```
<catalog name>.<schema name>.<sequence name>
```

Az egyes nevek az *ALTER ... RENAME TO* utasítással át is nevezhetőek.

Character Sets

A karakter halmaz vagy a teljes UNICODE kódkészlet vagy annak egy részhalmaza. Van jó pár előre definiált karakter halmaz, melyek közül a legfontosabbak:

- *SQL_TEXT* → a teljes UNICODE kódkészlet
- *SQL_CHARACTER* → az ASCII karakterek

Collation

A collation (egyeztetés) annak a megvalósítása, hogy a nemzeti lehetőségek figyelembevételével hasonlíthassunk össze 2 stringet. Az alapértelmezett collation az *SQL_TEXT*, ami az unicode karakter sorrend szerinti rendezést valósítja meg. A HSQLDB rendszerben létező összes collation neve így kérdezhető le a katalógusból:

```
SELECT * FROM INFORMATION_SCHEMA.COLLATIONS
```

Eredményként természetesen az *SQL_TEXT* is megjelenik. A magyar collation neve: *Hungarian*. További collation-ok létrehozására szolgál a *CREATE COLLATION*, bár erre ritkán van szükség. Ezt a mechanizmust több helyen lehet használni, de az egyik legnyilvánvalóbb a select *order by* utáni rendelkezési záradéka. Adjunk ki egy select parancsot:

```
select * from nevnap order by nev1
```

Ekkor például az *Ábel* a legvégén lesz, ami nyilvánvalóan helytelen, azonban így már jó lesz a sorrend:

```
select * from nevnap order by nev1 collate ➥
    Hungarian'
```

Tárolt eljárás

A HSQLDB támogatja az SQL 3 szabványban létező tárolt eljárásokat, amire itt most csak egy példát mutatunk:

```
create procedure getBook(in title varchar(30),➥
    out author varchar(30))
READS SQL DATA
begin atomic
  select title INTO author FROM books WHERE ➥
     title=books.title;
end
```

SQL 3 Típusok

Egy új típus a *CREATE TYPE* utasítással hozható létre:

```
CREATE TYPE MONEY AS NUMERIC(10, 2)
```

Ezután a *MONEY* egy olyan név, amit a tárolt eljárások írása során felhasználhatunk, hasonlóan a beépített típusokhoz.

Információk szerzése a sémáról

Ahogy korábban említettük, az adatbázis minden meta adatát a katalógus, azaz a *INFORMATION_SCHEMA* séma táblái tartalmazza. Ezt azt jelenti, hogy a séma összes objektumának a definíciója az itt lévő táblákban megtalálhatóak. Most csak ízelítőül láthatjuk az *ORSZAG* tábla szerkezetét, illetve a nemrég létrehozott *getBook* procedúra részleteit:

```
SELECT * FROM INFORMATION_SCHEMA.➥
    SYSTEM_COLUMNS where table_name='ORSZAGOK'
SELECT * FROM "INFORMATION_SCHEMA"."ROUTINES" ➥
    where specific_name='GETBOOK_12228'
```

Megszorítások (Constraints)

A HSQLDB ismeri a megszorításokat, azokat az Oracle vagy más hasonló környezetekhez hasonlóan lehet megadni, így védhetjük az adatbázis konzisztenciáját. Tekintsük át őket röviden!

A NOT NULL

A korábbiakban mutattunk egy példát a *nevek* tábla létrehozására, benne egy *nev1* mezővel, ahol a *NOT NULL* a legegyszerűbb megszorítások egyike:

```
CREATE TABLE nevnap (
  nev1 varchar(25) NOT NULL,
...
)
```

Ez annyit jelent, hogy a mező nem maradhat definiálatlan értékű, amikor egy új sort szúrunk a táblába.

Hivatkozási épség és a kulcs

Ez egy nagyon hatékony eszköz, hiányában sok programsort kéne arra fordítani, hogy a szülő-gyerek kapcsolatok épségét fenntartsuk (*foreign key*). Hozzunk létre 2 táblát:

- *AUTHORS*: A szerzők. A tábla kulcsa egy *ID* oszlop.
- *BOOKS*: Könyvek. A kulcs neve itt is *ID*. Az *AUTHOR_ID* a szerző azonosítóját tárolja, azaz csak olyan értéke lehet, ami előfordul az *AUTHORS* táblában. Ezt a tényt pontosan meg is adtuk a *FOREIGN KEY ... REFERENCES* részben, így a HSQLDB motor vigyázz arra, hogy csak olyan *BOOKS.AUTHOR_ID* legyen, ami létezik.

```
CREATE TABLE AUTHORS(ID INT PRIMARY KEY, NAME VARCHAR(25));
CREATE TABLE BOOKS(ID INT PRIMARY KEY, AUTHOR_ID INT, TITLE VARCHAR(100),
    FOREIGN KEY(AUTHOR_ID) REFERENCES AUTHORS(ID));
```

Példaképpen töltsük is fel a két táblát néhány rekorddal:

```
INSERT INTO AUTHORS(ID, NAME) VALUES(1, 'Jack London');
INSERT INTO AUTHORS(ID, NAME) VALUES(2, 'Honore de Balzac');
INSERT INTO AUTHORS(ID, NAME) VALUES(3, 'Lion Feuchtwanger');
INSERT INTO AUTHORS(ID, NAME) VALUES(4, 'Emile Zola');
INSERT INTO AUTHORS(ID, NAME) VALUES(5, 'Truman Capote');

INSERT INTO BOOKS(ID, AUTHOR_ID, TITLE) VALUES(1, 1, 'Call of the Wild');
INSERT INTO BOOKS(ID, AUTHOR_ID, TITLE) VALUES(2, 1, 'Martin Eden');
INSERT INTO BOOKS(ID, AUTHOR_ID, TITLE) VALUES(3, 2, 'Old Goriot');
INSERT INTO BOOKS(ID, AUTHOR_ID, TITLE) VALUES(4, 2, 'Cousin Bette');
INSERT INTO BOOKS(ID, AUTHOR_ID, TITLE) VALUES(5, 3, 'Jew Suess');
INSERT INTO BOOKS(ID, AUTHOR_ID, TITLE) VALUES(6, 4, 'Nana');
INSERT INTO BOOKS(ID, AUTHOR_ID, TITLE) VALUES(7, 4, 'The Belly of Paris');
INSERT INTO BOOKS(ID, AUTHOR_ID, TITLE) VALUES(8, 5, 'In Cold blood');
INSERT INTO BOOKS(ID, AUTHOR_ID, TITLE) VALUES(9, 5, 'Breakfast at Tiffany');
```

UNIQUE

A *unique* egy olyan megszorítás, ami arra figyel, hogy egy oszlop minden értéke különböző legyen. Ilyen például a *primary key* is, ahol emellett még a *NOT NULL* is feltétel ekkor:

```
CREATE TABLE Persons
(
    P_Id int NOT NULL UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),;
    Address varchar(255),
    City varchar(255)
);
```

CHECK Constraint

A *check* megszorítás egy oszlopra vagy a táblázatra értelmez egy ellenőrző algoritmust, így segítségével az összes többi constraint is implementálható lenne. Az alábbiak a *Persons* táblára adnak meg ellenőrzéseket:

```
CREATE TABLE Persons
(
 P_Id int NOT NULL CHECK (P_Id>0),
 LastName varchar(255) NOT NULL,
 FirstName varchar(255),
 Address varchar(255),
 City varchar(255)
)

CREATE TABLE Persons
(
 P_Id int NOT NULL,
 LastName varchar(255) NOT NULL,
 FirstName varchar(255),
 Address varchar(255),
 City varchar(255),
 CONSTRAINT chk_Person CHECK (P_Id>0 AND City➥
     ='Sandnes')
)
```

Az első check a *P_ID* oszlophoz kötődik, azonban a 2. példa már egy táblasor szintű ellenőrzést valósít meg.

Triggerek

A triggerek az SQL-99 szabványban jelentek meg, így azokat a HSQLDB is támogatja. Ezek

olyan események kezelői, amik a DML (INSERT, UPDATE, DELETE) műveletek során hívódnak meg. Már itt az elején szeretnénk kiemelni azt a gyakori hibát, amikor triggert használnak ott, ahol a check megszorítás lenne a kívánatos. Az alábbiak egy példát mutatnak egy HSQLDB trigger létrehozására:

```
/* the trigger throws an exception if a customer with the given last name already exists */
CREATE TRIGGER trigone BEFORE INSERT ON customer
    REFERENCING NEW ROW AS newrow
    FOR EACH ROW WHEN (newrow.id > 100)
    BEGIN ATOMIC
      IF EXISTS (SELECT * FROM CUSTOMER WHERE CUSTOMER.LASTNAME = NEW.LASTNAME) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'already exists';
      END IF;
END
```

Trigger esemény

Azt már tudjuk, hogy az események kibocsátása az INSERT, UPDATE, DELETE hatására történhet a HSQLDB-ben is. Lényeges lehetőség, hogy szűrhessünk egy *when* feltétellel, ugyanis általában nem kéne mindig elcsattannia a triggernek:

```
CREATE TRIGGER trig AFTER INSERT ON testtrig
    BEFORE othertrigger
    REFERENCING NEW ROW AS newrow
    FOR EACH ROW WHEN (newrow.id > 1)
BEGIN ATOMIC
    INSERT INTO triglog VALUES (newrow.id, ➥
        newrow.data, 'inserted');
    /* more statements can be included */
END
```

A működés finomsága

Más SQL környezetekhez hasonlóan itt is 2 szint lehetséges:

1. FOR EACH STATEMENT → a trigger csak az SQL kifejezésre, 1 alkalommal fut le (előtte vagy utána). Ez az alapértelmezett működés.

2. FOR EACH ROW → a trigger minden érintett sorra végrehajtott művelet alkalmából lefut (előtte vagy utána).

Mikor csattanjon el?

A DML művelet szempontjából 3 helyen lehet a triggert kiváltani:

- *Before*: A DML művelet előtt váltódik ki, segítségével abortálhatjuk a műveletet. Itt a többi tábla tartalma nem változtatható meg, de az aktuális táblának a sora igen.

- *After*: Az after trigger már nem változtathatja meg DML művelet eredményét (azokon a sorokon, amiken hatott), azonban a többi tábla változtatható és egyéb adminisztrációs tevékenységek is elvégezhetőek.

- *Instead of*: A view-on deklarált triggerek.

Hivatkozás a sorokra

Amikor a trigger kódját írjuk, hivatkozni kell tudni annak régi és új értékére is. Erre szolgál ez a 2 formalizmus:

- *REFERENCING NEW ROW AS newrow*

- *REFERENCING OLD ROW AS oldrow*

A *newrow* és *oldrow* változókkal érhetjük el a régi és új táblasort. Példa a használatára:

```
CREATE TRIGGER trig AFTER INSERT ON testtrig
    REFERENCING NEW ROW AS newrow
    FOR EACH ROW WHEN (newrow.id > 1)
    INSERT INTO TRIGLOG VALUES (newrow.id, ➥
        newrow.data, 'inserted')
```

Java alapú trigger írása

A *org.hsqldb.Trigger* interface egy implementációja a Java alapú trigger. Ennek 1 implementálandó metódusa van:

```
fire(int type, String trigName, String tabName
    , Object[] oldRow, Object[] newRow);
```

A paraméterek jelentése:

- type → a trigger típusa
- trigName → a trigger neve
- tabName → annak a táblának a neve, amire a triggert alkalmazni szeretnénk
- oldRow → old row
- newRow → new row

Az adatok elérése és változtatása

A HSQLDB SQL adatelérési (*select*) és azok megváltoztatási lehetősége teljes mértékben kompatibilis az SQL-2008-as szabvánnyal, azaz nagyon korszerű. A Java JDBC alrendszerből tudjuk, hogy Java oldalról SQL parancs ezen 2 interface valamelyikének használatával adható ki:

- *java.sql.Statement* → teljesen különböző SQL parancsok egymás utáni kiadására
- *java.sql.PreparedStatment* → azonos vázú, de paraméterezhető SQL parancsok egymás utáni kiadására

Amennyiben egy SQL select kerül kiadásra, úgy egy kurzor jön létre, ennek a Java reprezentációja a *java.sql.ResultSet*. A létrejött kurzor lehet *SCROLL* vagy *NO SCROLL* típusú, ahogy azt a JDBC szabvány is tartalmazza. Egy JDBC statement parameterben elvileg megadható a következő 2 opció, de a másodikat a HSQLDB JDBC nem támogatja:

- TYPE_FORWARD_ONLY
- TYPE_SCROLL_INSENSITIVE

Amennyiben a selectben megadjuk a *FOR UPDATE OF <column name list>* záradékot, hogy a létrejött SQL kurzoron keresztül módosítható az eredménytábla sora. Tekintettel arra, hogy ezen cikknek nem feladata az SQL részletekbe menő ismertetése, ezért a további lehetőségek megismeréséhez nyugodtan tanulmányozzunk SQL könyveket, a HSQLDB nagy valószínűséggel nagy részüket támogatni fogja. További fontos tudnivalókat tartalmaz ehhez a részhez a HSQLDB ide vonatkozó dokumentációja is: `http://hsqldb.org/doc/2.0/guide/dataaccess-chapt.html`.

SQL rutinok

A HSQLDB támogatja a tárolt eljárások és függvények készítését SQL és Java nyelveken egyaránt. A függvények jellemzője:

- Az ismert CREATE FUNCTION paranccsal hozzuk létre.
- Visszatérési értéke egy szimpla érték vagy egy tábla.
- A függvény futása során nem módosíthatjuk az adatbázist.
- Egy SQL parancs részeként használható.
- Tartalmazhat input paramétereket.
- Támogatja a polimorfizmust, azaz több különböző függvénynek is lehet ugyanaz a neve.

Az eljárások jellemzői:

- A CREATE PROCEDURE paranccsal hozzuk létre.
- Visszatérhet nulla vagy több értékkel vagy eredményhalmazzal (Result Set).

- Módosíthatja az adatbázis adatait.
- Elkülönítetten az SQL parancsoktól, a *CALL* hívással futtathatjuk le.
- Tartalmazhat input paramétereket.
- Támogatja a polimorfizmust, azaz több különböző függvénynek is lehet ugyanaz a neve.

Nézzünk 2 példát függvény létrehozására:

```
CREATE FUNCTION an_hour_before_or_now(t TIMESTAMP)
  RETURNS TIMESTAMP
  NO SQL
  LANGUAGE JAVA PARAMETER STYLE JAVA
  SPECIFIC an_hour_before_or_now_with_timestamp
  EXTERNAL NAME 'CLASSPATH:org.npo.lib.nowLessAnHour'

CREATE FUNCTION an_hour_before_max (e_type INT)
  RETURNS TIMESTAMP SPECIFIC an_hour_before_max_with_int
  RETURN (SELECT MAX(event_time) FROM atable WHERE event_type = e_type) - 1 HOUR
```

Példa egy táblával való visszatérésre:

```
RETURN TABLE ( SELECT a, b FROM atable WHERE e = 10 );
```

Az alábbiakban azt demonstráljuk, hogy egy saját függvény, hogyan használható egy SQL parancsból. Íme a függvény:

```
CREATE FUNCTION an_hour_before (t TIMESTAMP)
  RETURNS TIMESTAMP
  RETURN t - 1 HOUR
```

És a használata:

```
SELECT an_hour_before(event_timestamp) AS notification_timestamp, event_name FROM events;
```

Most nézzünk meg egy procedure definiálását és hívását!

```
CREATE PROCEDURE new_customer(firstname VARCHAR(50), lastname VARCHAR(50))
  MODIFIES SQL DATA
  INSERT INTO CUSTOMERS VALUES (DEFAULT, firstname, lastname, CURRENT_TIMESTAMP)
```

Hívás:

```
CALL new_customer('JOHN', 'SMITH');
```

Az eljárások és függvények készítési lehetőségeiről a további információkért olvassuk el az SQL szabványt.

Beépített függvények

Az SQL beépített függvények (*TO_CHAR(...)*, *ASCII()*, stb.) természetesen a HSQLDB-nek is a részei. A built-in-ek 3 fő csoportba sorolhatóak:

- SQL Standard függvények
- JDBC Open Group CLI függvények
- HyperSQL Built-In függvények

Itt most ízelítőül, magyarázat nélkül néhány dátumkezeléssel kapcsolatos belső függvény használatot mutatunk be:

```
-- Dátum konstans
select week(DATE '2012-12-01') from nevek
select quarter(DATE '2012-01-5') from nevek
select week(DATE '2012-05-5') from nevek

-- Timestamp SQL típus
select DAY(TIMESTAMP '2012-02-14 12:30:44') from tabla

-- Az év napjainak száma
select dayofyear(DATE '2012-02-29') from nevek
-- A hónap napjainak száma
select dayofmonth(DATE '2012-02-29') from nevek
-- 3 hónap hozzáadása
select DATE_ADD ( DATE '2008-11-22', INTERVAL 3 MONTH ) from nevek
-- 5 nap hozzáadása
select DATE_ADD ( DATE '2008-11-22', INTERVAL 5 DAY ) from nevek
-- 5 év hozzáadása
select DATE_ADD ( DATE '2008-11-22', INTERVAL 5 YEAR ) from nevek
-- 5 év kivonása
select DATE_SUB ( DATE '2008-11-22', INTERVAL 5 YEAR ) from nevek
-- 5 óra kivonása
select DATE_SUB ( '2008-11-22 14:50:01', INTERVAL 5 HOUR ) from nevek
```

Rendszer menedzsment

Az eddigiektől láttuk, hogy a HSQLDB szerver és beágyazott üzemmódban is használható, ugyanakkor ennek a menedzsmentje ettől nagyrészt független.

Az adatbázis típusa

Az adatbázis típusának beállítása:

- *mem*: csak memóriában létezik,
- *file*: perzisztencia van és fájlban tárolt vagy
- *res*: erőforrás leíróban tárolt.

Read Only mód

Egy adatbázis read only módba állítható a *readonly=true* property megadásával.

Nagy méretű adatbázis

A táblák és large objektumok *CACHED* módon is tárolhatóak, ami azt jelenti, hogy azoknak csak egy része töltődik a memóriába. Ezzel nagyméretű adatbázisok is kezelhetőek. Példa a LOB CACHED beállításra:

```
SET TABLE SYSTEM_LOBS.BLOCKS TYPE CACHED
SET TABLE SYSTEM_LOBS.LOBS TYPE CACHED
SET TABLE SYSTEM_LOBS.LOB_IDS TYPE CACHED
```

A *ACID* teljes körű támogatása

Az ismert ACID kifejezés az *Atomicity*, *Consistency*, *Isolation* és *Durability* szavakból képzett mozaikszó.

Backup - Restore

Az adatbázis mentésére a *BACKUP DATABASE* parancs szolgál. Az online mentés ezzel a paranccsal oldható meg:

```
BACKUP DATABASE TO <directory name> BLOCKING
```

Lehetséges az offline mentés is:

```
java -cp hsqldb.jar org.hsqldb.lib.tar.DbBackup --save tardir/backup.tar dbdir/dbname
```

A mentésből való visszaállítás restore utasítása így használható:

```
java -cp hsqldb.jar org.hsqldb.lib.tar.DbBackup --extract tardir/backup.tar dbdir
```

Titkosított adatbázis használata

Amennyiben rendelkezünk egy szimmetrikus titkosító kulccsal, úgy ezzel a sémával is kapcsolódhatunk az adatbázishoz:

```
jdbc:hsqldb:file:<database path>;crypt_key=604a6105889da65326bf35790a923932;crypt_type=blowfish
```

Ez lehetővé tesz a titkosított tárolás és adatelérés lehetőségét.

Monitorozás

Itt csak felsoroljuk a beépített lehetőségeket:

- External Statement Level Monitoring
- Internal Event Monitoring
- Log4J and JDK alapú naplózás
- Server Operation Monitoring

A HSQLDB menedszmentjéről többet a szoftver dokumentációjából tudhatunk meg: `http://hsqldb.org/doc/2.0/guide/management-chapt.html`.

Network Listeners

Amikor szerver módban indítjuk adatbáziskelőnket, akkor annak a szolgáltatásai hálózaton keresztül érhetőek el. Ilyenkor a kliens egy un. listener szolgáltatáson keresztül kapcsolódik a szerverhez. Ez a működés megegyezik más ismert adatbázis-kezelőknél alkalmazott módszerrel.

Text táblák

Érdekes lehetősége a HSQLDB környezetnek, hogy egy szövegfájlra úgy tudunk rátekinteni, mint egy táblára. A text táblákat hasonlóan kell definiálni, mint a közönséges táblákat, de a *text* kulcsszót meg kell adni:

```
CREATE TEXT TABLE <tablename> (<column ➥
    definition> [<constraint definition>])
```

Ekkor a tábla még üres és nem lehet írni. A következő paranccsal lehet ezt a táblát egy konkrét szövegfájlhoz rendelni:

```
SET TABLE <tablename> SOURCE <➥
    quoted_filename_and_options> [DESC]
```

Nézzünk egy példát! Van egy *text_table.txt* fájl a következő tartalommal:

```
aaaaaaa;bbbbbbbbbb;111111
xxxxx;yyyyyyyyyy;22222
```

Most a „;” jelet használjuk mező elválasztóként (*Field Separator*). Ebből egy 3 oszlopos text táblát az *alma.db* adatbázisban így tudunk készíteni:

```
CREATE TEXT TABLE text_table
(
  o1 varchar(20),
  o2 varchar(20),
  o3 integer
)

SET TABLE text_table SOURCE "text_table.txt;fs➥
    =\semi"
select * from text_table

O1      O2          O3
aaaaaaa bbbbbbbbbb  111111
xxxxx   yyyyyyyyyy  22222
```

Az 1. utasítás o1, o2, o3 oszlopokkal hozza létre a táblát, majd a 2. hozzárendeli a fájlt és azt is megadja, hogy mi a mezőelválasztó karakter. A 3. parancs egy teszt SQL select-tel mutatja az eredménytáblát. Természetesen ez a

tábla olyan, mint a többi, azaz például beszúrhatunk 1 sort, ami meg is jelent a textfájlban:

```
insert into text_table values ('Ez', 'Az', ➥
    1000)
```

A tranzakciókezelés használható az ilyen táblákra is. A *csv* fájl támogatás külön lehetőse is be van építve, ugyanis itt 2 további problémát is meg kell oldani. Az egyik az, hogy az első sorban az oszlopnevek lehetnek, ezért azt figyelmen kívül kell hagyni. A másik feladat pedig a csv szabvány szerinti minden értéket körülvevő idézőjelek elhagyásának bekapcsolása:

```
SET TABLE mytable SOURCE "myfile;ignore_first=true;all_quoted=true"
```

A text tábláról való lekapcsolódás:

```
SET TABLE mytable SOURCE OFF
```

Tekintettel arra, hogy egy textfájl kódolása sokféle lehet, ezért ebben a példában annak megadására is mutatunk példát:

```
SET DATABASE TEXT TABLE DEFAULTS 'all_quoted=true;encoding=UTF-8;cache_rows=10000;cache_size➥
    =2000'
```

SQL Tool

A *HyperSQL Utilities Guide* tartalmaz néhány leírást azokról a további eszközökről, amiket a HSQLDB tartalmaz és megkönnyíti a mindennapi életünket (*sqltool.jar* fájl). Az SQL Tool az egyik, ami bármelyik JDBC képes adatbáziskezelő irányába használható. A célja az, hogy kiváltsa a következő ismert konzol módú programok használatát:

- *isql* for Sybase,
- *psql* for Postgresql,
- *Sql*plus* for Oracle.

Az eszköz képessége valóban lehetővé teszi ezt, rengeteg parancssori trükk és mindennapos feladat oldható meg a segítségével. Így indítható el:

```
java -classpath hsqldb/lib/hsqldb.jar:hsqldb/lib/sqltool.jar -jar hsqldb/lib/sqltool.jar --help
```

Transfer Tool

Ez egy olyan GUI eszköz (helye: *hsqldbutil.jar*), ami lehetővé teszi, hogy sémákat és adatokat másoljunk 2 JDBC alapú adatforrás között. Az eszköz érdekessége és izgalmas lehetőségei abból is fakadnak, hogy ez a 2 adatforrás eltérő adatbázis motor is lehet. Két üzemmódja van:

- Direct Transfer
- Dump és Restore alapú Transfer

8. Tippek és Trükkök

A mindennapi életben naponta bukkannak fel érdekes, ötletes megoldások. Itt most 3 olyan egyszerű ötletet szeretnénk bemutatni, amik az elmúlt hónapokban hasznosak voltak számunkra.

Certificate mentés 1 sorban

Mindig szükségünk van a *HTTPS* protokollal kiszolgáló webhely tanúsítványára. Ezt sokan úgy szerzik meg, hogy a böngésző segítségével mentik le. Mindez az *openssl* program segítségével sokkal gyorsabb és jobban automatizálható. Ehhez előbb nézzük meg az *s_client* openssl parancs működését! Indítsuk el és adjuk ki a parancsot egy tetszőleges *host:port* párra:

```
openssl
OpenSSL> s_client -connect b2b.mol.hu:443
```

Az eredményt a lenti, hosszabb output képernyő mutatja:

```
CONNECTED(00000003)
depth=1 C = HU, L = Budapest, O = NetLock Kft., OU = Tan\C3\BAs\C3\ADtv\C3\A1nykiad\C3\B3k (Certification ➥
     Services), CN = NetLock \C3\9Czleti (Class B) Tan\C3\BAs\C3\ADtv\C3\A1nykiad\C3\B3
verify error:num=20:unable to get local issuer certificate
verify return:0
---
Certificate chain
 0 s:/C=HU/L=Budapest/O=MOL Nyrt./OU=MOL EAI/CN=b2b.mol.hu
   i:/C=HU/L=Budapest/O=NetLock Kft./OU=Tan\xC3\xBAs\xC3\xADtv\xC3\xA1nykiad\xC3\xB3k (Certification Services)/➥
       CN=NetLock \xC3\x9Czleti (Class B) Tan\xC3\xBAs\xC3\xADtv\xC3\xA1nykiad\xC3\xB3
 1 s:/C=HU/L=Budapest/O=NetLock Kft./OU=Tan\xC3\xBAs\xC3\xADtv\xC3\xA1nykiad\xC3\xB3k (Certification Services)/➥
     CN=NetLock \xC3\x9Czleti (Class B) Tan\xC3\xBAs\xC3\xADtv\xC3\xA1nykiad\xC3\xB3
   i:/C=HU/L=Budapest/O=NetLock Kft./OU=Tan\xC3\xBAs\xC3\xADtv\xC3\xA1nykiad\xC3\xB3k (Certification Services)/➥
       CN=NetLock Arany (Class Gold) F\xC5\x91tan\xC3\xBAs\xC3\xADtv\xC3\xA1ny
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIG2DCCBcCgAwIBAgIOSd0P5gH+pJdkh9kohp8wDQYJKoZIhvcNAQELBQAwgakx
CzAJBgNVBAYTAkhVMREwDwYDVQQHDAhCdWRhcGVzdDEVMBMGA1UECgwMTmV0TG9j
ayBLZnQuMTcwNQYDVQQLDC5UYW7DunPDrXR2w6FueWtpYWTDs2sgKENlcnRpZmlj
YXRpb24gU2VydmljZXMpMTcwNQYDVQQDDC5OZXRMb2NrIMOcemxldGkgKENsYXNz
IEIpIFRhbsO6c8OtdHbDoW55a2lhZMOzMB4XDTEyMDcxODA5NDQzOVoXDTE0MDcx
ODA5NDQzOVowWzELMAkGA1UEBhMCSFUxETAPBgNVBAcMCEJ1ZGFwZXN0MRIwEAYD
VQQKDAlNT0wgTnlydC4xEDAOBgNVBAsMB01PTCBFQUkxEzARBgNVBAMMCmIyYi5t
b2wuaHUwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC60vMwEmV1Lbss
gxzIIEruaCtkR05pHId0KAilu7jNOC9I2D/vkbYtIpXrXHW5+/0munLPhX2J84+o
yb2j9yd5bfBpL7wYDD1JLMzaSoKLTwLDMzLXUotLY0y7XBQ/CmOTj1RQF1rFfrct
Xd48kDfeGcR9iQt1zxPI2O7rDu/KvXVDVqTyzAdFSF24mIK4BlG/qvwwhOUn1iX7
NUSXvnaziCX5c7BSWOnjtv80vzOG+RB/0RGzluIbWjXaUzgCvEa3jS3AzRKBrDYF
Lkj51D7tYjv9Obv5IoCNJ2WaVHOyIDZfefuKtDF7c/wVV284dQ5WOygBvhj56Exg
1TJrF3s/AgMBAAGjggNJMIIDRTAMBgNVHRMBAf8EAjAAMA4GA1UdDwEB/wQEAwID
qDATBgNVHSUEDDAKBggrBgEFBQcDATAdBgNVHQ4EFgQUdXyQ/jxRmtv4/WGLWQoe
UYZk3b0wHwYDVR0jBBgwFoAU8SfE3lo9NCm48+oPjf520jkIz8EwgewGA1UdIASB
5DCB4TCB3gYNKwYBBAGbYwExicu5JTCBzDAnBggrBgEFBQcCARYbaHR0cDovL3d3
dy5uZXRsb2NrLmh1L2RvY3MvMIGgBggrBgEFBQcCAjCBkwyBkE5lbSBtaW7FkXPD
rXRldHQgdGFuw7pzw610dsOhbnkuIFJlZ2lzenRyw6FjacOza29yIGEgc3plbcOp
bHllcyBtZWdqZWxlbsOpcyBrw7Z0ZWxlesWRLiBTem9sZ8OhbHRhdMOhc2kgc3ph
YsOhbHl6YXQ6IGh0dHA6Ly93d3cubmV0bG9jay5odS9kb2NzLzCBngYDVR0fBIGW
MIGTMC+gLaArhilodHRwOi8vY3JsMS5uZXRsb2NrLmh1L2luZGV4LmNnaT9jcmw9
Y2JjYTAvoC2gK4YpaHR0cDovL2NybDIubmV0bG9jay5odS9pbmRleC5jZ2k/Y3Js
PWNiY2EwL6AtoCuGKWh0dHA6Ly9jcmwzLm5ldGxvY2suaHUvaW5kZXguY2dpP2Ny
bD1jYmNhMIIBPgYIKwYBBQUHAQEEggEwMIIBLDAsBggrBgEFBQcwAYYgaHR0cDov
L29jc3AxLm5ldGxvY2suaHUvY2JjYS5jZ2kwLAYIKwYBBQUHMAGGIGh0dHA6Ly9v
Y3NwMi5uZXRsb2NrLmh1L2NiY2EuY2dpMCwGCCsGAQUFBzABhiBodHRwOi8vb2Nz
cDMubmV0bG9jay5odS9jYmNhLmNnaTA0BggrBgEFBQcwAoYoaHR0cDovL2FpYTEu
bmV0bG9jay5odS9pbmRleC5jZ2k/Y2E9Y2JjYTA0BggrBgEFBQcwAoYoaHR0cDov
L2FpYTIubmV0bG9jay5odS9pbmRleC5jZ2k/Y2E9Y2JjYTA0BggrBgEFBQcwAoYo
aHR0cDovL2FpYTMubmV0bG9jay5odS9pbmRleC5jZ2k/Y2E9Y2JjYTANBgkqhkiG
9w0BAQsFAAOCAQEAKhSN9sSX5S6MIw3rtbr8gJfbE73gljjttlOzhH/h0jXIsGAD
atgoFL3Kz6YShzNFoiQ9hX4reM14ppXPH2QSbZXHeS4HgGiZwix19PpxtLdl5NwB
Yvt4IWY/CuQYl4fEYHOqxoWlvWqUIZygDgZ56zlIE4x2qWfQdTpKj72Ux0qwonwT
ngd6caivwYMkzb4ZWT2k/fTzojxov4uINTL+x1qMRjEsGUPEtm2jNJzqlWObT2pt
dv0O8kiyLEgJ1yfTBTi+N/GzHSplVhQBUVtzqYtbqyjFHAufWa4PCCAawxfj+pOc
91LydZIJcHL7QoJ9VFlhcv+HjVJNB9LAhvfWAw==
-----END CERTIFICATE-----
subject=/C=HU/L=Budapest/O=MOL Nyrt./OU=MOL EAI/CN=b2b.mol.hu
issuer=/C=HU/L=Budapest/O=NetLock Kft./OU=Tan\xC3\xBAs\xC3\xADtv\xC3\xA1nykiad\xC3\xB3k (Certification Services)➥
     /CN=NetLock \xC3\x9Czleti (Class B) Tan\xC3\xBAs\xC3\xADtv\xC3\xA1nykiad\xC3\xB3
---
```

```
No client certificate CA names sent
---
SSL handshake has read 3990 bytes and written 424 bytes
---
New, TLSv1/SSLv3, Cipher is DHE-RSA-AES256-SHA
Server public key is 2048 bit
Secure Renegotiation IS NOT supported
Compression: zlib compression
Expansion: zlib compression
SSL-Session:
    Protocol  : TLSv1
    Cipher    : DHE-RSA-AES256-SHA
    Session-ID:
    Session-ID-ctx:
    Master-Key: E352DFC780535B1ADB04EE752433B29130CB2CD8C044A197E20504B5B9C580FD75A17F58FC344E6E5C12387CDBB9DFAD
    Key-Arg   : None
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Compression: 1 (zlib compression)
    Start Time: 1355734537
    Timeout   : 300 (sec)
    Verify return code: 20 (unable to get local issuer certificate)
---
```

Nekünk most a certificate mentés a feladatunk, ezért a *—-BEGIN CERTIFICATE—-* és *—-END CERTIFICATE—-* közötti részt kell egy fájlba menteni az ismert parancssori eszközökkel és készen is vagyunk a feladattal:

```
openssl s_client -connect gep.nev.hu:443 </dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > ➥
    certificate.pem
```

Java - Nyelvfüggő rendezés

A Java *Collator* – ami egy abstract base class – osztály locale-sensitive *String* rendezést képes megvalósítani. Ez a fogalom már a HSQLDB írásban is felbukkant. A konkrét példányokat a *Collator.getInstance()* gyártómetódussal lehet lekérni, ahol azt is beállíthatjuk a paraméterben, hogy milyen lokalitást szeretnénk használni. A Java alapértelmezésben az unicode kódlap szerinti sorrendet használja, ezt tudjuk felülbírálni ezzel a lehetőséggel. Használata könnyen megérthető az alábbi példaprogram tanulmányozásával.

```
1  package org.cs.test;
2
3  import java.text.Collator;
4  import java.util.Locale;
5
6  public class Test
7  {
8      public static void main(String[] args)
9      {
10
11         String[] s = new String[4];
12         s[0] = "álma";
13         s[1] = "csata";
14         s[2] = "123";
15         s[3] = "CUKOR";
16
17 //         Collator myCollator = Collator.getInstance();
```

```
18          Collator myCollator = Collator.getInstance(new Locale("hu", "HU"));
19
20          String temp;
21          int j;
22          boolean b;
23          for (int i = 1; i < s.length; i++)
24          {
25              j = i;
26              b = true;
27              while (j > 0 && b)
28              {
29                  if (myCollator.compare(s[j], s[j - 1]) < 0)
30                  {
31                      temp = s[j - 1];
32                      s[j - 1] = s[j];
33                      s[j] = temp;
34                  } else
35                  {
36                      b = false;
37                  }
38                  j--;
39              }
40          }
41
42          for (int i = 0; i < s.length; i++)
43          {
44              System.out.println(s[i]);
45          }
46      }
47  }
```

A 11-15 sorok között hoztuk létre a rendezendő *s* nevű String tömböt. A 18. sorban létrehozott *myCollator* objektum a magyar kultúra szerint rendezni képes példányt jelent, amit a 29. sor feltétel vizsgálatánál használunk. A példában mi írtuk meg a rendezési algoritmust is, de talán sokkal „Jávássabb" lett volna implementálni a *Comparable* interfészt, amit átadhatunk a már kész rendező algoritmusoknak.

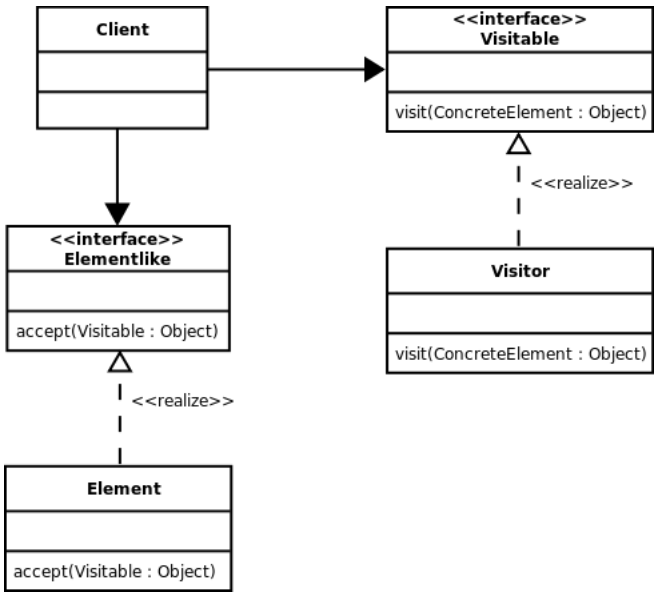
DocBook és PDF - dblatex

A DocBook egy XML szabvány, amiben komplett könyveket, publikációkat lehet XML-ben leírni. Persze a végén valamikor esztétikus kinézetű PDF dokumentumokat akarunk látni, amit a *dblatex* (webhelye: `http://dblatex.sourceforge.net/`) segédprogrammal tudunk könnyen elérni. Ez a háttérben a LaTeXkeretrendszert használja. Mi a problémába az *Ubmrello* nevű UML design eszköz kapcsán futottunk bele, aminek a dokumentum kimeneti formátuma a *DocBook*.

9. A Visitor tervezési minta

A Visitor (Látogató) minta célja az, hogy objektumok valamilyen halmazán műveleteket hajtsunk végre úgy, hogy ehhez az objektumok kódját nem kell megváltoztatnunk. Ezzel a technikával elkülönülten tarthatjuk az adatszerkezetet attól az algoritmustól, amit a benne lévő objektumokra alkalmazni szeretnénk. Ez utóbbi igény akkor merül fel, amikor új metódusokat adnánk az objektumhoz, azonban az nem tudjuk (vagy valamilyen megfontolások miatt nem akarjuk) megtenni.

Áttekintés



9.1. ábra: A Visitor minta

A *Visitor* egy olyan osztály, mely meg tud látogatni más osztályokat. Ez azt jelenti, hogy van egy annyi-szorosan túlterhelt metódusa (amit a példában most *visit()* névre kereszteltünk), ahányféle osztály meglátogatására szeretnénk felkészíteni. Ezen meglátogatandó osztályok objektumait szemléletesen ilyen neveken szoktuk emlegetni: csomópont (*node*), elem (*element*), hely, objektum. A *visit()* (vagy abban a szerepben lévő más nevű) metódus paramétere a meglátogatandó osztályobjektum, így a megfelelő metódus változat hívása egyértelmű lesz. A meglátogatandó szerkezet (például egy hálós adatszerkezet) általában több node-ból áll. A elemeknek rendelkezniük kell egy olyan metódussal (a példában *accept()*), amelyben a visitor *visit()* metódusa kerül visszahívásra (*callback* mechanizmus). Egy kliens programrész – ahogy azt a 9.1. ábrán látjuk – közvetlenül a *Visitable* (azaz egy tetszőleges *Visitor* objektum felülettel) és az *Elementlike* (azaz egy meglátogatható tetszőleges *Element* objektum látogatást elfogadó felületével) objektumaival érintkezik, azaz őket látja, a *visit()* és *accept()* jellegű metódusaikat tudja használni.

Egy példa a látogató mintára

A következőkben a meglátogatható objektumok egy lakás helyiségei lesznek, amin különféle műveleteket szeretnénk végezni. A helyiségek együttese lesz a Lakás, ami a helyiségek halmaza, így ez lesz a bejárandó container vagy adatszerkezet. A 9-1. Programlista *LakasHelyiseg* interfészét, ezzel a meglátogatható objektumok közös felületét vezeti be. Esetünkben bármely olyan objektum meglátogatható egy *LakasVisitor*-ral (9-2. Programlista), ami ezt az interfészt nyújtja. Lényeges észrevenni, hogy úgy tudtunk egy új műveletet hozzáadni a meglátogatott objektumhoz, hogy ezt az *accept()* hívja vissza, de a kódja a látogatóban lesz.

```
// 9-1. Programlista: LakasHelyiseg.java

package cs.test.dp.visitor;
// Egy lakáshelyiség képes fogadni egy ➥
    látogatót
public interface LakasHelyiseg
{
    void accept(LakasVisitor visitor);
}
```

Egy *LakasVisitor* felület annyi túlterhelt *visit()* metódust tartalmaz, ahány fajtájú lakáshelyiség meglátogatási képességére szeretnénk felkészíteni. Esetünkben ez most 4 helyiség típus lesz: szoba, konyha, mellékhelyiség és az egész lakás. Fontos kiemelni, hogy persze ez is csak egy felület, amivel sokféle feladat elvégzésére alkalmas, különféle látogatókat készíthetünk.

```
// 9-2. Programlista: LakasVisitor.java

package cs.test.dp.visitor;

//
// A lakás helyiségekből áll
//
public interface LakasVisitor
{
    public void visit( Szoba szoba );
    public void visit( Konyha konyha );
    public void visit( Mellekhelyiseg ➥
        mellekhelyiség);
    public void visit( Lakas lakas); // ez itt➥
         egy kulcsmomentum
}
```

A 9-3, 9-4 és 9-5. Programlisták a konyha, mellékhelyiség és szoba meglátogatható helyeket, azaz osztályokat adják meg. Az *accept()* metódus mindegyiknél ugyanazt a *visitor.visit()* visszahívást tartalmazza, átadva a *this* paramétert, amivel egyértelmű lesz, hogy a visitor melyik *visit()* metódusa lesz kiválasztva. Részletesebben szólva egy *Szoba* objektum esetén például a *visit(Szoba szoba)* változat lesz automatikusan visszahívva ezzel a *this* paraméterezéssel.

```
// 9-3. Programlista: Konyha.java

package cs.test.dp.visitor;

public class Konyha implements LakasHelyiseg
{
    public void accept(LakasVisitor visitor)
    {
        visitor.visit( this );
    }
}
```

```
// 9-4. Programlista: Mellekhelyiseg.java

package cs.test.dp.visitor;

public class Mellekhelyiseg implements ➥
    LakasHelyiseg
{
    private String name;

    public Mellekhelyiseg(String name)
    {
        this.name = name;
    }

    public void accept(LakasVisitor visitor)
    {
        visitor.visit( this );
    }

    /**
     * @return the name
     */
    public String getName()
    {
        return name;
    }
    /**
     * @param name the name to set
     */
    public void setName(String name)
    {
        this.name = name;
    }
}
```

```
// 9-5. Programlista: Szoba.java

package cs.test.dp.visitor;

public class Szoba implements LakasHelyiseg
{
    private String name;

    //A szoba neve a name
    public Szoba(String name)
    {
        this.name = name;
    }

    // Elfogadja és ezzel végrehajtja a ➥
        visitor tevékenységét
    public void accept(LakasVisitor visitor)
    {
        visitor.visit( this );
    }

    /**
     * @return the name
     */
    public String getName()
    {
        return name;
    }

    /**
     * @param name the name to set
     */
    public void setName(String name)
    {
        this.name = name;
    }
}
```

A 9-6. Programlista a *Lakas*, azaz a csomópontok adatszerkezetét hozza létre. A *LakasVisitor* megvalósítások (például *PrintLakasVisitor*) esetünkben még ennek az összetett szerkezetnek is megadják a *visit()* metódusát.

```
// 9-6. Programlista: Lakas.java

package cs.test.dp.visitor;

public class Lakas
{
    public LakasHelyiseg[] helyisegek;

    public Lakas()
    {
        this.helyisegek = new LakasHelyiseg[]
        {
            new Konyha(),
            new Szoba("Nappali"),
            new Szoba("Háló"),
            new Szoba("Gyerek"),
            new Mellekhelyiseg("WC"),
            new Mellekhelyiseg("Kamra"),
            new Mellekhelyiseg("Előszoba")
        };
    }
}
```

A *PrintLakasVisitor* (9-7. Programlista) egy nagyon egyszerű visitor megvalósítást mutat. A megvalósítandó feladat nyilván sok más látogató típus ötletét is életre keltheti. Amennyiben például a helyiségekben mért hőmérsékletek és azok feldolgozása számára akarnánk egy visitor-t készíteni, akkor annak a neve például *HomeroLakasVisitor* lehetett volna.

```
// 9-7. Programlista: PrintLakasVisitor.java

package cs.test.dp.visitor;

public class PrintLakasVisitor implements
    LakasVisitor
{
    public void visit(Szoba szoba)
    {
        System.out.println( "\nSzoba: " +
            szoba.getName() );
    }

    public void visit(Konyha konyha)
    {
        System.out.println( "\nKonyha " );
    }

    public void visit(Mellekhelyiseg
        mellekhelyiség)
    {
        System.out.println( "\nMellékhelyiség:
            " + mellekhelyiség.getName() );
    }

    public void visit(Lakas lakas)
    {
        for ( int i=0; i<lakas.helyisegek.
            length; i++ )
        {
            lakas.helyisegek[i].accept(this);
        }
    }
}
```

Végezetül teszteljük le a látogató minta alapján létrehozott osztályokat és vegyük észre, hogy ez a kliens tényleg csak a megfelelő interfészeken keresztül éri el az objektumokat (9-8. Programlista)!

```
// 9-8. Programlista: Test.java

package cs.test.dp.visitor;

public class Test
{
    static public void main(String[] args)
    {
        Lakas lakas = new Lakas();
        LakasVisitor printVisitor = new PrintLakasVisitor();
        printVisitor.visit(lakas);
    }
}
```

10. Az Adapter tervezési minta

Az adapter tervezési minta annak a helyzetnek a megoldása, amikor van egy osztályunk, ami alapvetően azt csinálná, amit a kliens használni szeretne, azonban azt nem a megfelelő felülettel szolgáltatja. Ez a probléma az informatikán kívül is számos helyen felbukkan. Gondoljunk csak a gépkocsi szivargyújtó és a mobiltelefonos töltő USB csatlakozójára.

Az adapter gondolata

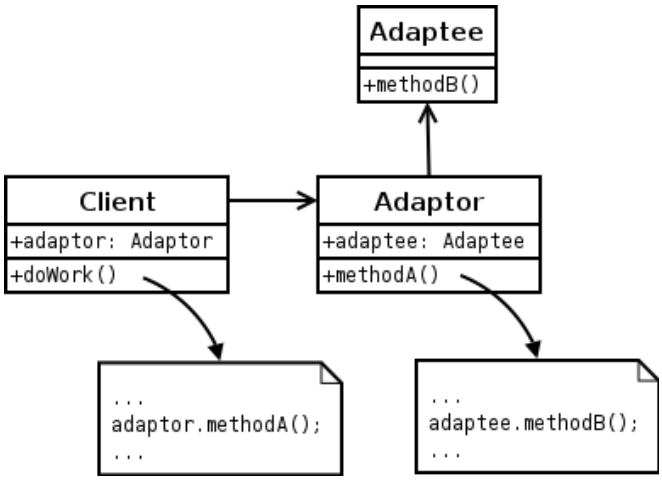
Az adapter lényegét a 10.1. ábra érzékelteti. Képzeljünk el 3 szereplőt:

1. Az ábra bal oldali csatlakozója: *BCS*
2. Az ábra jobb oldali adaptere: *JCS*
3. A fali konnektor (nincs az ábrán): *FK*

A bal oldali csatlakozó objektumot nem tudjuk bedugni a konnektorba, ezért a jobb oldali adaptert hívjuk segítségül. Az *FK* fali konnektor egy olyan szolgáltatás (áramot ad), amit *BCS* eszköz igénybe szeretne venni, de nem tudja, mert nem illeszthető *FK*-ba. A problémát a *JCS* adapter oldja meg.

10.1. ábra: Egy tipikus Adapter

A 10.2. ábra UML diagramja a fentiek modellezése. A *Client* osztály esetünkben a *BCS*, hiszen ő szeretne áramhoz jutni, azaz szolgáltatást igénybe venni. A *JCS* az adapter, azaz a diagram *Adaptor* osztálya. Végül az *FK* az adaptálandó osztály (*Adaptee*), a hozzá való illesztést kell megoldani.



10.2. ábra: Az Adapter Design Pattern

Egy Adapter példa

Van egy kliens programunk (10-4. Programlista), aminek szüksége lenne egy olyan veremre (stack adatszerkezet), amire *String* típusú objektumokat tudunk elhelyezni. A probléma az, hogy ilyen verem osztályunk nincs, azonban helyette rendelkezünk egy *EgyList* osztállyal (10-2. Programlista), aminek a funkcionalitása lényegében a verem lehetőségeket is tartalmazza. A *Stack* interface (10-1. Programlista) pontosan megfogalmazza azt a felületet, amire a kliens *Test* class-nak szüksége van. Ez egy lényeges pont, hiszen így tudjuk egyértelműen definiálni a kliens szükségletét. Amikor egy kliens program valamilyen szolgáltatás halmazt szeretne használni, szinte mindig előnyünk származik abból, ha először annak csak az interfészét határozzuk meg. Ez még akkor is igaz, ha végül ennek az interfésznek az implementációját nem adatperrel,

azaz egy köztes delegáló osztállyal oldjuk meg.

```
// 10-1. Programlista: Stack.java

package cs.test.dp.adapter;

public interface Stack<T>
{
    void push(T o);
    T pop();
    T top();
}
```

Az *EgyList* osztály metódusai olyanok, amik egy lista esetén megszokottak:

- insertTail
- removeTail
- getTail

Az implementációját most látjuk, de ez nem lényeges az Adapter minta szempontjából. A fontos csak az, hogy ismerjük a pontos specifikációját az egyes metódusainak, így ezzel a nyilvános API-ja pontosan ismert legyen a számunkra. Esetünkben Ő az *Adaptee* class.

```
// 10-2. Programlista: EgyList.java

package cs.test.dp.adapter;

import java.util.ArrayList;

public class EgyList<T>
{
    ArrayList<T> al;

    public EgyList()
    {
        al = new ArrayList<T>();
    }

    public void insertTail(T o)
    {
        al.add(o);
    }

    public T removeTail()
    {
       return al.remove( al.size() - 1 );
    }

    public T getTail()
    {
        return al.get( al.size() - 1 );
    }
}
```

Az adaptert az *EgyListStackAdapter* class implementálja, Ő van az *Adaptor* szerepben. Kizárólagos feladata, hogy egy *Stack* felületet biztosítson a kliensek számára, miközben ezt a szolgáltatást az *EgyList* adaptee-ra bízza. Esetünkben mindez nem túl bonyolult, ahogy azt a 10-3. Programlista is mutatja.

```
// 10-3. Programlista: EgyListStackAdapter.➥
    java

package cs.test.dp.adapter;

public class EgyListStackAdapter<T> extends ➥
    EgyList<T> implements Stack<T>
{
    public void push(T o)
    {
        insertTail(o);
    }

    public T pop()
    {
        return removeTail();
    }

    public T top()
    {
        return getTail();
    }
}
```

Ennyi előzetes után a *Test* kliens képes használni a *Stack* felületen az *EgyList* class szolgáltatásait.

```
// 10-4. Programlista: Test.java

package cs.test.dp.adapter;

public class Test
{
    public static void main(String[] args)
    {
        Stack<String> stack = new ➥
            EgyListStackAdapter<String>();
        for (int i=0; i<10; i++)
        {
            stack.push( "" + i );
        }

        for (int i=0; i<10; i++)
        {
            System.out.println( stack.pop() );
        }
    }
}
```