Clipper-programok Linux alatt

Emlékszik még valaki a Clipper (XBase) fejlesztői környezetre? Az elmúlt két évtizedben többmillió programsor született ezen a nyelven.

Jelentős azoknak a programozóknak a száma, akik jól ismerik, és még ma is készségszinten tudják használni ezt az eszközt. Napjainkban is számos Clipper-alapú rendszert használunk (a legismertebb talán a Volán Elektronika LIBRA ügyviteli csomagja), arra is találunk azonban példát, hogy az MS Visual FoxPróban új fejlesztések indulnak el. A DBase fájlformátum mindenesetre nagyon elterjedt, így minden rendszer alatt léteznek azok a segédprogramok, amelyekkel kezelni lehet őket. Valószínűleg ezek a felismerések keltették életre azt a projektet, amelyik a clip rendszert megalkotta és jelenleg is fejleszti. A clip csomag nagyon dinamikusan fejlődő GNU-program, amelyet 2001 nyarán fedeztem fel a ➲ http://www.linux.org gyűjteményben kutatva. A program és a fejlesztők honlapja a ➲ http://www.english.itk.ru/clipper/index.html címen érhető el, ahonnan mindig letölthető a legfrissebb változat.
A cikk írásának idején a `clip-prg-0.99-1.tgz` volt a legfrissebb csomag.
Annak idején a Clipper-környezet a Windows 9x/NT és a hálózatos (SQL-kiszolgáló) technológia térhódítása következtében került ki a fejlesztők eszköztárából. Miért? A két fő ok valószínűleg a következő volt:

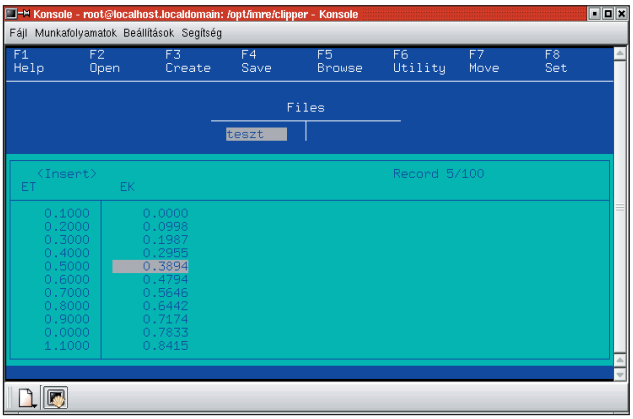
- A Clipper nem volt felkészítve a többrétegű alkalmazások előállítására, ezért az ebben írt programok szükségtelenül terhelték a hálózatot. Az adatok és a Clipper programok általában egy fájlkiszolgálón voltak találhatók, csakhogy a program futtatása és az adatkezelés az ügyfélgépen történt. Ez ebben a formában tarthatatlan volt, hiszen a hálózaton mindig minden adat átment a DBF-fájlok és a program memóriája között.
- A Clipper alapvetően nem grafikus felhasználói felületre lett kitalálva – a puritán kinézetű szöveges képernyők nem feleltek meg a kordivatnak.

A clip rendszer

A clip csomag telepítése után elkezdtem vizsgálni, vajon a létező régi nagyobb Clipper-programjaimat (teljesítményelszámolás, nyugdíjpénztár stb.) le tudom-e fordítani. A másik kérdés az volt, hogy a program használata során a sikeres fordítás és szerkesztés után tapasztalok-e valamilyen hiányosságot. A programok fordítása és szerkesztése a régi `Makefile`-ok átírását jelentette. Számomra hatalmas meglepetés volt, hogy a clip a több tízezer sorból álló programokat kifogástalanul lefordította. A bináris programok kipróbálása során semmilyen hibát nem tapasztaltam, az alkalmazások kifogástalanul működtek. Csodálatos volt arra gondolni, hogy ez a bináris kód valódi, ELF-formátumú Linux-program. A régi rendszerek okozta kellemetlenségek (`CLIPPER`, `FILES` környezeti változók, kevés memória) teljesen eltűntek. Ez a sikerélmény vezetett oda, hogy alaposabban is elkezdtem tanulmányozni a rendszert, amiről kiderült, hogy sokkal többre is képes, mint gondoltam.
A leírás szerint a környezet jelenleg a következő szolgáltatásokat kínálja:

- Az alapcsomag teljesen csereszabatos a CA Clipper 5.3-mal.
- Programozói illesztőfelület létezik az Oracle, PostgreSQL és MySQL adatbázis-kezelőkhöz. Ez lehetővé teszi, hogy adatainkat ne csak *.DBF fájlokban tároljuk, ami meglehetősen izgalmas és új nézőpont az XBase-programozók számára.
- Grafikus felhasználói kezelőfelület- (GUI) csomag (a GTK+ elemkészletre alapozva) használható.
- A TCP/IP csatolója elérhető. Ennek tanulmányozásához a clip SMTP-elem megvalósításának átnézését ajánlom, ami egyébként programjainkból elektronikus levelek könnyű és önműködő küldését teszi lehetővé.
- A `gzip`, `bzip2` tömörített formátumokat támogató csomag.
- A `crypto` titkosított formátumot kezelő csomag.
- Folyamatok kezelése (párhuzamos programozás).
- A soros kapu kapcsolattartását kezelő csomag.
- Teljes nemzeti támogatás (I18N).

A clipet a következő rendszerekre ültették át: Linux, FreeBSD, OpenBSD, Windows 9x/NT/2000.



1. kép A Clipper DBU programjának linuxos változata

A nagyobb programok fordításának és futtatásának bemutatására az *1. kép* szolgál. A kép a Clipper DBU programot futás közben mutatja. A program Clipper nyelven készült és lefordítása semmi gondot nem okozott a clip számára.

Érdekes észrevételek

Régi programjaim clippel fordított változatának nézegetése során rájöttem, hogy a `telnet`, illetve az `ssh` vagy az X-terminál használata során én igazából vékony ügyfél vagyok. Az adatok és a programok emiatt nem terhelik szükségtelenül a hálózatot. Ugye, milyen izgalmas? A Linux egyszerű módon biztosít olyan programhasználati környezetet, ami Windows alatt csak a kétrétegű alkalmazások bevezetésével érhető el. A bináris programok kisméretűek. Ez azért lehetséges, mert nincsenek összeszerkesztve a clip könyvtáraival. A Linux kor-

1. lista Függvénytábla készítése clippel

```
// fuggveny.prg - F ggvØnytÆbla
// lØtrehozÆsa
bKod = { | p1, p2 | 2*p1*p2 }
// Ez egy k dblokk

use TESZT      // a TESZT tÆbla megnyitÆsa
               // rØgi st lusban
zap            // a TESZT tÆbla sorainak
               // t rlØse

x = 0 // Az ET-on fut vØgig
y = 0 // Az EK-en fut vØgig

do while ( x < 10 )
  y = Transzformacio( x )
// alkalmazzuk a f ggvØnyt
  x = x + 0.1
// 0.1 a lØpØsk z
  append blank
//  res rekord lØtrehozÆsa
  replace ET with x        // ki rÆs a
  replace EK with y        // tÆblÆba
end do
use
// a TESZT tÆbla zÆrÆsa
?
return // a program vØge

// Ennek a f ggvØnynek kØsz tj k el
// a tÆblÆjÆt
function Transzformacio( par )
  t := sin( par )
  if ( t < 0 )
     t := 10
     t := Eval( bKod, t, 2*t)
// a k dblokk hasznÆlata (t == 400 lesz)
  end if
return ( t )*
```

szerű felépítésű osztott objektumkönyvtárainak (pl.: ***libclip.so***) használata gazdaságos méretű és gyorsan betölthető programok létrehozását teszi lehetővé.
A kis méretű programok és a Clipper nyelv nem annyira szigorú típuskezelése a jól alkalmazható „ragasztó” nyelvként való használatot is lehetővé teszi. A `run` parancs bármely tetszőleges Linux-parancsot lefuttatja (pl.: `run ls`). Ezt a kellemes tulajdonságot a clip kiváló előfeldolgozója (preprocessor) még teljesebbé teszi. A C/C++ nyelveken megírt kódrészletek egy clipprogramba könnyen beilleszthetők.
A GTK+-ra épülő illesztés szép kinézetű grafikus programok készítését teszi lehetővé, hogy azokat akár távoli X-terminálról (más szóval: X-kiszolgálón keresztül) is könnyedén használni lehessen.
A clip futtatható környezet a Java-bájtkódhoz nagyon közeli elgondolásra épül. A clip forrásprogramok is bájtkódra fordíthatók, amelyeket a clip virtuális gép hajt végre. Ez könnyen megfigyelhető, ha egy clip forrásprogramot C nyelvre fordítunk. Látni fogjuk, hogy a futtató környezet indítása egy új clip virtuális gép létrehozásával indul.

A clip korszerű nyelvi elemei

A Clipper 5.3 már elég fejlett nyelv volt. A régi – DBase-es parancsokat – az előfeldolgozó valódi függvényhívásokká alakította, gondolatvilágában tehát a C nyelvhez áll közel. Ez természetesen azt is megmutatja, hogy a clip milyen fejlett makrózási lehetőségekkel bír.
A `BEGIN SEQUENCE` utasítás korszerű kivételkezelést (Exception handling) valósít meg. Nézzük csak!

```
// KivØtelkezelØs
begin sequence
  ? "Hiba elıtt..."
  hiba = .T.
  if hiba
    ? "HibÆs Ællapot, break..."
    break 3*5 // a break utÆn egy Ætadand
              //  ØrtØk  rhat
  end if
  ? "Ide sohasem j n a vezØrlØs..."
recover using dobott_ertek
  ? "Kezelem a hibÆt...:", dobott_ertek
// a 3*5=15 kiirÆsa
end sequence
```

A fenti példában a `break` (hasonlóan a C++ vagy Java `throw` utasításához) kivételes helyzetet hoz létre, majd eldob egy értéket, mely kifejezés eredménye lehet. A kivétel kiváltása miatt a program a `recover` ágon folytatódik (a C++ vagy Java `catch` ágaihoz hasonlóan). A clip egy beépített `Error` osztályt is tartalmaz, így a `break` a `recover` ág felé akár ezt is dobhatja.
Újdonság a C `switch` utasításának bevezetése, amelyekkel hatékony elágazásokat lehet szervezni, hiszen a program itt a megfelelő ágra való közvetlen ugrással fut. A régi `case` utasítás továbbra is hasznos, néha azonban a `switch` sokkal hatékonyabb. A `case` hátránya, hogy fentről lefelé minden ágra kiszámolja a logikai kifejezés értékét, majd – ha létezik ilyen – az első igaz ágat hajtja végre.
A `@` műveletjel a clipben a referencia szerinti hivatkozást segíti, amiről a C++ és Java-programozók tudják, mennyire hasznos:

```
x = 100
y = @x    // Az y az x egy mÆsodneve lesz
? y       // 100-at fog ki rni
```

A clip új osztályokat is képes bevezetni, amit külön pontban ismertetünk, e helyütt csak annyit említünk meg, hogy a klasszikus `TBrowse`, `TBColumn`, `Get` és `Error` osztályok a Clipper 5.3-ban is léteztek. A nagy hiányosság mindig is az volt, hogy nem tudtunk új típust, azaz osztályt létrehozni. Nos, a clipben ezt is megtehetjük. Sőt, erre épül a rendszerbe vont lehetőségek jelentős része. Jellegzetes módszer a clip fejlesztőitől, hogy valamilyen új szolgáltatáshalmazt egy-egy új osztály bevezetésével tesznek könnyen elérhetővé.

A clip telepítése és fontosabb programjai

A clip jobb megismerése érdekében első lépésként telepítsük Linux-rendszerünkre!
A `clip-prg-<xxx>.tgz` nevű csomagot megtaláljuk a fejlesztők honlapján. A `<xxx>` mindig a pillanatnyi változatszámot jelenti, ami jelenleg `0.99 patchlevel1`. A clip rendszer fordítása és telepítése egyszerű: a telepítés célhelyét a */usr/local* vagy a */opt* könyvtár alatt érdemes kiválasztani. A példában a */opt* helyet választottam.

2. lista A program a Budapest és Esztergom szavakat írja ki a képernyőre

```
////////////////////////////////////////
// OsztÆly Øs objektum
lampa1 := TlampaNew()
// lØtrehozzuk a lampa1 objektumot
lampa2 := TlampaNew()
// lØtrehozzuk a lampa2 objektumot

// tagf ggvØny-h vÆsok, amivel
lampa1:setHely("Budapest")
// beÆll tjuk a lÆmpa helyØt
lampa2:setHely("Esztergom")
// az adattagokhoz k zvetlen l
? lampa1:hely
// nyœlunk hozzÆ
? lampa2:hely
?
return


////////////////////////////
// Egy œj osztÆly
////////////////////////////
function TLampaNew()

  obj := map() // egy œj objektum

// kØt adattagja
  obj:szin := "piros"
// lesz ennek az osztÆlynak
  obj:hely := "ismeretlen"

// a tagf ggvØnyeknek itt csak a neve adott
  obj:setHely  := @setHelyF()
  obj:nextSzin := @nextSzinF()

return obj // A lØtrehoz  mßvelet visszaadja
           // az œj objektumot

////////////////////////////////////////
// Itt vannak a tagf ggvØnyek megval s tÆsai
static function  setHelyF( pHely )
                 ➥::hely := pHely
return NIL

static function nextSzinF( pnextSzin )
                ➥::szin := pnextSzin
return NIL
```

A fenti csomagot másoljuk be a */opt* könyvtárba, majd adjuk ki a következő parancsot:

```
gunzip  clip-prg-0.99-1.tgz
tar xzvf clip-prg-0.99-1.tgz
```

A forrásprogramok telepítését ezzel befejeztük, helyük a */opt/clip-prg-0.99-1* könyvtár lett. Ebben a könyvtárban többféle `make parancsfÆjl` is található: `mkdeb`, `mkrpm`, `mklocal`. Az első két parancsfájl segítségével Debian vagy Red Hat csomagformátumú, terjesztésre is alkalmas változatot hozhatunk létre. Az `mklocal` parancsfájl a clip rendszert lefordítja és telepíti a helyi gépre. Az `mklocal` lefuttatása esetén a használható, bináris formátumú rendszer a */opt/cliproot* könyvtárban lesz. A telepítés utolsó lépéseként a *.bashrc* fájlban a következő változtatásokat tegyük meg:

```
export CLIPROOT=/opt/cliproot
PATH=$PATH:$CLIPROOT/bin
```

Pillantsunk be a *$CLIPROOT/bin* könyvtárba, ahol a fordítóprogram, a forrásnyelvi hibakereső és számos hasznos segédprogram lakozik. A legfontosabb program a `clip`. Egy **.prg* forrásprogramot a `-e -M` kapcsolóval fordíthatunk le. A `clip` a hivatkozott **.o* objekt és **.a* könyvtárak összefűzését is elvégzi (lásd a rendszerhez mellékelt `Makefile`-okat). A `clip -h` parancs az összes megadható kapcsolót kiírja. A `-b` a hibakiírásokat is beszerkeszti a programba. A `-P` csak a forrásprogram előfeldolgozását (makrók feloldását) végzi el. A `clip_cld` egy forrásnyelvi hibakereső. A további lehetőségek tanulmányozásához a `clip` szolgáltatásainak megvalósításait tartalmazó sok-sok **.prg* és **.c* forrásprogram tanulmányozását ajánlom.

Készítsünk függvénytáblázatot!

Kezdjük a munkát a clip rendszerrel! Első feladatként készítsünk egy függvénytáblázatot, amit a *teszt.dbf* fájlban fogunk tárolni (lásd az *1. képet*). Az `ET`=értelmezési tartomány `(numeric 10, 4)`, az `EK`=értékkészlet `(numeric 10, 4)`. A táblázatban a `TranszformÆci` : ET-->EK függvény értékeit a `(0, 10)` tartományban számoljuk ki.
Az *1. listában* látható program által elkészített táblázat egy részletét az 1. képen is láthatjuk. A *fuggveny.prg* forrásfájl fordítását a `clip -e -M fuggveny.prg` paranccsal végeztük el, ami létrehozza a `fuggveny` nevű futtatható programot.

Kifejezéskiértékelés

A mai napig kedvelem a Clipper azon tulajdonságát, ami lehetővé teszi, hogy kódrészleteket futás közben fordítsunk le. Ezzel a lehetőséggel könnyen készíthetünk algoritmus-szótárakat is. Az alábbi példa egy kifejezéskiértékelő program, ami az adat- és képletbekérő, valamint az eredménykiíró szakasszal együtt is csak nyolc sor.

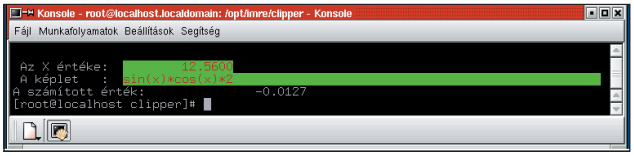
```
// makro.prg - Makr helyettes tØs
X = 0.0000
Keplet = "X" + space(50)

@ 1, 1 say "Az X ØrtØke: " get X picture
➥"9999999999.9999"
@ 2, 1 say "A kØplet    : " get Keplet
read
eredmeny = &Keplet
// Itt t rtØnik a kifejezØskiØrtØkelØs
// (az & jel hatÆsÆra)
? "A szÆm tott ØrtØk: ", eredmeny
```

Ugye, milyen egyszerűen oldottuk meg ezt az összetett feladatot? A programot a `clip -e -M makro.prg` paranccsal fordíthatjuk le. A kapott `./makro` program futását a *2. kép* mutatja.
A továbbiakban röviden nézzük meg a clip következő két haladóbb témakörét: az új osztályok készítését, valamint a külső C/C++ eljárások beépítését.

Az objektumközpontú programozás

A clip új lehetősége, hogy teljes értékű új típusokat készíthetünk benne. Példaképpen egy nagyon leegyszerűsített forgalomirányító lámpa osztályának elkészítését mutatom be. Egy új osztály bevezetésének módja engem leginkább a JavaScriptben alkalmazott megoldásra emlékeztet. A *2. listában* látható program egy `TLampaNew` nevű osztályt határoz meg. A `lampa1` és `lampa2` ennek az osztálynak két objektuma. A tagfüggvények, adattagok meghívását ebben a nyelvben az `objektum:tagf ggvØny`, illetve `objektum:adattag`



2. kép Kifejezéskiértékelés

írásmóddal tehetjük meg. A program a Budapest és Esztergom szavakat írja ki a képernyőre.
A kód tanulmányozásával könnyen megérthető egy új osztály kezdeti értékadása és a tagfüggvénynek a megvalósítása.
A `static` minősítő használatával a függvénynevek a fájlra nézve helyiek lesznek, azaz a megvalósítást elrejtjük a külvilág elől. Ezt célszerű így kódolni, hiszen a tagfüggvényeket úgyis csak az objektumokon keresztül fogjuk meghívni. Az eljárásokat megvalósító függvényeket és az objektumbevezetés során megadott tagfüggvényneveket a `@` műveletjel használatával kötöttük össze. Ez azt jelenti, hogy az `obj:setHely` eljárás egy másodnév a `setHelyF()` függvényre. A `":"` érvényességi kör műveletjel 2. listában való használata azt pontosítja, hogy például a `szin` nem helyi változó, hanem osztályunk egyik adattagja.
A forráprogramok ***clip/classes*** és ***cliplibs*** könyvtárainak programjait mindenki kedvére nézze át, mert sokat lehet tanulni belőle. Egyrészt több érdekes cliposztályt ismerhetünk meg behatóbban, másrészt azt a tudást is finomíthatjuk, amivel saját osztályokat készíthetünk.

Külső C/C++ nyelvű függvények beillesztése

A clip egyik erőssége, hogy könnyedén összeilleszthető vele egy C/C++ nyelven megvalósított algoritmus. Ez azért olyan fontos, mert C/C++-ban a teljes Linux-szolgáltatáskör elérhető. A bonyolult eljárásokat például a C++ ***STL*** (Standard Template Library) könyvtár használatával írhatjuk meg. Tekintettel arra, hogy a C/C++ eljárások előnye nem kíván külön magyarázatot, térjünk át inkább arra, hogyan kell ezt a lehetőséget használni. Első feladatunk egyszerű lesz. Írunk egy függvényt, ennek egyetlen `string` értéke lesz, amit átvesz a hívótól, majd vissza is adja neki.

```
// teszt.c
#include <errno.h>
#include <limits.h>
#include <string.h>
#include "clip.h" // Ez egy clippel szÆll tott
                  // fejÆllomÆny

int clip_TESZT_CFGV( ClipMachine *mp )
{
  _clip_retc(mp, (char *)_clip_parc(mp, 1) );
      return 0;
}
```

A `clip_TESZT_CFGV` függvényt a clip programból `TESZT_CFGV` néven hívjuk meg, azaz a „clip_” előtagot elhagyjuk. Vegyük észre, hogy C függvények értékadása mindig olyan, hogy értékként a futtató clip virtuális gépre irányított mutatót adjuk át. A visszatérés típusa mindig `int`. Nézzük meg a fenti függvény clip programbeli használatát is:

```
// TesztKulsoC.prg
extern TESZT_CFGV
// Ebben a f ggvØnyben Ætadunk egy sz veget,
// amit a f ggvØny lefutÆsakor visszakapunk.
? TESZT_CFGV("Ez egy pr basz veg...")
0?
```

A `./TesztKulsoC` program lefutásakor „Ez egy próbaszöveg…” mondat íródik ki a képernyőre.
Egy pillanatra tekintsünk ismét a `clip_TESZT_CFGV` C függvényre. Látható, hogy a értékek átvétele és átadása a `_clip_par?` és `_clip_ret?` függvények (a ? helyén most „c” van, ami a karakteres típusra utal) használatával történik. Ezen függvények első értéke szintén mindig a clip gépre mutató cím. A második érték egy szám, ami azt mondja meg, hogy a függvénynek átadott értékek közül hanydikat akarjuk lekérdezni.
Most megmutatom, hogyan kell lefordítani és összeszerkeszteni ezt a programot.
A C forrásfájl fordítása a ***gcc -c teszt.c*** parancsal történik (eredményül a ***teszt.o*** objektumfájlt kapjuk).
A futtatható program létrehozása a `clip -e -M TesztKulsoC.prg teszt.o` paranccsal történik.
Ennyi ismerkedés után nézzünk példát arra, hogy a clip fejlesztői miként valósították meg a clip belső `sin(x)` függvényét. A következő listán látható részlet a ***clip _math.c*** forrásfájlból származik.

```
// A clipbıl  gy h vjuk: x = sin(3.45)
int clip_SIN(ClipMachine * mp)
{
  int len, dec;
// egy double ØrtØk ÆtvØtele
  double d = _clip_parnd(mp, 1);
// a tulajdonsÆgok lekØrØse
// (hossz Øs tizedesek)
  _clip_parp(mp, 1, &len, &dec);
  dec = mp->decimals;
  _clip_retndp(mp, sin(d), len, dec);
//  rtØkk ldØs a h v nak.
  return 0;
}
```

A C++-ban írt függvények esetén ne felejtsük el, hogy kódoltan vannak jelen az ***object*** fájlban, így azokat az eljárásokat, amelyeket a clipből meg szeretnénk hívni, `extern "C"` jelzéssel lássuk el – ez megakadályozza a nevek kódolását.

Nyíri Imre
(inyiri@mol.hu) jelenleg a MOL Rt.-nél dolgozik. Informatikai vállalkozásában az Internet, a Linux, valamint a Java programozás gyakorlati hasznosításával foglalkozik, örök szerelme a C++ maradt.