1995. MÁRCIUS/ÁPRILIS ÁRA: 398 Ft.

FORRÁSKÓD

A programozók lapja

3.
4.

FORRÁSKÓD

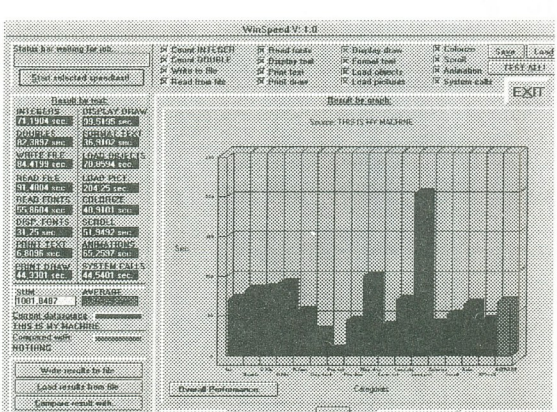
Tisztelt Előfizetőnk!

Lapunk belső problémák miatt összevont számmal jelentkezik. Ebből az előfizetőknek, természetesen nem lehet hátránya. Aki éves előfizetéssel rendelkezik tizenkét lapszámot kap, aki fél évet fizetett ennek a felét. Az előfizetőket mint minden lap mi is kényeztetni szeretnénk. Az előfizetők plusz információkhoz jutnak, számukra alkalmanként plusz lemezt küldünk, ezt F++ -nak neveztük. Ehhez a számhoz a magyar fejlesztésű WINSPEED sebességmérő programot adjuk, kizárólag az előfizetőknek.

Nagy Sándor

WINSPEED

A Windows nem a sebességéről híres. Futtatásához ezért nagy teljesítményű, jól hangolt hardver kell. A hardver valódi teljesítményét Windows alatt mérőprogramokkal állapíthatjuk meg. Ezek rávilágítanak rendszerünk gyenge pontjaira, segítséget nyújtanak annak behangolásához. A Forráskód a hazai fejlesztésű WINSPEED programot ajánlja, elsősorban racionális megközelítése miatt. Zavaros pontrendszer helyett egyszerűen, közérthetően az egyes tesztekhez szükséges időt hasonlíthatjuk össze ismert gépeken mért időkkel. A program lemezről nem fut, telepíteni Windows alól a SETUP.EXE indításával lehet. A program az általunk megadott könyvtárba költözik, de néhány file a Windows \ system alá is kerül. Ez a telepítéskor nyomon követhető, az esetleges takarításkor használt vesszük.



Óbuda
StarKing

MEGHÍVÓ

Tisztelettel meghívjuk Önt és munkatársait a
StarKing Óbuda Apple Center
bemutatótermébe, ahol

1995. április 13. - május 25. között
az "Open Day" napok keretében
megrendezésre kerülő szakmai előadások
csütörtökönként 16 órakor:

- **április 13.**: ArchiCAD-egy jobb építészetért
  [meghívott előadó: Seidl Tibor-Graphisoft/CAD Stúdió,
  bevezető: Callmeyer Ferenc építész, címzetes egy. tanár]
- **április 20.**: Object Master-avagy "Szoftverkészítés mesterfokon"
  [előadó: Flórea György, Varga György-StarKing]
- **április 27.**: Hard Disk Recording-Protools és AVID rendszerek
  [meghívott előadó: Dr. Erdélyi Gábor-Balázs Béla díjas
  hangmérnök, Erdélyi Péter-Vektor Kft.]
- **május 4.**: Újdonságok a Mac világban-IFABO beharangozó
  [meghívott előadó: Cserei Ferenc-HDSys Kft.]
- **május 11.**: **SZÜNET** [IFABO: BNV "A" pavilon 309/8. stand]
- **május 18.**: 4D-Nincsenek gondok, ha 4D az adat gondnok
  [előadó: Flórea György, Varga György-StarKing]
- **május 25.**: Az első magyar zenei CD ROM-Magyar jazz történet
  [előadó: Simon Géza Gábor a JOKA elnöke]

Örülnénk, ha elfogadná meghívásunkat, és megtisztelne bennünket
jelenlétével a jövőben is.

Szívélyes üdvözlettel:

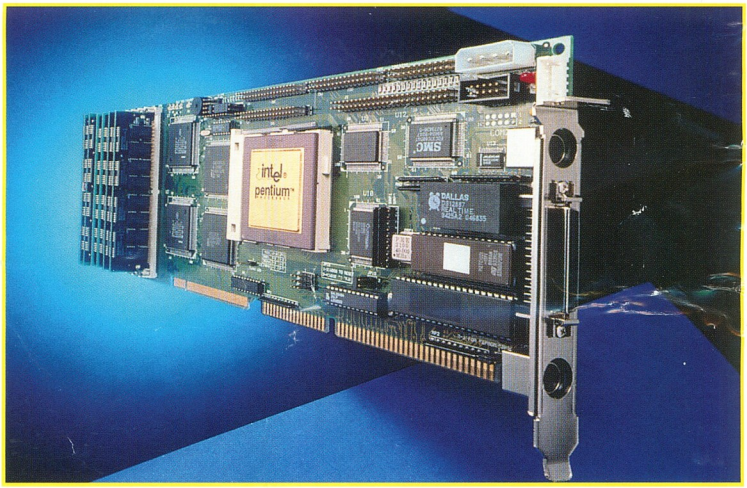
Horváth Péter

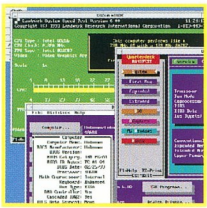
kereskedelmi és marketing igazgató

StarKing Óbuda Apple Center
H-1037 Budapest, Bécsi út 77-79. ☛ tel.: 250-4711 ☛ fax: 212-4832
bemutatóterem: 250-4717

25.

A Pc nem csak az irodákban és az otthonokban állja meg a helyét, de egyre szélesebb körben alkalmazzák az ipar különböző területein, ahol olcsó és megbízható adatgyűjtő, feldolgozó, folyamatirányító rendszerre van szükség.



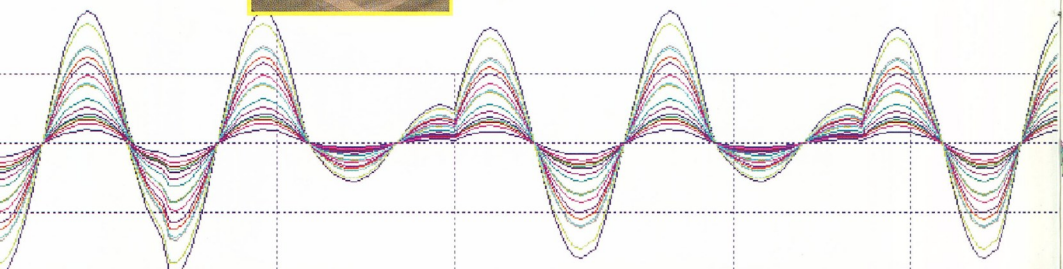


14.

Linux építő sorozatunkban egy, a rendszer bővítéséhez szükséges telepítő programot adunk, és rögtön munkába is állítjuk. Egy teljes és valódi 32-bites C és C++ fordítót telepítünk linuxunk alá, és elkezdjük vele az ismerkedést. A DOS formátumú lemezek kezeléséhez is adunk szerszámot.

44.

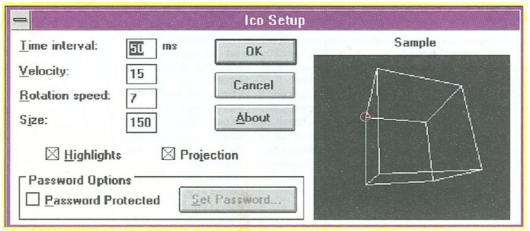
Egy átlagos kiépítettségű számítógép már alkalmas mozgó képek, animációk megjelenítésére, miért mondanánk le a használatukról. Ebben a számban az FLI formátumú animációs file-ok felépítését, kezelését ismertetjük.

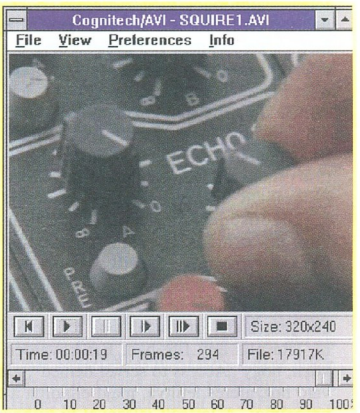


Házimozi

25.

Windows programozói rovatunkban most a haladóknak írunk. Egy képernyő-védő készítése kapcsán több trükköt, mesterfogást ismerhetnek meg.



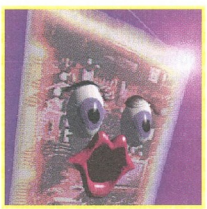


36.

A mérésadatgyűjtő szoftver sokak számára ismeretlen terület. A lehetőségekről az ismertebb termékekről írunk.

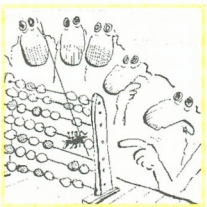
40.

Windows programozóknak még egy ragyogó témát indítunk.
Az MCI interfész felhasználásával egy AVI lejátszót, egy valódi házimozit készítünk két részben. Ebben a számban az alapokat tisztázzuk. A következő részben közöljük a forráslistát, Borland C nyelven, a futtatható változattal együtt.



49.

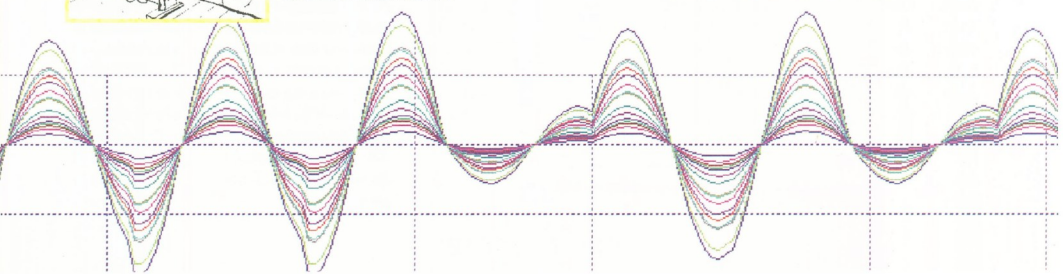
A Gravis hangkártyák programozását ismertetjük, teljes részletességgel.



2.

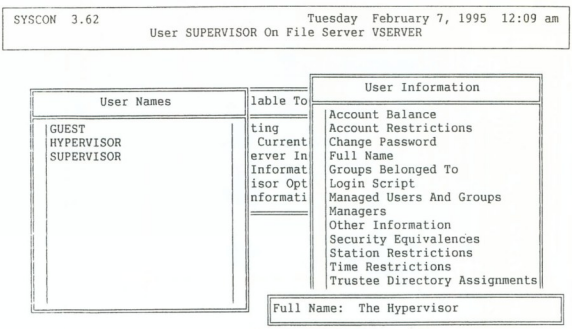
Egy új veszélyforrásról, a szervereket megtámadó vírus lelkéről, működéséről szól ez a cikkünk.

1656.
Budapest
Pf.: 16.



Támadás a szerverek ellen

A Novell NetWare létezése óta több sikeres és sikertelen próbálkozás történt már felhasználói jogok megszerzésére. A kihalófélben lévő 2.xx-es verzióknál sok mérnök kereste meg 1–5 üveg sörét az elfelejtett jelszavak keresgélésével. A 3.xx-es szériára a Novell kódolni kezdte a jelszavakat, mind az ezeket nyilvántartó bindery állományban, mind a kommunikációk során, így sokan kezdtek el új hatásos módszereken gondolkozni. A SUPERVISOR jelszavának megszerzésének legközönségesebb formája a „longin.com-os módszer" volt. A **login supervisor** parancsot kiadva nem a login.exe, hanem a login.com indul el, ami szintén bekéri a jelszavakat, azonban ahelyett, hogy ezt továbbítaná a szerverre, letárolja egy file-ban. Ezt követően közli, hogy „Access to server denied", azaz a szerverhez való hozzáférés letiltva, majd elindítja a tényleges login programot. A letárolt jelszóval ki-ki bejuthatott a hálózatra. Nehezebb esetekre készült később a 3.11-en működő setpass.nlm, ami a mai napig népszerű csodafegyver hardveres körökben. Ezt a nlm-et a console-nál, vagy rconsole-on keresztül lehet betölteni közvetlenül a szerverre. A futtatás után a SUPERVISOR password SUPERVISOR lesz, amit egyébként a Novell, hagyományos módon megadva azt, nem enged meg. Ez a módszer tehát a **ChangeBinderyObjectPassword** hívással utat nyit a serveren.

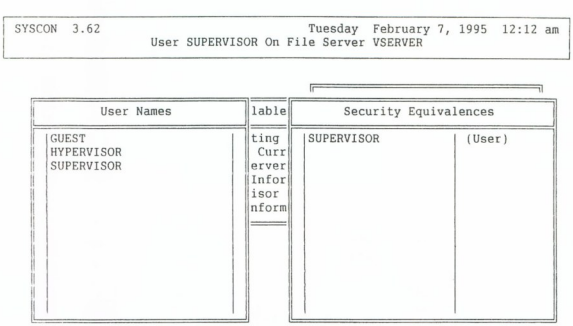


2. ábra

Már 2.xx-es NetWare-t is kénytelen volt a Novell, vírus miatt javítani. Az történt, hogy a Bulgáriában írt Eddie vírus (később több más vírus is) képes volt az **execute-only** attribútummal ellátott file-okat is megfertőzni. Az ilyen attribútummal rendelkező állományokat (hagyományos körülmények között) a Novell csak futtatni engedi, olvasni nem. Ez azért volt kellemetlen, mert sem a SUPERVISOR-nak sem pedig a víruskeresőknek nem volt joga ezen file-okat megnyitni, és így a fertőzést eltávolítani.

A vírusok jelentős része nem csinál mást csak terjed, azt is mondhatnánk egyik olyan mint a másik. Vírusok jönnek, lassan kihalnak és mások új vírusai bosszantanak minket, a bilógiai párhuzam már elkerülhetetlen. Szerencsére kevés vírusíró fejleszti kitartóan, verzióról verzióra vírusait, de ez is éppen elég ahhoz, hogy nagy fejfájást okozzanak mindennapi munkánk során. A BadSectors vírus több verziójának terjedése több mint egy éves múltra tekint vissza hazánkban. Ez a víruscsalád valószínűleg Romániából került Magyarországra. A legújabb hazánkban terjedő Hypervisor vírus szerzője valószínűleg azonos a BadSectors írójával. A Hypervisor egy 3120 byte méretű vírus, COM és EXE file-okat fertőzve terjed, új, félelmetes korszakot nyit a PC-s vírusok fejlődésében.

Ha egy fertőzött program elindul a vírus első lépésként a memória tetején a 640 K alatti területen rezidenssé teszi magát, boot vírusokra jellemző módszerrel a BIOS adatterületen 4 Kbyte-tal csökkenti a maximális memóriaértéket. Egy debug ellenes ciklussal másolja magát a kijelölt területre, így azokat akik nyomkövetéssel akarják visszafejteni könnyen csődbe húzhatja. Ezek után kódjának egy jelentős részét dekódolja. Ezen a területen NetWare REQUEST bufferek, sok szöveges információt tartalmazó adatterülete található, ami fő céljának eléréséhez lesz szükséges. Lekérdezi a NetWare verziószámot az INT 21/E3 Connection control híváson keresztül (ami csak az IPX lefutása után érhető el) és tárolja adatterületén, majd az INT 21, INT 25, INT 26 megszakítások vektorait állítja saját magára. Ezután futtatja a hordozó programot.



1. ábra

```
SYSCON  3.62                          Tuesday  February 7, 1995  12:10 am
                    User SUPERVISOR On File Server VSERVER

        Account Restrictions For User HYPERVISOR
Account Disabled:                     No
Account Has Expiration Date:          Yes
   Date Account Expires:              February 7, 1996
Limit Concurrent Connections:         Yes
   Maximum Connections:               3
Allow User To Change Password:        Yes
Require Password:                     Yes
   Minimum Password Length:           7
Force Periodic Password Changes:      Yes
   Days Between Forced Changes:       40
   Date Password Expires:             March 7, 1995
   Limit Grace Logins:                No
      Grace Logins Allowed:
      Remaining Grace Logins:
Require Unique Passwords:             No
```

3. ábra

```
O:\LOGIN>attrib
        R       O:\LOGIN\TOKEN.RPL
        R       O:\LOGIN\LOGIN.EXE
        R       O:\LOGIN\CONSOLE.COM
        R       O:\LOGIN\SLIST.EXE
    A   SH      O:\LOGIN\NET$OBJ.SYS
    A   SH      O:\LOGIN\NET$PROP.SYS
    A   SH      O:\LOGIN\NET$VAL.SYS

O:\LOGIN>
```

4. ábra

a vírus egy NetWare munkaállomásra kerül? Normál felhasználói jogok mellett természetesen csak azokat a hálózaton levő alkalmazásokat fertőzi meg, amihez az aktuális felhasználónak joga van. De ha SUPERVISOR jelentkezik be a hálózaton, egy fertőzést követően a Hypervisor azonnal célhoz ér. Ekkor ránéz, hogy a NetWare verziószáma nem kisebb-e 2.00-nál és ha nem, meghívja azt a rutint, amivel a SYSCON-hoz hasonlóan létre tud hozni a hálózaton egy HYPERVISOR nevű felhasználót. Először ellenőrzi, hogy létezik-e már ilyen felhasználói név a hálózaton, ha nincs rögtön létrehozza, ha van megnézi, hogy rendelkezik e SUPERVISOR SECURITY_EQUALS-al, azaz SUPERVISOR jogosultsággal. Ha nem úgy dönt, hogy ezt a HYPERVISOR felhaszhnálót nem ő hozta létre, így kitörli azt, majd újra létrehozza. Ezek után sorban megadja az új object-hez tartozó tulajdonságokat, szintén a Connection control-on keresztül. Először SUPERVISOR jogosultsággal látja el **(1. ábra)**, majd az IDENTIFICATION-t, azaz a felhasználó teljes nevét is kitölti „The Hypervisor" tartalommal **(2. ábra)**, ami a SYSCON-ban a full name alatt jelenik meg. Létrehozza a LOGIN_CONTROL tulajdonságot is, amivel biztosítja, hogy a HYPERVISOR névvel egy éven keresztül be lehessen jelentkezni a hálózatra **(3. ábra)**. De a leglényegesebb az, hogy

```
2/7/95 12:04am: 1.1.63 Bindery close requested by user SUPERVISOR on station 2
2/7/95 12:04am: 1.1.61 Bindery open requested by user SUPERVISOR on station 2
```

5. ábra

Innentől kezdve aktívan figyel néhány DOS funkciót. A fertőzött programok méretéből levonja saját hosszát, így a növekedés nem látszik, ha azt a DIR-rel vagy a Norton Commander paneljein próbáljuk észlelni. Így úszhatja csak meg, hogy ez a viszonylag nagy méretváltozás azonnal lebuktassa. Csak a Norton Commander View (F3) funkciójának használatakor derülhet ki, hogy a jobb felső sarokban megjelenő hossz nem egyezik a panelben levővel. Mivel a vírus file nyitásnál, program indításnál fertőz, az előző módszer mellett sajnos rögtön egy fertőzést váltunk ki. A vírus figyeli, hogy COM vagy EXE programról van e szó. Minden olyan esetleges problémára ügyel, ami a fertőzést megakadályozhatná vagy sikertelensége miatt őt lebuktatná, ezért az INT 1B, INT 24 megszakításokat is magára állítja a fetőzések alatt. Így túléli a Crtl-Break-et és a leragasztott lemezen történő DOS hibaüzeneteket, amik az írási próbálkozásra figyelmeztetnék a felhasználót. Nem fertőzi meg a 2048 bytenál kisebb, ill. 62159 byte-nál nagyobb COM programokat.

Szóló gépeken a fertőzésen kívül nem történik más: csak a WINDOWS dobja fel néha a talpát. De mi történik, ha password property-nek nem ad értéket, azaz nem állít be jelszót, így ezek után a HYPERVISOR névvel bejelentkezve bárki SUPERVISOR jogokat élvezhet a megtámadott szerveren. Ezek után a NetWare verziótól függően ellenőrzi, hogy a SYS:LOGIN/ alatt van e már a binderyről másolat. Nevezetesen léteznek e a NET$OBJ.SYS, NET$PROP.SYS, NET$VAL.SYS, ill. 2.xx-nél a NET$BIND.SYS, NET$BVAL.SYS hidden, system, atchiv flaggel ellátott file-ok. Ha nem, kiad egy close bindery-t és egy rövid rutinnal elkészíti a SYS:SYSTEM/ alól a másolatot a SYS:LOGIN/ alá **(4. ábra)**, majd egy open bindery-t használva újra a kiinduláshoz ér.

Az imént részletezett eljárást a Nowell NetWare sem megakadályozni, sem észlelni nem képes. A Security Update segédprogrammal ellátott szerver azonban észleli tudja a jelenséget: néhány sípszó kíséretében két figyelmeztető üzenet jelenik meg a szerveren **(5. ábra)**.

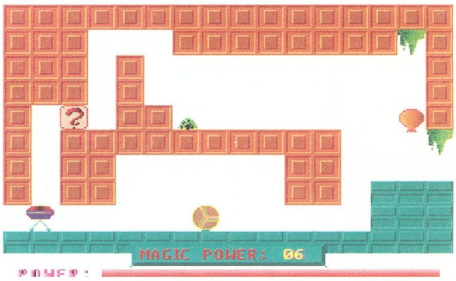
A Hypervisor vírust a Pasteur és a VirusBuster legfrissebb verziói felismerik és el is tudják távolítani.

Szőr Péter és Leitold Ferenc
víruszakértők

Játék assemblerben 4.

HUBERT

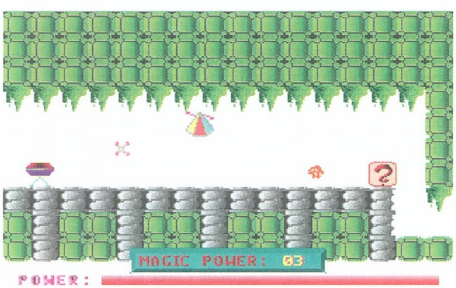
Ahogy az előző számban igértem most az ütközés vizsgálatával, a billentyűzetkezeléssel és a főciklus átnézésével foglalkozunk, mivel a mellékelt lemezen megtalálható a teljes forráskód, ezért mindenki láthatja működés közben is a játékot.



Ütközés vizsgálata:

Az ütközés vizsgálatát az „utkvizsg" procedura végzi el. Ha megnézzük a forrást, akkor rutinosabb programozók számára pár perc alatt kiderül a működési elmélet. Az elmélet nagyon egyszerű. Amit tudunk az az, hogy hol van a képernyőn a figuránk, és hogy éppen hanyadik fázis van kirakva.

Ez utóbbiból kiszámolhatjuk az adott fázis báziscímét. Ezt láthatjuk az első tíz sorban. DI-be beletesszük az képernyőn lévő figura kezdőcímét, az SI-be meg a báziscímét. Mivel tudjuk a figura méretét, és mivel ez a méret nem változik a program során, ezért két egymásba ágyazott ciklussal byte-onként végig tudjuk vizsgálni a képernyőn lévő, és a bázismemóriában lévő figura adatait. Ez az elmélet praktikus, csak több követelménynek teljesülnie kell. Ezek közül a legfontosabb, hogy az ütközésvizsgálatot a főciklusban egy olyan időben hívjuk meg, amikor a figura is, meg a spriteok is ki vannak rajzolva. Ez azért szükséges, mert mi tulajdonképpen azt vizsgáljuk, hogy megváltozott-e a figuránk képe a spriteok kirajzolása után. Magyarul ráírtunk-e vagy sem. A jelenlegi ütközésvizsgálati procedura ki lett bővitve egy vizsgálattal és egy feltételes elágazással. Ez annyit tesz, hogy amikor betöltöm az AL-be a képernyőn lévő byteot utána mielőtt bármit is kezdenék vele megnézem, hogy az értéke nem e nulla. Ha nulla, akkor nem is foglalkozom vele tovább, hanem továbblépek a következőre. Amennyiben nem nulla az értéke abban az esetben hasonlítom csak össze a képernyőn lévő byte, és a bázismemóriában található byte értékét. Ha egyforma az értékük, akkor nem volt ütközés ezen a ponton, ha nem egyforma akkor volt ütközés. A nulla vizsgálata nagyon hasznos dolog esetünkben, mivel tudjuk, hogy a figuránk háttérszíne a nulladik paletta színe. Így tulajdonképpen csak azokat a byteokat fogjuk megvizsgálni, amelyek beletartoznak a figura testébe. Jelen esetben a vizsgálat addig folyik, amíg eltérést nem talál az algoritmus, vagy amíg a végére nem ér a ciklusnak. Ez utóbbi esetben nem volt ütközés. Ütközés esetén betöltődik az AX-be a „power" változó értéke, ami a képernyő alján lévő powercsík hosszát jelenti. A powercsík akkor fogyott el, ha az AX értéke eléri a nullát. Tehát itt van ez egyik kilépő a programból, ha tovább akarja valaki fejleszteni. A másik kilépő a billentyűzet kezelésénél lesz, az „ESC"-nél. Találunk az „utk4" cimke alatti sorban egy „sub" utasítást. Itt határozzuk meg a powercsík fo-



Ahogy az előző számban igértem most az ütközés vizsgálatával, a billentyűzetkezeléssel és a főciklus átnézésével foglalkozunk, mivel a mellékelt lemezen megtalálható a teljes forráskód, ezért mindenki láthatja működés közben is a játékot.

Ütközés vizsgálata:

Az ütközés vizsgálatát az „utkvizsg" procedura végzi el. Ha megnézzük a forrást, akkor rutinosabb programozók számára pár perc alatt kiderül a működési elmélet. Az elmélet nagyon egyszerű. Amit tudunk az az, hogy hol van a képernyőn a figuránk, és hogy éppen hanyadik fázis van kirakva.
Ez utóbbiból kiszámolhatjuk az adott fázis báziscímét. Ezt láthatjuk az első tíz sorban. DI-be beletesszük az képernyőn lévő figura kezdőcímét, az SI-be meg a báziscímét. Mivel tudjuk a figura méretét, és mivel ez a méret nem változik a program során, ezért két egymásba ágyazott ciklussal byte-onként végig tudjuk vizsgálni a képernyőn lévő, és a bázismemóriában lévő figura adatait. Ez az elmélet praktikus, csak több követelménynek teljesülnie kell. Ezek közül a legfontosabb, hogy az ütközésvizsgálatot a főciklusban egy olyan időben hívjuk meg, amikor a figura is, meg a spriteok is ki vannak rajzolva. Ez azért szükséges, mert mi tulajdonképpen azt vizsgáljuk, hogy megváltozott-e a figuránk képe a spriteok kirajzolása után. Magyarul ráírtunk-e vagy sem. A jelenlegi ütközésvizsgálati procedura ki lett bővítve egy vizsgálattal és egy feltételes elágazással. Ez annyit tesz, hogy amikor betöltöm az AL-be a képernyőn lévő byteot utána mielőtt bármit is kezdenék vele megnézem, hogy az értéke nem e nulla. Ha nulla, akkor nem is foglalkozom vele tovább, hanem továbblépek a következőre. Amennyiben nem nulla az értéke abban az esetben hasonlítom csak össze a képernyőn lévő byte, és a bázismemóriában található byte értékét. Ha egyforma az értékük, akkor nem volt ütközés ezen a ponton, ha nem egyforma akkor volt ütközés. A nulla vizsgálata nagyon

hasznos dolog esetünkben, mivel tudjuk, hogy a figuránk háttérszíne a nulladik paletta színe. Így tulajdonképpen csak azokat a byteokat fogjuk megvizsgálni, amelyek beletartoznak a figura testébe. Jelen esetben a vizsgálat addig folyik, amíg eltérést nem talál az algoritmus, vagy amíg a végére nem ér a ciklusnak. Ez utóbbi esetben nem volt ütközés. Ütközés esetén betöltődik az AX-be a „power" változó értéke, ami a képernyő alján lévő powercsík hosszát jelenti. A powercsík akkor fogyott el, ha az AX értéke eléri a nullát. Tehát itt van ez egyik kilépő a programból, ha tovább akarja valaki fejleszteni. A másik kilépő a billentyűzet kezelésénél lesz, az „ESC"-nél. Találunk az „utk4" cimke alatti sorban egy „sub" utasítást. Itt határozzuk meg a powercsík fogyásának a sebességét. Ha nagyobb értéket adunk, akkor gyorsabban fogy a powercsík ütközés esetén. Ha nem vonunk le semmit, akkor egyáltalán nem fogy a powercsík.

Billentyűzet kezelése:

Ezt a funkciót a „Billbeolv" procedura végzi el.
Alapelv, hogy át kell venni a klaviatúra kezelését teljes mértékben. Erre azért van szükség, hogy ne forduljanak elő olyan esetek, amikor a gép a billenyűzetre vár, de legfőképpen azért, mert nem csak azt kell vizsgálni, hogy mi lett lenyomva, hanem azt is, hogy mi lett felengedve. Magyarul ha lenyomjuk a jobbra-nyilat, a figura elindul jobbra. De ha a jobbra-nyíl felengedése előtt lenyomjuk a balranyilat akkor a hagyományos esetben a figura továbbmenne jobbra, sőt a jobbra billentyű felengedése után hiába nyomnánk folyamatosan a balra nyilat, a figuránk nem menne balra. Egyszerűen arról van szó, hogy ha le van nyomva egy billentyű, akkor alapesetben nem jegyzi meg a buffer a másodszorra lenyomott billentyűt. Ezt egy egyszerű szövegszerkesztővel is ki lehet próbálni.
Tehát én azt az egyszerű megoldást választottam, hogy felvettem három byte szintű változót. Ezek a „leftj", „rightj" és a „spacej", amikbe alapállapotként nullát állít be a program (Install procedúra).
Maga a procedúra lelke egy „IN AL, 60h" utasítás. Te-

```
utkvizsg   proc
           mov    di,figcim
           mov    ax,figx
           and    al,15
           shr    al,1
           shr    al,1
           xor    ah,ah
           mov    cx,360              ; sprite_size
           mul    cx
           mov    si,offset fig1_1
           add    si,ax

           mov    cx,18
      utk1: mov   bx,cx
           mov    cx,20
      utk2: mov   al,es:[di]
           cmp    al,0
           je     utk3
           mov    ah,[si]
           cmp    ah,al
           jne    utk4
      utk3: inc   si
           inc    di
           loop   utk2
           add    di,300
           mov    cx,bx
           loop   utk1
           jmp    utkki

      utk4: mov    ax,power
           sub    ax,1    ;ha 1-et von le
           cmp    ax,0    ;akkor lassabban
           jne    utk5    ;fogy a powercsik!
           jmp    uninstall
      utk5: mov    power,ax
           call   poki
           call   beep1

      utkki: ret
           endp
;-------------------------------------
Billbeolv  Proc
           in     al,60h
           cmp    al,39h   ; space lenyomás
           jne    bill1
           mov    al,1
           mov    spacej,al
           jmp    billki
      bill1: cmp    al,4bh   ; left lenyomás
           jne    bill2
           mov    al,1
           mov    leftj,al
           jmp    billki
      bill2: cmp    al,4dh   ; right lenyomás
           jne    bill3
           mov    al,1
           mov    rightj,al
           jmp    billki
      bill3: cmp    al,1     ; esc lenyomás
           jne    bill4
           jmp    kilep
      bill4: cmp   al,0b9h  ;space felengedés
           jne    bill5
           xor    al,al
           mov    spacej,al
           jmp    billki
      bill5: cmp    al,0cbh  ;left felengedés
           jne    bill6
           xor    al,al
           mov    leftj,al
           jmp    billki
      bill6: cmp    al,0cdh  ;right felengedés
           jne    billki
           xor    al,al
           mov    rightj,al
      billki: ret
           Endp
;-------------------------------------
```

```
ind:
           cli
           call   Install

           call   Terepki
           call   poki
           call   figuki
           call   ny1ki
           call   ny2ki
           call   ny3ki
           call   kincski
      i0:  call   waits
           call   billbeolv
           call   ny1ki
           call   ny2ki
           call   ny3ki
           mov    keslel,al
           call   figuki
           mov    al,spacej
           cmp    al,0          ; space
           je     i01
           jmp    ugras
      i01:  mov    al,leftj
           cmp    al,0          ; balra
           je     i02
           jmp    left
      i02:  mov    al,rightj
           cmp    al,0          ; jobbra
           je     nyu
           jmp    right

      nyu:  call   talajvizsg
           call   ugrasp
           call   ny1szam
           call   ny2szam
           call   ny3szam
           call   figuki
           call   ny1ki
           call   ny2ki
           call   ny3ki
           call   utkvizsg
           call   kincsvizsg
           call   mpiki
           mov    dx,3dah
      nyu1:in     al,dx
           and    al,8  ; megvárja a képkioltást
           cmp    al,0
           jz     nyu1
           mov    cx,8000
      var: loop   var           ; var egy kicsit
           jmp    i0

      Left:  mov    ax,figx
           cmp    ax,0
           jne    Left0
           jmp    ujterepl
      Left0: mov    di,figcim
           dec    di
           dec    di
           mov    al,es:[di]
           cmp    al,0
           jnz    Leftki
           call   beep2
           add    di,5440
           mov    al,es:[di]
           cmp    al,0
           jnz    Leftki
           mov    ax,figx
           dec    ax
           dec    ax
           mov    figx,ax
           mov    al,1
           mov    figir,al
           mov    ax,figY
           mov    cx,320
           mul    cx
           add    ax,figX
           mov    figcim,ax
      Leftki: jmp    i02
```

```
Right: mov    ax,figx
       cmp    ax,297
       jle    Right0
       jmp    ujterepr
Right0: mov   di,figcim
       add    di,21
       mov    al,es:[di]
       cmp    al,0
       jnz    Rightki
       call   beep2
       add    di,5440
       mov    al,es:[di]
       cmp    al,0
       jnz    Rightki
       mov    ax,figx
       inc    ax
       inc    ax
       mov    figx,ax
       xor    al,al
       mov    figir,al
       mov    ax,figY
       mov    cx,320
       mul    cx
       add    ax,figX
       mov    figcim,ax
Rightki:jmp    nyu

Ugras: mov    al,figugr
       cmp    al,0
       jne    ugrki
       mov    di,figcim
       add    di,5760
       mov    al,es:[di]
       add    di,19
       or     al,es:[di]
       cmp    al,0
       je     ugrki
       call   beep2
       mov    al,1
       mov    figugr,al
ugrki: jmp    i01

Ujterepr:mov   al,terepsz
       inc    al
       mov    terepsz,al
       mov    ax,4
       mov    figx,ax
       call   cls
       call   terepki
       call   figuki
       call   ny1ki
       call   ny2ki
       call   ny3ki
       call   kincski
       jmp    i0

Ujterepl:mov   al,terepsz
       dec    al
       mov    terepsz,al
       mov    ax,296
       mov    figx,ax
       call   cls
       call   terepki
       call   figuki
       call   ny1ki
       call   ny2ki
       call   ny3ki
       call   kincski
       jmp    i0


kilep: sti
       call   UnInstall
```

Képernyővédő

A Windows programozók többsége kicsit idegenkedve tekint azokra a file-okra, amiknek nem .EXE vagy DLL a kiterjesztése. Windows programozói sorozatunknak ebben a részében ezen szeretnénk egy kicsit változtatni úgy, hogy bemutatjuk egy screen saver, azaz képernyőkímélő fejlesztését.

Bevezetés

A Microsoft Windows alatt minden screen saver (az .SCR kiterjesztés ellenére) egy futtatható EXE file, aminek írásakor néhány szabályt be kell tartanunk. Mint látni fogjuk, ezek a „szabályok" tulajdonképpen könnyítések, a rendszer sok feladatot levesz a vállunkról. Egy screen saver-nek tulajdonképpen csak egy dolgot kell tudnia: kirajzolnia valamit a képernyőre. Ha ehhez valamilyen, a felhasználó által állítható paraméterre van szükség (pl. egy pattogó labda sebessége), akkor csinálnunk kell egy dialógusablakot is, amiben ezeket a paramétereket paramétereket be lehet állítani.

Ebben a cikkben egy kockát 3 dimenzióban forgató screen saver-en keresztül mutatjuk be a fentieket. A program jelentős része a kocka forgatásával, mozgatásával és kirajzolásával foglalkozik. Ennek bemutatása nem célja a cikknek, a megjegyzésekkel ellátott teljes C forráskód a lemezmellékleten megtalálható.

ScreenSaverProc

Nézzük először az alapproblémát: hogyan működik egy screen saver? Mint minden Windows program, ez is üzeneteket kap a rendszertől. A fenti az üzenetek *ScreenSaverProc* nevű ablakfüggvénynek mennek, ezt a függvényt kell tehát megírnunk. Az üzenetek egy részét mi kezeljük le, a többit továbbadjuk a rendszernek. Ez utóbbit a *DefScreenSaverProc* hívásával tehetjük meg, ami a közönséges programoknál használatos *DefWindowProc*-tól eltérően kezel le néhány üzenetet (és ezzel nagyban megkönnyíti a dolgunkat).

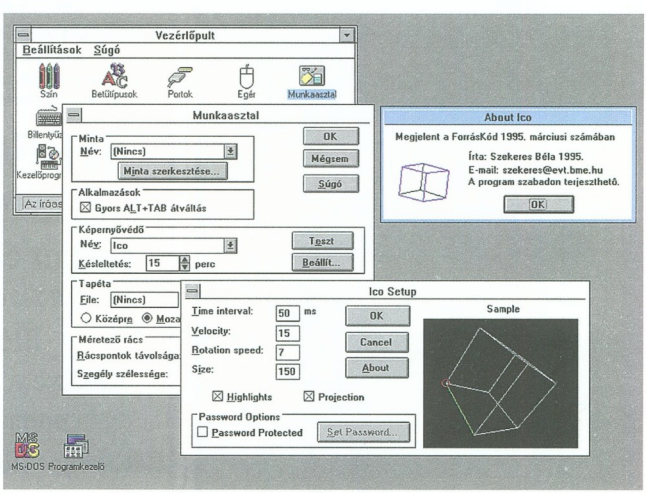
Amikor a beállított várakozási idő letelt, a rendszer betölti a programunkat és a szokásos WM_CREATE üzenet elküldésével inicializálja, majd törli a képernyőt és vár. Ha a felhasználó megmozdítja az egeret vagy lenyom egy billentyűt, akkor kapunk egy WM_DESTROY üzenetet, ami a program futásának a végét jelenti.

Ez arra már elég lenne, hogy a képernyőt a beégéstől megvédjük: a rendszer letörli a képernyőt és kész. Hogyan tudunk viszont mozgó ábrát a képernyőre rajzolni? Úgy, hogy a WM_CREATE üzenet megkapásakor elindítunk egy timer-t (*SetTimer*), ami megadott időnként küld egy WM_TIMER üzenetet, aminek hatására egy kicsit változtatunk a kirajzolt ábrán. Természetesen a WM_DESTROY üzenet hatására a timer-t leállítjuk (*KillTimer*) és a program futása véget ér.

Az ábra mozgatása nálunk a *MoveImage* nevű függvényben történik. Ennek tartalma most nem lényeges, mi itt a kockát forgatjuk és kirajzoljuk.

A SetTimer függvény paramétereiből látható, hogy a kirajzolt ábrán *wElapse* időnként változtatunk. Ez a paraméter (több más paraméter mellett) a felhasználó által állítható. A beállított értékeket a többi screen saver-hez hasonlóan mi is a CONTROL.INI-ben tároljuk, e paramétereket olvastuk be a *GetIniSettings*-ben. Itt állítunk be néhány olyan globális változót is, amit minden screen saver-ben definiálnunk kell. Értéküket a resource-ok közül olvassuk be a LoadString utasítással. A változók deklarációja ill. a resource azonosítók (pl. idsIsPassword) az SCRNSAVE.H file-ban találhatók. Ezektől az azonosítóktól eltérni nem tanácsos.

Bár a Windows SDK dokumentáció azt mondja, hogy ezek a változók az SCRNSAVE.LIB-ben definiálva vannak, ez nem így van. Nekünk kell őket definiálni, ellenkező esetben linkeléskor hibaüzenetet kapunk.



Bár a Windows SDK dokumentáció azt mondja, hogy ezek a változók az SCRNSAVE.LIB-ben definiálva vannak, ez nem így van. Nekünk kell őket definiálni, ellenkező esetben linkeléskor hibaüzenetet kapunk.

ScreenSaverConfigureDialog

A wElapse, stb. paramétereket betölteni már tudjuk, kérdés hol lehet őket beállítani. Erre szolgál a korábban már említett dialógusablak. A dialógusablakot nekünk kell megterveznünk, a resource azonosítónak DLG_SCRNSAVECONFIGURE-nek kell lennie, az idetartozó ablakfüggvénynek pedig a *ScreenSaverConfigureDialog* nevet kell adnunk. Mind a két azonosító a SCRNSAVE.H-ban van deklarálva. Ha van ScreenSaverConfigureDialog dialógusablakunk, akkor mindig definiálnunk kell egy *RegisterDialogClasses* nevű függvényt. Ez elvileg arra szolgál, hogy a dialógusablakban használt ablakosztályokat (window classes) regisztráljuk. Mi ilyet nem használunk, így mindig TRUE-val (nincs hiba) jövünk vissza.

A WM_INITDIALOG hatására beolvassuk a paramétereket a CONTROL.INI-ből. Erre szükség van, mert a screen saver konfigurálása teljesen független magától a screen saver-től, így eddig senki se olvasta be ezeket a paramétereket (a ScreenSaverProc nem kapott WM_CREATE üzenetet).

A paraméterek beolvasása után megnézzük, engedélyezve van e a jelszóvédelem, és ettől függően engedélyezzük ill. tiltjuk le a Set Password gombot. Ha később a felhasználó ki- ill. bekapcsolja a jelszóvédelmet, akkor ezt újra megtesszük (ID_PASSWORDPROTECTED).

A jelszóvédelmet természetesen ki is lehetne hagyni, ha viszont már beletettük, dolgoznunk kell még egy kicsit. Meg kell terveznünk 3 dialógusablakot: DLG_ENTERPASSWORD a jelszó beolvasására, DLG_CHANGEPASSWORD a jelszó megváltoztatására és DLG_INVALIDPASSWORD a felhasználó figyelmeztetésére, ha rossz jelszót írt be. Ezeknél a dialógusablakoknál is a SCRNSAVE.H-ban megadott azonosítókat kell használni. Legegyszerűbb egy az egyben átvenni az e programban használt 3 dialógusablakot, ezek azonosak a Windows-zal adott screen saver-ekével. Szerencsére az ablakfüggvényekkel nem kell törődnünk: ezeket már megírták, a SCRNSAVE.LIB-ben vannak, ezt pedig a programhoz hozzá fogjuk linkelni. A DLG_CHANGEPASSWORD-höz tartozót DlgChangePassword-nek hívják, a Set Password gomb lenyomására őt hívjuk meg. Az ablakfüggvény dolga a jelszó beolvasása, kódolása, nekünk ezzel semmi dolgunk nincs.

A modul definíciós file és a linkelés

A DEF file-ban egy átlagos programhoz képest nem sok változás van: a DESCRIPTION blokknak a „SCRNSAVE :" karaktersorral kell kezdődnie, továbbá néhány függvényt exportálnunk kell. Nem véletlenül maradt ki a RegisterDialogClasses függvényt, ezt nem szabad exportálni.

```
...
DESCRIPTION 'SCRNSAVE : Ico'
...
EXPORTS
        SCREENSAVERPROC             @1
        SCREENSAVERCONFIGUREDIALOG      @2
        DLGCHANGEPASSWORD           @3
        DLGGETPASSWORD              @4
        DLGINVALIDPASSWORD          @5
```

A linkelésnél egy dologra kell figyelni: be kell linkelni a SCRNSAVE.LIB-et.

Debuggolás, hibakeresés

Mindig gondot okoz egy programban a hibák keresése. Különösen igaz ez a screen saver-ekre, hisz a .SCR file-ok magukban nem futnak, ráadásul bármilyen gomb lenyomására a program futása véget ér.

A látszat persze itt is csal: az .SCR file tulajdonképpen egy futtatható .EXE file. Nevezzük át bármelyik screen saver-t .EXE-ve és futtassuk le. A konfigurációs dialógus ablakot láthatjuk. Próbáljuk meg a /s paraméterrel indítani a programot. Ennek hatására maga a screen saver lép működésbe. Ezek figyelembevételével már jobbak a kilátásaink.

A debug info-t minden további nélkül belefordíthatjuk a futtatható file-ba. A konfigurációs dialógus ablak debuggolása így már nem okozhat nagy gondot, itt nem gond, ha lenyomunk egy-egy gombot. Magának a screen saver résznek a debuggolása az integrált fejlesztői környezetből (IDE) nem lehetséges (legalábbis a Visual C++-ból nem). A CodeView azonban különösebb gond nélkül megbirkózik vele. CodeView-val szükség esetén az .SCR file is debuggolható: debuggoljuk a \WINDOWS\CONTROL.EXE-t, ezután a Run/Load paranccsal betölthetjük az .SCR file-t. Most már használhatjuk a File/Open Module utasítást és rakhatunk a forráskódba töréspontokat.

Ha valaki nem szereti a CodeView-t, akkor megírhatja a lényegi részt (ez általában a MoveImage-nek megfelelő rész és a timer elindítása/leállítása) egy közönséges futtatható program formájában, amit az IDE-ben is jól lehet debuggolni. Ha a program ablakának méretét nem változtatjuk menet közben, akkor a működés nagyon hasonló lesz a screen saver-hez. Az ablak nagyítása/kicsinyítése azért okozhat gondot, mert a screen saver mindig a teljes képernyőn dolgozik, nincs is értelme felkészíteni arra, hogy az ablak mérete menet közben megváltozik.

Végül ha csak arra vagyunk kíváncsiak, hogy egy adott részre rákerül-e a vezérlés (pl. működik e a timer, azaz jön-e WM_TIMER üzenet), használjuk a MessageBeep(-1) függvényt! Ez sípol egyet és a program fut tovább.

És ami nem csak egy screen saver-nél lehet hasznos

Minden Windows programnak saját magának kell tudni újrarajzolni a saját ablakát. Ha tehát egy másik ablakot kiraknak a mi programunk ablaka elé, amit aztán becsuknak, akkor a kirajzolt ábra egy része „elveszett", azt a rendszer nem rajzolja újra. Ilyenkor a rendszer az aktuális ablakfüggvénynek küld egy WM_PAINT üzenetet, aminek hatására a program újrarajzolja magát. Ez alól csak a dialógusablakok kivételek, ha ezeket újra kell rajzolni, azt a rendszer teszi meg (egy dialógusablakban általában néhány, a rendszer által ismert dolog van, pl. nyomógombok, szövegek, list box-ok). Ez viszont gondokat okoz esetünkben, hisz mi egy ábrát rajzolunk ki. Ha valami miatt újra kell rajzolni a dialógusablakot, azt a rendszer teljesen nem

tudja megtenni, dialógusablakokban viszont nincs WM_PAINT.
A megoldást az úgynevezett tulajdonos által megrajzolandó nyomógomb (owner draw button) adja. Ezek szabályos nyomógombok, csak a BS_OWNERDRAW flag-et be kell állítani a resource file-ban (.RC file). Ennek hatására a rendszer – valahányszor ezt a gombot újra kell rajzolni – egy WM_DRAWITEM üzenetet küld az ablakfüggvénynek. Az üzenet WPARAM paraméterben megkapjuk a gomb azonosítóját, így ha egy dialógusablakban több ilyen gomb van, akkor is el tudjuk dönteni, melyikről van szó. Az LPARAM paraméter egy információs struktúrára mutat, ami egyebek között tartalmazza a gomb méretét, az aktuális állapotát (lenyomott/felengedett, stb.). Ezek segítségével már nem gond az újrarajzolás.
Ezt persze általánosabban is fel lehet használni. Nem lehet például egy bitmap-et kirakni egy dialógusablakba, ezért egy ilyen gombot definiálunk, és ebbe rajzoljuk ki a bitmap-et. A gomb „hagyományos" funkcióját nem is használjuk, nem csinálunk semmit, ha megnyomják.
Az egyetlen gondot az okozza, hogy a gomb méretét nehéz pont akkorára választani, mint a bitmap-é, mert a gomb nagyságát nem pixel-ben, hanem egy speciális, ún. dialógusablak koordináta-rendszerben kell megadni. Ez a dialógusablakban használt betűtípustól függ. A megoldás az, hogy a WM_CREATE üzenet hatására átállítjuk a gomb méretét akkorára, mint a bitmap. Arra persze vigyázni kell, hogy, nagyjából jó méretű legyen már induláskor is a gomb, különben elég furán nézhet ki az eredmény.
A fent leírtak láthatók az ABOUT.C file-ban.
Ugyanezt a trükköt használjuk a konfigurációs dialógusablakban arra, hogy a kocka képét egy ablakban (ami persze egy nyomógomb) láthassuk. Természetesen ugyanaz a függvény (MoveImage) rajzolja ki a képet itt is, mint magánál a screen saver-nél, csak néhány paraméter más (például a sebesség 0). Egy dologra azonban figyelni kell: amikor egy program kirajzol valamit, akkor a rajznak az ablakon kívül eső részeit automatikusan levágja a rendszer (clipping). Szerencsére ugyanez a helyzet a felhasználó által rajzolandó gombok esetében is, amikor a WM_DRAWITEM üzenet jön. Most azonban egy WM_TIMER üzenet hatására rajzolunk, amikor a vágás csak a dialógusablak szélén történne meg. Nem akarunk a nekünk megadott területen kívül rajzolni, ezért beállítjuk, hol történjen a vágás.

```
void MoveImage(HWND hWnd, int nCaller)
{
...
  /* A rajzolás */
  hDC = GetDC(hWnd);
  if (nCaller==MV_SETUP){
     /* Ha a konfigurációs ablakba rajzolunk, akkor
     a vágást (clipping) be kell állítani */
     HRGN hRgn;
     hRgn=CreateRectRgn(0, 0, rcWnd.right,
     rcWnd.bottom);
     SelectClipRgn(hDC, hRgn);
     DeleteObject(hRgn);
  }
...
  ReleaseDC(hWnd, hDC);
} /* MoveImage() */
```

Befejezésül

Végül szeretnénk néhány tanácsot adni, amire egy screen saver írásánál nem árt odafigyelni.
Ne írjunk lassú screen saver-t! Egy screen saver olyankor indul el, ha a felhasználó sokáig nem csinált semmit. Ez gyakran azért van, mert egy nagy számításigényű program befejeződésére vár. Ha a programunk nagyon lelassítja a gépet, nem fogják használni. Ennek szellemében íródott a MOVE.C file is. Sok számítás eredményét – amit csak egyszer, vagy ritkán kell megcsinálni – statikus változóban tárolunk. Ez igen sokat gyorsít a programon.
Az, hogy induláskor törlődik a képernyő, könnyen megváltoztatható: a ScreenSaverProc kap egy WM_ERASEBKGND üzenetet, aminek hatására a DefScreenSaverProc törli a képernyőt. Kezeljük le mi ezt az üzenetet és térjünk vissza return TRUE-val.
Elvileg a ScreenSaverProc függvényben lekezelhetünk más üzeneteket is, pl. a WM_MOUSEMOVE-ot. Ha ez esetben nem hívjuk meg a DefScreenSaverProc-ot, az egér mozgatásakor nem áll le a screen saver, ehhez egy gombot kell lenyomnunk. Óvatosan bánjunk ezzel, egy screen saver-nek illik alkalmazkodni az általánosan elfogadotthoz. (Azért ki lehet próbálni, hogy WM_MOUSEMOVE esetén MessageBeep(-1)-et hívunk, majd return TRUE-val visszatérünk. Szép kis füttykoncertet kapunk, ha az egérhez nyúlunk.)
Reméljük, sikerült kedvet csinálni a screen saver íráshoz. Ha valakinek az itteni információ kevés lenne, az nézze meg a SCRNSAVE.H file-t.

Szekeres Béla
E-mail: szekeres@evt.bme.hu

```
/*
 * File:    ABOUT.C
 * Tartalom:  About Box az Ico Windows Screen Saver-hez
 * Nyelv:   Microsoft C 7.0 (Windows SDK)
 * Szerző:  ifj. Szekeres Béla (szekeres@evt.bme.hu)
 * Dátum:   1995. március 4.
 */
#include <windows.h>
#include "ico.h"

BOOL FAR PASCAL __export AboutDlg(HWND hDlg, UINT message, WPARAM wParam,
                  LPARAM lParam)
{
  static HBITMAP hBitmap;    /* A bitmap handle-je. */

  HDC   hMemoryDC;
  HWND  hWnd;      /* Az ID_FRAME gomb handle-je */
  HGDIOBJ hOld;
  BITMAP bm;       /* A bitmap adatai */
  POINT  p;        /* A kirajzolandó bitmap bal-felső csúcsának
                   koordinátái */
  RECT   rc;
  LPDRAWITEMSTRUCT lpDis;

  switch (message)
  {
      case WM_INITDIALOG:
          /* A bitmap betöltése és kirajzolása */
          hBitmap=LoadBitmap(hMainInstance, MAKEINTRESOURCE(IDB_ABOUT));
          if (hBitmap){
              GetObject(hBitmap, size of(BITMAP), (LPSTR) &bm);
```

```
            hWnd=GetDlgItem(hDlg, ID_FRAME);
            /* 'rc'-be az ID_FRAME gomb koordinátáinak beolvasása */
            GetWindowRect(hWnd, &rc);
            /* 'p' a kirajzolandó bitmap bal-felső koordinátája. */
            p.x=(rc.right+rc.left-bm.bmWidth)/2;
            p.y=(rc.bottom+rc.top-bm.bmHeight)/2;
            /* átkonvertálás a párbeszédablak koordináta-rendszerébe */
            ScreenToClient(hDlg, &p);
            /* az ID_FRAME gomb átméretezése */
            SetWindowPos(hWnd, NULL, p.x, p.y, bm.bmWidth, bm.bmHeight,
                  SWP_NOACTIVATE | SWP_NOZORDER);

        };
        return (TRUE);

    case WM_DRAWITEM:
        /* A bitmap kirajzolása. */
        if (hBitmap){

            lpDis=(DRAWITEMSTRUCT FAR *)lParam;
            hMemoryDC=CreateCompatibleDC(lpDis->hDC);
            hOld=SelectObject(hMemoryDC, hBitmap);
            BitBlt(lpDis->hDC, (lpDis->rcItem).left, (lpDis->rcItem).top,
                  (lpDis->rcItem).right-(lpDis->rcItem).left,
                  (lpDis->rcItem).bottom-(lpDis->rcItem).top,
                  hMemoryDC, 0, 0, SRCCOPY);
            SelectObject(hMemoryDC, hOld);
            DeleteDC(hMemoryDC);

            return (TRUE);
        };
        return (FALSE);

    case WM_COMMAND:
        switch (wParam)
        {

            case IDCANCEL:
                if (hBitmap) DeleteObject(hBitmap);
                /* ...és kilépünk */
                EndDialog(hDlg, TRUE);
                return (TRUE);
        }
        break;
  }
  return (FALSE);
}

/*
 * File:     ICO.C
 * Tartalom:  Az Ico Windows Screen Saver fő rutinjai
 * Nyelv:    Microsoft C 7.0 (Windows SDK)
 * Szerző:   ifj. Szekeres Béla (szekeres@evt.bme.hu)
 * Dátum:    1995. március 4.
 */

#include <windows.h>

#include "ico.h"

/*
 * Minden Screen Saver-ben kötelező globális változók.
 */
char szAppName[40];

HINSTANCE _cdecl hMainInstance;
```

```
HWND _cdecl hMainWindow;
char _cdecl szName[TITLEBARNAMELEN];
char _cdecl szIsPassword[22];
char _cdecl szIniFile[MAXFILELEN];
char _cdecl szScreenSaver[22];
char _cdecl szPassword[16];
char _cdecl szDifferentPW[BUFFLEN];
char _cdecl szChangePW[30];
char _cdecl szBadOldPW[BUFFLEN];
char _cdecl szHelpFile[MAXFILELEN];
char _cdecl szNoHelpMemory[BUFFLEN];
UINT _cdecl MyHelpMessage;
HOOKPROC _cdecl fpMessageFilter;

/*
 * Az Ico Screen Saver egyes paramétereit azonosító karakterkonstansok.
 */
char szSpeedName[] = "Speed";
char szVelocityName[] = "Velocity";
char szRotationName[] = "Rotation";
char szSizeName[] = "Size";
char szHighLightName[] = "HighLight";
char szProjName[] = "Projection";
char szName[]="Ico";

/*
 * Globális változók
 */

WORD wTimer;      /* A használt Timer azonosítója */
WORD wElapse;     /* Ilyen időközönként változik a kép */
WORD wVelocity;   /* Ilyen sebességgel mozog a kocka */
WORD wRotation;   /* Ilyen sebességgel forog a kocka */
WORD wSize;       /* A kocka mérete */
BOOL bPassword;   /* Jelszóvédett? */
BOOL bHighLight;  /* Kiemelünk egy élet és egy csúcsot? */
BOOL bProj;               /* A vetítés projektív? */

LONG FAR PASCAL __export ScreenSaverProc(HWND hWnd, UINT msg, WPARAM wParam,
                        LPARAM lParam)
{
  switch (msg)
  {
        case WM_CREATE:
            /* Az egyes paraméterek beolvasása a CONTROL.INI-ből */
            GetIniSettings();
            /* A Timer elindítása (wElapse időközönként) */
            wTimer = SetTimer(hWnd, ID_TIMER, wElapse, NULL);
            break;

        case WM_TIMER:
            /* Minden WM_TIMER-re mozditunk egy kicsit a  képen. A képet
            a hWnd által leírt területre kell rajzolni,
            és most a Screen Saver működik. */
            MoveImage(hWnd, MV_SCR);
            break;

        case WM_DESTROY:
            /* Kilépés előtt a Timer lelövése */
            if (wTimer) KillTimer(hWnd, ID_TIMER);
            break;
        }

   /* Az üzenetek többségét továbbküldjük a rendszernek */
   return DefScreenSaverProc(hWnd, msg, wParam, lParam);
} /* ScreenSaverProc() */
```

```
BOOL RegisterDialogClasses(HINSTANCE hInst)
{
  return TRUE;
}

BOOL FAR PASCAL __export ScreenSaverConfigureDialog(HWND hDlg, UINT msg,
                         WPARAM wParam, LPARAM lParam)

{
 static HWND hIDOK;
 static HWND hSetPassword;
 static HWND hSample;
 static HRGN hRgn;

  switch (msg)
  {
      case WM_INITDIALOG:
          /* Az egyes paraméterek beolvasása a CONTROL.INI-ből */
          GetIniSettings();
          /* .. és a párbeszédablak beállítása ezeknek megfelelően */
          SetDlgItemInt(hDlg, ID_SPEED,   wElapse, FALSE);
          SetDlgItemInt(hDlg, ID_VELOCITY, wVelocity, FALSE);
          SetDlgItemInt(hDlg, ID_ROTATION, wRotation, FALSE);
          SetDlgItemInt(hDlg, ID_SIZE,    wSize,    FALSE);
          SendDlgItemMessage(hDlg, IDC_HIGHLIGHT, BM_SETCHECK, bHighLight,
                   NULL);
          SendDlgItemMessage(hDlg, IDC_PROJ, BM_SETCHECK, bProj, NULL);
          SendDlgItemMessage(hDlg, ID_PASSWORDPROTECTED, BM_SETCHECK,
                   bPassword, NULL);
          /* A 'Set Password' letiltása/engedélyezése */
          hSetPassword=GetDlgItem(hDlg, ID_SETPASSWORD);
          EnableWindow(hSetPassword, bPassword);
          /* Az 'OK' és a 'Sample' gombok handle-je */
          hIDOK=GetDlgItem(hDlg, IDOK);

          hSample=GetDlgItem(hDlg, IDC_SAMPLE);
          /* A 'Sample'-ben látható kocka paraméterei:
            - 80 pixel és nem mozog (csak forog)
            - forgási sebessége és a képfrissítési idő a
              beállított paramétereknek megfelelő */
          wSize=80;
          wVelocity=0;
          /* A Timer elindítása */
          wTimer=SetTimer(hDlg, ID_TIMER, wElapse, NULL);
          return TRUE;

      case WM_TIMER:
          /* Minden WM_TIMER-re mozdítunk egy kicsit a képen.
             A képet a hSample által leírt területre kell rajzolni,
             és most a kofigurációs párbeszédablak működik. */
          MoveImage(hSample, MV_SETUP);
          return (FALSE);


case WM_DESTROY:
          /* A Timer lelövése. */
          if (wTimer) KillTimer(hDlg, ID_TIMER);
          return FALSE;

      case WM_DRAWITEM:
          /* Felhasználó által rajzolandó gomb. Ez csak akkor hívódik
             meg, ha újra kell rajzolni a gombot. Mivel mi a Timer-nek
             megfelelően mindig újrarajzoljuk, így most csak a hátteret
             töröljük le. */
          if (wParam==IDC_SAMPLE){
              FillRect(((LPDRAWITEMSTRUCT)lParam)->hDC,
                       &((LPDRAWITEMSTRUCT)lParam)->rcItem,
                        (HBRUSH)GetStockObject(BLACK_BRUSH));
              return TRUE;
          }
          break;

      case WM_COMMAND:
          switch (wParam)
          {

              case IDOK:
                  /* 'OK' -> megnézzük az új értékeket, amiket a felhasználó
                     állított be és kiírjuk őket a  CONTROL.INI-be */
                  wElapse  = GetDlgItemInt(hDlg, ID_SPEED,   NULL, FALSE);
                  wVelocity = GetDlgItemInt(hDlg, ID_VELOCITY, NULL, FALSE);
                  wRotation = GetDlgItemInt(hDlg, ID_ROTATION, NULL, FALSE);
                  wSize    = GetDlgItemInt(hDlg, ID_SIZE, NULL, FALSE);
                  bPassword = IsDlgButtonChecked(hDlg, ID_PASSWORDPROTECTED);
                  bHighLight= IsDlgButtonChecked(hDlg, IDC_HIGHLIGHT);
                  bProj    = IsDlgButtonChecked(hDlg, IDC_PROJ);
                  WriteProfileInt(szAppName, szSpeedName, wElapse);
                  WriteProfileInt(szAppName, szVelocityName, wVelocity);
                  WriteProfileInt(szAppName, szRotationName, wRotation);
                  WriteProfileInt(szAppName, szSizeName, wSize);
                  WriteProfileInt(szAppName, szIsPassword, bPassword);
                  WriteProfileInt(szAppName, szHighLightName, bHighLight);
                  WriteProfileInt(szAppName, szProjName, bProj);
                  EndDialog(hDlg, TRUE);

                  return TRUE;

              case IDCANCEL:
                  /* 'Cancel' gomb, bezárjuk az ablakot */
                  EndDialog(hDlg, FALSE);
                  return TRUE;

              case ID_ABOUT:
                  /* 'About' -> kirakjuk az DLG_ABOUTBOX párbeszédablakot */
              {
                  FARPROC fpDialog;

                  if((fpDialog = MakeProcInstance(AboutDlg,hMainInstance))
                           == NULL) return FALSE;
                  DialogBox(hMainInstance, MAKEINTRESOURCE(DLG_ABOUTBOX),
                            hDlg, fpDialog);
                  FreeProcInstance(fpDialog);
                  /* A fókuszt visszatérés után automatikusan a következő
                  gombra állítjuk. */
                  SendMessage(hDlg, WM_NEXTDLGCTL, hIDOK, 1l);
                  break;
              }

              case ID_SPEED:
                  if (HIWORD(lParam)==EN_KILLFOCUS){
                      /* A képfrissítési idő megváltozhatott. A Timer-t
                         átprogramozzuk, hogy tükrözze az új paramétert.*/
                      wElapse= GetDlgItemInt(hDlg, ID_SPEED, NULL, FALSE);
                      if (wTimer) KillTimer(hDlg, ID_TIMER);
                      wTimer=SetTimer(hDlg, ID_TIMER, wElapse, NULL);
                  }
                  break;

              /* A következő pár sor megváltoztatja a 'Sample'
                 megfelelő paraméterét, ha a megfelelő értéket a
                 felhasználó a párbeszédablakban módosította.
                 Így a kép mindig a valós helyzetet mutatja. */
```

```
            case ID_ROTATION:
                wRotation= GetDlgItemInt(hDlg, ID_ROTATION, NULL, FALSE);
                break;

            case IDC_HIGHLIGHT:
                bHighLight= IsDlgButtonChecked(hDlg, IDC_HIGHLIGHT);
                break;
            case IDC_PROJ:
                bProj= IsDlgButtonChecked(hDlg, IDC_PROJ);
                break;

            case ID_SETPASSWORD:
                /* Jelszó beállítása. Meghívjuk a rendszer által
                   definiált DlgChangePassword ablakfüggvényt. */
            {
                FARPROC fpDialog;

                if((fpDialog = MakeProcInstance(DlgChangePassword,
                            hMainInstance)) == NULL)
                    return FALSE;
                DialogBox(hMainInstance,
                    MAKEINTRESOURCE(DLG_CHANGEPASSWORD), hDlg, fpDialog);
                FreeProcInstance(fpDialog);
                SendMessage(hDlg, WM_NEXTDLGCTL, hIDOK, 1L);
                break;
            }

            case ID_PASSWORDPROTECTED:

                /* Engedélyezzük/letiltjuk a 'Set Password' gombot attól
                   függően, hogy a jelszó-ellenőrzés engedélyezve van-e. */
                bPassword=IsDlgButtonChecked(hDlg, ID_PASSWORDPROTECTED);
                EnableWindow(hSetPassword, bPassword);
                break;

        }
        break;
    }
    return FALSE;
} /* ScreenSaverConfigureDialog() */
```

```
static void GetIniSettings( void )
{
    /* A karakterláncok betöltése az EXE-ből */
    LoadString(hMainInstance, idsIsPassword, szIsPassword, 22);
    LoadString(hMainInstance, idsIniFile, szIniFile, MAXFILELEN);
    LoadString(hMainInstance, idsScreenSaver, szScreenSaver, 22);
    LoadString(hMainInstance, idsPassword, szPassword, 16);
    LoadString(hMainInstance, idsDifferentPW, szDifferentPW, BUFFLEN);
    LoadString(hMainInstance, idsChangePW, szChangePW, 30);
    LoadString(hMainInstance, idsBadOldPW, szBadOldPW, 255);
    LoadString(hMainInstance, idsHelpFile, szHelpFile, MAXFILELEN);
    LoadString(hMainInstance, idsNoHelpMemory, szNoHelpMemory, BUFFLEN);


    /* A paraméterek betöltése az INI file-ból */
    wElapse =  GetPrivateProfileInt(szAppName, szSpeedName,
                                DEF_SPEED, szIniFile);
    wVelocity = GetPrivateProfileInt(szAppName, szVelocityName,
                                DEF_VELOCITY, szIniFile);
    wRotation = GetPrivateProfileInt(szAppName ,szRotationName,
                                DEF_ROTATION, szIniFile);
    wSize =    GetPrivateProfileInt(szAppName, szSizeName,
                                DEF_SIZE, szIniFile);
    bPassword = (BOOL)GetPrivateProfileInt(szAppName, szIsPassword,
                                FALSE, szIniFile);
        bHighLight= (BOOL)GetPrivateProfileInt(szAppName, szHighLightName,
                                TRUE, szIniFile);
    bProj    = (BOOL)GetPrivateProfileInt(szAppName, szProjName,
                                TRUE, szIniFile);
} /* GetIniSettings() */


static void WriteProfileInt(LPSTR szSection, LPSTR szKey, UINT u)
{
    char achBuf[40];


    wsprintf(achBuf, "%u", u);
    WritePrivateProfileString(szSection, szKey, achBuf, szIniFile);
}
```

Linux 2..

Olvasóink ettől a számunktól kezdve rendszeresen találnak majd lemezmellékletünkön kisebb-nagyobb Linux programcsomagot. Ezek telepítését igyekeztünk egyszerűvé tenni, az e célra írt program segítségével.

Ahhoz, hogy telepitő kis Linux-os program a „helyére kerüljön", még dolgoznunk kell egy kicsit.
Mivel Linux-ból a floppy-t még nem tudjuk elérni , ezért először a programot fel kell másolnunk a merevlemezre. Ezzel együtt másoljuk fel a lemezen található többi Linux-os telepítőkészletet is.
Ezt DOS alól tehetjük meg a FELRAK.BAT parancs segítségével:

```
a:
\linux\felrak.bat
```

Ez létrehoz a c: drive-on egy \linuxins alkönyvtárat, amibe a szükséges file-okat felmásolja. Ezután indítsuk el a Linux-ot, lépjünk be *root*-ként és adjuk ki a következő parancsokat:

```
cp /DOS/linuxins/felrak /etc/felrak
chmod 744 /etc/felrak
felrak
```

Az első parancs felmásolja a Linux-os telepítő programot a „helyére", amit a második parancs futtathatóvá tesz. Nincs más dolgunk, mint elindítani a felrak parancsot. A parancs egyenként kiírja az installálandó programok nevét egy rövid leírás kíséretében. Ebben a számban 3 programcsomagot találnak: egy teljes 32 bites GNU C fordítót; egy rövid mintaprogramot C-ben és a shell script-ben és egy segédprogram csomagot, ami lehetővé teszi, hogy DOS-os floppy-val végezhessünk különböző műveleteket. Érdemes őket felrakni, a későbbiekben még nagy hasznukat fogjuk venni. Ha a telepítés sikeresen befejeződött, a c:\linuxins\felrak nevű file-ra már nem lesz szükségünk, aki akarja, nyugodtan letörölheti. A segédeszközök már a helyükön vannak, jöhet az igazi programozás. Ehhez célszerű *en* -ként belépni. Sok sikert az új világhoz!!

A GNU C fordító használata UNIX alatt

Először az UNIX-os C fordító használatáról lesz szó. Feltételezzük, hogy az Olvasó más operációs rendszer alatt már írt C programot. Ha a programcsomagok telepítését a korábban leírtaknak megfelelően végeztük, a saját alkönyvtárunkban (/home/en) találunk néhány C programot (vagy inkább csak programocskát). Ez jó lesz arra, hogy a fordító használatát kipróbálhassuk.

Vágjunk a közepébe és fordítsuk le a hello.c-t!

```
gcc hello.c
```

Az ls -l paranccsal megnézhetjük, mit kaptunk:

```
-rwxr-xr-x  1 en  users  15393 Mar 10 08:38 a.out*
-rw-r—r—    1 en  users    736 Mar  6 22:31 flcp.bash
-rw-r—r—    1 en  users   1633 Mar  6 22:31 flcp.c
-rw-r—r—    1 en  users    108 Mar 10 08:29 hello.c
-rw-r—r—    1 en  users     75 Mar 10 08:34 hivo.c
-rw-r—r—    1 en  users     77 Mar 10 08:36 hivott.c
```

A fordítás után az aktuális alkönyvtárban megjelent egy a.out nevű futtatható file. A fordító a futtatható file-nak alapértelmezés szerint ezt a nevet adja. Látható, hogy a gcc meghívta a linkert is. Próbáljuk ki:

```
./a.out
```

A ./ -re azért van szükség, mert az aktuális alkönyvtár nincs benne a PATH-ban. Győződjünk meg róla a *set* paranccsal! A UNIX tehát - ellentétben a DOS-szal - az aktuális alkönyvtárat magától nem nézi meg. Természetesen rá lehet venni a fordítót arra, hogy csak fordítson és ne hívja meg a linkert. Erre szolgál a -c kapcsoló:

```
gcc -c hello.c
```

Az alkönyvtárban megjelent a hello.o file. A UNIX alatt a .o kiterjesztés jelenti az object file-t. Próbáljuk ki tudományunkat egy másik példán. Két forrásfile-unk van: hivo.c és hivott.c.

hivo.c:

```
int main(int argc, char **argv)
{
  kiir("C programozas\n");
  return 0;
}
#include <stdio.h>
```

hivott.c:

```
int kiir(char *szoveg)
{
  printf(szoveg);
  return 0;
}
```

Készítsünk belőle egy próba nevű futtatható file-t. (Előbb azért egy kis rendrakás nem árt, letöröljük a felesleget.)

```
rm a.out *.o
gcc -c hivo.c hivott.c
```

Használjuk újra az ls -l parancsot!

```
-rw-r—r—  1 en  users  736 Mar  6 22:31 flcp.bash
-rw-r—r—  1 en  users 1633 Mar  6 22:31 flcp.c
-rw-r—r—  1 en  users  108 Mar 10 08:29 hello.c
-rw-r—r—  1 en  users   75 Mar 10 08:34 hivo.c
-rw-r—r—  1 en  users  225 Mar 10 08:45 hivo.o
-rw-r—r—  1 en  users   77 Mar 10 08:36 hivott.c
-rw-r—r—  1 en  users  175 Mar 10 08:45 hivott.o
```

A két file-t sikeresen lefordítottuk, jöhet a linkelés. Meghívhatnánk közvetlenül is a linkert, de ez így egyszerűbb:

```
gcc -o proba *.o
```

A várt eredményt kapjuk. A -o kapcsoló átnevezi a gcc által generált file-t. Persze ezt mi is könnyen megtehettük volna:

```
mv a.out proba
```

A gcc most is meg tudta volna csinálni egy lépésben a fordítást és a linkelést. Próbáljuk meg:

```
rm proba *.o
gcc -o proba hivo.c hivott.c
```

A következő változat is ugyanezt csinálta volna:

```
gcc -c hivo.c
gcc -o proba hivo.o hivott.c
```

Ez nem meglepő, ha tudjuk hogy dolgozik a gcc. Sorraveszi a neki átadott file-neveket. Ha talál .c kiterjesztésűeket, akkor azt lefordítja object formátumra (.o kiterjesztésű file). Ezután veszi az így kapott file-okat és a többi, neki parancssorban megadott .o file-t meghívja a linkert. Ha .a kiterjesztésű file-t talál, azt library-nek tekinti és szintén átadja a linkernek. (Azt, hogy történetesen a fordítás is 3 lépésben történik, most ne firtassuk. Ha valaki .i vagy .s kiterjesztésű file-okkal találkozik, akkor ezek a közbülső melléktermékek.)

A -c paraméter tulajdonképpen csak annyit jelent: a fordítás után állj le. Ha a -c paraméter mellett .c, .o és .a file-okat is megadunk, akkor a .o és .a file-okkal semmit sem csinál. Rájuk csak a linkeléskor lenne szükség, de arra már nem kerül sor.

Ha bármilyen átmeneti file-t kell létrehozni, akkor azokat a végén letörli. A -o paraméter annyit jelent: amikor a végére érsz, a megadott nevet add a file-nak. Ez persze a fordításnál is használható:

```
gcc -o out.o -c hello.c
```

Ha a fordítási fázist csinálja, akkor a generált file neve az alapértelmezés szerint azonos a forrásfile nevével, de .c helyett .o-ra végződik (pl. hello.o hello.c-ből). A linkernél már láttuk: ha nem adunk meg semmit, a.out lesz az eredmény.

Nézzünk még egy példát!

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

int main(int argc, char **argv)
{
  printf("%g\n", sqrt(2));
  return 0;
}
```

A program kiszámolja 2 négyzetgyökét. Most már mindenki tudja:

```
gcc -o gyok gyok.c
```

Kivéve a gcc-t:

```
/tmp/cca004701.o: Undefined symbol _sin referenced from text segment
```

A szabványos UNIX-os library-ben nincsenek benne a matematikai függvények, ezeket külön kell behivatkozni:

```
gcc -o gyok gyok.c -lm
```

A -lm paramétert továbbadja a linkernek. Jelentése: a libm.a library-t fordítsd be a programba. (A -l*library* általában a lib*library*.a library-t jelenti.) Nem árt odafigyelni hova tesszük a -lm -et! Csak a file-nevek után hatásos.

Nézzünk még egy-két kapcsolót, ami hasznunkra lehet. A hivo.c-ben hivatkoztunk a kiir nevű függvényre, de azt nem deklaráltuk. Az eredeti C-nek megfelelően ez azt jelenti: a függvény int-et ad vissza, a paraméterek pedig nem szoktak gondot okozni. Most szerencsénk volt, tényleg int-et adott vissza. Ne bízzuk a szerencsére. A -Wall (minden figyelmeztetés) kapcsoló segít:

```
gcc -c -Wall hivo.c
```

Így már figyelmeztet a fordító:

```
hivo.c: In function `main':
hivo.c:3: warning: implicit declaration of function `kiir'
```

Nem árt megjegyezni a -ansi kapcsolót. Mint az sejthető, a fordító közel ANSI C fordítóként viselkedik, letiltva olyan bővítéseket, mint az inline függvények. Végül egy apróság: a kapcsolók a legtöbb UNIX-os programtól eltérően nem vonhatók össze. Például a -dr jelentése igencsak más, mint a -d -r -é.

Ezek után nézzünk valami komolyabbat egy teljes lemezt egy menetben másoló programot. DOS alatt ezt meglehetősen nehéz lenne megírni. Egyrészt a lemezkezelés okoz némi problémát, másrészt 1.44MB-nyi memóriát se tudunk egyben foglalni (XMS vagy EMS kezelés kell hozzá). Mint mindjárt meglátjuk, UNIX alatt ezek egyike sem igazán probléma.

A programot igyekeztem úgy megírni, hogy minél több UNIX specifikus dolog kiderüljön belőle.

```
 1 #include <unistd.h>   /* szabvanyos
                          UNIX/POSIX rutinok */
 2 #include <stdlib.h>   /* ansi C lib              */
 3 #include <stdio.h>    /* ez a *printf miatt kell  */
 4 #include <fcntl.h>    /* ez meg az open miatt */
 5 #include <errno.h>     /* EROFS, stb             */
 6 #include <string.h>
 7
 8 char *FLOPPY="/dev/fd0";
 9 long FLOPPY_SIZE=1440L*1024;
10
11 int main(int argc,char *argv[])
12 {
13 int fl;   /* a magneslemez UNIX fajlsorszama*/
14 int n;   /* irt/olvasott bajtok szama          */
15 char *puff;   /* puffer cime                    */
16 char *letter; /* a hasznalt drive neve betuvel  */
17
18 argc—; argv++;
19 for (;argc>=0; argc—, argv++){

20  if (strcmp(*argv, "a:")==0){ FLOPPY="/dev/fd0";
    continue; }
21  if (strcmp(*argv, "b:")==0){FLOPPY="/dev/fd1";
    continue; }
22  n=atoi(*argv);
23  if (n==360 || n==720 || n==1200 || n==1440 ||
    n==2880)
24  FLOPPY_SIZE=n*1024L;
25  }
26
27  if (strcmp(FLOPPY, "/dev/fd0")==0) letter="az A:";
28 else                               letter="a B:";
29 fprintf(stderr,"Kerem a forraslemezt %s
    meghajtoba\n", letter);
```

```
30  fprintf(stderr, "Nyomjon meg egy gombot\n");
31  n=getchar();
32  while(n!='\n') n=getchar(); /* ez szepen
    megeszi a valasz maradekat */
33
34  fl=open(FLOPPY,O_RDONLY);
35  if(fl<0)
36  {           /* nem sikerult                    */
37  perror("File-megnyitas olvasasra");
38  return(1);
39  }
40  puff=malloc(FLOPPY_SIZE);
41  if(puff==0)
42  {           /*  Nincs eleg memoriank, ami eleg
    meglepo */
43  fprintf(stderr,"Elfogyott a memoria, tobb swap
    kellene\n");
44  return(1);
45  }
46  if(read(fl,puff,FLOPPY_SIZE)!=FLOPPY_SIZE)
47  {           /* baj van, hibakod meg nincs de nem
    is erdekel */
48  fprintf(stderr,"Olvasasi hiba");
49  return(1);
50  }
51  close(fl);
52  fprintf(stderr,"Kerem a cellemezt %s
    meghajtoba\n", letter);
53  fprintf(stderr, "Nyomjon meg egy gombot\n");
54  n=getchar();
55  do
56  {
57  while(n!='\n') n=getchar(); /* ez szepen
    megeszi a valasz maradekat */
58  if((fl=open(FLOPPY,O_WRONLY))<0)
59  {
60  if(errno==EROFS) fprintf(stderr,
    "%s irasvedett\n",FLOPPY);
61  else perror("File-megnyitas irasra");
62  return(1);
63  }
64  if((n=write(fl,puff,
    FLOPPY_SIZE))!=FLOPPY_SIZE)

65  {           /* baj van, hibakod meg nincs de nem is
    erdekel */
66  if(n) fprintf(stderr,"Irasi hiba\n");
67  else perror(FLOPPY);
68  return(1);
69  }
70  if(fsync(fl))
71  {           /* baj van, hibakod rendelkezesre all,
    hat kiirjuk*/
72  perror("Irasi hiba");
73  return(1);
74  }
75  close(fl);
76  fprintf(stderr,"Ha akar meg masolatot, tegye
    a cellemezt %s meghajtoba\n",
77  letter);
78  fprintf(stderr,"\a\rKer ujabb masolatot (I/n)? ");
79  n=getchar();
80  } while(n!='n' && n!='N');
81  }
```

A legnagyobb előny, a memóriakezelés szabadsága a 32 bites rendszernek köszönhető. A teljesen egybefüggően kezelt, szegmensek nélküli memóriakezelés fölszabadítja a DOS és Windows 3.1 korlátait. (Hasonló csak a Win32-ben van.)

Az másik szembeszökő dolog, hogy a hajlékony mágneslemez-egység hagyományos file-ként kezelhető. Semmi speciális rendszerhívásra nincs szükségünk, szemben a DOS bios_disk-jével. A 8. és 9. sorban beállítható, hogy melyik egységet kezelje alapértelmezés szerint a program (a második floppynak /dev/fd1 felelne meg), és annak mekkora a kapacitása. A read és write függvények ilyen néha részlegesen is meghiúsulhatnak, ami azt jelenti, hogy valamennyi adatot azért beolvastak, de nem a kért mennyiséget. A rendszer ilyenkor a sikeres rész teljesítését nyugtázza, majd a következő olvasási/frási kísérletre kapjuk vissza a hibakódot - ettől lett egy kicsit szokatlan a programnak a hibakezelést végző része.

Az fsync függvény (ill. rendszerhívás) részletesebb magyarázatot igényel. Mint minden rendes UNIX, a Linux is mindig használ lemez-gyorsítótárat (disk cache). Az fsync szolgál arra, hogy kivárjuk az adatátvitel tényleges befejeződését, a write már a gyorsítótárba történt másolás befejeztekor véget ér. Ez jó mindaddig, amíg nem akarunk lemezt cserélni, tehát lemezcsere előtt kivárjuk a művelet teljes befejeződését. Nem árt tudni, hogy a szabványos C könyvtárban nincs getch, ezért kell a sorvége jelet „lenyelni" a válaszok végén. A UNIX-ban járatos olvasóim visszakérdezhetnek, miért nem '/dev/rfd0' file-t használom a lemez elérésére. Erre a válasz rendkívül egyszerű: Linux-ban ilyen nincs. Hogy mások is értsék, miről beszélgetünk, elárulom, ha lenne ilyen, akkor nem lenne szükség az 'fsync' használatára, mert a gyorsítótárat elkerüli.

Aki még nem találkozott a 'perror' függvénnyel, annak azt is elárulom, hogy a rendszer-hibakódot (az 'errno' változóban található a numerikus értéke) írja ki szövegesen. A program megértéséhez ennyi bizonyára elég. Remélem ez a kis program egy kicsit rávilágított a UNIX-os programozás előnyeire. A további számokban igyekszünk majd kitérni további példákra is.

Nézzük meg a C példaprogram egy bash megvalósítását is! Ennek kapcsán mód nyílik újabb bash konstrukciók megismerésére.

```
 1  #!/bin/bash
 2  # alapertelmezett ertekek
 3  drive=/dev/fd0
 4  nblocks=1440
 5  #fuggvenydefinicio
 6  yesno()
 7  {
 8  savemode=`stty -g`
 9  stty -icanon min 1
10  ans=`dd if=/dev/tty bs=1 count=1 2>/dev/null`
11  stty $savemode
12  [ "$ans" = y -o "$ans" = Y -o "$ans" =
    i -o "$ans" = I -o "$ans" = "" ] && return 0
13    return 1
14  }
15  waitkey()
16  {
17  savemode=`stty -g`
18  stty -echo -icanon min 1
19  dd if=/dev/tty bs=1 count=1 2>/dev/null
```

```
20 stty $savemode
21 return 0
22 }
23 # parameterek feldolgozasa
24 for i
25 do
26 case $i in
27 [aA]:)
28 drive=/dev/fd0;;
29 [bB]:)
30 drive=/dev/fd1;;
31 360|720|1200|1440|2880)
32 nblocks=$i;;
33 -[0-7])
34 drive=/dev/fd${i#-};;
35 ds)
36 nblocks=360;;
37 dd)
38 nblocks=720;;
39 hd)
40 nblocks=1200;;
41 qd)
42 nblocks=1440;;
43 ed)
44 nblocks=2880;;
45 help) echo
46 cat << EOF
47
48 Floppy-lemez masolo program.
49
50 Hasznalata: ${0##*/} [parameterek]
51 Ahol parameterek lehetnek:
52 drive:     a: vagy b:
53 lemezmeret: 360, 720, 1200, 1440 vagy 2880
54 ds, dd, hd, qd vagy ed
55 segitseg:  help (bar erre mar ugyis rajott)
56
57 Peldaul:
58 ${0##*/} b: 1200     B: drive, 1.2MB-os floppy
59 ${0##*/} a: qd       A: drive, 1.44MB-os floppy
   (alapertelmezes)
60
61 EOF
62       exit;;
63   esac
64 done
65 [ "$drive" = "/dev/fd0" ] && letter="az A:"
66 [ "$drive" = "/dev/fd1" ] && letter="a B:"
67 echo "Kerem a forraslemezt $letter meghajtoba"
68 echo -n "Nyomjon meg egy gombot "
69 waitkey; echo
70 dd if=$drive of=/tmp/flcp.$$ bs=1k
   count=$nblocks 2>/dev/null
71 [ "$?" != "0" ] && { rm -f /tmp/flcp.$$; echo
   "Olvasasi hiba"; exit }
72 echo "Kerem a cellemezt $letter meghajtoba"
73 echo -n "Nyomjon meg egy gombot "
74 waitkey; echo
75 dd of=$drive if=/tmp/flcp.$$ bs=1k
   count=$nblocks 2>/dev/null
76 [ "$?" != "0" ] && echo "Irasi hiba"
77
78 echo "Ha meg egy masolatot akar, tegye
   a cellemezt $letter meghajtoba"
79 echo -n "Meg egy masolat (I/n)? "
80 while yesno
81 do
82 echo
83 dd of=$drive if=/tmp/flcp.$$ bs=1k
   count=$nblocks 2>/dev/null
84 echo "Ha meg egy masolatot akar, tegye
   a cellemezt $letter meghajtoba"
85 echo -n "Meg egy masolat (I/n)? "
86 done
87 rm -f /tmp/flcp.$$
88 echo
```

Az első új dolog ebben a programban a 6-14. és a 15–22. sorokban található függvénydefiníciók.

A *függvénynév() { függvénytörzs }* megadásával az adott bash programon belül használható új parancsot definiálunk.

A 8. sorban elmentjük a savemode változóba az érvényes képernyő-meghajtó beállításokat. Ehhez tudnunk kell, hogy a visszaidézőjelek (`) közé írt parancsot a bash végrehajtja, és a program által a szabványos kimenetre írt szöveget adja a változónak értékül. Például:

```
konyvtar=`pwd`
```

A fenti parancs hatására a *konyvtar* változó az aktuális alkönyvtár nevét fogja tartalmazni.

Az stty -g parancs pontosan azokat az adatokat írja ki, amire szükségünk van. A 9. sorban átállítjuk az input-kezelést úgy, hogy az olvasás művelet ne várjon sor végéig (-icanon opció), és fejeződjék be az első karakter leütése után (min 1 opció). Ezek hatására a terminálról történő olvasás pontosan egy karaktert olvas és nem vár újsor karakterre.

Apropó, terminál: mini-linuxunkon az Alt-F1...Alt-F4 billentyűkombinációkkal négy önálló virtuális terminál között ugrálhatunk (gyakorlatilag olyan, mintha Windows alatt különböző ablakokban dolgoznánk). A négy terminálhoz tartozó eszközök rendre /dev/tty1, /dev/tty2, /dev/tty3 és /dev/tty4. Jó lenne, ha a program minden terminálon változtatás nélkül futna. Erre szolgál a /dev/tty, amely mindig az aktuális terminált jelenti. Egyébként a terminál név az még az ős-UNIX rendszerektől ered, ahol a géphez soros vonalon (ez lehet modem segítségével telefonvonalon keresztül is) kapcsolt teletype készüléket jelentette. Ez persze ma is ugyanúgy lehetséges, de ezek közül ma már valódi jelentősége csak a modemes belépésnek van. A fentiek tükrében a 10. sor dd parancsa egy bájtot másol a terminálunkról a szabványos kimenetére, amit a visszaidézőjelek hatására a bash az ans változóba tesz. A bash while ciklusa minden egyes lefutásakor kiértékeli a while és do közötti parancssort, és annak sikeres (0 értéket visszaadó) volta esetén végrehajtja a do és done között megadott parancsokat.

Visszatérve lemezmásoló programunkhoz, annak 12. sora igaz (0) logikai értékkel tér vissza a hívás helyére yesno függvényünkből, ha az i, y vagy a RETURN billentyűt ütöttük le, ellenkező esetben a következő sorban álló hamis értéket adjuk vissza, végezetül a 14. sorban véget ér a függvénydefiníció. A 22. sor for ciklusának teljes alakja a következő lenne:

```
for változó in érték1 [érték2 ...]
```

Az ún. rövidített ciklusfejű formában az egyes értékek felso-

rolása elmarad (és ennek megfelelően az in szócska is). Helyette a parancs hívásakor megadott paramétereken lépked végig. Mivel egy bash programban az összes paraméterre együtt a $@ jelöléssel hivatkozhatunk (az egyes paramétereket $1, $2, $3, ... jelöli), így ez tulajdonképpen a for i in „$@" utasítással egyenértékű.
A bash case parancsával vizsgáljuk az egyes argumentumokat. Az egyes változatok az érték)-kel kezdődnek és a dupla pontosvesszőig tartanak. (C-hez szokottaknak: ez egy case ágnak felel meg.)
Értékként a UNIX-ban szabványos file-név-jokerek is használhatóak konstans értékeken felül. Ilyenek például:

opt*	Az összes opt-tal kezdődő szó (optimum, opt, optoelektronika, stb.).
opt?	Az összes opt-tal kezdődő és utána maximum 1 karaktert tartalmazó szavak (opt, opt1, opt-).
opt[02468]	Az összes opt-tal kezdődő és utána a szögletes zárójelben megadott karakterek közül pontosan egyet tartalmazó szavak (pl. opt2, opt8, de nem opt, opt28).
opt[0-9]	Az összes opt-tal kezdődő, majd pontosan 1 számjegyet tartalmazó szó. A '-' segítségével tartományt adhatunk meg.
opt[-a-z0-9]	Az összes opt-tal kezdődő, majd pontosan 1 kisbetűt, számjegyet vagy mínuszjelet tartalmazó szavak (pl. opt-, opt8, de nem optA). Figyeljük meg, hogy a '-' speciális jelentését (tartomány kijelölése) úgy hatástalanítottuk, hogy a legelejére vagy a legvégére írtuk.

Visszatérve a programhoz: A 0-7 rövidítés értelemszerűen a 01234567-et jelenti, és azt programunk a hajlékony mágneslemez eszközsorszámnak értelmezi. Bár a Linux lekezel 8 floppy-t, a legtöbb gépben azért nincs kettőnél több floppy drive. Mégis, fő a biztonság... A 31–44. sorok az egyes lemezméreteket kezelik, és az nblocks változót beállítják a kilobyte-ban kifejezett kapacitásra. A 46. sorban a felhasználót tájékoztatjuk a program paramétereiről, használatáról.
A cat parancs kilistázza egy file tartalmát, ill. file-név hiányában a bemenetére (standard input) érkező szöveget a kimenetére másolja. Az itt használt konstrukció (<< EOF) azt jelenti, hogy vedd a szöveget a következő EOF karakterláncig, és azt add a cat parancs bemenetére. Ez tehát a 47-60. sorok tartalmát a cat bemenetére adja, ami azt kiírja. Ezt nem túl jó magyar fordításban itt a dokumentum-nak szokták írni. A 63. sorban zárjuk a case parancsot, majd a done paranccsal a for ciklust. A 70. sor beolvassa egy ideiglenes file-ba a lemez tartalmát. A file-név képzésére a $$ bash változót használjuk, ami a futó program egyedi azonosítóját jelenti. Ezt processzsorszámnak hívják, és garantált, hogy egy időben csak egyszer osztja ki a Linux ezt a számot, így véletlenül sem próbálunk többen ugyanabba a file-ba írni.
A beolvasott adatok visszaírását egy hasonló utasítás végzi, ez látható a 75. és 83. sorokban.
A bash while ciklusa úgy működik, hogy a while és do kulcsszavak között megadott parancs sikeres (0 visszatérési értékű) lefutása esetén a do és done kulcsszavak közötti parancsokat végrehajtja, különben kilép a ciklusból. Az üres echo parancsok egy soremelést eredményeznek, így ha a kérdésre nem a RETURN gombot nyomjuk le, akkor is jó lesz a képernyőkiírás (enélkül ugyanabba a sorba írná a legközelebbi szöveget, mint a kérdést). A program végén persze (87. sor) töröljük az ideiglenes file-t. Látható, hogy bash alatt megírva is egyszerű programot kaptunk, ami ráadásul már lekezeli a különböző floppykat és lemezméreteket. Sok sikert a használatához!

Az MTOOLS programcsomag

Mint az bizonyára eddig is érezhető volt, megpróbáljuk a DOS-ról a UNIX-ra való átállást minél könnyebbé tenni.
A továbbiakban egy olyan programcsomagról lesz szó, amivel Linux alól tudunk a DOS-os file-rendszerhez hozzáférni.
A programcsomag segítségével a floppy-val végezhetünk különböző műveleteket. A parancsok neve a DOS alatt megszokotthoz hasonló, csak M betűvel kezdődnek.

mdir

Az mdir paranccsal kilistázhatjuk, milyen file-ok vannak a floppy egy alkönyvtárában. A formája:

```
mdir [-w] [filenév]
```

Például:

```
mdir a:/
mdir -w "b:/linux/*.tgz"
```

Az első parancs hatására megjelenik az a: drive főkönyvtárában levő file-ok jegyzéke. Figyeljük meg, hogy a \ jel helyett a / jelet használjuk. A második parancs a b: drive linux alkönyvtárában levő .tgz kiterjesztésű file-okat listázza ki „széles", azaz többoszlopos formában. A file-nevet azért kellett idézőjelek közé tenni, mert *-ot tartalmaz. Minden mtools segédprogramra igaz: ha *-ot akarunk használni egy DOS-os file nevében, idézőjeleket kell használnunk. Mit ír ki a következő parancs?

```
mdir
```

DOS alatt a dir az aktuális alkönyvtár tartalmát listázná ki. Most viszont az aktuális alkönyvtár UNIX-os. A fenti parancs az a:/ tartalmát fogja kilistázni. A megoldást az mcd parancs adja.

mcd

Az mtools programok nyilvántartanak egy aktuális DOS-os alkönyvtárat is a UNIX-os mellett. Ennek értékét megnézni ill. megváltoztatni az mcd parancs segítségével lehet. Használata:

```
mcd [alkönyvtárnév]
```

Ha nem adunk meg paramétert, kiírja az aktuális DOS-os alkönyvtárat. Ez induláskor a:/. Az alkönyvtárnév - a DOS-tól eltérően - tartalmazhat drive-ot is, pl.

```
mcd b:/linux
```

Végül egy tanács: **ne használjunk mcd ..** -ot. A tapasztalat szerint elég sok gond van vele.

mcopy

Az mcopy segítségével a floppyról másolhatunk a UNIX filerendszere alá (és persze fordítva is). Használata:

```
mcopy [-tnm] forrásfile célfile
mcopy [-tnm] forrásfile1 [forrásfile2...] célfile
```

Az első alak egy file másolására szolgál. Vagy a forrás-, vagy a célfile-nak tartalmaznia kell egy drive megjelölést, innen tudja a program, hogy melyik a UNIX-os oldal. A következő parancs például a floppyról felmásolja a proba.txt file-t a

UNIX rendszer /tmp alkönyvtárába a proba.text néven. A -t kapcsoló hatására a DOS-os és a UNIX-os text file-ok között szükséges konverziót a másoláskor megcsinálja.

```
mcopy -t a:/szoveg/proba.txt /tmp/proba.text
```

Lássunk még egy példát:

```
mcopy "a:/*" ~
```

Ez a parancs az A: drive főkönyvtárából minden file-t felmásol a saját alkönyvtárunkba. (A ~ UNIX alatt mindig a saját alkönyvtárunkat jelenti. Ha en-ként léptünk be, ez a /home/ent jelenti.)

Vegyük észre, hogy a * egy kicsit másként működik, mint DOS alatt. Ott a fenti parancs csak a kiterjesztés nélküli file-okat másolta volna fel. A UNIX-os programok (és így az mcopy is) a file-nevet egy és oszthatatlannak tekinti. A kiterjesztést sem veszi külön. Így pl. a *. jelenti a pontra végződő neveket, azaz amiknek nincs kiterjesztésük, a * magában minden file-t jelent. A *C jelenti a C-re végződő file-neveket, mint FLCP.C, KIS.CC és KANOC is. Ha a C file-okat akarjuk kiszűrni, itt is működik a *.C szekvencia.

Lássunk egy példát, amikor több file-t adunk meg:

```
mcopy -n ~/src/*.c ~/src/*.pas b:/src
```

Ez a saját alkönyvtárunkból nyíló src alkönyvtárból (pl. /home/en/src) másol néhány file-t a b:/src alá. A UNIX-os oldalon nem szabad idézőjelet használnunk. Az mcopy általában visszakérdez, mielőtt egy létező file-t felülír. Az -n kapcsoló hatására viszont szó nélkül felülírja. Végül az alábbi parancsok az -m kapcsoló hatására megőrzik a file utolsó módosításának idejét (a két változat azonosan működik).

```
mcopy -mt ~/txt/fontos.txt a:/
mcopy -m -t ~/txt/fontos.txt a:/
```

mdel

Az mdel parancs DOS-os file-ok törlésére való. Használata:

```
mdel [-v] filenév1 [filenév2...]
```

A DOS-os del-től eltérően ennek több file-nevet is megadhatunk, pl.

```
mdel -v a:/proba.bak a:/*.tmp
```

A fenti parancs törli a megadott file-okat. A -v kapcsoló hatására minden törölt file nevét kiírja.

mmd és mrd

Az mmd paranccsal DOS-os alkönyvtárakat hozhatunk létre, az mrd-vel üres DOS-os alkönyvtárakat szüntethetünk meg. Használatuk:

```
mmd [-v] alkönyvtárnév1 [alkönyvtárnév2...]
mrd alkönyvtárnév1 [alkönyvtárnév2...]
```

Ezek használata nyilvánvaló. Egy dologra azért oda kell figyelni. Mit csinál a következő parancs?

```
mmd b:/proba+
```

A + karakter ugyanis DOS-os névben nem megengedett. A fenti parancs létrehoz egy b:/probax nevű alkönyvtárat. Ha megadjuk a -v kapcsolót, akkor ilyen esetekben kiírja az általa generált nevet.

Sajnos az mmd parancsnak is van egy kis „szépséghibája". Nem zavarja, ha olyan nevű file már létezik, mint az új alkönyvtár. Létrehozhatunk pl. b:/proba.txt nevű alkönyvtárat akkor is, ha ilyen néven már van egy file. Az ilyen nevű alkönyvtárba aztán DOS alatt nem tudunk bemenni, ott file-okhoz hozzáférni (az mcopy, mcd, stb. UNIX alatt megy). Ezt a hibát ráadásul a DOS-szal adott chkdsk.exe nem tudja javítani (a Norton Disk Doctor megbirkózik vele). Ha már az mmd nem teszi, figyeljünk mi oda.

mtype

Az mtype a DOS-os type megfelelője: file-ok tartalmának kiírására szolgál. Használata:

```
mtype [-t] filenév1 [filenév2...]
```

A következő parancs például kiírja az a:/autoexec.bat file tartalmát.

```
mtype -t a:/autoexec.bat
```

A -t kapcsoló az mcopy-nál már említett szövegformátumok közti konverziót hajtja végre automatikusan.

mattrib

Ennek használata teljesen megegyezik a DOS alatti attrib.exe-ével.

```
mattrib [-a|+a] [-r|+r] [-s|+s] [-h|+h] filenév1 [filenév2...]
```

Az már más kérdés, hogy az mcopy-t és mdel-t nem zavarja a csak olvasható file (r flag), az mdir mindig kilistázza rejtett ill. rendszer file-okat (h ill. s flag) is...

mren

Az mren paranccsal egy DOS-os file-t vagy alkönyvtárat nevezhetünk át. Használata:

```
mren [-v] forrásfile célfile
```

Ha a célfile neve nem szabályos DOS név, ez a program is átnevezi új névre. Ezt követhetjük nyomon a -v kapcsoló segítségével. Ha egy file-t megpróbálunk proba+ -ra átnevezni, akkor az mren - tudván, hogy ez DOS alatt nem szabályos név - inkább a probax nevet adja. **Kerüljük az olyan neveket, amiket az mren megpróbálhat átnevezni.**

mlabel

Egy lemez nevét változtathatjuk meg vele. Használata:

```
mlabel [-v] drive:
```

A használata a DOS alatt megszokott. Ez a program is átnevezi a neki (ill. a DOS-nak) nem tetsző neveket, de ezzel még nem volt probléma. Egyébként még így is jóval többféle nevet enged meg, mint a DOS. Mivel a DOS-t ez nem zavarja, így csak jól járunk.

mread és mwrite

Alacsonyszintű másolás DOSÆUNIX ill. UNIXDOS irányban. Használata:

```
mread [-tnm] dosfile unixfile
mread [-tnm] dosfile1 [dosfile2...] unixalkönyvtár
mwrite [-tnmv] unixfile dosfile
mwrite [-tnmv] unixfile1 [unixfile2...] dosalkönyvtár
```

A paraméterek szerepe azonos az mcopy parancsnál látottakkal. Mivel itt a parancsból kiderül, melyik a UNIX-os és melyik a DOS-os file-név, így itt lehet drive megadása nélkül is hivatkozni a DOS-os file-ra.

Folytatás következik

A későbbiekben is igyekszünk olyan programokat adni, amelyekkel a DOS és UNIX kapcsolata fájdalommentesebbé tehető. Következő számunk mellé egy „nagyágyút", egy teljes DOS emulátort adunk. Akiket érdekel, ebben a számban egy kis kedvcsinálót találnak róla.

Bálint Nagy Endre
E-mail: bne@cert.hunix.hu

Szekeres Béla
E-mail: szekeres@evt.bme.hu

DOS programok LINUX alatt

Egyre többen használják és szeretik a Linux operációs rendszert. A viszonylag kis hardveren is jól futó PC-s UNIX méltán lett népszerű. Újabb és újabb programok jelennek meg hozzá, ezek általában szabadon másolhatók. Ma már a szövegszerkesztőtől a táblázatkezelőig, a teljes matematikai programcsomagoktól a hálózatmenedzser programokig széles a választék. Ez a kör egy hasznos, a rendszer további elterjedését nagyban elősegítő programmal bővült: a DOS emulátorral.

Következő számunk lemezmellékletén megtalálják majd ennek a programnak a legfrissebb változatát, most ennek szeretnénk egy kis reklámot csinálni.

A programot a Linux rendszerből elindítva betöltődik a merevlemezünkön található DOS, és innen kezdve (majdnem) minden DOS-os programunkat használhatjuk. Kihasználva a Linux lehetőségeit, átválthatunk egy másik képernyőre, ahol UNIX-os programokat futtathatunk. Ha kedvünk van hozzá, elindíthatunk további DOS emulátorokat is, és a DOS-os képernyők között gombnyomásra váltogathatunk.

Ha egy kicsit erősebb gépünk van, amire telepítettük az X-Windows-t (ez egy UNIX-ra épülő grafikus környezet), és beérjük karakteres DOS alkalmazásokkal, akkor egy képernyőn több DOS ablakot nyithatunk, ezek jól megférnek UNIX-os programjainkkal. Ha mégis ragaszkodunk a grafikus programokhoz, akkor az X-Windows-os képernyő mellett még mindig rendelkezésünkre áll néhány szöveges képernyő, ahol szintén lehet DOS emulátort futtatni - ez esetben teljes képernyős módban, grafikával. És mindezt egy 486DX-33-mas számítógépen 8MB RAM-mal!!

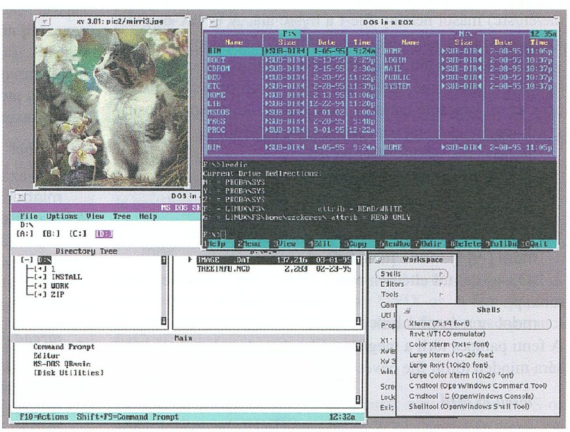
Többen kérdezhetik, miért jó ez nekünk? Ezt a Microsoft Windows is tudja. Ezt még igen, de egyrészt UNIX alá szabadon másolható programok széles választéka áll rendelkezésünkre, másrészt a DOS emulátorral kihasználhatjuk az alatta futó UNIX szolgáltatásait is.

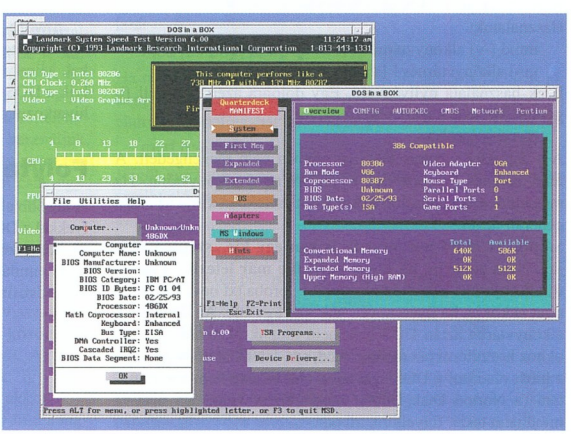
A DOS emulátor alól elérhetjük a UNIX filerendszerét, annak minden előnyével együtt. Egyrészt lehetőségünk van több felhasználót definiálni, a file-okhoz való hozzáférést korlátozni, amit természetesen a DOS emulátor alól se lehet kikerülni. Ha Józsi dolgozik a gépen, akkor ő a DOS emulátor alatt is csak a saját file-jaihoz férhet hozzá. Másrészt több UNIX-os gépet hálózatba összekötve lehetőségünk van printerek, merevlemezek közös használatára. Az irodában csak Judit gépében van CD-ROM, de mivel mindkét gépen Linux fut, így ahhoz Józsi is hozzáfér; természetesen a DOS emulátor alól is.

Mivel a DOS emulátor igen jól konfigurálható, így fejlesztők is nagy hasznát vehetik. Beállíthatjuk milyen processzort emuláljon (286, 386 és 486) és legyen-e co-processzor. Mit szól a programunk, ha túl kevés (vagy esetleg meglepően sok) memória van a gépben? Megadhatjuk mennyi memóriát lásson a DOS, ez ráadásul lehet több is, mint amennyi fizikailag a gépben van, a Linux majd virtuális memóriának használja a merevlemezt (swap-et). Konfigurálhatjuk a soros és párhuzamos port-okat. Biztos, hogy a soros port vezérlésére írt programunk működik COM3-mon is? Kipróbálni nem tudtuk, mert ilyen port nincs a gépünkben: most majd beállítjuk, hogy COM3-ként lássa a fizikailag COM1-re konfigurált soros portot, és IRQ 5-öt használjon hozzá az eredeti IRQ 4 helyett.

Reméljük sikerült kedvet csinálni ehhez a hasznos segédeszközhöz. A következő számban részletesen olvashatnak róla.

Szekeres Béla
E-mail: szekeres@evt.bme.hu





BACKTRACK

Sokan kérnek tőlünk programozói fogásokat, algoritmusokat. A következőkben egy olvasónk készülő könyvéből közlünk egy általánosan használható érdekes keresési eljárást.

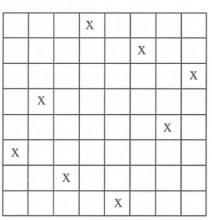
A backtrack (visszalépéses keresés) algoritmus ismertetése előtt nézzünk meg három olyan feladatot, melyeket ezen módszerrel sikeresen meg tudunk oldani.

1. Feladat

Legyen egy SZEMÉLYEK nevű adatbázisunk, melynek rekordszámát a reccount() fügvénnyel lehet meghatározni. Az adatbázis szerkezete tartalmazza – sok más adat mellett – a személy azonosítóját és foglalkozását. Keressünk 5 olyan embert, akinek a foglalkozása megegyezik.

2. Feladat (A 8 királynő probléma)

Ezzel a klasszikus feladattal Gauss foglalkozott először. A kérdés az, hogy hányféle módon lehet 8 királynőt elhelyezni a sakktáblán úgy, hogy azok ne üssék egymást. Egy lehetséges állást az alábbi ábra mutat, amely azt is bizonyítja, hogy van legalább egy megoldás.



A királynők nem ütik egymást, ha teljesülnek az alábbi feltételek.

(i) nincsenek egy sorban
(ii) nincsenek egy oszlopban
(iii) nincsenek egy átlón
(egy négyzet 2 db átlón is van)

3. Feladat

Adott egy HOSSZ hosszúságú acélrúd. A rúdból mindig a vagy b vagy c hosszú darabokat vághatunk le mindaddig, amíg el nem fogy. A fenti darabokból a fenti sorrendnek megfelelően A, B, C Ft/darab értékű terméket gyárthatunk. Adjuk meg azt, hogy az egyes termékekből mennyit gyártsunk az alábbi feltételek mellett:

(i) az acélrúdból ne maradjon hulladék,
(ii) maximális értéket termeljünk.

A backtrack algoritmus feladatának meghatározása

Legyen n darab ($n \in N$) halmazunk, melyek nem üresek (azaz legalább 1 db elemet tartalmaznak) és végesek (azaz egyiknek sincs végtelen sok eleme). Az egyes halmazok elemei tetszőleges dolgok lehetnek. Jelöljük az egyes halmazokat a következő szimbólumokkal: $H_1, H_2, \ldots, H_n$. Definiáljunk egy H halmazt, amely ezen halmazok descartes sorozata:

$$H:=H_1xH_2x\ldots H_n$$

A H halmaz elemei rendezett n-esek, azaz $(h_1, h_2, \ldots, h_n)$ sorozatok, ahol az első pozíción lévő elem a H_1, a második pozíción lévő elem a H_2, …, az n. pozíción lévő elem a Hn halmazból kerül ki. Ha a Hi halmaz elemeinek számát $|H_i|$ jelöli, akkor a H halmazban lévő sorozatok száma: $|H_1| * |H_2| * \ldots * |H_n|$. A descartes sorozat alapján a reláció fogalma könnyen megadható: Egy R reláció a H halmaz egy részhalmaza. Meghatározásunk szerint tehát az R reláció egy olyan halmaz, amely bizonyos H halmazbeli sorozatokból áll. A leírt fogalmak jobb megértése érdekében az 1. és 2. feladatok segítségével konkrét példákat adunk.

1. Példa

A SZEMÉLYEK adatbázist egy halmaznak képzelhetjük el, ahol a halmaz elemei az egyes rekordok. Tekintsünk most 5 darab halmazra: $H_1=\ldots=H_5$=SZEMÉLYEK. Látható, hogy az egyes halmazok most nem különböznek egymástól. Képezzük ezen halmazok H descartes szorzatát, amelybe minden elképzelhető ember-ötös benne van. Az R reláció szöveges megfogalmazása a feladatban már megvan: azonos a foglalkozásuk (természetesen különböző emberekről van szó, hiszen H-ban olyan sorozat is van, amelynek minden tagja ugyanaz az ember). Az R halmaz tehát ($R \subseteq H$) azon ember-ötösök halmaza, akik ugyanazzal foglalkoznak.

2. Példa

A sakktábla 8 darab oszlopa jelentsen nekünk 8 darab ($H_1, H_2, \ldots H_8$) halmazt, melyek mindegyike a {1,2,3,4,5,6,7,8} halmazzal egyezik meg. Az egyes halmazelemek egy-egy négyzetet jelentenek, amiből minden oszlopban 8 darab van (mert a tábla 8 soros). Képezzük a H halmazt, mint az előző 8 darab halmaz descartes szorzatát. A H halmaz elemei most olyan nyolcas sorozatokból állnak, ahol a sorozat első eleme az első oszlop egy megadott sorban lévő négyzete, a sorozat második eleme a második oszlop egy megadott során lévő négyzet stb… Ha egy tetszőleges H halmazbeli sorozatnak megfelelően teszünk le 8 királynőt, akkor azok vagy üthetik egymást, vagy sem. Az R reláció szöveges megfogalmazása: egyik királynő se ütheti a másikat. Az R halmaz tehát azon H-beli sorozatokból fog állni, amely sorozatok elemei R (= nem ütik egymást) relációban állnak egymással. A 2. feladat megfogalmazásakor megadtuk az R reláció egy ekvivalens megfogalmazását az i-ii-iii pontokban. Vizsgáljuk meg, hogy két királynő mikor van egy átlón! Az első észrevétel az, hogy két átló típus létezik:

– BJ átló: balról jobbra emelkedő,
– JB átló: jobbról balra emelkedő.

Mindkét átlótípusból 15 darab van a sakktáblán. A második és harmadik észrevétel nagyon izgalmas a feladat megoldása szempontjából:
2. észrevétel: Egy-egy BJ típusú átló mentén minden négyzetre igaz, hogy az oszloppozíció és sorpozíció különbsége egy konstans szám. A főátlón ez nulla, alatta eggyel +1, felette –1.
3. észrevétel: Egy-egy JB típusú átló mentén minden négyzetoszlop és sorpozíciójának összege állandó. A mellékátlón ez az összeg például 9.
Az észrevételek alapján könnyen megfogalmazhatjuk azt, hogy egy (x1, y1) és (x2, y2) pozícióra feltett két darab királynő mikor nincs egy átlón (az első szám az oszlop, a második a sorpozíciót jelöli):
Nincs egy átlón=($x1-y1 \neq x2-y2$ és $x1+y1 \neq x2+y2$)
A fenti feltétel tömörebben is megfogalmazható, ugyanis:

$$x1-y1 \neq x2-y2 \Leftrightarrow x1-x2 \neq y1-y2 \text{ és}$$
$$x1+y1 \neq x2+y2 \Leftrightarrow x1-x2 \neq y2-y1 = -(y1-y2)$$

A fentiek együttes teljesülésének ekvivalens alakja:

$$x1-x2 \neq |y1-y2| .$$

A backtrack feladat általános megfogalmazása

A fenti példákon is érzékelhető, hogy a backtrack algoritmus olyan feladatokat old meg, ahol létezik egy H descartes szorzattal létrehozott halmaz, melyben olyan sorozatot (mint H halmazbeli elemet) kell megkeresnünk, amelyben a sorozatelemek egymással valamilyen R relációban állnak. Tekintettel arra, hogy ez a problémamegfogalmazás igen általános, ezért a backtrack algoritmus széles körben használható a mesterséges intelligencia területén (pl. Prolog programozási nyelv). A backtrack algoritmus precíz megfogalmazhatósága érdekében vezessünk be néhány jelölést és definíciót.

Legyen n darab függvényünk, melyek:

$$\left.\begin{array}{l} f_1: H_1 \rightarrow \{1,2,\ldots, |H_1|\} \\ f_2: H_2 \rightarrow \{1,2,\ldots, |H_2|\} \\ \ldots\ldots \\ f_n: H_n \rightarrow \{1,2,\ldots, |H_n|\} \end{array}\right\}$$

Ezek a függvények a H_i halmazok elemeit besorszámozzák egytől az utolsó elemig (azaz ezek kölcsönösen egyértelmű leképzést jelentenek). Ez azért jó, mert így a H_i halmlaz valamelyik elemére annak sorszámával is hivatkozhatunk, ami egy kódolást jelent. Ezután a H_i halmaz helyett a neki megfeleltetett egész számok halmazát is használhatjuk az algoritmus kifejtése során. Tehát a Hi halmaz egy alias halmaza: $f_i(H_i)$ függvényérték halmaz. A leírtak szellemében a Hi.j jelentse a H_i halmaz j. elemét. Ez mindig egy számot jelent, csak a jelölésben azt is feltüntettük, hogy melyik halmazból származik.

A feladatot általánosan megoldó backtrack algoritmus

A keresett R relációban lévő n-est tároljuk a BTV[1..n] of NATURAL vektorban. A BTV a backtrack vektor rövidítése. A NATURAL típus értékei a nem negatív egész számok lehetnek.

```
// ---------------------------------------------------------------------
//Backtrack algoritmus. Kitölti a BTV vektort úgy, hogy abba
//egy olyan számsorozat kerül, amely eleget tesz egy R relá-
//ónak. Ez egy keresési feladat a H descartes sorozat halma-
//zon, így nem biztos, hogy van R-beli elem. Emiatt a Back-
//track függvény visszatérési értéke TRUE, ha a BTV érvé-
//nyes, különben FALSE.
// ---------------------------------------------------------------------
boolean Backtrack) BTV )     //a BTV vektor
// ---------------------------------------------------------------------
{

    //Az index jelöli, hogy most éppen a BTV melyik indexét
    //akarjuk kitölteni
    index=1;
    BTV[index]=0;
    //Ciklus amíg a BTV vektor minden elemét nem állítottuk
    //be, vagy az index nem 0. Ha az index=0, akkor a
    //BTV[1]-nek se lehet jó értéke, ami a keresés kudarcát je-
    //lenti.
    while ( 1≤index and index≤n )
    {

        //Az adott BTV[index] érték keresése
        if( Keres( index ))
        {
            index=index+1;     //ha talált jöhet a következő
            halmaz
            if( index≤n ) BTV[index]=0;
        }
        else
        }
            index=index–1;     //Visszalépés az előző indexre
        }
    }
    return( index n );

}//end backtrack

// ---------------------------------------------------------------------
//A Keres(index) függvény folytatja a keresést az f_index(H_index)
//halmazban BTV[index]+1 elemtől kezdődően mindaddig,
//amíg nem talál olyan elemet, amely megfér az R relációval.
//Ha nincs ilyen elem akkor FALSE, különben TRUE a
//visszatérési értéke a Keres-nek
// ---------------------------------------------------------------------
boolean Keres( index )          //a halmaz sorszáma, amiben
                                most keresünk
// ---------------------------------------------------------------------
{
    i=BTV[index];               //Jelenlegi érték az indexedik
                                pozíción. Ez a H_index.i elem.
    while ( TRUE )
        i=i+1;
        if(i > |H_index| )
        {
            return( FALSE );
        }
        if( !Rossz( index, i ))  //Ha nem Rossz (azaz jó)
        {
            BTV[index]=i;
            return( TRUE );
        }
    }
}//end Keres

// ---------------------------------------------------------------------
//A Rossz( index, i ) logikai függvény arra ad választ, hogy a
//H_index.i elem az adott R reláció kialakítása szempontjából
//megfér-e a már eddig kiválasztott
//H_1.BTV[1], … H_index-1.BTV[index-1] elemekkel. Ha igen, ak-
//kor a visszaadott érték TRUE
// ---------------------------------------------------------------------
boolean Rossz( index, i )       //A H_index halmaz i. elemét
                                vizsgáljuk
// ---------------------------------------------------------------------
    {
    j/1;
    while( j<index and H_j.BTV[j] és H_index.i pár megfér egy-
    mással )
    {
        j=j+1;
    }
    return( j<index );
}//end Rossz
```

Az 1. feladat megoldása az általános algoritmus alapján

A BTV vektor most 5 dimenziós lesz, hiszen 5 darab halmazból kell választani. Ennek megfelelően az n=5. A halmazok elemszáma: reccount(). Ezen adatoktól eltekintve a Backtrack függvény változatlan marad. A Keres függvény is csak annyit módosul, hogy a $|H_{index}|$ = reccount(). A Rossz(index, i) változik a legtöbbet, hiszen ezen a szinten kell megmondanunk azt, hogy a „megfér" igaz-e. Erre egy függvényt érdemes írni:

```
boolean megfér( index1, index2 )
{
    if(   SZEMÉLYEK.BTV[index1].Foglalkozás==SZEMÉ-
    LYEK.BTV[index2].Foglalkozás
                                and
          SZEMÉLYEK.BTV[index1].Azon !=
          SZEMÉLYEK.BTV[index2]. Azon
              return( TRUE );
else
              return( FALSE );
}
```

A függvényben alkalmazott jelölés értelmezése: A SZEMÉLYEK.k az állomány k. rekordját jelenti. A minősített hivatkozás folytatása ezen rekord Foglalkozás mezőjének az értéke. A feltétel második része biztosítja azt, hogy a BTV vektor elemei most különbözőek legyenek.

A 2. feladat megoldása az általános algoritmus alapján

A BTV vektor most 8 dimenziós lesz, ugyanis itt most egy olyan nyolcas sorozatot keresünk, ahol az a reláció teljesül, hogy nem ütik egymást. A konkrét BTV vektorunk tehát: BTV[1..8] of { 1,2,3,4,5,6,7,8 }. Ennek megfelelően például a BTV[5] azt mondja meg nekünk, hogy az 5. oszlopban melyik soron van a királynő. Fogalmazzuk meg ezek alapján a „megfér" logikai függvényt!

```
// ---------------------------------------------------------------------
//Az index1 és index2 most 2 különböző oszlopra utal. Ez már
//kizárja azt, hogy azonos oszlopra kerüljön két királynő
// ---------------------------------------------------------------------
boolean megfér( index1, index2 )
{
    //Ha azonos sorban vannak, akkor ütik egymást, így nem
    //férnek meg
    if( BTV[index1]==BTV[index2] return(FALSE);

    //Ha nincsenek egy átlón, akkor megférnek
    if( Abs( BTV[index2]–BTV[index1] ) !=index2–index1 )
        return( TRUE );
    else
        return( FALSE );
}
```

A 3. feladat megoldása az általános algoritmus alapján

Ezt a feladatot még nem fogalmaztuk meg a halmazok és relációk nyelvén, ezért először ezt tegyük meg. Az a tény, hogy 3 különféle termék van sejteti, hogy 3 alaphalmazunk lesz. De mik lesznek ezek? Konstruáljuk meg úgy az alaphalmazunkat, hogy először azt döntjük el, hogy milyen lesz a BTV vektor. A BTV legyen 3 dimenziós, mégpedig azért, hogy a BTV[1] megmondja, hogy az „a" hosszú termékből mennyit, a BTV[2] azt, hogy a „b" hosszú termékből mennyi, valamint a BTV[3] azt, hogy a „c" hosszú termékből mennyit kell készítenünk. A legkevesebb amit gyárthatunk az nulla. A legtöbb az az, amikor csak az egyik termékből gyártunk, illetve vegyük ezek maximumát:

M = Egész(max { 1+HOSSZ/a, 1+HOSSZ/b, 1+HOSSZ/c }+1)

Ez alapján definiálható a H1=H2=H3={0,1,2,…,M } halmaz. Innen a H descartes szorzat már megvan. Még az R reláció megalkotása van hátra. Ez most nagyon egyszerű, hiszen csak annak kell teljesülnie, hogy a HOSSZ hosszú rúdból nem marad hulladék, azaz minden olyan hármas sorozat jó, ahol az elemek együttesére teljesül a:

HOSSZ=BTV[1]*a+BTV[2]*b+BTV[3]*c

feltétel. Azt eddig is tudtuk, hogy az R halmaz több elemből is állhat, azonban mindig megálltunk a keresésben, ha kaptunk egy megoldást. Itt most ezt nem tehetjük meg, ugyanis a feladat csak akkor megoldott, ha megmondjuk azt az R halmazbeli elemet, amely még azzal a speciális tulajdonsággal is rendelkezik, hogy:

(BTV[1] * A + BTV[2] * B + BTV[+3] * C)

kifejezés maximális értékű. Ez egy maximumkeresés az R halmaz felett, amihez pedig végig kell mennünk az R halmaz minden elemén. Ez nem csak ezen feladat sajátja, hiszen ugyanezt kellett volna tennünk akkor is, ha például meg akarjuk mondani azt, hogy hány ütésmentes felállítása van 8 királynőnek. A jelenlegi algoritmusunk hibája, hogy megáll, ha találtunk egy R halmazbeli elemet. A baj a Backtrack függvény ciklusának vezérlése miatt adódik, ezért fogalmazzuk át úgy a függvényt, hogy csak az R halmaz bejárása után álljon le, eközben keressen maximumot is.

```
// ---------------------------------------------------------------------
//2. verzió (minden R halmazbeli elemet érint)
//Backtrack algoritmus. Kitölti a BTV vektort úgy, hogy abba //egy
olyan számsorozat kerül, amely eleget tesz egy R reláció//nak. Ez
egy keresési feladat a H descartes szorzathalmazon, //így nem biz-
tos, hogy van R-beli elem. Emiatt a Backtrack //függvény visszaté-
rési értéke TRUE, ha a BTV érvényes, kü//lönben FALSE.
// ---------------------------------------------------------------------
boolean Backtrack( BTV ) //a BTV vektor
// ---------------------------------------------------------------------
{
    //A mindenkori maximumot tárolja
    Max=0;
    //TRUE lesz, ha az R halmaz nem üres
    RNotEmpty=FALSE;
    //Az idex jelöli, hogy most éppen a BTV melyik indexét
    akarjuk kitölteni
    idex=1;
    BTV[index]=0;

    while( 1≤index )
    {
    //Az adott BTV[index] érték keresése
    if( Keres( index ))
    {
        index=index+1;          //Ha talált jöhet a következő
                                halmaz
        if( index≤n )BTV[index]=0;
    }
    else
    {
        index=index –1;         //Visszalépés az előző indexre
    }
    //Ha komplett lett a BTV, akkor ezt elkönyveljük
    if( index==n+1 )
    {
        if( !RNotEmpty )        //Az első R halmazbeli megta-
                                lálásakor
        {
            RNotEmpty=TRUE;
            Max=MaxKifejezés();
        }
        else if( Max  MaxKifejezés() )
        {
            Max=MaxKifejezés();
        }
        index=index–1;          //Visszalépés Hn halmazra
      }
    }
    return( RNotEmpty );
}//end backtrack
```

A módosított backtrack függvény n=3 esetén megfelel a 3. feladat követelményeinek. A MaxKifejezés függvényt már ismerjük. A feladat végső megoldásához még a „megfér" függvényt kell megírni. A függvény érdekessége, hogy logikai értéke most csak az index2 értékétől függ.

```
boolean megfér( index 1, index2 )
{
    if( index2==1 and BTV[1]*a  HOSSZ ) return( TRUE );
    else if( index2== and BTV[1]*a+BTV[2]*b  HOSSZ ) re-
    turn( TRUE )≥
    else if( index2==3 and
    BTV[1]*a+BTV[2]*b+BTV[3]*c== HOSSZ )
                                    return( TRUR );
    else return( FALSE );
}
```

Nyíri Imre

Oktatás

Államilag elismert, alapfokú számítógép-kezelői szakképesítést nyújtó ***tanfolyam*** *indul. A 150 órás képzés keretében* ***gépkezelés, szövegszerkesztés, táblázatkezelés, adatbázis-kezelés*** *szerepel.*
A képzés díja: ***29 000 Ft + vizsgadíj.***
A tanfolyamok ideje:
délelőtt, délután és ***hétvégén is*** *!*
Csoportoknak kedvezmények! Kérje tájékoztatónkat !

Micro Studio Kft.
Tel/Fax:: 129-1205 1775. Budapest Pf. 38.
Árak az Áfát tartalmazzák. Az árváltoztatás jogát fenntartjuk.

Akciós árak!

CÉGSZERVIZ
1087 Bp., Luther u. 1/b.
☎ 113-1677

✔Mono SVGA monitor	11 800,-
✔Dual Floppy	11 400,-
✔Verbatim lemezek	800,- /10db
✔CD ROM drive 2x	17 800,-
✔Star Lc-20	20 800,-
✔Színes scanner A/4	72 000,-
✔Diktafonok	3 980,-tól
✔Elektromos írógépek	16 800,-tól
✔Manager kalkulátorok	1 880,-tól
✔Iratmegsemmisítők	14 600,-tól
✔Telefon + üzenetrögzítő	8 800.-tól

Magyarországon a legolcsóbban
DIGITÁLIS Gyorsmásolás
2.40 + áfa /oldaltól
Az árak áfa nélküliek.

Száloptika

HA ÖN SZÁLOPTIKÁBAN ÉRDEKELT, AKKOR RÁNK VAN SZÜKSÉGE!

Cégünk optikai átviteltechnikai termékeket forgalmaz raktárról.

- kábelek
- csatlakozók
- szerelt kábelek
- műszerek
- szerszámok
- segédanyagok
- oktatás

JAFCO

ÁTVITELTECHNIKAI KERESKEDELMI KFT.
BUDAPEST, VIII., HORVÁTH MIHÁLY TÉR 14.
TEL.:113-5270, 133-2315, FAX.:113-5279

Lemez

Megérkeztek a megújult
SENTINEL adathordozók.

3,5" 2HD, 135 TPI, *színes, papírdobozos*	**846 Ft**
3,5" 2HD, 135 TPI, *színes, műanyag dobozos*	**1 041 Ft**
CD-R 750 Mbyte (74') *műanyag dobozos*	**1 770 Ft**

Bármely más adathordozóra megrendelést elfogadunk !
Viszonteladók jelentkezését is várjuk !

Micro Studio Kft.
Tel/Fax:: 129-1205 1775. Budapest Pf. 38.
Árak az Áfát nem tartalmazzák. Az árváltoztatás jogát fenntartjuk.

Szakkönyv

Amerikai szakkönyvek legnagyobb választéka.

PC, MAC, UNIX, Mainframe könyvek, több mint 100 kiadótól!

SoftWare Station
T.: 201-6523
1012 Bp. Kosciuszkó Tádé u. 22.

16.000 tételes adatbázis, nyomtatott katalógus, növekvő raktérkészlet.

Oktatás

Oktatói gyakorlattal is rendelkező profi számítógépes szakember, iskolák számára oktatásra alkalmas MACINTOSH, vagy PC-s számítógép rendszerek telepítését üzemeltetését vállalja betanítással. Oktatásra, nyelvoktatásra alkalmas szoftvercsomagok beszerzése! Megrendeléseket: a

1399 Bp, Pf.:701/15 címen, vagy
1631-771-es telefonon/faxon fogadunk.
LEZLISOFT Computer Graphics

Újdonság

A LEZLISOFT Computer Graphics
bemutatja hazai fejlesztésű szoftvereit:

TipoMaker V: 1.11
betűmintakönyv készítő program

- Automatikusan elészíti a Windows alatt üzembehelyezett betűkről, szabadon definiálható karakterekből a betűmintakönyvet.
- Minden betűről részletes tipográfiai jellemzőket készít, pl. "n" számítás az oldaltükör alapján, hibaszázalék-számítás, stb.

Ingyenes helyszíni bemutató és demopéldány. Bizományi értékesítőket keresünk!
Tel.: 1631-771
Cím: Bp. 1399 Pf. 701/15

WinSpeed V:1.0 - átfogó sebességteszt Windows környezetre

- 16 féle mérési szempont;
- Összehasonlítási lehetőség;
- Ábrázolás grafikonon és táblázatban.

COHERENT

C Programozóknak , Fejlesztőknek

GCC/GCC++ 2.5.6.GNU fordító	**9600** Ft +áfa
COHERENT 4.2 UNIX	**15200** Ft +áfa
Komplett UNIX op.rendszer, C és assembler fejlesztővel kpl.	
Teljes GNU sorozat CD-n !!!	**4800** Ft+áfa
X-Windows fejlesztő X11R5	**4800** Ft+áfa
X-Windows fejlesztő X11R6	**4800** Ft+áfa

Nem csak SZOFTVERT, de HARDVERT is érdemes nálunk vásárolni a PC-hez és MACINTOSH-hoz egyaránt.

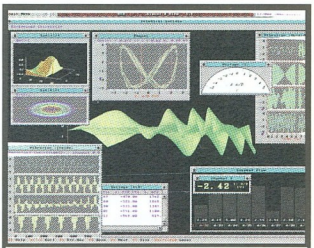
Akciós áron:

Macintosh LC475 129,900 Ft+áfa

Számítógépébe a nálunk vásárolt alkatrészeket díjmentesen beszereljük! SAMSUNG monitorokat, nyomtatókat és faxokat viszonteladók részére is forgalmazunk.

"BECO"Kft. 1091 Budapest, Üllői út 119. (bejárat a Mihálkovics utcából) tel/fax:218-4578, üzenetrögzítő: 217-8592

Mérésadatgyűjtés



Mérésadatgyűjtés és vezérlés 2. Szoftver

SIGNAL * VIEW (COMPFAIR 94 vásárdíjas)
Általános mérésadatgyűjtő és monitorozó program (szabadon konfigurálható mérések, grafikus monitorok és műszerek, széleskörű trigger funkciók, real-time matematika).

SIGNAL * MATH
Jelfeldolgozó és matematikai program (FFT, digitális szűrés, szűrőtervezés).

rtdLinx Driverek
Standard TSR driver az RTD kártyáihoz.

Egyedi szoftverfejlesztés

rtd® "Accessing the Analog World"

Real Time Devices Európa, KFT.
1024 Budapest, Fillér u. 88. II/9.
Tel: 325-1130 Fax: 212-0260

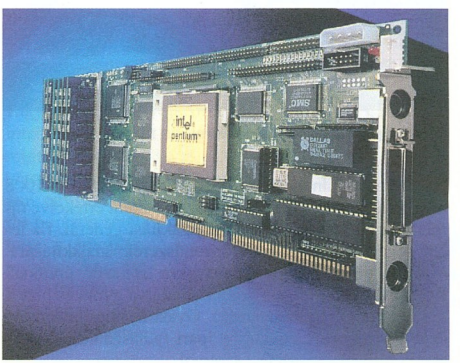
Merev

A Forráskód Hardware melléklete

Egy közelmúltban megrendezett kiállítás apropóján a Merev havi témája a Pc alkalmazása ipari, laboratóriumi környezetben.Az első cikk témája az ipari Pc felépítése, kialakítása. A második cikkünk a digitális mérésadatgyűjtő és feldolgozó rendszerekről szól, írunk az ismertebb szoftverekről is.

Az ipari Pc felépítése eltér a megszokottól. Az ipari környezetben több kellemetlen hatás érheti a számítógépet, amitől meg kell védenünk. Az esetleges javításokat a helyszínen, mostoha körülmények között gyorsan kell elvégezni. Az ipari Pc házak ezek figyelembe vételével lettek kialakítva. A masszív acéllemezből készült házak szellőzését, hűtését különös gonddal oldják meg, szűrt levegővel. Külön leszorítók rögzítik az átlagosnál nehezebb bővítő kártyákat, védve őket az esetleges vibrációtól. Az ipari Pc-t 19"-os kapcsolószekrénybe (rack), vezérlőpultba, műszerfalba, esetleg falra szerelik. Ennek megfelelően alakultak ki a különböző ház típusok. Egyesekbe monitort, újabban mono vagy színes LCD-t esetleg plazma kijelzőt építenek, nemegyszer érintő képernyővel. A panelba építhető házak előlapjára védett billentyűzetet is szerelnek. A lemezmeghajtókat zárható ajtók mögött helyezik el. A 2.-6. ábrán néhány jellegzetes ipari Pc házat láthatunk.

A Pc házba egy csak a párhuzamosan kapcsolt buszcsatlakozókat a tápfeszültség csatlakozót, és a bekapcsolt állapotot jelző LED diódákat tartalmazó un. passzív alaplapot építenek. Ezen a ház méretétől függően 4 - 20 ISA vagy újabban PCI busz található. Egyes nagyobb alaplapoknál több csoportban, melyek egy híd elemmel összekapcsolhatók. Így egy házban akár több Pc-t is elhelyezhetünk, ezek önállóan vagy egymás tartalékaiként dolgozhatnak. A hagyományos Pc-k alaplapját a tulajdonképpeni számítógépet egy normál AT kártyára építik, ezzel a gép javítása teljesítményének növelése egy kártyacserére egyszerűsödik. A processzor kártyákat teljes hosszúságú és un. félhosszú (185x122 mm.) kivitelben gyártják. Kihasználva technikai lehetőségeket a gyártók minél több funkciót igyekeznek a processzor kártyára integrálni. Az 1. ábrán látható teljes méretű kártyán egy Intel Pentium processzor dolgozik, maximum 192 MB ram elhelyezésére van lehetőség. A kártyán az enhanced IDE vezérlőn kívül amihez négy merevlemez és két floppi csatlakoztatható, még gyors SCSI-II vezérlő is található. A kártya két RS-232 soros és kétirányú párhuzamos porton túl külön PS/2 egércsatlakozót tartalmaz. Az ábrán látható kártya kétféle busszal rendelkezik. A PCI busz átbocsátó képessége 132 MB/s szemben az ISA busz 8 MB/s átviteli sebességével. A bővíthetőséget elősegítendő gyakran építenek a kártyákra PC/104 buszt. Bekapcsoláskor a kártya önellenőrző teszteket hajt végre, ezekről a kártyán lévő LED-ek tájékoztatnak. Megtalálható a kártyán az ipari Pc-k jellegzetessége a Watchdog timer, ez az áramkör figyeli a



1. ábra Egy erős processzor kártya.

2. ábra Vezérlőpultba, szekrénybe építhető ipari Pc ház színes LCD kijelzővel.

3. ábra Ipari Pc tokozott monitorral.

busz adatforgalmát és ha a beállított ideig (0.5-1008 sec) nem történik adatmozgás, azt hibaként értékelve újraindítja a rendszert, vagy megszakítást generál. Ez lehetőséget biztosít hibatűrő rendszerek kialakítására. Más hasonló processzorkártyákra sokszor építenek ROM lemezt, ami boot floppit emulálva gyors és biztos rendszerindítást tesz lehetővé. Gyakori, hogy a kártyán lévő soros portok egyike az ipari felhasználásnak jobban megfelelő RS-485 szabványú.

Amennyiben a rendszerben monitort is található meghajtására használhatunk normál VGA kártyát, vagy helytakarékosság miatt a processzor kártyára illeszthető PC/104 VGA modult. Speciális igényekre léteznek többféle megjelenítő akár egy idejű kezelésére is alkalmas videó kártyák is. A rendszerépítéshez védett monitorok, védett billentyűzetek, érintő képernyők is kaphatók. Jellegzetes alkatrészei az ipari Pc-nek a félvezetős tárolók. Egy normál AT kártyán egy, vagy két egyenként maximum 6 MB kapacitású RAM/ROM/FLASH lemez alakítható ki, ezek normál lemezmeghajtóként viselkednek. Kaphatók PCMCIA változatok is, ezek ára ma még elég magas. Elsősorban a gyors, biztos rendszer indításhoz, a szoftver védelméhez, nagysebességű adatgyűjtéshez használhatók. Egy gépbe legfeljebb négy félvezetős meghajtó építhető. A szervíz munkákhoz, rendszer bővítéshez, teszteléshez többféle buszbővítő, toldó kártya kapható.

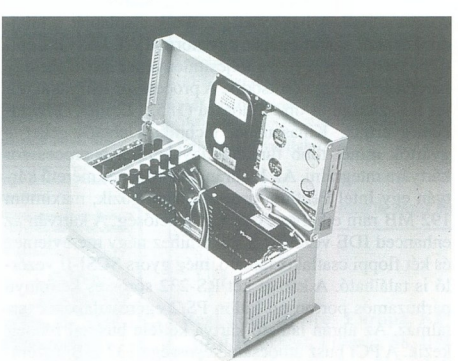
Mire használható a Pc az iparban?

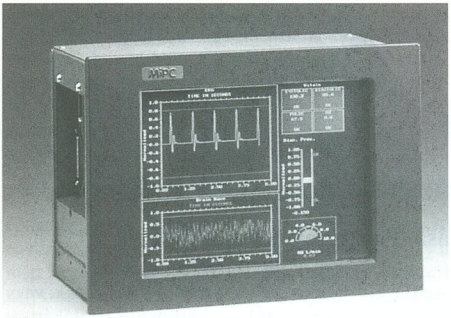
A legegyszerűbb alkalmazás ha a Pc-t inteligens mérőműszerként használjuk. Ilyenkor „csak" a bejövő adatok gyűjtéséről, feldolgozásáról, megjelenítéséről, naplózásról kell gondoskodnunk. Bonyolultabb a helyzet, ha a folyamatokat kézben is kell tartani. Egy Pc-s irányító rendszer kialakításakor alapvetően két módszer közül választhatunk. Az egyikben mindent a Pc végez, gyűjti, feldolgozza az adatokat és ezek alapján vezérli, irányítja a folyamatokat (DCS). A másik módszer szerint a rendszert „hagyományos" módon irányítják (például PLC-vel), a Pc csak felügyeli, naplózza az eseményeket (SCADA). Mindkét megoldásnak vannak előnyei, hívei. Természetesen a két rendszert kombinálni is lehet. Konkrét esetben a feladat, az előzmények az anyagi lehetőségek ismeretében választható ki az optimális megoldás.

A Pc-vel mérhetünk folyamatosan változó adatokat: hőmérséklet, súly, folyadékáram, nyomás stb. ha ezeket valamilyen az eredeti adatokkal arányos elektromos jellemzővé alakítjuk. Ugyanakkor érzékelhetünk kétállapotú jeleket, be vagy kikapcsolt állapot, valamilyen határérték túllépése. Ezeket az analóg és digitális bejövő jeleket Pc-be illesztett kártyákkal vagy soros vonalon elérhető eszközökkel alakíthatjuk a Pc számára feldolgozható adatokká, számokká. Ha vezérelni is akarunk, szintén a Pc-be illesztett kártyák ill. soros vonali eszközök segítségével tehetjük. Ki be kapcsolhatunk dolgokat ill. folyamatosan változó vezérlő jelet is szolgáltathatunk. A pc-be illeszthető analóg mérőkártyákat a maximálisan elérhető mintavételezési sebesség, a felbontás (a mérés pontossága), a mérhető csatornák száma alapján osztályozhatjuk. Egy átlagos kártya 30 KHz sebességgel, 12 bit felbontással mérhet 16 csatornát. A drágább kártyák elérhetik a 330 KHz mintavételezési sebességet, és a 16 bites febontást. A szokásos mérési tartomány 0 - 5 volt, de sok kártyán találhatunk erősítőt, amivel a mérési tartományt kapcsolókkal esetleg programból választhatjuk meg. A témáról bővebben olvashatunk a következő cikkben. A folyamatosan változó kimenő jelet előállító D/A kártyák különböző vezérlési feladatokhoz használhatók. Egy kártyára általában 2 - 16 csatornát építenek, melyek 0 - 5 volt, vagy 4 - 20 mA tartományban szolgáltatnak kimenőjelet 12 bites felbontásban. (Ez azt jelenti, hogy a kimenőjel 4096 féle értéket vehet fel.) A túlterheléstől általában biztosítéket védi a kimeneteket.

A digitális I/O kártyákat elektromos berendezések be-,

4. ábra Ipari Pc doboz hat kártya részére, falra szerelhető, szekrénybe építhető.





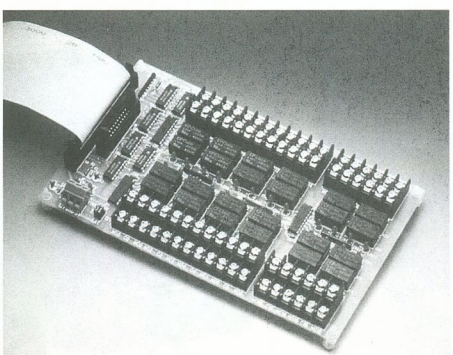
5. ábra Panelba építhető ipari Pc ház plazma kijelzővel négy félméretű kártya részére

és kikapcsolására, be-, vagy kikapcsolt állapot ellenőrzésére, külső eszközök vezérlésére használhatjuk. A be-, ill. kimeneti jel többnyire TTL szintű, Egy kártyán 8-tól 144-ig I/O vonal található /mindig nyolc egészszámú többszöröse/, sokszor egyéb áramkörök /pl. számláló időzítő/ társaságában. Működés és programozás szempontjából kétféle kártyát ismerünk: amelyik megszakítást generál és amelyik nem. A megszakítást generáló kártyák lehetőséget biztosítanak eseményvezérelt programok írására, háttérben történő monitorozásra anélkül, hogy a csatornákat folyamatosan le kellene kérdezni. Az egyes csatornákhoz /vagy csoportokhoz/ prioritási szinteket rendelhetünk, letilthatjuk őket. Több megszakítást kérő csatorna esetén megszakítás-maszkot haszálhatunk. Ez azt jelenti, hogy pl. nyolc csatorna esetén a maszk byte adott bitjéhez rendelt csatornáról engedélyezett a megszakítás ha a bit nulla, egyébként tiltott. A generált megszakítás általában IRQ 2 - 7. Programozásukról a januári számunkban írtunk.

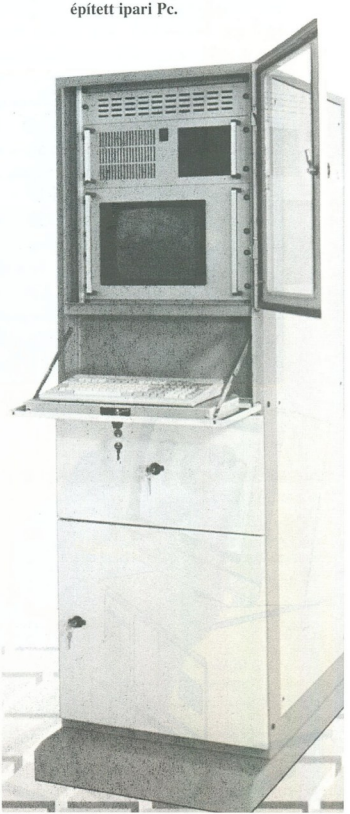
Nagyon fontos dolog, hogy ipari körülmények között nem szabad a mérendő jeleket közvetlenül a Pc-ben lévő kártyára csatlakoztatni! Az esetleges túláramok, villamos zavarjelek, zárlatok, villámcsapás tönkretehetik a gépet, sőt balesetet okozhatnak! Ennek elkerülésére leválasztó, jelformáló egységet kell közbeiktatni. Digitális bemeneteket optocsatolós leválasztással, digitális kimeneteket relés kimenettel védjük. Ezek külön panelra szerelve 16 vagy 24 csatornás kivitelben kaphatók. A Pc-hez szalagkábellal, a külvilág felé sorkapocssal csatlakoztathatók. Egy sínre szerelhető változat valószínűleg praktikusabb lenne. A fejlődés nagyon gyors, így lehet, hogy már ilyen is van.

Az analóg jelek leválasztása már költségeseb, bonyolultabb feladat, de itt is többféle termék közül választhatunk. Különböző panelra szerelt, vagy modulokból összeállítható esetleg egyéb funkciót (erősítő, multiplexer, azonos idejű több csatornás mintavevő) ellátó illesztő kapható. A szabványos 4-20 mA-es jelek fogadására olcsó alternatívaként szigetelt mérőkártya az optimális megoldás. Néhány csatornás analóg kimeneti kártyák is léteznek szigetelt változatban. Az analóg és digitális be és kimeneti kártyákon kívül számtalan egyébb feladatra készített Pc kártya kapható, külön említést érdemelnek az időzítő, robotokhoz szerszámgépekhez az egy és három tengelyes léptetőmotor vezérlő és a laboratóriumi műszerek illesztését megvalósító IEEE-488 kártya. A hordozható számítógépekhez laboratóriumi, vagy terepi adatgyűjtéshez megjelentek a PCMCA II. interfésszel készült adatgyűjtő modulok. Úgy néznek ki mint egy jól meghízott szapantartó ami egy félméteres pórázon sétáltatja a hitelkártyáját. Az ára mint minden újdonságnak, elég magas. Kaphatók notebook számítógépekhez készült külső adatgyűjtő egységek, ezek A/4 méretű dobozba épített egy vagy két adatgyűjtő kártyát tartalmaznak. A notebookhoz soros, vagy párhuzamos porton csatlakoznak, egyesek az un. „docking station" csatlakozón át közvetlenül a Pc buszhoz. Remélem a fentiekből látszik, hardver oldalról már nincs akadálya a Pc alkalmazásának ipari, vagy laboratóriumi adatgyűjtő rendszerként.

7. ábra 24 csatornás relés illesztő panel digitális kimenetekhez.



6. ábra 19"-os szekrénybe épített ipari Pc.



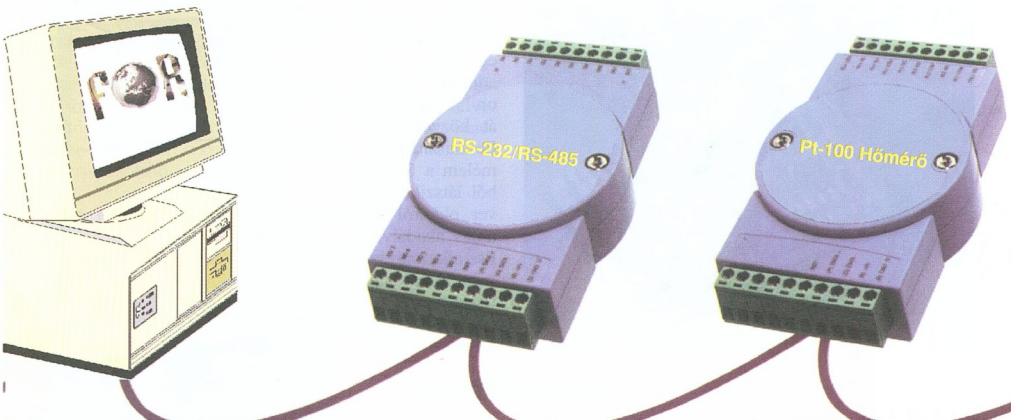
Lánc-lánc...

Soros vonali eszközök alkalmazása Pc-s irányítástechnikában

A Pc soros vonali csatlakozóját eleve arra tervezték, hogy a Pc külső eszközzel adatokat cseréljen. Kézenfekvő tehát, hogy ipari vagy laboratóriumi adatgyűjtésre, folyamat-irányításra is olyan eszközöket használjunk, melyek soros vonalon komunikálhatnak a Pc-vel. A soros vonali eszközök haszálatának előnye, hogy a számítógépet nem kell a zord ipari környezetben tartani. Így nem kell drága megoldásokkal gondoskodni a gép védelméről, egy normál Pc is megteszi az irodában. A Pc BIOS négy RS-232C szabványú soros adapter meglétét engedi, de a DOS csak kettőt kezel. Ahol ennél többre van szükség inteligens, saját processzorral rendelkező soros kártyák 8, külső adapterrel esetleg 32 soros vonal kezelését is lehetővé teszik, és ezekből akár négyet is elhelyezhetünk a gépünkben. Így ma egy Pc legfeljebb 128 soros vonallal rendelkezhet, kezelésükhöz speciális meghajtóprogram szükséges. Az RS-232 szabbványú soros vonal nem a legalkalmasabb az ipari környezetben végzett munkára. Alapvetően két eszköz összekapcsolására készült, a legnagyobb áthidalható távolság 15 méter, az átviteli sebesség nem túl nagy, és nem védett a túláramoktól sem. Ezen problémák megoldásárra vezették be az RS-485 szabványú soros adat átvitelt. Ez már 1200 méter áthidalására is alkalmas, és egy hálózatban maximálisan 32 meghajtó, 32 vevő lehet. Ez utóbbi lehetővé teszi ismétlők (repeater) alkalmazásával 256 modul egy vonalra kötését. Létezik szigetelt változatban is. Az RS-485 hálózat kialakítható két és négy vezetékes rendszerben is. A kábelezéshez olcsó csavart érpárt használhatunk.

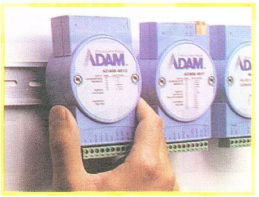
Az elinduláshoz két módszer közül választhatunk, vagy egy megfelelő RS-485 szabbványú illesztőkártyát teszünk a gépünkbe, vagy RS-232/RS-485 konverterrel csatlakozunk a meglévő soros portra. Az első megoldás olcsóbb, elegánsabb de a kártya kezelése programból eltérhet a normál soros port kezelésétől. A külső konverter általában lassabb, külön tápfeszültséget igényel, de megvédi, leválasztja gépünket a környezeti ártalmaktól, ha lehet ezt a megoldást válasszuk. Soros vonalon nagyon sokféle adatgyüjtő, vezérlő eszközzel létesíthetünk kapcsolatot, szinte gyártónként más-más módon. Egyszerre van jelen a szabványosítás iránti igény és a konkurenciaharc. Ebből aztán többféle kvázi szabbványként emlegetett protokol jött létre. A következőkben egy érdekes a Pc lelkivilágához közel álló megoldást mutatunk a soros vonali adatgyűjtésre, vezérlésre.

Az ADVANTECH cég kínálatában szereplő ADAM soros vonali adatgyűjtő család kis szappantartóra emlékeztető moduljaiból legfeljebb 256-ot köthetünk egy kétvezetékes RS-485 hálózatba 1200 méteren belül. A már elmondottak értelmében 32 modulonként, vagy nagyobb telepítési távolságnál ismétlőt kell alkalmazni. A modulok szerelősínre pattinthatók, vagy panelre, egymásra csavarozhatók. Az adatforgalom legnagyobb sebessége 19.200 Baud, ami nem túl sok, de mivel a komunikációhoz rövid néhány karakteres üzeneteket használunk bőven megfelel. A modulok energia ellátását szabályozatlan 10-30 voltos egyenfeszültséggel biztosíthatjuk, fogyasztásuk 1-2W. A modulok védettek a

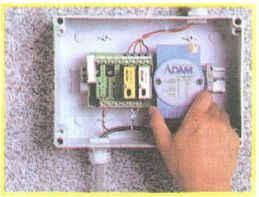


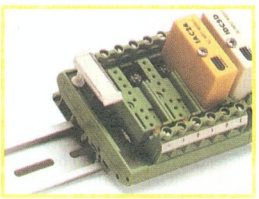
túlterheléstől, a túlfeszültségtől 500V-ig. A jelenlegi választékban öt féle analóg bemeneti modul található. A mérőmodulok 16-bites felbontással dolgoznak, különböző méréstartományokra programozhatók a Pc segítségével. A konverziós idő kb. 0,1 sec. ami elég hosszú, behatárolja a modulok használhatóságát, valamit segít, hogy egyidejű mintavételezésre is utasíthatók. Külön modul kapható a hőelemes ill. az platinaellenállásos hőmérsékletméréshez. Ez egyik modulba multiplexert építettek, igy nyolc csatorna jelét mérheti, persze csak egymás után. Egy másik modul kijelzőt tartalmaz, így a mért értékről, vagy a Pc-ről jött üzenettel a mérés helyszínén tájékoztat. Kapható analóg kimeneti modul, ami a szabványos tartományokban 12-bit felbontással szolgáltat folyamatosan változó vezérlő jelet.

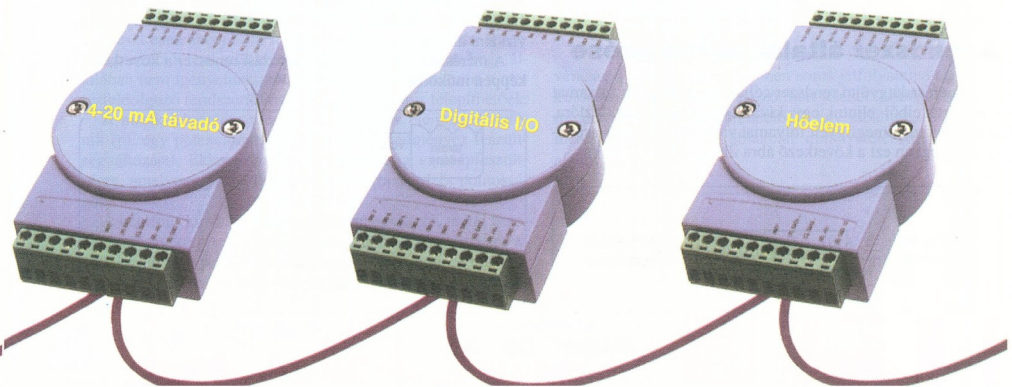
A digitális jelek fogadására kétféle modulból választhatunk, köztük a védettségben van különbség, ill. az egyik kimeneteket is tartalmaz. A digitális kapcsoló jeleket az előbb említett modullal és külön relés leválasztással használhatjuk, de ahol a kapcsolandó áram nem haladja meg a fél Ampert használhatjuk a négy csatornás relés kimeneti modult. Újdonság a kijelzős számláló és frekvenciamérő modul. A modulok az iparban szokásos mérési, vezérlési feladatok zömében használhatók. Segítségükkel viszonylag olcsó, könnyen bővíthető, változtatható mérő, vezérlő hálózat építhető. Alkalmazása különösen ajánlható a gyakran változó technológiával dolgozó, vagy nagy területen elhelyezkedő üzemekben. Ajánlható azoknak akik nem egy komplett rendszert akarnak vásárolni, hanem fokozatosan térnének rá, vagy át a Pc-s adatgyűjtésre, vezérlésre. Telepítés után a modulokat programozni kell, erre a legalkalmasabb a hozzájuk adott DOS utiliti, de mivel csak néhány karakter kiküldéséről van szó a soros vonalon, ez más programból is könnyen megoldható. A programozással állítjuk be a modul címét, méréstartományát, és hogy az adatokat milyen formában szolgáltassa. A modul ha szükséges többször átprog-ramozható. Használat közben a komunikáció néhány karakteres ASCII parancsokkal, illetve az erre adott válaszok, szintén karaktersorozatok, vissza olvasásával történik.



Az RS-485 soros vonalra felfűzhető adatgyűjtő modulok. Egy portra 1200 méteren belül max. 256 modul kapcsolható különböző típusok vegyesen, olcsó csavart érpáras kábelezéssel. A modulok a mérés helyszínén sínre, vagy dobozba szerelhetők. Szinte minden mérési, vezérlési feladatra alkalmas, bővít-hető rendszer építhető belőle.







Digitális mérésadatgyűjtő és feldolgozó rendszerek

Ipari folyamatok szabályozásának és felügyeletének elterjedő módszere a számítógépes mérésadatgyűjtés. A módszer lényege, hogy a megfigyelt analóg jelet digitalizálják, majd a digitális jel feldolgozását a számítógép vagy egy külön erre a célra tervezett mikroprocesszor végzi. A klasszikus, analóg mérésadatgyűjtő rendszerekkel szembeni előny abban jelentkezik, hogy a digitalizálást, illetve a jelfeldolgozást végző egység (célprocesszor, vagy PC) sokoldalúsága miatt, a mért értékekből, gyakorlatilag akármilyen numerikus eredmény (vagy legalábbis ezek széles skálája) gyorsan és pontosan számítható. Ily módon, bonyolult rendszerek mérési adataiból, gyorsan lehet a rendszer alapvető jellemzőit kinyerni, amelyek a működés szempontjából alapvető fontosságúak. Sőt a kritikus paraméterek kinyerése és ezeknek bizonyos optimális referenciaértékekkel való összehasonlítása után, a mérésadatgyűjtő rendszer vezérlő jeleket állíthat elő, a megfigyelt folyamat szabályozása érdekében.

Akár megfigyelésről vagy szabályozásról legyen szó, a digitális rendszerek legfőbb jellemzője, hogy a mért adatok értékelését nem korlátozza a feldolgozó algoritmusok komplexitása. Ez új lehetőséget jelent olyan alkalmazásokban, ahol az igényelt számítási bonyolultság miatt eddig automatikus mérésadatgyűjtés és feldolgozás még nem valósulhatott meg.

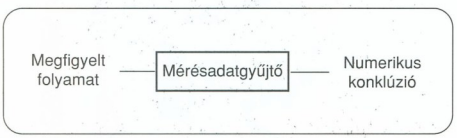
A cikk a következő gondolatmenet alapján mutatja be a mérésadatgyűjtő és feldolgozó rendszereket:

- Először a rendszer alkotóelemei kerülnek röviden összefoglalásra, amit egy alkalmazási példa követ a mérésadatgyűjtés és feldolgozás konkrét demonstrálására.
- Ezután a rendszer elemeinek részletesebb ismertetése következik, néhány megvalósítási elvet is bemutatván. A cikket a mérésadatgyűjtő rendszerek korlátainak, illetve kritikus paramétereinek az áttekintése zárja, számot adván a fejlődési irányokról.

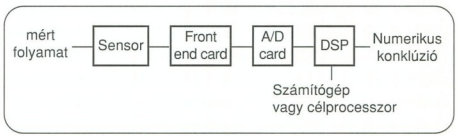
Az egyes elemek működését az RTD termékein illusztráljuk, amelyek lehetőséget nyújtanak egy sokoldalú adatgyűjtő és monitorozó rendszer felépítéséhez.

A rendszer általános felépítése

A mérésadatgyűjtő rendszer célja egy megfigyelt folyamat mért értékeiből eljutni egy kívánt numerikus konklúzióra, amely felfedi a megfigyelt folyamat valamely lényeges aspektusát, ahogyan ezt a következő ábra illusztrálja:



A megfigyelt folyamat leképezése a mérés útján kiszámolni kívánt jellemzőbe, a következő lépcsőfokokban történik.



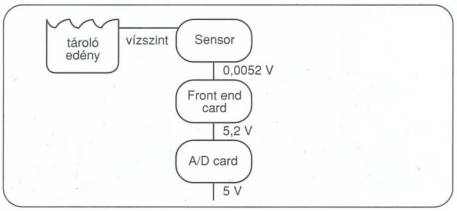
A rendszer egyes elemeinek működése a következőképpen foglalható össze:

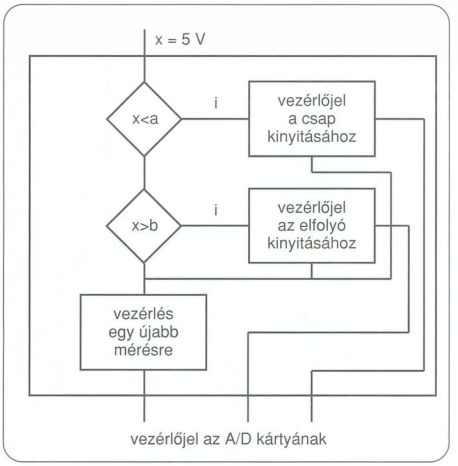
- A sensor alakítja át a ténylegesen mérni kívánt jelenséget (ami lehet hőmérséklet, elmozdulás, nyomás, súly… stb.) elektromos feszültséggé.
- A Front End kártya a keletkezett feszültséget valamilyen szabványos értéktartományba transzformálja, amelyen az őt követő kártyák működnek. Így a jelet illesztette az univerzális jelfeldolgozó egységek számára.
- Az A/D kártya a folytonos jelből időben és amplitúdóban diszkrét sorozatot állít elő (mintavételezés és kvantálás).
- A DSP egység végzi, a mostmár digitális formában megjelenő mérési sorozaton, a szükséges számításokat a mérés kiértékeléséhez, amely a kívánt paraméter kinyeréséhez vezet.

Példa mérésadatgyűjtő rendszer alkalmazására

Az előbb áttekintett folyamatot egy példával illusztrálván, tételezzük fel, hogy a feladat egy tárolóedény vízszintjének a szabályozása. Ha a vízszint egy kritikus érték *(a)* alatt van akkor egy töltőcsap megnyitása a cél, amint a vízszint eléri a kívánt *a* értéket, akkor a töltőcsapot el kell zárni. Ha a vízszint egy kritikus *(b)* értéket halad meg akkor viszont egy elfolyó vezeték megnyitása a feladat. Tételezzük fel, hogy *x* jelöli a vízszint aktuális értékét.

A mérésadatgyűjtési és szabályozási rendszer a következőképpen működik:





Amint a fenti folyamatábrából kiderült, a DSP feladata jelen esetben a mért jel triggerelése amely alapján beavatkozó jeleket szolgáltat a szabályozott rendszernek (a töltő csap kinyitásáról, vagy a lefolyó megnyitásáról, illetve a mérés folytatásáról gondoskodik. Természetesen ennél sokkal bonyolultabb funkciókra is képes, például digitális szűrés vagy Fourier-transzformáció megvalósítására a bejövő jelsorozaton. Így a mérési sorozat meglehetősen nagy komplexitású kiértékelése is lehetővé válik.

A mérésadatgyűjtő rendszer elemeinek részletes leírása

A mérésátalakító és érzékelő (sensor):

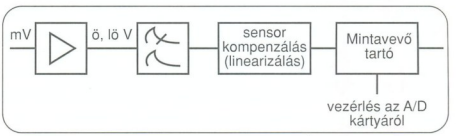
Az érzékelő és mérésátalakító feladata, a nevéből következően, hogy a mérni kívánt mennyiséggel (elmozdulás, súly, fordulatszám… stb.) arányos feszültséget bocsásson ki. Megvalósítása történhet aktív elemekkel (pl. kristály, tekercs… stb.), amelyek a mérendő mennyiséget villamos jelekké transzformálják valamilyen fizikai alapelv alapján. Ezek hátránya, hogy többnyire igen kis feszültséget adnak (ezért az őket követő erősítőnek lehetőleg kis zajúnak kell lennie), másrészt esetenként erősen nem lineáris működésűek, amely kompenzációt tesz szükségessé. Az előnyük viszont, hogy általában nem igényelnek precíziós tápfeszültséget, ezért a mérésfeldolgozó rendszertől nagyobb távolságra is telepíthetőek.

A másik lehetséges megvalósítás passzív elemekkel történik (pl. egy potenciométer elmozdulása eredményez feszültségváltozást). Ekkor mindenképpen szükség van tápfeszültségre, amelynek pontossága meghatározza a mérés pontosságát.

A két típus közötti választás gyakran egyszerű, mivel bizonyos mérésekre csak egyféle elven működő szenzor létezik, más esetekben pedig a körülmények (mérendő tartomány, környezeti tényezők, hőmérséklet stb.) határolják be a lehetőségeket.

A Front End kártya:

A kártya felépítését a következő blokkdiagram szemlélteti:

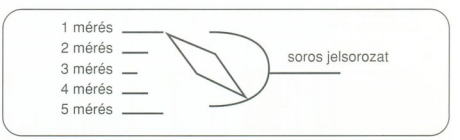


A jelerősítésre azért van szükség, hogy az aktív átalakítók mV nagyságrendű kimenőjelét egy szabványos tartományba transzformáljuk, amely általában a (0,5, (0,10), (–5,5) vagy a (–10,10) V-os intervallum lehet. Az erősítőt követő aluláteresztő szűrőre azért van szükség, hogy a bejövő jelet sávkorlátozza, hiszen a mintavételi tétel értelmében csak sávkorlátozott jelet tudunk egyértelműen rekonstruálni a mintáiból. Az A/D kártya mintavételezési frekvenciájától függően az aluláteresztő szűrő határfrekvenciája széles tartományban (20 Hz–500 kHz) helyezkedhet el. A kompenzálásra, a mérésátalakító esetleges nem lineáris karakterisztikájának linearizálása miatt van szükség. Mivel ez a rész erősen szenzorfüggő, ezért a Front End kártya maga mindig adott típusú mérésátalakítókhoz készül. Az RTD által gyártott TS16 vagy TMX32 például hőmérséklet-érzékelőkhöz kapcsolódó Front End kártyák, amelyek multiplexálási feladatokat is elvégeznek (lásd a következő bekezdést).

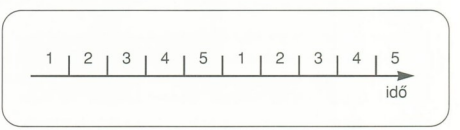
A multiplexer:

A multiplexerek lehetnek a Front End kártyákon is, illetve napjainkban már majdnem minden A/D kártyán is vannak, ezért itt mint csak funkciót részletezzük.

Multiplexelésre azért van szükség, mert az A/D konverter drága, viszont mintavételezést vezérlő jele elég nagy frekvenciás lehet ahhoz, hogy egyszerre több mérést is ki tudjon szolgálni A szimultán módon lefolytatott mérésekre, mint csatornákra hivatkozunk a továbbiakban. A multiplexer működését jól szemlélteti a következő ábra:



amelyben az A/D kártya maximális mintavételezési frekvenciával forgó kapcsoló időben soros jelfolyammá alakítja a különböző csatornákról érkező mért jeleket. Látható, hogy az minden méréshez hozzárendelődik egy időablak, és pl. az 1 mérés minden ötödik ablakban mintavételeződik (lásd az alábbi ábrát).

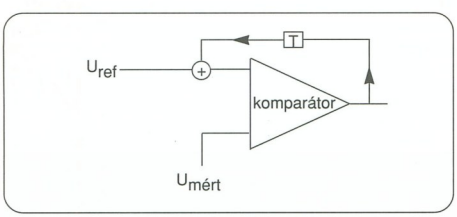


Azt, hogy a multiplex hány csatorna jelét tudja időben sorossá tenni, a kártya maximális mintavételi frekvenciájának, illetve a mérendő folyamatok sávkorlátaiból adódó, szükséges mintavételi frekvenciák aránya határozza meg. Pl. egy 500 kHz max. mintavételi frekvenciájú kártya 20 db 25 kHz-es mintavételi frekvenciájú mérést tud lekezelni. A multiplexerek kaszkádba kapcsolhatók, így egy 16 csatornás A/D kártya (amelyen a 16 csatorna is multiplexelt) elé kötve egy 32 csatornás multiplexer Front End kártyát, 16*32 azaz 512 csatorna jelét mérhetjük.

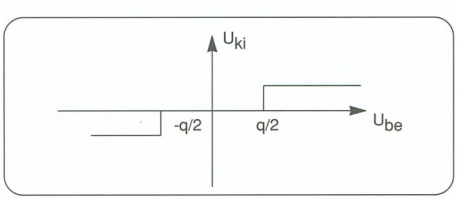
Az A/D kártya:

Az A/D kártyára az időbeli multiplexálás után még mindig folytonos értékkészletű feszültség minták érkeznek, amelynek értékeit diszkretizálni kell, hiszen bináris sorozatokkal csak egy diszkrét jelkészlet elemeit lehet kódolni. Az érkező mért minták bináris sorozatokba való leképezését kvantálásnak hívják. A kvantálás egyik legfontosabb paramétere a diszkretizált feszültségértékekhez rendelt bináris kódszavak hossza, hiszen ez determinálja milyen pontos (finom) a kvantálás, mennyi információt vesztettünk az eredeti mérésből. Mérésadatgyűjtő rendszerekben általában 8-tól 24 bitesig terjedő, lineáris kvantálást alkalmaznak.

A forgalomban levő A/D kártyákon számos különböző kvántálási elv került megvalósításra. Az egyik legnépszerűbb módszer a szukcesszív approximáció (folyamatos megközelítés) elvén alakul, amelyet a következő ábra szemléltet.



A komparátor karakterisztikája a következő:



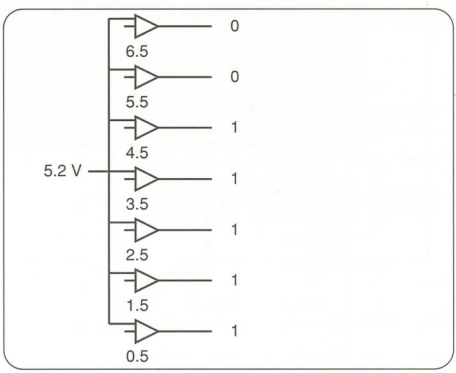
ahol q jelöli a kvantumlépcső nagyságát. Az így kapott diszkretizáló állapotegyenlete a következő:

$$U_{ref}(k+1)=U_{ref}(k)+sgn\,(U_{ref}(k)-U_{mért})$$

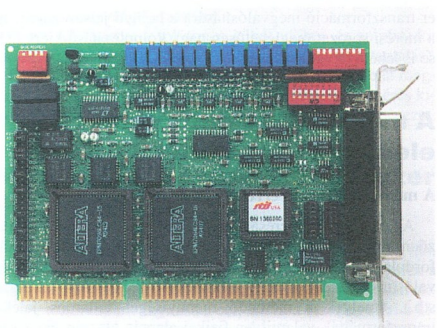
Könnyű belátni, hogy $U_{ref}(k)$ ahol a diszkrét feszültségértékhez fog tartani, aminek a távolsága legkisebb lesz $U_{mért}$ től.

Ezután az eljárás után egy egyszerű táblázatból kiolvasható a diszkretizált mintához tartozó bináris sorozat. Az eljárás hátránya, hogy esetleg sok ideig kell várni a konvergenciához. Előnye viszont az egyszerű hardverfelépítés. Ezen az elven alapul az RTD legtöbb A/D kártyája.

Egy másik, gyorsabb módja az A/D átalakításnak az úgynevezett „flash converter"-es technika. Az analóg mintához tartozó kvantált bináris sorozatot ilyenkor egy komparátor lánc kimenetén kapjuk, amit a következő ábra illusztrál:

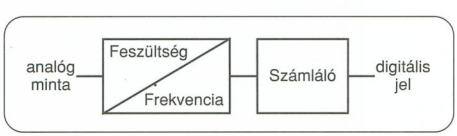


Ez a típusú analóg digitális átalakító nagyon gyorsan működik, azonban a komparátorok száma meglehetősen nagy (megegyezik a kvantálási szintek számával).

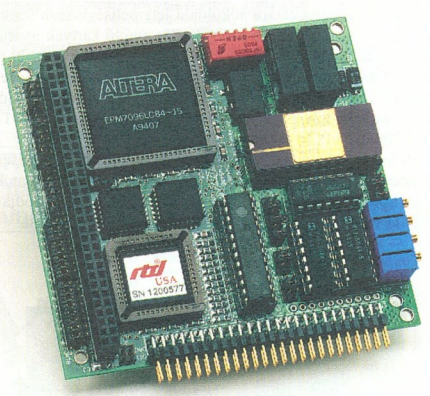


Gyors AD kártya

Egy másik analóg digitális átalakító a frekvenciamérés (digitális rendszerekben egyszerűen pulzusszámlálás) elvén alapszik. Ekkor a mért feszültséggel arányos frekvenciává alakítjuk a jelet, majd a frekvenciát számoljuk, amely csak egész (kvantált) értéket adhat, az alábbi ábra szerinti elrendezésben.



Ennek előnye a nagy pontosság, hátránya viszont, hogy a számlálás miatt (egy analóg mintát soros jelfolyammá alakítottunk) hosszú ideig tart. A fenti elv gyakorlati megvalósítása az RTD VF910-es A/D átalakítója.



AD kártya, PC/104 méretben

Létezik még a Dual Slope integráló konverter, ahol a bemeneti jel egy kondenzátort tölt, majd azt állandó árammal kisütve mérjük a kisülés idejét. Ebből következtethetünk a bemeneti jel nagyságára. Ennek a módszernek nagy előnye, hogy az elvből adódóan kiszűri a nagy frekvenciájú zajt, hátránya viszont a meglehetős lassúság (néhány tíz Hz-es mintavételi frekvencia). Az RTD ADA520-as kártyája működik integráló konverterrel.

A digitális jelprocesszió (DSP):

Ez a funkció a modern mérásadatgyűjtő rendszerek „lelke", mert lehetővé teszi a mért adatoknak, szinte tetszőleges bonyolultságú számítások alapján történő, feldolgozását. A jelfeldolgozás célja általában valamilyen, a mért adatokból számolandó, a megfigyelt folyamat szempontjából kritikus paraméter meghatározása. Ezen paraméter birtokában a mérésadatgyűjtő rendszer visszahathat a megfigyelt folyamatra, megfelelő vezérlőjelek formálásával (szabályozási rendszer esetén), vagy riasztást adhat a beavatkozásra (monitorrendszerek esetén).

A mért adatok kiértékelése elvileg tetszőleges számításokkal történhet, amely kivitelezhető egyszerű aritmetikai műveleteket (pl. összeadás és szorzás) tartalmazó algoritmusok alapján. Mivel az ilyen algoritmusok osztálya igen nagy, ezért a DSP gyakorlatilag majdnem bármilyen kiértékelési feladat megoldására bevethető. A valóságban azonban bizonyos feladatok (szűrés, konvolúció, FFT–IFFT) az esetek nagy többségében elegendőek a mérési adatok kiértékeléséhez, így ezekre speciális (a komplexitás szempontjából optimális) algoritmusok állnak rendelkezésre.

A DSP kivitelezésére két alternatíva kívánkozik:

- Digitális jelfeldolgozás DSP kártyával
- Digitális jelfeldolgozás PC-n DSP szoftverrel

A továbbiakban ezeket vesszük sorra.

Digitális jelfeldolgozás DSP kártyával

A DSP funkció implementálása a legtöbb esetben nem igényel külön kártyát, hanem az A/D kártyán van integrálva.

A DSP kialakítása hardver úton lehetővé teszi, hogy a célfeladatokat (FFT–IFFT, szűrés… stb.) egyszerűen lehessen programozni, illetve ezek végrehajtása optimalizálva legyen. A célprocesszor sokkal gyorsabban végzi el a műveleteket, mint egy általános processzorra írt program. Pl. az RTD DSP200-as kártyája 5 µsec muntavételezési időközönkénti mintákon képes szűréseket, illetve egyéb algoritmusokat végrehajtani.

A DSP kártyás jelfeldolgozás előnye, hogy az ezt követő PC processzorát mentesíti egy sor komplex feladat elvégzésétől, ezért az pl. teljes mértékben a grafikus feladatokra fordíthatja a feldolgozási kapacitását.

A DSP együttműködhet a kártyán implementált memóriával vagy pedig a PC memóriájával.

Digitális jelfeldolgozás PC-n, DSP szoftverrel

Ebben az esetben a jelfeldolgozási feladatot egy alkalmas program valósítja meg (pl. az RTD SIGNAL*VIEW vagy SIGNAL*MATH), amely a jelfeldolgozási feladatok elvégzésén kívül széles grafikai lehetőségekkel rendelkezik a mért minták megjelenítésére, illetve vezérlési feladatokra is alkalmas lehet szabályozási rendszerek esetén.

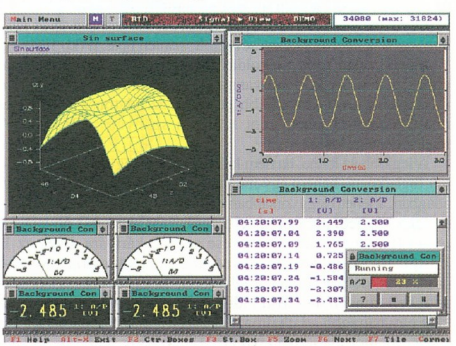
Egy pár tipikus funkció a SIGNAL*VIEW mérésadatgyűjtő és monitorprogram esetében:

- Gyors méréssorozatok (minden más funkció letiltva) 200 kHz-es sebességű mintákon is
- On-line mérés szimultán jelfeldolgozással és grafikával
- Egyszeri mérések lassan változó jelek analíziséhez
- Analóg, digitális triggerfunkciók
- Real time aritmetikai műveletek a mintákon (egyszerű matematika)
- Analóg, digitális méterek
- 2 és 3 dimenziós grafika

Néhány főbb jellemzője a SIGNAL*MATH jelfeldolgozó programnak:

- FFT–IFFT (spektrumanalízis)
- Szűrés és szűrőtervezés (FIR, IIR)
- Statisztikai átlagok és lineáris regresszió
- Wigner transzformáció, spektrogram

A PC-n történő jelfeldolgozás előnye, hogy a szoftverúton realizált jelfeldolgozási algoritmusok sokrétűbbek.



Signal VIEW

A mintákon végzendő transzformációk flexibilisebbek (pl. adaptív jelfeldolgozás). Jelentős hátrány viszont, hogy a PC processzorának kell az összes feladatot ellátnia (nemcsak jelfeldolgozási algoritmus véghezvitelét, hanem adatmozgatást... stb.) ami jelentősen csökkentheti a feldolgozási sebességet. Pl. egy 30–40 MHz-s órajelű PC gyakorlatilag 1 MHz körüli minták esetén már a minták tárolására sem alkalmas (a korlátozó tényezők áttekintése a következő fejezetben található).

A digitális mérésadatgyűjtő rendszerek kritikus paraméterei és jelenlegi korlátai

A digitális mérésadatgyűjtő rendszerek jelfeldolgozási sebességének három főbb korláta van:

- a multiplexer sebessége
- szoftveres feldolgozás esetén a PC processzorának sokoldalú leterheltsége
- az adatbusz sebessége.

A multiplexer sebessége (néha ez kihasználatlanul hagyja az A/D kártya magasabb mintavételi frekvenciáját). Ennek tipikus korláta a jelenlegi kártyák esetén 2–5 sec-os mintavételezési időköz. A másik lényeges korlátot, a PC-s szoftverrel történő digitális jelfeldolgozás esetén, a processzor sokoldalú leterheltsége jelenti. Ennek illusztrálására, vizsgáljuk meg milyen feladatokat kell elvégezni, ha csak a mért adat képernyőn való megjelenítése a cél. Ekkor először az adat beolvasására kerül sor, majd a mért adat alapján átszámolás történik a képernyő-koordinátákra, végül pedig egy pontot kell kitenni a képernyőre. Ez legalább két darab memóriaműveletet, illetve a koordináták meghatározásához, aritmetikai számításokat igényel. A PORT műveleteket is belekalkulálva, néhány száz MHz-nél magasabb feldolgozási sebesség nem érhető el a jelenlegi processzorok mellett.

A másik jelentős korlátozó tényező az adatbusz (ISA) sebessége, amelyeken az adatokon kívül parancsoknak is áramlania kell. Ennek jelenlegi sebessége 8 MHz, ami az adatáramlást, illetve jelfeldolgozási sebességet szintén 1 MHz környékére korlátozza.

Problémát jelent, ha a szoftver esetében minden egyes mintavételnél el kell dönteni, hogy jött-e minta (rendelkezésre álle már), és ha igen, akkor elvenni a kártyától majd eltárolni a memóriába. Ez a folyamat igen hosszú időt vehet igénybe, és ez idő alatt a gép semmi mással nem tud foglalkozni.

Egyik megoldás az interruptos működés, itt azonban az az idő, amit csak az interrupt kezelésével tölt, szintén hosszú (kb 100 µs).

Másik, talán jobb megoldás a DMA használata, azonban ennek is komoly korlátai vannak. 64 kB-nál több adatot nem tud kezelni, és a korszerűbb gépeken az adatbusz sebessége miatt lassabb is lehet, mint a tisztán szoftveres (REPINSW utasítás) megoldás.

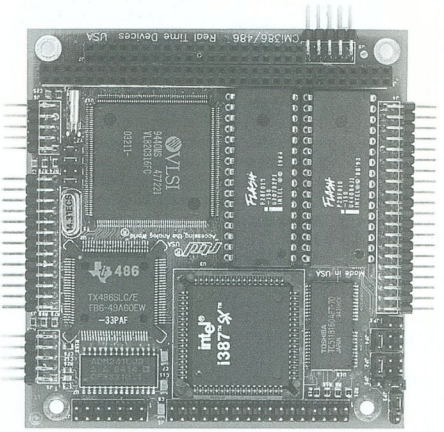
A legjobb az lenne, ha nem kellene minden egyes mintával külön foglalkozni, hanem egy meghatározott számú mérés után egy adagban, a lehető leggyorsabban elvenni és tárolni az adatokat.

Igen elterjedt módszer a feldolgozozási kapacitás növelésére, a FIFO memóriák elhelyezése az A/D kártyára. Ezek az 1 kB-tól 16 kB-ig terjedő memóriák „megkönnyítik" a processzorok dolgát, olyan értelemben, hogy a processzornak nem kell minden mintával individuálisan foglalkoznia, hanem elég egy nagyobb adathalmazt egyszerre mozgatni a kártya és a PC memóriája között (RTD DataMaster kártyák, ADA 3110, 3300, DM5408).

A mérésadatgyűjtő rendszerek fejlődési irányai

Az egyik alapvető fejlődési irány a jelfeldolgozási sebesség növelésére összpontosul. Mivel PC-s feldolgozás esetén ennek főleg a processzor sokoldalú leterheltsége szab határt, ez főleg „okosabb", több szolgáltatást nyújtó kártyák gyártását jelenti. Ennek az irányzatnak az egyik törekvése, hogy egyre több algoritmus kerüljön hardver megvalósításra, felszabadítván a processzort az adatfeldolgozási műveletek alól. Ezért egyre több esetben előzi meg DSP kártya (pl. RTD DSP200-as) a PC-s feldolgozást.

A másik irányvonal a rendszer méretének a csökkentésére koncentrál, hiszen így az alkalmazáshoz (valódi folyamatokhoz) közeli adatgyűjtés válik lehetővé. Ennek egyik módja a PC104-es kis méretű (mint egy söralátét) PC alkalmazása.

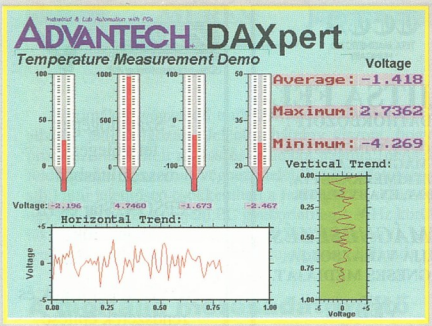


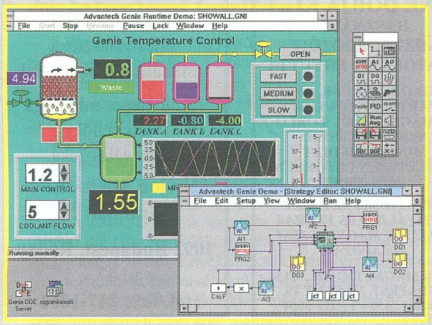
486-os alaplap, söralátét méretben

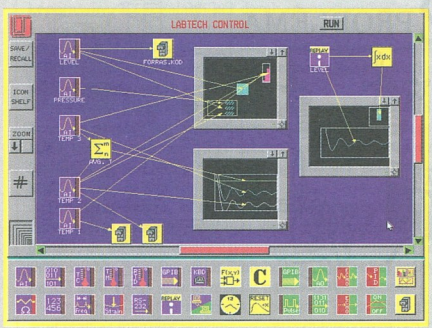
A fenti képen látható kártya igazi szenzáció, TI 486 SLC processzor dolgozik rajta 33 MHz sebességgel. Intel 387Sx koprocesszorral kiegészítve. Elfér rajta 4 MB RAM és 2 MB ROM vagy 1 MB Flash lemez. Természetesen (!) IDE merevlemez és floppi vezérlőt két soros és egy párhuzamos portot, billentyű és egércsatlakozót is találunk a söralátét méretű kártyán.

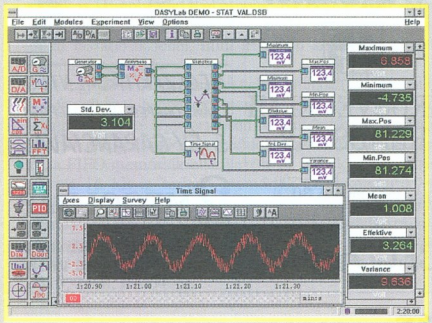
Januári számunk után sokan érdeklődtek a PC/104 modulok ára után. Az ár sok mindentől függhet, ezért nem szívesen írunk róla, valódi újdonságról lévén szó most kivételt teszünk. A fenti kártya ára a gyártó prospektusában kiépítettségtől, a mennyiségtől függően 580 $-tól indul.

A fejlesztések harmadik területe, az új jelfeldolgozási algoritmusok kutatása. Ez az irányvonal az adatfeldolgozás számítási eljárásainak a tökéletesítésére fókuszál. Jelentős kutatások folynak az adaptív jelfeldolgozás algoritmusainak, valamint a neurális hálózatok, illetve fuzzy logikai rendszerek alkalmazásának témakörében.









Adatgyűjtő, folyamatirányító programok

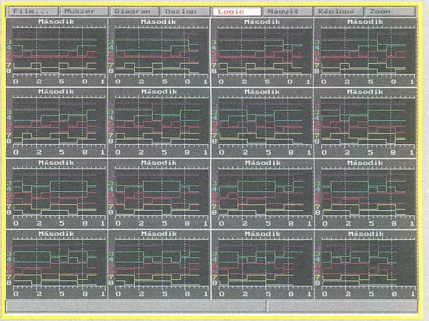
A Pc-s folyamatirányítás gyenge pontja a működtető szoftver. A feladat nehézségét a hardver elemek sokfélesége, gyártónként különböző programozhatósága adja. Ezért, termékeiket támogatandó, a kereskedelmi szoftverek többségét a hardver gyártók készítik, forgalmazzák. Ezen szoftverek előnye a viszonylagos olcsóság, hátrányuk, hogy más gyártók termékeit nem támogatják. Az irányítástechnikai szoftverek készítésének másik nehézségét a megfelelő operációs rendszer hiánya adja. Sem a DOS sem a Windows nem igazán alkalmas komolyabb valós idejű rendszer kialakításához. A nagyobb rendszereket vagy valamilyen gyártóspecifikus operációs rendszer, vagy Qnix, újabban OS/2 alá írják. A hardvergyártók szoftvereire az előző cikkben említett RTD termékek mellet jó példa az Advantech DOS alatt futó az első képen látható DAXpert programja. A program láthatóan Borland C-ben íródott a Turbovision felhasználásával. A konfiguráláshoz a karakteres képernyőn kell táblázatokat kitöltenünk. A konfigurálás, a beállított értékek mentése után a program már futtatható is. Támogatja a cég termékeinek nagy részét, adatgyűjtésre, vezérlésre egyaránt alkalmas, mint a karakteres, mint a grafikus megjelenítést tud. Féláron külön csak futtató változat is kapható. A cég másik terméke a második képen látható Windows alatt futó Genie. Felépítésében már a komolyabb vezérlő programokat idézi. Az alkalmazás felépítése a logikai egységeket jelképező ikonok egérrel történő kiválasztása, a megfelelő helyre vontatása után a logikai kapcsolatokat ezek összekötésével alakítjuk ki. Az egyes logikai blokkokat az ikonokra kétszer rákattintva előugró táblázatok segítségével konfigurálhatjuk. A képernyőt jelképező ikonra kattintva egy másik szerkesztő képernyőt kapunk, ahol a program megjelenését tervezhetjük meg. Élő diagramokat, számokat, szövegeket jeleníthetünk meg a mért adatok függvényében. Egérrel vezérelhető kezelő és beállító szerveket helyezhetünk el a képernyőn, mögéjük tetszés szerinti bittérképek tölthetők. Az alkalmazás mentése után, egy külön modullal futtatható. Külön érdekesség a DDE támogatás így adataink egy másik alkalmazásba folyamatosan átvihetők (pl élő grafikon az Excelben), vagy más programok adatait átvehetjük, feldolgozhatjuk, megjeleníthetjük. Olcsó, aranyos kis program, de szörnyű lassú. Elsősorban kezdőknek, vagy nem túl gyorsan változó események kezelésére, figyelésére ajánljuk. A független fejlesztők termékei közül talán a legsikeresebbek a Labtech cég programjai a Notebook és a Control (harmadik kép). A két program szinte azonos, de míg az előbbi csak adatgyűjtésre, megjelenítésre használható, az utóbbival vezérelhetünk is. Mindkét programnak létezik DOS és Windows verziója. Mindkét prog-ram különböző korlátozásokkal több teljesítmény fokozatban kapható, Nagyon sok hardvergyártó termékét támogatja, egynél több cégét külön pénzért. Az utolsó képen a Dasytech Windowsos programja a DASYlab látható. Mérésadat-gyűjtésre, megjelenítésre való. Szép kidolgozása mellett kiemelkedő a sebessége (a Windows programok között).

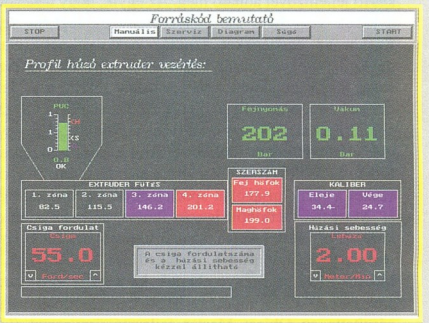
Adatgyűjtő, programok házilag.

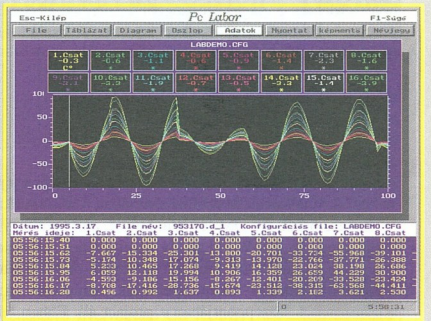
Mielőtt belefogunk pontosan tisztázzuk az igényeinket, és lehetőségeinket. Egy nagy méretű, nagy biztonságot nyújtó irányítási rendszer kifejlesztése combos feladat, hagyjuk ezt a profikra. Amit bátran megpróbálhatunk, egy inteligens műszer, egy adatgyűjtő, megjelenítő program némi adatfeldolgozással, naplózással, alarm (riasztó) funkciókkal. A fejlesztést mindenképpen DOS alatt kezdjük az egyértelmű portkezelés, megszakításkezelés, és az elérhető sebesség miatt. A program építése során törekedjünk a moduláris felépítésre, lehetőleg külön modul végezze az adatgyűjtést, a feldolgozást, a megjelenítést, és a naplózást. A programindításnál amit lehet ellenőrizzünk le, a start adatokat írjuk ki a lemezre, a program lezárásakor tegyük ugyanezt, utólagos hibakeresésnél jól jön. A mérőkártyákhoz, a gyártó a leíráson túl minden esetben mellékel C, és Basic nyelvű példaprogramokat, néha lefordított meghajtókat "obj" file-okat. Ha van ilyen, használjuk, mert ezek alaposan tesztelt kipróbált algoritmusok. Az adatgyűjtést végezhetjük egy egyszerű ciklussal is, de célszerűbb egy szoftver megszakítást írni, így függetleníthetjük a futás sebességét a számítógép sebességétől. A befektetett munka megtérül, ha az egyes jellemzőket nem azonos időközönként kell mérni. Egyes kártyák hardver megszakítást generálnak, ezt feltétlenül érdemes kihasználni, futásidőt takarítunk meg, kvázi multitaszkos működést, háttér monitorozást valósíthatunk meg. A kártyák portcímeit a csatornák számát, az egyes csatornákra vonatkozó adatokat (erősítés, név, mértékegység...) ne a programban tároljuk, olvassuk be egy szöveg file-ből. Így a kisebb változásokhoz nem kell újrafordítani a programot, egy editorral újra konfigurálhatjuk. A megjelenítést magunk is megoldhatjuk, de több cég kínál kész függvénykönyvtárakat, használhatjuk ezeket is. A jobb oldalon lévő demonstrációk a Quin Curtis cég Real-Time Graphics & Measurement/Control Tools függvénykönyvtárával készültek. A csomag a különböző élő grafikonokon túl tartalmazza a soros vonali kommunikációhoz egérkezeléshez szükséges függvényeket, hőelem linearizálást, Fourier analízist, PID szabályzást, a grafikus hardcopyt és sok mást. A legújabb változat az ablakkezelést is támogatja. Gyorsíthatjuk a program futását, ha a képernyőt csak akkor, frissítjük, ha a változások meghaladnak egy előre beállított értéket, és csak ott ahol szükséges. A grafikus megjelenítéshez szükséges konstansokat, feliratokat tartsuk a lemezen, ha van elég memóriánk háttérként használjunk ismert formátumú kép file-t. Szükségtelen többféle képernyő kezelése a VGA ma már alapkövetelmény, nagyobb felbontás pedig egy szokványos monitoron nem nagyon követhető.

Ha adatokat gyűjtünk lemezre, ne tartsuk feleslegesen nyitva az adatfile-okat, adatvesztéshez vezethet. Ha különös okunk nincs rá ne használjunk egzotikus file formátumot. Általában megfelel a vesszővel, vagy tabulátorral elválasztott adatokat tartalmazó szövegfile . Ezzel megkönnyítjük a feldolgozást, az adatok böngészését.

Sok sikert.







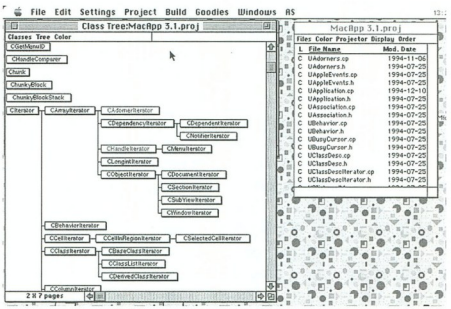
Object Master

Az osztályelső class browser.

Kezdetekben voltak a lyukkártyák, a lyukszalagok, később a file-ok. Ezek mindaddig megfeleltek a programozók igényeinek, amíg a technika nem terjedt túl a modularitáson. A használt tervezési szempontok mint a struktúrált program fejlesztés, vagy a használt program nyelvek (C, Pascal, Algol, Fortran,...) nem követeltek túlságosan sokat a file-editoroktól. A kezdeti igen spártai eszközök után a fejlesztő környezetek egyre fejlettebb file editorokat tartalmaztak. Az objektum-orientált programfejlesztés térhódításával új igény keletkezett. Az objektum-orientált fejlesztés leegyszerűsítve két fő jellemzővel adható meg. Az egyik, hogy az adatokat és a rajtuk végezhető műveleteket egy egységként kezeljük. Ezeket az egységeket szokták osztályoknak (classnak) nevezni. Az egyes osztályokból létrehozott "élő" reprezentánsokat hívják objektumoknak. A másik szerint a programban előforduló azonos jellegű illetve azonos alapokra visszavezethető dolgokat egy-egy osztályba "erőltetjük". Így egy öröklődési hierarchiát hozunk létre, amit a class-tree reprezentál. Ebben az ősök képviselik a közös alapokat, míg a "gyerekek" ezeket a tulajdonságokat specializálják ill. újakkal egészítik ki. Most már nem volt igazán fontos, hogy az adott osztály leírása hol van megadva, sokkal fontosabbak lettek az egyes osztályok tulajdonságai, viselkedése, kapcsolataik, öröklődésük. Ezt az igényt elégítik ki a class browserek. Az amerikai ACI US anyacége belső fejlesztésének segítésére készítette az Object Master fejlesztőkörnyezetet (OM) az Apple Macintosh számítógépekre. Később úgy gondolták, hogy érdemes a nagyközönséggel is megismertetni a programot. Elgondolásukat siker követte. A család jelenleg három tagból áll:

- o Object Master for Think C (Mac)
- o Object Master Universal (Mac)
- o Object Master for Windows (Win)

A Macintosh változatokat több híres cég használja a fejlesztéseihez. A MacWorld '94 áprilisi számában 4 csillagot kapott a lehetséges ötből. Az OM for Windows-t, ami tavaly év végén jelent meg, az InfoWorld az év szoftverének választotta és megkapta a Land Mark Technology Award-ot. A cikk további részében a sikerek okait próbáljuk bemutatni. Az OM szervezésének középpontjában a project áll. Ebben lehetnek source file-ok, resource file-ok, egyéb szöveg file-ok, (a Macintoshon) help file-ok. Ezt többnyire más környezetek is megengedik. A project ablakban viszont több adatot jeleníthetünk meg a file-okról (path, méret, ...) és színekkel is kódolhatjuk file-jainkat a könnyebb osztályozáshoz, amit így együtt igen ritkán lehet megtalálni. Mindemellett lehetővé teszi a file-ok valamilyen archiváló rendszerbe ágyazását (SVCC, SourceServer). Az osztályok rendszerét a class-tree ablakban nézhetjük. Az egyes osztályokhoz hozzáirattathatjuk fieldjeit és methodjait (data member ill. member function). Ha csak egy rész áll érdeklődésünk középpontjában, a nem szükséges dolgokat elrejthetjük. Természetesen az osztályokat is kiszínezhetjük. A színek mellett lehetőség van az osztályok és file-ok saját ill. library cimkével történő ellátására. Ha valamelyik eljárás jobban érdekel minket, egy dupla-klikk és már a Browserben tanulmányozhatjuk, javíthatjuk azt. A program legtöbbet használt része ez az ablak. Mivel az OM mindent tud projectünkről, rendkívül könnyedén tudunk függvényhívásokat generáltatni vele. Így mentesülünk a paraméterek utáni bogarászás időt rabló tevékenysége alól. Ugyanez áll az egyes elemek leírására. A programírást tovább gyorsítja a makrók alkalmazásának lehetősége. Így a program helyettünk hozza létre az if-else, stb. struktúrákat. Keresni többféleképpen lehet. Lehetőség van az éppen elöl lévő file-ban/methodban keresni. Lehetőség van az egész projectben keresni (grep-szerűen), a keresés helyét leszűkíthetjük a színek vagy más kritériumok alapján. Végül lehetőség van a project elemei közt (osztályok, függvények,...) keresni. Ez utóbbi leginkább a hypertext linkhez hasonlít. A felesleges (hiba kereső) fordítások kiváltására, akár automatikusan mentéskor, lehetőség van szintakszis ellenőrzésre. További segítség az egyes szintaktikus elemek egyedi stílussal, színnel való írása. Az objektum orientált technika alkalmazása nem mentesít attól, hogy egész file-okat editáljunk. Gyakorlatilag mindaz, amit a Browser ablak szerkesztési lehetőségeiről elmondtunk, igaz a file editorra is. Az egyes függvények gyors megkeresését teszik lehetővé az



A class tree és a project window egymás mellett

Automarkerek. Hasonló markereket hoz létre a hibák helyeinél is. Resource-ainkat is nyilván tartja a program, sőt néhány fontosabb típust a programon belül is meg tudunk nézni. A szokásos dupla-klikk után a resource máris javítható az általunk választott resource editorunkban. Az OM szinte az összes fontos fordítókörnyezettel képes együttműködni, hiszen nélkülük nem lehetne programot készíteni. A PC-n készült projectek minden további nélkül használhatók a Macintoshon és fordítva. Mindemellett kihasználja a platform nyújtotta egyéb lehetőségeket is.Pl. a Macintoshon teljeskörű AppleScript támogatással rendelkezik. Ez egy rendkívül hasznos és viszonylag egyszerű "programnyelv", amivel batch file-szerűségeket készíthetünk. Pl. ahhoz, hogy az összes T betűvel kezdődő osztályt gyári függvénnyé alakítsunk ennyit kell írni: set library of (every class whose name begins with "T") to true. Persze ezt akármilyen mélységig tovább bonyolíthatjuk. A program mai ára, 36.000,- Ft, igen hamar megtérül. Ha figyelembe veszük, hogy kb. 20%-os teljesítménynövekedést érhetünk el, még a magyarországi nyomott keresetek mellett is 1-2 hónap alatt behozza a befektetést.

Varga György
StarKing Óbuda Apple Center

Házimozi

Multimédia eszközök programozása WINDOWS alatt - AVI fájlok lejátszása

Napjainkban a számítógépek egyre többször zenélõgépek, képmutogató és mozigépek is. Az új eszközök új kihívásokat jelentenek a programozóknak. A WINDOWS születése óta törekszik rá, hogy elfedje a hardver sokféleségét elõlünk. A WINDOWS Multimédia Bõvítés, ami a 3.1-es verzió óta beépült az alaprendszerbe, a Multimedia Control Interface (MCI - Multimédia Vezérlõ Felület) eszközeit ajánlja, mint magas szintû megoldást a problémára. Itt a magas szintû az általános szinonímájaként szerepel.

Természetesen a feladat nem hasonlít ahhoz, mint amikor egy egeret kell kezelni. Mert mit várunk az egértõl - mozogjon, lehessen kattintani a gombjaival. A mi szempontunkból az egereket csak a gomb-jaik száma különbözteti meg.

Ugyanakkor egy multimédia anyag lehet egy audio CD - ekkor le akarjuk játszani a rajta levõ zenét. Lehet egy animáció, ezt el akarjuk helyezni egy ablakban, és ott lejátszani - és azt is meg kell mondanunk, melyik fájl az, amit látni akarunk. Ezért az MCI definiálja a kötelezõen megvalósított paran-csokat (required commands), amelyeket minden vezérlõnek támogatnia kell, ezekkel lehet megnyitni, lekérdezni az eszközöket. Így lehet megtudni azt is, melyek a további megvalósított parancsok. Definiál alapvetõ parancsokat (basic commands), amelyek támogatása opcionális, de ha egy eszköz használja, azt szabványos módon teszi, ilyenek a lejátszó, lezáró parancsok. A harmadik csoportba a kiterjesztett parancsok (extended commands) tartoznak, amik az eszközfüggõ tevékenységeket végzik - ilyenekkel lehet megadni például, hogy az a bizonyos animáció hol jelenjen meg a képernyõn. Az MCI definiálja és támogatja az alapparancsok kibõvítését is.

Mielõtt rátérnénk a példákra, tisztázni kell, hogy a prog-ramozási nyelvekbõl hogyan adhatók ki a parancsok. Az MCI kétféle megoldást nyújt. A C (és persze PASCAL, C++ stb.) programokhoz bináris felületet ad, ahol C struktúrákon keresztül történik a kommunikáció. A driver mindig ilyen parancsokat kap. A WINDOWS azonban tartalmaz egy értelmezõ felületet, amin keresztül az eszköznek szövegesen is elküldhetõ egy parancs. Ez Visual Basic programokban, multimédia keretrendszerekben (például a ToolBook) használható.

Nézzük milyen parancsokra lesz szükségünk egy AVI fájl lejátszásához. Elõször is meg kell nyitni a drivert - ezt a következõ kódrészlet végzi:

```
#include <mmsystem.h>

MCI_OPEN_PARMS mciOP;
char                    dt[]="AVIvideo";
char                    en[]="c:\\WINDOWS\\hangldvi.avi";
WORD                    wId=0;
char                    str[120];
DWORD                   r;

mciOP.dwCallBack=NULL;                  // nem használ visszajelzést
mciOP.wDeviceID=0;
mciOP.wReserved0=0;
mciOP.lpstrDeviceType=dt;       // system.ini-ben szereplõ tipusnév
mciOP.lpstrElementName=en;      // a fájl neve
mciOP.lpstrAlias=NULL;                  // majd ID-vel hivatkozik rá

r= mciSendCommand(wID,MCI_OPEN,MCI_WAIT|MCI_OPEN_ELE-
MENT,&mciOP);
if (r!=0l)
        {
        // Valami hiba történt
        r=mciGetErrorString(r,str,120);
        MessageBox(GetFocus(),str,MB_OK|MB_ICONSTOP);
        }
else
        {
        wID=mciOP.wDeviceID; // ezentúl ez az azonosító
        }
```

A függvény harmadik paraméterének jelentése:

MCI_WAIT - addig nem tér vissza, míg be nem fejezi a mûködést, lehetne:

MCI_NOTIFY - azonnal visszatér, és a dwCallBack mezõben megadott ablaknak küld egy üzenetet, ha végzett.

Ezek az értékek minden parancsnál ilyen jelentést hordoznak. Az **MCI_OPEN**_ELEMENT flag azt jelzi, hogy az lpstrElementName mezõ érvényes adatot tartalmaz . A struktúrákba beírt paramétereket általában is itt kell érvényesíteni. A bináris felület használata esetén nem kell megadni fájlnevet - utólag, **MCI_LOAD** paranccsal is be lehet tölteni a fájlt.

Ugyanez Visual Basicben a következõképpen nézne ki:

```
DECLARE SUB mciExecute Lib
    "mmsystem.dll" (byval SendString$)
...
mciExecute "open c:\WINDOWS\hangldvi.avi
type AVIvideo alias mm"
```

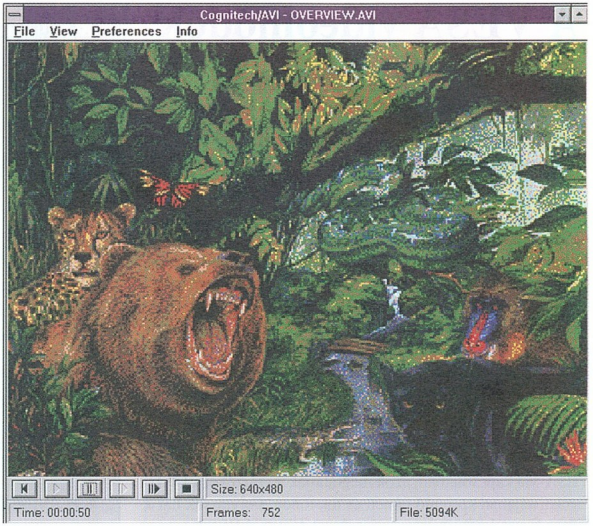
Ha fájl is tartozik az eszközhöz, a szöveges felület használata esetén a megnyitáskor a legtöbb vezérlőnél a fájl nevét kötelező megadni. Utána jön a típus neve, ami a SYSTEM.INI-ben felsoroltak közül az egyik, majd megadjuk azt a nevet (alias), amit a továbbiakban a fájlnév helyett akarunk használni. Az mciExecute parancs hiba esetén automatikusan megjelenít egy üzenetet, amiben kiírja a hiba szövegét. Következzék a video lejátszása. Először a C nyelvű változat:

```
MCI_PLAY_PARMS    mciPP;

mciPP.dwCallBack=(DWORD)hMainWnd;
            // Az ablak amit értesíteni kell
mciPP.dwFrom=0;
mciPP.dwTo=0;
r=mciSendCommand(wID,MCI_PLAY,MCI_NOTI
FY|MCI_PLAY_FROM,&mciPP);
```

Visual Basicben:

```
mciSendString "play mm"

Majd a végén le kell zárni az eszközt:

MCI_GENERIC_PARMS          mciGP;
...
mciPP.dwCallBack=NULL;                    // MCI_WAIT
r=mciSendCommand(wID,MCI_CLOSE,MCI_WAIT,&mciGP);
...

illetve:

mciExecute "close mm"
```

Az előbbieket felhasználva a következő hónapban egy kis AVI videó lejátszó programot írunk WINDOWS alá, BORLAND C 3.1 verzió alatt . Ebben a programban - más érdekességek mellett - példát mutatunk nem csak az eddig említett parancsokra, hanem a státusz lekérdezésre (lejátszási idő, aktuális pozició), a lejátszási pozíció beállítására és az MCI - WINDOWS ablak összerendelésre, stb...



A VPLAYER.EXE program főképernyője.

A téma iránt mélyebben érdeklődők az MCI-ről további információkat a fejlesztő programokhoz adott MCI Reference (szöveges felület) és Multimédia Reference (bináris felület) súgó fájlokban találhatóak. Az ismerkedést talán célszerű a szöveges felülettel kezdeni - itt áttekinthetőbb a parancsok leírása.

Ha az alapkészletben nincs meg a leírása annak az eszköznek, amit kezelni kell, akkor általában segít, ha egy hasonló eszközt keresünk. Például az AVI video bővítések (amikhez nincs leírás) nagyrészt megegyeznek az animáció bővítéseivel.

Adolf Mihály
COGNITECH
Informatikai Kft.

A VESA videomódok programozása II.

Az előző hónapban indult cikksorozat e havi témája a 16 színű VESA módok használata

Az előző cikkben közöltekhez még két dolgot fűznénk hozzá, az egyik az, hogy a SuperVGA-VESA információkat tároló struktúrát (VESAInfoBlock) biztonsági okokból kiegészítettük plusz 256 byte-al (Security), mivel egy-két VESA-DRIVER több információt próbál meg beleírni, és ez könnyen okozhat gondot (pl.: lefagy a gép). A plusz információknak egyébként nincs jelentősége, mi ezek nélkül is ki tudjuk használni a VESA üzemmódokat. A lemezen a már látott VESA01.PAS program javított változatát is adjuk mert az előzőben egy kis technikai gond volt. Akiknek bármilyen problémája, észrevétele volt, van, lesz a programokkal vagy a VESA-val kapcsolatban, azok megtalálják a szerzőket a UNIT forráslistájában szereplő címen és telefonon. Folytassuk hát a VESA birodalmában tett utazásunkat a 16 színű VESA üzemmódok ismertetésével. A túloldali táblázatban a VESA VBE (Video Bios Extension) megszakításainak leírását találhatjuk. Természetesen vannak még ezen kívül is megszakítások, ezekről, mivel ritkán alkalmazhatóak majd csak a végén írunk. A 16 színű üzemmódok kezeléséről már elég sok helyen írtak, nem árt ha a standard VGA-n tudjuk őket programozni. Mint a többi dolog a VESA-n keresztül, egy kicsit ez is másképpen van. Már a 16 színű üzemmódokban is kell lapozni, mivel a 640×480 még belefér 64 KByte-ba, de a 800×600 (ami a 16 színű VESA-ban a legkisebb felbontású üzemmód) már nem. A videomemóriát 64 KByte-onként lapozzuk, vannak viszont olyan kártyák, amelyek ennél kevesebbel tolják el a virtuális ablakot a tényleges videomemórián, ezért az üzemmódinformációs blokkból az „Az ablak szemcsemérete kilobyte-ban", Pascal-ban pedig a ModeInfoBlock.WindowGranularity mezővel el kell osztani a 64 KByte-ot, és ennyiszer kell eltolni az ablakot. Egyébként a szemcseméret alatt mindíg azt a KByte-ban meghatározott értéket értjük amellyel el tudjuk tolni a virtuális ablakot. (Ez függ a kártyától, de általában azért 64 KByte) A VESA UNIT-on keresztül az alábbi konstansokkal tudjuk kezelni a 16 színű üzemmódokat:

m800×600x16
m1024×768x16
m1280×1024x16

A 16 színű üzemmódoknál az attributumregiszterek mutatnak a palettaregiszterekre, ezt mi úgy állítottuk be, hogy az első 16 palettaregiszterre mutasson. A UNIT-ban található pontkirakó és pontlekérdező a szokásos 16 színű rutin bővített változata. A bővítés a bankfigyelésből és a koordinátafigyelésből áll. Ezekben a rutinokban a mode 2 írásmódot és a mode 0 olvasásmódot használjuk. A következő cikkben a 256 színű üzemmódokkal foglalkozunk, addig is használják a 16 színűt sok sikerrel. A cikkhez tartozó példa programok a lemezmellékleten megtalálhatók.

Fneisz–Virágos

1. A SuperVGA információk lekérdezése:

AX	=	4F00H
ES:DI	=>	Az 512 byte-os buffer a SuperVGA információknak.
INT 10H		
Visszatérés:		
AL	=	4FH ha a funkció használható.
AH	=	Státusz 00H végrehajtva 01H megszakítva

2. A SuperVGA módinformációk lekérdezése:

AX	=	4F01H
CX	=	SuperVGA videomód száma.
ES:DI	=>	A 256 byte-os buffer a SuperVGA módinformációknak.
INT 10H		
Visszatérés:		
AL	=	4FH ha a funkció használható.
AH	=	Státusz 00H végrehajtva 01H megszakítva

3. A SuperVGA videomód beállítása:

AX	=	4F02H
BX	=	Videomód száma (A 15. bit beállítása esetén nem törlődik a videomemória)
INT10H		
Visszatérés:		
AL	=	4FH ha a funkció használható.
AH	=	Státusz 00H végrehajtva 01H megszakítva

4. Az aktuális videomód lekérdezése:

AX	=	4F03H
INT 10H		
Visszatérés:		
BX	=	A videomód száma.
AL	=	4FH ha a funkció használható.
AH	=	Státusz 00H végrehajtva 01H megszakítva

5. A videomemória vezérlése:

AX	=	4F05H
BH	=	Alfunkció
	00H	Videomemóriaablak kiválasztása DX = Ablak címe a videomemóriában (szemcseméretben megadva) BL = Ablak száma 00H „A" Ablak 01H „B" Ablak
	01H	Videomemóriaablak címének lekérdezése
INT 10H		
Visszatérés:		
DX	=	Ablak címe a videomemóriában (szemcseméretben megadva)
AL	=	4FH ha a funkció használható.
AH	=	Státusz 00H végrehajtva 01H megszakítva

Clipper & Assembler

A rutinosabb programozóknak most egy kicsivel komolyabb programot szeretnék ismertetni. Biztos vagyok benne, hogy aki clipperben programozik, az rakott már ki a képernyőre ablakokat, programunk ehhez nyújt segítséget. Gondolom mindenki egyetért azzal, hogy egy ablak ma már árnyék nélkül nem is ablak. A programunk segítségével egy adott ablaknak árnyékot rajzolhatunk. A clipper utasítás formátuma a következő:

ARNYEK (x1 , y1 , x2 , y2)

Azoknak akik járatlanok az assembler nyelvben azoknak sem kell megijedniük, mivel ha a forrást masm-al obj-re fordítjuk (masm arnyek.asm ;; + enter), akkor azt egyszerűen beilleszthetik saját könyvtárukba (lib sajat.lib +arnyek.obj ;; + enter) Akinek ez is bonyolult, az használja nyugodtan a forras.lib-et amiben megtalálhatóak az eddig megjelent utasítások is.

A procedúra működik monokróm és színes kártyákon egyaránt, mivel az „i0" címke után egyből azzal kezdi, hogy megnézi a grafikus kártya típusát és beállítja a helyes képernyőmemória kezdőcímét. Ezután kiszámolja az ablak X és Y hosszát, és az adot értékhosszakkal kirakja az árnyékokat vízszintesen és függőlegesen is. A program automatikusan elvégzi a képernyő vágását, így az olyan részekkel amelyek nem kerülnének a képernyőre, nem kell foglalkoznunk.

A segédrutin segítségével gyorsabb lesz a programunk.

SZALAY ZSOLT

```
INCLUDE  EXTENDA.INC

CODESEG  arnyek

CLpublic  <ARNYEK>

 CLfunc void ARNYEK <int y1,int x1,int y2,int x2>
     CLcode

     jmp    i0

arki   proc
     push    ax
     push    cx
     cmp     ax,24
     ja      arkii
     cmp     dx,79
     ja      arkii
     mov     cl,160
     mul     cl
     add     ax,dx
     add     ax,dx
     inc     ax
     mov     bx,ax
     mov     al,7
     mov     es:[di+bx],al
arkii:pop    cx
     pop     ax
     ret
arki  endp
```

```
arny  proc
      push    bx
      mov     ax,y2
      inc     ax
      mov     dx,x1
      add     dx,2
      mov     cl,bl
      xor     ch,ch
ar1:  call    arki
      inc     dx
      loop    ar1
      pop     bx
      push    bx
      mov     ax,y1
      inc     ax
      mov     dx,x2
      inc     dx
      mov     cl,bh
      xor     ch,ch
ar2:  call    arki
      inc     dx
      call    arki
      dec     dx
      inc     ax
      loop    ar2
      pop     bx
      ret
arny   endp

i0:   mov     ax,0
      mov     es,ax
      mov     di,463h
      mov     ax,es:[di]
      cmp     ax,3b4h
       jnz    i1
      mov     ax,0b000h
       jmp    i2
i1:    mov     ax,0b800h
i2:    mov     es,ax
       mov     ax,Y2
      sub     ax,Y1
      cmp     ax,0
      jne     tov1
      jmp     vege

tov1:mov      bh,al          ; YH a BH-ba
      inc     bh
      mov     ax,X2
      sub     ax,X1
      cmp     ax,0
      jne     tov2
      jmp     vege

tov2:mov      bl,al          ; XH a BL-be
      inc     bl
      mov     di,0
      call    arny

vege:
     CLret
End
```

Animációóó

Az FLI formátumú animációs file-ok lejátszása

Biztosan sokan el tudnának képzelni programjaikban egy tetszetős animációt. Ennek elkészítésére számtalan program létezik (AutoDesk Animator, 3D Studio stb.), a kész grafikus anyagok lejátszását viszont saját magunknak kell megoldanunk. Most induló sorozatunkban ezt tárgyaljuk, bemutatunk egy FLI/FLC player-t, majd foglalkozunk az animációk egy speciális fajtájával, a vektor-animációval is.

Ahogy a PC-k egyre nagyobb iramban fejlődnek, úgy hódítja meg fokozatosan a multimédia a szoftver-piacot. Öt évvel ezelőtt még elképzelhetetlen volt, hogy valaki pár óra alatt elkészítsen otthon egy 16 millió színű ray-trace animációt. Ennek ma már semmi akadálya nincsen (kivéve talán szűkös pénztárcánkat). Ennek az intenzív fejlődésnek köszönhetően a multimédia egyre szélesebb felhasználási területével lassan minden programozó szívébe belopja magát (ne feledjük, „ami késik, az nem múlik"). Rendezni kellett az ezen a területen is egyre jobban eluralkodó káoszt, amely mára már valamelyest csillapodott. Ennek egyik mérföldköve volt az AutoDesk által kifejlesztett FLI formátum, ami az 1991-ben piacra dobott AutoDesk Animator 1.0-ban jelent meg először. FLI formátumban csak 320×200 felbontású, 256 színű grafikus üzemmódban szerkesztett animációt tárolhatunk. Ez ma már kevésnek tűnhet, de a maga idejében bőven elégnek bizonyult. Egy FLI file közepesen összetett, azonban a mögötte rejlő ötlet nagyon egyszerű: minek tároljuk egy frame-nek (animációs fázisnak) azokat a részleteit, amelyek megegyeznek az előző frame egyes részleteivel? Ez a módszer nem csak a file méretének csökkenését eredményezi, hanem a lejátszás sebességét is növeli. Az FLI file-ok egy 128 byte-os fejléccel kezdődnek, majd ezt követik az animációs fázisok adatai. Egy frame ún. chunk-okra (darabokra) osztható, amiket ugyancsak további egységekre, csomagokra bonthatunk. Az animációs fázisok tömörítve szerepelnek a file-ban, az első frame a többitől eltérő tömörítési séma szerint van tárolva. (Néha előfordulhat, hogy az első és/vagy a többi frame adatai nincsenek tömörítve). Az FLI végén van egy extra frame, amely az első és utolsó fázis közötti különbségeket tartalmazza.

A frame fejléc után következnek a chunk-ok, amelyek felépítik az animációs fázist. Először egy szín chunk jön, ha a paletta megváltozott az előző frame-hez képest. Ezután következik egy pixel chunk, ha a pixelek változtak az előző fázis óta. Ha a frame teljesen megegyezik az előtte lévővel, nincsenek chunk-ok a fejléc után. Ki gondolná ezek után, hogy a chunk-ok is fejléccel kezdődnek?

Az összes tömörítési séma byte orientált. Ha a tömörített adat páratlan számú byte-ból áll, egy „kitöltő" byte kerül beszúrásra, így az FLI_COPY-k mindig páros címen kezdődnek, ezzel lehetővé téve a gyorsabb DMA átvitelt. Ezek után nézzük talán meg az egyes chunk-ok felépítését egy kicsit részletesebben is!

Az FLI fejléc felépítése :

Byte offset	Méret	Név	Magyarázat
0	4	size	File mérete. Azok a programok használják, amelyek ha lehetséges beolvassák az egész FLI-t.
4	2	magic	$AF11-et tartalmaz. Ha megváltoztatjuk a formátumot, célszerű ide más értéket írni, így az Auto Desk Animator nem fogja megkísérelni az FLI beolvasását.
6	2	frame	Az animációs fázisok száma. Egy FLI file -ban maximum 4000 frame lehet.
8	2	width	Képernyő szélessége (320).
10	2	height	Képernyő magassága (200).
12	2	depth	Egy pixel színmélysége bitekben (8).
14	2	flags	0-t kell tartalmaznia.
16	2	speed	A függőleges rasztervisszafutások száma egy frame alatt.
18	4	next	0-ra van állítva.
22	4	frit	0-ra van állítva.
26	102	expand	Minden byte-ja nulla. A későbbi bővítésekre fenntartva.

Ezután jönnek a frame-ek, mindegyik tartalmaz egy fejlécet :

Byte offset	Méret	Név	Magyarázat
0	4	size	A frame mérete byte-okban. Az AutoDeskanimator megköveteli, hogy kisseb legyen 64 KByte-nál. (A fejléc is beleszámít !)
4	2	magic	Mindíg $F1FA-t tartalmaz.
6	2	chunks	Chunk'-ok (darabok) száma a frame-ben.
8	8	expand	Minden byte-ja nulla. A későbbi bővítésekre fenntartva.

FLI COLOR típusú chunk-ok:

Az első 16 bites szó a csomagok száma ebben chunk-ban. Ezt közvetlenül követik az egyes csomagok.
Egy csomag első byte-ja megmutatja, hogy hány színt kell átugrani. A második byte a változtatni kívánt színek számát adja meg. Ha ez a byte 0, akkor mind a 256 színt meg kell változtatni. Ezután következik színenként 3 byte (piros, zöld, kék színösszetevők).

FLI LC típusú chunk-ok:

Ez a leggyakrabban előforduló, és egyben a legösszetettebb chunk. Az első 16 bites szó azoknak a soroknak a száma a képernyő tetejéhez viszonyítva, amelyek nem változtak az előzőfázishoz képest. (Pl. ha a képnek csak a legalsó sora változott meg, akkor innen 199-et olvashatunk ki.) A következő 16 bites szó adja meg a módosítani kívánt sorok számát. Ezt követik a megváltozott sorok adatai. Minden sor egyenként van tömörítve. Minden tömörített sor első byte-ja a sorban lévő csomagok számát tartalmazza. Ha a sor változatlan az előző frame óta, akkor ennek értéke 0. Egy ilyen csomag első byte-ja (SkipCount) informál minket az átugrandó pixelek számáról. (Magyarul ennyivel kell növelni a vízszintes képernyőpozíciót, de ez a forráslistából úgyis ki fog derülni.)

A chunk-fejléc felépítése :

Byte offset	Méret	Név	Magyarázat
0	4	size	Chunk mérete byte-okban. (A fejléc is beleszámít!)
4	2	type	Chunk típusa (ld. lejjebb).

Öt féle chunk létezik a standard FLI formátumban:

Szám	Név	Magyarázat
11	FLI_COLOR	Tömörített paletta.
12	FLI_LC	Soronként tömörített képadatok. A leggyakrabban előforduló típus. Azokat a pixeleket tárolja, amelyek megváltoztakaz előző fázis óta.
13	FLI_BLACK	Az egész képernyőt törli (0.színűre.Csak az első frame-nél.)
15	FLI_BRUN	Byte-onként tömörített képadatok. (Csak az első frame-nél.)
16	FI_COPY	Jelzi egy 64000 byte-os, tömörítetlen bittérkép következtét. (Ha a tömörített adatok több helyet foglalnának,mint a tömörítetlen bitmap.)

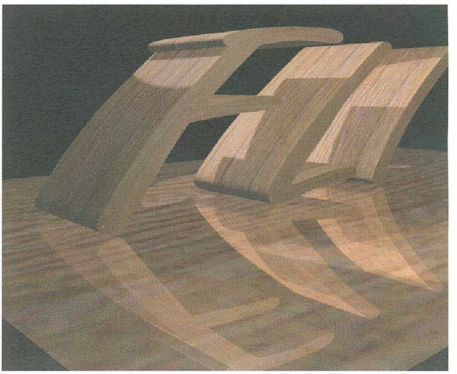
előfordulhat, hogy 255-nél több pixelt kell kihagyni, ezt csak két csomagban lehet tárolni.
A második byte (SizeCount) előjeles értéket tartalmaz. Ha pozitív, akkor SizeCount számú byte követi, amiket ki kell másolnunk a képernyőre. Ha negatív, csak egy byte áll utána, amit – SizeCount – szor kell kiírni. A legrosszabb esetben egy FLI_LC chunk akár 70 KByte is lehet, ezért ha a mérete 60000 bytenál nagyobb lenne, akkor a frame nem lesz tömörítve, hanem egy FLI_COPY típusú chunk-ban lesz tárolva.

FLI BLACK típusú chunk-ok :

A legegyszerűbb típus, semmiféle adatot nem tartalmaz, egyszerűen a képernyő törlésére ad utasítást. Sok jelentősége nincsen, én még egyetlen egy FLI-ben sem találkoztam vele.

FLI BRUN típusú chunk-ok:

Nagyon hasonlít az FLI_LC chunk-ra, a különbség csak annyi, hogy ez nem tartalmaz a kihagyandó pixelek számára vonatkozó adatokat. Közvetlenül az első sor adataival kezdődik, és sorról-sorra leírja az egész bitmap-et (természetesen tömörítve), tehát mind a 200 sor adatait tartalmazza. Egy sor első byte-ja a sorban lévő csomagok száma. Minden csomag első byte-ja (SizeCount)a csomagban lévő pixelek számát tartalmazza. Ha ez pozitív, akkor csak egy byte követi, amit ki kell raknunk a képernyőre SizeCount – szor. Ha negatív, – SizeCount számú adatbyte-ot találunk utána, amiket ki kell másolnunk a képernyőre. Ha a chunk mérete 60000 byte-nál nagyobbra jön ki, ugyanaz vonatkozik rá, mint az FLI_LC típusúra.



FLI COPY típusú chunk-ok:

Ez is egy egyszerű chunk, 64000 byte-ot tartalmaz, amiket módosítás nélkül ki kell másolnunk a képernyőre. Ez a leírás egy kicsit talán nehezen érthető ilyen szárazon, a program alapján azonban remélhetőleg mindenki meg fogja érteni, hogy miről is van szó. Első lépésként egy Init nevű eljárást hajtunk végre, amely először ellenőrzi, hogy van-e 64 KByte szabad hely a heap-ben a file-puffer számára. Ezt követően ellenőrizzük a paramétereket, megnyitjuk a file-t és lefoglalunk 64 KByte-ot, amelybe az aktuális frame adatait fogjuk majd beolvasni. Ezután beolvassuk a file első 128 byte-ját egy FileHeader típusú rekordba, majd a Magic mező segítségével megállapítjuk, hogy valóban egy FLI file-ról van-e szó. Ha az inicializálás hiba nélkül megtörtént, jöhet az animáció lejátszása. Ezt a Play nevű procedúra végzi, amely első lépésként beállítja a 13H számú grafikus üzemmódot a Set 320×200×256 nevű rutin meghívásával. Ezután a DrawFrame eljárás segítségével kirajzoljuk az első fázist. A filemutató a DrawFrame rutinból visszatérve a következő frame elején áll, amit el is tárolunk a SecondFrame-Pos változóban. Azt már tudjuk, hogy az FLI-ben van egy extra frame, amely az első és utolsó fázis közötti különbségeket

```
{$I-}{$G+}
Program FLI_Player ;

Uses Crt, Dos ;

Type
  FileHeaderType = Record
    Size   : Longint ;
    Magic  : Word ;
    Frames : Word ;
    Width  : Word ;
    Height : Word ;
    Depth  : Word ;
    Flags  : Word ;
    Speed  : Word ;
    Next   : Longint ;
    Frit   : Longint ;
    Expand : Array[0..101] Of Byte ;
  End ;

  FrameHeaderType = Record
    Size   : Longint ;
    Magic  : Word ;
    Chunks : Word ;
    Expand : Array[0..7] Of Byte ;
  End ;

Var
  FileHeader  : FileHeaderType ;
  FrameHeader : FrameHeaderType ;
  FileName    : String ;
  FLIFile     : File ;
  FramePtr    : Pointer ;
  Frame       : Word ;
  Chunk       : Word ;
  CurrentOfs  : Word ;

Procedure Done ;
  Begin
    FreeMem(FramePtr, 65535) ;
    Close(FLIFile) ;
    TextMode(3) ;
  End ;

Procedure Error(ErrorText : String) ;
  Begin
    Done ;
    Writeln(ErrorText) ;
    Halt(1) ;
  End ;

Procedure Init ;
  Begin
    Writeln('FLI Player V1.0') ;
    Writeln('K,sz¡tette : Vir  gos Adri  n ,s ifj. Fneisz J¢zsef.') ;
    Writeln('Forr  sk¢d 1995.'+#10) ;
    If MaxAvail < 65535 Then
      Begin
        Writeln('Nincs elég memória a file-pufferhez.') ;
        Halt(1) ;
      End ;
    If ParamCount = 0 Then
      Begin
        Writeln('Használat : PlayFLI filenév.FLI') ;
        Halt(1) ;
      End ;
    FileName := ParamStr(1) ;
    Assign(FLIFile, FileName) ;
    Reset(FLIFile, 1) ;
    If IOResult <> 0 Then
      Begin
        Writeln('A megadott file nem létezik.') ;
        Halt(1) ;
      End ;
    GetMem(FramePtr, 65535) ;
    BlockRead(FLIFile, FileHeader, 128) ;
    If IOResult <> 0 Then
      Error('Hiba az FLI file olvasása közben.') ;
    If FileHeader.Magic <> $AF11 Then
      Error('Hib  s FLI file-fejl,c.') ;
  End ;

Procedure Set320x200x256 ; Assembler ;
  Asm
    MOV   AX,0013H
    INT   10H
  End ;

Procedure FLI_COLOR ;
  Var
    FirstColor : Byte ;
    ColorNum   : Byte ;
    Packet     : Byte ;
    Packets    : Byte ;

  Begin
    Inc(CurrentOfs, 6) ;
    Packets :=
MemW[Seg(FramePtr^):Ofs(FramePtr^)+CurrentOfs] ;
    Inc(CurrentOfs, 2) ;
    For Packet := 1 To Packets Do
      Begin
        FirstColor :=
Mem[Seg(FramePtr^):Ofs(FramePtr^)+CurrentOfs] ;
        ColorNum   :=
Mem[Seg(FramePtr^):Ofs(FramePtr^)+CurrentOfs+1] ;
        Inc(CurrentOfs, 2) ;
        Asm
                PUSH  DS
                MOV   DX,3C8H
                MOV   AL,FirstColor
                OUT   DX,AL
                MOV   AL,ColorNum
                OR    AL,AL
                JZ    @001
                MOV   BL,3
                MUL   BL
                JMP   @002
          @001: MOV   AX,768
          @002: MOV   CX,AX
                MOV   DI,CurrentOfs
                ADD   CurrentOfs,AX
                LDS   SI,[FramePtr]
                ADD   SI,DI
                MOV   DX,3C9H
                CLD
                REP   OUTSB
                POP   DS
        End ;
      End ;
  End ;

Procedure FLI_LC ;
  Var
    SkipCount : Byte ;
    SizeCount : Shortint ;
    Line      : Word ;
    FirstLine : Word ;
    Lines     : Word ;
    VideoOfs  : Word ;
    Packet    : Byte ;
    Packets   : Byte ;

  Begin
    Inc(CurrentOfs, 6) ;
    FirstLine :=
MemW[Seg(FramePtr^):Ofs(FramePtr^)+CurrentOfs] ;
    Lines     :=
MemW[Seg(FramePtr^):Ofs(FramePtr^)+CurrentOfs+2] ;
    Inc(CurrentOfs, 4) ;
    For Line := FirstLine To FirstLine+Lines-1 Do
      Begin
        VideoOfs := (Line)*320 ;
        Packets :=
Mem[Seg(FramePtr^):Ofs(FramePtr^)+CurrentOfs] ;
        Inc(CurrentOfs) ;
        For Packet := 1 To Packets Do
          Begin
            SkipCount :=
Mem[Seg(FramePtr^):Ofs(FramePtr^)+CurrentOfs] ;
            SizeCount :=
Mem[Seg(FramePtr^):Ofs(FramePtr^)+CurrentOfs+1] ;
            Inc(CurrentOfs, 2) ;
            Inc(VideoOfs, SkipCount) ;
            If SizeCount < 0 Then
              Begin
                Asm
                  PUSH  DS
                  MOV   BX,0A000H
                  MOV   ES,BX
                  MOV   DI,VideoOfs
                  MOV   BX,CurrentOfs
                  LDS   SI,[FramePtr]
                  ADD   SI,BX
                  LODSB
                  MOV   CL,SizeCount
                  NEG   CL
                  XOR   CH,CH
                  CLD
                  REP   STOSB
                  POP   DS
                  MOV   VideoOfs,DI
                End ;
                Inc(CurrentOfs) ;
```

```
          End
        Else
          Begin
            Asm
              PUSH  DS
              MOV   BX,0A000H
              MOV   ES,BX
              MOV   DI,VideoOfs
              MOV   BX,CurrentOfs
              LDS   SI,[FramePtr]
              ADD   SI,BX
              MOV   CL,SizeCount
              XOR   CH,CH
              CLD
              REP   MOVSB
              POP   DS
              MOV   VideoOfs,DI
            End ;
            Inc(CurrentOfs, SizeCount) ;
          End ;
      End ;
    End ;
  End ;

Procedure FLI_BLACK ;
  Begin
    Inc(CurrentOfs, 6) ;
    Asm
      PUSH  0A000H
      POP   ES
      XOR   DI,DI
      MOV   CX,32000
      XOR   AX,AX
      REP   STOSW
    End ;
  End ;

Procedure FLI_BRUN ;
  Var
    SizeCount : Shortint ;
    Line      : Word ;
    VideoOfs  : Word ;
    Packet    : Byte ;
    Packets   : Byte ;

  Begin
    Inc(CurrentOfs, 6) ;
    For Line := 1 To 200 Do
      Begin
        VideoOfs := (Line-1)*320 ;
        Packets :=
Mem[Seg(FramePtr^):Ofs(FramePtr^)+CurrentOfs] ;
        Inc(CurrentOfs) ;
        For Packet := 1 To Packets Do
          Begin
            SizeCount :=
Mem[Seg(FramePtr^):Ofs(FramePtr^)+CurrentOfs] ;
            Inc(CurrentOfs) ;
            If SizeCount > 0 Then
              Begin
                Asm
                  PUSH  DS
                  MOV   BX,0A000H
                  MOV   ES,BX
                  MOV   DI,VideoOfs
                  MOV   BX,CurrentOfs
                  LDS   SI,[FramePtr]
                  ADD   SI,BX
                  LODSB
                  MOV   CL,SizeCount
                  XOR   CH,CH
                  CLD
                  REP   STOSB
                  POP   DS
                  MOV   VideoOfs,DI
                End ;
                Inc(CurrentOfs) ;
              End
            Else
              Begin
                Asm
                  PUSH  DS
                  MOV   BX,0A000H
                  MOV   ES,BX
                  MOV   DI,VideoOfs
                  MOV   BX,CurrentOfs
                  LDS   SI,[FramePtr]
                  ADD   SI,BX
                  MOV   CL,SizeCount
                  NEG   CL
                  XOR   CH,CH
                  CLD
                  REP   MOVSB
                  POP   DS
                  MOV   VideoOfs,DI
                End ;
                Inc(CurrentOfs, -SizeCount) ;
              End ;
          End ;
      End ;
  End ;

Procedure FLI_COPY ;
  Begin
    Inc(CurrentOfs, 6) ;
    Asm
      PUSH  DS
      PUSH  0A000H
      POP   ES
      XOR   DI,DI
      LDS   SI,[FramePtr]
      MOV   CX,32000
      REP   MOVSW
      POP   DS
    End ;
    Inc(CurrentOfs, 64000) ;
  End ;

Procedure WaitVideoTicks ;
  Var
    VideoTicks : Word ;

  Begin
    For VideoTicks := 1 To FileHeader.Speed Do
      Begin
        Asm
              MOV   DX,03DAH
          @001: IN    AL,DX
              AND   AL,8
              JZ    @001
          @002: IN    AL,DX
              AND   AL,8
              JNZ   @002
        End ;
      End ;
  End ;

Procedure DrawFrame ;
  Begin
    BlockRead(FLIFile, FrameHeader, 16) ;
    If IOResult <> 0 Then
      Error('Hiba az FLI file olvasása közben.') ;
    If FrameHeader.Magic <> $F1FA Then
      Error('Hibás FLI frame-fejléc.') ;
    BlockRead(FLIFile, FramePtr^, FrameHeader.Size-16) ;
    If IOResult <> 0 Then
      Error('Hiba az FLI file olvasása közben.') ;
    CurrentOfs := 0 ;
    For Chunk := 1 To FrameHeader.Chunks Do
      Begin
        Case
MemW[Seg(FramePtr^):Ofs(FramePtr^)+CurrentOfs+4] Of
          11 : FLI_COLOR ;
          12 : FLI_LC ;
          13 : FLI_BLACK ;
          15 : FLI_BRUN ;
          16 : FLI_COPY ;
        Else
          Error('Hib  s FLI chunk-tipus.') ;
        End ;
      End ;
    WaitVideoTicks ;
  End ;

Procedure Play ;
  Var
    SecondFramePos : Longint ;
  Begin
    Set320x200x256 ;
    DrawFrame ;
    SecondFramePos := FilePos(FLIFile) ;
    Repeat
      Seek(FLIFile, SecondFramePos) ;
      For Frame := 1 To FileHeader.Frames Do
        DrawFrame ;
    Until KeyPressed ;
  End ;
Begin
  Init ;
  Play ;
  Done ;
  ReadKey ;
End.
```

tartalmazza egy FLI_LC vagy FLI_COPY chunk-ban. Célszerű a későbbiekben az első fázis kirajzolására ezt használni, mert az első frame általában egy hosszú FLI_BRUN típusú chunk-ot tartalmaz. Egy for-ciklussal sorban kirajzoljuk az összes frame adatait (beleértve az extra frame-t is!), majd az egészet kezdjük elölről a második frametől kezdve (SecondFrame-Pos filepozíció), egészen addig, amig le nem ütünk egy billentyűt. Ja, és arra vigyázzunk, hogy az extra frame nem számít bele a fejléc Frames nevű mezőjének értékébe! A DrawFrame rutin első lépésként beolvassa a következő frame fejlécét egy FrameHeader nevű változóba, majd a Magic mező segítségével ellenőrzi, hogy a fejléc jó-e (Magic mező=$F1FA).

Ha jó, beolvassa a frame-t a 64 KByte-os pufferbe. Hogy éppen melyik byte feldolgozásánál tartunk az aktuális framen belül, azt a CurrentOfs változó tárolja. A frame-ben lévő chunk-ok száma kiderül a frame fejlécéből, ezért egy egyszerű for-ciklussal sorban feldolgozhatjuk őket. A for-ciklusban a CurrentOfs mindíg egy chunk fejlécére mutat. A WaitVideoTicks rutin a függőleges rasztervisszafutásokra hivatott várakozni. Annyi visszafutást vár meg, amennyi a FileHeader.Speed változó értéke. A $03DA port 3. bitjének 1 értéke jelzi a függőleges visszafutást. Az FLI_COLOR chunkok feldolgozása a csomagok számának megállapításával kezdődik, amiket egymás után szépen feldolgozunk. Az átugrandó színek száma felfogható úgy is, minha az első módosítandó szín sorszámáról beszélnénk, ezért ezt kiküldjük a $03C8-as portra (RGB íráscímregisztere), majd színenként 3 byte-ot a $03C9-es portra (RGB adatregiszter) küldünk (a piros, zöld, kék színösszetevők sorrendjének megfelelően). Az FLI_LC rutinban deklarálunk egy VideoOfs nevű változót, amely a videomemória aktuális byte-jának offsetjét fogja tartalmazni. Értékét minden sor elején Sor*320 értékre állítjuk. A csomagok feldolgozásánál a kihagyandó pixelek számát (SkipCount) hozzáadjuk VideoOfs-hoz, majd a SizeCount által meghatározott adatokat kimásoljuk a video-memóriába $A000:VideoOfs címtől kezdődően. Lényeges dolog, hogy megnöveljük ezután a VideoOfs értékét (SizeCount szerint), mert különben a következő csomag adatai rossz helyen jelennek meg a képernyőn. Ezt nagyon elegánsan oldhatjuk meg, ha kihasználjuk a MOVS és a STOS utasítások tulajdonságát (aki nem tudná : aDI regiszter értéke végrehajtás után inkrementálódik). Másik fontos dolog a CurrentOfs értékének növelése, ezt sem célszerű elfelejteni.

Az FLI_BRUN chunk feldolgozása nagyon hasonlít az LC-hez, SkipCount nevű változót azonban nem használunk, és a sorokat feldolgozó for-ciklus 0-tól 199-ig fut. Az I_BLACK rutin a legegyszerűbb, 32000-szer végrehajtva egy STOSW utasítást, törli a képernyőt. Az FLI_COPY procedúrában a REP MOVSW utasítás segítségével átmásolunk 64000 byte-ot a file pufferből a video-memóriába. A program végén meg kell még hívnunk a Done eljárást, amely felszabadítja a lefoglalt memóriát, bezárja az FLI file-t, és visszaállítja a szöveges üzemmódot. Hát ennyi lenne egy FLI animáció lejátszása. A programban rengeteg dolgot lehetne még optimalizálni, azonban a jobb érthetőség kedvéért ezektől – még ha lassab is így a program – eltekintettünk.

Következő hónapban az FLC típusú animációs file-ok lelkével, használatával foglalkozunk.

Virágos Adrián
Fneisz József

Gravissimo

Az előző számunkban tett ígéretünk szerint részletesen ismertetjük a GRAVIS hangkártyák programozását.

A mellékelt forrásliták csak rövid segítségként, inkább illusztrációként szolgálnak, mintsem követend ő példaként. Mindent lehet jobban csinálni, és ez ebben az esetben is igaz. Mindenkit buzdítanék új fogások, speciális effektusok kifejlesztésére, ahelyett, hogy „csak" begépelje a példákat. Egy regiszterszintű programozáshoz nélkülözhetetlenek a portok, és regiszterek teljes ismerete. Következzék tehát egy részletes ismertet ő. A BASE szimbóluma baseportot jelenti, ez alapértelmezésben 220h (h=hexadecimális). A "..." jelölés a bit nem használt mivoltát jelzi, javasolt értékét 0-ra állítani.

MIDI portok:

Control	BASE+100h (Ir)
Status	BASE+100h (Olvas)
Küldendő adat	BASE+101h (Ir)
Beérkező adat	BASE+101h (Olvas)

A MIDI interface egy a MIDI specifikációnak megfelelő UART egység.

Control port:

bit
0. = reset
1. = reset
2. = ...
3. = ...
4. = ...
5. = Küldendő adat utáni IRQ
6. = Küldendő adat utáni IRQ
7. = Beérkező adat utáni IRQ

Ha a RESET jelű biteket logikai 1-re állítjuk, majd nullázzuk a MIDI áramkör reseteli regisztereit. Ha az 5. bit 1, és a 6. bit 0, akkor minden küldött adat után IRQ kérelem érkezik. Ha a 7. bit 1, akkor minden beérkező adat generál egy IRQ-t.

Status port:

bit
0. = Beérkező adat regiszter megtelt
1. = Küldendő adat regiszter kiürült
2. = ...
3. = ...
4. = Időzítési hiba
5. = Túlfutási hiba
6. = ...
7. = IRQ küldve

A 7. bit értéke jelzi, hogy valamilyen IRQ esemény történt. A 0.,1.,4.,5.,bitek értékeiből az esemény okait kiolvashatjuk.

Joystick portok:

Időzítő	201h (Ir)
Joystick adatok	201h (Olvas)

A portok megegyeznek a PC analóg joystick portjaival. Kezelésük ugyanaz, BIOS és regiszter szinten is.

Kártyavezérlő portok:

Mixer	BASE
IRQ csatorna	BASE+0Bh
DMA csatorna	BASE+0Bh
IRQ status	BASE+006h
időzítő control	BASE+008h
időzítő adat	BASE+009h

Mixer

bit
0. = Analóg bemenet engedélyezése (0)
1. = Analóg kimenet engedélyezése (0)
2. = Mikrofon bemenet engedélyezése (1)
3. = IRQ és DMA kérések engedélyezése (1)
4. = MIDI IRQ és GUS IRQ összekeverése (1)
5. = MIDI Txd és Rxd vezetékek összecsatolása (1)
6. = DMA vagy IRQ beállítás
7. = ...

A port 6. bitje meghatározza, a BASE+0Bh port funkcióját. Ha ez a bit 1 akkor az IRQ csatornát állítjuk be ha 0, akkor pedig a DMA-t. Lehetőségünk tehát megvan rá, hogy átprogramozzuk ezeket.

IRQ csatorna

bit
0. = GUS IRQ
1. = GUS IRQ
2. = GUS IRQ
3. = MIDI IRQ

4. = MIDI IRQ
5. = MIDI IRQ
6. = MIDI IRQ és GUS IRQ összekeverése
7. = ...

GUS IRQ, MIDI IRQ:

0=nincs IRQ
1=IRQ 2
2=IRQ 5
3=IRQ 3
4=IRQ 7
5=IRQ 11
6=IRQ 12
7=IRQ 15

Két azonos IRQ-t ne válasszunk ki, mert az konfliktust okozhat! Ha a 6. bit 1,akkor a második IRQ-t ki kell kapcsolni!

DMA csatorna

bit
0. = DMA 1
1. = DMA 1
2. = DMA 1
3. = DMA 2
4. = DMA 2
5. = DMA 2
6. = DMA 1 és DMA 2 összekeverése
7. = ...

DMA 1, DMA 2:
0 = nincs DMA
1 = DMA 1
2 = DMA 3
3 = DMA 5
4 = DMA 6
5 = DMA 7

IRQ status

bit
0. = MIDI elküldött adat
1. = MIDI beérkező adat
2. = 1. Időzítő
3. = 2. Időzítő
4. = ...
5. = Lejátszás közben
6. = Töréspont (ADSR)
7. = DMA végzett

Ezekből a bitekből megtudhatjuk, hogy mi okozta az IRQ kérelmet.

Időzítő Control

bit
0. = 1. Időzítő indítása
1. = 2. Időzítő indítása
2. = ...
3. = ...
4. = ...
5. = 2. Időzítő letiltása
6. = 1. Időzítő letiltása
7. = Időzítő IRQ indítása

Az 5. és 6. bitekkel leállítjuk az időzítőket amíg programozzuk azok értékeit. Ha ezzel megvagyunk újból 0-ra állítjuk, és elindítjuk a 0. és 1. bitekkel. Ha kaptunk egy IRQ-t, akkor a 7. biten állítanunk kell, különben nem lesz több IRQ esemény. Vigyázat! A két időzítő sebessége különböző!

GUS portok:

aktív csatorna ..... BASE+102h
aktív regiszter ..... BASE+103h
reg. adat LOW ..... BASE+104h
reg. adat HIGH ..... BASE+105h
ON-BOARD adat .....BASE+107h

A BASE+102h porton keresztül kiválaszthatjuk a programozni kívánt csatorna számát. (0–1Fh) Ezután kiválasztjuk a kívánt regisztert a BASE+103h portonkeresztül. Ha a regiszter 16 bites, akkor mind a BASE+104h-es (alsó byte)és a BASE+105-ös (felső byte) portot írjuk, 8 bites esetén csak az utóbbit!Ha a GUS memóriáját nem DMA-val kezeljük, akkor a megcímzett memóriaelemet a BASE+107h porton keresztül érhetjük el. A regisztereket két fő csoportra oszthatjuk: általános regiszterek, és hangregiszterek. Az utóbbiak a hanglejátszásának miképtjéért felelősek, míg az előbbiek a fennmaradó feladatokért felelősek.

Általános Regiszterek:

41h .....GUS memória DMA control (8 bit)
42h .....DMA kezdeti cím (16 bit)
43h .....GUS memória cím alsó (16 bit)
44h .....GUS memória cím felső (8 bit)
45h .....Időzítő control (8 bit)
46h .....1. időzítő adat (8 bit)
47h .....2. időzítő adat (8 bit)
48h .....Mintavételezési frekvencia (8 bit)
49h .....Mintavételezés control (8 bit)
4bh .....Joystick (8 bit)
4ch .....RESET (8 bit)

GUS memória DMA control

bit
0. = DMA mehet!
1. = GUS-ból olvasás(1), GUS-ba írás(0)
2. = 8 (0) vagy 16 (1) bittes DMA csatorna
3. = DMA sebesség
4. = DMA sebesség
5. = DMA végzett IRQ engedélyezése
6. = 8 (0) vagy 16 (1) bittes digitalizált hang
7. = Előjeles (0) vagy nem előjeles (1) digitalizált hang

DMA kezdeti cím

Ez a 16 bites cím az ON-BOARD memóriában az első címre mutat. Mivel a regiszter csak 16 bit széles, ezért a 20 bites cím felső 16 bitjét használjuk, az alsó négy kötelezően 0.

GUS memória cím alsó

Ez az alsó 16 bitje annak a címnek a GUS memóriában, ahová a BASE+107h porton keresztül írni/olvasni szeretnénk.

GUS memória cím felső

Ez a felső 4 bitje annak a címnek a GUS memóriában, ahová a BASE+107h porton keresztül írni/olvasni szeretnénk.

Időzítő control

A regiszter 2. és 3. bitje engedélyezi az 1. és 2. időzítő IRQ kéréseit

1. időzítő adat és 2.időzítő adat

Az ide beírt értékek az időzítők sebességét adják meg. Minél magasabb az érték,annál gyorsabb az időzítés. A 2. időzítő négyszer lassabb mint az 1.

Mintavételezési frekvencia

A értéket az alábbi képletből kaphatjuk meg:
Érték=9878400/(16*(frekvencia+2))

Mintavételezés control

bit
0. = Mintavételezés mehet!
1. = Mono (0), Sztereo(1)
2. = 8 (0) vagy 16 (1) bites mintavételezés
3. = ...
4. = ...
5. = DMA végzett IRQ engedélyezése
6. = ...
7. = Előjeles (0) vagy nem előjeles (1) mintavételezés

RESET

bit
0. = RESET
1. = D/A áramkőr engedélyezése
2. = Összes IRQ engedélyezése
3. = ...
4. = ...
5. = ...
6. = ...
7. = ...

Ameddig a 0. bit logikai 0, addig a GUS reset állapotban van. Egy kis időre hagyjuk így, majd 1-et írva „újraindíthatjuk" a hangkártyát. Ha az 1. bit 0, semmilyen hang képzésére nem képes a GUS. A 2. bit az IRQ kérések főkapcsolója.
Ameddig ez a bit 0, addig nincs semmilyen IRQ.

Hangregiszterek:

Cím	*bit*	*Funkció*
0h	8	Lejátszás control
1h	16	Lejátszás frekvenciája
2h	16	Hang kezdete Hi
3h	16	Hang kezdete Lo
4h	16	Hang vége Hi
5h	16	Hang vége Lo
6h	8	Burkológörbe meredeksége
7h	8	Burkológörbe kezdeti érték
8h	8	Burkológörbe végső értéke
9h	16	Aktuális hangerő
Ah	16	Aktuális lejátszási cím Hi
Bh	16	Aktuális lejátszási cím Lo
Ch	8	Balance érték
Dh	8	Hangerő Control
Eh	8	Maximális D/A csatornák
Fh	8	IRQ status

Ha a regiszterek értékeit olvasni akarjuk, akkor a regiszterek címéhez 80h-t kell hozzáadni.

Lejátszás control

bit
0. = 1 ha a hang megállt
1. = Hang stop!
2. = 8(0) vagy 16(1) bites hang
3. = loop engedélyezés
4. = ping-pong loop
5. = hang lejátszása közben IRQ
6. = (0) előre vagy (1) hátrafelé lejátszás
7. = IRQ jelzés

Lejátszás frekvenciája

A 15-10 bit egész, 9-1 maradék része a léptetőértéknek. Kiszámítása a következő: A kívánt frekvenciát el kell osztani egy konstansal, amely függ a maximális D/A csatornák számától:

konstans=1 000 000 / (1.619695497*aktívnak kijelölt D/A csatornák száma)

Ha a csatornák száma kisebb mint 14, akkor is 14-el kell számolnunk! Tehát 14 csatorna esetén a kapott eredmény: 44100.00001. A legegyszerűbb módszer csupán a legfelső két tizedesjeggyel számolni. Ha tehát 14 csatornánk van, és egy hangot 22 KHz-en akarom lejátszani, akkor a regiszterbe
22 000/44 kerül.

Hang kezdete Hi és Hang kezdete Lo

Ebből a 32 bites értékből az alsó 20 bit definiálja a hang kezdetét a GUS memóriában.

Hang vége Hi és Hang vége Lo

A 32 bites értékből az alsó 20 bit definiálja a hang végét a GUS memóriában.

Burkológörbe meredeksége

A 5-0 bitek tartalmazza az értéket, amit ki kell vonni, (vagy hozzáadni) az előző hangerőértékből. 7-6 bitek pedig a változás sebességét határozzák meg. 0 értéke a leglassabb, 3-as értéke a leggyorsabb. A változás sebessége szintén függ az aktív D/A-k számától.

Burkológörbe kezdeti értéke

Ez a kezdeti hangerő, amelyet csökkenteni, vagy növelni kell.

Burkológörbe végső értéke

Ezt az értéket kell elérnie a hangerőnek. Ha IRQ engedélyezve van, akkor mikor a hangerő eléri ezt az értéket IRQ-t kapunk.

Aktuális hangerő

Ez a regiszter íráskor az aktuális hangerő lesz, felülírva az előzőt, olvasva pedig az éppen aktuális hangerőt adja vissza.

Aktuális lejátszási cím Hi és Aktuális lejátszási cím Lo

Ezek az értékek a GUS belső mutatóját állítják a GUS memóriában. Az érték bárhova mutathat, akár az aktuális hang határain túl is.

Balance érték

Ez az érték a hang „helyét" határozza meg. 0 értéke teljesen balra, 15 értéke teljesen jobbra helyezi a hangot.

Hangerő Control

bit
0. = Hangerő elérte a végértéket
1. = Hangerő változás stop!
2. = Túlcsúszás engedélyezése
3. = Automatikusan újrainduló hangerő változás
4. = ping-pong hangerő változás
5. = IRQ amikor végértéket elértük
6. = (0) hangerő növekedés, (1) hangerő csökkenés
7. = IRQ jelzés

Ez a regiszter irányítja a burkológörbe képzését. Pogramozása fokozott figyelmet igényel, különben összevissza fog szólni minden.

Aktívnak jelölt D/A csatornák

Itt határozzuk meg az aktív D/A csatornák számát 0-tól kezdve. 14-nél kisebb értékeket automatikusan 14-re alakít át a GUS. Maximális értéke 31. Hogy miért van rá szükség? Minél kisebb a szám annál szebben szólnak a hangok a GUS-ból. Természetesen 32 csatorna esetén is kiváló minőséget kapunk, de kevesebb csatorna esetén a GUS processzora több interpolácót tud számolni hangonként.

IRQ status

bit
0. = Az IRQ-hoz tartozó csatorna
1. = Az IRQ-hoz tartozó csatorna
2. = Az IRQ-hoz tartozó csatorna
3. = Az IRQ-hoz tartozó csatorna
4. = Az IRQ-hoz tartozó csatorna
5. = mindig 1!
6. = Hangerő változás IRQ az adott csatornán
7. = A Hang lejátszása közben IRQ kérelem az adott csatornán

Mivel a GUS maximálisan 2 IRQ-t tud meghívni, ezért azonosítani kell az IRQ kérelmet. Ez a regiszter tartalmazza a hang lejátszásával összefüggő eseményeket. Ha itt nem találtunk semmit, ellenőrizzük a BASE+06h regisztert! Na, végre rátérhetünk a programozásra is. A rutinok assembly nyelven íródtak a nagyobb hatékonyság kedvéért, de ez senkit se riasszon vissza attól, hogy magas szintű (pl.: PASCAL) nyelveken írjanak GUS programokat. Kezdjük hát! A baseport megtalálásának több módja is van. Első és legjobban ajánlott módja a DOS ENVIRONMENT végigböngészése. Ha a felhasználó esetleg törölte az ide vonatkozó bejegy-zéseket az AUTOEXEC.BAT-ból, akkor két további lehetőségünk marad. A primitívebb megoldás az, hogy rákérdezünk a beállításokra. Ez PASCAL-ban egyszerű, assembly-ben viszont hosszadalmas és eléggé uncsi is. Ekkor fordulhatunk a második módszerhez, a GUS megkereséséhez. A jó kereső rutin ismérve az, hogy NEM használ OUTPUT műveleteket. Ez azért elkerülendő, hisz egy másik modulkártyát is össze zavarhatunk bizonyos output műveletekkel. Tapasztalataim szerint az ETHERNET kártyás gépeken bizonyos GUS detektáló programok egész egyszerűen kifagynak. Rutinunknak tehát olyan GUS tulajdonságokat kell(ene) keresnie, amelyek puszta INPUT tevékenységekkel megállapíthatóak. Egyik legjobb módszer erre a MIDI status lekérdezése. Egy rövid kis példa:

```
DETECT          PROC                    ;vissza: ax=baseport (0=ha
nem talált)
        mov di,210h
        mov cx,7
KERES: mov dx,di
        add di,10h
        add dx,100h
        in al,dx
        cmp al,02
        je LETEZIK
        loop KERES
        mov ax,0
        ret                             ;nem találtuk.
LETEZIK:
        mov ax,di
        sub ax,10h
        mov BASE,ax                     ;megvan!
        ret
DETECT  ENDP
```

Következő feladatunk a kártya resetelése. Itt biztosítjuk a „biztos alapot" a programunk felépítéséhez. Ebben a kis szubrutinban megpróbáltam összegyűjteni mindent, amire esetleg szükségünk lehet. A rutint természetesen tovább kell bővíteni, ha bizonyos IRQ és DMA szolgáltatásokat is használni akarunk. Megjegyzem, a detektálás nagy múltra visszatekintő iparág, de használata nem ajánlott, a „kérdezgetős" programok pedig eleve ellenérzést váltanak ki a felhasználóból, annak ellenére, hogy 3 gombnyomással túl van rajtuk ez ember. A DOS environment-et direkt erre a célra tervezték, és talán ez a DOS egyetlen jövőbe mutató szolgáltatása, ezért az igényesebb szoftverek meg is követelik a használatát.

```
RESET PROC
        mov dx,BASE
        mov al,00000011b
        out dx,al               ;Audio kimenet lekapcsolása

        mov cx,32
HANGOK:         mov dx,base
        add dx,102h
        mov al,cl
      dec al
        out dx,al
        inc dx
        mov al,0
        out dx,al
        add dx,2
        mov al,03h
        out dx,al
        loop HANGOK             ;összes D/A csatorna stop!
        mov dx,base
        add dx,103h
        mov al,4ch
        out dx,al
        add dx,2
      mov al,1
      out dx,al
      mov cx,65535
WAIT:  in al,dx
      loop WAIT
        mov al,7
        out dx,al               ;reset GUS

        mov dx,base
        add dx,103h
        mov al,0eh
        out dx,al
        add dx,2
        mov al,14-1
        out dx,al               ;maximális D/A csatornák száma : 14
        mov dx,base
        mov al,00001000b
        out dx,al               ;kimenet engedélyezése

        ret
RESET ENDP
```

Nagyon fontos, hogy a „kimenet engedélyezése" részt, a saját elképzeléseink szerint írjuk meg, hisz itt más, fontosabb bitek is vannak! A meglehetősen furcsa várakozó részt a gyakorlati -és kényelmi- szempontok eredménye, természetesen tetszőlegesen módosítható. Én a D/A-k számát 14-re állítottam be, ezért továbbiakban a 44-es konstanst fogom használni a frekvenciák kiszámításához. Miután megtaláltuk a beállításokat, és reseteltük a GUS-t, meg kell állapítani egy másik nagyon fontos paramétert: a memória nagyságát. Bár a GUS-t alapkiépítésben 256K memóriával árusítják, tapasztalati tény, hogy a legutóbb GUS tulajdonos 512 Kbyte vagy 1 Mbyte memóriával rendelkezik. Na, de biztos ami biztos, vizsgáljuk meg. Tehát már tudjuk, hogy mekkora a memória. De mit tegyünk ha kevesebb mint amennyi nekünk kellene? Nos, létezik egy megoldás, ami a GUS interpolációs képességére hagyatkozik: Küldjük el a hang csak minden második adatát, és fele akkora frekvenciával szóltassuk meg. Ekkor bár némi minőségromlást szenved el a hang, de kétszer annyi hangmemóriával gazdálkodhatunk. Ha programunk nem a legkiválóbb minőségű zenei élmények visszaadására törekszik, hanem csak aláfestésként szolgál, javallott ezt a módszert használni. Megjegyzem egy „normál snassz mezei" programnál ezt egy átlag felhasználó észre sem veszi, különösen, ha figyelembe vesszük a korábban forgalomba hozott hangkártyák min őségét. Ezek után jöhet a GUS memória feltöltése. Természetesen a GUS-t is lehet DMA lejátszásra (az a bizonyos mixelgetős technika) használni. Ennek egyet-

```
; Ez a rutin a DI:SI 20 bites memóriacímre helyezi AL érteket

POKE    PROC
        push ax
        mov dx,BASE
        add dx,103h
        mov al,43h
        out dx,al
        inc dx
        mov ax,si
        out dx,al
        inc dx
        mov al,ah
        out dx,al                       ;a cím  alsó 16
bitje
        sub dx,2
        mov al,44h
        out dx,al
        add dx,2
        mov ax,di
        out dx,al                       ;a maradék 4 bit
        add dx,2
        pop ax
        out dx,al                       ;adat elkuldése
        ret
POKE    ENDP

;A DI:SI 20 bites címről beolvas egy byte-ot az AL regiszterbe

PEEK    PROC
        mov dx,BASE
        add dx,103h
        mov al,43h
        out dx,al
        inc dx
        mov ax,si
        out dx,al
        inc dx
        mov al,ah
        out dx,al                       ;alsó 16 bit
        sub dx,2
        mov al,44h
        out dx,al
        add dx,2
        mov ax,di
        out dx,al                       ;felső 4 bit
        add dx,2
        in al,dx                        ;beolvas adat
        RET
PEEK    ENDP

;Visszaadja AX-ben a kártyán található memória mennyiséget

GUSMEM  PROC
        mov si,0
        mov di,0
MEG:    mov al,88h
        mov si,0
        call POKE
        inc si
        mov al,89h
        call POKE
        dec si
        call PEEK
        cmp al,88h
        jne ELFOGY
        inc si
        call PEEK
        cmp al,89h
        jne ELFOGY
        add di,1
        cmp di,16
        je ELFOGY
        jmp MEG
ELFOGY: mov ax,di
        shl ax,6
        RET
GUSMEM  ENDP
```

len előnye van, nevezetesen az hogy teljesen mindegy, hogy mennyi memória van a GUS-on, mégis szólni fog. Ebben az esetben a gép memóriája szerepel audio háttértárnak. Meglehetősen nagy hírű játékprogramok (DOOM, RAPTOR) is ezt a módszert használja igaz csak a hangeffektusok képzésére. Ezt a technikát csak azoknak javasolnám akik hasonló programozástechnikával rendelkeznek, mint a fenti programok szerzői. Tehát lássunk 2 példát a GUS memória feltöltésére. Az első sima I/O műveletekkel tölti fel a GUS-t:
A DX regiszterben a hang számát küldjük el. Későbbiekben csak ezzel a számmal hivatkozunk az elküldött adatmennyiségre. Csupán a teljesség kedvéért és „profibbak" örömére következzen egy DMA-as feltöltő rutin:

```
;Ez a rutin az ES:BX memória címtől CX darab byteot küld el
; DI:SI-tól kezdve a GUS memóriába.
;CHANNEL=A megfelelő DMA csatorna (0-3)
;MEGA=kiválaszt a gép memóriájában a megfelelő 1 Mbyte-os
lapot
;0=0-1Mb 1=1-2Mbyte stb.
;PAGETAB DW 87h,83h,82h,81h

TODRAM  PROC                ;A GUS felprogramozása
        pusha
        push ax
        shl di,12
        shr si,4
        add di,si

    mov dx,BASE
    add dx,103h
        mov al,42h
        out dx,al
    inc dx
        mov ax,di                   ;GUS DMA-t megcímezzük a
célterületre
        out dx,ax

    mov dx,BASE
    add dx,103h
        mov al,41h
        out dx,al
    add dx,2
        mov al,00100001b            ;GUS DMA control regiszter
                                    ;felprogramozása
        pop cx
        add al,cl
        out dx,al
        popa
        call PROGDMA
        ret
TODRAM  ENDP
PROGDMA PROC            ;A DMA felprogramozása
        push cx
        mov dx,es
        mov cl,4
        rol dx,4
        mov ah,dl
        and dl,0f0h
        add dx,bx
        adc ah,0
        and ah,0fh
        pop cx          ;Adatok átkonvertálása a DMA számára

        push ax
        push dx
        dec cx
        mov ax,CHANNEL
        or al,000000100b
        out 0ah,al          ;DMA csatorna letiltása
        xor al,al
        out 0ch,al          ;DMA Flip/Flop előkészítése
        mov al,49h
        and al,11111100b
        mov dx,CHANNEL
        add al,dl
        out 0bh,al          ;DMA/Output művelet
        xor al,al
        out 0ch,al
        mov dx,CHANNEL
        shl dx,1
        pop ax
        out dx,al
        xchg al,ah
        out dx,al

        inc dx
        mov ax,cx
        out dx,al
        xchg al,ah
        out dx,al
        dec dx

        mov bx,dx
        mov dx,PAGETAB[bx]
        pop ax
        xchg al,ah
        or al,MEGA                  ;Megabyte kiválasztása
        out dx,al

        mov ax,CHANNEL
        out 0ah,al                  ;DMA csatorna engedélyezése

        ret
PROGDMA ENDP
```

A DMA-s módszer bonyolultabb, és több hibalehetőség van, de viszont gyorsabb, és a CPU is szabad. Mivel nemcsak a konvencionális memóriából, hanem XMS-ból is képes olvasni, ezért akar több megabyte hosszú digitalizált effektusokat is lehet lejátszani csupán egy 1/4 Mbyteos GUS-on. Az első módszer is eléggé gyors, de a CPU-t már leterheli. Ellenben ha egy olyan programot akarunk írni, amely egy „végtelen" (nagyon hosszú) digitalizált szöveget szólaltat meg, akkor csak az első módszert választhatjuk, hisz a lemezműveletek alkalmával a DMA megakadhat. Ha viszont a hanganyag elfér a memóriában és mást is akarunk végeztetni a CPU-val, (Például egy animáció lejátszását) akkor csak a második módszer a járható út. Miután definiáltuk a hang kezdetet és véget, meg kellene határoznunk annak lejátszási frekvenciáját, hangerejét, loopját stb.is.

```
;DI,SÍ a hangon belüli relatív mutatók a loop elejére és végére.
;CX=lejátszási frekvencia
;BL=Hangerő (0-63)
;BH=Balance (0-15)
;DX=hang száma
THINGSPROC
        push bx
        mov bx,dx
        mov bh,0
        shl bx,1
        mov ax,SSTARTO[bx]
        mov dx,SSTARTS[bx]
        add ax,di
        adc dx,0
        mov SLOOPBO[bx],ax
        mov SLOOPBS[bx],dx          ;loop eleje
        mov ax,SSTARTO[bx]
        mov dx,SSTARTS[bx]
        add ax,si
        adc dx,0
        mov SLOOPEO[bx],ax
        mov SLOOPES[bx],dx
        pop ax                  ;loop vége

        mov DEFVOL[bx],ax           ;Balance és hangerő
        mov DEFREQ[bx],cx           ;Frekvencia
        ret
THINGS  ENDP
```

Ez a rutin első GUS programjaim egyikéből ragadtam ki. Természetesen ki-ki saját ízlése szerint alakíthatja, vagy akár el is hagyhatja. Tapasztalatom szerint hasonló rutin alkalmazása leegyszerűsíti a hangkezelést. És most következik az igazság pillanata: a lejátszás. Az alábbi rutinhoz szükség van a GUSLOAD és a THINGS által felöltött tömbökre.

```
;BX=csatorna száma ahol megszólal a hang (0-14)
;DX=hang száma

PLAY    PROC
        mov si,dx
        shl si,1

        mov al,bl
        mov dx,BASE
        add dx,102h
        out dx,al           ;csatorna kiválasztása

        mov dx,BASE
        add dx,103h
        mov al,0ah
        out dx,al
        inc dx
        mov ax,SSTARTO[si]
        mov cx,SSTARTS[si]
        shr ax,7
        shl cx,9
        or ax,cx
        out dx,ax
        dec dx
        mov al,0bh
        out dx,al
        inc dx
        mov ax,SSTARTO[si]
        shl ax,9
        out dx,ax           ;hang kezdete

        dec dx
        mov al,04h
        out dx,al
        inc dx
        mov ax,SENDO[si]
        mov cx,SENDS[si]
        shr ax,7
        shl cx,9
        or ax,cx
        out dx,ax
        dec dx
        mov al,05h
        out dx,al
        inc dx
        mov ax,SENDO[si]
        shl ax,9
        out dx,ax           ;hang vége

        dec dx
        add bx,2
        mov al,02
        out dx,al
        inc dx
        mov ax,SLOOPBO[si]
        mov cx,SLOOPBS[si]
        shr ax,7
        shl cx,9
        or ax,cx
        out dx,ax
        dec dx
        mov al,03
        out dx,al
        inc dx
        mov ax,SLOOPBO[si]
        shl ax,9
        out dx,ax           ;hang loop start

        dec dx
        mov al,01
        out dx,al
        inc dx
        mov ax,DEFREQ[si]
        push bx
        push dx
        mov dx,0
        mov bx,44           ;a konstans 44
        div bx
        pop dx
        pop bx
        out dx,ax           ;frekvencia

        dec dx
        mov al,0ch
        out dx,al
        add dx,2
        mov ax,DEFVOL[si]
        mov al,ah
        out dx,al           ;balance

        sub dx,2
        mov al,0dh
        out dx,al
        add dx,2
        mov al,3
        out dx,al           ;stop burkológörbe (ha volt)

        sub dx,2
        mov al,09
        out dx,al
        inc dx
        mov bx,DEFVOL[si]
        mov bh,0
        shl bx,10
        mov ax,bx
        out dx,ax           ;Hangerő

        mov ax,0+32         ;+32 ha IRQ kell
        mov bx,SLOOPEO[si]
        cmp SLOOPBO[si],bx
        je NINCS
        mov ax,8
NINCS:  mov dx,base
        add dx,103h
        push ax
        mov al,0
        out dx,al
        add dx,2
        pop ax
        out dx,al           ;lejátszás elindítása

        RET
PLAY    ENDP
```

A lejátszásnak van egy hiányossága, mégpedig a „loop vége". Ha figyelmesen tanulmányozzuk a rutint, láthatjuk, hogy a hang a loop kezdetétől a hang végéig fog loopolni. Ez azért van, mert a loop már egy effektus, és ezt a GUS hardver szinten már nem tudja. A loop elejét is úgy „szimuláltuk", hogy azt a „hang kezdete" regiszterbe tettük, és az „aktuális mutató"-t pedig a hang elejére állítottuk. Amikor a GUS elért a hang végére, a „hang kezdete" pozícióra ugrik vissza, ami tehát nem más mint a loop kezdete. És itt jön az IRQ. Úgy programozzuk fel a GUS-t, hogy amikor elérte a „hang vége" pozíciót, akkor adjon egy IRQ-t. Az IRQ-n ülő rutin pedig átállítja a „hang vége" pozíciót a loop végére, és lekapcsolja az IRQ kérést. Fontos, hogy az IRQ-t lekapcsoljuk, mert ellenkező esetben minden loop végén kapunk egy felesleges kérelmet. A hang elnémítása ennél jóval egyszerűbb:

```
;AL=csatorna száma
STOP    PROC
        mov dx,base
        add dx,103h
        mov al,0
        out dx,al
        add dx,2
        mov al,00000011b
        out dx,al                    ;stop csatorna
    ret
STOP    ENDP
```

Ez egy egyszerű STOP rutin. Ha most ezen a csatornán egy új hangot játszanánk le egy kis kattanást hallanánk. Ezért ajánlatos egy lecsengető rutint betenni a STOP elé. (És egy ellenkezőt a PLAY elé). Ha ugyanis a lecsengést a stop után hívnánk meg, nem történne semmi, ugyanis burkológörbét csak aktív hangokon generál a GUS. A lekapcsolás folyamata tehát a következő: ha le kell állítani egy hangot, akkor nagyon gyorsan lecsengetjük a hangerejét. Ha elérte a nulla szintet IRQ-t ad, és akkor állítjuk meg a lejátszást.

```
;AL=csatorna száma

LECSENG PROC
        mov dx,BASE
        add dx,103h
        mov al,0dh
        out dx,al
        add dx,2
        mov al,01000011b
        out dx,al                       ;stop burkológörbe
        sub dx,2

        mov al,89h
        out dx,al
        inc dx
        in ax,dx
        mov bx,ax
        dec dx
        mov al,8
        out dx,al
        add dx,2
        mov al,bh

        out dx,al               ;az aktuális hangerő kezdet
        sub dx,2
        mov al,7
        out dx,al
        add dx,2
        mov ax,1
        out dx,al                       ;0.01 a cél
        sub dx,2
        mov al,6
        out dx,al
        add dx,2

        mov al,00111111b   ;burkológörbe meredekség
        out dx,al
        mov dx,BASE
        add dx,103h
        mov al,0dh
        out dx,al
        add dx,2
        mov al,01100000b
        out dx,al  ;Burkológörbe engedélyezése IRQ-val
        RET

LECSENG ENDP
```

Nagyon fontos, hogy a burkológörbe kezdete regiszter mindig nagyobb érteket mutasson, mint a burkológörbe végregiszter! A fenti szubrutinokból egy egyszerűbb program már összerakható. Természetesen sok dolgot (memóriafoglalás, hang betöltése, IRQ kezelés) nem részleteztem, hisz azok programnyelv függőek, és minden programfejlesztő másképp csinálná. Ezek a rutinok csupán a GUS vezérlését próbálják illusztrálni, kezdetnek beilleszthetők bárhova, de idővel úgyis mindenki a saját rutinjait fogja használni. Remélem az első lépéseket sikerült megkönnyíteni, és ezúton buzdítanám a kevésbé tapasztaltabb programozókat is, hogy próbálkozzanak meg a GUS regiszterszintű programozásával.

Vig 'Alex' Sándor

PC-ROBOT AZ ISKOLÁBAN

Beszélő számítógép segíti az oktatást

Iskolánkban az elmúlt hónapokban lehetőség nyílt a PC-ROBOT, asztali PC-n működő beszédszintetizátor kipróbálására, amelyhez a fejlesztők (NIKOL ELEKTRONIKA) egy speciális, iskolai iskolai oktatóprogramot is kifejlesztettek. A beszélő számítógép teljesítménye számomra -- és más tanárok számára is -- lenyűgöző tudományos eredmény. A beszédszintetizátor ugyanis lehetővé teszi, hogy bármilyen hosszú, magyarul leírt szöveg beszéd formájában megszólaljon. És mindez érthető, magyar kiejtéssel! Nem hiszem, hogy hangsúlyoznom kell e speciális, hazai fejlesztés jelentőségét, ugyanis erre a teljesítményre (magyar beszéd korlátok nélkül) semmilyen általános hangkártya, sem szoftver nem teszi képessé a számítógépet. A magyarul beszélni tudó számítógép többek között új színt hozhat a 20. századi oktatásba. Iskolánkban nagy érdeklődéssel próbáltuk ki a rendszerrel szállított különleges oktató programcsomagot, amely beszédtámogatással is segíti, színesebbé tenni az oktatást, gyakorlást. A programmal való első ismerkedéskor mind a gyerekek, mind pedig a tanárok körében nagy tetszést aratott, hogy a számítógép életre kelt, azaz nem némán dolgozott, hanem beszélt. A beszéd szolgáltatással sok esetben felszabadul az az energia, hogy a képernyőt kell állandóan nézni, mivel szóban is elhangzik például a feladat, a közlés, az eredményre utaló megjegyzés. Széles programválaszték áll a tanár (és a tanuló) rendelkezésére, így véleményünk szerint a legtöbb tantárgy oktatásába be lehet kapcsolni a PC-ROBOT játékos beszélő programjait. A számítógéppel való játék során a gép beszél a diákhoz, így a gyerekek szinte észrevétlenül sajátítják el a gyakorolt anyagrészt. A számítógép türelmes, toleráns, ugyanakkor szigorú is az elbírálásban. Beszéddel dicsér, de figyelmeztet is a hibákra. Igazi kommunikációs társ. A játék során mindenki a saját tempójában oldhatja meg a feladatokat, így a begyakorlás, a gondolkodási készség fejlesztése egyénre szabottan történhet. A játékos oktató-programok közül a DIKTI, SZTORI, SZOTAG, FELSOROL nyitott szerkezetűek, ami azt jelenti, hogy a tanár (a felhasználó) saját maga is (ha akar) meghatározhat feladatokat , azok tartalmi felépítését kidolgozhatja (írhat kérdéseket, válaszokat, dicsérő szöveget, figyelmeztetéseket stb.), majd a programot az általa kidolgozott feladatokkal futtathatja, használhatja oktatásra, gyakoroltatásra (lásd később a példákban).

Az iskolai felhasználásban igen fontos az a tény, hogy nincs korlátozva, hogy mit mondjon el a gép. A nyitott szerkezet biztosítja, hogy sokféle felhasználási területnek megfelelően külön gyakorlási anyag készíthető a programcsomag egyes elemeire (pl. általános iskolai anyag, kisegítő iskolai feladatok, speciális, továbbá rehabilitációs szakintézmények oktatási anyagai, egyénre szabott feladatok otthoni gyakorlásra).

A PC-ROBOT rendszerrel a következő, oktatásban, gyakorlásban használható programokat kapta meg az iskola:

Általános programok:

-- **KIMOND** - a klaviatúrán begépelt szöveget a gép alaphelyzetben kimondja (játékos írásgyakorlásra használjuk; ismerkedés a klaviatúrával)

-- **OLVAS** - a számítógépbe írt magyar szöveget (ASCII szövegfájlt) felolvassa, illetve a felhasználó által készített énekeket elénekli.
-- **KITALÁL** - számkitalálós játék. A gép gondol egy számra és azt kell kitalálni. A játék során a gép beszél a játékoshoz.
-- **VOXEDIT** - beszéd és ének szerkesztõ program (játékra, kreatív munkára használjuk)
-- **GRAFIKA** - képekkel illusztrált felolvasásokat lehet készíteni a programmal, vagyis a gyerekek által rajzolt (PCX formátumú) színes képek megjelení thetõk és alájuk szövegek olvastathatók fel. (játékra, kreatív munkára használható).

Nyitott szerkezetű, szövegfájlból dolgozó kérdezz--felelek programok:

-- **SZTORI** - a gép egy -- a tanár által megírt -- feladat szövegét olvassa fel és ezután -- a tanár által megírt -- kérdéseket olvas fel az elhangzott szöveg tar talmára vonatkozóan. Beszédmegértési készség, memorizálási képesség felmérésére, továbbá minden olyan játékra használható, ahol egy témával kapcso latban több válaszlehetõségbõl legalább egyet el kell találni. A szöveganyag tetszés bõvíthetõ, alakítható.
-- **DIKTI** - helyesírást gyakoroltató program. A gép diktál, a tanuló írásban válaszol és a gép szóban értékel. A szöveganyag bõvíthetõ.
-- **SZÓTAG** - elválasztást, szótagolást játékosan gyakoroltató program. A gép diktál és a hallott szöveget az elválasztás szabályai szerint kell leírni. Más hasonló feladatokra is felhasználható. A szöveg bõvíthetõ.
-- **FELSOROL** - a gép beszéddel kérdez és a kérdésre adható válaszokat írásban kell felsorolni (pl. rokon értelmű szavak keresése, földrajzi nevek felsorolása, állatnevek gyűjtése, stb.). Maximum 7 válasz adható meg egy kérdésre. A szöveganyag bõvíthetõ.

A fenti 4 kérdezz--felelek program mindegyike szövegfájlokból dolgozik. A szövegfájlokban lehet megadni, hogy mit mondjon el, illetve mit kérdezzen a gép és milyen válaszokat várunk el a játékostól a feladat megoldása során. Ilyen szövegfájlok bárki által készíthetõk a legegyszerűbb szövegszerkesztõvel. Példaképpen nézzük meg a FELSOROL játék egyik feladatának szövegfájlját! A fájl neve: foldrajz.dkt

```
$ Három földrajzi kérdést fogsz hallani.
$ Mindegyikre felsorolással kell válaszolni.
$ Ügyelj a helyesírásra.
# Sorold fel a nevezetes szélességi köröket.
@ Északi-sarkkör
@ Ráktérítő
@ Egyenlítő
@ Baktérítő
@ Déli-sarkkör
@
@
# Gyűjtsd össze Magyarország szomszédait.
@ Szlovákia
@ Ausztria
@ Szlovénia
@ Horvátország
@ Szerbia
@ Románia
@ Ukrajna
# Sorold fel az Északi-középhegység részeit.
@ Börzsöny
@ Cserhát
@ Mátra
@ Bükk
@ Zempléni-hegység
@
@
```

Észrevehetõ, hogy a fájlban háromféle szöveget különböztetünk meg. A $ jellel kezdõdõ sorokat a gép mindig elmondja a feladat indításánál. Ez a bevezetõ szöveg.
A # jellel kezdõdõ sor mindig a feladat megfogalmazása. Ez kérdés is lehet (pl: Melyek az Északiközéphegység részei?).
A @ jellel kezdõdõ sorokban adjuk meg a várt helyes válasz(oka)t. Itt maximum 7 válaszra biztosít helyet a program. A fenti példa is bizonyítja, hogy az oktató program által használt szövegfájlok szerkezete rendkívül egyszerű, így lehetõséget biztosít még a számítástechnikában kevéssé járatos felhasználó (tanár, diák) számára is új szövegfájlok létrehozására. Matematikai gyakorlást segítõ programok
-- SZÁMOK: 1x1, fejszámolás (összeadás, kivonás, szorzás) gyakorlása. A gép diktálja a feladatot és az eredményt megadott idõn belül be kell írni a klaviatúrán. A feladat típusát és a megoldási idõt a tanuló állíthatja be. A programok egyszerű változatai egyjegyű számokkal dolgoznak
(pl. mennyi 2+4, 8-3, 9x8?), a komolyabb fejszámolási műveletekhez pedig már kétjegyű számokat is alkalmaznak (pl. mennyi 5x14, 16+68, 24-13, 16x12?).

Kitekintés más nyelvekre:

Külön érdekessége még a rendszernek, hogy nemcsak magyarul, de (korlátozott formában) idegen nyelven is megszólal a gép, vagyis az alapszintű idegen nyelv tanítást is támogatja. Erre három program is használható:
-- **ANGOL , NÉMET --** angol, illetve német tõszámnevek, sorszámnevek, valamint a hónapok neveinek kiejtése és írása. Tanuló, gyakorló és teszt üzem módban használható a program, akár egyéni gyako rlás formájában is (a program eredeti angol, illetve német kiejtéssel beszél).
--FRANCIA - francia tõszámnevek kiejtése és írása eredeti francia kiejtéssel.

Programozási feladatok támogatása:

Végezetül ki kell emelnem azt a rendkívül hasznos elgondolást, hogy a fejlesztõk közreadják
a PC-ROBOT legtöbb programjának forráskódját is. Így a programozást tanuló, illetve abban már
jártas diákoknak is adhatók érdekes feladatok, például más beszélõ programok fejlesztése. Illusztrációként lásd a KITALÁL - számkitalálós beszélõ játékprogram teljes forrását mellékelve.
Fontos, hogy a fejlesztõk az iskoláknak az oktató programcsomagot -- oktatási támogatásként -- ajándékba adják, amennyiben a PC-ROBOT rendszert az iskola megvásárolja. További, részletes felvilágosítást és segítséget a rendszer használatához, a következõ telefonszámon
kaphatnak az iskolák: 1557-122/218 mellék

Szilágyi Erzsébet
pedagógus
Kossuth Lajos Angol tagozatos Általános Iskola
1221 Budapest, Kossuth Lajos u. 22.

FELRAK

Mini installáló program, a mini Linuxhoz

A lemezmellékletünkön található egy Linux-os installáló program, amivel mostantól a havonta érkező anyagokat telepíthetjük. Megígértük, hogy megmutatjuk a programot „belülről" is. Következzék hát először a forráskód, utána pedig a részletes magyarázat.

```
1  #!/bin/bash
2  # felrak - programcsomag telepítõ a mini-Linux-hoz
3  # Copyright 1995 Bálint Nagy Endre
4  # Használd egészséggel!
5  # lássuk, van e mit telepíteni!
6  ls /DOS/linuxins/*.tgz 2>/dev/null >/tmp/$$
7  [ ! -s /tmp/$$ ] && { echo nincs mit felrakni ; exit 1 }
8  for i in /DOS/linuxins/*.tgz
9  do
10   # van leírása is ?
11   if [ -f ${i%.tgz}.lsm ]
12   then
13     cat ${i%.tgz}.lsm
14   else
15     # sajna csak a nevét tudjuk
16     echo -n ${i##*/}
17   fi
18   echo -n " telepítsem?"
19   read val
20   if [ "$val" = i -o "$val" = I -o "$val" = y -o "$val" = Y ]
21   then
22     # egy kis takarítás
23     rm -f ${ROOT}/install/doinst.sh
24     tar xvzf $i -C ${ROOT-/}
25     if [ -f ${ROOT}/install/doinst.sh ]
26     # ha érkezett doinst.sh, végrehajtjuk
27     then
28       cd ${ROOT-/}
29       bash ${ROOT}/install/doinst.sh
30     fi
31     echo -n "a telepítõ készletet töröljem?"
32     read val
33     if [ "$val" = i -o "$val" = I -o "$val" = y -o "$val" = Y ]
34     then
35       rm -f $i
36       [ -f ${i%.tgz}.lsm ] && rm -f ${i%.tgz}.lsm ]
37     fi
38   fi
39 done
```

Ezt be lehet gépelni a 'deco' szövegszerkesztõjébõl is (természetesen sorszámok nélkül), de persze a lemezen is megtalálható. Lássuk elõször a deco-s beírást! A programnak célszerű a /etc/felrak nevet adni. Ebbe az alkönyvtárba csak a root felhasználó tud írni, így célszerű root-ként belépni.
Ha megpróbáljuk létrehozni a fájlt, azonnal akadályba ütközünk: a deco-ban csak létezõ fájlt lehet szerkeszteni. Ezért elõször kell csinálni valahogy egy üres fájlt. A következõ parancs segít ebben:

```
echo > /etc/felrak
```

Ezután már tényleg jöhet a deco! Az ékezetes betűkre a bash kifejezetten allergiás, gépeljük ékezetek nélkül - úgysem volt még szó arról, hogy lehet az ékezetes betűket elõcsalogatni, majd legközelebb sor kerül erre is.
Azok, akik inkább megtakarítanák a fenti program begépelését, csinálják végig a lemezmellékleten található Linux-os programok telepítését úgy, ahogy azt korábban (ennek a számnak egy másik cikkében) leírtuk. Végre kellett hajtanunk az alábbi parancsokat is:

```
# cp /DOS/linuxins/felrak /etc
# chmod 744 /etc/felrak
```

Az első paranccsal nincs is semmi bajunk: átmásolja a megadott fájlt a /etc alkönyvtárba. Sokkal érdekesebb a második sor. Ez utóbbi parancs teszi végrehajthatóvá a prog-ramot, tehát ha a fáradságosabb utat választottuk (begépeltük a deco-ban), akkor is csináljuk meg! Nézzük, mit is eredményezett a szabványos „chmod" UNIX parancs:

```
# ls -l /etc/felrak
-rwxr--r--   1 root   root    959 Feb 27 12:49 /etc/felrak
```

azaz, végrehajtani csak root-ként lehet, olvasni pedig mindenki tudja. És most már tényleg magáról a programról: Az első sor megadja a Linux-nak, hogy a bash tudja végrehajtani a programot. A megjegyzések után a 6. sor összegyűjti az installálható programokat. A 2>/dev/null szekvencia az esetleges hibaüzeneteket (stderr) eldobja. Az eredményt a /tmp alkönyvtárban egy átmeneti fájlban tárolja. A $$ helyére a bash mindig a futó program ún. processz-azonosítóját helyettesíti. Erre garantált, hogy egy időben két programnak nem lehet ugyanaz a száma. A 7. sor megvizsgálja, találtunk-e valamit. Ha nem, egy üzenet után a programból kilépünk. A szögletes zárójelbe zárt utasításrészletet a bash a test részeként fogja fel.. A felkiáltójel negálja a vizsgált feltételt, a -s az argumentumaként megadott fájlt vizsgálja, és igaz (0) értéket ad vissza, ha a fájl létezik, de nem üres. Az && utáni rész csak akkor fut le, ha az első feltétel igaz volt. Ha hamis lett volna, akkor a feltétel már garantáltan hamis eredményt ad (logikai és miatt), ezért a bash továbblép. Az && konstrukció ezért egy rövidített if-ként használható, aminek csak then ága van. Mivel csak egy parancsból állhat, ezért a kap-csos zárójelek használatával összevonjuk egy blokká a hibaüzenet kiírását és a kilépést. A pontosvesszővel elválasztjuk a két parancsot. (Egy bash parancssor általában sor végéig vagy az első pontosvesszőig tart.) A 8. sor egy ciklust nyit. A for utasítás általános alakja

```
for változó in érték1 [érték2 ...]
do
  utasítás
  ...
done
```

Természetesen a fentebb elmondottaknak megfelelően lehet pontosvesszőt is használni, ekkor

```
for változó in érték1 [érték2 ...]; do utasítás;...; done
```

UNIX-ban a parancsértelmező (nálunk a bash) dolga a „joker"-t tartalmazó fájlnevek behelyettesítése, tehát a for parancs már nem látja a *-ot, csak a behelyettesített file-neveket. A 11-17 sorokban kiírjuk a leírását az installálható programcsomagnak, illetve leírás hiányában a file nevét. Ebben látható két újabb változó-behelyettesítési trükk, a ${változó%minta} levágja a változó értékének a végéről a megadott mintát, míg ha a % helyett # áll, akkor az elejéről. Ha a % vagy # jelet megduplázzuk, akkor a leghosszabb illeszkedő szövegrészt vágja le, egyébként a legrövidebbet.
A következő példákban:

```
temp=abababcdcdcd
str=/usr/lib/libc.a
```

Így például:

```
${temp%c*d}       ( abababcdcd
${temp%%c*d}      ( ababab
${temp#a*b}       ( ababcdcdcd
${temp##a*b}      ( cdcdcd
${str%/*}         ( /usr/lib
${str%%/*}        ( ""      ez üres string!!!
${str#/*}                   ( /lib/lic.a
${str##/*}        ( libc.a
```

A 18. sorban az echo bash parancs -n opciójának használatát látjuk, hatására nem kerül újsor karakter a kiírt szöveg után. Ezután a read belső paranccsal beolvassuk a választ a val változóba. A sor végéről az újsor karaktert a read levágja.
A 20 sorban a választ vizsgáljuk. A -o operátor a két oldalán álló logikai értéken a logikai vagy műveletet végez, az = operátor a szövegösszehasonlítást végzi. A változó kihelyettesítést azért kellett idézőjelbe tenni, hogy az üres értékű változó megmaradjon üres értékű argumentumnak, enélkül hibaüzenetet kapnánk üres (csak RETURN-ból álló) válasz esetén (ugyanis az if [= I] értelmetlen). A -o a logikai vagy kapcsolatot jelenti. Pozitív válasz esetén letöröljük az előző telepítésből fennmaradt "utánigazító" parancsot (23. sor). A -f kapcsoló hatására az rm szó nélkül töröl. A 24. sorban kibontjuk a csomagot a tar paranccsal.
A ${ROOT-/} kifejezés "/"-nek értékelődik ki, ha a ROOT változónak nem adtunk értéket, vagy értéke üres. Ennek később nagy lesz a szerepe, ha esetleg megpróbálunk áttérni a saját partícióban lakó Linux telepítésére. A programcsomag installálását saját "utánigazító" programjának végrehajtásával tesszük teljessé (25.-30. sorok).
Végül igény esetén takarítunk. Itt a -f filenév kifejezés igaznak értékelődik ki, ha a megadott fájl létezik és tényleg fájl (nem alkönyvtár). A "telepítsem" if parancs zárása után a done paranccsal zárjuk a ciklust is. Az if, then, else, fi, for, do és done parancsok mind belső parancsok, ahogy az elvárható. Ha valakit érdekel a shell (pl. bash) programozás, az nézze meg az FLCP.BASH -ról szóló cikket is, abban is sok érdekeset találhat. Jó shell programozást!!

ASCII grafika

Újra divatba jöttek a számítástechnika hõskorából ismert az ASCII grafikus képek, napjainkban az Internet hálózatokon küldözgetik. Az alább közölt programmal bárki átalakíthatja saját képeit.A működés alapelve: az átalakításra váró grafikus képet elõször valamilyen módon 256 szürke árnyalatúra illetve .RAW formátumúra kell hozni. Ezeket a feladatokat nem valósítottuk meg, mivel több grafikai program is rendelkezik ilyen funkcióval. A képpontok és az ASCII karakterek megfeleltetését egy elõre definiált maszk (toa_def.txt) segítségével végezzük. Elsõ lépésben megkeressük a grafikus kép legsötétebb ill. legvilágosabb pontjait, ezen pontok által meghatározott intervallumot fogjuk felosztani a maszkban szereplõ karakterek száma szerint. A maszk elemeinek száma 2-256-ig változhat. Minden egyes pixelnek egy karakter fog megfelelni, amelyet a maszkból rendelünk hozzá, a szürkeségének arányában. A maszknak tehát úgy kell felépülnie, hogy a benne szereplõ karakterek világosság szerint növekvõ sort alkossanak. A maszk határozza meg leginkább a keletkezõ ASCII kép minõségét. Nem csak a világosság fontos, hanem a karaktert felépítõ pontok egyenletes eloszlása is. Azt tapasztaltuk, hogy a maszk elemszámának növelése nem minden határon túl javítja a képminõséget. Kiindulásként közöljük az általunk legjobbnak ítélt karaktersorozatot:

.`:;^=+<)%*tfCYFVXE#R0WQ

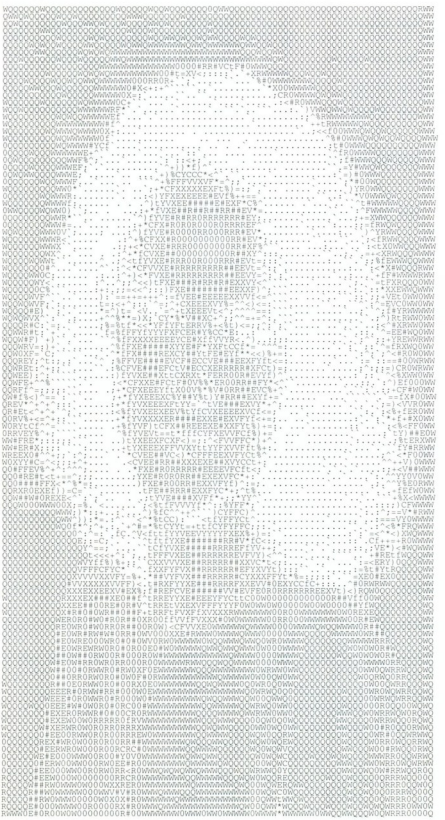
A program használata:
toascii.exe grafikusképfile.raw [ASCIIképfile.txt] maszk.txt]
Ha világos háttéren sötét karakterekkel dolgozunk /pl. nyomtatás/ , akkor válasszuk az inverz generálási módot. A képek EGA-line üzemmódban látszanak a legjobban. Nyomtatáskor ügyelni kell a sorköz beállítására, nehogy a kép elnyújtott legyen. Sok sikert a kísérletezéshez! Minden észrevételt szívesen fogadunk !

nemethj@mszi.pmmf.hu , aronm@mszi.pmmf.hu ,
tothp@mszi.pmmf.hu

```
/*  toascii.c -   raw file-ok konvertalasa ASCII-ba */

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

void cserel (unsigned short *);
char ch[256];
short finom;

int main (int argc,char *argv[])
{
        FILE *txt,*infile,*outfile;
        unsigned short xhossz,yhossz,palhossz;
        unsigned char adat,max=0,min=255;
        int c,inverz;
        int x,y;

        if (argc<2 || argc>4)
        {
        puts("Hibas parameterlista");
        puts("Hasznalat: toascii.exe infile.raw [outfile.txt] [betudef.txt]");
        exit(-1);
        }
```

Fent az eredeti 80x123 szürke pontból álló RAW formátumú mintakép.

Lent a konverzió erredménye.



```
        if (argc<=3) txt=fopen ("toa_def.txt","rt");
        else                    txt=fopen (argv[3],"rt");
        if (txt==NULL)
        {
                puts ("Nem létezik a definiciós file !!!");
                exit (-1);
        }
        finom=strlen(fgets(ch,256,txt));
        fclose(txt);
        infile=fopen(argv[1],"rb");
        if (infile==NULL)
        {
                puts ("Nem létezik az infile !!!");
                exit (-1);
        }
        if (argc==2)
                outfile=fopen("untitled.txt","wt");
        else
                outfile=fopen(argv[2],"wt");
        if (outfile==NULL)
        {
                puts ("Az outfile nem nyitható meg!!!");
                exit (-1);
        }
        fseek(infile,8L,SEEK_SET);
        fread(&xhossz,2,1,infile); /* A grafikus kép szélessége */
        cserel(&xhossz);
        fseek(infile,10L,SEEK_SET);
        fread(&yhossz,2,1,infile); /* A grafikus kép magassága */
        cserel(&yhossz);
        fseek(infile,12L,SEEK_SET);
        fread(&palhossz,2,1,infile); /* Palettahossz */
        cserel(&palhossz);
        fseek(infile,0x20+palhossz*3,SEEK_SET); /* Bitminta eleje */
        do
        {
                puts("Inverz <Igen,Nem> ?  ");
                c=tolower(getch());
        }
        while (c!='n' && c!='i');
        inverz= (c=='i') ? 1 : 0;
        for (y=0;y<yhossz;y++)   /* Legsötétebb ill. legvilágosabb */
        for (x=0;x<xhossz;x++)   /* pontok megkeresese             */
                {
                adat=fgetc(infile);
                if (adat>max)
                max=adat;
                if (adat<min)
                min=adat;
                }
        fseek(infile,0x20+palhossz*3,SEEK_SET);
        for (y=0;y<yhossz;y++)       /* Konvertálás */
        {
        for (x=0;x<xhossz;x++)
        {
        adat=fgetc(infile);
        if (inverz)
        fputc(ch[finom-((adat-min)*finom)/(max-min+1)-1],outfile);
        else
        fputc(ch[((adat-min)*finom)/(max-min+1)],outfile);
        }
        fputc('\n',outfile);
        putchar('*');
        }
        puts("");
        fclose(outfile);
        fclose(infile);
        return 1;
}

void cserel (unsigned short *adat)  /* A ket byte-os adat alsó és felsõ  */
{                                   /* 8 bit-jet felcseréli              */
        unsigned short tmp;
        tmp=(*adat)>>8;
        tmp|=(*adat & 0xff)<<8;
        *adat=tmp;
}
```

Screen Machine

Cikksorozatunk első fejezetében összefoglaltuk a C-ben hívható SM II funkciókat és egy-két alapvető rutinnal illusztráltuk is azokat. A 2. részben a Windows MCI kapcsolat lehetőségével ismertetjük meg olvasóinkat.

A Screen Machine-t, mint szabványos overlay kártyát vezérelhetjük a Windows multimédia API MCI parancsaival, de az alapvető MCI utasításokon túl további utasítások bevezetése, illetve az alaputasítások paraméterkészletének kibővítése révén a kártya összes lehetősége elérhető a médiavezérlő felületen keresztül. Az utasítások elküldésére használhatjuk az mciSendString, illetve az mciSendCommand függvényt (MMSystem.dll). Az előbbi azért előnyösebb, mert ha a kártyához tartozó program bővülése miatt a használt adatstruktúrák megváltoznak, nem kell az általunk írt programokat újrafordítanunk.
A továbbiakban ismertetjük azokat az MCI parancsokat, és ezek fontosabb paramétereit, amelyekkel vezérelhetjük a Screen Machine kártyát, a bővítésekhez magyarázatot fűzve.

CAPABILITY:
szintaxisa: capability *eszköz paraméter*

standard paraméterek : **can eject; can freeze; can play; can record; can save; can stretch; compound device; device type; has audio; has video; uses files;**

új paraméterek:

can switch fields : 'true' értékkel tér vissza, ha a vezérelt eszköz képes a páratlan, ill. páros félképekre kapcsolni.

has dialog *box* : 'true' értékkel tér vissza, ha az eszköz használja a *box* által meghatározott dialógusdobozt.

A *box* a következő konstansok egyike lehet :

color	színbeállító dialógusdoboz
source	forrásbeállító dialógusdoboz
display	képbeállító dialógusdoboz
open	fájlmegnyitó dialógusdoboz
save	fájlmentő dialógusdoboz
audio	hangbeállító dialógusdoboz
chroma	színességbeállító dialógusdoboz (csak SM II esetén)
effect	effektusbeállító dialógusdoboz (csak SM II esetén)

has balance : 'true' értékkel tér vissza, ha az eszköz támogatja a hangcsatorna- kiegyenlítést.

has bass : 'true' értékkel tér vissza, ha az eszköz támogatja a hang mélykiemelésének beállítását.

has brightness : 'true' értékkel tér vissza, ha a világosság értéke beállítható.

has composite color : 'true' értékkel tér vissza, ha az eszköz kompozit színkódolást használ

has contrast : 'true' értékkel tér vissza, ha a kontraszt értéke beállítható.

has hue : 'true' értékkel tér vissza, ha a színezet értéke állítható.

has intensity : 'true' értékkel tér vissza, ha az intenzitás értéke állítható.

has interlace : 'true' értékkel tér vissza, ha az eszköz támogatja az interlace módú (két félképes) megjelenítést.

has key color : 'true' értékkel tér vissza, ha color keying (színlyukasztás) használható az eszköz segítségével.

has noise filter : 'true' értékkel tér vissza, ha az eszköz képes a zajszűrés vezérlésére.

has rgb : 'true' értékkel tér vissza, ha a vörös- zöld- kék arány értéke állítható.

has saturation : 'true' értékkel tér vissza, ha a színtelítettség értéke állítható.

has scale : 'true' értékkel tér vissza, ha a képméret ill. képméretarány változtatható.

has sharpness : 'true' értékkel tér vissza, ha a képélesség állítható.

has source ntsc : true' értékkel tér vissza, ha a forrás lehet NTSC videojel.

has source pal : 'true' értékkel tér vissza, ha a forrás lehet PAL videojel.

has source secam : 'true' értékkel tér vissza, ha a forrás lehet SECAM videojel.

has treble : 'true' értékkel tér vissza, ha a hang magaskiemelése vezérelhető.

has volume : 'true' értékkel tér vissza, ha a hangerősség vezérelhető.

has xoffset : 'true' értékkel tér vissza, ha a kép x irányban eltolható.

has yoffset : 'true' értékkel tér vissza, ha a kép y irányban eltolható.

maximum audio input : a lehetséges hangbemenetek számával tér vissza.

maximum input : a lehetséges vidobemenetek számával tér vissza.

OPEN:

szintaxisa : open *eszköz paraméter*

standard paraméterek : **alias *device_alias*; parent; shareable; style *type*; type *dev_type***

új paraméter :

product *pro_type* : beállítja az overlay-kártya típusát. a *pro_type* lehet:

sm	Screen Machine Classic.
smii_8	8 bites Screen Machine II.
mm	Movie Machine (Pro)

CLOSE:

szintaxisa : close *eszköz* vagy close all

SET:

szintaxisa : set *eszköz paraméter*

standard paraméterek : **audio all off; audio all on; audio left off; audio left on; audio right off; audio right on; video off; video on**

új paraméterek:

audio balance to *val*; audio bass to *val*; audio input to *val*; audio treble to *val*; audio volume to *val*; bandpass to *val*; brightness to *val*; clipping off; clipping on; composite color off; composite color on; contrast to *val*; fields to *val*; hue to *val*; input to *val*; intensity to *val*; interlace off; interlace on; key color to *val*; noise filter to *val*; rgb blue to *val*; rgb green to *val*; rgb red to *val*; saturation to *val*; scale to *val*; sharpness to *val*; source to ntsc v. pal v. secam; xoffset to *val*; yoffset to *val*

A *val* a beállítandó értéknek megfelelő szám. A paraméterek jelentését ld. a **capability** parancsnál !

STATUS:

szintaxisa : status *eszköz paraméter*

standard paraméterek: **media present; mode; ready; stretch; window handle**

új paraméterek : megegyeznek a **set** parancséival, természetesen a *val* paraméter nélkül.

INFO:

szintaxisa : info *eszköz paraméter*

standard paraméterek : **file; product; window text**

PUT:

szintaxisa : put *eszköz paraméter*

standard paraméterek : **video; video at *rectangle*; frame; frake at *rectangle*;source; source at *rectangle*; destination; destination at *rectangle***

új paraméter:

frame best *size* : fix ablakméretet állít be. A *size* lehet

full a forrás teljes méretében jelenik meg a kép
half a forrás- : képméret 2:1
quarter a forrás- : képméret 4:1

WHERE:

szintaxisa : where *eszköz paraméter*

standard paraméterek : **video; frame; source; destination**

új paraméterek :

best full : a teljes képmérettel tér vissza
best half : a fél képmérettel tér vissza
best quarter : a negyed képmérettel tér vissza
image *fájlnév* : a fájlban található kép méretével tér vissza

LOAD:

szintaxisa : load *eszköz fájlnév* [*paraméter*]

standard paraméter : **at *rectangle***

SAVE:

szintaxisa : save *eszköz fájlnév* [*paraméter*]

standard paraméter : **at *rectangle***

új paraméterek :

best *size* : fix méretet ad az elmentendő képnek a *size* értéke a **put** parancsnál látottakhoz hasonló.
to *rectangle* : megadja az elmentett kép mértét.
use row correction : bekapcsolja a sor korrekciót a mentéshez.
with *argument* : a fájlformátumra vonatkozó paramétereket adhatjuk meg.

FREEZE:

szintaxisa : freeze *eszköz paraméter*

standard paraméter : **at *rectangle***

UNFREEZE:

szintaxisa : unfreeze *eszköz paraméter*

standard paraméter: **at *rectangle***

WINDOW:

szintaxisa : window *eszköz paraméter*

standard paraméterek : **fixed; handle default; handle widow_handle; state *st*; stretch; text caption**

új paraméter : **handle from active:** A video overlay az aktív ablakban jelenik meg.

DIALOG : (új parancs)

szintaxisa : dialog *eszköz paraméter*

paraméterek :

open *box* : megnyitja a *box* által specifikált dialógusdobozt. A *box* lehetséges értékeit ld. a **capability** parancs **has dialog box** paraméterénél!
close *box* : lezárja a megadott dialógusdobozt.
popup menu off : letiltja a menüt.
popupmenu on : engedélyezi a menüt.

Végül tekinsünk két példát a parancsok használatára. A következő parancssor létrehoz a képernyőn egy overlay ablakot, és bekapcsolja az élő videoképet. Ezt követően megállítja a képet, és elmenti TGA formátumban 16 színnel. Ezután ismét elindítja az élőképet, majd lezárja az overlay eszközt.

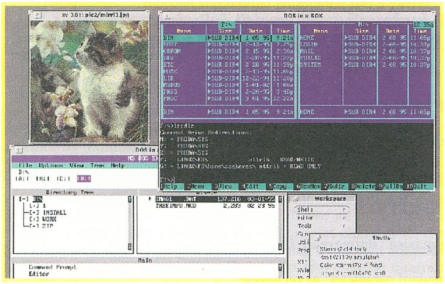
```
open overlay alias sm
window sm handle default
set sm video on
freeze sm
save sm C:\capture.tga with -a16
unfreeze sm
close sm
```

A második példa egy előzőleg létrehozott hwnd leírójú ablak egy megadott helyére tölti vissza az elmentett képünket :

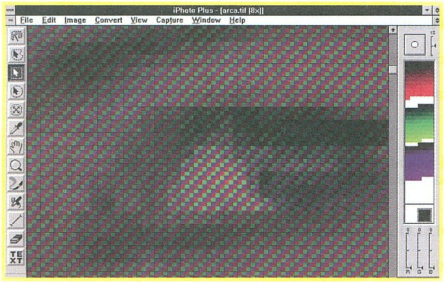
```
open overlay alias sm
window sm handle hwnd fixed
put sm frame at 100 100 200 150
load sm C:\capture.tga
close sm
```

Lernyei Csaba
ALLEGRO

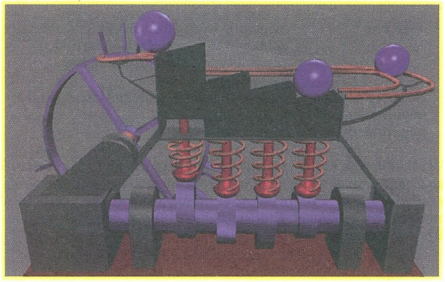
Terveink



Linux építő sorozatunk következő részében a DOS programok futtatására alkalmas emulátort adunk.



A 256 színű VESA üzemmódok ismertetése kapcsán egy trükkel 16 millió színű képeket jelenítünk meg.



Az animáció sorozatban az FLC file felépítése és lejátszása következik.

Mini programozói versenyünk legjobb megoldásait következő számunkban ismertetjük.

FORRÁSKÓD

A programozók lapja

Megjelenik havonta 3.5"-os mágneslemez melléklettel

Alapító főszerkesztő:
Nagy Sándor

Felelős kiadó:
Nagy Sándorné

Nyomdai munkák:
Révai Nyomda Kft.
F.V.: Bánáti László
T.sz.:2832.

ISSN : 1218-6414

Szerkesztőség:
1085.Budapest
Népszínház u 31.I/4/a.

Terjesztés:
Hírker Rt.
Nemzeti Hírlapkereskedelmi Rt.
Extrahír Rt.
Alternatív terjesztők

Hirdetésfelvétel:
Bauer Csilla
Bozsó Katalin
Filadelfi Zsuzsanna
Tel/Fax : 134-1273.

Előfizetés:

Egész évre:	3360.- Ft.
Fél évre :	1990.- Ft.

OTP Bank Rt. XVII.ker fiók
541-003578-3
NASA Kft. számláján.

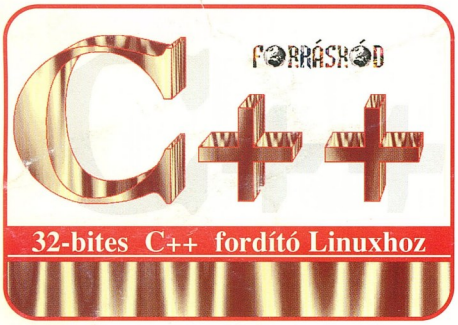
Levélcímünk:

1656. Budapest
Pf. 16.

ÁRA : 398 Ft. 3. SZÁM 1995. Március - Április

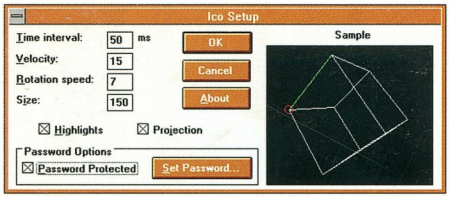
FLI lejátszó
pascal nyelven

Lemezmellékletünk ANI alkönyvtárában az FLI formátumú animációs file-ok lejátszásához adunk egy pascal nyelvű példaprogramot, a lapban ismertetjük az FLI formátum felépítését, kezelését .



Linux
fejlesztőeszköz

Linux építő sorozatunkban egy teljes és valódi 32-bites C, és C++ fordítót telepítünk linuxunk alá, és elkezdjük vele az ismerkedést. A DOS formátumú lemezek kezeléséhez szerszámot adunk.



WINDOWS
Képernyővédő

Windows programozói rovatunkban most a haladóknak írunk. Egy képernyő-védő készítése kapcsán több trükköt, mesterfogást ismerhetnek meg.
Az About box-ot feltétlenül próbálja ki !!!

Assembler
játékprogram

Végre közöljük teljes terjedelemben az assemblerben írt játékprogram forráslistáját, és futtatható változatát. A lemezen tömörítve található, másoljuk át, indítsuk el a hubasm.exe nevű file-t!