1995. JANUÁR ÁRA: 298 Ft.

FORRÁSKÓD

A programozók lapja

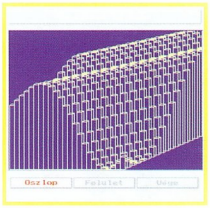
1.

Start!

Indul a Forráskód, az első magyarnyelvű, 64 oldalas, 3.5"-es lemezmellékklettel megjelenő, színes, programozói magazin. Mindazoknak szól, akiket nem elégítenek ki a kész programok, hanem saját kezükbe akarják venni a számítógép irányítását. Segítjük a programnyelvek oktatását, támogatjuk a hazai szoftverfejlesztőket.

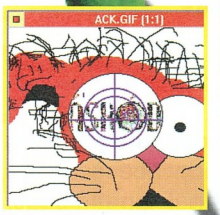
Előfizethető a kiadónál, a mellékelt csekken. Megjelenik: havonta.

60. A Forráskód lemezmellékletén a hónap kódjain túl egy érdekes animációt, képeket és három ajándék szoftvert talál.

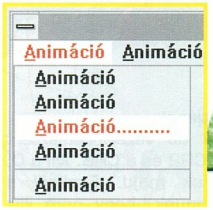


44. Programjaink önálló arculatának kialakításában segít cikksorozatunk. Ebben a számban grafikus gombmenüt és élő grafikont készítünk.

12. Középhaladóknak szóló assemler sorozatunk témája egy komplett játékprogram elkészítése.

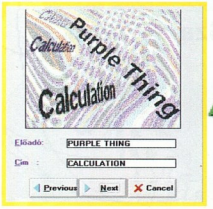


8. A Windows erőforrásairól és használatukról olvashatunk ebben az írásban.



48. Átlátszó animációt varázsolunk a Windows képernyőre.

16. A Pc túléli önmagát. Az irodák után a való világot is meghódítja, de előbb átöltözik.



52. dBase for Windows lekérdező ablakot készítünk.

387-es COPROCESSZOR

Biztosan többször találkoztatok azzal a problémával, hogy meg kellett volna vizsgálni, van-e a számítógépben matematikai segédprocesszor, és ha nincs, milyen jó lett volna ha be tudjátok kapcsolni annak emulálását. Ez a megoldás segíthet olyan programok futtatásánál, amelyek igénylik a 387-es segédprocesszort. Ennek a kis assembler programnak a segítségével akár még sikerülhet is, és beépíthetjük saját programjainkba.

Ennek a kis programnak a segítségével megtudhatjuk, hogyan kell detektálni egy numerikus processzort egy számítógépben, és ha nem találtunk, akkor hogyan kell annak az emulációját bekapcsolni. Ha lefordítjuk, akkor egy olyan kis programot kapunk, ami kiírja a képernyőre hogy van-e segédprocesszor, és ha nincs, akkor bekapcsolja annak az emulálását. Felhasználhatjuk különböző nyelvekben, például C, Pascal vagy Clipper programunkban, ha írunk egy detektáló és egy emulálást bekapcsoló rutint. Ennek semmi akadálya.

A forráslistában találhatunk általunk még nem ismert regisztereket, de nem kell megijedni, ennek magyarázata, hogy a program 386-os valós üzemmódban lett megírva. Azért választottam ezt a megoldást, mert eleve a 387-est emulálom, ami csakis 386-os vagy 486-os gépekben található. Mivel azonban a processzor 0. vezérlőregisztere 32 bites, 32 bites üzemmódban dolgozom, így legalább erre is látunk példát. Ugyan lehetőség van 286-os vezérlőregiszter kiolvasására is, de abban az esetben csak detektálni tudunk.

A 32 bites üzemmódról egyenlőre elég annyit tudni, hogy az általános regisztereink (AX, DX, CX, BX) melyek 16 bitesek ugyanúgy megmaradnak, de használhatjuk a 32 bites változatait is. Ezek sorra a EAX, EDX, ECX és az EBX regiszterek. Az eddigi 16 bites regiszterek ugyanúgy használhatók továbbra is, mint amikor az AL regisztert használjuk, tehát az AX most megfelel az EAX alsó 16 bitjének.

A processzorban 4 db vezérlőregisztert találunk, ezek a CR0, CR1, CR2 és a CR3. Mi a 0. vezérlőregisztert használjuk majd. Ezekről a regiszterekről nem árt tudnunk, hogy nem illik csak úgy összevissza írni még olvasni, és azt is csak 0. privilégizált módban lehet. Fizikai olvasásuk és írásuk csakis a MOV utasítással lehetséges. A CR0 regiszter felelős a védett üzemmód engedélyezéséért, a lapozásért és a matematikai segédprocesszor beállításáért.

Detektáláshoz a CR0 4. bitjét használjuk, ugyanis ha ennek értéke 1, akkor ez azt jelenti, hogy 387-es van a rendszerben, (ez 486-osnál mindig 1) 0 esetében pedig azt, hogy 287-es vagy még az sem található.

Emuláláshoz a CR0 2. bitjét használjuk. Ennek értéke 1, akkor a lebegőpontos utasítások, melyeket a numerikus processzor hajtana végre, átirányítódnak a rendszerhez, és ott szoftveres vagy egyéb úton lesz megoldva. Így a programozónak lehetősége van olyan programot írni, ami használ segédprocesszort.

Szalay Zsolt

```
.386p

PROG    SEGMENT use16
        ASSUME  CS:PROG,DS:PROG
        org     100h

START:
        jmp     indul

        szov1   db  0dh,0ah,'387-es CoProcessor Emulátor v1.0',0dh,0ah,'$'
        szov2   db  0dh,0ah,'Van CoProcessor a számítóg,pben, nem kell emulálni.'
                ,0dh,0ah,'$'
        szov3   db  0dh,0ah,'Nem találtam CoProcessort.',0dh,0ah,'$'
        szov4   db  0dh,0ah,'CoProcessor emulálás bekapcsolva.',0dh,0ah,'$'

indul:
        push    cs
        pop     ds
        mov     dx,offset szov1
        mov     ah,9
        int     21h

        mov     eax,CR0
        and     al,16
        cmp     al,0
        je      t1
        mov     dx,offset szov2
        mov     ah,9
        int     21h
        jmp     vege
t1:
        mov     dx,offset szov3
        mov     ah,9
        int     21h
        mov     eax,CR0
        or      al,4
        mov     CR0,eax
        mov     dx,offset szov4
        mov     ah,9
        int     21h
vege:
        MOV     AX, 4C00H
        INT     21H
PROG    ENDS

        END     START
```

Óra a kereten

Egy Windows-os program készítésének alapelemeit mutatja be a Forráskód egy példán keresztül, a már nem teljesen kezdő programozóknak. A példaprogram, elterjedtsége okán a Borland C++ fordító 3.1 verziójára épül, de az alapelvek más hasonló rendszerekre is igazak.

Egy windows program készítésekor az egy vagy több forrásfile megírásán túl szükség van még egy modul definíciós állományra, és szükség szerint egy az erőforrásokat tartalmazó állományra. Először nézzük a létrehozandó programot leíró .DEF kiterjesztésű filet. A modul definíciós állomány egy szövegfile, aminek kötelezően tartalmaznia kell a következő sorokat:

```
NAME        Win_Ora          //Az exe file neve
DESCRIPTION 'Forráskód'      //Szöveg, bele kerül a file-ba
EXETYPE     WINDOWS          //Az EXE tipusa
STUB        'WINSTUB.EXE'    //Beépül az alkalmazásba,
                             //nem Windows alól indítva kilép
CODE      PRELOAD MOVABLE    //A kódszegmens kezelése
DATA      PRELOAD MOVABLE    //Az adatszegmens kezelése
HEAPSIZE    1024             //Lokális memória a halomban
STACKSIZE   8192             //A szükséges verem mérete
EXPORTS     WndProc          //Az ablakkezelő függvény
```

A CODE és a DATA sorokban a szerepeltethető kulcsszavak:

PRELOAD :	A program indításkor a kód bekerül a memóriába.
MOVABLE :	A szegmenst a Windows áthelyezheti.
DISCARDABLE :	A Windows törölheti a szegmenst ha szükséges.
MULTIPLE :	Csak DATA sorban használható. Ha a program több példányban fut minden példánynak saját adatszegmense van.

A Windows erőforrásokkal egy másik írásunkban foglalkozunk, nézzük tehát a forrás filet. A program létrehoz egy Windows ablakot, megjeleníti. A létrehozáskor betölt az erőforrásból egy bittérképet. Majd másodpercenként tizennyolcszor lekérdezi a rendszeridőt és a rendelkezésre álló szabad memóriát (csere file is), majd megjeleníti a kereten. A program demonstratív jellegű, egyes moduljait eltérő színűre szedtük, a hozzájuk tartozó magyarázatot azonos színű keretben találjuk. Ezt az írást egy állandó Windows programozói rovat indításaként gondoljuk. Akikben sikerült felkelteni az érdeklődést a Windows programozás iránt, írják meg mi érdekli őket, hogy a tematika kialakításánál figyelembe vehessük. Akinek kész programja van, azt is várjuk.

Nagy Sándor

A program a C nyelv szabályai szerint kezdődik. Beillesztjük a windows.h állományt, ezt minden Windows programnak tartalmaznia kell, ebben deklarálódnak a Windows függvényei, állandói.
A time.h beillesztését az adott feladat indokolja, ezt követi néhány deklaráció.

A főprogram által hívott függvények is a szokásos C szintaxist követik. Ami feltűnik: a szokatlan tipusazonosítók, és az ismeretlen függvények. A tipus definíciókat a már említett windows.h file tartalmazza. Láthatóan ezeket csupa nagybetűvel írjuk.
Nagyon sokszor használunk struktúrákat, ezekre is az előbbiek érvényesek. A Windows, objektumai azonosítását 16 bites számokkal oldja meg, ezek az azonosítók előre deklaráltak és nagy H-val kezdődnek (handle). Az ismeretlen függvények a Windows API részei (több mint 600 van belőlük).

Minden Windows programnak kötelezően tartalmaznia kell egy **WinMain** nevű függvényt. Ez a program belépési pontja, a főprogram.

A WinMain felépítése :

- **Deklarációk.**
- **Az alkalmazás ablak osztályainak meghatározása.** Ez egy stuktúra kitöltéséből, a RegisterClass függvény meghívásából áll.
- **Az alkalmazás ablakának létrehozása.** Meghívjuk a CreateWindow függvényt.
- **Az alkalmazás ablakának megjelenítése.** Meghívjuk a ShowWindow és a UpdateWindow függvényt.
- **Üzenetek továbbítása az ablakkezelő függvénynek.** Minden ablakhoz tartozik egy ablakkezelő függvény, amit mi írunk, de közvetlenül sohasem hívjuk meg. A WinMain végén található végtelen ciklussal minden Windowsos esemény: billentyű leütés, egér mozgatás stb. esetén négy számból álló üzenetet küld az Windows az ablakkezelő függvénynek.

Az ablakkezelő függvény egyszerű szerkezet. A szükséges deklarációk, függvényhívások után egy switch-case utasításpárral elkapjuk azokat az üzeneteket, melyeket mi akarunk feldolgozni, ezekre megírjuk a szükséges eljárásokat. A többi utasítást hagyjuk rácsorogni a Windows saját ablakkezelő függvényére, a DefWindowProc eljárásra.

```
#include "windows.h"
#include "time.h"
#define  ID_TIMER 1

long FAR PASCAL WndProc( HWND, WORD, WORD, LONG);
HANDLE hInst;


void Ora( HWND hWnd, HDC hDC)
{
  char                buff[30];
  LONG                ido;
  RECT                rect;
  struct tm           * dt;
  int                  i = -1;
  DWORD               fmem;

  time( &ido);
  dt      = localtime( &ido);
  fmem = GetFreeSpace( 0);
  sprintf( buff," %2d:%2d:%2d   %.3f MB free",
              dt->tm_hour, dt->tm_min, dt->tm_sec, fmem/1024.0/1024.0);
  GetClientRect( hWnd, &rect);
  rect.top   = rect.top + 4;
  rect.right = rect.right - 38;
  DrawText( hDC, buff, i, &rect,DT_SINGLELINE | DT_RIGHT | DT_TOP );
  }


void PaintBitmap( HWND hWnd, HDC hdc, HBITMAP hBitmap)
{
 HDC                hdcm;
 HBITMAP            hBitmapm;
 BITMAP             logo;
 RECT               rect;
 int                szel, magas;

    GetObject( hBitmap, sizeof( BITMAP), (LPSTR) &logo);
    GetClientRect( hWnd, &rect);
    szel       = logo.bmWidth;
    magas      = logo.bmHeight;
    hdcm       = CreateCompatibleDC( hdc);
    hBitmapm  = SelectObject( hdcm, hBitmap);
    BitBlt( hdc, 0, 0, szel, magas, hdcm, 0, 0, SRCCOPY);
    SelectObject( hdcm, hBitmapm);
    DeleteObject( hBitmapm);
    DeleteDC( hdcm);
}


int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{

   MSG     msg;
   HWND  hWnd;
   WNDCLASS wndc;

   if ( lpCmdLine );          hInst  = hInstance;

   if (!hPrevInstance)
     {
          wndc.style          = CS_HREDRAW | CS_VREDRAW;
          wndc.lpfnWndProc    = WndProc;
          wndc.cbClsExtra     = 0;
          wndc.cbWndExtra     = 0;
          wndc.hInstance      = hInstance;
          wndc.hIcon          = LoadIcon(NULL, IDI_APPLICATION);
          wndc.hCursor        = LoadCursor(NULL, IDC_ARROW);
          wndc.hbrBackground  = GetStockObject(LTGRAY_BRUSH);
          wndc.lpszMenuName   =  NULL;
          wndc.lpszClassName  = "Forraskod";
          RegisterClass( &wndc);
     }
```

```
hWnd = CreateWindow("Forraskod",
                    "Óra demo",
                    WS_OVERLAPPEDWINDOW,
                    CW_USEDEFAULT,
                    CW_USEDEFAULT,
                    480,135,
                    NULL,
                    NULL,
                    hInstance,
                    NULL );


   ShowWindow(hWnd, nCmdShow);

   UpdateWindow(hWnd);

   SetTimer(hWnd,1,100,NULL);

   while (GetMessage (&msg,NULL,NULL,NULL))
   {
   TranslateMessage(&msg);
   DispatchMessage(&msg);
   }
}


long FAR PASCAL WndProc      (HWND hWnd, WORD message,
                              WORD wParam, LONG lParam)
{

  RECT              rect;
  HDC               hdc ,h2dc;
  PAINTSTRUCT       ps;
  static HBITMAP    hBitmap;

  hdc = GetDC(hWnd);
  GetClientRect(hWnd,  &rect);
  ReleaseDC(hWnd, hdc);
  h2dc = GetWindowDC(hWnd);
  ReleaseDC(hWnd,h2dc);


  switch (message)
            {
             case WM_CREATE:
                        hBitmap = LoadBitmap( hInst, "forraskod");
                        hdc = GetDC( hWnd);
                        PaintBitmap( hWnd, hdc, hBitmap);
                        ReleaseDC( hWnd, hdc);

              case WM_PAINT:
                        hdc = BeginPaint( hWnd, &ps);
                        PaintBitmap( hWnd, hdc, hBitmap);
                        EndPaint( hWnd, &ps);
                        return FALSE;

             case WM_TIMER:
                        Ora (hWnd,h2dc);
                         break;


             case WM_DESTROY:
                        KillTimer (hWnd,ID_TIMER);
                        PostQuitMessage(0);
                        return FALSE;

             }

  return DefWindowProc(hWnd, message, wParam, lParam);

}
```

Az eszközfüggetlen bittérkép

A Windows standard képformátuma a BMP kiterjesztésű eszközfüggetlen bittérkép. Nézzük meg ennek felépítését egy rövid C programmal szemléltetve.

Az OS/2 Presentation Manager-ben vezették be azt a képfile formátumot, amelyik az egyes képpontokat a konkrét színinformációval, színbitekkel írja le. A színek közvetlenül az RGB (piros, zöld, kék) színösszetevők arányával vannak meghatározva, így nem függenek a megjelenítő eszköz tipusától.
Az egyes pontokat meghatározó színbitek száma többféle lehet, így többféle színmélységgel tárolhatunk képet:

Színbitek száma:	Megjeleníthető színek száma:
1	2
4	16
8	256
24	16 millió

Nézzük a Windows standard képformátumának számító BMP file felépítését. A BMP file két részből áll:

1. Fejrész : tartalmazza a kép adatait.
2. Leíró rész : tartalmazza a képpontokat az alsó sortól kezdődően 1 - 24 bitenként egyet - egyet. A leíró rész lehet tömörített formátumú is.

A fejrész az érdekesebb, előre definiált struktúrákból épül fel, melyek deklarációit a fordítók windows.h állománya tartalmazza.
A BMP file a **BITMAPFILEHEADER** nevű struktúrával kezdődik.

A BITMAPFILEHEADER struktúra mezői:

WORD	**bfType**	A file tipusa bittérképnél "BM".
DWORD	**bfSize**	A file teljes mérete.
WORD	**bfReserved1**	Foglalt mező feltöltve 0 -val.
WORD	**bfReserved2**	Foglalt mező feltöltve 0 -val.
DWORD	**bfOffBits**	A leíró rész kezdete file-ban.

Ezt a struktúrát egy újabb struktúra követi, aminek elemei maguk is struktúrák.

```
typedef struct tagBITMAPINFO
        {
        BITMAPINFOHEADER    bmiHeader;
        RGBQARD             bmiColors[1];
        } BITMAPINFO;
```

A BITMAPINFOHEADER struktúra mezői:

DWORD	**biSize**	A struktúra mérete byte-ban.
DWORD	**biWidth**	A bittérkép szélessége pixelben.
DWORD	**biHeight**	A bittérkép magassága pixelben.
WORD	**biPlanes**	A színsíkok száma mindig 1.
WORD	**biBitCount**	Egy pixelt hány bit ír le.
DWORD	**biCompression**	Az esetleges tömörítés formáját jelzi.
DWORD	**biSizeImage**	Kép mérete byte-ban.
DWORD	**biXPelsPerMeter;**	Megjelenítő eszköz vízszintes felbontása.
DWORD	**biYPelsPerMeter;**	Megjelenítő eszköz függőleges felbontása.
DWORD	**ClrUsed;**	Az aktuálisan használt színek száma. Ha 0, mindet használja.
DWORD	**ClrImportant;**	Az értelmes megjelenítéshez szükséges színek száma. Ha 0, mindet használja.

Ezt a struktúrát követi egy tömb, aminek annyi eleme van, ahány színt használunk. A tömb elemei a **RGBQUAD** struktúrák, melyek azonosítják az egyes színeket. A struktúra mezői:

BYTE	**rgbBlue**
BYTE	**rgbGreen**
BYTE	**rgbRed**
BYTE	**rgbReserved**

Az rgbReserved értéke mindig nulla, a többi mező a színösszetevő értékét tartalmazza. Ez így elég száraz és bonyolult, de a valóságban könnyen elboldogulunk vele. Nézzünk egy példát: egy egyszerű C nyelvű programmal beolvassuk a file fejlécét, a legfontosabb adatokat kiíratjuk a képernyőre, és ha 16 színű a bittérkép, meg is jelenítjük.
A file elején beillesztjük a külön bmp.h file-ben lévő struktúradeklarációkat. Ha paraméterrel indítottuk a programot, meghívja a BMPinform nevű függvényt, ha nem, kilép és figyelmeztet a helyes indításra. A BMPinform függvény megnyitja a parancssorban megadott nevű filet olvasásra, és feltölti a struktúrákat adattal.
Ha az első két byte nem "BM", a program hibaüzenettel kilép. Ha a file bitmap, kiírjuk a struktúra néhány mezőjét képernyőre.
Ha a BMPinform függvény visszatérési értéke, a pixelenkénti bitek száma 4, azaz 16 színű a bittérkép, meghívjuk a BMPout nevű függvényt. Újra megnyitjuk a filet, a file pointert a leíró rész elejére állítjuk. Megnyitjuk a grafikus rendszert, átdefiniáljuk a palettát és pontonként kirakjuk a képet. A paletta módosítására a BGI grafika és a Windows standard színeinek különbözősége miatt van szükség. Az eredmény így is csak közelítő.

```
Edit: bcload.bmp                          Line 1    Col 1     143,258 Free
00000000  42 4D 76 58  02 00 00 00  00 00 76 00  00 00 28 00  BMvX..... .v...(.
00000010  00 00 00 02  00 00 E0 01  00 00 01 00  04 00 00 00  ..Ç..α..Ω.♦...
00000020  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000030  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ............Ç..Ç
00000040  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ...Ç.Ç...Ç.Ç.ÇÇ
00000050  00 00 00 00  00 00 C0 C0  C0 00 00 00  FF 00 00 FF  ..ÇÇÇ.└└└... ..
00000060  00 00 00 FF  FF 00 FF 00  00 00 FF 00  FF 00 FF FF  ... . ... . .
00000070  00 00 FF FF  FF 00 00 00  00 00 00 00  00 00 00 00  ..  ............
00000080  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000090  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000000A0  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000000B0  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000000C0  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000000D0  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000000E0  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
```

16 színű bittérkép fejléce

```
/*              bmp.h                   */
typedef struct tagBITMAPFILEHEADER
{
        unsigned int  bfType;                   /* "BM" azaz 0x4D42     */
        unsigned long bfSize;                   /* file mérete byte     */
        unsigned int  bfReserved1;
        unsigned int  bfReserved2;
        unsigned long bfOffBits;                /* adatok kezdete file-ben */
} BITMAPFILEHEADER;

typedef struct tagBITMAPINFOHEADER
{
        unsigned long biSize;                   /* struktúra mérete     */
        unsigned long biWidth;                  /* Szélesség pixelben   */
        unsigned long biHeight;                 /* Magasság  pixelben   */
        unsigned int  biPlanes;                 /* Szinsík =  1         */
        unsigned int  biBitCount;               /* szinbit per pixel    */
        unsigned long biCompression;            /* Kompresszió          */
        unsigned long biSizeImage;              /* Kép mérete  byte     */
        unsigned long biXPelsPerMeter;          /* Vizszintes felbontás */
        unsigned long biYPelsPerMeter;          /* Függöleges felbontás */
        unsigned long biClrUsed;                /* Használt szinek száma*/
        unsigned long biClrImportant;           /* Import szinek száma  */
} BITMAPINFOHEADER;
```

```
 /*             bmp.c            */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <graphics.h>
#include <alloc.h>
#include <conio.h>
#include "bmp.h"
#define VBUFFSIZE 10240

BITMAPFILEHEADER BMPHeader;
BITMAPINFOHEADER BMPInfo;
unsigned _stklen = 10240;
struct palettetype pal = { {MAXCOLORS},
                                        { EGA_BLACK,
                                          EGA_RED,
                                          EGA_GREEN,
                                          EGA_CYAN,
                                          EGA_BLUE,
                                          EGA_MAGENTA,
                                          EGA_BROWN,
                                          EGA_DARKGRAY,
                                          EGA_LIGHTGRAY,
                                          EGA_LIGHTRED,
                                          EGA_LIGHTGREEN,
                                          EGA_YELLOW,
                                          EGA_LIGHTBLUE,
                                          EGA_LIGHTMAGENTA,
                                          EGA_LIGHTCYAN,
                                          EGA_WHITE
                                        }
};

void BMPout(char *BMPfile)
{
FILE *Fn;
int gdriver=DETECT,gmode, x, y, maxx, maxy;
char Color, buff[80];

Fn = fopen(BMPfile,"rb");
fseek(Fn,BMPHeader.bfOffBits,SEEK_SET);
setgraphbufsize(VBUFFSIZE);
if ((BMPInfo.biWidth * BMPInfo.biHeight) >= farcoreleft())
                {puts("Kevés a memória"); exit(1);}
```

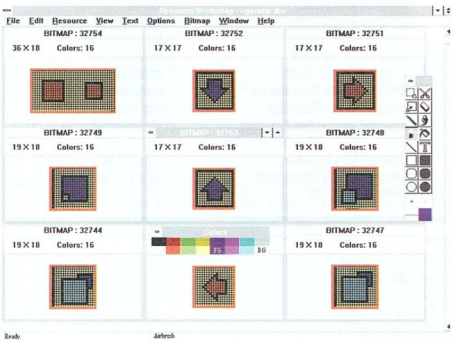
```
initgraph(&gdriver, &gmode, "");
x = graphresult();
if (x != grOk)
{printf("InitGraph> %s\n", grapherrormsg(x)); exit(1);}
x = getmaxx();
y = getmaxy();
maxx = min(x,BMPInfo.biWidth);
maxy = min(y,BMPInfo.biHeight);
setallpalette(&pal);
x = 1;
cleardevice();
y = maxy;
do
{
                for (x=0; x < maxx; x++)
                {
                fread(&Color,sizeof(Color),1,Fn);
                putpixel(x++,y,(Color>>4));
                putpixel(x,y,(Color&0xf));
                }
} while (y-- > 0);
getch();
closegraph();
fclose(Fn);
}

int BMPinform(char *BMPfile)
{
 FILE *Fn;

 Fn = fopen(BMPfile,"rb");
 fread(&BMPHeader,sizeof(BMPHeader),1,Fn);
 fread(&BMPInfo,sizeof(BMPInfo),1,Fn);
 fclose(Fn);
 if (BMPHeader.bfType!=0x4D42)
   {printf("\nNem BMP file");exit(1);}
 clrscr();
 printf("\nFilen,v : %s",BMPfile);
 printf("\nTipus                  : ");
      if (BMPInfo.biBitCount == 1) printf("Monokróm");
      else if (BMPInfo.biBitCount == 4) printf("16-Szinü");
      else if (BMPInfo.biBitCount == 8) printf("256-Szinü");
      else printf("24-Bites RGB");
 if (BMPInfo.biCompression==0) printf("\nNem tömöritett");
 else     printf("\nTömörités              : %u",BMPInfo.biCompression);
 printf("\nA BMP file mérete          : %ul",BMPHeader.bfSize);
 printf("\nKép szélessége pixelben    : %u",BMPInfo.biWidth);
 printf("\nKép magassága pixelben     : %u",BMPInfo.biHeight);
 if (BMPInfo.biClrImportant)
 printf("\nHasznált szinek száma      : %u",BMPInfo.biClrImportant);
 return(BMPInfo.biBitCount);
}

void main (int argc, char *argv[])
{
int bmp;
 if (argc == 2)
   {
    bmp = BMPinform(argv[1]);
    if(bmp==4)
                {
                printf("\n\n> Kirajzolom...");
                getch();
                BMPout(argv[1]);
                clrscr();
                }
                 else printf("\n\nCsak 16 színût tudok megjeleniteni..." );
    }
    else puts(" Használat BMP filenév ");
}
```

Kopott az ablaka? Fesse át!

Erőforrások a Windowsban

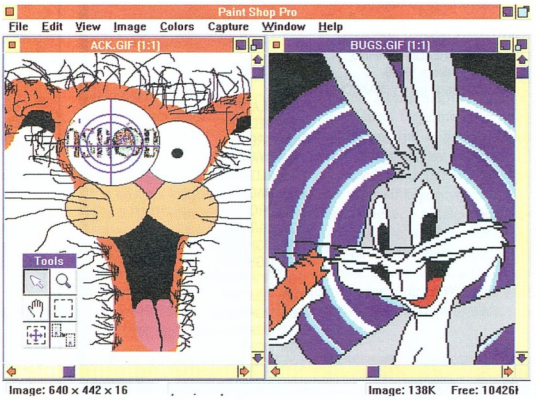
Resource Workshop

A Windows programozásban a menüket, dialógus dobozokat, ikonokat, bittérképeket, kurzorokat, fontkészleteket, string táblázatokat, gyorsbillentyűket erőforrásoknak nevezzük. A Windows különlegesen kezeli ezeket, beépülnek ugyan a futtatható EXE, vagy a dinamikusan szerkeszthető DLL file-okba, de nem részei a program adatszegmensének. A program indításakor, általában az erőforrások nem töltődnek be a memóriába, csak ha az alkalmazásnak szüksége van rá. Ugyanakkor ha a Windowsnak memóriára van szüksége a feleslegesnek ítélt erőforrások helyét a memóriában felszabadítja, és ha megint szükség van rá újra betölti. További előny, hogy ha egy program több példányban fut, erőforrásait csak egyszer tölti be és közösen használják azokat. Program - fejlesztéskor az erőforrásokat egy szövegfile-ban írjuk le, aminek az RC kiterjesztést adjuk. Az így elkészített filet az erőforrás fordító az RC.exe segítségével bináris RES formátumra fordíthatjuk, és a programhoz fűzhetjük. Az itt leírt folyamat parancssorból is vezérelhető, de a gyakorlatban project-file, make-file segítségével oldjuk meg.



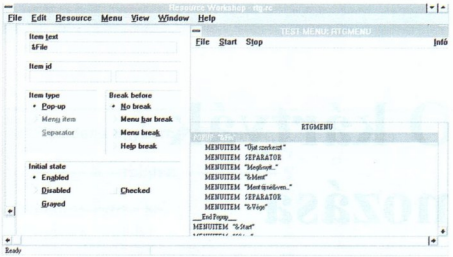
1. ábra A VGA.DRV módosítása fent a munka képernyő, lent az eredmény.

Az erőforrás a programfile meghatározott helyére kerül, onnan alkalmas programmal kivehető, módosítható, visszatölthető. Ez lehetőséget teremt alkalmazások arculatának utólagos megváltoztatására, például menük magyarítására. A legismertebb erőforrás szerkesztő program a Borland fordítókhoz adott RESOURCE WORKSHOP azaz RWS.EXE. A Windows alatt futó RWS kényelmes lehetőséget biztosít erőforrások létrehozására, módosítására úgy, hogy bármilyen formátumot megnyithatunk, dialógus ablakok segítségével megváltoztathatjuk a tartalmukat és tetszés szerinti formátumban menthetjük, vagy visszatölthetjük az alkalmazásba. Példaként a Windows képernyő meghajtó programját a VGA.DRV állományt nyitottuk meg, és az ablakok kirajzolásánál használt bittérképeket kikerestük és módosítottuk, így saját Windows felületet kaptunk. Mielőtt valaki a nyomunkba ered ne feledkezzen meg a kötelező óvatosságról, módosítás előtt a file-okról készítsen másolatot. Az RWS igazi felhasználási területe természetesen az új alkalmazások erőforrásainak elkészítése. Az új erőforrás készítésekor először a tipusát kell kiválasztanunk. Szerkeszthetünk erőforrásleíró file-t, kurzort, ikont, fontot, bittérképet. Ezután rögtön a megfelelő szerkesztő ablakba jutunk (2. 3. 4. ábra). Az eszköz palettákról egérrel választhatjuk ki a szükséges eszközt vagy erőforrást, mint a Windowsos rajzprogramokban. A megfelelő helyen leejtve, és kétszer rákattintva megnyílik a leírásukhoz szükséges



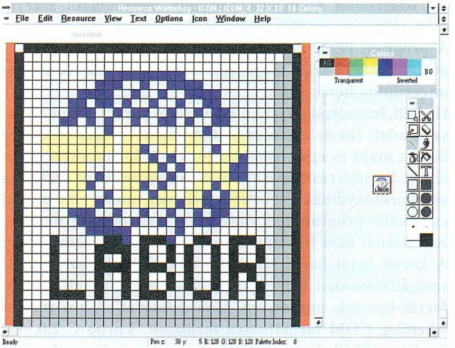
dialógus ablak. Az erőforrásokat a programon belül tesztelhetjük is. A munka végeztével az eredményt lehetőleg RC kiterjesztésű szöveg file-ba mentsük. Ez lehetőséget biztosít az utólagos finomításokra, módosításokra. Egy ilyen RWS erőforrásleíró szövegfile-t láthatunk az 1. listán.

Látható, hogy egy-egy erőforrás leírása az erőforrás megnevezéséből, a rajta lévő feliratból, esetleges módosító jelzőkből, az esetleges koordinátákból és az azonosító számból áll. Ezt az azonosító számot használhatjuk fel a Windows programban az erőforrás azonosítására.

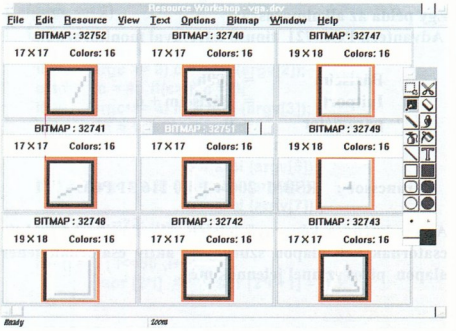


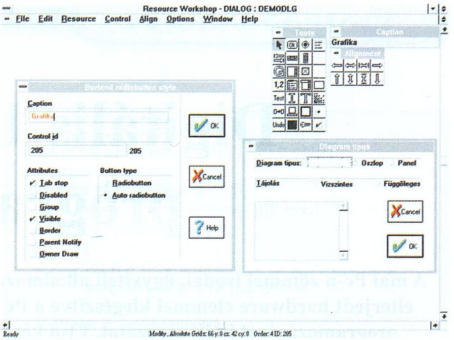
2. ábra Menű készítés az erőforrás szerkesztővel.

3.ábra Ikonok bittérképek kezelése az RWS-ben.



5. Akár más ablakozós rendszereket is utánozhatunk.





4. ábra Dialógus ablak készítése.

1. lista Az erőforrás szerkesztővel készült RC file.

```
RTGABLAKMENU MENU
BEGIN
        POPUP "&File"
        BEGIN
                MENUITEM "Új ", 102
                MENUITEM SEPARATOR
                MENUITEM "Meg&nyit...", 22
                MENUITEM "&Ment", 23
                MENUITEM "Ment új né&ven...", 24
        END

        POPUP "&Szerkeszt"
        BEGIN
                MENUITEM "&Új diagram... ", 218
                MENUITEM SEPARATOR
                MENUITEM "&Háttér...", 201
                MENUITEM "Háttér &Törlés", 240
        END

        MENUITEM "\a&Névjegy...", 18
END

DEMODLG DIALOG 249, -77, 193, 131
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION
                                        |WS_SYSMENU
CLASS "BorDlg"
CAPTION "Diagram tipus"
FONT 8, "Helv"
BEGIN
        CONTROL "Ok", 1, "BorBtn", 1 | WS_CHILD | WS_VISIBLE
                | WS_TABSTOP, 145, 91, 36, 24
        CONTROL "Cancel", 2, "BorBtn", 0 | WS_CHILD | WS_VISIBLE
                | WS_TABSTOP, 145, 53, 36, 24
        CONTROL "&Diagram tipus:", 206, "BorShade", BSS_GROUP
                | WS_CHILD | WS_VISIBLE | WS_GROUP, 10, 7, 170, 12
        CONTROL "Grafika", 205, "BorRadio", BS_AUTORADIOBUTTON
                | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 66, 8, 42, 8
        CONTROL "Oszlop", 203, "BorRadio", BS_AUTORADIOBUTTON
                | WS_CHILD | WS_VISIBLE, 108, 8, 40, 8
        CONTROL "Panel", 204, "BorRadio", BS_AUTORADIOBUTTON
                | WS_CHILD | WS_VISIBLE, 145, 8, 33, 8
        CONTROL "&Tájolás", 204, "BorShade", BSS_GROUP | WS_CHILD
                | WS_VISIBLE, 10, 29, 170, 13
        CONTROL "Vízszintes", 207, "BorRadio", BS_AUTORADIOBUTTON
                | WS_CHILD | WS_VISIBLE, 67, 30, 51, 11
        CONTROL "Függőleges", 208, "BorRadio", BS_AUTORADIOBUTTON
                | WS_CHILD | WS_VISIBLE, 131, 30, 50, 10
        LISTBOX 209, 10, 55, 95, 70
END
ICON_4 ICON "tex.ico"
BITMAP_1 BITMAP "life.bmp"
```

Digitális I/O kártyák programozása

A mai Pc-n zömmel irodai, ügyviteli alkalmazások, játékok futnak, de néhány ma még nem túl elterjedt hardware elemmel kiegészítve a Pc másra is képes. Lapunk ezen hardware elemek programozásáról indít sorozatot. Első két részben a digitális I/O kártyák programozását ismertetjük.

A digitális I/O kártyákat elektromos berendezések be-, és kikapcsolására, be-, vagy kikapcsolt állapot ellenőrzésére, külső eszközök vezérlésére használhatjuk.
Megjelenésüket nézve normál Pc-s bővítő kártyák többnyire 8 bites ISA busszal. A be-, ill. kimeneti jel többnyire TTL szintű, ha nem az, akkor védeni kell a számítógépet az esetleges túláramoktól. Néha a külvilágot kell védeni a számítógéptől, például robbanásveszélyes környezetben. A védelmet opto-csatolós, relés leválasztással oldják meg, terjednek a szilárdtest-relés megoldások. Kis áramok és kevés csatorna esetén a leválasztást ráépítik a kártyára, más esetekben a gépen kívül helyezik el és szalagkábellel csatlakoztatják a kártyához. Egy kártyán 8-tól 144-ig I/O vonal található /mindig nyolc egészszámú többszöröse/, sokszor egyéb áramkörök /pl. számláló időzítő, analóg érzékelők/ társaságában. Működés és programozás szempontjából kétféle kártyát ismerünk: amelyik megszakítást generál és amelyik nem. A megszakítást nem generáló kártyák programozása nem más, mint egy kártyafüggő portcím írása, olvasása, IN és OUT utasítással. A portcímet a kártyán kapcsolókkal vagy jumperekkel választhatjuk ki.

Szokásos címtartományok /hexa/:

200 - 277
300 - 377
380 - 3AF
3C0 - 3CF
3E0 - 3EF

A portra byte-okat írunk ill. onnan byte-okat olvasunk, így egyszerre nyolc csatornát írunk vagy olvasunk. A byte-okat nekünk kell bitekből összeállítani vagy olvasáskor a byte-ot bitekre bontani. Egyes kártyáknál írás vagy olvasás előtt be kell állítani a szükséges üzemmódot, másoknál nem. Az üzemmód-beállítás egy un. inicializáló portra küldött inicializáló paraméter írásából áll. Erre vonatkozóan a kártyához kapott leírásból tájékozódhatunk. 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 használhatunk. 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.
Ebben a cikkben a hagyományos digitális inputkártyával történő háttér-monitorozásra mutatunk példát.
A megszakítást generáló kártyák programozására egy konkrét alkalmazási példán keresztül még visszatérünk.
Az ismertetett C nyelvű program használható szerviz célra vagy egyszerűbb feladatok figyelésére.
A program rezidensen bent ül a memóriában és másodpercenként 18-szor megkapja a vezérlést az 1c /hexa/ interrupton keresztül.
Ha kell, inicializáló jelet küld a vezérlő portra, majd olvassa az adott címen levő portokat. A kapott értékeket bitekre bontja majd és egy stringbe írja, végül kiírja a stringet a színes karakteres képernyőre. A program nem vizsgálja saját példányainak létét a memóriában, onnan eltávolítani csak külső programmal, vagy a gép újra indításával lehet. Kiiratásnál nem figyeljük az elektronsugár visszafutását. A közölt listát bárki szabadon módosíthatja, az említett szolgáltatásokkal kiegészítheti, de számolnia kell az erőforrás-igények ugrásszerű növekedésével. A programot lehetőleg COM formátumra fordítsuk. Turbo C 2.0 -val fordítva 8560 byte méretű programot kapunk, ami futáskor kb. 10 Kbyte helyet foglal a memóriából.
Egy példa az alkalmazásra:
Advantech PCL-721 típusú kártyával monitorozunk.

Báziscím	= 200h
Initport	= Báziscím + 6 ,
paraméter	= 0
Csatorna szám	= 32

Parancssor : RSBM 200 4 1 6 0 116 31 PCL - 721

Az eredmény a képernyő tetején jelentkezik. Az inaktív csatornák kék alapon szürke, az aktív csatornák fehér alapon piros színnel jelennek meg.

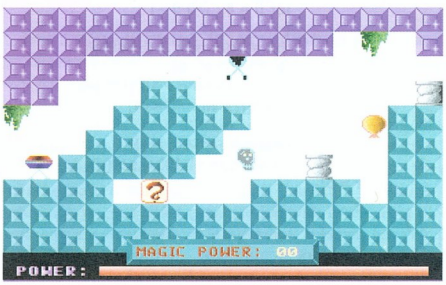
```
/* Rezidens egysoros bitmonitor  */
/*                               */
/* ————— rsbm.c ————— */

#include <stdio.h>
#include <dos.h>

int hexatoi (char*);
static void interrupt (*oldtimer)(void);
static void interrupt newtimer(void);

char sor[160], bit, cb[5];
int basel,ln,i,j,a,c,num[8],r,o,p,e,h,v =0xb800;

void main(argc,argv)
int argc;
char *argv[ ];
{
if      (argc == 1)
 {
 puts (" REZIDENS EGYSOROS BITMONITOR");
 puts (" ==============================");
 puts ("Indítás: \n");
 puts ("RSBM [startcím] [portszám] [sor] [initport]
        [initparam] [szín1] [szín2] [cimke]\n");
 puts ("startcim  = első port címe hexadecimálisan ");
 puts ("portszám= az egy sorban kijelzett 8 bites por-
        tok száma  /max = 8/ ");
 puts ("sor       = a képernyő hányadik sorában
                    történik a kijelzés");
 puts (" initport  = az inicializáló port relatív
                    távolsága startcímtől");
 puts ("initparam =  az inicializáláshoz az initportra
                    írandó paraméter");
 puts ("szín1      =  inaktívszín ");
 puts ("szín2      =  aktívszín ");
 puts ("cimke      =  a sorban feliratot
                      helyezhetünk el\n");
 puts ("Paramétert elhagyni csak a végéről lehet. ");
 puts ("A gép teljesítményétől, a rendelkezésre álló
        memóriától függően ");
 puts ("egyszerre több példányban és különböző
        paraméterezéssel is futhat.");
 puts ("Futásakor 10 Kb. memóriát használ
                példányonként.");
 }

if(argc >= 2)
 {
  basel = hexatoi(argv[1]);
  if      (argc >= 3) c = atoi (argv[2]);
  else     c = 4;  if(c>8) c = 8;
  if      (argc >= 4) ln = atoi (argv[3]);
  else     ln = 1;
  if      (argc >= 6) { o = atoi (argv[4]);
                        p = atoi (argv[5]);}
  if      (argc >= 8) { e = atoi (argv[6]);
                        h = atoi (argv[7]);}
  else    { e =23; h =116;}

  for (i = 0 ; i<=80 ;i++)
          { sor [2*i]  = ' ';  sor [2*i +1] = h;}
  sprintf(cb,"%04X",(basel));
   for (i = 0 ; i<4; i++)   sor[2*i +2] = cb[i];
   i = 0;
   if (argc >=9) while(argv[8][i])
           { sor [2*i+12] = argv[8][i]; i++;}
   for (j = 0 ; j<=(c-1) ;j++)
           {
           for (i = 8 ; i>0 ;i—)
           sor[158 -18*c + 18*j+2*i]=i+'0';
           }
   oldtimer = getvect(0x1c);
   setvect(0x1c, newtimer);
   keep (0,600);
 }
}


int hexatoi(char *szoveg )
{
int szam = 0;
int i = 0;
for(i=0;  (szoveg[i]>='0' && szoveg[i]<='9') ||
          (szoveg[i]>='a' && szoveg[i]<='f') ||
          (szoveg[i]>='A' && szoveg[i]<='F')  ;    i++)
{
 if  (szoveg[i]>='0' && szoveg[i]<='9')szam = 16*szam
+ szoveg[i] -'0' ;
 if  (szoveg[i]>='a' && szoveg[i]<='f')szam = 16*szam
+ szoveg[i] -'a'+ 10 ;
 if  (szoveg[i]>='A' && szoveg[i]<='F')szam = 16*szam
+ szoveg[i] -'A'+ 10 ;
}
return(szam);
}



static void interrupt newtimer()
{
   (*oldtimer)();
   if (r == 0)
   {
      r = 1;
      if (o) outportb(basel + o , p);
      for (j = 0 ; j<=(c-1) ;j++)
           {
                    num[j] = inportb(basel+j);
                    num[j]  = num[j]<<8;
                     for (i = 8 ; i>0 ;i—)
                     {
                      bit = !(!(num[j] & 0x8000));
                      a =( bit ? e : h);
                      sor[159 - 18*c + 18*j+2*i]=a;
                      num[j] = num[j] <<1;
                     }
                    for (i = 0 ; i<160; i++)
                    pokeb(v,(ln-1)*160+i,sor[i]);
          r = 0;
          }
   }
}
```

Játék assemblerben, HUBERT 2.

Aki nem olvasta a Compfair próbaszámban az első részt, az is hamar behozhatja lemaradását, mivel az előző rész forráslistáit megtalálja a lemezen. Ezek tulajdonképpen a video inicializáló, rgb beállítása, képernyőtörlés, egy billentyűre várakozó rutin és egy wait rutin.

A játéktér felépítése:

Mivel a teljes játéktér nem fér ki a képernyőre, ezért azt a klasszikus megoldást választottam, hogy felosztottam a játékteret megfelelően kis darabokra, amit már meg lehet jeleníteni. A "megfelelően kis" méret meghatározása egy játék megírásánál nagyon fontos. Itt dől el minden. Korlátaink csupán az a számítógép, amelyikre írjuk a programot, és saját képességeink, ha őszinték akarunk lenni. Tehát figyelembe kell venni a gép sebességét, a mi esetünkben megfelel egy 286-os is. Ennek függvényében meghatározhatjuk azt az adatmennyiséget amit mozgatni kívánunk a képernyőn. A mi esetünkben elég egy 286-os sebessége, mert egy időben 4 db mozgó figura van. Ez összesen 1440 byte mozgatását jelenti egy időegység alatt. Egy időegység az az idő ami alatt az összes változtatást és vizsgálatot el kell végezni. Ha jó gyorsra írjuk meg a rutinokat, akkor nem lesz gondunk a játék folyamatosságával, és marad időnk a képernyő szinkronizálására. Erre feltétlenül szükség van már egyszerű játékoknál is. A képernyő felépítése egy **16x10-es négyzetháló**, melynek elemei határozzák meg az éppen kirajzolandó terepet. Így ha elosztjuk a 320-at ami a képernyőnk vizszintes felbontása 16-al, akkor megkapjuk egy téglatest X irányú felbontását, ami pontosan 20. Mivel a képernyő alsó részét használjuk a magic power és a pontszám megjelenítéséhez, ezért függőlegesen csak 180 pixel magas a 10 tégla magasságú terep. Tehát 180/10=18 pixel magas egy téglatest. Mivel a terepeket előre megterveztük, gondoskodni kell az eltárolásukról. Ez önmagában nem probléma, mert rendelkezésünkre áll egy egész adatszegmens, amibe 65536 byte-ot tudunk eltárolni. Ha abból indulunk ki, hogy 1 byte elég egy tégla tárolására, hiszen feltételezzük, hogy nem lesz több téglafajta 255-nél, akkor egy képernyőnyi terep leírásához elég 16x10, azaz **160 byte**. Fontos, hogy ahol nulla van oda nem kerül tégla, az lesz az a terület ahol a mi figuránk, meg a 3db sprite mozogni fog. Ha megnézzük a kész exe-t, akkor láthatjuk, hogy se a figuránk, se a spriteok nem lépnek bele a téglákba, hanem az üres mezőkben maradnak. Később látni fogjuk, hogy rengeteg szerepe van ennek a momentumnak. Saját figuránk mozgását például aszerint engedélyezzük vagy nem engedélyezzük, hogy ott ahova éppen lépni akar mit találunk a képernyőn. Tehát ha nem üres az előtte lévő byte akkor ott biztosak lehetünk, hogy ott tégla van. Az előzőekből kiindulva a terepek megtervezésénél máris 2 aranyszabályt be kell tartanunk. Az egyik az, hogy **ahol nincs**



tégla oda mindig 0-t kell tenni a képernyőn. Ezt úgy érjük el a terep kirakásánál, hogy ahol a terepleíróban 0 van, oda egy olyan téglát rakunk a képernyőn, ami csak nullából áll. A másik az, hogy a **tégla grafikus adataiban nem szerepelhet a 0**. Ha feketét akarunk használni, akkor egy másik palettát kell választani, aminek RGB-je hasonlóan fekete.
Egy terep teljes leírása tartalmazza még a begyűjtendő kincsesláda X,Y koordinátáit a képernyőn, és tartalmazza a 3db sprite adatait, sorban. Egy sprite adatrekordja a következő.
X,Y koordináta - word,
Xrelatív,Yrelatív - byte és az elmozdulását jelenti pixelben a spritenak. Előjeles érték tehát, az 1=1 és a -1=129.
Max lépés - byte: ez határozza meg, hogy összesen hányszor lép egy irányba a sprite.
Spriteszám - byte: ez határozza meg, hogy melyik sprite kerüljön oda.
Tipus - byte: ez határozza meg, hogy milyen tipusú a sprite (2 tipust különböztetünk meg 0,1).
Ha most összeszámoljuk egy **teljes terepleíró** hosszát, akkor láthatjuk, hogy **191 byte hosszú**, ami egész jó. A jelenlegi 16 db terep tárolásához elég 16x191=3056 byte. A program jelenlegi állapota megenged **256 db terepet** amiből mi csak 16-ot terveztünk meg. Később bárki tervezhet még, vagy a már meglévők helyett készíthet újat. A 256 terep is csak 48896 byte hosszú.

A TerepKi procedúra elmélete:

A TerepKi procedúra elmélete pofon egyszerű, és a hosszától sem kell megijedni. Először is beolvassa, hogy hanyadik terepet kell kirajzolni a terepszám memóriaváltozóból. Ezt a számot megszorozza 191-el, így kap egy számot amit hozzáad a legelső terep legelső byte-jának az offsetcíméhez. Ezáltal megkaptuk azt a címet, ahonnan elkezdhetjük felolvasni az aktuális terep adatait. Azt már tudjuk, hogy ez 160 db byte, ami a terep tégláit tárolja olvasási irányban. Tehát ha elkezdjük felovasni sorban a 160 db byte-ot, akkor tulajdonképpen balról jobbra és fentről lefelé haladva megkapjuk a kirakandó téglák számát.

Ezután következik egy 10x16-os kettős ciklus, ami tulajdonképpen úgy néz ki, hogy egy 16-os ciklust 10-szer hajt végre. Ez azért jobb mintha egyszerűen egy 160-as ciklust írtam volna, mert így nem kell vizsgálni, hogy mikor érek egy sor végére. A belső ciklusmag a döntő. Itt olvasom ki a tégla számát, és állapítom meg a téglaszám alapján a kirajzolandó tégla adatainak címét a memóriában. Ezt is megoldhattam volna egyszerűen szorzással, tehát a tégla sorszámát megszorzom egy tégla adatainak a hosszával, de akkor ragaszkodni kellett volna a méretazonossághoz, és az adatok sorrendiségéhez a memóriában. Bár a méretazonosság a programban megvan, én inkább a hosszabb kódot igénylő, de még így is tökéletesen gyors egyenkénti vizsgálatot választottam. Itt van időnk, mert ha belegondolunk ez a rutin minden terep elején fut csak le, és csak egyszer. A végleges TerepKi procedúrában ezután még beolvassa a kincs X,Y koordinátáit, utána meg a 3 sprite fent ismertetett adatait a saját rendszerváltozóinkba. Erre azért van szükség, mert azok a rutinok amelyek mozgatják a sprite-kat, mindig innen olvassák ki azokat az adatokat amelyekre szükségük van. Így nem kell minden egyes spritehoz külön megírni azokat a rutinokat, amelyek kezelik a mozgását.

Következő rész tartalma:

A következő részben teljes egészében átvesszük a spriteok méreteit, adatait, hogy kell megrajzolni, kirakni és nem utolsósorban mozgatni őket. Szó lesz a közös paletta használatáról, és a rendszerváltozók teljes körű ismertetéséről. Így az első három rész ismeretében ezek a rutinok teljesen áttekinthetőek lesznek.

A következő oldalon megtaláljuk a forrásokat.

Szalay Zsolt

```
Terepki Proc
        push    cx
        push    si
        push    di

        mov     al,terepsz
        mov     cl,191
        mul     cl
        mov     si,offset terepek
        add     si,ax                   ; Aktuális
                                        ;terep adatainak a cime
        mov     di,0
        mov     cx,10

tc1:    push    cx
        mov     cx,16
tc0:    mov     al,ds:[si]
        cmp     al,1                    ; Tégla 1
        je      tegla2
        cmp     al,2
        je      tegla3                  ; Tégla 2
        cmp     al,3
        je      tegla4
        cmp     al,4
        je      tegla5
        cmp     al,5
        je      tegla6
        cmp     al,6
        je      tegla7
        cmp     al,7
        je      tegla8
        cmp     al,8
        jne     tc7
        jmp     tegla9
tc7:    cmp     al,9
        jne     tc6
        jmp     tegla10
tc6:    cmp     al,10
        jne     tc5
        jmp     tegla11
tc5:    cmp     al,11
        jne     tc2
        jmp     tegla12
tc2:    cmp     al,12
        jne     tc3
        jmp     tegla13
tc3:    cmp     al,13
        jne     tc4
        jmp     tegla14
tc4:    cmp     al,14
        jne     tegla1
        jmp     tegla15
tegla1: push    si
        mov     si,offset tegl0
        call    spritem
        pop     si
        jmp     ttov

tegla2: push    si                      ; Tégla1 kirakása
        mov     si,offset tegl1
        call    spritem
        pop     si
        jmp     ttov

tegla3: push    si                      ; Tégla2 kirakása
        mov     si,offset tegl2
        call    spritem
        pop     si
        jmp     ttov

tegla4: push    si                      ; Tégla3 kirakása
        mov     si,offset tegl3
        call    spritem
        pop     si
        jmp     ttov

tegla5: push    si                      ; Tégla4 kirakása
        mov     si,offset tegl4
        call    spritem
        pop     si
        jmp     ttov

tegla6: push    si                      ; Tégla5 kirakása
        mov     si,offset tegl5
        call    spritem
        pop     si
        jmp     ttov

tegla7: push    si                      ; Tégla6 kirakása
        mov     si,offset tegl6
        call    spritem
        pop     si
        jmp     ttov

tegla8: push    si                      ; Tégla7 kirakása
        mov     si,offset tegl7
        call    spritem
        pop     si
        jmp     ttov

tegla9: push    si                      ; Tégla8 kirakása
        mov     si,offset tegl8
        call    spritem
        pop     si
        jmp     ttov

tegla10:push    si                      ; Tégla9 kirakása
        mov     si,offset tegl9
        call    spritem
        pop     si
        jmp     ttov

tegla11:push    si                      ; Tégla10 kirakása
        mov     si,offset tegl10
        call    spritem
        pop     si
        jmp     ttov

tegla12:push    si                      ; Tégla11 kirakása
        mov     si,offset tegl11
        call    spritem
        pop     si
        jmp     ttov

tegla13:push    si                      ; Tégla12 kirakása
        mov     si,offset tegl12
        call    spritem
        pop     si
        jmp     ttov

tegla14:push    si                      ; Tégla13 kirakása
        mov     si,offset tegl13
        call    spritem
        pop     si
        jmp     ttov

tegla15:push    si                      ; Tégla14 kirakása
        mov     si,offset tegl14
        call    spritem
        pop     si
        jmp     ttov

ttov:   inc     si                      ; Ha üres a tégla
        add     di,20
        loop    tc000
        jmp     ttcc
tc000:  jmp     tc0
ttcc:   add     di,5440
        pop     cx
        loop    tc1b
        jmp     tctov
tc1b:   jmp     tc1

tctov:  mov     ax,[si]                 ; kincs koordinátái
        mov     kincsx,ax
```

```
        mov     ax,[si+2]
        mov     kincsy,ax
        mov     al,1
        mov     kincsj,al

        mov     ax,[si+4]
        mov     ny1_x,ax
        mov     ny1_xer,ax
        mov     ax,[si+6]
        mov     ny1_y,ax
        mov     ny1_yer,ax
        mov     al,[si+8]
        mov     ny1_xr,al
        mov     al,[si+9]
        mov     ny1_yr,al
        mov     al,[si+10]
        mov     ny1_mlep,al
        mov     al,[si+11]      ;szám
        xor     ah,ah
        mov     cx,2880
        mul     cx
        mov     ny1_cim,ax
        mov     al,[si+12]      ;tip
        mov     ny1_tip,al

        mov     ax,[si+13]
        mov     ny2_x,ax
        mov     ny2_xer,ax
        mov     ax,[si+15]
        mov     ny2_y,ax
        mov     ny2_yer,ax
        mov     al,[si+17]
        mov     ny2_xr,al
        mov     al,[si+18]
        mov     ny2_yr,al
        mov     al,[si+19]
        mov     ny2_mlep,al
        mov     al,[si+20]      ;szám
        xor     ah,ah
        mov     cx,2880
        mul     cx
        mov     ny2_cim,ax
        mov     al,[si+21]      ;tip
        mov     ny2_tip,al

        mov     ax,[si+22]
        mov     ny3_x,ax
        mov     ny3_xer,ax
        mov     ax,[si+24]
        mov     ny3_y,ax
        mov     ny3_yer,ax
        mov     al,[si+26]
        mov     ny3_xr,al
        mov     al,[si+27]
        mov     ny3_yr,al
        mov     al,[si+28]
        mov     ny3_mlep,al
        mov     al,[si+29]      ;szám
        xor     ah,ah
        mov     cx,2880
        mul     cx
        mov     ny3_cim,ax
        mov     al,[si+30]      ;tip
        mov     ny3_tip,al
        xor     al,al
        mov     ny1_lep,al
        mov     ny2_lep,al
        mov     ny3_lep,al
        mov     ny1_faz,al
        mov     ny2_faz,al
        mov     ny3_faz,al
        pop     di
        pop     si
        pop     cx
        call    mpoki
        call    poki
        ret
        EndP
```

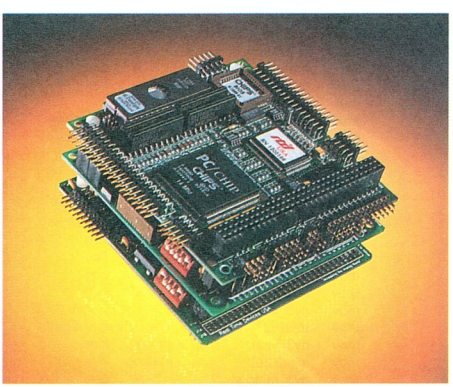
```
;Egy terepleíró rekord adatszerkezete a forrásfile-ban:

terepek
        db      1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
        db      1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,1
        db      1,1,0,0,0,0,0,0,0,1,1,1,1,1,1,1
        db      1,0,0,0,0,4,0,0,0,0,0,0,0,0,0,0
        db      1,0,0,0,0,1,1,1,0,0,0,0,4,0,0,0
        db      1,0,0,0,0,0,0,1,1,0,0,1,1,0,0,0
        db      1,1,1,0,0,0,0,0,0,0,0,0,0,0,1,1
        db      1,1,1,0,0,4,0,0,0,0,0,0,0,1,1,1
        db      1,1,1,1,1,1,1,0,0,0,0,0,1,1,1,1
        db      1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

        dw      240,18          ;kincs(x,y)
        dw      70,36           ;ny1(x,y)
        db      0,2             ;ny1(xr,yr)
        db      45              ;ny1(m_lep)
        db      1               ;ny1(sz_m)
        db      1               ;ny1(tip)
        dw      195,54          ;ny2(x,y)
        db      0,3             ;ny2(xr,yr)
        db      30              ;ny2(m_lep)
        db      9               ;ny2(sz_m)
        db      0               ;ny2(tip)
        dw      100,18          ;ny3(x,y)
        db      3,0             ;ny3(xr,yr)
        db      40              ;ny3(m_lep)
        db      8               ;ny3(sz_m)
        db      0               ;ny3(tip)
```

PC/104

Pc Szendvics

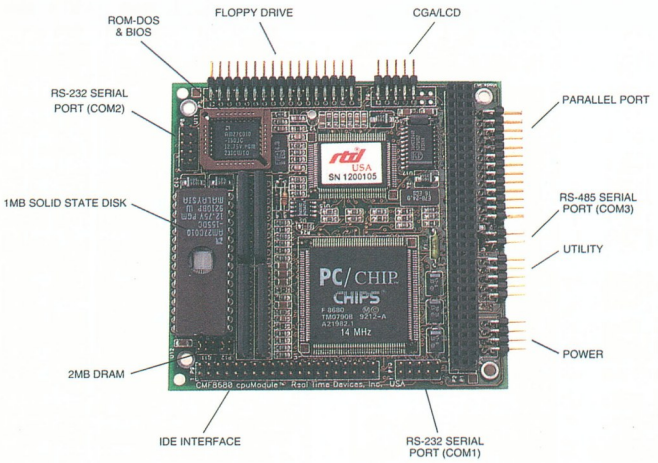
Egy technológiai újításnak köszönhetően a Pc világ új területekre törhet be, az asztali alkalmazásokon túl az ipar és a háztartások gépeiben élhet tovább.

Az elmúlt tíz évben a PC-k igen elterjedté és népszerűvé váltak, ellepték az irodákat, betörtek a lakásokba is. Ma a számítógépek fejlődése soha nem látott tempóban halad előre. Egyre többen ismerkednek meg a kezelésével és sokan a működésével is. A nagytömegű gyártás következtében az árak, különösen ha a teljesítményhez viszonyítjuk, rohamosan csökkennek. A technika, a mikroprocesszorok és egyéb csipek rohamos fejlődése a computerek méretének és energiafelhasználásának igen jelentős csökkenését eredményezte. A relatív olcsóság, a széles körben ismert felépítés, a szoftverek elterjedtsége, szinte szabványossága olyan előnyök, melyeket a fejlesztők nem hagyhatnak figyelmen kívül. Így egyre több területen ahol valamit mérni, irányítani, vezérelni kell, Pc-arhitektúrát alkalmaznak. Háztartási gépekbe, orvosi és laboratóriumi műszerekbe, robotokba építenek Pc-t. Ezzel jelentősen csökkenthetik a fejlesztési időt, a költségeket és a kockázatot, hiszen egy ismert, bevált utat járnak. A Pc kifejezés hallatán a többség egy asztali számítógépet, egy fémládát, egy monitort képzel maga elé. Ebben a formában az esetek többségében a méretei miatt nem alkalmazható. A rezgéseket, a vibrációt rosszul tűri, szűk hőmérséklet-tartományban használható. Máskor az okoz problémát, hogy a Pc-alkatrészeket senki sem minősíti sem megbízhatóság, sem a várható élettartam szempontjából. Mit volt mit tenni, a fenti hiányosságokat megoldva újra kellett tervezni a Pc-t úgy, hogy a kompatibilitás ne sérüljön. Az új Pc-t a technikailag lehető legkisebbre építették, a Pc-buszt masszív tűs csatlakozókra vezették ki. A bővítő kártyákat azonos méretűre tervezték, azonos megbízható 64 + 40 pontos tűs csatlakozókkal, így egyszerűen egymásba dughatók. Az új rendszer a csatlakozó tűk számáról a PC/104 nevet kapta. A modulok CMOS technológiával készülnek, így kevés energiát igényelnek. A kártyák mérete egységesen 96 x 91 mm. A kártyák egymásba helyezésével önálló felépítmények hozhatók létre. A modulok közötti távolság ilyenkor 15 mm. Egy három modulos felépítmény méretei pl: 9.1 x 9.6 x 5 cm. A PC/104 tökéletesen eleget tesz a hagyományos PC-k szoftver-, és hardver követelményeinek, 1992-től szabványnak tekinthető. A hagyományos Pc-kártyák mindegyike létezik ebben a méretben is. Gyártók tucatjai készítenek CPU, SVGA, Arcnet, Ethernet, modem és fax-kártyákat. A szoftvert általában ROM-diszkről futtatják, ezt néha ráépítik az alapkártyára, de nagyobb kapacitásigény esetén külön kártyán helyezik el. Háttértárként jellemző a PCMCIA kártyák használata. A külvilággal való kapcsolattartásra rengeteg lehetőség adódik. Több tucat digitális io kártya, analóg mérő-, és vezérlő kártya, időzítő és soros (RS-485) kártya kapható. Egyre bővül az apró alkatrészek, kábelek, dobozok, tápegységek választéka is, bár jelenleg ez a rendszer gyenge pontja. A felhasználás általában úgy történik, hogy az egymásba dugott modulokból épített Pc-t közvetlenül beépítik a vezérelt gépbe. A modulok alkalmazhatóak alkatrészszerűen is. Ilyenkor a modulokat



1.ábra Egy számítógép szendvics.

valamilyen - általában a felhasználó által tervezett - alapkártyában helyezik el, amely más egyéb, az alkalmazásnak megfelelő áramköröket tartalmazhat. A PC/104 terjedésének bizonyítéka, hogy a más logikát követő hagyományos ISA buszos processzor kártyákon is feltűnt a PC/104 csatlakozó. Az egymásba dugható modulok kíválóan alkalmasak arra, hogy többféle modult helyezhessünk ugyanarra a helyre. Ez könnyebbé teszi a készülék későbbiekben való továbbfejlesztését, valamint lehetővé teszi ideiglenes modulok használatát teszteléskor, hibakereséskor. A rendszer terjedésének manapság még gátat szab az ár. Napjainkban még az azonos tudású hagyományos Pc-kártyák árának kétszereséért vehetünk PC/104 kártyát.



2.ábra Egy tipikus PC/104 CPU kártya.
Mérete : 96x91 mm.
Fogyasztása: 1.25 W

A 2. ábrán egy tipikus PC/104 processzor kártyát láthatunk, ez felel meg a normál pc alaplapjának, sőt több is annál. Nézzük a legszembetűnőbb kölönbségeket. Abban még nem lenne semmi különös, hogy a CPU kártyára integrálva találjuk az IDE lemezvezérlőt, de ezen felül még egy 1 Mb kapacitású ROM is található. Erről indul az operációs rendszer és sokszor az alkalmazói program is. Ez gyors és biztos indítást tesz lehetővé, például vírusfertőzéstől nem kell tartanunk. Ezt egészíti ki az úgynevezett Watchdog Timer. Ez egy olyan áramkör ami figyeli a pc-busz adatforgalmát. Ha egy előre beállított ideig nincs adatforgalom, azt hibaként értékeli, és újraindítja a rendszert. Ezért is fontos a gyors és korrekt indítás. Az alaplapon 2 Mb RAM be van forrasztva, így egy része RAM-lemezként használható. Operációs rendszerként vagy a DOS egy régebbi változata (kisebb helyet igényelnek) vagy egy DOS kompatibilis multitaszkos operációs rendszer alkalmazható (RTKernel). A kis méret ellenére elfért egy kétirányú párhuzamos és három soros port, két RS-232 és egy RS-485. Ipari környezetben szívesen használják az RS-485 szabványú soros kapcsolatot, nagyobb távolság hidalható át akár két vezetékkel is. Az elektromos zavaroktól is jobban védett.

Természetesen a Pc billentyűzet és hangszórócsatlakozó is megtalálható, az már különlegesség, hogy CGA kompatibilis LCD panel meghajtóját is el tudták helyezni egy 3.5"-os lemeznél is kisebb kártyán. A csatlakozásokhoz a kis méretek miatt speciális kábelek, csatlakozók kellenek.

A kártya processzora egy XT-kompatibilis 14 Mhz-en futó 16-bites processzor. Más gyártók erősebb processzorral is készítenek CPU kártyát, természetesen akkor valami lemarad róla, vagy ahogy már említettem a CPU kártyát ISA buszos változatban készítik el, PC/104 csatlakozóval. Létezik 100 Mhz órajelű 486-os processzorral 64 Mbyte RAM-mal szerelt VESA buszos változat is. A rendszer erősségét a rendszerbe illeszthető mérő-, és beavatkozó kártyák sokasága adja, melyek száma napról napra bővül.

Nagy Sándor

UNIX Rovat

Próbaszámunkban két rövid shell programot ismertettünk, melyeket a nagy érdeklődésre való tekintettel most újra leközlünk. Az egyik a magyar ékezetek használatához adott segítséget, a második a kurzor pozicionálására adott ötletet. Ezenkívül a shell használatának előnyeiről esett szó és shell programok írására buzdítottuk az olvasókat.

Még mielőtt folytatnánk a múltkor megkezdett témát, néhány gondolatot fordítsunk arra, hogy mennyivel nyújt többet a UNIX használata.

Az úgynevezett multiscreen funkció amely lehetővé teszi, hogy egyszerre több ablakban dolgozhassunk, nagyon jól kihasználható. Az egyik ablakban írjuk a programunkat, a másikban fordíthatjuk, a harmadikban futtathatjuk és a hibakeresést végezhetjük. Persze ez ma már nem nagy újság a Windows használói számára. Ezek felszíni hasonlóságok, az igazi eltérés abban van, hogy a programok futás közbeni kommunikációját az operációs rendszer számos funkciója támogatja. Ezek programozókat érdeklő kérdések. A felhasználók szempontjából talán a legfontosabb, hogy a UNIX-os rendszerben dolgozók megtanulják a szervezett keretben való csoportmunkát. A rendszer keretet ad az együttműködéshez.

Az előző részben ígéretet tettem a kurzormozgatás egy gyorsabb megvalósítására. Az ötlet igen egyszerű. A kurzor mozgatási utasítást tároljuk egy környezeti változóban. A shell és általában a programok többsége első feladatként éppen ezeket a változókat olvassák ki, hogy tartalmaz e beállítást. Így a beállítást a program a tényleges program előtt hajtja végre.

A leggyakoribb programozói feladat a menü készítés. Lássunk egy shell menü programot mintaként. A program megértéséhez érdemes kezdőknek átnézni a do és a case használatát.

```
:menu program, ennek a mintájára írható a
saját
USAGE='usage: menu'
ANSWER=""
while true
do
        if test "$ANSWER"=""
        then
#a CLEAR változót vagy a shell-ben vagy a
.profile változóban
#meg kell adni
#cat SCREEN-tôl SCREEN-ig kiír mindent a
képernyôre
        cat <<SCREEN
$CLEAR     UNIX MENU
Number Name  For
0     Exit  Kilépés
1     vi    Fájl szerkesztés
2     sh    új shell indítása

Add meg a menu számát, vagy DEL-t
kilépéshez

SCREEN
read ANSWER COMMENT
fi
case $ANSWER in
0|exit)  exit 0 ;;
1|vi)
        echo'Add meg a fájlnevet'
        read ANSWER COMMENT
        vi $ANSWER $COMMENT
        ;;
2|sh)    sh;;
*)       echo 'Ez nem megfelelô szám!';;
esac
echo '
Nyomj ENTER-t ha újra látni akarod a menüt'
        read ANSWER
done
```

Ebben a sorozatban a UNIX shell használóinak, programozóinak szeretnénk fórumot biztosítani. Lehetővé szeretnénk tenni gyakorlattal rendelkezőknek a tapasztalatcserét és a kezdők találkozását életszagú mintákkal. Szívesen veszünk minden platformról forrásokat, észrevételeket, tapasztalatokat és kéréseket.

Aki már használta a UNIX shellt, annak nem kell bizonyítani, hogy a mai formájában rendelkezik ugyanazokkal a képességekkel, mint a 4. generációs programnyelvek. Könnyen tanulható. Rendkívül tömör programokat készíthetünk. Fájlok csoportjait kezelhetjük egyszerre. A bővítése egyszerű: saját shell kisügyeseink mellett, akármely a gépünkön rendelkezésre álló programnyelven megírhatjuk kiegészítéseinket. Az awk, grep, sed segítségével meglepően bonyolult adatkezeléseket is meg lehet oldani. A fejlesztésre fordítandó idő lényegesen kevesebb, mint bármely más programnyelv esetén. Prototípusok, minták ismeretében még hatékonyabbá tehető a fejlesztői munka. Annak a számára, aki ezt megtanulta, jártas benne, minden UNIX rendszer nyitott, szabad a mozgástere. Persze nem futási sebességrekordokat kívánunk megdönteni, de a gyors eredmény mindenkit kárpótol. Ne ítéljünk előre. A futási sebességgel kapcsolatban kellemes meglepetések érik a rutinos shell programozókat. A bonyolultabb rendszerek telepítése is shell programok segítségével történik. Ezek akár néhány száz k méretűek is lehetnek. Id Lotus 123, Wordperfect, Dataflex. Ne ijedjünk meg tőlük, kíváló ötlet tárházak! Tanuláshoz, hibakereséshez használjuk az operációs rendszerrel kapott kézikönyvet, vagy amennyiben érdeklődés mutatkozik, néhány szak-könyvet a későbbiekben ajánlhatunk. Indítsuk ezt a sorozatot egy egyszerű programmal, amely a magyar ékezetes karakterek használatát segíti. Hasznos minden olyan eszközök használatában, amely az ASCII tábla 127 felett bizonyos kódokat saját maga értelmez, pl. nroff,troff.

```
1:#!/usr/bin/ksh
2:# Készült COHERENT 4.0-ra
3:# Magyar karakterek át és visszacsoportosítása
4:# 1993.aug."BECO"Kft.
5:while test $# -gt 0
6:do
7:        case $1 in
8:                -mc) arg=$1;;
9:                -ms) arg=$1;;
10:               -n*) o1=$1;;
11:               -r*) reg=$1;;
12:               *) szveg=$1;;
13:       esac
14:shift
15:done
16:
17:# az óúíş marad, a többi átalakul:
18:
19:cat $szveg\
20:|tr  "öüóűéáÖÜÓŰÉÁëä" "Ľ«_ń__˝»®Ń¬žŞ˙_"\
21:|nroff $arg $o1 $reg\
22:|tr "Ľ«_ń__˝»®Ń¬žŞ˙_" "öüóűéáÖÜÓŰÉÁëä"
```

Második programunk egy ügyes kurzor pozicionáló módszert mutat be.
A parancssor legyen:
$cursor sor oszlop
A kurzor pozicionálás terminál függő, a módját megtaláljuk a /etc/termcap fájlban, (cm) bejegyzés alatt. Hozzunk létre egy CURSOR nevű környezeti változót, cursor ezt fogja megtalálni környezetében. Egy pc terminál részére írjuk be az alábbi sort a .profile fájlunkba. (^[az ESC karakter, vagy Ctrl [)

```
export CURSOR=^[[%d\;%dH
```

A cursor program:

```
1:#!/usr/bin/sh
2:# készült COHERENT 3.2-re
3:# 1992.maj."BECO"Kft.
4:# !/usr/bin/ksh, COHERENT 4.0-ra OK!
5:# 1993.aug."BECO"Kft.
6:USAGE='usage: cursor sor oszlop'
7:EUSAGE=1
8:
9:        if test "$#" -ne 2
10:       then
11:       echo $usage 1>&2
12:       exit $EUSAGE
13:       fi
14:
15:echo `echo "$CURSOR"
16:sed   "s/%./$1/
17:       s/%./$2/"`\\c
```

A cursor, clear, echo segítségével néhány sorban látványos dolgokat rajzolhatunk a képernyőre, és a read segítségével beolvashatjuk a felhasználó válaszait. A képernyőn mezőket törölhetünk a kurzor mozgatásával és betűközök írásával. Ezzel a módszerrel a képernyő színezése is megvalósítható. Ötleteket várunk. Legközelebbi alkalommal a cursor-nak egy továbbfejlesztett, gyorsabb változatát fogjuk bemutatni. Várjuk leveleiket, ötleteiket. Ne feledjék el megadni, hogy programjuk eredeti változata mely operációs rendszer alatt készült és lett kipróbálva.

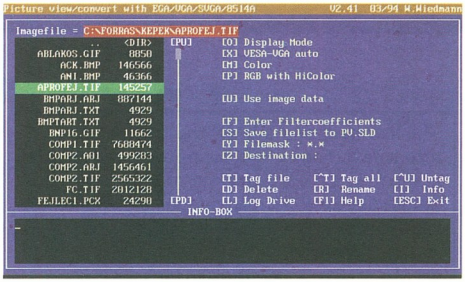
Berta Sándor
tel.: 217-4578 "BECO"Kft.

PVE

Az utóbbi idők legsokoldalúbb nézőkéje a német eredetű pv.exe illetve angol nyelvű megfelelője a PVE.EXE

A lemezmelléklet ANI alkönyvtárában található PVE.EXE az egyik legsokoldalúbb nézőke. Az 1.listán látható az illesztett formátumok listája. Már a megjeleníthető képformátumok száma is impozáns, de ezen kívül animációkat is lejátszhatunk és MOD zene file-okat is megszólaltathatunk. Az összes elterjedt videokártyával működik: ATI, Paradise, Ahead, Video7, OAK, Chip & Technology, ET3000/4000 és Trident 8900 1024x768 felbontásig 256 színig. Ismeri 8514/A, TIGA és XGA szabványokat, a 1280x1024x16/256 és 1600x1280x16/256 üzemmódok is be vannak építve, de kipróbálva nincsenek. Ezek szerint a szerzőnek sem megy jobban mint nekünk. Csodák persze nincsenek, ha túl nagy kép-filet akarunk megjeleníteni az nem fog sikerülni, "kevés a memória" üzenettel elszáll a programunk. Ezt azért lehetne elegánsabban is. Újdonság a text file grafikus megjelenítése. Azokat a file- kat amiket kiterjesztés alapján nem ismer fel az általunk megadott módon próbálja megjeleníteni. Az egyszerű PVE indítás után filemanager ablak jelentkezik, paraméterrel indítva ez elmarad, és a program a paraméterben megadott módon viselkedik. Használhatjuk képformátumok egymásba való átalakítására is az 1. lista szerint.

1.ábra Egy kép a lemezről



2.ábra A PVE képernyője

1.lista Illesztett formátumok

Írja olvassa az alábbi formátumokat:

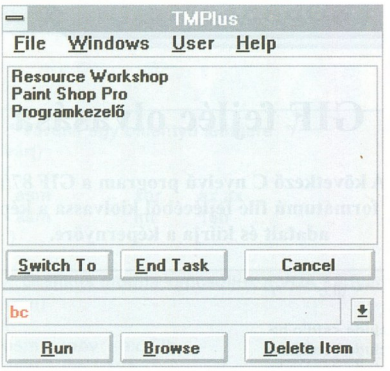
- JPG	JPEG 8R8.
- TIF	IBM és Macintosh
- TGA	Targa
- PCX	Paintbrush
- PIC	PcPaint, Pictor.
- SCx,RIX	Colorix, Winrix
- CUT	Dr Halo
- LBM,IFF	Deluxe Paint
- GIF	Compuserve
- BMP,RLE,DIB	Windows 3.0,3.1 and OS/2.
- IMG,DTA	Kontron, ZEISS

Csak olvassa az alábbi formátumokat:

- PCD	Kodak Photo CD 768x512.
- CVP	Passport image 512x512 24 szín
- JPG,JIF	JPEG 9R6
- JTF	TIFF (JPEG tömörítéssel)
- CEG	Edsun
- GEM-IMG,XIMG	GEM and Ventura Publisher, Atari ST.
- MSP	MS-Paint, Windows 2.0.
- MAC	Macintosh-Paint.
- PIC	Macintosh PICT in 1,2,4,8 BPS.
- WPG	Raster graphic
- ICO	Windows Icons.
- PMC	A4TECH Scanner.
- SFI	SIS Framegrabber.
- EPS	Encapsulated Postscript
- RAS	Sun raster file
- VI	Jovian.
- DCX	FAX Multiple PCC File Format.
- IM	KO-23 satellite
- SGF	Starwriter
- SAT	Eumetsat
- SCR	Word capture
- CDR,CCH	Corel draw Icon
- SKD	Autosketch Icon
- RAW	
- PNM	UNIX Portable Bitmap 1,8,24 Bps
- ACB,BBM	IFF Brushes and ACBM Files.
- DAT	Framegrabber Video 1000/2000
- FLM	Screenmachine FAST-Electronic.
- AVI	Video for Windows
- FLI	Autodesk Animator
- FLC	Autodesk Animator
- VM	Videomachine FAST-Electronic.
- ANI	IFF animation files.
- DL	DL animations run in realtime.
- GL	
- TXT,DOC,BAT,HLP	Text
- ANS	ANSI 80x25
- MOD	Amiga zene file

TmPlus

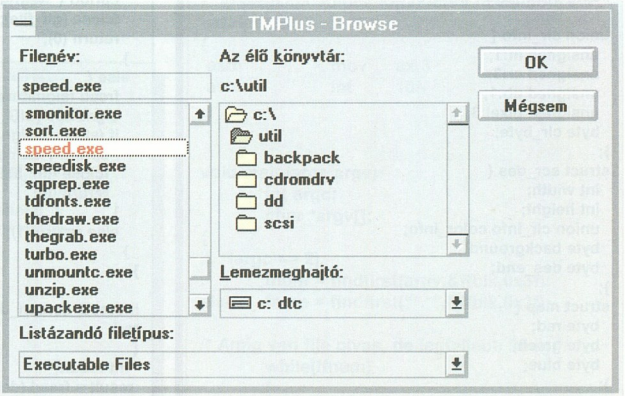
Sokan nem szeretjük a Windowst, de nem is nélkülözhetjük. Mit tehetünk mást, javítgatjuk.



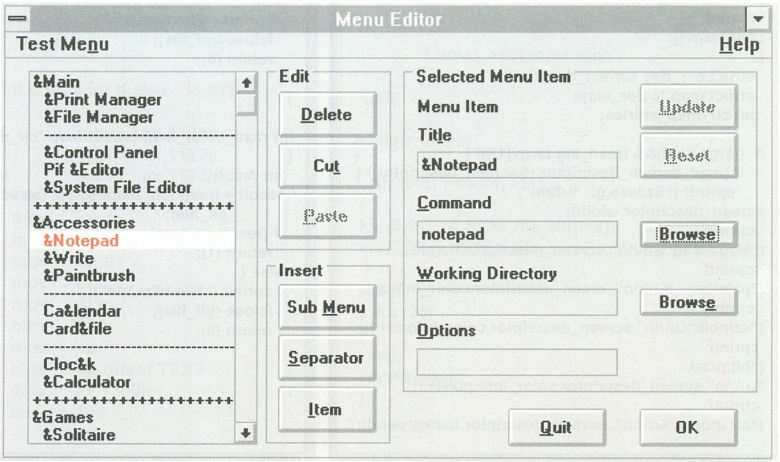
1.ábra A TMPLUS ablaka a parancssorral.

A lemezmelléklet TMPLUS alkönyvtárában találjuk az önkicsomagoló TMPLUS_.EXE filet. Egy üres könyvtárban elindítva az eredményt másoljuk a WINDOWS könyvtárba. Ezek után a SYSTEM.INI file boot szekciójában helyezzük el a TASKMAN.EXE=TMPLUS.EXE sort. Ez után a Ctrl-Esc billentyű kombináció leütése után az 1.ábra szerinti popup ablak jelenik meg a megszokott task-manager helyett. Azonnal látszik az újdonság a menü és a parancssor. Egy régi DOS felhasználónak valószínűleg ez utóbbi az érdekesebb tetszés szerinti programot indíthatunk, és a DOS külső parancsait futtathatjuk. A belső parancsokról és a kimenet átirányításáról le kell mondanunk. A file-ok indítását megkönnyíti a file-kereső ablak. Az utoljára használt parancsokat legördülő menüből választhatjuk ki. Ezeket a parancsokat a program egy szövegállományban tárolja így könnyen szerkeszthetők.

A másik újdonság a felső menüsor ahonnan Windows programok indíthatók illetőleg a Windows alapvető funkcióit vezérelhetők. Ami miatt ez valóban használható: a menü egy szerkesztő ablak segítségével tetszés szerint módosítható szerkeszthető. A menüt a program egy bináris állományban tárolja, kézzel ne piszkáljuk.



2.ábra Fent a file-kereső, lent a menűszerkesztő ablak.



GIF fejléc olvasása

A következő C nyelvű program a GIF 87/a formátumú file fejlécéből kiolvassa a kép adatait és kiírja a képernyőre.

```
/*        gi.c       */

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define byte char
#define SIG_SIZ 6
#define GIF_SIG "GIF87a"

union clr_info {
  unsigned m:1;
  unsigned cr:3;
  unsigned ub:1;
  unsigned pixel:3;
  byte clr_byte;
};
struct scr_des {
  int width;
  int height;
  union clr_info color_info;
  byte background;
  byte des_end;
};
struct map {
  byte red;
  byte green;
  byte blue;
};
FILE *gif_file;


void main(argc,argv)
int argc;
char *argv[];
{
  struct scr_des screen_descriptor;
  struct map *color_map;
  int clr_map_entries;

if  ((argc == 2)&& (read_sig (argv[1]))) {
    if (read_screen_descriptor (&screen_descriptor)) {
      cprintf ("Sz,less,g:  %d\r\n",
screen_descriptor.width);
  cprintf
("Magasság:%d\r\n",screen_descriptor.height);
  cprintf
("paletta:    %u\r\n", creen_descriptor.color_info.m);
  cprintf
("szinbit:%u\r\n",screen_descriptor.color_info.cr+1);
 cprintf
("bit/pixel :
%u\r\n",screen_descriptor.color_info.pixel+1);
 cprintf
("clr index :%d\r\n", screen_descriptor.background);
 if (screen_descriptor.color_info.m)
  {clr_map_entries =
          2^(screen_descriptor.color_info.pixel+1);
color_map = (struct map *) malloc (clr_map_entries);
read_color_map (color_map, clr_map_entries);
     }
   }
  }
else puts(" Használat :   Gl filenév ");
}

int read_sig (char *file_name)
{
  char signature [SIG_SIZ+1];
  gif_file = fopen (file_name, "rb");
  if (!gif_file) {
    cprintf ("Nem találom.\r\n");
    fclose (gif_file);
    return (0);
  }
  else {
    fread (signature, SIG_SIZ, 1, gif_file);
    signature [SIG_SIZ] = '\0';
    if (strcmp (signature, GIF_SIG)) {
      cprintf ("Ez nem GIF file\r\n");
      fclose (gif_file);
      return (0);
    }
    else return (1);
  }
}

int read_screen_descriptor (struct scr_des *descrip-
tor)
{
  int result;
 result = fread (descriptor, sizeof (struct scr_des), 1,
           gif_file);
  if (result == 1)
    return (1);
  else {
    cprintf ("File hiba.\r\n");
    fclose (gif_file);
    return (0);
  }
}

int read_color_map (struct map *clr_map, int num)
{
  int result;
  result = fread (clr_map, sizeof (struct map), num,
           gif_file);
  if (result == num)
    return (1);
  else {
    cprintf ("File hiba.\r\n");
    fclose (gif_file);
    return (0);
  }
}
,
```

Karakterek a VGA képernyőn

Az alábbi program kilistázza az aktuális könyvtár tartalmát, vagy a paraméterként megadott szűrő szerinti fileokat a VGA képernyőn a grafikus kártya saját karakterkészletével. A program jól szemlélteti az inline assembler használatát. Fordításkor a TASM.EXE-nek a PATH-ban megadott útvonalon kell lennie.

```
/*********************************************/
/*                                           */
/*              vgadir.c                     */
/*                                           */
/*    fordítás :  TCC -mt -lt   vgadir       */
/*                                           */
/* TASM.EXE -nek  PATH ban kell lennie!  */
/*********************************************/

#include <dir.h>

char file_dir[100][13];
char file_size[100][8];
char file_date[100][13];

int fnum,i=0,z,db,j,c,h,v,imax=0;
struct ffblk ffblk;

/* Az x,y pontba p pont magas betűvel */
/* c színben d számú karaktert kiírunk */
/* a   TEXT  stringből                 */

kiir(int x,int y,int p,int c,int d,char *TEXT)
{
 if(p==8)  asm          mov    ax,1123h
 if(p==14) asm          mov    ax,1122h
 if(p==16) asm          mov    ax,1124h
  asm             int     10h
  asm             mov     ax,ds
  asm             mov     es,ax
  asm             mov     bh,0
  asm             mov     bl,c
  asm             mov     dl,x
  asm             mov     dh,y
  asm             mov     cx,d
  asm             mov     bp,offset TEXT
  asm             mov     ax,1300h
  asm             int     10h
}
```

```
/* Várunk egy billentyű leütésre  */
var()
{
  asm             xor     ax,ax
  asm             int     16h
 }

/* Grafikus módba kapcsoljuk a VGA kártyát  */
vga()
{
asm    mov    ax,12h
asm    int    10h
}

/* Visszakapcsolunk karakteres üzemmódra */
karakteres()
{
 asm             mov     ax,3
 asm             int     10h
 }


void main(argc,argv)
        int argc;
        char *argv[];
{
if   (argc == 2)
        fnum = findfirst(argv,&ffblk,0x3f);
else    fnum = findfirst("*.*",&ffblk,0x3f);

/* Amíg van file olvas, de legfeljebb 99-ig */
        while(!fnum)
        {
        strcpy (file_dir[i],ffblk.ff_name);
        ltoa (ffblk.ff_fsize,file_size[i],10);
        fnum = findnext (&ffblk);
        imax=i;
        i++;
        if (imax >99) imax =99;
        }
vga();

while(z<imax)
{
  for (i=0; i <= 24; i++)
 {
 kiir(3,2+i,14,15,12,file_dir[i+z]);
 kiir(17,2+i,14,15,6,file_size[i+z]);
  }
 var();
 z = z+25;
 }
 var();
 karakteres();
 }
```

Színes APRÓHIRDETÉS

Hirdessen a Forráskódban

Fejléc 125x8 mm.

A Forráskód egy rugalmas hirdetési lehetőséget ajánl Önnek.

Az új típusú blokkhirdetés alapegysége egy 60 x 45 mm.-es hirdetési felületet és egy 60 x 8 mm.-es fejléc, ami mind vízszintes mind függőleges irányban többszörözhető. Hirdetési megrendelést elfogadunk színes és fekete fehér oldalra is. A hirdetések szedését a szerkesztőség végzi, lehetőség van lemezen hozott bittérképek betöltésére, ezért felárat nem számítunk fel. Hirdetési kedvezmények ennél a hirdetési formánál is változatlanul igénybe vehetők.

Fejléc 60x8 mm.

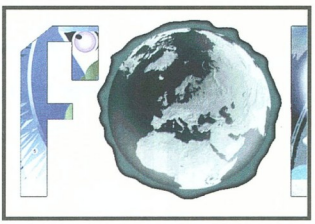
Színes oldalon
színesen szedve
Hirdetési felület 60 x 45 mm.

8.000.-
+ áfa

Fejléc 60x8 mm.

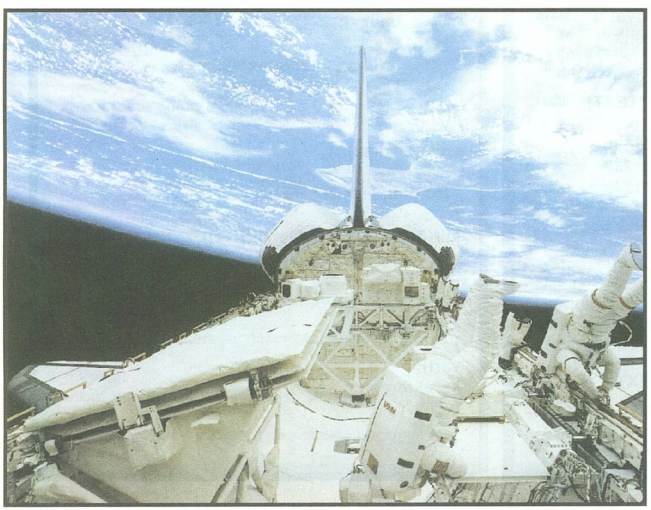
7 pontos betűvel szedve
9 pontos betűvel szedve
10 pontos betűvel szedve
12 pontos betűvel szedve
14 pontos betűvel szedve
18 pontos betűvel sze
24 pontos betűv

Fejléc 60x8 mm.



Fejléc 60x8 mm.

Bittérképet betöltünk.



Fejléc 60x8 mm.

Színes oldalon
színesen szedve

Hirdetési felület 60 x 100 mm.

16.000.

+ áfa

Várjuk érdeklődését a:
134 - 1273-as
telefonszámon
NASA reklámiroda.

Fizesse elő az első magyarnyelvű, programozói magazint!

Havonta lemezmelléklettel megjelenő lapunk ára 298 Ft.-.
Ha Ön előfizeti, kedvezményesen jut lapunkhoz.

Előfizetési díj fél évre:	1494 Ft.-	Így Önnek egy szám csupán:	249 Ft.-
Előfizetési díj egy évre:	2388 Ft.-	Így Önnek egy szám csupán:	199 Ft.-
Iskoláknak egy évre :	1188 Ft.-	Így egy szám csupán	99 Ft.-

Előfizethető a mellékelt csekken,
vagy átutalási postautalványon a
NASA Kft.
218-98172/541-003578-3
számú számláján.

Ctrl - F9
PASCAL ISKOLA

Biztosan előfordult már önnel, hogy arra gondolt: "...de jó lenne erre a problémára egy rövid kis program..., vagy legalább én megtudnám írni ...

A sorozat végigolvasása, a listák végigböngészése után nem okoz gondot egy Pascal program megírása, azaz néhány hónap múlva tudni fog "pascalul", és mindemellett kezében lesz egy file-kereső program forráskódja is. Ez, egy nagyobb rendszer részeként szolgálhat be-, és kimeneti file-k megtalálására, de önmagában is használható directory-váltóként. A kész program a lemezmellékleten található "exe" formátumban. Tesztelje le! Hibát talált? Hiányol valamit? Itt lesz a lehetőség, hogy kijavítsa, vagy átírja az Ön egyéni igényei szerint. Amire szükségünk lesz: mindenekelőtt egy Turbo Pascal fordítóra, ezen belül azonban 5.5-ön felül bármi megteszi. A gépével szembeneni követelmény csak annyi, hogy a monitor szöveges üzemmódban 80x25-ös felbontású legyen.

A filekezelő ablaka (1.ábra) négy részre tagolódik:

1 : Beolvasó rész: Itt lehet megadni egy file elérési útvonalát, maximum 18 karakterben, és itt lehet meghajtót váltani. A begépelt parancssort "Enter"-rel, vagy "Tab"-al fogadhatjuk el. Ha a program értelmező része hibát észlel, automatikusan visszaadja az utolsó útvonalat, majd továbblép.

2: File-lista: Ha jól adtuk meg a directory-t ez az ablak válik aktívvá, és itt lehetőségünk nyílik arra, hogy tovább lépkedjünk a könyvtárak között, vagy kiválasszunk egy file-t. Ha ez megtörtént, üssünk "Enter"-t, vagy "Tab"-ot.

3: OK-gomb: Hatása ugyanaz, mintha "Enter"-t ütöttünk volna az előző részben. Innen is tovább lehet menni "Tab"-al.

4: Kilép-gomb: Az éppen aktuális könyvtárban maradunk, de nem választunk ki file-t. Hatása megegyezik az "Esc"-gomb hatásával.

Egy Turbo Pascal program általános szerkezetét mutatja a **2. ábra.** Látható, hogy egy program három részre tagolódik, úgy mint: programfej, programtörzs, valamint a kettő közötti deklarációs rész. (Kapcsos zárójelek közé kerülő információkat nem veszi figyelembe a fordító.) Ez utóbbiban adhatjuk meg a teljes program futása alatt érvényes globális változókat (var), az általunk definiált új tipusokat (type), iletve konstansokat (const). A fejrészhez tartozó "program" kulcsszó, illetve az utána következő programnév opcionális, azaz ha nem írjuk ki, akkor sem okoztunk hibát, ha azonban programnevet adunk meg, akkor ügyeljünk arra, hogy még egyszer ezt a szót a forráskódban nem szerepeltethetjük. Ugyanez értendő a programnyelvben használt lefoglalt kulcsszavakra is (pl.: begin, end, var, procedure stb.). A globális kapcsolók, illetve a unitok helye a listában kötött, a deklarációs rész elemeié azonban nem. A végrehajtandó utasításokat a "begin....end." által közrefogott programtörzsbe írjuk egymástól ";"-vel elválasztva. A forráskód fordítása lineáris, azaz a "program"-nál kezdődik és "end."-ál ér véget. Ha egy utasítássort többször szeretnénk használni, akkor nem célszerű azt annyiszor leírni ahányszor végre akarjuk hajtatni. Ennek kikerülésére szolgálnak a "procedure"-k (eljárások), illetve "function"-ok (függvények).

Az előre deklarált típusok, amelyeket az I. listában használunk:

1. **Integer:** 2 byten ábrázolt egész típus, értéke: 32768 és 32767 között változhat.
2. **Byte:** 1 byten ábrázolt egész típus, értéke 0 és 255 között változhat.
3. **Char:** Ez a változó 1 byten tud egy karaktert tárolni.
4. **Array:** A fenti két adattípussal szemben (amik egyszerű adattípusok) a tömb az összetett típusok közé tartozik. A tömb elemeinek típusát és számát előre rögzítjük.
5. **String:** Karakterlánc típus maximum 255 karakter tárolására alkalmas változó típus.
6. **Record:** A rekord típusú változó tetszőleges számú és tulajdonságú mezőt (változót) tartalmazhat.

Minden változóhoz hozzá kell rendelni egy előre deklarált típust, amely megadja a változó által felvehető értékek típusát és értékkészletét.

A Turbo Pascal standard utasításai:

Értékadó utasítás: Ennek szintaxisa: a:=b. Az "a"helyére kerül mindig az a változó, aminek értéket akarunk adni, és "b" helyére az a változó (vagy kifejezés) aminek az értékét át szeretnénk adni.

Elágazások:

1. Kétirányú elágazást tesz lehetővé az "if" szerkezet :

```
if   logikai kifejezés   then
                          begin
                           utasítások1;
                          end
                         else
                          begin
                           utasítások2;
                          end;
```

A feltétel teljesülése esetén a "then" ág, ellenkező esetben az "else" ág hajtódik végre. Az "else" ág opcionális, elhagyhatjuk ha nincs végrehajtandó utasítás. Ilyenkor a "then" ág end-je után ";"-t kell tenni! A "begin end" is elhagyható ha csak egy utasítást adunk meg. (Programozástechnikai szempontból érdemes ilyenkor is kitenni, ha több "if" szerkezetet ágyazunk egybe: nem csak a programozó nem fogja tudni, hogy melyik ág melyik logikai kifejezéshez tartozik, hanem a fordító sem.)

2. Többirányú elágazást tesz lehetővé a programban a "case" szerkezet:

```
case  kifejezés  of
 cimke1: begin  utasítások1  end;
 cimke2: begin  utasítások2  end;
 :
else begin  UtasításokN  end;
end;
```

1.ábra A file kereső ablaka.

```
{ Programfej }
program Nev;
{ Globális direktíváik }

{ Deklarációs rész }
uses unit1, unit2, ...
type
cons
var
procedure
function

{ Programtörzs }
begin
end.
```

2.ábra

```
{ Eljárásfej }
procedure Nev( parameterlista:integer );

{ Deklarációs rész }
type
var

{ Eljárástörzs }
begin
end;
```

3.ábra

```
uses Crt;

type
 Sor = array[ 1..25 , 1..2 ]of Byte;
 Kep = array[ 1..80 ]of Sor;

 AblakParam = record
              BFO   : Integer  ;
              BFS   : Integer  ;
              Mag   : Integer  ;
              Szel  : Integer  ;
              Cim   : String   ;
             end;

 NyGombParam = record
               BFO    : Integer ;
               BFS    : Integer ;
               Szoveg : String  ;
              end;
```

```
{-------------------Kepmentes-------------------}
 procedure Kepmentes( var MentettKep : Kep );
  var LathatoKep : Kep absolute $b800:$0 ;
  begin
   MentettKep := LathatoKep;
  end;

{-------------------Kepkiiras-------------------}
 procedure Kepkiiras( MentettKep : Kep );
  var LathatoKep : Kep absolute $b800:$0 ;
  begin
   LathatoKep := MentettKep;
  end;

{-------------------Inverz-------------------}
 procedure Inverz( InvString : String );
  begin
   TextBackGround( White );
   TextColor( Black );
   Write( InvString );
   TextBackGround( Black );
   TextColor( White );
  end;

{-------------------Hatter-------------------}
 procedure Hatter;
  var  Cikl1 , Cikl2 : Integer ;
  begin
   ClrScr;
   for Cikl1 := 2 to 79 do
    for Cikl2 := 2 to 24 do
     begin
      GotoXY( Cikl1 , Cikl2 );
      Write( #176 );
     end;
  end;

{-------------------NyomoGomb-------------------}
 procedure NyomoGomb( GombParam : NyGombParam );
  begin
   GotoXY( GombParam.BFO , GombParam.BFS );
   Inverz( #16 + ' ' + GombParam.Szoveg + ' ' +#17 );
  end;

{-------------------Keret-------------------}
 procedure Keret( Allapot : Boolean ; KeretParam : AblakParam );
  var RajzTomb : array[ 1..6 ]of Char;
      Cikl , KodChr , CimKezdet : Integer ;
  begin
   if Allapot
        then
         begin
          KodChr := 201;
          for Cikl := 1 to 6 do
           begin
            RajzTomb[ Cikl ] := Chr( KodChr );
            case KodChr of
             201 : KodChr := 205;
             205 : KodChr := 186;
             186 : KodChr := 187;
             187 : KodChr := 200;
             200 : KodChr := 188;
            end;
           end;
         end
        else
         begin
          KodChr := 218;
          for Cikl := 1 to 6 do
```

```
        begin
         RajzTomb[ Cikl ] := Chr( KodChr );
         case KodChr of
          218 : KodChr := 196;
          196 : KodChr := 179;
          179 : KodChr := 191;
          191 : KodChr := 192;
          192 : KodChr := 217;
         end;
        end;
       end;
      GotoXY( KeretParam.BFO , KeretParam.BFS );
      Write( RajzTomb[1] );
      for Cikl := 1 to KeretParam.Szel do
       begin
        GotoXY( KeretParam.BFO+Cikl , KeretParam.BFS );
        Write( RajzTomb[2] );
        GotoXY( KeretParam.BFO+Cikl , KeretParam.BFS+KeretParam.Mag+1 );
        Write( RajzTomb[2] );
       end;
      for Cikl := 1 to KeretParam.Mag do
       begin
        GotoXY( KeretParam.BFO , KeretParam.BFS+Cikl );
        Write( RajzTomb[3] );
        GotoXY( KeretParam.BFO+KeretParam.Szel+1 , KeretParam.BFS+Cikl );
        Write( RajzTomb[3] );
       end;
      GotoXY( KeretParam.BFO+KeretParam.Szel+1 , KeretParam.BFS );
      Write( RajzTomb[4] );
      GotoXY( KeretParam.BFO , KeretParam.BFS+KeretParam.Mag+1 );
      Write( RajzTomb[5] );
      GotoXY( KeretParam.BFO+KeretParam.Szel+1 ,
      KeretParam.BFS+KeretParam.Mag+1 );
      Write( RajzTomb[6] );

      CimKezdet := ( KeretParam.Szel - Length( KeretParam.Cim ) )div( 2 ) ;
      GotoXY( KeretParam.BFO+CimKezdet+1 , KeretParam.BFS );
      Write( KeretParam.Cim );
    end;

{-------------Abalak-------------}
procedure Ablak( Jellemzok : AblakParam );
 begin
  Keret( True, Jellemzok );
  Window( Jellemzok.BFO+1 , Jellemzok.BFS+1 ,
          Jellemzok.BFO+Jellemzok.Szel , Jellemzok.BFS+Jellemzok.Mag );
  ClrScr;
  Window( 1, 1, 80, 25 );
 end;

{-------------UzenoAblak-------------}
procedure UzenoAblak( UzenoAblCim , Uzenet : String );
 var FoKep : Kep ;
     UzenoParam : AblakParam ;
     EnterGomb : NyGombParam ;
 begin
  KepMentes( FoKep );
  UzenoParam.Cim := '[ ' + UzenoAblCim + ' ]' ;
  UzenoParam.Mag := 3 ;
  UzenoParam.BFS := 10 ;
  if Length( UzenoParam.Cim ) >= Length( Uzenet )
   then
    begin
     UzenoParam.BFO := 40 - ( ( Length( UzenoParam.Cim ) + 2 )div( 2 ) );
     UzenoParam.Szel := Length( UzenoParam.Cim ) + 2 ;
    end
   else
    begin
     UzenoParam.BFO := 40 - ( ( Length( Uzenet ) + 10 )div( 2 ) );
     UzenoParam.Szel := Length( Uzenet ) + 10 ;
    end;
```

```
  Ablak( UzenoParam );
  GotoXY( UzenoParam.BFO+(UzenoParam.Szel-Length( Uzenet ))div(2) + 1,
          UzenoParam.BFS + 2 ) ;
  Write( Uzenet );
  EnterGomb.Szoveg := 'Enter';
  EnterGomb.BFO := UzenoParam.BFO + UzenoParam.Szel -
                   Length( EnterGomb.SZoveg ) - 3;
  EnterGomb.BFS := UzenoParam.BFS + UzenoParam.Mag + 1 ;
  NyomoGomb( EnterGomb );
  ReadLn;
  Kepkiiras( FoKep );
 end;

{-------------Foprogram-------------}

begin
 Hatter ;
 UzenoAblak( 'üzenet' ,' ...ez bizony hibátlan!' );
 GotoXY( 1, 1 );
 Write( 'Folytatás januárban!' );
end.
```

1.Lista

A címke típusa mindig megegyezik a kifejezés típusával. Az "else" ág akkor teljesül, ha a címkék közül egyik sem egyezik meg a kifejezés aktuális értékével. Az "else" ág (ugyanúgy mint az "if" szerkezetnél) opcionális. Itt is a "begin end"-k elhagyhatók, ha csak egy utasítást akarunk végrehajtatni.

Ciklusok:

A Turbo Pascal-ban három ciklusszervező utasítás van, amiket akkor használunk, ha egy parancssort többször szeretnénk végrehajtatni egymás után.

1. while logikai kifejezés do
begin
utasítások; {ciklusmag}
end;

A ciklusmagban található utasítások mindaddig végrehajtódnak, amíg a logikai kifejezés igaz. Ha a kifejezés már akkor sem igaz amikor a program futása először a logikai kifejezés kiértékelésére kerül, akkor "begin end" közötti ciklusmag egyszer sem hajtódik végre.

2. repeat
utasítások; {ciklusmag}
until kilépési feltétel ;

Itt addig történik az utasítások végrehajtása míg a kilépési feltétel nem teljesül. Mivel a tesztelés a ciklus végén található, ezért a ciklusmag egyszer mindenféleképpen végrehajtásra kerül.

3. for ciklusváltozó := indulóérték to végérték do
begin
utasítások; {ciklusmag}
end;

vagy

for ciklusváltozó := indulóérték downto végérték do
begin
utasítások; {ciklusmag}
end;

Mindkét esetben meg kell egyeznie a ciklusváltozó típusának az induló és végérték típusával. Első esetben az induló érték nagyobbnak (vagy egyenlőnek), a másodikban kisebbnek (vagy egyenlőnek) kell lennie mint a végértéknek, ha azt akarjuk, hogy a ciklusmag legalább egyszer végrehajtódjon. (Az első esetben a ciklusváltozó léptetése felfelé, a másodikban lefelé történik.) A fent említett okok miatt a "while" és a "for" ciklus ún. elöltesztelős, a "repeat until" pedig hátultesztős ciklus.

A "procedure"-k

vagy másnéven eljárások általános felépítése (3. ábra) a program felépítéséhez hasonló. Ezért a "procedure"-ket (és a "function"-öket) alprogramoknak is szokták nevezni; változóit pedig lokális változóknak. Ezekről a változókról tudni kell, hogy addig "élnek", amíg a vezérlés az adott alprogramon van, azaz mihelyt a program futása a függvényt (vagy eljárást) lezáró "end;"-hez ér tartalmuk elvész, és helyük a memóriában felszabadul. A legtöbbször azonban a meghívó blokknak (ami lehet a főprogram, de alprogram is) szüksége van az adott müveletsor közben létrejött értékre, vagy értékekre. Ilyenkor a paraméterlistában "megjelöljük" ezeket a változókat egy "var" kulcsszóval. A "var" kulcsszó elmarad, ha a változó csak input jellegű. A függvény meghívásánál ügyelnünk kell arra, hogy míg az utóbbi esetben az adott változó helyére akár egy konkrét értéket, akár egy másik változót írunk, addig az előbbi esetben csak változót írhatunk be. (Amennyiben nem adunk és veszünk át értéket a "procedure"-től, a paraméterlista és az azt közrefogó zárójel elmarad.) Az alprogramok használatának hármas haszna van :

- az utasítássort elég a forráslistában egyszer deklarálni, s később csak hivatkozni kell rá,
- átláthatóbbá teszi a programlistát,
- gyorsabb lesz a program.

Mit láthatunk ezek közül az I. listán?

Először is deklaráltam 4 új tipust: Elsőként egy "Kep"-et ami tulajdonképpen egy vektor, aminek minden egyes mezője megfelel egy "Sor" nevű tömbnek, aminek minden egyes mezője byte tipusú. Huh. Bizony ennél lehetet volna egyszerűbben is. (A kép tipus egy háromdimenziós tömb, aminek minden eleme byte tipusú):

```
Kep = array[ 1..80, 1..25, 1..2]of Byte;
```

(A LathatoKep[X,Y,1] byte-ja tartalmazza az (X,Y) pozición található karakter ASCII kódját, a LathatoKep[X,Y,2] byte-ja, pedig az (X,Y) pozíció színleíró byte-ja.) A második AblakParam típus egy record, aminek egyes mezői egy ablak jellemző paramétereit tartalmazzák:

- BFO : a Bal-Felső-Oszlop tartalmazza az ablak bal felső sarkának oszlop (X) koordinátáját,
- BFS : a Bal-Felső-Sor megadja az ablak balfelső sarkának sor (Y) koordinátáját,
- Mag : az ablak "belső" magasságát tárolja, a
- Szel pedig a "belső" szélességét,
- Cim : Az ablak keretének felső részében, középen található szöveg. (az ablak neve.)

A NyGombParam értelmezése a fentiek alapján, már nem okoz gondot. Tekintsünk a főprogramra, ami két utasítást tartalmaz. Az első (Hatter) jó példa egy paraméter nélküli procedure meghívására. (Itt két egymásba ágyazott "for" ciklust találhatunk, aminek segítségével feltöltjük a képernyőt a háttérmintát adó karakterrel.) Ezután következik az UzenoAblak meghívása. (Megjegyzés: Az I. listában közölt függvények természetesen már a file-keresőhöz készültek, de önmagukban is használhatók. Erre jó példa ez a kis program ami egy ablakot nyit ki a képernyőn, és közli velünk, hogy "...ez bizony hibátlan!".) Nézzük röviden, mi történik akkor amikor a program futása ideér: az eljárás átveszi a főprogramtól az UzenoAblCim változóban az ablak címét, valamint az üzenetet az Uzenet-ben. Miután lefoglalta a helyet a lokális változóknak a memóriában, belép a procedure-törzsbe. Elsőként elmenti az éppen aktuális képernyőképet, majd meghatározza az ablak paramétereit:

- Cím : a címkét három részből fűzi össze: egy "["-ből, az az UzenoAblak változóból, valamint egy "]"-ből.
- Mag : a magassága 3 karakter lesz, mert így egy sorba ki fog férni az üzenet, s még alul és felül is marad egy-egy üres sor.
- BFS : a 10-ik sorban lesz a keret felső széle.

Most elérkeztünk ahhoz a ponthoz, hogy el kell döntenünk, hogy melyik string hosszabb: a címé, vagy az üzeneté (a hosszt a Length függvény adja vissza. Részletesebben a következő részben fogok rá kitérni), hiszen milyen kényelmetlen lenne, ha pl.: az üzenet túllógna az ablak keretein. A döntés függvényében kiszámoltatjuk a BFO és Szel mező értékét. Ezek után felrajzoltatjuk magát az ablakot (Ablak(UzenoParam)), és elhelyezzük benne az üzenetet. Miután megjelenik az "Enter-gomb" sematikus rajza, nincs más hátra, minthogy a programunkat várakozásra kényszerítjük (ReadLn) mindaddig, amíg egy "Enter"-rel zöld jelzést nem adunk a további futásnak. Ezután már csak annyi történik, hogy újra megjelenik a nyitó képernyő. Illetve történik még egy fontos dolog: a vezérlés visszakerül a főprogramhoz, ami még kiírja, hogy: "Folytatás januárban!"

Burai Zsolt

VAX VMS

Az alábbi parancs file-ok aVAX VMS rendszer alatt futtathatók, ill. az EVE editor használatát segítik. Szükséges magyarázatok a fileokban megtalálhatók.

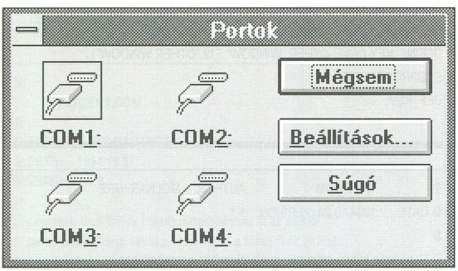
```
$!
$!     FRAGMENT.COM
$!
$! AUTHOR:  MOLNAR IMRE
$! DATE:    1994.10.27
$! VERSION: 1.1
$!
$! Lemezen lévő fájlok töredezettségét jelzi ki az alábbi
$! parancsfájl. Kiírja nevet a méretet és a töredékek számát.
$! Az adatokat elhelyezi a felhasználó default könyvtárában
$! a FRAGMENT_FILES.DAT állományban.
$!
$      SAVE_DEF = F$ENVIRONMENT("DEFAULT")
$      SAVE_VER = F$VERIFY(0)
$      Set control=Y
$      DEV_STR = ""
$      On WARNING then Goto ENDIT
$      SAY = "Write SYS$OUTPUT"
$      If P1 .nes. "HELP" .and. P1 .nes. "help" then Goto INIT
$      SAY "A command inditasa: @FRAGMENT.COM [P1]
$      SAY "Parameter (P1):    [] Path |] (default: [])"
$      SAY "                   [] HELP |]"
$      Goto PROC_END
$ INIT:
$      If P1 .eqs. "" then Goto START
$      SET DEF 'P1'
$      P1 = F$ENVIRONMENT("DEFAULT")
$      DEV_STR = F$ELEMENT(0,":",P1) + ":"
$      P1 = F$ELEMENT(1,"[",F$ELEMENT(0,"]",F$ELEMENT(1,":",P1)))
$      SET_DEF = DEV_STR + "[" + P1 + "]"
$ START:
$      SAY F$ENVIRONMENT("DEFAULT")
$      SAY ""
$      DIR_STR = "[" + P1 + "]"
$      DIR_STRI = ""
$      DIR_EXT = "*.*;*"
$      LIS_NAME = SAVE_DEF + "FRAGMENT_FILES.LIS"     ! Könyvtár
lista
$      DAT_NAME = SAVE_DEF + "FRAGMENT_FILES.DAT"     !
Töredezettség
$      WRK_NAME = SAVE_DEF + "FRAGMENT_WORKS.DAT"     ! Fájl-
struktúra
$      SEARCH1 = "Map area"
$      SEARCH2 = "Retrieval pointers"
$      SEARCH3 = "Count:"
$      SEARCH4 = "LBN:"
$      On WARNING then Goto ENDIT
$      Directory 'DIR_STR'/Nohead/Output='LIS_NAME'
$      Open TREE_WORK 'LIS_NAME'
$      Open/Write FRAG_DATA 'DAT_NAME'
$ READ_LOOP:
$      Read/End_of_file=ENDIT TREE_WORK FIL_NAM
$      If FIL_NAM .eqs. ""
$           then    Read/End_of_file=ENDIT TREE_WORK FIL_NAM
$                Goto READ_LOOP
$      Endif
$      Spawn/Nolog
Dump/File/Header/Block=Count:0/Output='WRK_NAME' 'FIL_NAM'
$      FL = 1
$      SAY FIL_NAM
$      Open FRAG_WORK 'WRK_NAME'
$      Close FRAG_WORK
$      Open FRAG_WORK 'WRK_NAME'
$ WREAD_LOOP:
$      Read/End_of_file=ENDITW FRAG_WORK FRG_REC
$      W = F$LENGTH(FRG_REC)
$      If FL .eq. 1
$           then    If F$LOCATE(SEARCH1,FRG_REC) .eq. W
$                        then Goto WREAD_LOOP
$                        else    FL = FL+1      ! MAP AREA
$                             NM = 0
$                             NB = 0
$                Endif
$      Endif
$      If FL .eq. 2
$           then    If F$LOCATE(SEARCH2,FRG_REC) .eq. W
$                        then    Goto WREAD_LOOP
$                        else    FL = FL+1      ! RETRIEVAL POINTERS
$                Endif
$      Endif
$      If FL .eq. 3 .and. F$LOCATE(SEARCH3,FRG_REC) .eq. W then Goto
WREAD_LOOP
$                                        ! COUNT
$      W = F$ELEMENT(1,":",FRG_REC)
$      W = F$ELEMENT(0,"L",W)
$      Z = F$EDIT(W,"COLLAPSE")
$      N = F$INTEGER(Z)
$      NM = NM+1
$      NB = NB+Z
$      Goto WREAD_LOOP
$ ENDITW:
$      If NM .gt. 1
$           then    SAY "FRAGMENTALT"
$                SAY FIL_NAM," - "," MERET:",NB," DARAB:",NM
$                Write FRAG_DATA FIL_NAM," - "," MERET:",NB,"
DARAB:",NM
$      Endif
$      Close FRAG_WORK
$      Goto READ_LOOP
$ ENDIT:
$      Set noon
$      Close FRAG_DATA
$      Close TREE_WORK
$      Define/user_mode SYS$ERROR  NL:
$      Define/user_mode SYS$OUTPUT NL:
$      Delete 'WRK_NAME';*
$      Define/user_mode SYS$ERROR  NL:
$      Define/user_mode SYS$OUTPUT NL:
$      Delete 'LIS_NAME';*
$ PROC_END:
$      Set nocontrol=Y
$      SAVE_VER = F$VERIFY(SAVE_VER)
$      Set default 'SAVE_DEF'
$      Delete/Symbol/All
$      Exit
```

```
! Az EVE editor (EDIT/TPU) használatában nyújt segítséget
! az alábbi néhány definíció amennyiben két fájlt kívánunk
! egyszerre szerkeszteni.
!
! ONE_WINDOW          egy ablak esetében, fájlt válthatunk
!                     két ablaknál kikapcsoljuk a nem aktuálisat
! TWO_WINDOW          egy ablak esetében, bekapcsolunk egy másodikat
!                     két ablaknál, fájlt válthatunk az aktuálisban
! TWO_WINDOW_NEXT_PAGE      lapdobás előre mindkét ablakban
! TWO_WINDOW_PREV_PAGE      lapdobás vissza mindkét ablakban
!
! A DEF_KEY procedúrában található az eljárások és a hozzájuk
! tartozó billentyűk összerendelése. Az E5,SHIFT_KEY = Gold-Nextpage.
! Az F19 definició teszi lehetővé a két ablak közti váltást.
! Ez olyankor is használható mikor wildcard-ot adtunk meg a
! szerkesztendő állomány nevében. Ebben az esetben az EVE felkínál
! egy külön ablakot ahol az összes - wildcard-nak megfelelő -
! fájlnevet felajánlja. Ekkor az F19 lenyomásával ez lesz
! aktuális, ezután a kurzort a megfelelő névre pozicionálva és
! a Select gombot majd az F19-et újra lenyomva az általunk
! kívántat olvassa be a program.
PROCEDURE ONE_WINDOW
        IF EVE$X_NUMBER_OF_WINDOWS = 1
                THEN EVE_NEXT_BUFFER;
                ELSE EVE_ONE_WINDOW;
        ENDIF;
ENDPROCEDURE;
PROCEDURE TWO_WINDOW
        IF EVE$X_NUMBER_OF_WINDOWS = 2
                THEN EVE_NEXT_BUFFER;
                ELSE EVE_TWO_WINDOWS;
        ENDIF;
ENDPROCEDURE;
PROCEDURE TWO_WINDOW_NEXT_PAGE
        IF EVE$X_NUMBER_OF_WINDOWS = 2
                THEN    EVE_NEXT_SCREEN;
                        EVE_OTHER_WINDOW;
                        EVE_NEXT_SCREEN;
                        EVE_OTHER_WINDOW;
        ENDIF;
ENDPROCEDURE;
PROCEDURE TWO_WINDOW_PREV_PAGE
        IF EVE$X_NUMBER_OF_WINDOWS = 2
                THEN    EVE_PREVIOUS_SCREEN;
                        EVE_OTHER_WINDOW;
                        EVE_PREVIOUS_SCREEN;
                        EVE_OTHER_WINDOW;
        ENDIF;
ENDPROCEDURE;
PROCEDURE DEF_KEY
DEFINE_KEY
("TWO_WINDOW_NEXT_PAGE",KEY_NAME(E6,SHIFT_KEY),"NEXT
SCREEN");
DEFINE_KEY
("TWO_WINDOW_PREV_PAGE",KEY_NAME(E5,SHIFT_KEY),"PREV
SCREEN");
DEFINE_KEY ("ONE_WINDOW",F18,"ONE WINDOW");
DEFINE_KEY ("TWO_WINDOW",F20,"TWO WINDOWS");
DEFINE_KEY ("EVE_OTHER_WINDOW",F19,"OTHER WINDOW");
ENDPROCEDURE;
DEF_KEY;
```

```
$!          TAPE.COM            AUTHOR:   MOLNAR IMRE
$! DATE:    1994.10.20. VERSION:  1.1
$!
$! Nap végén induló automatikus mentésekhez a meghajtóban
$! lévő szalagnevet vizsgálja. Meghatározza, hogy a hét
$! melyik napja van. Csak bizonyos nevü szalagokat fogad el !
$! A NAPK változóban ez beállítható.
$!
$           SET NOVERIFY
$           SET NOON
$           SAY="WRITE SYS$OUTPUT"
$           DAYS="Monday/Tuesday/Wednesday/Thursday/"
$           DAYS=DAYS + "Friday/Saturday/Sunday"
$           NAPK="HETFO/KEDD/SZERDA/CSUTORTOK/"
$           NAPK=NAPK + "PENTEK/SZOMBAT/VASARNAP"
$           DAY=F$CVTIME("TODAY",,"WEEKDAY")
$           ANSW=" "
$           EXST=3
$           INX=0
$ LOOP:                                     ! napnév összehasonlítás
$           NAP=F$ELEMENT(INX,"/",DAYS)
$           IF NAP .NES. DAY
$                   THEN    INX=INX+1
$                           GOTO LOOP
$           ENDIF
$           NAP=F$ELEMENT(INX,"/",NAPK)
$           INX=INX+1
$ CYCL:                                     ! szalagnév összehasonlítás
$           LAB=F$GETDVI("MUA0:","VOLNAM")
$           SAY LAB+" CIMKÉJÜ SZALAG VOLT A MEGHAJTÓBAN"
$           DISM MUA:
$           ASK="Helyezze be a napi mentés kazettáját ("
$           ASK=ASK + NAP + ") és RETURN, KILÉPÉS=K: "
$           READ/PROMPT="'ASK'" SYS$COMMAND ANSW
$           IF ANSW .EQS. "K" THEN EXIT 3
$           MOUNT/NOASSIST/OVERRIDE=(ID,OWN) MUA0:
$           LAB=F$GETDVI("MUA0:","VOLNAM")
$           IF LAB .NES. F$EXTRACT(0,6,UNAP) THEN GOTO CYCL
$           DISM MUA:/NOUNLOAD
$           EXST=1
$           EXIT 'EXST'
```

Az IBM PC portkiosztása

(A különböző gyártmányoknál előfordulhatnak bizonyos eltérések)



000-00F 8237 DMA controller
000 Channel 0 address register
001 Channel 0 word count
002 Channel 1 address register
003 Channel 1 word count
004 Channel 2 address register
005 Channel 2 word count
006 Channel 3 address register
007 Channel 3 word count
008 Status/command register
009 Request register
00A Mask register
00B Mode register
00C Clear MSB/LSB flip flop
00D Master clear temp register
00E Clear mask register
00F Multiple mask register

010-01F 8237 DMA Controller (PS2 model 60 & 80), reserved (AT)

020-02F 8259A Master Programmable Interrupt Controller
020 8259 Command port (see 8259)
021 8259 Interrupt mask register (see 8259)

030-03F 8259A Slave Programmable Interrupt Controller (AT,PS2)

040-05F 8253 or 8254 Programmable Interval Timer (PIT, see 8253)
040 8253 channel 0, counter divisor
041 8253 channel 1, RAM refresh counter
042 8253 channel 2, Cassette and speaker functions
043 8253 mode control (see 8253)
044 8254 PS/2 extended timer
047 8254 Channel 3 control byte

060-067 8255 Programmable Peripheral Interface (PC,XT, PCjr)
060 8255 Port A keyboard input/output buffer (output PCjr)
061 8255 Port B output
062 8255 Port C input
063 8255 Command/Mode control register

060-06F 8042 Keyboard Controller (AT,PS2)
060 8042 Keyboard input/output buffer register
061 8042 system control port (for compatability with 8255)
064 8042 Keyboard command/status register

070 CMOS RAM/RTC, also NMI enable/disable (AT,PS2, see RTC)
071 CMOS RAM data (AT,PS2)

080 Manufacturer checkpoint port
080-090 DMA Page Registers
081 High order 4 bits of DMA channel 2 address
082 High order 4 bits of DMA channel 3 address
083 High order 4 bits of DMA channel 1 address

090-097 POS/Programmable Option Select (PS2)
090 Central arbitration control Port
091 Card selection feedback
092 System control and status register
094 System board enable/setup register
095 Reserved
096 Adapter enable/setup register
097 Reserved

0A0 NMI Mask Register (PC,XT) (write 80h to enable NMI, 00h disable)
0A0-0BF Second 8259 Programmable Interrupt Controller (AT, PS2)
0A0 Second 8259 Command port (see 8259)
0A1 Second 8259 Interrupt mask register (see 8259)

0C0 TI SN76496 Programmable Tone/Noise Generator (PCjr)
0C0-0DF 8237 DMA controller 2 (AT)
0C2 DMA channel 3 selector (see ports 6 & 82)

0E0-0EF Reserved

0F0-0FF Math coprocessor (AT, PS2)
0F0-0F5 PCjr Disk Controller
0F0 Disk Controller
0F2 Disk Controller control port
0F4 Disk Controller status register
0F5 Disk Controller data port

0F8-0FF Reserved for future microprocessor extensions

100-10F POS Programmable Option Select (PS2)
100 POS Register 0, Adapter ID byte (LSB)
101 POS Register 1, Adapter ID byte (MSB)
102 POS Register 2, Option select data byte 1
Bit 0 is card enable (CDEN)
103 POS Register 3, Option select data byte 2
104 POS Register 4, Option select data byte 3
105 POS Register 5, Option select data byte 4
Bit 7 is (-CHCK)
Bit 6 is reserved
106 POS Register 6, subaddress extension (LSB)
107 POS Register 7, subaddress extension (MSB)

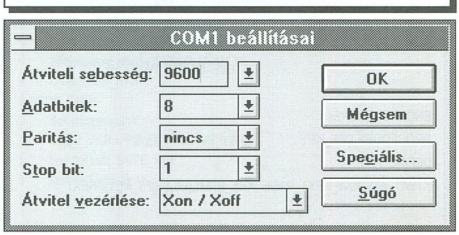
110-1EF System I/O channel

170-17F Fixed disk 1 (AT)
170 disk 1 data
171 disk 1 error
172 disk 1 sector count
173 disk 1 sector number
174 disk 1 cylinder low
175 disk 1 cylinder high
176 disk 1 drive/head
177 disk 1 status

1F0-1FF Fixed disk 0 (AT)
1F0 disk 0 data
1F1 disk 0 error
1F2 disk 0 sector count
1F3 disk 0 sector number
1F4 disk 0 cylinder low
1F5 disk 0 cylinder high
1F6 disk 0 drive/head
1F7 disk 0 status

200-20F Game Adapter (see GAME PORT or JOYSTICK)

210-217 Expansion Card Ports (XT)
210 Write: latch expansion bus data
read: verify expansion bus data
211 Write: clear wait,test latch



Read: MSB of data address
212 Read: LSB of data address
213 Write: 0=enable, 1=/disable expansion unit
214-215 Receiver Card Ports
214 write: latch data, read: data
215 read: MSB of address, next read: LSB of address

21F Reserved

220-26F Reserved for I/O channel

270-27F Third parallel port (see PARALLEL PORT)
278 data port
279 status port
27A control port

280-2AF Reserved for I/O channel

2A2-2A3 MSM58321RS clock

2B0-2DF Alternate EGA, or 3270 PC video (XT, AT)

2E0 Alternate EGA/VGA
2E1 GPIB Adapter (AT)

2E2-2E3 Data acquisition adapter (AT)

2E8-2EF COM4 non PS2 UART (Reserved by IBM) (see UART)

2F0-2F7 Reserved

2F8-2FF COM2 Second Asynchronous Adapter (see UART)
Primary Asynchronous Adapter for PCjr

300-31F Prototype Experimentation Card (except PCjr)
Periscope hardware debugger
320-32F Hard Disk Controller (XT)
320 Read from/Write to controller
321 Read: Controller Status, Write: controller reset
322 Write: generate controller select pulse
323 Write: Pattern to DMA and interrupt mask register
(see ports 0F,21,C2)
324 disk attention/status

330-33F Reserved for XT/370

340-35F Reserved for I/O channel

360-36F PC Network

370-377 Floppy disk controller (except PCjr)
372 Diskette digital output
374 Diskette controller status
375 Diskette controller data
376 Diskette controller data
377 Diskette digital input

378-37F Second Parallel Printer (see PARALLEL PORT)
First Parallel Printer (see PARALLEL PORT)
378 data port
379 status port
37A control port

380-38F Secondary Binary Synchronous Data Link Control (SDLC) adapter
380 On board 8255 port A, internal/external sense
381 On board 8255 port B, external modem interface
382 On board 8255 port C, internal control and gating
383 On board 8255 mode register
384 On board 8253 channel square wave generator
385 On board 8253 channel 1 inactivity time-out
386 On board 8253 channel 2 inactivity time-out
387 On board 8253 mode register
388 On board 8273 read: status; Write: Command
389 On board 8273 write: parameter; read: response
38A On board 8273 transmit interrupt status
38B On board 8273 receiver interrupt status
38C On board 8273 data

390-39F Cluster Adapter

3A0-3AF Primary Binary Synchronous Data Link Control (SDLC) adapter
3A0 On board 8255 port A, internal/external sense
3A1 On board 8255 port B, external modem interface
3A2 On board 8255 port C, internal control and gating
3A3 On board 8255 mode register
3A4 On board 8253 counter 0 unused
3A5 On board 8253 counter 1 inactivity time-outs
3A6 On board 8253 counter 2 inactivity time-outs
3A7 On board 8253 mode register
3A8 On board 8251 data
3A9 On board 8251 command/mode/status register

3B0-3BF Monochrome Display Adapter (write only, see 6845)
3B0 port address decodes to 3B4
3B1 port address decodes to 3B5
3B2 port address decodes to 3B4
3B3 port address decodes to 3B5
3B4 6845 index register, selects which register (0-11h)
is to be accessed through port 3B5
3B5 6845 data register (0-11h) selected by port 3B4,
registers 0C-0F may be read. If a read occurs without
the adapter installed, FFh is returned. (see 6845)
3B6 port address decodes to 3B4
3B7 port address decodes to 3B5
3B8 6845 Mode control register
3B9 reserved for color select register on color adapter
3BA status register (read only)
3BB reserved for light pen strobe reset

3BC-3BF Primary Parallel Printer Adapter (see PARALLEL PORT)
3BC parallel 1, data port
3BD parallel 1, status port
3BE parallel 1, control port

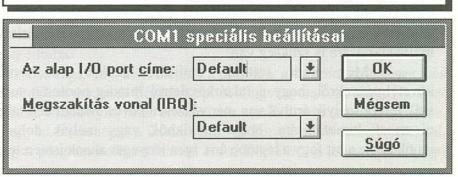
3C0-3CF EGA/VGA
3C0 VGA attribute and sequencer register
3C1 Other video attributes
3C2 EGA, VGA, CGA input status 0
3C3 Video subsystem enable
3C4 CGA, EGA, VGA sequencer index
3C5 CGA, EGA, VGA sequencer
3C6 VGA video DAC PEL mask
3C7 VGA video DAC state
3C8 VGA video DAC PEL address
3C9 VGA video DAC
3CA VGA graphics 2 position
3CC VGA graphics 1 position
3CD VGA feature control
3CE VGA graphics index
3CF Other VGA graphics

3D0-3DF Color Graphics Monitor Adapter (ports 3D0-3DB are
write only, see 6845)
3D0 port address decodes to 3D4
3D1 port address decodes to 3D5
3D2 port address decodes to 3D4
3D3 port address decodes to 3D5
3D4 6845 index register, selects which register (0-11h)
is to be accessed through port 3D5
3D5 6845 data register (0-11h) selected by port 3D4,
registers 0C-0F may be read. If a read occurs without
the adapter installed, FFh is returned. (see 6845)
3D6 port address decodes to 3D4
3D7 port address decodes to 3D5
3D8 6845 Mode control register (CGA, EGA, VGA, except PCjr)
3D9 color select palette register (CGA, EGA, VGA, see 6845)
3DA status register (read only, see 6845, PCjr VGA access)
3DB Clear light pen latch (any write)
3DC Preset Light pen latch
3DF CRT/CPU page register (PCjr only)

3E8-3EF COM3 non PS2 UART (Reserved by IBM) (see UART)

3F0-3F7 Floppy disk controller (except PCjr)
3F0 Diskette controller status A
3F1 Diskette controller status B
3F2 controller control port
3F4 controller status register
3F5 data register (write 1-9 byte command, see INT 13)
3F6 Diskette controller data
3F7 Diskette digital input

3F8-3FF COM1 Primary Asynchronous Adapter (see UART)



Számítógép a testkultúra szalonban

Lapunkban rendszeresen megszólaltatunk gyakorló profikat. Ebben a számban egy teskultúra szalon ügyvitelét, irányítását felügyelő alkalmazásról tudósítunk.

Egy testkultúra szalon sok egységet foglalhat magában. Megtalálható benne például a zenés kondicionáló torna, a body building - testépítő - terem, a szauna, a szolárium, egy olyan bár vagy presszó, amely a sportolóknak ételt, italt, gyümölcsöket kínál, de sok esetben ez a szalon sportruházatot, kiegészítő kellékeket is árusít. Az ilyen összetett rendszer a szervező számára igen sok problémát, de sok szép és újszerű feladatot rejt magában. Elég, ha csak abból indulunk ki, hogy az intézménynek tulajdonosai vannak, akik önmaguk is dolgoznak az üzletben, és sokkal több felelősséggel - így sokkal több joggal - rendelkeznek, mint az alkalmazottak. Szélsőséges esetben az alkalmazottnak az az érdeke, hogy minél kevesebb munkával, minél rövidebb idő alatt a legtöbb pénzt keresse és ingyenszolgáltatást kapjon. A tulajdonos viszont arra törekszik, hogy mindenki ledolgozza a munkaidőt és meg tudja állapítani, hogy a dolgozók az egyes tevékenységekből mennyi bevételt szereztek az üzletnek.

Így már meg is találtuk a problémák első szintjét. A következő megoldandó feladat a vendégek oldaláról jelentkezik. Egy ilyen intézmény szolgáltatásait igénybe lehet venni bérlettel vagy napijeggyel is. Sajnos, a bérleteket el lehet hagyni, a bérletek lejárnak és hamisítani is lehet ezeket. Ezenkívül nagyon sokfajta bérletet kell és célszerű használni. Például vannak havi bérletek, vannak adott alkalomra (10, 15, 20 alkalomra) szóló, de csak egy bizonyos időn belül felhasználható bérletek is. Célszerű a vendégeket különféle kedvezményes - diák, katona, stb. - bérletekkel csalogatni. Ez a sokfajta lehetőség már igen jelentős nyilvántartási feladatot tartalmaz. Egy másik ide tartozó probléma a vendégek edzésközbeni fogyasztása a presszóban. Itt a gond abban rejlik, hogy aki edzőruhában van, annál nincs pénz, tehát majd az edzés befejezésével fog csak fizetni. Így akár az egyik, akár a másik fél részéről adódhatnak problémák: elfelejtés, vita hogy ki mennyit fogyasztott, ki fizessen stb.

Komoly nehézséget jelenthetnek a szoláriummal kapcsolatos nyilvántartási, karbantartási és üzemeltetési rendszerek. Például a szolárium megfelelő és célszerű üzemeltetéséhez speciális fénycsövek szükségesek. Ezek élettartama véges és meg van adva hozzájuk egy célszerű és észszerű üzemóra, amit be kell tartani. (Ezen túlmenően egy gépben általában két fajta - így két eltérő üzemórájú - cső van.) Ha több szolárium-berendezéssel dolgoznak, akkor szinte törvényszerű, hogy azok nem egyidőben üzemelnek, így ahány gép van, annyiszor 2 üzemórát kell adminisztrálni. Ha vendég érkezik, kell hagyni neki egy kis időt amíg levetkőzik, majd a gépet be kell kapcsolni és az általa kért idő eltelte után kikapcsolni. (A kikapcsolást nem bízhatjuk a vendégre, mert túlzott igénybevétel esetén esetleg "bőrrákot" kapna.) Figyelembe kell venni azt is, hogy a csöveknek kikapcsolás után szükségük van némi hűlési időre, így nem célszerű rögtön ráindítani. Természetesen a szolárium-bérletek nyilvántartására is szükség van.

Még nem tettünk említést azokról a fontos követelményekről, hogy a raktárkészletről mindig pontosan tudni kell, mikor melyik árúból van már vészesen kevés. Módot kell találni annak kimutatására, hogy melyikből, vagy melyik dolgozó munkaideje alatt fogy a legtöbb áru. Igen lényeges annak ismerete is, hogy a bevételek és a kiadások milyen arányban állnak egymással.

A vázolt feladatok végrehajtása, az egyes tevékenységek és folyamatok figyelemmel kísérése eléggé összetett nyilvántartási és ellenőrzési rendszer kialakítását teszi szükségessé. Ennek manuális módszerekkel történő ellátása nagy adminisztrációt és sok munkaórát követelne, nem is beszélve a hibalehetőségekről. Tehát itt állunk egy remek, összetett, számítógéppel prímán megoldható feladat előtt.

A rendszer sémája a következő:

A számítógépes nyilvántartási rendszer kiépítéséhez mindenekelőtt szükségünk van a dolgozók nyilvántartására, aminek a személyi adatokon kívül tartalmaznia kell egy belépési jelszót - password-ot -, amiből következően különböző jogokat kap a belépett személy. Így el lehet különíteni a dolgozókat és a tulajdonost. Ebbe a nyilvántartásba be kell vezetni a belépési és a kilépési időt, hogy a megfelelő kapcsolatokat és eredményeket kigyűjthessük. Szükséges egy raktárnyilvántartás, amire ráépül a presszó- és a bérletnyilvántartás. A szolárium-vezérlés és nyilvántartás mellett a szolárium-bérletek nyilvántartása egyedi, hiszen ezen a felhasznált perceket kell vezetni.

A PRESSZÓ vezérléséhez egy önálló gépre van szükség, hiszen a presszó állandó forgalma miatt a gyors kiszolgálás érdekében ezt a gépet nem lehet megosztani más feladatok megoldására.

A SZOLÁRIUM vezérlés és nyilvántartás is teljesen önálló gépet foglal el, mert több szolárium gép esetén - esetünkben 4 - a számítógép állandóan a vezérléssel van elfoglalva. A szoláriumbérletek egyszerű, de egyedi nyilvántartását a számítógép megszakításrendszerére támaszkodva a vezérléssel kvázi párhuzamosan meg lehet valósítani.

Kénytelenek vagyunk tehát a többi BÉRLET nyilvántartására egy harmadik gépet felhasználni. Mivel itt a forgalom nem állandó, így a gép átkapcsolható közvetlen raktári eladásra, ahol a sportruházat, sporteszközöket lehet a raktárból kiválasztani.

Elgondolásunk konkrét gépekkel történő megvalósítása esetén az alábbiakból kitűnik, hogy szükség van egy negyedik gépre is.

A 3 kiszolgáló egység egy-egy 386 SX IBM kompatibilis számítógép 2 Mb RAM-mal, alap kiépítésben, de winchester nélkül. A gépeket LANTASTIC hálózatba kötjük, mert használata egyszerű, az erőforrásokat mindenki szabadon használhatja, a célnak tökéletesen megfelel. A legfontosabb, hogy erre a kiépítésre, ezzel a teljesítmény-igénnyel ennek az ára a legkedvezőbb.

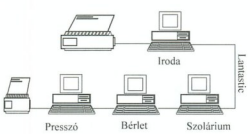
Felvetődhet, hogy ha már kedvező árra törekszünk, miért nem egy 286-os számítógépet használunk? A válasz a DOS 640 Kbyte-os memória korlátjából adódik. A DOS és a LANTASTIC memória

igénye miatt a későbbiekben bemutatott rendszerrel fejlesztett programnak nem lenne meg a szükséges - kb. 550 Kbyte - memóriája. Viszont a 2 Mbyte-os 386-tal a DOS egy részét és a LANTASTIC-ot teljesen fel lehet tölteni a felső memóriába, így az alapmemória kb. 620 Kbyte-ra szabadul fel.

A szükséges háttértár biztosítására vezetjük be a negyedik gépet - IRODA -, mint szervert. Ezzel az irodában lehetőség van a bármikori lekérdezésre. Miközben a többi gépen futnak a programok, így ellenőrzni lehet azokon a munkát, le lehet kérdezni a raktárkészletet és különféle listák előállítására is itt lesz lehetőség.

Ez a gép célszerűen egy 386 DX 40-es alaplapon 4 Mb RAM-mal, legalább 120 MB Winchesterrel, a programok és az adatok tárolására alkalmas. A kiszolgáló gépek egy boot lemezzel indulnak és utána felveszik a kapcsolatot a szerverrel és -szükség szerint- egymással is.

Eddig még nem szóltunk a nyugtaadási kötelezettségről. A nyugtaadást a rendszerhez úgy lehet beépíteni, hogy az egyik munkaállomáshoz kapcsolunk egy blokknyomtatót, amin a törvénynek megfelelő nyugtát ki lehet nyomtatni. Ez a gép is - a LANTASTIC hálózat specialitása miatt - ezentúl szerver lesz, ezáltal a több kiszolgáló egység is látni fogja a hozzá kapcsolt blokknyomtatót. Így mindenhonnan ki lehet nyomtatni a nyugtát (bérletről, egy turmixról, edzőnadrágról stb.). Az IRODAI szerverhez



egy nyomtatót kell kapcsolni , mert az ÁFA-s számlát csak azon lehet kinyomtatni. Mivel a számla önindigós, csak mátrix nyomtatóról lehet szó.

A rendszer végleges hardware felépítése:

A gépek adataihoz csak a bejelentkezés után lehet hozzáférni. Innen meg lehet tudni, hogy ki mennyit dolgozott, mit adott el és mennyi bevételt ért el.

A fizetési kötelezettségeket úgy lehet megoldani, hogy ha valaki rendel valamit a presszóból, akkor az azonosítója alapján - név, kulcsszám - azt a számlájához írják. Amikor elmegy, leadja a kulcsot a BÉRLET gépnél, ahol a képernyőn mindig látható az a táblázat, amely tartalmazza azoknak a kulcsoknak a számát, amelyeken tartozás van. Így kiszűrhető, ha valaki még nem fizetett. Ezt úgy lehet megvalósítani, hogy a BÉRLET gépen a billentyűzet megszakításon ülve, 1 másodpercenként a PRESSZÓ állományból kigyűjtött Tartozik file-ból frissítjük a képernyőn azoknak a kulcsoknak a sorszámát, amelyeken tartozás van.

A SZOLÁRIUM vezérlést és nyilvántartást nézzük meg részletesebben.

A szoláriumgép és a számítógép kapcsolatának alapvető fázisai:

- a szolárium beindítása
- a megadott idő után a szoláriumgép lekapcsolása
- a hűtési idő betartása.

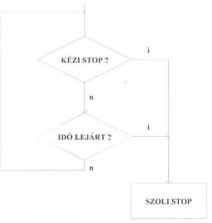
Nézzünk egy példát, hogy miként zajlik a szoláriumgép használata: A vendég megérkezik, vagy napi jegyet vesz, vagy a bérletét használja. Megmondja, hogy hányszor 5 percig kívánja a szoláriumot használni, és turbo, vagy normál szoláriumot szeretne. Ezután bemegy a szoláriumhoz, levetkőzik és befekszik a gépbe.

Az első kérdés: mikor induljon a szolárium? Amikor a vendég bemondja, hogy hány percig kíván szoláriumozni, ezt az időtartamot a számítógépen be kell állítani és egy késleltetési idő után elindítani. Ezután a szolárium akkor kapcsol be, ha letelt a késleltetés, vagy ha a vendég már befeküdt és megnyomta az indító gombot.

A következő megoldandó feladat: mikor álljon le a szoláriumgép? A szolárium addig üzemel, amíg megkapja a tápfeszültséget. Tehát a vendég által kért ideig a feszültséget tartani kell, majd az idő lejártával le kell kapcsolni. De hogy mindenre fel legyünk készülve (pl.: a vendég megunta), módot kell találni arra, hogy a számítógépen az idő lejárta előtt is parancsot tudjunk adni a kézi leállításra.

A szoláriumgép leállása után a hűtési idő betartását egy számlálóval biztosítjuk. Amíg az le nem jár, addig a képernyőn látható, hogy hűtés van (de szükség esetén be lehessen indítani újra).

Vezérlőjelek a szolárium és a számítógép között:



Ahhoz, hogy a számítógép esetünkben vezérelni tudjon egy szoláriumot, a számítógépnek egy input és egy output jelre van szüksége. Megoldandó feladat, hogy ezek a berendezések hogyan csatlakozzanak a számítógéphez. Egyik megoldás lehet, hogy közvetlenül a számítógép I/O csatornájába helyezünk egy input-output csatolást megvalósító kártyát. Ennek a megoldásnak az a hátránya, hogy egy külön kártyát kell építeni a gépbe. Ez ugyan megnöveli a költségeket, viszont előnye, hogy a számítógép által nyújtott portokat nem használjuk fel a saját célunkra.

Másik megoldás, hogy a gépben általában már található Centronics szabványú nyomtatócsatlakozót használjuk fel. A kártya párhuzamos kommunikációra képes, 8 bit szélességben, valamint vezérlő- és státuszjeleket tartalmaz. Ezek a kártyák "elvileg" az adatátvitelt a 8 portjukon oda-vissza meg tudják valósítani, de ezt általában nem támogatják, így csak kimenetnek lehet felhasználni. De szerencsére a vezérlő jeleivel kialakíthatunk akár 8 bemenetet is, úgy, hogy az AUTO FEED kimenettel vezérelünk egy multiplexert és az ACK, BUSY, PE, SLCT a nyomtató által küldött vezérlő és státuszjeleket, mint bemenetet használjuk.

Mivel esetünkben 4 szoláriumot vezérlünk, nincs szükség a multiplexeres megoldásra, így csak a párhuzamos portot önmagában használhatjuk, a megoldás a legegyszerűbb és legolcsóbb.

A számítógép port és a szolárium berendezés összekötésére célszerű a szilárd test relét alkalmazni, mert az teljes galvanikus leválasztást biztosít a számítógép és az erősáramú egységek között.

Miután a szoláriumgépek és a számítógép összekapcsolását megoldottuk, térjünk át a vezérlő- és nyilvántartó program felépítésének ismertetésére, kiragadva a SZOLÁRIUM programmodult a teljes programrendszerből.

A program modulárisan épül fel, a gyors javítás és a könnyű továbbfejleszthetőség érdekében. A modulok a rendszerben több helyen felhasználhatók. A modularitás érdekében és a gyors, jól átlátható, könnyen debuggolható programozás érdekében a programokat egy általam jól ismert és régóta használt fejlesztői környezetben írtam. A programozási nyelv a Turbo Pascal, a felhasznált fordító a Borland Pascal 7.0-ás verziója.

A program felépítése, működtetése:

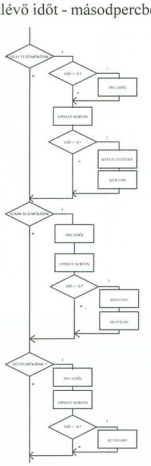
A program neve SZOLI. EXE. Nézzük végig eljárás szinten a forrás file felépítését. A Szoli.pas maga a program, amely az inicializálás és a lezárás megvalósítását és az overlay technika meghívását tartalmazza. Az exe overlay hívásokkal dolgozik, így a program kevesebb memóriát foglal el futás közben. A forrás elején azoknak a unitoknak a felsorolása található, amelyeket ki kívánok tenni az overlay file-ba.

Az első függvény, amit meghív a program, a futáshoz szükséges más unitokban deklarált Boolean típusú függvényeket hív meg. Ha ezek közül valamelyik false értékkel tér vissza - nem sikerült a futása, vagy ki kíván lépni a felhasználó - akkor az init is false értéket ad, így a program befejezi futását. Ha az inicializálás lefutott az alkalmi

jegyek kiadását, a bérletek kezelését végző főprogramba jutunk.
Az ügyfélnyilvántartás alatt a szoláriumok vezérlése folyamatban van, és miközben a bérletekkel bármit is csinálunk, a program - ha szükség van rá - le is kapcsolja a szolárium gépeket. A nyilvántartó alatt a gépek még hátralevő futási ideje másodpercenként frissülve látható a képernyőn.
A vezérlést 3 tömb valósítja meg: Késleltető, Számláló, Hűtő. Mindhárom a következő típusú:

```
type
  SzamlaloTipus    =
    Record
        Mukodik          :       Boolean:
        Perc             :               Byte;
        Ido              :               Time;
        Kezdes           :       Time;
        Dolgozo          :       HivSzamTipus;
    end;
  SzamlaloTombTipus = Array[1..SzoliNr] of SzamlaloTipus;
```

A Mukodik azt tartalmazza, hogy az adott tömb az adott szoláriumnál működik-e vagy sem. A Perc az elindított percek számát, az Ido a még hátralévő időt - másodpercben -, a Kezdes pedig az adott tömb indításának pillanatában a gép órájának állását tartalmazza. A számláló eljárás egy ciklust tartalmaz, ami végigfut a szoláriumok számán 1-től kezdve. A ciklusmag három részből, a 3 számláló vizsgálatából áll. Felépítésük egységes: ha az adott szolárium számlálója működik és az idő még nem 0, akkor csökkenti a hátralévő időt a számlálóban, felfrissíti a képernyőt.



A képernyőn az adott szoláriumhoz egy vonal hossza szimbolikusan mutatja, hogy még mennyi idő van hátra az adott műveletből, és ez percben is leolvasható. Ha a csökkentés által az idő lejárt, a cikus lekapcsolja a megfelelő egységet és bekapcsolja a következőt.
Ha szükség van az idő előtti leállításra : ha csak a késleltetés működött, akkor az áll le. Ha a szolárium, akkor a szolárium leáll és a hűtés bekapcsol.
A rendszer bevezetésének gazdasági előnyei elsősorban az adminisztrációs munka és az ezzel együttjáró hibajavítások megtakarításában nyilvánulnak meg. A számítógépes nyilvántartás szakszerű és pontos, kiszűri a hiányokat, a tévesztéseket, az esetleges csalásokat. A kimutatások mindig gyorsak és szó szerint percre készek. (A nyilvántartás percben van kifejezve.) Mindezeken túl a számítógépes vezérlés megoldja - és talán ez a legfontosabb - a szolárium biztonságos, balesetmentes üzemeltetését (bekapcsolás, kikapcsolás, hűtési idő betartása stb.) is. A következő hasábban a szolárium port írását olvasását, számláló frissítését végző rutinok láthatók.

Balla László villamos mérnök
Telefon: 226-8962

```
procedure UpdateSzamlalo;
 var
  N : Byte;
 begin
  for N := 1 to SzoliNr do
   begin
    with KesleltetoTomb[N] do
     if Mukodik
      then begin
       if Ido <> 0 then Ido := Kezdes-CurrentTime;
       UpdateKesleltetoScreen(N);
       if Ido <= 0
        then begin
         KesleltetoOff(N);
         SzoliOn(N);
        end;
      end;
    with SzamlaloTomb[N] do
     if Mukodik
      then begin
       Ido := Kezdes-CurrentTime;
       UpdateSzamlaloScreen(N);
       if Ido <= 0
        then begin
         SzoliOff(N);
         HutesOn(N);
        end;
      end;
    with HutesTomb[N] do
     if Mukodik
      then begin
       if Ido <> 0 then Ido := Kezdes-CurrentTime;
       UpdateHutesScreen(N);
       if Ido <= 0 then HutesOff(N);
      end;
   end;
 end;

function ReadSzolariumPort(N : Byte) : Boolean;
 begin
  ReadSzolariumPort := not BitAreTurn(Port[SzoliConfigRec.SzoliPort+1],
N+2);
 end;

procedure WriteSzolariumPort(N : Byte; K : Boolean);
 var
  B : Byte;
 begin
  B := Port[SzoliConfigRec.SzoliPort];
  TurnBit(B, N-1, K);
  Port[SzoliConfigRec.SzoliPort] := B;
 end;

procedure UpdatePort;
 var
  N : Byte;
 begin
  for N := 1 to SzoliNr do
   begin
    if KesleltetoTomb[N].Mukodik and ReadSzolariumPort(N)
     then KesleltetoTomb[N].Ido := 0;
    WriteSzolariumPort(N, SzamlaloTomb[N].Mukodik);
   end;
 end;

procedure MyGetKey;
 var
  H, M, S : Word;
  Szazad  : Word;
 begin
  GetTime(H, M, S, Szazad);
  If Szazad = 99
   then begin
    UpdateSzamlalo;
    UpdatePort;
   end;
 end;
```

ASSEMBLER ISKOLA

Azoknak akik szeretnének megtanulni valamilyen programozási nyelvet, segítséget nyújt folyóiratunk. Ennek keretében inkább a gyakolati, mint elméleti oldaláról ismertetjük az assembly nyelvet.

Ez az a nyelv, ahol egy program elkészítéséhez nem elég csupán az utasítások ismerete, hanem egy átfogó hardware-ismeretre is szükség van. Az assembly a számítógép saját nyelvén való programozása.
Egy program megírása tulajdonképpen nem más, mint az adott feladat elemeire való bontása. Ezek az elemek magasabb szintű nyelveknél nagyobbak, az assemblynél a lehető legapróbbak. Ebből is látszik, hogy nem tartozik a könnyen megtanulható nyelvek közé. A programozás során több új fogalommal kell majd megismerkednünk, amik egyben a programozó eszköztárát is képezik. Ezek határozzák meg a számítógépes programozás jellegét.
A programozás eszközei:

Eljárások: olyan rutinok, amire a program során többször is szükség van. Ilyenkor azt elegendő egyszer megírni és a továbbiakban csak hivatkozni rá. Ilyen eset például egy szöveg kiíratása, vagy egy vonal rajzolása, stb.

Változók: nem mások, mint a memóriában tárolt különböző típusú adatok. Egyes feladatok megoldása elképzelhetetlen pusztán regiszterek használatával, a memória mérete pedig bőségesen megengedi, hogy abban adatokat tároljunk. Ezeket ne tévesszük össze a magasabb szintű nyelveknél megismert változókkal, mert nagyon sok különbség van köztük.

Ciklusok: ez egyike a leggyakrabban használt eszközöknek, ugyanis segítségével az egymás után többször végrehajtandó programrészeket elegendő csak egyszer megírni és egy ciklus segítségével többször végrehajtani. Például az a feladat, hogy csippanjon ötöt a gép, akkor nem kell ötször megírni a rutint, hanem elegendő egyszer és utasítani a gépet arra, hogy azt ötször hajtsa végre.

Feltételek vizsgálata, elágazások: igen gyakran előfordul, hogy el kell döntenünk, hogy egy eredménnyel mit kezdjen a gép, vagy adott helyzetben választani kell a különböző lehetőségek közül. Erre a célra szolgálnak a feltételek és elágazások.

Alapértelmezés szerint a számokat decimálisan írjuk. Ha ettől eltérően ábrázoljuk azokat, akkor jelölni kell, hogy az adott szám melyik számrendszer szerint értelmezendő, így a bináris számok után egy 'b', hexadecimális számok után pedig egy 'h' betűt. A 16 bites regisztereket használjuk címzésre, egy 16 bites számmal megcímezhető legnagyobb memóriacím a 65535, azaz 64 KByte. Nos ennél még a leggyengébb XT-ben is több van. A megoldás az, hogy a gépben lévő memóriát felszeleteljük és laponként kezeljük. Így jött létre a szegmentált címzés, amihez szükség van egy szegmens és egy index címre. Ennek lényege, hogy a szegmenscím segítségével kiválasztunk egy 64K-s lapot a memóriából, és az indexcím segítségével határozzuk meg a pontos címet. A teljes memóriacím 20 bit hosszú. Ez úgy alakul ki, hogy a szegmenscímhez képest négy bittel el van tolva az indexcím. Ebből adódóan két szomszédos szegmens egymástól 16 Byte (egy paragrafus) távolságra van. A megoldás hátránya, hogy csak un. paragrafushatáron kezdődhet egy szegmens. A tanuláshoz szükségünk lesz egy assembler-fordítóra (a Turbo Assemblert ajánljuk), valamint egy egyszerű szövegszerkesztőre. Először minden magyarázat nélkül közlünk egy programlistát az 1. listán és a fordításhoz szükséges parancsfilet a 2. listán.

1. LISTA.

```
CODE    SEGMENT PUBLIC 'CODE'
        ASSUME CS:CODE , DS:CODE
        ORG    100h              ; COM file-t készítünk.
START:
        JMP    INDUL
        SZOVEG DB     ' Helló világ ! $'
INDUL:
        PUSH   CS
        POP    DS
        MOV    AH,09h            ; DOS funkció beállítása
                                 ; 09h = String kiíratása.
        MOV    DX,offset SZOVEG; Kiírandó string kezdete
        INT    21h               ; DOS megszakítás
                                 ; hívása
        MOV    AX,4C00h          ; 4Ch = Kilépés
        INT    21h               ; DOS megszakítás
                                 ; hívása
CODE    ENDS
END     START
```

2. LISTA

Turbo Assembler-hez COM-ra fordító BAT-file.

```
TASM %1
TLINK -T %1
DEL %1.OBJ
DEL %1.MAP
```

Számkiíró

A Pascal nyelvű függvény a NUMERIC paraméterben megadott numerikus formájú számot alakítja át - a magyar helyesírási szabályoknak megfelelően - betűvel leírt számmá.

```
unit Num2Str;

interface

type
  NumericTipus = Longint;

function NumericToString(Numeric : NumericTipus) : String;

{*****************************************************************************}
{* A függvény  a  NUMERIC  - egész típusú - paraméterben megadott numerikus  *}
{* formájú számot alakítja át a magyar helyesírási szabályoknak megfelelően  *}
{* betűvel leírt számmá.                                                     *}
{* A Longint típusú változó a -2147483648..2147483647 intervallumban lehet.  *}
{* Készítette : Balla László                                                 *}
{*****************************************************************************}

implementation

function NumericToString(Numeric : NumericTipus) : String;
 const
   Ezer      = 'ezer';
   Millio    = 'millió';
   Milliard  = 'milliárd';
   Valaszto  = '-';
   NumStrTable : Array[0..40] of String[10] =
   ('', 'egy', 'kettő', 'három', 'négy', 'öt',
        'hat', 'hét', 'nyolc', 'kilenc',
    '', 'tíz', 'húsz', 'harminc', 'negyven', 'ötven',
        'hatvan', 'hetven', 'nyolcvan', 'kilencven',
    '', 'tizen', 'huszon', 'harminc', 'negyven', 'ötven',
        'hatvan', 'hetven', 'nyolcvan', 'kilencven',
    '', 'száz', 'száz', 'száz', 'száz', 'száz',
        'száz', 'száz', 'száz', 'száz', '');
 var
   Kitevo   : Byte;
   Modulus  : NumericTipus;
   TModulus : NumericTipus;
   EzerStr  : String[5];
   Str      : String;
   StrHossz : Byte absolute Str;
 begin
   if Numeric > 2500
     then EzerStr := Ezer+Valaszto
     else EzerStr := Ezer;
   Str := '';
   Kitevo := 0;
   repeat
     Inc(Kitevo);
     Modulus := Numeric mod 10;
     if Numeric mod 1000 <> 0 then
       case Kitevo of
         4  : Str := EzerStr+Str;
         7  : Str := Millio+Valaszto+Str;
         10 : Str := Milliard+Valaszto+Str;
       end;
     case Kitevo mod 3 of
       0 : Str := NumStrTable[Modulus]+NumStrTable[30+Modulus]+Str;
       1 : Str := NumStrTable[Modulus]+Str;
       2 : if TModulus = 0
             then Str := NumStrTable[10+Modulus]+Str
             else Str := NumStrTable[20+Modulus]+Str;
     end;
     Numeric := Numeric div 10;
     TModulus := Modulus;
   until Numeric = 0;
   Str[1] := UpCase(Str[1]);
   if Str[StrHossz] = Valaszto then Dec(StrHossz);
   NumericToString := Str;
 end;
end.
```

Karakteres rutinok

Akik elégedetlenek a C fordító beépített függvényeivel, írhatnak gyorsabbat, jobbat. Példaként néhány karakteres üzemmódban használható függvényt mutatunk be.

```
/* ———————— ibmpc.c ———————— */
#pragma inline
#include <dos.h>
static union REGS rg;

/* ———————— Kurzor pozicionálás ———————— */
void cursor(int x, int y)
{
        rg.x.ax = 0x0200;
        rg.x.bx = 0;
        rg.x.dx = ((y << 8) & 0xff00) + x;
        int86(16, &rg, &rg);
}

/* ———————— Kurzor pozició lekérdezése ———————— */
void curr_cursor(int *x, int *y)
{
        rg.x.ax = 0x0300;
        rg.x.bx = 0;
        int86(16, &rg, &rg);
        *x = rg.h.dl;
        *y = rg.h.dh;
}

/* ———————— Kurzor típus beállítása ———————— */
void set_cursor_type(int t)
{
        rg.x.ax = 0x0100;
        rg.x.bx = 0;
        rg.x.cx = t;
        int86(16, &rg, &rg);
}

/* ———————— Képernyő feltöltés karakterrel ———————— */
void clear_screen(char attrib)
{
        cursor(0, 0);
        rg.h.al = ' ';
        rg.h.ah = 9;
        rg.x.bx = attrib;
        rg.x.cx = 2000;
        int86(16, &rg, &rg);
}
```

```
/* ———— video mód lekérdezés ———— */
int vmode()
{
        rg.h.ah = 15;
        int86(16, &rg, &rg);
        return rg.h.al;
}

/* —- karakter és attriibutum írása a video RAM -ba—- */
void vpoke(unsigned vseg, unsigned adr, unsigned chr)
{
if (vseg == 0xb000)             /* monochrome mód      */
poke(vseg, adr, chr);
else        {
        _DI = adr;              /* offset              */
        _ES = vseg;             /* video segmens       */
         asm cld;
        _BX = chr;              /* the attributum és a karakter */
         _DX = 986;             /* video status port   */
        do
        asm in  al,dx;
        while (_AL & 1);
        do
        asm in  al,dx;
        while (!(_AL & 1));
        _AL = _BL;
        asm stosb;
        do
        asm in  al,dx;
        while (_AL & 1);
        do
        asm in  al,dx;
        while (!(_AL & 1));
        _AL = _BH;
        asm stosb;
        }
}

/* —- karakter és attriibutum  olvasás a  video RAM -ból—- */
int vpeek(unsigned vseg, unsigned adr)
{
if (vseg == 0xb000)             /* monochrome mód      */
return peek(vseg, adr);
asm push ds;
_DX = 986;                      /* video status port */
_DS = vseg;                     /* video segment address */
_SI = adr;                      /* video character offset */
  asm cld;
        do
        asm in  al,dx;
        while (_AL & 1);
        do
        asm in  al,dx;
        while (!(_AL & 1));
        asm lodsb;
        _BL = _AL;
        do
        asm in  al,dx;
        while (_AL & 1);
        do
        asm in  al,dx;
         while (!(_AL & 1));
asm lodsb;
_BH = _AL;
_AX = _BX;
asm pop ds;
return _AX;
}
```

dBASE 5.0

Az új dBASE adatformátuma eltér a régitől, új mező típusok léptek be. Az ismert dbhead programot ennek megfelelően módosítottuk.

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

typedef struct
{
unsigned char ver;
unsigned char date[3];
long recno;
unsigned int headsize;
unsigned int recsize;
unsigned char blank[20];
}HEAD;

typedef struct
{
char name[11];
char typ;
char addr[4];
unsigned char len;
unsigned char dec;
unsigned char reserved[14];
}FIELD;

FIELD field;
HEAD head;
FILE *fp;

main (int argc,char *argv)
{
int fieldnum,i;
if (argc==2)
   {
    if(!(fp = fopen(argv[1],"r")))
     { puts ("Hiba"); exit(0);}
    fread(&head,32,1,fp);
    clrscr();
    puts (argv[1]);
    printf("Dátum  :%02d.%02d.%02d\n",head.date[0],head.date[1],head.date[2]);
    printf("Rekordszám :  %3d\n",head.recno);
    fieldnum = (head.headsize-31)/32;
    printf("Mezőszám   :  %3d\n",fieldnum);
    printf("Rekordméret:  %3d\n\n",head.recsize);
    printf("%3s %-10s %9s   %5s %3s\n" ,"No","Mezőnév","Tipus","Hossz","Dec");
    for(i=1; i <= fieldnum; i++)
    {
    fread(&field,32,1,fp);
    printf("%3d %-10s" ,i, field.name);
    if(field.typ=='N')   printf("%12s " , "Szám   ");
    if(field.typ=='D')   printf("%12s " , "Dátum  ");
    if(field.typ=='C')   printf("%12s " , "Szöveg ");
    if(field.typ=='B')   printf("%12s " , "Bináris");
    if(field.typ=='G')   printf("%12s " , "OLE    ");
    if(field.typ=='L')   printf("%12s " , "Logikai");
    if(field.typ=='M')   printf("%12s " , "Memo   ");
    printf("%5u %3u\n", field.len, field.dec);
    }
fclose (fp);
}
}
```

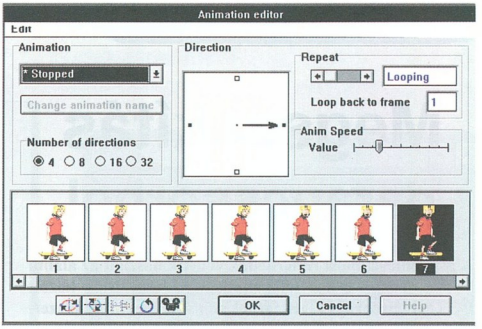
A programozásról kezdőknek

Elmélkedés a programozásról a programnyelvekről.

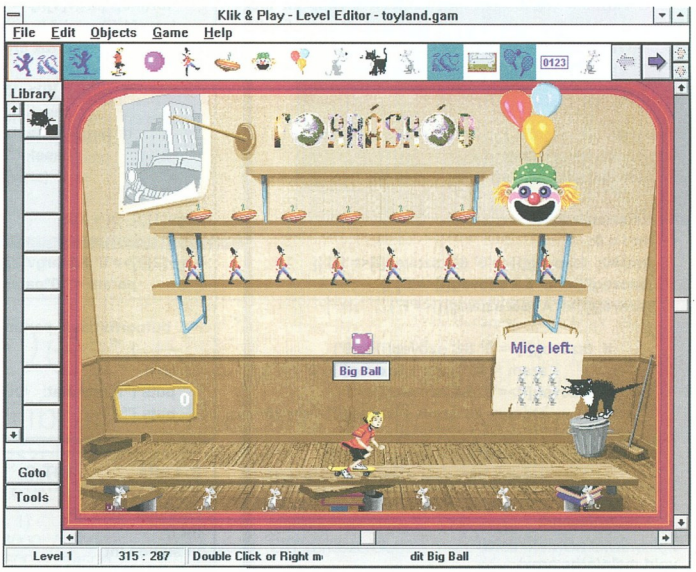
A számítógép egyszerű masina, meghatározott nagyságú számokkal végez egyszerű, előre beépített műveleteket. A műveleteket szintén számokkal azonosítjuk. Egy számítógép program nem más, mint az utasításokat és az adatokat meghatározott rendben tartalmazó számsorozat. A gép működése során ezeket a számokat olvassa, végrehajtja azokat az utasításokat amiket ezek a számok jelentenek, például egy perifériát vezérel vele. A számítógép programozása ennek a számsorozatnak az előállítása valamilyen adathordozón. Hogyan lehet ezt a számsorozatot előállítani? A legegyszerűbbnek az tűnik, hogy egy alkalmas szövegszerkesztővel egyszerűen sorban leírjuk őket. A dolog működik, de van két hatalmas probléma. Egyrészt nagyon kell ismerni a gép működését, másrészt egy használható program nagyon sok utasításból áll. Már az is segítség lenne, ha az egyes utasításokra névvel hivatkozhatnánk, és a többször előforduló részeket nem kellene mindannyiszor leírni, esetleg a legfontosabbak már előre meg lennének írva, nekünk csak hivatkozni kellene rájuk. Az ilyen szöveggel a számítógép nem tud mit kezdeni, azt egy alkalmas programmal át kell alakítani a megfelelő számsorozattá. Az ilyen programot assembler-fordítónak, a játékszabályt pedig, ami a neveket és az írásmódot tartalmazza, assembly nyelvnek nevezzük. Munkát takarítunk meg, ha a programokat szétválasztjuk két részre; egyik tartalmazza azokat a rutinokat amelyekkel a számítógép saját erőforrásait kezeli; amik minden programhoz kellenek; a másik rész csak az adott alkalmazás rutinjait tartalmazza. Az első részt operációs rendszernek, a másodikat felhasználói programnak hívjuk. Az első részt szakcégek készítik, így az egyes alkalmazásoknál csak a második részt, a felhasználói programot kell megírni, természetesen meg kell oldani a két rész között a kapcsolattartást. Komoly alkalmazások fejlesztése még így sem egyszerű, mert még mindig sok kódot kell írni és még mindig nagy szakértelem, a számítógép működésének alapos ismerete kell a programozáshoz. A programozás hatékonyságát úgy növelhetjük: ha a feladatot logikai elemekre bontjuk ésezeket az elemeket előre megírva, a programozáskor ezekből építkezünk. Csak hivatkoznunk kell rájuk, illetve a köztük lévő logikai kapcsolatokat kell leírnunk. Sok múlik azon, hogy ezeket az építőelemeket hogyan választjuk meg. Ha kicsik, akkor a programozáskor még mindig sok kódot kell írni, ha nagyok akkor vagy nagyon sokfélére van szükség, vagy nem illeszkednek a feladathoz. Egy lehetséges kiút, hogy az elemeket kicsire választjuk, de ezekből szabadon építhetünk nagyobbakat, amiket ezek után egy egységként használhatunk. A logikai kapcsolatok leírásában az előre megírt rutinok hívásának módjában az írásmódban célszerű megállapodni, ezeket hívják programnyelveknek. Mivel a feladatok és a programozói igények sokfélék, természetesen nincs egyedül üdvözítő megoldás, ezért sok programnyelv alakult ki, ezek állandóan fejlődnek és a sikeres megoldásokat átveszik egymástól. Szerencsés ha a programnyelv nem tartalmaz egy adott géptípusra jellemző utasításokat, mert így egy alkalmazás átvihető egy másik géptípusra, hordozhatóvá válik. Ezeken a programnyelveken megírt szöveg a forrásszöveg, vagy forráskód nem végrehajtható program. A felhasználására két út kínálkozik, az egyik: egy megfelelő program soronként elolvassa, értelmezi és végrehajtja az abban foglalt utasításokat. A másik: egy fordító, szerkesztő programmal előállítjuk belőle a futtatható programot, a már sokszor hivatkozott számsorozatot. A programkészítés még ezután is a beavatottak kiváltsága marad, hiszen meg kell tanulni egy programnyelvet, a fordító használatát. Az igazi áttörést a programkód nélküli fejlesztő eszközök jelentik. Általában grafikus felhasználói felület mellett, előre beépített lehetőségekből választva, azokat paraméterezve állíthatunk elő alkalmazásokat. Ebből következnek a rendszer korlátai: csak a beépített lehetőségekből választhatunk, így csak bizonyos fajta alkalmazások fejleszthetők, a programozás is és a futtatás is nagyon erőforrás igényes. Nézzünk egy példát: egy játékprogram fejlesztő képeit láthatjuk a továbbiakban. A program jelen esetben játék fejlesztése csupán egérrel oldható meg. Első lépésként megrajzoljuk a beépített rajzoló programmal a játék hátterét, az álló objektumokat, a mozgó figurák minden mozdulatát, természetesen más rajzoló programmal készült képek is átvehetők. Ez után meg kell adni, hogy az egyes figurák hogyan mozogjanak, a lehetőségeket egy táblázatból választhatjuk ki, illetve egy animáció-szerkesztővel állíthatjuk össze.

(1. ábra) Ha meghatároztuk az álló és mozgó figurákat, készítenünk kell egy táblázatot, (3.ábra) mi történjen ha ezek összetalálkoznak. Végül a háttérre egérrel fel kell raknunk a szereplőket és indulhat a játék.(2.ábra)

Természetesen nem csak játékprogramot fejleszthetünk a megismert módon, hanem bármit, amihez hasonló fejlesztőeszközt írnak. Egy ilyen programozói felület elkészítése nagyon sok munkát jelent, ezért csak nagy cégek készítenek ilyet. A felhasználhatóságuk és ezen keresztül a vevőkörük korlátozott, ezért drágák, lassan terjednek. Pedig ezekre az eszközökre a nagyközönségnek, a felhasználóknak lenne szükségük. A programozók továbbra is a hagyományos módon, a hagyományos programnyelvekkel dolgoznak, hiszen ezeket az alkalmazásokat valakinek valahogyan elő kell állítani. Ezzel a kör bezárult. Mit csináljon az, aki most kezd érdeklődni a számítógép, vagy annak programozása iránt, mivel kezdjen, merre induljon? Az új grafikus programozási lehetőségek és a hagyományos programnyelvek, pláne az assembler nem pótolják egymást, ki ki döntse el mihez van kedve, affinitása.és hagyja magát elcsábítani. A Forráskód a hagyományosan programozók, a számítástechnika iránt mélyebben érdeklődők lapja kíván lenni, de számolunk a realitással, a számítógéppel foglalkozók többsége felhasználó.



1. ábra
A szereplő mozgását meg kell terveznünk.



2 ábra
A kész háttérre feltesszük az álló és mozgó objektumokat és már indulhat a játék.

3.ábra.
Táblázatba kell foglalnunk az eseményeket.

		All the events										
GOTO	1	• leaves the play area on the top	✓					✓				
HELP	2	• leaves the play area on the left or right	✓					✓				
INFO	3	• leaves the play area on the bottom + Only one action when event loops	✓					✓				
➕	4	• Collision between and	✓					✓				
00:00 ▶	5	• Collision between and	✓			✓		✓	✓			
▪▪▪ ▶												

Megszakítás parancssorból

A gép lelkével most ismerkedők vehetik hasznát az alábbi három, Turbo C-ben írt rutinnak. Az első listán közölt POKE.C program segítségével adott szegmens és ofszetcímre írhatunk egy tetszőleges decimális számot. A második listában lévő OUT.C a gép tetszőleges PORT-jára ír egy decimális számot. A harmadik listában lévő INT.C rutinnal tetszőleges megszakítást hívhatunk meg parancssorból. A programokat COM formátumra fordítsuk, használatukat önmaguk magyarázzák.

1. Lista POKE.C

```
unsigned int seg,offs,i;
unsigned char param;
int hexatoi(char *szoveg )
{
    int szam = 0;
    int i = 0;
    for(i=0;  (szoveg[i]>='0' && szoveg[i]<='9') ||
    (szoveg[i]>='a' && szoveg[i]<='f') ||
    (szoveg[i]>='A' && szoveg[i]<='F') ;    i++)
    {
          if  (szoveg[i]>='0' && szoveg[i]<='9')
                szam = 16*szam + szoveg[i] -'0' ;
          if  (szoveg[i]>='a' && szoveg[i]<='f')
                szam =  16*szam + szoveg[i] -'a'+ 10
;
          if  (szoveg[i]>='A' && szoveg[i]<='F')
                szam = 16*szam + szoveg[i] -'A'+ 10
;
    } return(szam);
}
void main(argc,argv)
    int argc;
    char *argv[];
{
if(argc == 4)
    {
    seg    = hexatoi(argv[1]);
    offs   = hexatoi(argv[2]);
    for(i=0; argv[3][i]>='0' && argv[3][i]<='9';i++)
    param = 10*param + argv[3][i] -'0';
    pokeb(seg,offs,param);
    }
else
    {     puts ("Használat:  POKE  szegmenscím
                offsetcím     érték");
    puts ("       /hex/        /hex/        /dec/");
    exit(0);
    }
```

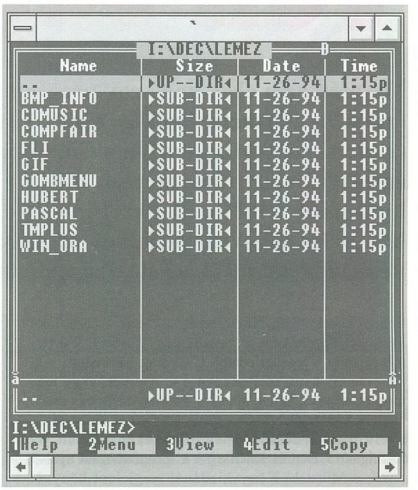
2. Lista. OUT.C

```
unsigned int basel,i;
unsigned char param;

void main(argc,argv)
    int argc;
    char *argv[];

{
if(argc == 3)
    {
     for(i=0;  (argv[1][i]>='0' && argv[1][i]<='9') ||
    (argv[1][i]>='a' && argv[1][i]<='f') ||
    (argv[1][i]>='A' && argv[1][i]<='F') ;    i++)
          {
           if  (argv[1][i]>='0' && argv[1][i]<='9')
                 basel = 16*basel + argv[1][i] -'0' ;
           if  (argv[1][i]>='a' && argv[1][i]<='f')
                 basel = 16*basel + argv[1][i] -'a'+ 10 ;
                  if  (argv[1][i]>='A' && argv[1][i]<='F')
                                basel  =  16*basel +
argv[1][i] -'A'+ 10;                          }
                                f  o  r  (  i  =  0  ;
argv[2][i]>='0' && argv[2][i]<='9';i++)
          param = 10*param + argv[2][i] -'0';

     outportb(basel,param);
          }
    else
    puts ("Használat:  OUT   portcím     érték");
    puts ("                    /hex/      /dec/");
```

3. Lista INT.C

```
#include <dos.h>
int i;
unsigned int
intnum,axparam,bxparam,cxparam,dxparam;
static union REGS rg;

int hexatoi(char *szoveg )
{
int szam = 0, int i = 0;
for(i=0;  (szoveg[i]>='0' && szoveg[i]<='9') ||
    (szoveg[i]>='a' && szoveg[i]<='f') ||
    (szoveg[i]>='A' && szoveg[i]<='F') ;    i++)

    {
    if  (szoveg[i]>='0' && szoveg[i]<='9')
          szam =  16*szam + szoveg[i] -'0' ;
    if  (szoveg[i]>='a' && szoveg[i]<='f')
          szam =  16*szam + szoveg[i] -'a'+ 10 ;
    if  (szoveg[i]>='A' && szoveg[i]<='F')
          szam = 16*szam + szoveg[i] -'A'+ 10 ;
    }
return(szam);
}
```

```
void main(argc,argv)
    int argc;
    char *argv[];
{
if (argc >= 2)
    {
    intnum = hexatoi(argv[1]);
    if (argc >= 3){ axparam = hexatoi(argv[2]);
          rg.x.ax =  axparam;}
    if (argc >= 4){ bxparam = hexatoi(argv[3]);
          rg.x.bx =  bxparam;}
    if (argc >= 5){ cxparam = hexatoi(argv[4]);
          rg.x.cx =  cxparam;}
   if (argc == 6){ dxparam = hexatoi(argv[5]);
          rg.x.dx =  dxparam;}
   int86(intnum, &rg, &rg);
    }
else
    {
    puts ("Megszakítás parancssorból\n");
    puts ("Használat:     INT     intnum  [axparam]
    [bxparam]  [cxparam]  [dxparam]");
    puts ("                        /hex/  /hex/      /hex/
    /hex/      /hex/ ");
    }
}
```



Az ön programjának van saját arca?

Ma már nem elég, hogy egy program működik, emberbarátnak is kell lennie. Ennek érdekében érthető, logikus, esztétikus, könnyen kezelhető felhasználói felülettel kell rendelkeznie. Megkezdjük a Compfair próbaszámban beígért sorozatunkat, amiben egy DOS alatt futó egérrel, billentyűvel egyaránt kezelhető grafikus ablakos felhasználói felületet fejlesztünk TURBO C nyelven. Az első részben a standard könyvtári függvények felhasználásával egy gombmenüt fejlesztünk, és szemléltetés képpen bemutatjuk az élő diagramok készítésének módját.

Ebben a sorozatban feltételezzük a C nyelv középfokú ismeretét, fordítóként a Borland cég Turbo C 2.0 termékét, mint a legelterjedtebb C fordítót használjuk. Aki nem ismeri kellő mélységben a C nyelvet, vagy más fordítót használ, annak ajánljuk a ComputerBooks kiadó "POGRAMOZZUNK C NYELVEN" című könyvét. A nyelv iránt érdeklődőknek alapműként, Brian W. Kernighan - Dennis Ritchie "A C programozási nyelv" című, a Műszaki Könyvkiadónál megjelent könyvét ajánljuk.

A gombmenü elkészítésénél a menü adatait érdemes struktúrába foglalni, hiszen egy alkalmazásnál több menü lehet és több menüpont. Így rövidebb, áttekinthetőbb kódot kapunk.
A **menüpontok** struktúrája az alábbi mezőket tartalmazza:

- a gomb felirata,
- az indítandó rutin, függvény címe,
- az átadandó paraméter, vagy paraméterek típusonként.

Példánkban:

```
typedef struct
        { char *name;
          (*rutin)();
          int param;
        } menupontok;
```

A **menü** struktúrája az alábbi adatokat tartalmazza:

- a menü bal felső sarkának koordinátái,
- a menü szélessége magassága,
- a gomboszlopok és -sorok száma,
- a kiválasztott menüpont száma,
- a menüpontokat tartalmazó struktúra,
- az aktív és az inaktív gombokat azonosító adatok.

Példánkban:

```
typedef struct {        int x0;
                        int y0;
                        int mszel;
                        int mmagas;
                        int oszlop;
                        int sor;
                        int select;
                        menupontok *m ;
                        int gombszin;
                        int aktivgombszin;
                        int feliratszin;
                        int aktfeliratszin;
                       } gombmenu;
```

A gombmenü kirajzolása sokféle módon történhet, például bittérképeket másolunk a képernyőre, és alma az aktív, körte az inaktív menüpontot jelenti. Így a gombmenü struktúrában az erre vonatkozó adatok különbözőek lehetnek. Példánkban a standard grafikát felhasználva rajzolunk gombokat. Az egyszerűség kedvéért a gombok nem maradnak bekapcsolt állapotban, de ennek sincs akadálya.
Nézzük a gomb rajzolást, a felirat később kerül rájuk.

```
void inbox( int x1, int y1, int x2, int y2 )
{       setcolor(8);
        line( x1, y1, x1, y2 );     /*+--- */
        line( x1, y1, x2, y1 );     /*|   */
        setcolor(15);
        line( x1+1, y2, x2, y2 );   /*  |*/
        line( x2, y1, x2, y2 );     /* --+ */
}

void outbox( int x1, int y1, int x2, int y2 )
{       setcolor(15);
        line( x1, y1, x1, y2 );     /*+---- */
        line( x1, y1, x2, y1 );     /*|   */
        setcolor(8);
        line( x1, y2, x2, y2 );     /*    |*/
        line( x2, y1, x2, y2 );     /* ---+ */
}

void lap( int g1, int g2, int sz, int m, int c )
        {outbox( g1, g2, g1+sz-1, g2+m-1 );
        setfillstyle(1, c );
        bar (g1+1, g2+1, g1+sz-2, g2+m-2);
        }

void gomb( int g1, int g2, int sz, int m, int c )
        {rectangle( g1, g2, g1+sz-1, g2+m-1 );
        outbox( g1+1, g2+1,g1+sz-2,g2+m-2 );
        setfillstyle(1, c );
        bar (g1+3, g2+3, g1+sz-3, g2+m-3);
        }
```

A gombmenü tényleges kirajzolását a minden akció után meghívott **gombra** függvény végzi. Kirajzolja az összes gombot és az aktív gombot újra rajzolja. Ahol a futásidő kritikus, ennél takarékosabb megoldás is elképzelhető, csak a változó gombokat frissítjük.

```
gombra(m)
gombmenu *m;
{
 int gszel, gmagas,i,j,x,y,amx,amy,tav=2,len;

  gszel  = ((m->mszel ) - (m->oszlop+1) *tav)/m->oszlop;
  gmagas  = ((m->mmagas) - (m->sor+1) *tav)/m->sor;
  setcolor(m->feliratszin);
  settextjustify(0,2);
  settextstyle(0,0,1);
                for (i = 0; i <= (m->sor - 1); i++)
                  { for (j = 0; j <= (m->oszlop - 1); j++)
                                {
                                x = m->x0+j * (gszel  +  tav) + tav;
                                y = m->y0 +i * (gmagas +  tav) + tav;
                                gomb( x, y , gszel, gmagas,m->gombszin);
                                len=8 * strlen((m->m+i*m->oszlop+j)->name);
                                outtextxy (x+(gszel-len)/2,y+4,
                                          (m->m+i*m->oszlop+j)->name);
                                }
                  }
  amx = m->select%m->oszlop  * (gszel  +  tav) + tav+m->x0;
  amy = m->select/m->oszlop  * (gmagas +  tav) + tav+m->y0;
  gomb( amx, amy, gszel, gmagas,m->aktivgombszin);
  setcolor(m->aktfeliratszin);
  len= 8 * strlen( (m->m+m->select)->name);
  outtextxy (amx+(gszel-len)/2,amy+4, (m->m+m->select)->name);
}
```

A gombmenüt a kurzor vezérlő billentyűkkel, ENTER-rel, ESC-pel vezérelhetjük. A billentyűk kódját a jobb olvashatóság kedvéért a fejléc állományba helyeztük. Nézzük tehát a kezelő függvényt:

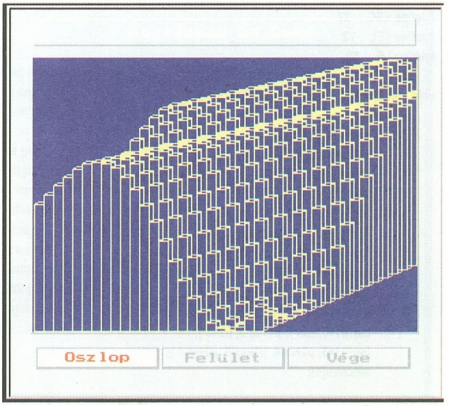
```
menuproc(m,c)
gombmenu *m;
int c;
{
    switch( c )
            {
            case ESC : quit(0);break;
            case JOBB: if ( m->select == m->oszlop * m->sor-1)
                                    m->select = m->select - m->oszlop+1;
                                    else  m->select ++;
                                    gombra(m);break;
            case BAL:  if ( m->select == 0) m->select  = m->oszlop-1;
                                    else m->select  --;
                                    gombra(m);
                                    break;
            case FEL: if ( m->select <= (m->oszlop-1))  break;
                                    else   m->select = m->select - m->oszlop;
                                    gombra(m);
                                    break;
            case LE:  if ((m->oszlop * m->sor)-m->select <= m->oszlop)  break;
                                    else   m->select = m->select + m->oszlop;
                                    gombra(m);
                                    break;
            case ENTER : gombra(m); if(*((m->m+m->select)->rutin)!=NULL)
                                    (*((m->m+m->select)->rutin))((m->m+m->select)->param);
                                     break;
            }
}
```

ESC-re a quit() függvény hívódik meg, ez biztosítja az ellenőrzött kilépést a programból, ezt természetesen nekünk kell megírni. ENTER-re elindítjuk a kiválasztott rutint. Ezeket úgy kell megírni, hogy az átadott paramétert át is tudják venni, ellenkező esetben érthetetlen dolgokat művel a programunk.

A menüprogram szemléltetésére egy élő, háromdimenziós grafikont mutatunk be a TURBO C standard grafikus függvényeinek felhasználásával.

A háromdimenziós grafikonok rajzolásának az a nehézsége, hogy a takart éleknek, felületeknek nem szabad látszani, ez pedig számításigényes feladat. Példánkban ezt azzal a trükkel oldjuk meg, hogy a grafikont hátulról rajzoljuk előre. Így viszont hamar kiszaladnánk a rajzfelületből. Ezért minden sor rajzolása előtt a már elkészült rajzot hátrább léptetjük a scroll() függvénnyel. Vigyázzunk a getimage() és a putimage() függvények legfeljebb 64K adattal dolgozhatnak.

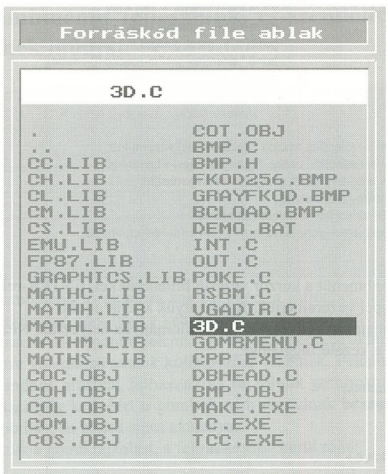
```
scroll(x0,y0,x1,y1,dx,dy,hatszin)
int x0,y0,x1,y1,dx,dy,hatszin;
{
size_t size;
int *buff;

            size = imagesize(x0,y0+dy,x1-dx,y1);
            buff = malloc( size );
            getimage(x0,y0+dy,x1-dx,y1,buff);
            putimage(x0+dx,y0, buff, COPY_PUT);
            setfillstyle(SOLID_FILL,hatszin);
            bar(x0,y0,x0+dx-1,y1);
            bar(x0+dx,y1-dy+1,x1,y1);
            free(buff);
}
```



Grafikus file-kereső

Témánk kapcsolódhat a gombmenüt ismertető cikkünkhöz, a grafikus képernyőn egy file-kereső ablakot hozunk létre.

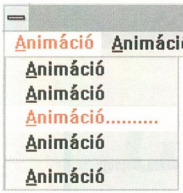
A szabványos TURBO C grafikát használó rutinban először egy keretet rajzolunk és kifestjük a belsejét. Ha a programot a paraméterként megadott szűrő szerint indítjuk, megkeressük az összes file-t az adott könyvtárból (de legfeljebb negyvenet). A megtalált file-okat két oszlopban az előre megrajzolt keretbe írjuk. Az elsőt letakarjuk egy kék oszloppal, és fehér betűvel újra kiírjuk. Az ablak tetején egy fehér oszlopban szintén kiírjuk kék színnel az első file-t. Ezután egy végtelen ciklussal olvassuk a billentyűzetet. Esc billentyűre kilépünk, a fel és le nyilakra eggyel növeljük, ill. csökkentjük egy számláló értékét (db.), ennek megfelelően az előző filet egy háttérszínű oszloppal letakarjuk, majd a nevét alapszínnel újra írjuk. A kiválasztott file-t fehér alapon kék színnel az ablak tetején, ill. kék alapon fehér színnel a saját helyén újra írjuk. Természetesen figyelnünk kell, melyik oszlopban vagyunk.



1. ábra A grafikus file-kereső ablaka.

```
#include <dir.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <dos.h>
#include <stdio.h>
#include <graphics.h>

typedef  char szo[13];
szo file_dir[40];

#define ESC    27
#define FEL    72
#define LE     80

void panel( int p1, int p2, int sz, int m ,int p, int c )
            {
            settextstyle(0,0,1);
            settextjustify(0,2);
            setcolor(15);
            line( p1, p2, p1+sz-1, p2);
            line( p1, p2, p1, p2+m );
            line(p1+sz, p2+1, p1+sz, p2+m );
            line( p1+1, p2+m, p1+sz, p2+m );
            setfillstyle ( 1, 7);
            bar(p1+1, p2+1, p1+4, p2+m-1 );
            bar(p1+sz-4, p2+1, p1+sz-1, p2+m-1 );
            bar(p1+5, p2+1, p1+sz-5, p2+5 );
            bar(p1+5, p2+m-4, p1+sz-5, p2+m-1 );
            setcolor(8);
            setcolor(8);
            line( p1+5, p2+5, p1+sz-5, p2+5 );
            line( p1+5, p2+5, p1+5, p2+m-5 );
            setcolor(15);
            line(p1+sz-5, p2+6, p1+sz-5, p2+m-5 );
            line( p1+6, p2+m-5, p1+sz-6, p2+m-5 );
            setfillstyle( p, c );
            bar(p1+6, p2+6, p1+sz-6, p2+m-6 );
            }


void gdir(int argc,char *argv,int x,int y)
{
int fnum,q,p,i=0,z,db,j,c,h,v,imax=0;
struct ffblk ffblk;
char path [MAXPATH];

if  (argc == 2)
    strcpy (path,argv);
else strcpy (path,"*.*");
fnum = findfirst(path,&ffblk,0x3f);
            while(!fnum)
```

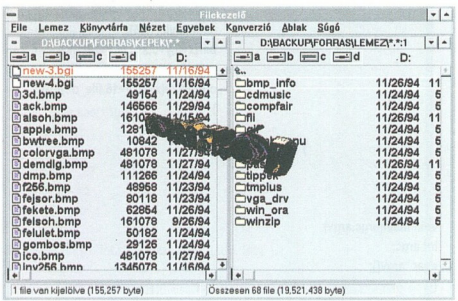
```
            {
            if(i<=39) strcpy (file_dir[i],ffblk.ff_name);
            fnum = findnext (&ffblk);
            imax=i;
            if (imax>=39) imax=39;
            i++;
            }
 panel(x,y,220,250,1,7);
 setfillstyle(1,15);
 bar(x+6,y+6,x+214,y+26);
 setcolor(1);
 outtextxy(x+60,y+16,file_dir[0]);

 z=0;
 p=y+40;
 q=x+10;
 setcolor(8);
 for (i=z; i <= (z+19); i++)
   {
            outtextxy( q,p,file_dir[i]);
            p = p + 10;
   }
p=y+40;
q=x+110;

 for (i= (z+20); i <= (z+39); i++)
   {
            outtextxy(q,p,file_dir[i]);
            p = p + 10;
   }
 setfillstyle(1,1),
 bar (x+8,y+38,x+108,y+48);
 setcolor(15);
 outtextxy(x+10,y+40,file_dir[0]);
 db = 0;
  while( (c=getch())!=ESC )
                        {
    switch( c )
              {         case FEL :  if (db >= 1)
                                    {
                                    if(db<=19)
                                    {h=x;
                                          v=y+db*10;
                                          }
                                          if(db>=20)
                                          {h=x+100;
                                          v=y+(db-20)*10;
                                          }
                                    setfillstyle(1,7);
                                    bar (h+8,v+38,h+108,v+48);
                                    setcolor(8);
                                    outtextxy(h+10,v+40,file_dir[db]);
                                    db = (db-1);
                                          if(db<=19)
                                    {h=x;
                                          v=y+db*10;
                                          }
                                          if(db>=20)
                                          {h=x+100;
                                          v=y+(db-20)*10;
                                          }
                                    setfillstyle(1,1);
                                    bar (h+8,v+38,h+108,v+48);
                                    setcolor(15);
                                    outtextxy(h+10,v+40,file_dir[db]);
                                    setfillstyle(1,15);
                                    bar(x+6,y+6,x+214,y+26);
                                    setcolor(1);
                                    outtextxy(x+60,y+16,file_dir[db]);
                                    }
                                    break;
                        case LE :  if (db <=imax-1)
                                          {
                                    if(db<=19)
                                    {h=x;
                                          v=y+db*10;
                                          }
                                          if(db>=20)
                                          {h=x+100;
                                          v=y+(db-20)*10;
                                          }
                                    setfillstyle(1,7);
                                    bar (h+8,v+38,h+108,v+48);
                                    setcolor(8);
                                    outtextxy(h+10,v+40,file_dir[db]);
                                    db = (db+1);
                                          if(db<=19)
                                    {h=x;
                                          v=y+db*10;
                                          }
                                          if(db>=20)
                                          {h=x+100;
                                          v=y+(db-20)*10;
                                          }
                                    setfillstyle(1,1);
                                    bar (h+8,v+38,h+108,v+48);
                                    setcolor(15);
                                    outtextxy(h+10,v+40,file_dir[db]);
                                    setfillstyle(1,15);
                                    bar(x+6,y+6,x+214,y+26);
                                    setcolor(1);
                                    outtextxy(x+60,y+16,file_dir[db]);
                                    }
                                    break;
                              }
  }
}

void main(argc,argv)
int argc;
char *argv[];
{
 int graphdriver=DETECT,graphmode,errorcode;
 initgraph(&graphdriver,&graphmode,"c:\\borlandc\\bgi");
 errorcode = graphresult();
 setbkcolor(7);setcolor(1);
 panel(100,70,220,25,1,12);
 setcolor(15);
 outtextxy(130,80,"Forráskód file ablak");
 gdir (argc,argv[1],100,100);
 getch();
 closegraph();
 }
```



Animáció a Windows képernyőn

Egy érdekes, sokszor használható, látványos effektust mutatunk be a Windows programozóknak. A teljes Windows képernyőre varázsolunk élő animációt.

A korábbiakban azt írtuk: minden Windows programnak ablaka, annak pedig ablakkezelő függvénye van. Ezt a kijelentést szeretnénk most helyesbíteni alábbi példánkkal, melyben egy érdekes lehetőségről, a teljes Windows képernyőn futó élő animáció készítéséről írunk. Ezt az effektust jól használhatjuk saját programjainkban figyelem felkeltésre, szemléltetésre. A program a windows.h beillesztése, a szükséges deklarációk után meghatározzuk a kirakandó bittérképek bal felső sarkát. Első lépésként elmentjük az animáció mögött lévő képernyő területet, a Ment() függvénnyel. A tulajdonképpeni animációt a Paint() függvény valósítja meg azzal, hogy mozgásfázisonként maszkot generál az erőforrásból beolvasott bittérképhez, majd a memóriában lévő bittérképet maszkolás



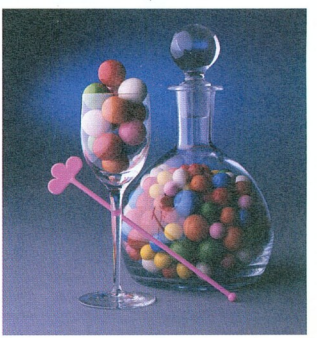
után kiteszi a képernyőre. Példánkban a fázisok cserélgetését, a Paint() függvény hívogatását egy egyszerű ciklus végzi. Valódi alkalmazásban célszerű ezt a WM_TIMER üzenetre bízni. Az egyes mozgásfázisokat bittérképként az erőforrás file-ba kell beépítenünk. Fekete hátterű, figurális, 16 színű bittérképet ajánlunk. Az animáció után természetesen takarítani kell, helyre kell állítani a hátteret, ezt a Vissza() függvény végzi, amihez fontos, hogy a Bbitm kezelő globálisnak legyen deklarálva. A forrásfile Borland C rendszerrel fordítható, a szükséges file-ok a lemezmellékleten találhatók.

```
/*                          win_ani.c                          */
/*          Átlátszó animáció a Windows képernyőn              */

#include <WINDOWS.H>
#define SZEL   320
#define MAGAS  200

HANDLE hInst;
HBITMAP   Bbitm;



void Paint (HDC ScreenDC, int epizod, POINT pont)
{
   HDC     FrameDC, MemDC, BackDC, MaskDC;
   HBITMAP OldBitmap, MBitmap, Mask, OldMask,
           FrameBitmap, OldFrameBitmap, OldBackBitmap;
   MaskDC        = CreateCompatibleDC (ScreenDC);
   Mask    = CreateBitmap (SZEL, MAGAS, 1, 1, NULL);
   OldMask = SelectObject (MaskDC, Mask);
   FrameBitmap = LoadBitmap (hInst, MAKEINTRESOURCE (epizod));
   FrameDC       = CreateCompatibleDC (ScreenDC);
   OldFrameBitmap = SelectObject (FrameDC, FrameBitmap);
   SetBkColor (FrameDC, RGB (0, 0, 0));
   BitBlt (MaskDC,0,0,SZEL,MAGAS,FrameDC,0,0,SRCCOPY);
   MemDC         = CreateCompatibleDC (ScreenDC);
   MBitmap    = CreateCompatibleBitmap (ScreenDC, SZEL, MAGAS);
   OldBitmap = SelectObject (MemDC, MBitmap);
   BackDC        = CreateCompatibleDC (ScreenDC);
   OldBackBitmap = SelectObject (BackDC, Bbitm);
   BitBlt (MemDC, 0, 0, SZEL, MAGAS, BackDC,  0, 0, SRCCOPY);
   BitBlt (MemDC, 0, 0, SZEL, MAGAS, MaskDC,  0, 0, SRCAND);
   BitBlt (MemDC, 0, 0, SZEL, MAGAS, FrameDC, 0, 0, SRCPAINT);
   BitBlt (ScreenDC,pont.x,pont.y, SZEL, MAGAS, MemDC,0,0,SRC-
COPY);
   SelectObject (FrameDC, OldFrameBitmap);
   DeleteDC (FrameDC);
   DeleteObject (FrameBitmap);
   SelectObject (MemDC, OldBitmap);
   DeleteDC (MemDC);
   DeleteObject (MBitmap);
   SelectObject (BackDC, OldBackBitmap);
   DeleteDC (BackDC);
   SelectObject (MaskDC, OldMask);
   DeleteDC (MaskDC);
   DeleteObject (Mask);
}
```

```
void Ment (POINT pont, int szel,int magas)
{
   HBITMAP OldBitmap;
   HDC     ScreenDC, BackDC;
   ScreenDC = GetDC (NULL);
   BackDC      = CreateCompatibleDC (ScreenDC);
   Bbitm = CreateCompatibleBitmap (ScreenDC, szel, magas);
   OldBitmap = SelectObject (BackDC, Bbitm);
   BitBlt (BackDC, 0, 0, SZEL, MAGAS, ScreenDC,pont.x, pont.y, SRCCOPY);
   SelectObject (BackDC, OldBitmap);
   DeleteDC (BackDC);
   ReleaseDC (NULL, ScreenDC);
}


void Vissza (POINT pont, int szel,int magas)
{
   HBITMAP OldBitmap;
   HDC     ScreenDC, BackDC;

   ScreenDC = GetDC (NULL);
   BackDC = CreateCompatibleDC (ScreenDC);
   OldBitmap = SelectObject (BackDC, Bbitm);
   BitBlt (ScreenDC, pont.x, pont.y, szel, magas, BackDC, 0, 0, SRCCOPY);
   SelectObject (BackDC, OldBitmap);
   DeleteObject (Bbitm);
   DeleteDC (BackDC);
   ReleaseDC (NULL, ScreenDC);
}


int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevhInstance,
                   LPSTR lpCmdLine, int nCmdShow)
{
   HDC   hdc;
   MSG msg;
   POINT pont;
   int epizod = 1,j;
   hInst =  hInstance;
   pont.x = (GetSystemMetrics (SM_CXSCREEN) - SZEL)/2;
   pont.y = (GetSystemMetrics (SM_CYSCREEN) - MAGAS)/2;
   Ment(pont, SZEL,MAGAS);
  for(j =0; j<40; j++)
  {
  hdc = GetDC (NULL);
  Paint (hdc, epizod, pont);
  ReleaseDC (NULL, hdc);
  epizod++;
  if(epizod>=9) epizod =1;
  }
 Vissza(pont,SZEL,MAGAS);
 PostQuitMessage(0);
}
```

```
/*                    win_ani.rc                    */

1  BITMAP PRELOAD DISCARDABLE "forr0001.BMP"
2  BITMAP PRELOAD DISCARDABLE "forr0002.BMP"
3  BITMAP PRELOAD DISCARDABLE "forr0003.BMP"
4  BITMAP PRELOAD DISCARDABLE "forr0004.BMP"
5  BITMAP PRELOAD DISCARDABLE "forr0005.BMP"
6  BITMAP PRELOAD DISCARDABLE "forr0006.BMP"
7  BITMAP PRELOAD DISCARDABLE "forr0007.BMP"
8  BITMAP PRELOAD DISCARDABLE "forr0008.BMP"
```

```
/*          win_ani.def                    */

NAME         WIN_ANI
DESCRIPTION  'animacio'
EXETYPE      WINDOWS
STUB         'WINSTUB.EXE'
CODE         PRELOAD MOVEABLE DISCARDABLE
DATA         PRELOAD MOVEABLE MULTIPLE
HEAPSIZE     8192
STACKSIZE    8192
```

Fotó:Patyi Árpád

Scroll

Karakteres képernyő első sorában képújságszerűen scrollozhatjuk a paraméterben megadott stringet. Jól használható a rutin BAT file-okban figyelemfelkeltésre, de be is építhetjük programjainkba is. A lista Agárdi Gábor munkája.

```
Scroll          Segment
assume  cs:Scroll,ds:Scroll

Row             equ     0       ;A szöveg helye a képen.
Color           equ     15      ;A szöveg színe.

Start:          xor     al,al   ; van-e paraméterrész
                cmp     es:[128],al
                jz      End     ;Ha nincs, ugrik a végére.
                mov     si,129  ;A szöveg kezdőcíme.
                mov     ax,0b800h ;A Videomemória
                                ;szegmenscímét
                mov     es,ax   ;es regiszterbe tölti.
                mov     al,160  ;Kiszámolja a megadott sor
                mov     bl,Row  ;utolsó karakterének  címét
                mul     bl
                add     ax,158
                mov     di,ax
                mov     ah,Color ;Ah regiszterben tárolja a
                                ;kiírandó szöveg színét.
.2_Scroll:  xor     bx,bx       ;A mutató nullázása.
.3_Scroll:  mov     al,[si+bx]  ;Kiolvassa az adott karaktert
                cmp     al,13   ;Ha az enter, akkor
                                ;elölről az egészet.
                jz      .2_Scroll
                mov     es:[di],ax ;Ha nem, ki lehet írni a
                                ;képernyőre.
                inc     bx
                mov     ax,100h ;Figyeli, van-e lenyomott
                                ;billentyű.
                int     16h
                jnz     End     ;Ha igen, akkor kiélépés.
                mov     dx,3dah ;Vertical Blank időzítés.
.4_Scroll:  in      al,dx
                test    al,8
                jz      .4_Scroll
                mov     cl,79   ;A kijelölt sort egy
                                ;karakterrel balra tolja.
                sub     di,158
.5_Scroll:  mov     ax,es:[di+2]
                mov     es:[di],ax
                add     di,2
                loop    .5_Scroll
                jmp     .3_Scroll

End:            mov     ax,4c00h ;Kilépés a DOS-hoz.
                int     21h

Scroll          Ends
                End     Start
```

Hangoló

A hardverrel ismerkedők, játékprogramírók és bárki más jó hasznát veheti az alábbi kétoktávos Turbo C nyelven íródott trillának.

```
#include <dos.h>


void hang(int f,int t)
{
  int h, l, p;
  long i;

l = (1190000L/f) % 256;
h   =(1190000L/f) / 256;
outp(67,182);
outp(66,l);
outp(66,h);
p =inp(97);
outp(97,p | 3);
for (i=0; i<1230L*t; i++);
outp(97,p & ~3);
}

main()
{
  int i,j;
  for (i=0; i<=2;i++)
  {
  for (j = 440;j<= 1760;j+=j/12 )
  hang(j,20);
  }

}
```

Remix

Nem gondoljuk, hogy mi vagyunk a legjobbak. Ha jobb megoldást tudsz, a te programod gyorsabb, ügyesebb, írd meg! Mi közzétesszük.

Clipper & Assembler

Ez a cikkünk is egy induló sorozat része, amiben szinte "leváltjuk" az ismert adatbáziskezelő rutinjait.

Nagyon örülök, hogy alkalmat kaptam egy olyan probléma megoldásának ismertetésére, amely régebben elég sok gondot okozott nekem is. Általában azok a clipper programozók, akik valamit is adnak arra, hogy programjuk igényes legyen, próbálkoztak már saját rutinok és függvények készítésével. Ebben az esetben nagyon előnyös lenne ha a rutinokat nem clipperben, hanem assemblerben tudnánk megírni. Csak azok kedvéért akik még nem kóstoltak bele, illik tudni, hogy sok mindenre nem képes a clipper, amit assemblerben vígan megoldhatunk. Készíthetünk olyan clipper alá írt assembler rutinokat, melyek gyorssá és hatékonnyá, esetleg széppé teszik programunkat.

Én magam is arra törekedtem mindíg, hogy a képernyőkezelést és a rendszerfüggő problémákat assemblerben kezeljem. Két dolog miatt is érdemes erre ügyelni. Az egyik, hogy ahogy gyűlnek a saját rutinjaim, úgy hagyhatom el azokat az idegen rutinokat, melyek működnek ugyan, de fogalmam sincs arról, mit csinál a változokkal, a memóriával stb. A másik, hogy ha megnézek egy átlagos adatbáziskezelő programot, már az arculata is borzalmas. Márpedig az, hogy hogyan jelenik meg egy program a képernyőn, rengeteget számít, mivel egy program kezelhetősége részben függ a monitoron megjelenő információtól is.

Nézzük mit kell tudni egy ilyen függvényről. Először is a felépítése. Kétfajta módja is van annak, hogy elkészítsük, mi a könnyebbiket választjuk. Ez annyit jelent, hogy a példalistában vastaggal írt szövegrész (CLfunc,CLcode,CLret) tulajdonképpen az EXTENDA.INC nevű fileban megírt makrók. Ezek segítségével tudjuk beilleszteni eljárásunkat a clipper programba. Ha belenézünk ezekbe a makrókba, akkor látható, hogy a verem igen fontos szerepet játszik a paraméterek átadásában. Most inkább arról, amit feltétlenül tudni kell. Az első sorban az includeval adjuk meg a fordítónak, hogy honnan vegye a makrókat amiket használunk. A codeseg mögé csak egy nevet kell írni, tulajdonképpen bármit. A CLpublic utáni, kisebb-nagyobb jel közé írt string lesz az eljárás neve, tehát a clipperből majd ezen a néven hivatkozhatunk rá. Jelen esetben ez "if_at()" lesz. Ez idáig tulajdonképpen deklarációs rész.

A tényleges rutin írása ezután kezdődik. A CLfunc makró után megadjuk a függvény típusát, tehát, hogy milyen típusú adattal tér vissza majd. Ezután megadjuk az eljárás nevét, amit már a CLpublic-nál megadtunk. A függvény típusa lehet "none", "log", "int" és "char". Ha a none-t írjuk akkor procedúráról van szó, aminek nincs visszatérő értéke. A többi esetben boolean, egész szám és karakter típusú a függvényünk. A miénk visszatérő értéke az AX regiszter értékétől függően true vagy false. Ha az adott gép amin fut a program AT, akkor true, ha XT akkor false. Magáról a paraméter átvételéről később írnék. A Clcode és a CLret közé írjuk a saját kis programunkat.

A fordítás: ASM filéból OBJ a masm.exe-vel. Utána a LIB.EXE segítségével libet készítünk.

Szalay Zsolt

```
INCLUDE  EXTENDA.INC

CODESEG  sp11

CLpublic <IF_AT>

   CLfunc log IF_AT <>
      CLcode
      push    es
      mov     ax,0f000h
      mov     es,ax
      mov     di,0fffeh
      mov     al,es:[di]
      pop     es
      cmp     al,0fch
      jne     no_at
      mov     ax,1
      jmp     kif
no_at: mov    ax,0
kif:
      CLret   ax
End
```

Rec	LABEL	CATALOG	CONFIG	ARTIST	TITLE	UPC	LABEL_ABBR
1	7674	10673	4	MCENTIRE*REBA	IT'S YOUR CALL	881106734000	MCA1
2	7	[illegible]	4	SA[illegible]	[illegible]E DELUXE	7464531784000	COL
3	[illegible]	[illegible]	4	E[illegible]	[illegible]HERD MOONS	7599267754000	WAR
4	[illegible]	[illegible]	[illegible]	TH[illegible]	[illegible]OOOHHH-ON THE TLC TIP	3008260034000	LAF
5	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]E GAVE ALL	3145106354000	POL2
6	7822	18658	4	BROOKS & DUNN	BRAND NEW MAN	7822186584000	ARI
7	2061	61265	4	QUE[illegible]	GREATEST [illegible]S	2061612654000	HOL
8	7777	98531	4	MEGA[illegible]	[illegible]	7777985314000	[illegible]
9	7464	48881	4	CAR[illegible] CHAPIN	[illegible]	[illegible]	[illegible]
10	7599	45153	4	ERASURE	POP!-FIRST 20 HITS	[illegible]	[illegible]
11	7464	48710	4	KRIS KROSS	TOTALLY KROSS[illegible]	[illegible]	[illegible]
12	7502	3	4	VERY SPECIAL CHRISTMAS	[illegible]	[illegible]	[illegible]
13	7863	66044	4	AL ARAN[illegible]	[illegible]	[illegible]	[illegible]
14	7777	460[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
15	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

dBASE 5.0 for Windows

Lemez nyilvántartó
Egy muzikális alkalmazás

Az új dBase for Windows nem nagyon engedi szóhoz jutni a programozót, vagy tálán mégis? Lehetőségeinek bemutatására nézzünk egy példát. A lemez-nyilvántartó program listájában sok ismerős dolgot találunk.

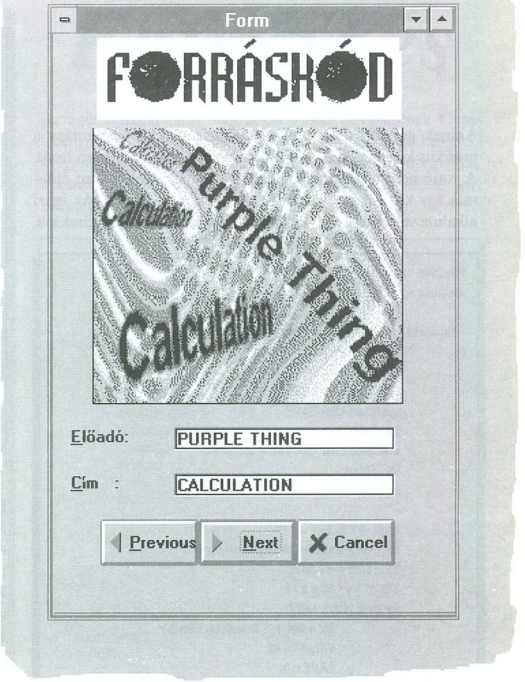
A dBase for Windows segítségével könnyen, kényelmesen hozhatunk létre adatbázisokat, készíthetünk jelentéseket, lekérdező-ablakokat. Közben kódot sem kell írni, de elfogad a saját nyelvén készült programszöveget, azt egy átmeneti bináris kódra fordítja és futtatja. Az igazi újdonság, hogy ha kell a kódot is megírja. Az új dBase a konkurens Windowsos adatbáziskezelőkhöz hasonlóan a DBT állományban képet, hangot és tetszőleges Windows objektumot is tárolhat. Példánkban egy képeket és adatokat tartalmazó adatbázishoz készítettünk egy lekérdező ablakot a vizuális programozás lehetőségét nyújtó FORM generátor segítségével. Az eredményt WFM kiterjesztésű szövegfileba mentettük, a következő oldalon olvasható. A futtatás eredménye egy ablak, ahol az előadó és a lemez címén túl a lemez borító is is látható. Gombok szolgálnak az előre és hátra való keresésre ill. az ablak lezárására. A fentieken túl egy logó elhelyezésére is volt mód, de az ékezetes karaktereket sehogyan sem tudtuk elfogadtatni.

dBASE 5.0 for Windows - [Table Records (MUSIC.DBF)]

Edit View Table Properties Window Help

CATALOG	CONFIG	ARTIST	TITLE
10673	4	MCENTIRE*REBA	
46443	4	SADE	
61049	4	ENYA	
45037	4	T...	
10...			

```
** END HEADER — do not remove this line*
* Generated on 12/04/94
*
LOCAL f
f = NEW FORRKODFORM()
f.Open()
CLASS FORRKODFORM OF FORM
 Set Procedure To C:\DBASEWIN\SAMPLES\BUTTONS.CC additive
  this.Width =        51
  this.Top =          0
  this.Left =         1
  this.Text = "Form"
  this.Height =       25.7051
  this.HelpFile = ""
  this.HelpId = ""
  this.View = "music.qbe"
  this.ScrollBar =    2

 DEFINE RECTANGLE RECTANGLE1 OF THIS;
    PROPERTY;
      Width       50,;
      Top         -1.0586,;
      Left        0,;
      Border .T.,;
      Text "Rectangle1",;
      ColorNormal "N/W",;
      Height      26.1172

 DEFINE TEXT TEXT1 OF THIS;
    PROPERTY;
      Width       15,;
      Top         0.5,;
      Left        1,;
      Border .F.,;
      FontSize    18,;
      Text "Lemez katalógus",;
      ColorNormal "W/W",;
      Height      2.0293

 DEFINE IMAGE IMAGE1 OF THIS;
    PROPERTY;
      Width       38,;
      Top         4,;
      Left        5,;
      Height      12,;
      DataSource "BINARY MUSIC->COVER"

 DEFINE TEXT TEXT2 OF THIS;
    PROPERTY;
      Width       15,;
      Top         17,;
      Left        1,;
      Border .F.,;
      Text "  &Eloado:",;
      ColorNormal "B+/W",;
      OldStyle .T.,;
      Height      1


DEFINE ENTRYFIELD ENTRYFIELD1 OF THIS;
    PROPERTY;
      Width       27,;
      Top         17,;
      Left        15,;
      Border .T.,;
      ColorNormal "N/W+",;
      DataLink "MUSIC->:ARTIST:",;
      Height      1
```




```
DEFINE TEXT TEXT3 OF THIS;
   PROPERTY;
     Width       15,;
     Top         19,;
     Left        1,;
     Border .F.,;
     Text "  &Cim  :  ",;
     ColorNormal "B+/W",;
     OldStyle .T.,;
     Height      1


DEFINE ENTRYFIELD ENTRYFIELD2 OF THIS;
   PROPERTY;
     Width       27,;
     Top         19,;
     Left        15,;
     Border .T.,;
     ColorNormal "N/W+",;
     DataLink "MUSIC->:TITLE:",;
     Height      1

DEFINE CANCELBUTTON CANCELBUTTON1 OF THIS;
   PROPERTY;
     Width       12,;
     Top         21,;
     Left        30,;
     Group .T.,;
     Height      2

DEFINE NEXTBUTTON NEXTBUTTON1 OF THIS;
   PROPERTY;
     Width       12,;
     Top         21,;
     Left        18,;
     Group .T.,;
     Height      2


DEFINE PREVBUTTON PREVBUTTON1 OF THIS;
   PROPERTY;
     Width       12,;
     Top         21,;
     Left        6,;
     Group .T.,;
     Height      2


DEFINE IMAGE IMAGE2 OF THIS;
   PROPERTY;
     Width       41.5,;
     Top         -0.2939,;
     Left        3.5,;
     Height      4.2939,;
     DataSource "FILENAME K256.BMP",;
     Alignment   2

ENDCLASS
```

Sprite

Akinek gondot jelent egy tetszőleges alakzat kirajzolása a grafikus képernyőre, ajánljuk a figyelmébe a következő listát. A rutin nem túl gyors, ezért nagyobb alakzatokat nem érdemes így kirakni és az ábra leírása is hosszadalmas. Az igazi alkalmazási lehetőség, kis ábrák, "sprite"-ok, logók, kirakása.

```
#include <dos.h>
#include <graphics.h>

void Sprite( int x, int y, char *bitmap, int byteszel, int sor)
{
        char far *ptr;
        register int i, j;
        unsigned char XOR;
        unsigned char SAVE;
        int  Shift;
        unsigned char ANDVAL = 0;
        int  YStart;
        int  SaveShift;

        Shift  = x % 8;
        for (i=0; i<Shift; ++i)ANDVAL += i<<1;
        SaveShift = 8 - Shift;
        YStart = y * 80 + x / 8;
        for (j=0; j<sor; ++j) {
                ptr = MK_FP( 0xA000U, YStart );
                YStart += 80;
                SAVE = 0;
                for (i=0; i<byteszel; ++i) {
                XOR = (*bitmap >> Shift) + SAVE;
                SAVE = ( (*bitmap & ANDVAL) << SaveShift );
                *ptr++ ^= XOR;
                bitmap++;
                }
                *ptr   ^= SAVE;
        }
}

unsigned char A[145] = {
        0x00,  0x00,  0x80,  0x00,  0x00,
        0x00,  0x01,  0xc0,  0x00,  0x00,
        0x00,  0x01,  0xc0,  0x00,  0x00,
        0x00,  0x01,  0xe0,  0x00,  0x00,
        0x00,  0x03,  0xe0,  0x00,  0x00,
        0x00,  0x03,  0xf0,  0x00,  0x00,
        0x00,  0x06,  0xf0,  0x00,  0x00,
        0x00,  0x06,  0xf8,  0x00,  0x00,
        0x00,  0x0c,  0xf8,  0x00,  0x00,
        0x00,  0x0c,  0x7c,  0x00,  0x00,
        0x00,  0x18,  0x7c,  0x00,  0x00,
        0x00,  0x18,  0x3e,  0x00,  0x00,
        0x00,  0x30,  0x3e,  0x00,  0x00,
        0x00,  0x30,  0x1f,  0x00,  0x00,
        0x00,  0x60,  0x1f,  0x00,  0x00,
        0x00,  0x60,  0x0f,  0x00,  0x00,
        0x00,  0xc0,  0x0f,  0x80,  0x00,
        0x00,  0xc0,  0x0f,  0x80,  0x00,
        0x01,  0xff,  0xff,  0xc0,  0x00,
        0x01,  0xff,  0xff,  0xc0,  0x00,
        0x03,  0x80,  0x03,  0xe0,  0x00,
        0x03,  0x00,  0x03,  0xe0,  0x00,
        0x07,  0x00,  0x01,  0xf0,  0x00,
        0x06,  0x00,  0x01,  0xf0,  0x00,
        0x0e,  0x00,  0x00,  0xf8,  0x00,
        0x0c,  0x00,  0x00,  0xf8,  0x00,
        0x1c,  0x00,  0x00,  0xfc,  0x00,
        0x3e,  0x00,  0x00,  0xfe,  0x00,
        0xff,  0xc0,  0x07,  0xff,  0x80
};

void main()
{
        int graphdriver=VGA, graphmode=VGAHI;
        initgraph(&graphdriver,&graphmode,"");
        Sprite( 120, 100, A, 5, 29 );
        getch();
        closegraph();
}
```

Ha sikerült begépelni a fenti listát, TURBO C-vel fordíthatjuk. Eredményként egy nagy fehér "T" jelenik meg a képernyőn. Ha már így belejöttünk, csináljuk ezt meg színesben. A képkirakás ezen a módon lényegesen egyszerűbb, a képet tartalmazó tömb megírása viszont gondot okozhat. A következő számban ehhez adunk egy kis segítséget. A forrás és a futtatható file a következő havi lemezen lesz. A bemutatott példa egy tátott szájú figurát rajzol a képernyőre, amit a kurzormozgató billentyűkkel mozgathatunk. Az Esc billentyű lenyomására vége a csodának. Fordítás szintén TURBO C-vel.

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

#define ESC    27
#define JOBB   77
#define BAL    75
#define FEL    72
#define LE     80

main()
{
 int driver, mode,c, x=300, y=120;

 char image[294]  = {
                                        19,  0, 23,  0,  0,254,  0,  0,254,  0,
                                         0,254,  0,255,  1,240,  3,255,128,  3,
                                       255,128,  3,255,128,252,  0,112,  7,143,
                                       192,  7,143,192,  7,143,192,248,  0, 48,
                                        31,119,240, 31, 23,240, 31, 23,224,224,
                                       112,  0, 31,119,224, 31, 55,224, 31, 55,
                                       192,224,112, 16, 63,119,192, 63, 23,192,
                                        63, 23,128,192,112,48,127,143,128,127,
                                       143,128,127,143,  0,128,  0,112,127,255,
                                         0,127,255,  0,127,254,  0,128,  0,240,
                                       255,254,  0,255,254,  0,255,252,  0,  0,
                                         1,240,255,252,  0,255,252,  0,255,248,
```

```
                                0, 0, 3,240,255,248, 0,255,248, 0,
                               255,240, 0, 0, 7,240,255,240, 0,255,
                               240, 0,255,224, 0, 0, 15,240,255,248,
                                0,255,248, 0,255,240, 0, 0, 7,240,
                               255,252, 0,255,252, 0,255,248, 0, 0,
                                3,240,255,254, 0,255,254, 0,255,252,
                                0, 0, 1,240,127,255, 0,127,255, 0,
                               127,254, 0,128, 0,240,127,255,128,127,
                               255,128,127,255, 0,128, 0,112, 63,255,
                               192, 63,255,192, 63,255,128,192, 0, 48,
                                31,255,224, 31,255,224,31,255,192,224,
                                0, 16, 31,255,240, 31,255,240, 31,255,
                               224,224, 0, 0, 7,255,192, 7,255,192,
                                7,255,192,248, 0, 48, 3,255,128, 3,
                               255,128, 3,255,128,252, 0,112, 0,254,
                                0, 0,254, 0, 0,254, 0,255, 1,240,
                                0, 0, 0, 0, 0, 0, 0, 0, 0,255,
                               255,240, 0, 0
                              };

  driver=EGA;
  mode=EGAHI;
  initgraph(&driver, &mode, "");

outtextxy(10,10,"Nyilakkal mozgathatja, Esc-re Kilép");

  putimage(x,y,image,COPY_PUT);

while( (c=getch())!=ESC )
                        {
                        switch( c )
                         {
                            case FEL :  if(y>10)
                                        {
                                        putimage(x,y,image,XOR_PUT);
                                        y = y - 5;
                                        putimage(x,y,image,XOR_PUT);
                                        }
                                        break;
                            case LE  :  if(y<340)
                                        {
                                        putimage(x,y,image,XOR_PUT);
                                        y = y + 5;
                                        putimage(x,y,image,XOR_PUT);
                                        }
                                        break;
                          case BAL  :  if(x>10)
                                        {
                                        putimage(x,y,image,XOR_PUT);
                                        x = x - 5;
                                        putimage(x,y,image,XOR_PUT);
                                        }
                                        break;
                          case JOBB :  if(x<620)
                                        {
                                        putimage(x,y,image,XOR_PUT);
                                        x = x + 5;
                                        putimage(x,y,image,XOR_PUT);
                                         }
                                        break;
                        }
                        }

  closegraph();

}
```

Egerészés

Az egér használatáról még sokat írunk. Bevezetőnek fontos tudni, hogy az egeret a meghajtó programok a hexa 33 megszakításra telepítik. A megszakítás hívásához egy példa: a karakteres képernyőn bekapcsoljuk az egérkurzort és egy billentyű leütése után kikapcsoljuk.

```
#include <dos.h>
#include <alloc.h>
int a;


int mouse(int m1, int m2,int m3, int m4)
{
        union       REGS       mousreg;

        mousreg.x.ax = m1;
        mousreg.x.bx = m2;
        mousreg.x.cx = m3;
        mousreg.x.dx = m4;
        int86(0x33, &mousreg, &mousreg);
        return (mousreg.x.ax);
}



void main()
{

a = mouse (0x00,0,0,0);
if (a== 0xffff)
{
puts ("Van egér, bekapcsoltam a kurzort");
puts ("Bármely billentyűre kikapcsolom  ");
mouse( 0x01,0,1,0);
getch();
mouse( 0x02,0,0,0);
}
else puts ("nincs egér");
}
```

Visszabeszél a könyvelőprogram

Lapunkban rendszeresen megszólaltatunk gyakorló profikat. Az első számban az apropót egy könyvelőprogramban alkalmazott érdekes megoldás adta.

Én, mint szoftverfejlesztő, ügyviteli, nyilvántartási és vezérlési programokat írok. Az egyik ügyfelem egy könyvelési cég, amely kevés alkalmazottal több, mint 100 vállalkozás bérszámfejtését és könyvelését végzi. Egy ilyen cégnél nagyon fontos az időtényező. A sebességet a számítógépekkel és a programokkal bizonyos határok között ugyan lehet növelni, de az adatok rögzítése nagyságrendekkel több időt visz el, mint a feldolgozás. Jelentősen csak a rögzítés gyorsításával javíthatjuk a teljesítményt. Hogyan valósítható ez meg?

Először is mivel a könyvelő egyik keze foglalt, a programban az adatok rögzítéséhez elég csak a keypadot a numerikus billentyűket használni. A mentésre a *, egy lista lekérésére a + billentyű szolgál, így a rögzítő egyik keze felszabadul, a másikban foghatja a rögzítésre váró számlákat. Már csak azt kellene megoldani, hogy ne kelljen állandóan a monitorra felpillantani, mert hát ugye az ellenőrzés nagyon fontos. Erre egy remek, egyszerű és nem túl drága megoldás, a Nikol Elektronika fejlesztése a PC-ROBOT kínálkozik. Ez a rendszer lehetővé teszi a magyar nyelvű szöveg érthető kimondását. Tartozik hozzá egy beszédszintetizátorkártya hangszóróval és néhány program. A rendszer felélesztése egyszerű, a beszédet egy kb. 75 Kb-os rezidensprogram állítja elő. Már csak a programunkba való beillesztés van hátra. Az unit meghívásakor a RobotInit eljáráson keresztül teszteli, hogy a beszélőrendszert installálták-e, a szükséges rezidensprogram elindult-e. Ehhez a RobotInitedProcot hívja meg. Ez a függvény az AX-et feltölti F000 hexával és meghívja a 2F hexa szoftver interruptot, ami AL = FF hexaértékkel jelzi a rendszer installálását. A RobotInited globális változó, a teszt eredményére áll be. Ezután a ReadRobotOpcioRec felolvassa a már letárolt opciókat, vagy betölti az alapértelmezettet. Ezek az opciók a RobotOpcioRec globális változóban érhetők el. Egy szöveget kimondatni úgy lehet a rendszerrel, hogy az AX-be F001 hexát kell tölteni, és a DS:DX-et a kimondandó szöveg kezdő címére kell állítani. Ezután már lehet hívni az előbb említett interruptot. A beszéd opcióit -- sebesség, hangmagasság, hangtónus, suttogás, hangosság, intonáció, tagolás és hanghosszúság -- az ESC vezérlőkarakter és egy kód kimondatásával lehet beállítani. Az opciók beállítását célszerű a felhasználóra bízni, így a beszédet az saját fülének megfelelőre hangolhatja. Erre célszerű a felhasználói programban, például a rendszerparaméterek menüpontban egy beolvasó ablakot nyitni. A könyvelői programba való beillesztés előnye, hogy a számlákról nem kell felnézni, mert a hangszóróból -- ha több gépen használják, akkor a fülhallgatóból -- kontrollként vissza lehet hallani a begépelt vagy kiválasztott adatokat, pl: „Kiss István tartozik Tízezer forint + huszonöt százalék áfával". Ezen a megoldáson kívül még számtalan helyen fel lehet használni a megadott egyszerű uniton -- RobotDrv.Pas -- keresztül a PC-ROBOT beszélőrendszert.

Balla László vill. mérnök
Telefon: 226-8962

1. Lista ROBOTDRV.PAS

```
unit RobotDrv;

interface type

(* A robot opció rekord típus deklarációja *)

  RobotOpcioRecTipus = record
    Hasznalatban     : Boolean;
    Sebesseg         : Byte;
    Hangmagassag     : Word;
    Hangtonus        : Byte;
    Suttogas         : Byte;
    Hangossag        : Byte;
    Intonacio        : Byte;
    Tagolas          : Byte;
    Hanghosszusag    : Byte;
  end;


procedure RobotKimond(St : String);
procedure SetRobotSebesseg(S : Byte);
procedure SetRobotHangmagassag(P : Word);
procedure SetRobotHangtonus(V : Byte);
procedure SetRobotSuttogas(W : Byte);
procedure SetRobotHangossag(A : Byte);
procedure SetRobotIntonacio(I : Byte);
procedure SetRobotTagolas(X : Byte);
procedure SetRobotHanghosszusag(T : Byte);
procedure SetRobotOpcioRec(Rec :
RobotOpcioRecTipus);


function WriteRobotOpcioRec : Boolean;
function ReadRobotOpcioRec : Boolean;
```

```
var

(* Globális változók *)

  RobotInited   : Boolean;
                  (* A rendszer inicializálva van-e?*)
  RobotOpcioRec : RobotOpcioRecTipus;
                  (* Az aktuális opciókat tárolórekord*)
implementation
uses
 Dos;

const
  EscStr             = #27;
  RobotFName         = 'ROBOT.DAT';
(* Annak file-nak a neve, amiben az opciókat tároljuk*)
DefaultSebesseg      = 7;
DefaultHangmagassag  = 100;
DefaultHangtonus     = 1;
DefaultSuttogas      = 0;
                  (* Az alapértelmezett beállítások *)
  DefaultHangossag   = 10;
  DefaultIntonacio   = 1;
  DefaultTagolas     = 1;
  DefaultHanghosszusag = 0;

procedure ErrorMessage(St : String);
                        (* Hibaüzenet kiírása. *)
        (* Az aktuális környezetben ezt az eljárást kell
                                          kicserélni.
*)
  begin
    WriteLn(St);
  end;

function Long2Str(L : LongInt) : String;
                  (* Egész sztringgé alakítása. *)
  var
    St : String;
  begin
    Str(L, St);
    Long2Str := St;
  end;

function RobotInitedProc : Boolean;       (* A beszélő
rendszer installálását teszteli. *)
  var
    R : Registers;
    S : Word;
  begin
    R.AX := $F000;
    Intr($2F, R);
(* 2F hexa szoftver interrupton keresztül lehet elérni.*)
    RobotInitedProc := (R.AL = $FF);
  end;

        (* Ha a visszatérési érték FF hexa, a *)
        (* rendszer installáva van. *)

procedure LowRobotKimond(St : String);

        ( * A rendszer tesztelése nélkül kimondja
              a megkapott St sztringet. *)
```

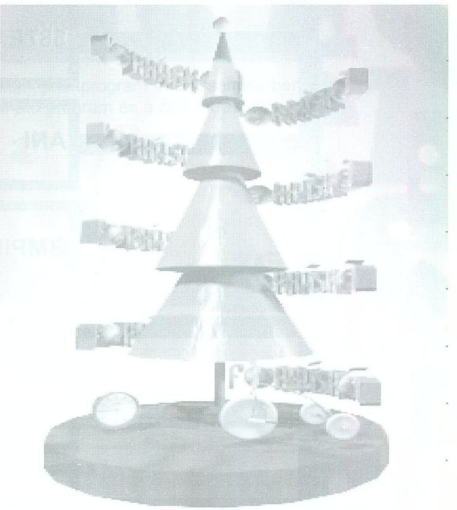
```
var
   R : Registers;
   S : Word;
 begin
   R.DS := Seg(St);
   R.DX := Ofs(St);
   R.AX := $F001;
  (* Az AL=01 jelzi, hogy a DS:DX az St kezdőcímére
                                          mutat *)
   Intr($2F, R);
   S := R.AX;
   if S > 0
    then begin
      ErrorMessage      ('PC Robot hiba :
'+Long2Str(S));
      exit;
    end;
 end;

procedure RobotKimond(St : String);
        (* Ha a rendszer installálva van, a megkapott
                St sztringet átadja kimondásra. *)
 begin
   if not RobotInited then exit;
   LowRobotKimond(St);
 end;

procedure SetRobotSebesseg(S : Byte);
        (* A kimondás sebességét  állítja be. *)
 begin
   if not (S in [1..7]) then S := DefaultSebesseg;
   RobotKimond(EscStr+'s'+Long2Str(S));
 end;

procedure SetRobotHangmagassag(P : Word);
(* A kimondás hangmagasságát  állítja be. *)
 begin
   if (P < 20) or (P > 400) then P :=
DefaultHangmagassag;
   RobotKimond(EscStr+'p'+Long2Str(P));
 end;

procedure SetRobotHangtonus(V : Byte);
        (* A kimondás hangtónusát állítja be. *)
 begin
   if not (V in [1..2]) then V := DefaultHangtonus;
   RobotKimond(EscStr+'v'+Long2Str(V));
 end;

procedure SetRobotSuttogas(W : Byte);
(* A suttogást lehet ki-be kapcsolni. *)
 begin
   if not (W in [0..1]) then W := DefaultSuttogas;
   RobotKimond(EscStr+'w'+Long2Str(W));
 end;

procedure SetRobotHangossag(A : Byte);
(* A kimondás hangosságát  állítja be. *)
 begin
   if not (A in [1..13]) then A := DefaultHangossag;
   RobotKimond(EscStr+'a'+Long2Str(A));
 end;
```

```
procedure SetRobotIntonacio(I : Byte);
(* A kimondás intonációját állítja be. *)
  begin
    if not (I in [0..1]) then I := DefaultIntonacio;
    RobotKimond(EscStr+'i'+Long2Str(I));
  end;

procedure SetRobot Tagolas(X : Byte);
(* A kimondás tagolását  állítja be. *)
  begin
    if not (X in [0..6]) then X := DefaultTagolas;
    RobotKimond(EscStr+'x'+Long2Str(X));
  end;

procedure SetRobotHanghosszusag(T : Byte);
(* A kimondás hanghosszúságát állítja be. *)
  begin
    if not (T in [0..3]) then T := DefaultHanghosszusag;
    RobotKimond(EscStr+'t'+Long2Str(T));
  end;

procedure SetRobotOpcioRec(Rec :
RobotOpcioRecTipus);
(* A megkapott robot opció rekord alapján beállítja az
                                        opciókat. *)
  begin
    with Rec do
      begin
        SetRobotSebesseg(Sebesseg);
        SetRobotHangmagassag(Hangmagassag);
        SetRobotHangtonus(Hangtonus);
        SetRobotSuttogas(Suttogas);
        SetRobotHangossag(Hangossag);
        SetRobotIntonacio(Intonacio);
        SetRobotTagolas(Tagolas);
        SetRobotHanghosszusag(Hanghosszusag);
      end;
  end;

function WriteRobotOpcioRec : Boolean;
          (* Az aktuális robot opció rekordot menti le. *)
  var
    F : File of RobotOpcioRecTipus;
    I : Integer;
  begin
    WriteRobotOpcioRec := false;
    Assign(F, RobotFName);
    Rewrite(F);
    Write(F, RobotOpcioRec);
    Close(F);
    I := IOResult;
    If I <> 0 then
      begin
        ErrorMessage('Write hiba <RobotOpcioRec> :
'+Long2Str(I));
        exit;
      end;
    WriteRobotOpcioRec := true;
  end;

function ReadRobotOpcioRec : Boolean;
(* Az aktuális robot opció rekordot olvassa fel, vagy
          feltölti az alap,értelmezett adatokkal. *)
  var
    F : File of RobotOpcioRecTipus;
    I : Integer;
  begin
    ReadRobotOpcioRec := false;
    Assign(F, RobotFName);
    Reset(F);
    I := IOResult;
    case I of
      0 : begin (* Beolvassa a rekordot *)
            Read(F, RobotOpcioRec);
            Close(F);
            I := IOResult;
            if I <> 0
              then begin
                ErrorMessage('Read hiba
<RobotOpcioRec> : '+Long2Str(I));
                exit;
              end;
          end;
      2 : begin (* Feltölti a rekordot *)
            FillChar(RobotOpcioRec,
SizeOf(RobotOpcioRecTipus), 0);
            with RobotOpcioRec do
              begin
                Sebesseg := DefaultSebesseg;
                Hangmagassag := DefaultHangmagassag;
                Hangtonus := DefaultHangtonus;
                Suttogas := DefaultSuttogas;
                Hangossag := DefaultHangossag;
                Intonacio := DefaultIntonacio;
                Tagolas := DefaultTagolas;
                Hanghosszusag := DefaultHanghosszusag;
              end;
            if not WriteRobotOpcioRec then exit;
          end;
      else begin
        ErrorMessage('Read hiba <RobotOpcioRec> :
'+Long2Str(I));
        exit;
      end;
    end;
    ReadRobotOpcioRec := true;
  end;

procedure RobotInit;
          (* A unit az első meghívásakor teszteli,
          hogy a rendszer inicializálva van-e
          és felolvassa, majd beállítja az opciókat *)
  var
    Rec : RobotOpcioRecTipus;
  begin
    RobotInited := RobotInitedProc;
    if not RobotInited then exit;
    if not ReadRobotOpcioRec then exit;
    RobotInited := RobotOpcioRec.Hasznalatban;
    if not RobotInited then exit;
    SetRobotOpcioRec(RobotOpcioRec);
  end;

begin
  RobotInit;
end.
```

2. Lista

A kipróbáláshoz egy egyszerű példa: Demo.Pas

```
program Demo;

uses
  RobotDrv;

procedure Kimond(S : String);
  begin
    RobotKimond(S+'.');
  end;

begin
  WriteLn('PC Robot');
  Kimond('PC Robot');
  WriteLn('Százhúszezerötszáz');
  Kimond('120500');
end.
```

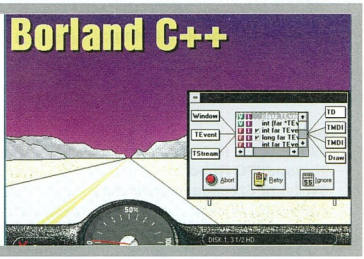


387EMU A lap 2.-3. oldalán lévő cikkhez forrás-file.

ANI Bővebben a lap 20. oldalán.

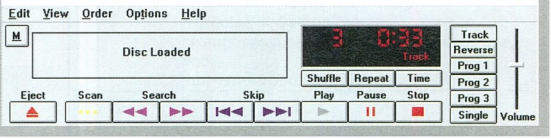
BMPINFO

A BMP file-formátum olvasása és megjelenítése egy C nyelvű programmal.



CDMUSIC

Ajándék CD lejátszó Windows alá tömörítve.

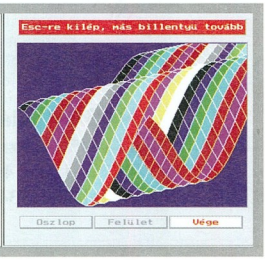


COMPFAIR

Próbaszámunk legjobb témái forrásnyelven.

GOMBMENU

Saját felhasználói felület kialakításához, grafikus gombmenü két példával.



HUBERT Az igazi izgalom, játékprogram írás assemblerben. A kész, futtatható program és a cikk forráslistái.

PASCAL A PASCAL iskola forrásai.

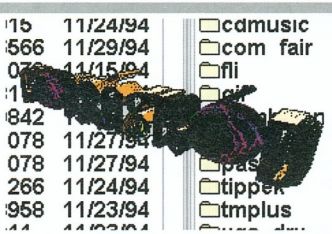
TIPPEK dBASE fejléc olvasása.

TMPLUS Bővebben a lap 21. oldalán.

UNIX Shell programok a cikk alapján.

WIN_ANI

Transzparens animáció a Windows képernyőn.



WIN_ORA Windows iskolánk első forrásai.

BASIC

Keretező

Mit ér egy kép keret nélkül ? Egy DEMO programban néhány példát mutatunk a különféle keretezési eljárásokra BASIC nyelven.

```
1000 KEY OFF: SCREEN 2: CLS
1030 LINE (0,0) - (639,199),,B
1040 LOCATE 11,15
1050 PRINT STRING$(20,219)SPACE$(2)"1.Keret"SPACE$(2)STRING$(20,219)
1060 LOCATE 15,32
1070 PRINT "Üssön le billentyűt!"
1080 WHILE INKEY$="": WEND
1100 CLS
1110 LINE (0,0) - (639,199),,B: LINE (2,2) -(637,197),,B
1120 LOCATE 11,15
1130 PRINT STRING$(20,219)SPACE$(2)"2. Keret"SPACE$(2)STRING$(20,219)
1140 LOCATE 15,32
1150 PRINT "Üssön le billentyűt!"
1160 WHILE INKEY$="": WEND
1200 CLS
1210 LINE (0,0) - (639,199),,B: LINE (1,1) - (638,198),,B: LINE (2,2) -(637,197),,B
1220 LOCATE 11,15
1230 PRINT STRING$(20,219)SPACE$(2)"3. Keret"SPACE$(2)STRING$(20,219)
1240 LOCATE 15,32
1250 PRINT"Üssön le billentyűt!"
1260 WHILE INKEY$="": WEND
1300 CLS
1310 LOCATE 1,1
1320 PRINT STRING$(80,219);
1330 LINE (0,0) - (20,199),,BF: LINE (619,0) - (639,199),,BF
1340 LOCATE 25,1
1350 PRINT STRING$(80,219);
1360 LOCATE 11,15
1370 PRINT STRING$(20,219)SPACE$(2)"4. Keret"SPACE$(2)STRING$(20,219)
1380 LOCATE 15,32
1390 PRINT "Üssön le billentyűt!"
1395 WHILE INKEY$="": WEND
1400 CLS
1410 FOR X = 199 TO 0 STEP-2
1420 LINE (0+X, 0+X) - (639-X, 199-X),,B
1425 NEXT X
1430 LOCATE 11,36
1440 PRINT "5. Keret "
1450 LOCATE 15,32
1460 PRINT "Üssön le billentyűt!"
1470 WHILE INKEY$="": WEND
1500 CLS
1510 FOR X = 16 TO 0 STEP-1
1520 LINE (0+X, 0+X) - (639-X, 199-X),,B
1530 NEXT X
1540 LOCATE 11,21
1550 PRINT STRING$(15,219)SPACE$(2)"6. Keret"SPACE$(2)STRING$(15,219)
1560 LOCATE 15,32
1570 PRINT "Üssön le billentyűt!"
1580 WHILE INKEY$="": WEND
1600 CLS
1610 FOR X = 57 TO 0 STEP -20
1620 LINE (0+X,0+X) - (639-X, 199-X),,B
1630 NEXT X
1640 LOCATE 11,32
1650 PRINT STRING$(10,219)SPACE$(2)"7. Keret"SPACE$(2)STRING$(10,219)
1660 LOCATE 15,32
1670 PRINT "Üssön le billentyűt!"
1680 WHILE INKEY$="": WEND
1700 CLS
1710 FOR X=0 TO 199 STEP 7
1720 LINE (0+X,0+X) - (639-X, 199-X),,B
1730 NEXT X
1740 LOCATE 11,36
1750 PRINT "8. Keret "
1760 LOCATE 15,32
1770 PRINT "Üssön le billentyűt!"
1780 WHILE INKEY$="": WEND
1800 CLS
1810 FOR X = 0 TO 20 STEP 5
1820 LINE (0+X,0+X)  - (639-X,199-X),1,B
1830 NEXT X
1840 LOCATE 11,32
1850 PRINT STRING$(10,219)SPACE$(2)"9. Keret"SPACE$(2)STRING$(10,219)
1860 LOCATE 15,32
1870 PRINT "Üssön le billentyűt!"
1880 WHILE INKEY$="": WEND
1900 CLS
1910 FOR L = 20 TO 40 STEP 5
1920 LINE (0+L,0+L) - (639-L, 199-L),,B
1930 NEXT L
1940 FOR X = 0 TO 20 STEP 1
1950 LINE (0+X,0+X) - (639-X, 199-X),,B
1960 NEXT X
1970 LOCATE 11,32
1975 PRINT STRING$(10,219)SPACE$(2)"10. Keret"SPACE$(2)STRING$(10,219)
1980 LOCATE 15,32
1985 PRINT "Üssön le billentyűt!"
1990 WHILE INKEY$="": WEND
2000 CLS
2005 FOR W = 15 TO 0 STEP -1
2010 LINE (0+W,0+W) - (639-W,199-W),,B
2015 NEXT W
2020 FOR X = 29 TO 0 STEP -3
2025 LINE (0+X,0+X) - (639-X, 199-X),,B
2030 NEXT X
2040 FOR Y =  59 TO 0 STEP -5
2045 LINE (0+Y,0+Y) - (639-Y, 199-Y),,B
2050 NEXT Y
2060 LOCATE 11,36
2065 PRINT STRING$(10,219)SPACE$(2)"11. Keret"SPACE$(2)STRING$(10,219)
2070 LOCATE 15,32
2080 PRINT "Üssön le billentyűt!"
2090 WHILE INKEY$="": WEND
3000 CLS
3010 LOCATE 1,1
3015 PRINT STRING$(160,176);
3020 LOCATE 23,1
3030 PRINT STRING$(160,176);
3040 FOR X = 10 TO 629 STEP 640
3050 LINE (10+X,10+X) -(629-X, 181-X),,B: LINE(20+X,20+X)-(619-X,171-X),,B
3060 NEXT X
3070 LOCATE 11,26
3075 PRINT STRING$(10,219)SPACE$(2)"12. Keret
"SPACE$(2)STRING$(10,219)
3080 LOCATE 15,32
3085 PRINT "Üssön le billentyűt!"
3090 WHILE INKEY$="": WEND
3100 CLS
3110 FOR X = 100 TO 600 STEP 2.5
3120 LINE (0+X,0+X) - (639-X,199-X),,B
3130 NEXT X
3140 LOCATE 11,35
3150 PRINT "13. Keret "
3160 LOCATE 17,32
3170 PRINT "Üssön le billentyűt!"
3180 WHILE INKEY$="": WEND
3200 CLS
3210 FOR X=0 TO 40 STEP 1.5
3220 LINE (0+X,0+X) - (639-X, 199-X),,B
3230 NEXT X
3440 LOCATE 11,35
3445 PRINT "Ez a 14. és nincs tovább "
3450 LOCATE 17,32
3455 PRINT "Üssön le billentyűt a kilépéshez!"
3460 WHILE INKEY$="": WEND
3470 CLS:SYSTEM
```

Sirató

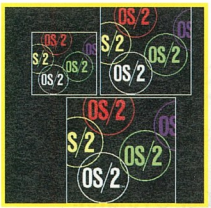
Amiga

Érdekes hangvételű írást kaptunk a lap összeállítása során Cs. Gyulától, az egykor nem minden alap nélkül favorizált, csodált géptípus kimúlásáról. Közben valahogy elveszítettük egymást, kérem ha olvassa a cikkét, keressen meg bennünket.

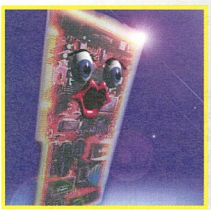
Mi fán terem a M68000-es mikroprocesszor?
Ez bizony nem Chio, hanem Motorola chip. Reklámozni sem lehet úgy, hogy kidugja egy kéz a TV képernyőjén, mire egy pasi vigyorogva bólogat, valamint nem ropogós és friss, beleharapni sem célszerű. Ennek a ronda fekete százlábúnak a fejlesztését 1977-ben kezdték el, de nem azért mert egyre kínosabbá vált az a felismerés, hogy a 8 bites processzorokat 64 Kb-nál több memória esetén körülményes alkalmazni, hanem mert e cikk írója éppen akkor lett 10 éves. Igen hízelgő a Motorola cégtől, hogy ezt a jelentős eseményt figyelembe vették. Akinek kicsit réginek tűnik a 1977-es dátum, annak igaza van, nem éppen tegnap dobták a piacra a M68000-est. Vegyük viszont figyelembe, hogy akkor kezdtek bele a fejlesztésbe, amikor jószerint csak az volt az érdekes, hogy a processzorok teljesítsék feladatukat és azt hitték a legjobbnak, amelyik a legtöbb utasítást megértette. A meglevő tárkapacitás azonban egyre inkább szűkösnek bizonyult és egy-egy hatékonyabb program esetében már nemcsak az számított, hogy elfér-e a memóriában, hanem elfogadható idő alatt lehet-e lefuttatni. E problémákat felismerve kezdett bele a Motorola egy (akkor) gyors 16 bites processzor fejlesztésébe. Természetesen a cég nem találta fel a spanyolviaszt, így jónéhány más processzornál már kedvelt és bevált tulajdonságot, javított formában, beépítettek a M68000-esbe, így például az Intel központi egységeinek számos funkcióit is átvették. A M68000-es felépítése családszerkezetű (mint a 80086), tehát a teljesítményt illetően léteznek új és újabb változatok, ezek felülről lefelé kompatibilisek (elvileg), de maradjunk az őskorban az M68000-nél. A félvezetők gyártástechnológiáját is fejleszteni kellett, mert egy ilyen összetett mikroprocesszor előállításához az áramkörsűrűség nem volt megfelelő. Ehhez a procihoz kb. 68000 tranzisztorra, illetve 13000 kapuműveletre van szükség, ami egy Pentium többmilliós nagyságrendjéhez viszonyítva nevetségesen alacsony, ám vegyük figyelembe, hogy ez a 68000-es család első generációja. Bármilyen hihetetlennek tűnik, a központi egységben is van ROM, ez tartalmazza a proci saját utasításkészletét, illetve a processzorba érkező gépi kódú utasítások mikroprogramjait. Ez úgy is felfogható mint processzor a processzoron belül. Az érthetőség kedvéért nézzünk meg egy rövid példát. A MULU utasítás azt jelenti, hogy a központi egységnek szoroznia kell. A programozót nem érdekli, hogy a proci hogyan bűvészkedik, számára az a lényeg hogy meglegyen a művelet. A beérkező utasítás a processzornak csak egy kód, méghozzá a megfelelő mikroprogram kódja, amit azonosítás után elindít, tehát egy makroutasítás sok mikroutasítás végrehajtását váltja ki. Mindezekből következik, hogy egy ilyen processzor utasításkészletének változtathatósága és bővíthetősége viszonylag egyszerű, ha van még némi szabad ROM terület egy-két újabb mikroprogram számára. A M68000-es kifejlesztésénél az egyik legfontosabb követelmény pedig a bővíthetőség volt. Ennek a processzornak mindössze 56 assembler utasítása van, ez más processzorokhoz képest kevés. A 14 címzésmód miatt, viszont több mint 1000 utasítása van. (Remélem senki sem szorozta meg az 56-ot 14-el.) Tulajdonképpen könnyen programozható ez a chip. Az Amiga 1000 a standard 8Mhz-es változatot kapta meg, először 1984-ben. Ennek teljesítménye akkor bőven elégséges volt, és hardware környezetével az akkori legjobb gépnek számított. 1986-ban az A500 az év gépe lett, mert sok olyan profi dolgot tartalmazott, amelyet más személyi számítógépek nem, pl: két grafikus co-proci is gubbasztott benne, valamint a palettája már RGB volt, vagyis a színek a piros-zöld-kék alapszínekből voltak kikeverhetők. Az már más kérdés, hogy 1991-92-ben (a személyi számítógépek körében elérhető áron) még mindig szinte ugyanazt a hardware-t dobta piacra a Commodore, amikor a PC világ már régen a még nagyobb teljesítmény még alacsonyabb ár vonalán nyomult. Először szinte mindenki lecserélte régi C64-esét Amiga 500-asra és dícsérte, hogy az milyen jó meg szép. Ma ha megkérdezek valakit, akinek 500-asa volt (és most 486-osa van) azt mondja: "ja igen, ...", pedig a gép azóta mit sem változott, és éppen ez a baj. Az újabb (kisebb kategóriájú és elérhető árú) Amigák nem igazán váltották be a hozzáfűzött reményeket, ahol szerintem két dolog volt a buktató: az első a még mindig nemmoduláris felépítés (nehéz és drága bővíthetőség), a második az alacsony teljesítmény (A500+, A600, A1200). Hát igen, a 7Mhz-es M68000, illetve a 14Mhz-es M680EC20 nem igazán illik bele a korszerű személyi számítógépről alkotott képbe. Mai szemmel nézve a jó öreg M68000 rém lassú, ami pedig igen jó eséllyel felvehetné a harcot a 486-os procikkal szemben, a 68040 és a 68060 pedig rém drága.

Cs. Gyula

Terveinkből:

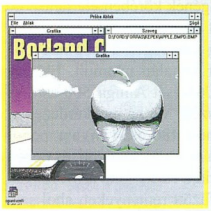


Megjelent az IBM operációs rendszerének új változata.
Vallatjuk: hogyan viselkedik mint fejlesztő környezet.
Vizsgáljuk a kompatibilítást és a sebességet.



Hangkártyák programozásáról írunk.
Nem maradnak ki sem a GRAVIS, sem a SOUNDBLASTER kártyák.

Ablakkezelés Dos alatt, TURBO C nyelven. Folytatjuk grafikus felhasználói felületünk építését az ablakkezelés elméletével és gyakorlatával.



Windows programozó sorozatunkban gyermekablakok létrehozását, kezelését, az MDI alkalmazását (Multiple Document Interface) ismertetjük következő számunkban.

Folytatódik assembler sorozatunk a játékprogram írás rejtelmeiről.

HIRDETŐINK:

AKTÍV REKORD
BECO KFT.
DÉMA KFT.
MTA SZTAKI
NASA REKLÁMIRODA
PIXEL GRAPHICS
PRESENT COMPUTER
STARKING ÓBUDA APPLE CENTER
SCANDER KFT.
TEX STUDIO KFT.

FORRÁSKÓD
A programozók lapja

Megjelenik havonta
3.5"-os mágneslemez melléklettel

Alapító főszerkesztő:
Nagy Sándor

Felelős kiadó:
Nagy Sándorné

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

ISSN : 1218-6414

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

Terjesztés:
Hírker Rt.
Nemzeti Hírlapkereskedelmi Rt.
Alternatív terjesztők

Hirdetésfelvétel:
Bauer Csilla
Kériné Bors Melinda
Tel/Fax : 134-1273

Előfizetés:

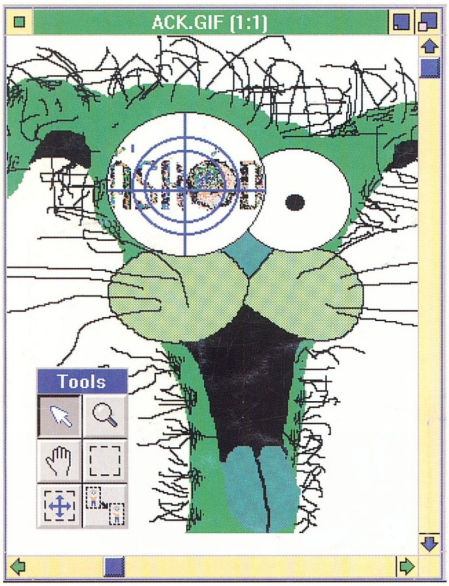
Fél évre:	1494.- Ft.
Egész évre:	2388.- Ft.
Iskoláknak:	1188.- Ft.

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

ÁRA : 298 Ft. 1. SZÁM 1995. JANUÁR

lemez

WINDOWS
Erőforrások



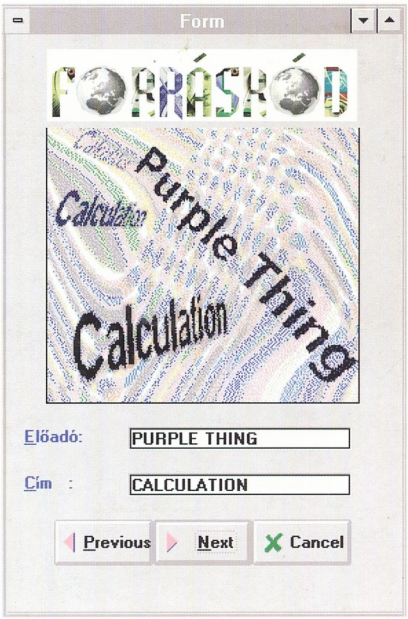
Assembler kezdőknek és haladóknak.

Írjunk Játékprogramot!!!

Grafikus felület C nyelven.

PASAL ISKOLA

dBASE for Windows



Forráskód a programozók lapja