III. évfolyam 3. szám * 1992. május — június * Ára: 49,- Ft

ENTERPRESS

KÉTHAVILAP AZ ENTERPRISE SZÁMÍTÓGÉPEK FELHASZNÁLÓINAK

Hol vagy Kiadó?

Sok olvasónk bizonytalankodik amiatt, hogy Kiadónknak szánt leveleit hova küldje, a Kiadót telefonon/faxon milyen számon keresse?

A Mátrix Kft. "lakcíme" a lap indulása óta háromszor változott: induláskor még a *Dózsa György tér 10.* volt, aztán *Honvéd utca 8.*-ra változott, most pedig *Zichy Liget 10.* lett a Kiadó székhelye.

Ebből nem szabad azt a következtetést levonni, hogy a Mátrix afféle vándortársulat lenne. Szó sincs erről! Állítólag rendszerváltás zajlott le a közelmúltban, és ennek egyik látszata az volt, hogy össze-vissza váltogatták az utcai névtáblákat a városokban, így Székesfehérváron is. A Mátrix esetében az is közrejátszott a címek változásában, hogy sarokházban laknak, két utcához (melyikhez?) tartoznak. Egyszóval a három cím ugyanazt a házat, ugyanazt helyiséget jelenti.

A telefon és fax-szám pedig mindenféle bonyolult, a szerkesztőség számára felfoghatatlan egyezkedések (veszekedések) során változott.

Tehát bármelyik címre írhatnak olvasóink, az mindenképpen eljut a Kiadóhoz. Célszerű azonban a legfrissebb (legutolsó) szám impresszumából kikeresni ezeket az adatokat.

A szerkesztők

TARTALOM 92/3

A külső is számít

(folytatás)

Komoly programozó már régen elfelejtette a karakteres képernyőt, és programjai grafikus képernyőn futnak. (Ezekre természetesen az előző részben leírt megállapítások továbbra is igazak.) De ne csapjunk a közepébe! Miért is lett oly népszerű a grafikus képernyő?

A számítógépek nagyrésze parancssoros (értsd: bepötyögős) módszerrel kommunikál a felhasználóval. S mindez nem elég, hiszen ahhoz, hogy a gépet használni tudjuk, a számunkra érdekes programot elindíthassuk mindenféle elveket, varázsigéket kell megtanulnunk. Olyan ez, mintha egy kocsit akarnánk elindítani, és indulás előtt mindig babrálnunk kellene a porlasztóval, akkumulátorral, gyujtáselosztóval stb.

Ugyanez vonatkozik a programokra is: a menüpontokban szereplő idegen (szak)szavak igencsak megizzasztják a felhasználót, akit meg kell kímélni minden felesleges munkától, mert mindent a kezébe kell adnunk!

Az emberhez sokkal közelebb áll, ha a program elindulásakor valamilyen grafikus képernyőn jelentkezik be, s az egyes funkciókat valamilyen jópofa, egyértelmű jelentéssel bíró kisméretű rajzocskák ún. *ikonok* szimbolizálják.

Mit jelent ez? Tegyük fel, hogy egy ilyen, grafikus felületen futó szövegszerkesztővel dolgozunk, s egy szót törölni akarunk: először kiválasztjuk az olló ikont (a szövegből kivágunk vele egy darabot), a kurzoral kijelöljük a szót, végül a szemeteskosár ikonnal eldobjuk. Az ilyen elvet még egy analfabéta is megérti. (Érdekes ember lehet az az analfabéta, aki szövegszerkesztővel dolgozik. *A szerk.*)

A grafikus felület megprogramozása kemény dolog, csak néhány feladat vele kapcsolatban:

- Gyors megjelenítés.
- Gyors keret, árnyék stb. rajzolás.
- Az ikonok és a hozzájuk tartozó feliratok kitétele.
- Több ablak együttes kezelése.
- Az ablakokban állandó editálási lehetőségek (törlés, beszúrás, felülírás, másolás, kivágás stb.).
- Az ablakok, ikonok, feliratok tetszőleges helyre mozgatása, törlése, módosítása.
- Gyors, esztétikus színezés.
- Az eltakart sajátosságok (pl. alsóbb ablak) "könyvelése". Grafikus ablakoknál a memóriaigény igen nagy. Ha egy ablak letakar egy másikat, akkor nem kell elmenteni az alatta levő (letakarásra kerülő) képernyőrészt, hanem könyvelni kell, hogy mi volt ott, s ha bezárjuk az ablakot, akkor alatta mindent újra kell rajzolni...
- Hatékony memóriakezelés. A tárban el kell férnie az ablak-, menükezelő modulnak, az alkalmazásnak és a változóknak.
- Biztosítani kell azt, hogy az ablakok különböző objektumok legyenek. Ha az egyikre írok, az a másikon ne jelenjen meg.
- A meglévő (nem ikonos) menük gyors megjelenítése.
- Biztosítani kell az ablakok mozgathatóságát, méretének változtathatóságát.
- Scrollozási lehetőség az ablakban.
- Stb.

Van még egy fontos követelmény: a programot elsősorban egérrel lehessen használni! Az egeret természetesen a háttérben működteti egy mindentől független, interruptos (!) kezelőprogram... Ez az Enterprise-on önmagában sem egyszerű feladat.

Minden komolyabb gépre létezik már ilyesmi, többjüknél maga az operációs rendszer ilyen: Geos-C64, GEM-Atari (többek között), System 7-Apple Machintosh, Windows-AT/386/486, NeXTStep-Next. Ezeknél az "ablakozós" lehetőség nem egy program(bővítés), hanem mindezeket (sok egyéb szuper jellemző mellett) az operációs rendszer alapban nyújtja. Egy ilyen grafika-orientált operációs rendszer kifejlesztése iszonyúan nagy munka: az IBM kompatíbilis gépeken futó Microsoft Windows-t már 7 éve fejleszti 30 ember.

Álmodik a nyomor (mondaná Ady Endre), nekünk ezeknél jóval-jóval szerényebb környezet is megfelelne! Lesz valaha ilyenünk?

ASSEMBLY

11. rész

Miután megismertük a VIDEO: eszköz lehetőségeit, a karakteres és a grafikus képernyők szervezését, az escape szekvenciák használatát éppen ideje, hogy tovább haladjunk. Irány a fájlkezelés!

Egyszerű, mint a pofon

Véleményem szerint a videoval kapcsolatos ismeretek a legmacerásabbak az EXOS-nál. A többi eszköz és lehetőség (fájl-, hang-, sorosvonali kezelés stb.) lényegesen egyszerűbb. (Természetesen ezek nem is olyan sokoldalúak, mint a képkezelés.)

Ezt támasztja alá az *1.lista*. A fájlkezelésnél megnyitnunk egy csatornát, elvégezzük karakterenként ill. blokkonként az írást/olvasást, majd lezárjuk. Ennyi.

```
 1          org 01000h
 2
 3  fchn    equ 1
 4  txtlen  equ 19
 5
 6          ld a,fchn
 7          ld de,fname
 8          exos 2          ;iras
 9
10          ld a,fchn
11          ld bc,wrlen
12          ld de,txt
13          exos 8
14
15          ld a,fchn
16          exos 4
17
18          ;*******
19
20          ld a,fchn
21          ld de,fname
22          exos 1          ;olvasas
23
24          ld a,fchn
25          ld bc,txtlen+0  ;>0 EOF
26          ld de,buff
27          exos 6
28
29          ld a,fchn
30          exos 3
31
32          ret
33
34  txt     defm "A kimentendö blokk!"
35  wrlen   equ $-txt
36
37  buff    defs 200
38
39  fname   defb lenwr-1
40          defm "TEST.DTA"
41  lenwr   equ $-fname
42
43          end
```

1. lista

Az EXOS funkcióhívások különbözőek írásnál-olvasásnál, de paraméterezésük azonos. A funkciókat foglalja össze a *táblázat*:

	Nyitás	Karakter	Blokk	Lezárás
Olvasás	EXOS 1	EXOS 5	EXOS 6	EXOS 3
Írás	EXOS 2	EXOS 7	EXOS 8	EXOS 4

A program kiír (eszköznév hiányában) magnóra ill. lemezre egy szöveget, majd ezt visszaolvassa. Először megnyitjuk írásra a csatornát (6-8). A-ban a csatornaszám van, DE pedig a fájlnévre mutat. A szöveget blokként írjuk ki (10-13): BC a kiírandó bájtok számát tartalmazza, DE pedig a szöveg elejére mutat. A blokkírás végeztével lezárjuk a csatornát (15-16).

Az olvasás nem sok újat tartogat. A megnyitás ugyanolyan logikájú, mint ahogy fent láttuk. Beolvasásnál (24-27) DE arra a memóriahelyre mutat, ahová a szöveget (pontosabban a bájtokat) tárolni akarjuk. Bezárul a sor a lezárással (29-30).

A *2.listán* látható program ugyanazt az eredményt adja, mint az első. A különbség abban van, hogy ez utóbbi karakterenként (bájtonként) ír/olvas. A megnyitási és a lezárási részek nem különböznek az előzőektől.

```
 1          org 01000h
 2
 3  fchn    equ 1
 4  txtlen  equ 19
 5
 6          ld a,fchn
 7          ld de,fname
 8          exos 2          ;iras
 9
10          ld hl,txt
11          ld b,wrlen
12  wrloop  push bc
13          ld a,fchn
14          ld b,(hl)
15          exos 7
16          inc hl
17          pop bc
18          djnz wrloop
19
20          ld a,fchn
21          exos 4
22
23          ;*******
24
25          ld a,fchn
26          ld de,fname
27          exos 1          ;olvasas
28
29          ld hl,buff
30  rdloop  ld a,fchn
31          exos 5
32          cp 0
33          jr nz,rdend
34          ld (hl),b
35          inc hl
36          jr rdloop
37
38  rdend   ld a,fchn
39          exos 3
40
41          ret
42
43  txt     defm "A kimentendo blokk!"
44  wrlen   equ $-txt
45  buff    defs 200
46  fname   defb lenwr-1
47          defm "WRITE.TXT"
48
49  lenwr   equ $-fname
50
51          end
```

2. lista

A karakterenkénti kiírás egyszerű. HL a szövegben a kiírandó karaktert címzi. Ez "kezdéskor" nyilván a szöveg eleje (10). DJNZ-s ciklusban írunk, ehhez B-be kell tölteni a szöveg hosszát (11). A ciklus elején (12) B-t a stack-re tesszük, majd A-ba a csatornaszámot töltjük (13). Tudjuk, hogy EXOS 7-nél B tartalmazza a karaktert (14), ez nyilván a HL által mutatott tárrekesz tartalma. Jöhet az írás (15). Megcímezzük a következő karaktert (16), levesszük a veremről a ciklusszámláló értékét (17), és jöhet - ha kell - az újabb karakter kiírása.

Olvasás előtt HL arra a tárrészre mutat, ahová a szöveget letároljuk (29). Az olvasó ciklus elején A-ba a csatornaszámot töltjük. De álljunk csak meg! Hol marad a ciklusszámláló?

Biztosan mindenkiben felmerült már a kérdés: Mi történik akkor, ha több adatot akarunk a fájlból beolvasni annál, mint ami egyáltalán van benne? Az EXOS ránk szól. 0E4h kódú hibajelzést küld, ami EOF-ot (*End Of File, a fájl vége*) jelent. Ha ilyet kaptunk, nincs értelme tovább olvasni a fájlt.

Ezt használja ki az *rdloop* ciklus is. A beolvasás (31) után megvizsgáljuk, hogy az akkumulátor tartalma egyenlő-e nullával (32)? Ha igen, akkor folytathatjuk a beolvasást. HL címre letároljuk a karaktert (34), növeljük HL-t (35) a következő pozícióhoz, és folytatódhat a beolvasás.

Ha a funkcióhívás nem nulla akkumulátor-tartalommal tér vissza,

akkor a fájl végéhez értünk. Ekkor ki kell ugrani a ciklusból (33). (Megjegyzés: A program nagyvonalúan bánik a vizsgálattal, hiszen nem az EOF kód beérkezését figyeli, hanem azt, hogy van-e valami hiba? Így leáll akkor is, ha történetesen lenyomjuk a Stop gombot. Ennél a példánál persze az EOF bekövetkezése a legnagyobb valószínűségű hiba, de komolyabb rutinokban ezt szebben kell megoldani. Azon pedig lehet filozofálni, hogy az EOF egyáltalán (általános eszköz-) hiba-e? Szerencsésebb talán *figyelmeztetésnek* nevezni.)

Az első valami

A *3.listán* az első olyan munkánk látható, amely használható is valamire: ún. *file-dump*-ra. A program beolvas egy fájlt és a tartalmát soronként az alábbi formátumban jeleníti meg:

XXXX YY YY YY YY YY YY YY YY ZZZZZZZZ, ahol

- *XXXX* 16 bites hexadecimális szám, a fájlon belüli karakter pozíció (offszet).
- *YY* a karakter hexadecimális kódja.
- Z maga a (megjeleníthető) karakter.

Lehet gondolkodni a működésén! Magyarázat a következő részben.

-HCs-

```
;*******************************************

        FILEDUMP.ASM

;*******************************************

        org 01100h

fchn    equ 1           ;innen olvasunk
asmscr  equ 070h        ;ASMON VIDEO:
chr     equ 8           ;chr szelesseg
x       equ 30          ;AT 0,X...

;*** a kiserletezes idejere...
        ld a,fchn
        exos 3          ;lezarja a csatornat
;***
        ld a,fchn
        ld de,fname
        exos 1          ;iras
        ret nz          ;hiba->vege

        ld hl,0         ;offszet szamlalo
        ld (cnt),hl     ;letoltese

rdloop  ld a,fchn
        ld bc,chr
        ld de,buff
        exos 6          ;blokk olvasasa

        ld a,chr        ;max. karakterszam
        sub c           ;C-ben a hatralevo
        jp z,total      ;semmit sem olvasott!
        ld b,a          ;olvasott karakterek szama

        push bc         ;HEX-resznek
        push bc         ;karakter-resznek

;az offszet kiirasa HEX formaban
        ld hl,cnt+1     ;HIGH resz!
        ld ix,table     ;HEX szamok
        ld b,2          ;2*2 hex szam
cycl1   push bc
        xor a           ;rld-nek
        ld b,2          ;ket szam
cycl2   push bc
        rld
        ld (dis+2),a    ;trukk
        push af         ;HL-t vedi!
dis     ld b,(ix+0)     ;hex szam megfeleloje
        ld a,asmscr
        exos 7          ;kiirva
        pop af
        pop bc
        djnz cycl2
        rld             ;eredeti HL!
        dec hl
        pop bc
        djnz cycl1

;az offszet kovetkezo erteke (+chr)
        ld hl,(cnt)
        ld bc,chr
        add hl,bc
        ld (cnt),hl

;a "hatarolojel" kitetele
        ld a,asmscr
        ld b,":"        ;kettospont hatarol
        exos 7

;HEX szamok kirakasa
        pop bc          ;a "szamok szama"
        xor a           ;b7-b4 nulla legyen
        ld hl,buff      ;a buffer
loop0   push bc         ;ennyi hex szamot irunk
        ld b,2          ;a hex szam hossza
loop1   push bc         ;a hossz-szamlalo
        rld             ;rotalas balra
        ld (direct+2),a ;eltolas trukk
        push af         ;masolat A-rol
direct  ld b,(ix+0)     ;beolvasas
        ld a,asmscr     ;kepernyo csat.
        exos 7          ;iras
        pop af          ;A vissza
        pop bc          ;hossz vissza
        djnz loop1      ;ciklus
        rld             ;eredeti allapot (HL)
        ld a,asmscr     ;kepernyo csat.
        ld b," "        ;space kozejuk
        exos 7          ;iras
        inc hl          ;uj szam
        pop bc          ;van meg szam?
        djnz loop0      ;igen/nem

;ugras a kurzorral a karakteres reszhez
        ld a,asmscr
        ld bc,esclen
        ld de,pos
        exos 8

;a karakterek kiirasa
        pop bc          ;a karakterek szama
        ld hl,buff      ;puffercim
loop3   push bc
        ld a,(hl)       ;maga a karakter
        cp " "          ;nyomtathato?
        jr nc,cont      ;igen
        ld a,"."        ;nem, "." lesz
cont    ld b,a          ;a kesz karakter
        ld a,asmscr
        exos 7
        inc hl
        pop bc
        djnz loop3

;uj sor elejere ugrik a kurzor
        ld a,asmscr
        ld bc,2
        ld de,crlf
        exos 8

;johet a kovetkezo sor
        jp rdloop

total   ld a,fchn       ;befejezes
        exos 3
        ret

;filenev es nevhossz merese
fname   db fnlen-1
        defm "FILENAME.EXT"
fnlen   equ $-fname

;ide jonnek a szamok
buff    defs 256

;soremeles
crlf    defb 13,10

;PRINT AT 0,X...
pos     defb 27,"=",32,32+x
esclen  equ $-pos

;HEX szamok kodjai
table   defm "0123456789ABCDEF"

;offszet szamlalo
cnt     defs 2
        end
```

2. lista

A Pascal

11. rész

A Turbo-Pascal eljárásai és függvényei (2.)

Továbbra is maradunk a jó öreg Turbo-Pascal mellett. Aki csak a HiSoft Pascal szolgáltatásaihoz fér hozzá, most kicsit mellőzöttnek érezheti magát; később igyekszünk majd pótolni a mulasztást és kárpótolni őket.

A képernyőkezelés; szöveges üzemmód

A "szabványos" Turbo-Pascal TEXTMODE eljárása, amely a képernyő szöveges üzzemmódra állítását lenne hivatott elvégezni, az ENTERPRISE gépen nincs megvalósítva. Ennek oka igen hétköznapi: a nyelv grafikus elemei itt nincsenek megvalósítva, így nincs is grafikus üzemmód, amelyről át lehetne kapcsolni szövegesre. Emiatt ennek a fejezetnek sincs meg a párja, amelynek címe *A képernyőkezelés; grafikus üzemmód* lenne...

Ha azonban a szöveges üzemmódot használjuk (ez ugye gyakorlatilag elkerülhetetlen), hasznos és szükséges segítség a képernyőtörlés, amit a Turbo-Pascalban a CLRSCR eljárás valósít meg. Az eljárás végrehajtása során a kurzor szokás szerint a bal felső sarokba kerül, ennek a pozíciónak a koordinátái (1, 1).

Nem foglalkozunk a terminál installálásával kapcsolatos CRTINIT és CRTEXIT eljárással, akit ez érdekel, utánanézhet a kézikönyvben.

A képernyő törlése után a következő legfontosabb funkció a képernyő címzése. A Turbo-Pascal szabványos eljárása erre a feladatra a GOTOXY névre hallgat, szintakszisa

```
GOTOXY( oszlop, sor )
```

Ez egy parányit megzavarhatja az IS-Basic PRINT AT utasításának - talán kicsit logikusabbnak tűnő - *sor, oszlop* sorrendű paramétermegadásához szokott felhasználót. Ez a fordított sorrend talán a grafikus üzzemmód paraméterezésével való párhuzam miatt jött létre, a grafikus eljárásoknál ugyanis *x, y* a koordináták sorrendje, a koordinátageometria pedig vízszintesen az *x*, függőlegesen az *y* koordinátát szereti felrajzolni. Aki nem tud kibékülni ezzel a sorrenddel, az nyugodtan definiálhat magának egy áthidaló eljárást, amit azután minden programjába beszerkeszthet:

```
PROCEDURE AT( SOR, OSZLOP : INTEGER );
    BEGIN
      GOTOXY( OSZLOP, SOR );
    END { AT };
```

Ha tehát a képernyő közepére szeretnénk kiírni egy üzenetet, akkor azt a következő kis eljárással ill. hívással tehetjük meg (a STRING80 típust a főprogrambankell definiálni, célszerűen 80 elemű stringként):

```
PROCEDURE KOZEPRE( SOR : INTEGER;
                   SZOVEG : STRING80 );

BEGIN
    GOTOXY( ( 80 - LENGTH( SZOVEG ) ) DIV 2, SOR );
    WRITE( SZOVEG );
END { KOZEPRE };
    ...
KOZEPRE( 12, 'itt a világ közepe!' );
```

Ha nem tudjuk, hova is tettük a kurzort, megtudhatjuk a pozícióját a WHEREX és a WHEREY függvény meghívásával. Nem ér senkit meglepetésként, hogy az első függvény a kurzorpozíció oszlopkoordinátáját, a második függvény a kurzor sorkoordinátáját adja meg, mindkettő értelemszerűen egész érték. Ennek megfelelően a

```
GOTOXY( WHEREX, WHEREY );
```

programrészlet ugyanoda teszi a kurzort, ahol az eddig is volt, azaz nem csinál semmit.

A CLREOL eljárás (ennek a nevét már segítünk megfejteni: clear to end of line) a kurzor sorának a jobboldali végét törli a kurzor pozíciójától a sor végéig. Akkor lehet hasznos, ha a képernyő adott pozíciójára írunk, és nem szeretnénk, ha az előzőleg kiírt szöveg hosszúra sikeredett vége ottmaradna, ugyanakkor nem akarjuk újraírni a szöveg elejét. A sorrend mindegy: pozícionálunk, törlünk, írunk, vagy pedig pozícionálunk, írunk, törlünk (a többi kombináció nem nyer!). Készülő szövegszerkesztő programunk képernyő-újraíró eljárása nem nélkülözheti ezt az eljárást.

Ha az egész sort akarjuk törölni, a DELLINE eljárást érdemes használni. Ez meg is szünteti a sort, azaz a képernyő további sorait eggyel feljebb ugrasztja. Ha csak a sor *tartalmát* akarjuk törölni, de a helyét meg akarjuk tartani, használjuk a CLREOL eljárást, miután a kurzort a sor elejére pozícionáltuk. (Ha nem tudjuk, melyik sorban vagyunk, használjuk a WHEREY függvényt!)

A DELLINE eljárás párja az INSLINE, ez beszúr egy üres sort, ehhez a kurzor sorát és az alatta levő sorokat lejjeb tolja.

Ha kedvünk van, játszhatunk a szöveg színével, persze, csak a megengedett lehetőségeken belül. A LOWVIDEO eljárás a *gyenge* szövegtónust kapcsolja be, a NORMVIDEO eljárás a *normális* tónust, a HIGHVIDEO eljárás pedig az *erős* tónust. A normális tónus használható a mezei kiírásra, az erős tónus a szöveg kiemelésére. A gyenge tónus jelölheti például egy menürendszerben az adott pillanatban valamilyen okból éppen nem használható menüpontokat.

A három eljárás által ténylegesen beállított szín természetesen az adott rendszertől függ. Az ENTERPRISE esetében mindez az IS-DOS által beállított palettának megfelelően fog működni. Sajnos, az IS-DOS elég "keményen" osztja ki a színeket, nehéz rajtuk változtatni. Feketefehér képernyőn, ha a monitorkimenetet használjuk, nincs különbség a háromféle kiírás között. Ez elég baj, és persze nem a Turbo-Pascal hibája (ugyanez a beépített hardverhiba okozza, hogy a kurzor és a szöveg azonos tónusú a fekete-fehér képernyőn). Egy kis szoftveres barkácsolással megoldhatjuk a problémát, sőt, az INVVIDEO eljárást bevezetve inverz (világos alapon sötét) karaktereket is kiírhatunk.

Ha sikerül felélesztenünk az inverz kiírást, kiíráskor ügyeljünk arra, hogy még a WRITELN utasítás kiadása előtt állítsuk vissza a normális üzemmódot, különben a sor üresen maradó vége is inverzre változik!

A billentyűzet kezelése

A képernyő után foglalkozzunk egy kicsit a billentyűzettel is! A billentyűzetet kétféle módon szokás kezelni. Az egyik, amikor bevitelre várunk, és mindaddig nem megyünk tovább, amíg az meg nem történik. A READ és a READLN eljárás így működik.

Sokszor azonban a programnak folyamatosan mennie kell, és csak akkor kell valami eltérőt cselekedni, ha a felhasználó hozzányúlt a billentyűzethez. Ez lehet egy játék során, de pl. folyamatszabályozásnál, szimulációnál és még sok más esetben. Ehhez nyújt segítséget a KEYPRESSED függvény, amely logikai (boolean) típusú. A függvény mindaddig FALSE, azaz hamis értéket ad vissza, amíg a billentyűzethez nem nyúlunk, és TRUE, azaz igaz értéket ad, amikor egy billentyűt lenyomunk. A

```
REPEAT UNTIL KEYPRESSED;
```

programrészlet mindaddig vár, amíg egy tetszőleges billentyűt le nem ütünk. Kis kiegészítéssel felhasználhatjuk reflexeink gyorsaságának mérésére, vagy pedig egy véletlenszám-sorozat kezdőértékezésére:

```
S := 0;
REPEAT
  S := S + 1;
UNTIL KEYPRESSED;
```

Természetesen meg is tudhatjuk, hogy melyik billentyűről van szó:

```
REPEAT
{ itt folyamatosan történik valami }
    IF KEYPRESSED THEN BEGIN
      READ( KBD, C );
      CASE C OF
      { itt sokfelé ágazunk }
      END { CASE };
    END { IF };
UNTIL VEGE;
```

Mivel a KEYPRESSED függvény konfliktusba kerül az operációs rendszer billentyűzetkezelő szolgáltatásaival, használata esetén a C fordítási opciónak kikapcsolt állapotban kell lennie. Az opciókkal majd később foglalkozunk részletesen, most csak annyit előzetesen, hogy a programban valahol helyezzünk el egy

```
{$C-}
```

megjegyzést (így, ahogy van, szóközök nélkül). Vigyázzunk, mert ilyenkor az elindított programot nem tudjuk leállítani sem a STOP gombbal, sem pedig a CTRL-C billentyűkombinációval! Ha viszont a programot a RESET gombbal állítjuk meg, kikerülünk az IS-DOS szintjére, így a programunk elvész, ha nem mentettük el. Futtatás előtt mindig mentsünk!

Egyéb matematikai függvények

Mint korábban láttuk, a szabványos Pascal a matematikai függvények közül csak viszonylag keveset valósít meg. A hiányzó függvények a megfelelő matematikai összefüggések ismeretében a meglévőkből előállíthatók. A Turbo-Pascal némiképpen bővíti a megvalósított matematikai függvények körét. Az ismertetésnél a sorozat 9. részében bevezetett jelöléseket alkalmazzuk.

A FRAC(x) függvény, amelynek paramétere egész vagy valós lehet, valós eredményt ad, mégpedig a paraméter tört részét. Hasonlóan, az INT(x) függvény az egész vagy valós paraméter egész részét adja vissza, de valós típusúként! Nem használhatjuk tehát valós-egész típuskonverzióra (erre a célra a ROUND függvény alkalmas). Azt gondolhatnánk, hogy az INT függvény "házilag" is előállítható. Gondolatmenetünk mindaddig helyes, amíg a valós szám nem haladja meg az ábrázolható legnagyobb (negatív szám esetén legkisebb) egész számot, akkor ugyanis az eljárásunk már nem működik.

Ha dobókockát szimulálunk, molekulák ütközését vagy légörvények hatását modellezzük, elengedhetetlen a *pszeudo-véletlenszámok* generálása. A RANDOM függvény, ha *paraméter nélküli* változatát hívjuk

meg, egy *valós* véletlenszámot ad vissza a 0 ... 1 intervallumban. Az intervallum elölről zárt, hátulról nyitott; ez azt jelenti, hogy a 0 része az intervallumnak, az 1 viszont nem. Nullát adhat vissza a függvény, 1-et azonban nem, csak olyan számot, amelyik tetszőlegesen megközelíti 1-et, de el nem éri. Ha a függvény *paraméteres változatát* hívjuk meg, amely RANDOM(i) alakú, akkor az egy *egész* típusú véletlenszámot ad vissza a 0 ... i intervallumban; az intervallum most is elölről zárt, hátulról nyitott. Egész számok esetén ez átfogalmazható: a függvény a 0 ... i - 1 tartományban ad eredményt.

A kockadobás szimulálása tehát így nézhet ki:

```
WRITELN( RANDOM( 6 ) + 1 );
```

(a RANDOM(6) hívás a 0 ... 5 számsorozatból ad meg egy számot, az 1 hozzáadásával ezt az 1 ... 6 intervallumra korrigáljuk). Ha a képernyő véletlenszerűen kiválasztott pozíciójába akarunk egy csillagot kírni, akkor ezt így tehetjük meg:

```
GOTOXY( RANDOM( 80 ) + 1, RANDOM( 24 ) + 1 );
WRITE( '*' );
```

Ha a csillag helyett egy véletlenszerűen kiválasztott *nagybetűt* akarunk kírni, akkor a megoldás ez lehet:

```
WRITE( CHR( RANDOM( ORD( 'Z' ) - ORD( 'A' ) + 1 ) +
    ORD( 'A' ) ) );
```

Az ORD('Z') - ORD('A') + 1 kifejezés egy 0 ... 25 közé eső véletlenszámot állíttat elő a RANDOM függvénnyel. Ha ehhez hozzáadjuk 'A' karakterkódját, akkor az 'A' ... 'Z' közé eső kódokat kapjuk. A megoldás megfogalmazásának előnye, hogy tetszőleges kódtáblázattal működik, ha ott a nagybetűk folyamatos kódtartományt foglalnak el, és így nem is kellett megírásához kódtáblázatot elővennünk.

A RANDOM függvény a program minden indításakor ugyanazzal a pszeudovéletlen számmal kezdi a generált sorozatot; ez előnyös lehet a program belövésénél, viszont kevésbé volna tisztességes egy szerencsejátékban így használni. Ha azonban végrehajtjuk a RANDOMIZE eljárást, a véletlenszám-generátor egy előre nem ismert kezdőértéket kap, így a véletlenszám-sorozat nem fürkészhető ki előre.

Egyéb eljárások, függvények

Van még néhány eljárás és függvény, amely hasznos lehet a programozási munkában.

A DELAY(i) eljárás várakozást iktat be a programba. Az i paraméter ms-ban, azaz ezredmásodpercben értendő. Sajnálatos, hogy az ENTERPRISE géphez ez az eljárás nincs következetesen illesztve, ezért a tényleges késleltetést kísérletileg kell megállapítani. Ha a ms-ban számított késleltetést elosztjuk 1.4-del és ezt az értéket adjuk át, nagyjából jó eredményt kapunk.

Sajnos, a Turbo-Pascal SOUND és NOSOUND eljárása itt nem működik, így zenei hangot csak barkácsolással tudunk a programunkból kicsalni. Azért ne keseredjünk el, valami hangot csak ki lehet hozni a gépből. A CTRL-G vagy BELL (azaz csengő) (08H kódú) vezérlőkarakter rendeltetésszerűen működik, egy furcsa pendülésszerű hangot ad ki:

```
CONST
    BELL = CHR( 8 );
    ...
WRITE( BELL );
```

Sokkal nagyobb hiba, hogy az EXIT eljárás is hiányzik. Az emlékedett programozási stílusnak gyakorlatilag elengedhetetlen kelléke az éppen végrehajtás alatt lévő blokkból való kiugrást eredményező EXIT eljárás. Aki visszaemlékszik a sorozat bevezető részére, az tudja, hogy egy programszerkezetnek csak egy belépési és csak egy kilépési pontja lehet. Az EXIT eljárás sok esetben jelentősen le tudja egyszerűsíteni a program szerkezetét anélkül, hogy ezt az alapszabályt fel kéne rúgni. Sajnálatos, hogy az ENTERPRISE rajongóinak enélkül kell dolgozniuk.

Megvan viszont a HALT eljárás; ez befejezi a program futását, a vezérlés visszaadódik a Turbo-Pascal fejlesztői környezetnek vagy az operációs rendszernek.

Az UPCASE(c) függény, amely paraméteréhez hasonlóan karakter típusú, a paraméterben adott karaktert nagybetűsíti. Ha az nagybetű vagy nem betű, változatlanul adja vissza. Ha egy egész stringet akarunk negybetűsíteni, akkor egy eljárást kell írnunk (a Turbo-Pascal későbbi verzióiban ugyanezt már függvénnyel is megoldhatjuk):

```
PROCEDURE UPPERCASE( VAR S : STRING255 );
VAR
    I : INTEGER;
BEGIN
  FOR I := 1 TO LENGTH( S ) DO BEGIN
    S[ I ] := UPCASE( S[ I ] );
  END { FOR I };
END { UPPERCASE };
```

Az eljárás a Turbo-Pascal stringkezelésének azt a nem mindenütt dokumentált tulajdonságát használja ki, hogy a string tulajdonképpen egy sajátos karaktertömb, amelyik a nulladik elemében tartalmazza a string hosszát. A fordítóprogram megengedi, hogy a string karaktereit indexelve, egyenként elérjük. Ha ez nem volna, akkor egy ilyen nagybetűsítést a COPY és a CONCAT függvény sokszorosan ismételt használatával érhetnénk csak el, ami sokkal lassabb volna, és felírni is kényelmetlenebb.

Ugyancsak a gyorsabb végrehajtást és az egyszerűbb programozást segíti a MOVE eljárás. Szintakszisa:

```
MOVE( forrás, cél, hossz )
```

forrás és *cél* tetszőleges típusú változó lehet, míg *hossz* nyilván egész. Az eljárás a *forrás* változóval kezdődő tárterület tartalmát másolja át a *cél* változóval kezdődő területre, a harmadik paraméter a másolt szakasz hosszát adja meg bájtban. Az eljárás használható egyrészt egy nagy adatszerkezet kisebb darabjának az átviteléhez, de fordítva is, sok-sok (de csak szomszédos) adat egy menetben történő áthelyezéséhez. Mind a két esetben gyorsabb, mintha felsorolnánk az áthelyezendő elemeket egy-egy értékadásban. Az átviendő adatmennyiség meghatározásánál hasznunkra lehet a korábban megismert SIZEOF függvény, amelyik megadja az adatelem vagy -szerkezet hosszát.

Mivel *forrás* és *cél* eltérő típusú változó is lehet, elképesztő galibákat tudunk okozni az eljárás meggondolatlan használatával. Akkor is érhet meglepetés, ha a másolandó terület belelóg a fogadó területbe, hiszen ekkor egy ponton túl a már felülírt területet másoljuk tovább. Legyünk hát óvatosak!

Még néhány függvény azoknak, akik egészen gépi szinten akarnak programozni. A HI(i) illetve a LO(i) függvény az egész paraméter felső illetve alsó bájtját adja vissza, egész értékként. A SWAP(i) függvény ugyanezt a két bájtot egymással felcserélve adja vissza. Akik foglalkoztak gépi kódú programozással, azok tudják, hogy erre miért van szükség.

Egyéb megoldások

Ugyan nem függvény, nem is eljárás a most következő néhány megoldás, mégis itt érdemes megemlíteni őket. A memória és a bemeneti-kimeneti portok közvetlen elérését a Pascal nem eljárásszerűen, hanem ú.n. előre definiált tömbök útján oldja meg.

A MEM tömb úgy viselkedik, mintha azt a felhasználó

```
VAR
    ARRAY[ 0 .. $FFFF ] OF BYTE;
```

deklarációval hozta volna létre. Ráadásul a tömb első eleme a memória 0. bájtjára illeszkedik, így a tömb elemei éppen a memória bájtjaira illeszkednek. (A beavatottak tudják, hogy a MEM tömb valóban létrehozható az

```
ARRAY[ 0 .. $FFFF ] OF BYTE ABSOLUTE 0
```

deklarációval; ez a programozó által kiválasztott *abszolút címtől* kezdődően helyezi el a tömböt.)

A tömb írható és olvasható, tehát a memória (egészen pontosan, az IS-DOS alatt látszó memória) teljes tartományát közvetlenül kezelni tudjuk. Ezzel megint csak hatalmas baklövéseket lehet elkövetni, ugyanis óvatlanul belenyúlhatunk akár a saját programunk szövegébe vagy kódjába, akár a Turbo-Pascal rendszerbe, akár az operációs rendszerbe, és ekkor beláthatatlan dolgok történhetnek. Ha viszont ügyesen kezeljük, működés közben, *in vivo* boncolhatjuk fel az operációs rendszert vagy bármi mást a memóriában, vagy akár *turbósított* (azaz a memóriában dolgozó) fordítóprogramot készíthetünk. Sajnos, a teljes memóriatartomány kezelését megnehezíti az, hogy a az ENTERPRISE gépeken futó Turbo-Pascalból még hiányzik a WORD típus, amely az egész számokat előjelesen, a -32768 ... +32767 tartományban kezelő INTEGER típussal szemben azokat a 0 ... 65535 tartományban tudja kezelni.

Ha az éppen nem elérhető memórialapokra is kíváncsiak vagyunk, meg kell oldanunk a belapozásukat, ami nem könnyű, mert a Turbo-Pascal és az IS-DOS alul-felül elfoglalja a memóriát, középtájon pedig a mi programunk van; így normál esetben nem találunk szabad szegmenst: bármelyikhez szegmenshez is nyúlunk, valamit mindig "kilapozunk magunk alól". Segít, ha a Turbo-Pascal indításakor nem töltjük be a hibaüzeneteket, vagy ha nem a fejlesztői környezetben futtatjuk a programot, hanem .COM fájllá lefordítva, közvetlenül IS-DOS-ból. A Turbo-Pascal megadja, hogy az egyes adatterületek hol kezdődnek, fordítás után pedig a hosszukat is elárulja. Figyeljünk ezekre az adatokra!

A belapozást végző eljárásokhoz, meg persze sok egyéb trükkhöz használhatjuk a szintén előre definiált PORT tömböt. Amint a neve elárulja, ez a bemeneti-kimeneti portok kezelését teszi lehetővé, mindkét irányban, tehát írásra és olvasásra is. A PORT tömb deklarációja - a Z-80 mikroprocesszor környezetében - mintha

```
ARRAY[ 0 .. 255 ] OF BYTE
```

lenne.

A 0. lapregiszter tartalmát így olvashatjuk be:

```
CONST
  PAGE0 = $B0;
VAR
  B : BYTE;
  ...
B := PORT[ PAGE0 ];
```

Ha csak olvassuk a portokat, nem történhet baj, de ha írunk is rájuk, könnyen elszállhat a rendszer. Ha viszont ügyesek vagyunk, a DAVE chip közvetlen kezelésével akár zenei szintetizátort is csinálhatunk. *(folytatjuk)* *-UL-*

Az IS-LISP

1. rész

A LISP (LISt Processor) nyelvet az 1950-es években alkották meg, elsősorban listafeldolgozás, és a mesterséges intelligencia kutatások elősegítésére. Bár azóta sok esztendő eltelt, a LISP ma is igen népszerű a programozók körében. Ezért (is) szeretnénk két részben az IS-LISP-et bemutatni olvasóinknak.

A LISP tanulmányozásához, valamint az alapok elsajátításához magyar nyelven Zimányi Magdolna - Kálmán László - Fadgyas Tibor : *A LISP programozási nyelv (Műszaki Könyvkiadó, 1989)* című könyvét ajánljuk.

Az IS-LISP sajátosságai

A változók nevei betűkből, számjegyekből, valamint speciális karakterekből állhatnak (ezeket képes változó, ill. függvénynévként értelmezni). A szögletes zárójel (]) a "nagyzárójel" szerepét tölti be (az összes nyitott nyitózárójelet lezárja).

A szövegszerkesztő segítségével készíthetjük LISP-"programjainkat" (amelyek gyakorlatilag függvények, melyek által szolgáltatott eredmények újabb függvények paramétereiként jelenhetnek meg).

A már lefordított (értelmezett) függvények újraszerkesztését, alakítását a

(FEDIT *név*)

utasítás segítségével tehetjük meg, ahol *név* a függvény neve.

Az így szerkesztett programban azonban már a *név* helyén LAMBDA szó áll (ez a már korábban ismertté vált, már értelmezett utasítás belső jelzésére szolgál). A szerkesztés végét az Esc billentyűvel közölhetjük a programmal, amely a módosításnak megfelelően újrafordítja azt.

Programok kimentése, betöltése

A LISP-rendszer bővítése után kapott új állapotot (függvények, változók) elmentve, a későbbiek során ettől az adott szinttől folytathatjuk munkánkat.

A mentést a

(SAVE "*név*")

utasítással, a visszatöltést pedig a

(LOAD "*név*")

segítségével tehetjük meg.

IS-LISP függvények

A következőkben a beépített függvényeket soroljuk fel, megadva a hívási paramétereiket, valamint a típusukat. Ez a felsorolás lehetőség szerint az irodalomtól való eltérést mutatja.

A típusok a következőek lehetnek:

subr - olyan függvény amely kiértékeli az összes argumentumát (paraméterét).

fsubr - az összes argumentum kiértékelése nem kötelező.

id - speciális LISP azonosító.

var - a rendszerben használható (beépített) változó.

(folytatjuk)

Légrádi Gábor

Függvény	szolgáltatott érték	leírás	típus
(ABS szám1)szám1	szám1=abs(szám)	abszolútérték	subr
(ADD1 szám1)	szám2	szám2=szám1+1	subr
(APPEND lista1 lista2)	lista	lista=lista1+lista2	subr
(AT sor oszlop)	NIL	a kurzort a sor,oszlop helyre mozgatja	
(ATOM x)	T vagy NIL	T (igaz) értéket adha az x atom, nem lista	subr
(BAND2 szám1 szám2)	szám	szám=(szám1 AND szám2)	subr
BLANK	space	a space karatert tárolja	id
(BNOT szám)	szám	bitenként negál	subr
(BORDER szám)	NIL	a keretszínt állítja be	subr
(BOR2 szám1 szám2)	szám	bitenkénti VAGY művelet	subr
(BXOR@ szám1 szám2)	szám	szám=szám1 XOR szám2	subr
(CAPTURE régi új)	NIL	átirányítja a régi csatornáról történő olvasási műveleteket az új csatornára	subr
(CHARACTER szám)	betű	a számnak megfelelő ASCII karaktert adja	subr
(CHARP x)	T vagy NIL	T ha a paraméter változó NIL egyébként	subr
(CHARS par)	szám	a par hosszát adja	subr
(CLEAR)	NIL	képernyőtörlés	subr
(CODEP x)	T vagy NIL	T ha x kódpointer NIL egyébként	subr
(COMMENT szöveg)	NIL	megjegyzés	fsubr
(CREATE csatorna név)	szám	név file megnyitása	subr
(CURSOR kif)	NIL	kurzor ki-be	subr
(DIGIT x)	T vagy NIL	T ha x neve számmal kezdődik, NIL egyébként	subr
(EDIT kif)	—	a kif szerkesztésére	subr
(ERROR szám)	—	hibagenerálás	subr
(FKEY kszám szöveg)	NIL	funkcióbill. definiálás	subr
(GETCHAR)	—	egy karakter beolvasása	subr
(GRAPHICS)	NIL	grafikus megjelenítés	subr
(IN ioport)	szám	portról olvas	subr
(LITER x)	T vagy NIL	T ha x neve betűvel kezdódik, NIL ha nem	subr
(MAX2 szám1 szám2)	szám	visszaadja a nagyobbat	subr
(OBLIST)	lista	a rendszernevek listában	subr
(OUT érték ioport)	érték	portra ír	subr
(TEXT)	NIL	szöveges megjelenítés	subr
(TIME)	szöveg	az időt adja vissza	subr
(DEFVIDEO mód gmód gszín)	NIL	szöveges és grafikus paraméterek beállítása (mód text-hez 40 vagy 80; gmód=video mode; gszín=video color)	subr

Adatok rendezése

A programozás során gyakran visszatérő feladat a különböző adatok sorrendbe szedése. Pl. lottószám generálás, szavak abc sorrendű tárolása. Az első esetben numerikus, a másodiknál sztring adatokkal van dolgunk. A rendezés egyszerű relációs összehasonlítások sorozatán alapszik.

A számok összehasonlítása X és Y változó esetében:

```
IF X>Y THEN CALL ELJÁRÁS
```

A sztringek relációja visszavezethető a numerikus adatoknál alkalmazott módra. Olyan számot kell az eljárásnak képeznie, ahol minden számjegy 0-255 közötti értéket vehet fel. Ez a 256-os számrendszernek felel meg. A BASIC nyelvű reláció csak azonos sztringhosszak esetében ad helyes eredményt. X$ és Y$ sztringváltozó esetében:

```
IF X$>Y$ THEN CALL ELJÁRÁS
```

Eltérő karakterszámú sztringeknél gondoskodni kell hogy az összehasonlítás azonos hosszúságnál jöjjön létre. Úgy oldható meg, hogy a rövidebb sztringet kipótóljuk a mögé helyezett szóközkarakterekkel.

```
100 IF LEN(X$)<(Y$) THEN X$=X$&" ":GOTO 100
110 IF LEN(Y$)<(X$) THEN Y$=Y$&" ":GOTO 110
120 IF X$>Y$ THEN CALL ELJÁRÁS
```

A 100, 110 sorok időigényesek. Gyorsabb megoldás a következő. Programunk elején deklarálunk egy S$ sztringváltozót. Annyi szóközkaraktert töltünk bele, amennyit a leghosszabb szó igényel.

```
S$=" " vagy változó hosszúság esetén
S$=""
FOR X=1 TO MAXLEN
  S$=S$&" "
NEXT
```

Az S$-t a későbbiekben a sztringek hosszabbítására fogjuk felhasználni.

```
IF X$&S$(LEN(X$)+1:)>Y$&S$(LEN(Y$)+1:) THEN
CALL ELJÁRÁS
```

A rendezésre többféle módszer is létezik. Általánosítható, hogy az egyszerűbb algoritmusok időigénye gyorsabban növekszik, ha növeljük a tagok számát. A futásidő függ még az adatok rendezettségi állapotától is. Fordított sorrend esetén megnő a mozgatások száma. Három gyakran használt algoritmusra fogok kitérni.

Buborékrendezés (Bubble Sort)

Egyszerűsége folytán a leggyakrabban használt algoritmus. Az egymás melletti tagokat összehasonlítjuk, és fordított reláció esetén megcseréljük őket. Ha a tagok száma N, akkor (N-1)-szer végrehajtott művelettel valamennyi tagot elérhetünk. Ekkor a legnagyobb (legkisebb) tag az N-ik helyre került. A rendezetlen tömbünk N tagszáma 1-gyel csökkent. Az eljárást addig ismételjük, amíg a tagok száma 1-re csökken. A példa esetében a tagokat előzetesen D(1 to N) tömbbe helyeztük.

```
FOR I=N-1 TO 1 STEP-1
  FOR J=1 TO I
    IF D(J)>D(J+1) THEN
      LET A=D(J)
      LET D(J)=D(J+1)
      LET D(J+1)=A
    END IF
  NEXT
NEXT
```

A futásidő, és a tagok száma közötti összefüggést a $t=k*N\hat{}2$ képlet adja, ahol a k időállandót egy futtatás idejéből lehet kiszámolni: $k=t/N\hat{}2$

Szekvenciális rendezés

Az előző módszer ciklusmagjának futásideje csökkenthető.

```
FOR I=N-1 TO 1 STEP-1
  LET A=D(I+1)
  FOR J=1 TO I
    IF A<D(J) THEN
      LET B=D(J)
      LET D(J)=A
      LET A=B
    END IF
  NEXT
  D(I+1)=A
NEXT
```

A sztringtömb rendezése D$(1 TO N) esetén:

```
FOR I=N-1 TO 1 STEP-1
  LET A$=D$(I+1)
  FOR J=1 TO I
    IF A$&S$(LEN(A$)+1:)<$(J)&
       S$(LEN(D$(J))+1:) THEN
      LET B$=D$(J)
      LET D$(J)=A$
      LET A$=B$
    END IF
  NEXT
  D$(I+1)=A$
NEXT
```

A következő assemly rutin a KEZD címtől, HOSSZ számú bájtot rendezi nagyság szerint.

```
QUICK:   !LD DE,KEZD
         !LD BC,HOSSZ-1
         !LD H,B
         !LD L,C
         !ADD HL,DE
         !EX DE,HL!DE=KEZD+N-1
QUICK10: !PUSH HL!KEZD ELMENTÉS
         !PUSH BC!HOSSZ ELMENTÉS
         !LD A,(DE)!UTOLSO BÁJT "A"-BA
QUICK20: !CP (HL)!RELÁCIÓ
         !JR NC,QUICK30!UGRÁS HA =
         !EX AF,AF'!"A" MENTÉS
         !LD A,(HL)!BÁJT KIVÉTELE
         !EX AF,AF'!"A" VISSZA
         !LD (HL),A!CSEREÉRTÉK BEÍRÁSA
         !EX AF,AF'!ÚJ "A" ÉRTÉK
QUICK30: !CPI !POINTER NÖVELÉS
         !JP PE,QUICK20!NEXT
         !LD (DE),A!LEGNAGYOBB ÉRTÉK BEÍRÁS
         !DEC DE!UTOLSÓ PTR CSÖKKENTÉS
         !POP BC!HOSSZ VISSZA
         !CPI !HOSSZ CSÖKKENTÉS
         !POP HL!KEZD VISSZA
         !JP PE,QUICK10!NEXT
         !RET
```

Gyorsrendezés (Quick Sort)

Használatával kedvezőbb futásidőt érhetünk el rendezetlen tömb, és magasabb tagszám esetén. A futásidő, rendezett vagy fordított sorrendű tömbnél $t=k*N\hat{}2$, amely a rendezetlennél kb. $t=k*N*LOG(N)$-re csökkenhet. Az időnyereséget a tömb fastruktúrájú, résztömbökre való felosztása okozza.

A módszer logikai menete: A rekurzív eljárás átveszi a rendezni kívánt tömb alsó és felső határát (A-F). Kezdő esetben ez a teljes tömböt magába foglalja (1-N). Az első tag értékét a referencia változóba helyezi (K). A felső határtól lefelé haladva, K-nál kisebb értéket keres. Ezt az értéket megkapja az alsó tag. Az alsó határ 1-gyel növekszik. Most felfelé haladva kell K-nál nagyobb értéket keresni, melyet a felső tag fog megkapni. Addig folytatódik az eljárás, amíg az alsó és felső határ azonos nem lesz. Ebbe a tagba visszakerül a referencia érték (K). A tömb ilymódon 3 részre lett osztva. "Középen" a referenciát tartalmazó tag, fölötte az azonosak és nagyobbak, alatta pedig az azonosak és kisebbek. Azt a tömböt amelyik 1-nél több tagot tartalmaz, rekurzív módon ismét átadjuk a rendező eljárásnak, megadva az alsó és felső határát. A QUICK1 program DO-LOOP ciklust használva egyszerűbb, míg a QUICK2 FOR-NEXT ciklussal gyorsabban fut.

Az algoritmus rekurzív mélysége legrosszabb esetben N, amíg a BASIC rekurzív hívás maximális fokszáma 30-40 között van. Ha a programunk pl. N=100 rendezett tömböt kap, "elszáll". A rekurziót ki lehet alakítani veremhasználattal is, ily módon nagyobb mélység is elérhető. A vermet célszerűen V(N*3) tömbváltozóval oldjuk meg (A,F,E-t kell hívásonként menteni). Az elmentésnél (PUSH) a verempointer (I) által megcímzett változóba írjuk az adatot, majd I-t 1-gyel növeljük. Visszakérésnél (POP) csökkentjük I-t, majd kiolvassuk V(I) értékét. A program a hívásra a kisebb memóriaigényű GOSUB-RETURN megoldást alkalmazza.

(A programlisták a következő oldalon találhatók) *(Hsoft)*

```
100 PROGRAM "quick.vrm"
110 PRINT "Szamok gyorsrendezese basic
    veremkialakitassal. 1990. Hsoft:"
120 INPUT PROMPT "N=":N
130 PRINT "n=";N "ideje kb."
    INT(.042*N*LOG(N)) "masodperc"
140 RANDOMIZE
150 NUMERIC T(N),V(N*3)
160 FOR X=1 TO N
170   LET T(X)=RND(10000)
180 NEXT
190 TIME "00:00:00"
200 LET V(1),A,E,I=1
210 LET V(2),F=N
220 GOSUB 300
230 PRINT TIME$
240 FOR X=1 TO N
250   PRINT USING "#####":T(X);
260   IF NOT MOD(X,7) THEN PRINT
270 NEXT
280 PRINT
290 END
300 LET V(I+2)=E:LET I=I+3
310 LET E=A:LET U=F:LET K=T(E)
320 DO UNTIL E=U
330   FOR U=U TO E+1 STEP-1
340     IF K>T(U) THEN
350       LET T(E)=T(U)
360       FOR E=E+1 TO U-1
370         IF K<T(E) THEN LET T(U)=T(E)
                      :LET U=U-1:EXIT FOR
380       NEXT
390       EXIT FOR
400     END IF
410   NEXT
420 LOOP
430 LET T(E)=K
440 IF A<E-1 THEN LET V(I)=A:LET V(I+1)=F:
    LET F=E-1:GOSUB 300
450 IF E+1<F THEN LET V(I)=A:LET V(I+1)=F:
    LET A=E+1:GOSUB 300
460 LET I=I-3:LET A=V(I):LET F=V(I+1):
    LET E=V(I+2):RETURN
```

```
100 PROGRAM "quick2"
110 PRINT "Szamok gyorsrendezese, basic
    rekurziv hivassal. /1990 Hsoft:"
120 INPUT PROMPT "N=":N
130 PRINT "n=";N "ideje kb."
    INT(.036*N*LOG(N)) "masodperc"
140 RANDOMIZE
150 NUMERIC T(N)
160 FOR X=1 TO N
170   LET T(X)=RND(10000)
180 NEXT
190 TIME "00:00:00"
200 CALL Q(1,N) !rendezes
210 PRINT TIME$
220 FOR X=1 TO N
230   PRINT USING "#####":T(X);
240   IF NOT MOD(X,7) THEN PRINT
250 NEXT
260 PRINT
270 DEF Q(A,F)
280   NUMERIC E
290   LET E=A:LET U=F:LET K=T(E)
300   DO UNTIL E=U
310     FOR U=U TO E+1 STEP-1
320       IF K>T(U) THEN
330         LET T(E)=T(U)
340         FOR E=E+1 TO U-1
350           IF K<T(E) THEN LET T(U)=T(E):
                      LET U=U-1:EXIT FOR
360         NEXT
370         EXIT FOR
380       END IF
390     NEXT
400   LOOP
410   LET T(E)=K
420   IF A<E-1 THEN CALL Q(A,E-1)
430   IF E+1<F THEN CALL Q(E+1,F)
440 END DEF
```

```
100 PROGRAM "quick1"
110 PRINT "Szamok gyorsrendezese, basic
    rekurziv hivassal. /1990 Hsoft:"
120 INPUT PROMPT "N=":N
130 PRINT "n=";N "ideje kb."
    INT(.0457*N*LOG(N)) "masodperc"
140 RANDOMIZE
150 NUMERIC T(N)
160 FOR X=1 TO N
170   LET T(X)=RND(10000)
180 NEXT
190 TIME "00:00:00"
200 CALL Q(1,N) !rendezes
210 PRINT TIME$
220 FOR X=1 TO N
230   PRINT USING "#####":T(X);
240   IF NOT MOD(X,7) THEN PRINT
250 NEXT
260 PRINT
270 DEF Q(A,F)
280   NUMERIC E
290   LET E=A:LET U=F:LET K=T(E)
300   DO UNTIL E=U
310     DO UNTIL E=U OR T(U)<K
320       LET U=U-1
330     LOOP
340     IF E<U THEN
350       LET T(E)=T(U):LET E=E+1
360       DO UNTIL E=U OR T(E)>K
370         LET E=E+1
380       LOOP
390       IF E<U THEN LET T(U)=T(E):
                    LET U=U-1
400     END IF
410   LOOP
420   LET T(E)=K
430   IF A<E-1 THEN CALL Q(A,E-1)
440   IF E+1<F THEN CALL Q(E+1,F)
450 END DEF
```

Assembly rutinok II.

Az előző lapszám NUM_STR.ASM rutinját egy újabbal bővítjük. A neve ERROR.ASM. Hívásakor az akkumulátorban lévő hibakód szöveges megfelelőjét az alapértelmezésű csatornára írja. A rutin a hibaszöveget EXOS 28-as hívással kéri le. DE regiszter a hibapufferre mutat. Visszatéréskor az első bájt megadja a hosszúságot, melyet a hibasztring követ. Szöveges magyarázat hiányában (nulla hossz), decimális és hexadecimális formában kiírja a hibakód számát. A rutin valamennyi regiszter tartalmát megőrzi. 64 bájtos RAM-puffert igényel, melynek a kezdőcímét ERRBUFF néven kell megadni.

A tesztelésére ASMON alatt futó kódot írtunk. Kíratjuk a 128-255 közötti EXOS-hibaüzeneteket a képernyőre. A [Z]-opcióval beállítjuk:

Memory assembly	YES
Memory offset	0

[A]-val fordítunk, [G]3000-rel futtathatjuk.

(Hsoft)

```
;EXOS 1-255 HIBAUZENETEK KIIRASA:
              ORG 3000H
              LD A,128        ;KODSZAMLALO KEZDOERTEK
CIKLUS:       LD C,1          ;VEZETO SZOKOZOK
              PUSH AF
              CALL DECAPRINT  ;HIBAKOD DECIMALISAN
              CALL SPPRINT    ;SZOKOZ KIIRAS
              POP AF
              CALL ERRORPRINT ;HIBASZOVEG KIIRAS
              INC A           ;
              JR NZ,CIKLUS    ;CIKLUS SZAMLALAS
              RET
              INCLUDE NUM_STR.ASM
              INCLUDE ERROR.ASM
ERRBUFF:      DEFS 64         ;HIBASZOVEG PUFFER
              END
```

Ha minden fólia szakad...

... arra az esetre jól jön egy professzionális billentyűzet, pl. az IBM XT-é. A kérdés csupán az: hogyan illesszük kedvenc ENTERPRISE-unkhoz?

Két megközelítési mód létezik. Az egyik szerint az XT-billentyűzet elektronikáját elfelejtjük, csak a gombokat használjuk fel. Rettentő munkával az ENTERPRISE fóliájának megfelelően összekötve azokat, egy nyolc- és egy tízerű vezetékkel a fólia helyére csatlakoztatva az eredetinek teljesen megfelelő, "csak" nagyságrendekkel jobb tasztatúrát kaphatunk. Ennek hátránya, hogy bele kell nyúlnunk a gép belsejébe, miáltal a garancia érvényét veszti (bár az ENTERPRISE-kereskedelem helyzetét elnézve, ez ma már alig fenyeget valakit), valamint nem használhatjuk ki az XT billentyűzetének néhány kellemes lehetőségét, például a számmezőt.

A másik megközelítési mód szerint az XT-billentyűzetet egységes egészként kezeljük, és némi hardverrel egybekötött szoftverrel oldjuk meg az ENTERPRISE-hoz való illesztést. Rögtön az elején meg kell azonban vallanom szinte már kegyetlen őszinteséggel, hogy ezt a megoldást nem akceptálja a játékprogramok azon része, amelyek a billentyűzetet közvetlenül, I/O-címeken keresztül kérdezik le. Sajnos ezek teszik ki a több, mint kétharmados többséget, az EXOS billentyűzet-csatornáját használó játékok csupán törpe minoritást képeznek. Viszont azok a géptársak - és nagy örömömre jelentős számban találhatók ilyenek - akik elsősorban nem játéknak, hanem "komolyan" használják a gépet, remek hasznát vehetik egy "igazi" XT-billentyűzetnek.

Lássuk mindenekelőtt, hogy is működik a szóbanforgó tasztatúra! 83 (az amerikain), vagy 84 (az európaiakon) gomb található rajta, amelyeket egy célmikroprocesszor kérdez le (ti. az a mikroprocesszor célja, hogy lekérdezze a gombokat). Ha egy gombot lenyomunk, a processzor a hozzá tartozó kódot (pl. az ESC-re 5Ah-t, az "A"-ra a 1Fh-t, a legmagasabb bit mindig 0) küldi el a központi számítógépnek. Ha a gombot nyomva tartjuk, a kód bizonyos időközzel újra és újra továbbítódik (ismétlés). A központi gép onnan tudja, hogy elengedték végre a gombot, hogy a billentyűzet ennek megfelelő kódot generál: ez ugyanaz, mint a lenyomáskori kód, de a legmagasabb bit 1-ben van (azaz ESC-nél DAh, "A"-nál 9Fh).

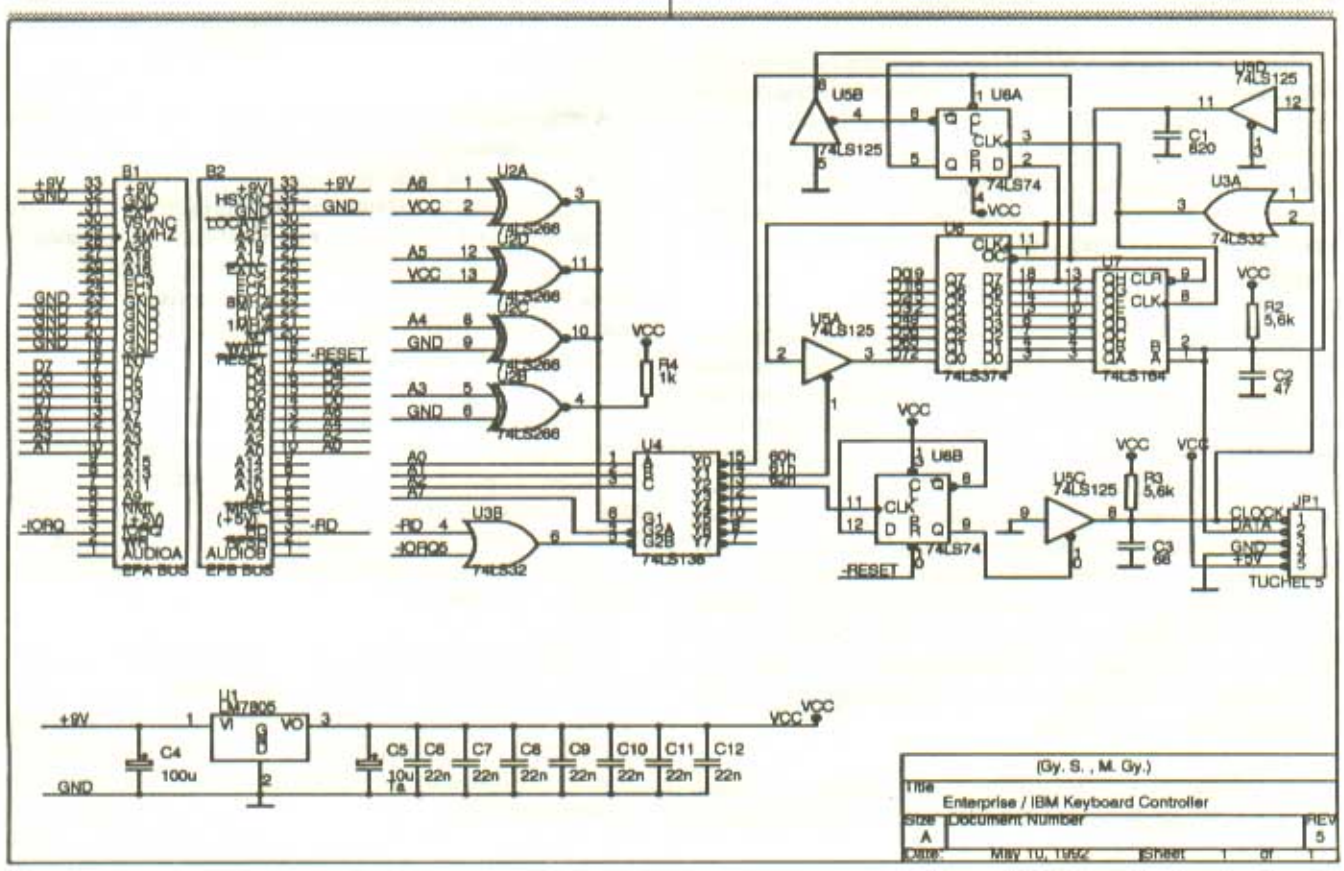
Fizikailag az adatátvitel másfélirányú soros vonalon történik. A fönti kijelentést úgy kell érteni, hogy a billentyűzet adatokat küld a számítógépnek (ez eddig egy irány), a számítógép pedig "üzenni" tud a billentyűzetnek, hogy foglalt, illetve a billentyűzetet inicializálni akarja (ez a fél irány). A kapcsolat általában 4 szál dróton keresztül valósul meg. Elvileg a billentyűzetnek van egy ötödik - RESET - bemenete, ezt azonban többnyire hanyagolják. Egy drót a föld, egy meg a +5 volt, ami szinte nélkülözhetetlen. A maradék kettőből az egyiken az adatok, a másikon az órajel áramlik az alábbi módon:

A billentyűzet először megvizsgálja, hogy az adatvonalat a számítógép nem húzta-e le (ez a foglaltság jelzése). Ha nem, és volt lenyomott, vagy elengedett billentyű, a megfelelő nyolcbites kódot, előtte egy START bittel, szépen sorban elküldi, miközben ennek megfelelően 9 órajelet generál. Az adatbitek az órajel alacsony állapotában érvényesek. Ha a számítógép az óravonalat húzza le legalább 40 ms időre, a billentyűzet önteszt végrehajtása után alapállapotba kerül, az önteszt eredményét (55h, ha minden rendben, FCh, ha valami gond van) pedig elküldi a számítógépnek.

Most, hogy már mindent tudunk az elméletről, rátérhetünk a konkrét illesztőkártya tanulmányozására. A billentyűzet a JP1 ötpólusú tuchel-aljzatra csatlakozik. Soros adatait az U7 léptetőregiszter fogadja, ide érkezik az órajel is, amelynek felfutó élére tárolódik, és lép eggyel tovább a bejövő adat. Legelől jön a START bit, amely a kilencedik ütemre U7-ből az U8A D-tárolóba kerül (ekkorra U7-ben a nyolcbites kód található). Ennek hatására U8A kimenetei állapotot váltanak, letiltják az esetleges további órajeleket, hogy a beérkezett kód ne íródhasson felül, U5B-n keresztül lehúzzák az adatvonalat, ezzel jelezve a foglaltságot a billentyűzetnek, és végül, de elsősorban átírják a kódot U7-ből az U6 regiszterbe, valamint U5A segítségével jelzik a számítógépnek a 61h I/O cím 7. bitjén, hogy billentyűkód érkezett. Maga a kód U6-ból a 60h címen olvasható ki.

A 62h cím olvasásával billegtethető az U8B D-tároló: így lehet a billentyűzetet az órajel-vonal U5C-n keresztüli lehúzásával programozottan alapállapotba állítani. A fent említett címek dekódolását az U2, U3B, U4 áramkörök végzik, U1 pedig a stabilizált tápfeszültséget állítja elő a kártya és a billentyűzet számára.

(folytatjuk...) *Gyányi Sándor & Mészáros Gyula*



MUSIC BOX by Gyányi Mester:
Sanyi bácsi lemezesládája

A lemezen, amit kipróbálásra megkaptunk, néhány fájl és két alkönyvtár található. A HELP alkönyvtárban található MUSICBOX.HLP nevű fájl, mint sejtettük, egy igen gondosan elkészített, a program kezelését részletesen ismertető - sajnos, "éktelen": ékezetek nélküli magyarsággal megírt - kézikönyvet takar. A TUNES alkönyvtárban sok-sok .TUN kiterjesztésű fájl található, nyilván ezek a lejátszható zenszámok. A gyökérkönyvtárban lévő fájlok egyike egy konfigurációs fájl, amely az alapbeállításokat tárolja a program következő indításához. Ez követendő példa. Egy további fájl azoknak a baját enyhíti, akik csak magnóval rendelkeznek. Végül a nemes egyszerűséggel START névre hallgató fájl maga a program. Csak egy gombnyomás és indul...

Nézzük meg, mit mond a program önmagáról:

ENTERPRISE MUSICS BOX version 1.2
(C) 1991 by Nihil software
Written by Gyanyi Sandor
Demo musics from OUTRUN,GRYZOR,BOBSLEIGH
and etc. (Over 49 musics!)

Csak sejthetjük, hogy milyen jogviszonyban van az alkotó, Gyányi Sándor, a program terjesztői jogával rendelkező *Nihil software* céggel. Hadd kezdjük kötözködéssel (reméljük, a szerző megbocsájt, ugyanis ezek a legsúlyosabb kifogásaink...): A MUSICS többes száma itt felesleges (a kézikönyv-fájlban már helyesen szerepel). Az etc. (latinul *at caetera*, magyarul *és a többi*) előtt felesleges és értelmetlen az *and*. Nem tudjuk, mit kezdjünk a "több mint 49" zeneszámmal. Mennyi az? Ötven? (Tényleg több, mi 106-ot számoltunk, igaz, ezek között volt néhány nyilvánvalóan "amatőr" próbálkozás is.)

Lássuk a lényeget. A MUSIC BOX egy zenei szerkesztő és visszajátszó program. Ha elindítjuk, bonyolult menüszerkezetet rajzol ki a képernyőre. A kurzort akárhova vihetjük a képernyőn, ott a szóköz billentyűt (vagy a tűzgombot) megnyomva, aktivizálódik a megfelelő funkció. Ügyes. Azért kiegyeznénk, ha a tűzgomb mellett az ENTER is indítaná a kiválasztott funkciót, igazodva a számítógépes világban kialakult *de facto* szabványokhoz.

Később már kevésbé vagyunk elégedettek a menürendszerrel. Egyrészt, ha többszörös mélységben hívogatunk almenüket és al-almenüket (van még egy pótkötél...), nem lehet tudni, legalábbis fekete-fehér monitoron, hogy az egymásra szuperponált menüablakok közül éppen melyik az aktív. Nagyon élvezetes a teljes képernyőn szabadon száguldozni, de ez felesleges, ha csak egy néhány négyzetcentiméternyi felületen fogad el a program valamit. Elegendő lenne mindig csak az érvényes ablakban vagy ablakokban mozogni, azt pedig feltűnően megkülönböztetni a többitől. Még egy apróság: ha kiválasztottunk egy betöltendő fájlt, felesleges még külön megkérdezni, hogy tényleg be akarjuk-e tölteni (különben mi a ööööööő választottuk volna ki?). Ezzel négyről háromra csökkenne az egyetlen fájl betöltéséhez megmozgatott menük száma.

A program az ENTERPRISE hanggenerátorát használja, annak minden pozitívumával és korlátjával. A megszólaló zeneszámok először érdekesek, később a hangzás kicsit egysíkúvá válik. A szintetikusan előállított illetve mintavételezéssel tárolt hangminták használata izgalmasabb hanghatásokat adna. A hasonló hangzás kicsit összemossa a "több mint 49" zeneszám nagy részét. Egyvalamit azonban nem lehet elhallgatni: A sok-sok egy kaptafára készült "noname" zeneszámnak mind-mind megvan a címe és a szerzője; az egyetlen *régebbi* zenedarabnak a címe viszont csak *Classical music*, szerzője pedig... egy kérdőjel és egy újabb "noname" név. Pedig J. S. Bach zsenijét már az első néhány hang után fel lehet ismerni... Csak feltételezzük, hogy maguk a zeneminták egy másik géptípus "kottatárából" vannak.

Lássuk, mi van belül... A program három csatornát (plusz egy zajcsatornát) képes kezelni és szinkronizálni. A zene hangterjedelme 8 oktávot foghat át. Az egyes hangok hosszát 1/32-ekben kell megadni. A tényleges tempó persze ettől függetlenül beállítható.

A szerkesztés megoldása nagyon célszerű. A zenei szólamok úgynevezett *motívumokból* állnak. A motívumot elég egyszer megszerkeszteni, ezután csak hivatkozni kell rá, hogy megszólaljon. Ezzel mind a "zeneszerzői" (inkább azt mondanám, *zeneszerkesztői*) munka, mind a zenét tartalmazó fájl hossza jelentősen lerövidül. Mint tudjuk, bizonyos fajta zeneszámok igen sokszor ismétlődő motívumokból állnak... Azt is eldönthetjük, hogy a teljes zenemű végére érve a lejátszás befejeződjön vagy újra elinduljon. Lejátszáskor látjuk mindhárom csatorna ablakában az éppen hallott motívum nevét, és szemünk előtt peregnek az éppen hallott hangok kódjai.

A hangok burkológörbéjét grafikusan szerkeszthetjük, és a kész burkológörbéket elmenthetjük, így máskor újra felhasználhatjuk azokat.

A program számtalan opcióval gondoskodik a kényelmünkről. Ha nem igazán szeretjük a hangok nevét megadni szerkesztéskor, a klaviatúrát mintegy zongorabillentyűzetként használva "beklimpírozhatjuk" a dallamot, később korrigálva a ritmust. Ha akarjuk, hallhatjuk is közben a megfelelő hangot. A transzponáló effektus ("kontár kántorok" segédeszköze) nem csak azokon segít, akik nem szeretik a "fekete billentyűket" használni, hanem lehetővé teszi a már lejátszott dallam újraírás nélküli ismétlését más és más fekvésben.

Az elkészített zenemű nem csak betölthető .TUN fájlként menthető el, hanem, hogy az alkotó terminológiáját használjuk, gépi kódú programként is, azaz ellátható beépített vezérlővel, amely képes a dallamot lejátszani, és amellyel együtt a zenemű önálló kiegészítésként a felhasználó saját programjához hozzászerkeszthető. Semmi akadálya tehát, hogy a képhez méltó hanghatás társuljon...

Mi hiányzik ebből a programból? Talán csak egy igazi kottaszerkesztő, amellyel hangjegyekként írhatjuk be szerzeményünket, vagy fordítva, a megszerkesztett dallamot kottaként adja vissza. Csak ajánlani tudjuk a MUSIC BOX programot mindenkinek, aki zenét akar szerezni, szerkeszteni vagy programjaihoz applikálni.

-UL-

AMIKOR AZ EXOS NEM OSZTOTT LAPOT...

Egy 5-ös fejlécű (NAP) program 100h-tól BFFF-ig terjedhet. Nemrégen történt, hogy olyan programot kívántam futtatni, amelynek a vége a második lapra esett. A program nem működött. A bogarászás végén ellenőriztem az első és második lapszegmens számát. Az első lapon a második lapra kívánkozó, a másodikon pedig a 255-ös szegmenset találtam. Az EXOS elfelejtett, vagy nem tudott helyesen belapozni. A feladatot a következő módon oldottam meg. Az EXOS az előző felhasználói szegmenseket felszabadítja, és kettőt lefoglal a program részére. Az első felhasználó által foglalt szegmens az első lapra, a második pedig a másodikra kerül.

(Hsoft)

```
ORG 100H
LD SP,100H
LD A,255
OUT (0B2H),A
LD HL,(0BF9CH)
INC HL
LD A,(HL)
OUT (0B1H),A
INC HL
LD A,(HL)
OUT (0B2H),A
```

Szevasztok ENTERPRISE-osok!

Örömmel jelenthetem be ezennel másodszori megjelenésem! Családi okokból sajnos a múlt számban nem tudtam jelentkezni, de vigasztalásul akkor is olvashattátok megjegyzéseimet a játékleírásokban. Most újult erővel térhetek vissza a lap hasábjaira. Továbbra is várom leveleiteket címemre: ENTERPRESS-EPY.

Virág Attila budapesti felhasználó a *The Great Escape* leírását várja tőlem. Igyekezni fogok, l. alább.

Mocsári Balázs, Eger: Előreláthatólag a *Sir Fred* leírása egy számban fog megjelenni a közeljövőben a The Great Escape-pével.

Olli & Lissa

The ghost of the Shilmoore castle

Egyszer régen, valahol Skóciában a gonosz *Sir Humphrey* szelleme elfogta, és kastélyába zárta a "gyönyörűséges" *Lissát* és a kastélyban dolgoztatta.

De élt azidőtájt szintén Skóciában egy nemes lelkű "ifjú lovag" nevezetesen *Olli*, aki *Lissa* szívszerelme és vőlegénye, aki elhatározta, hogy megmenti szíve hölgyét.

Azon nyomban el is ment a szellem kastélyába, ám amikor belépett, rögtön a szellemmel találta magát szembe. A kísértet elmondta neki, hogy csak azért fogta el *Lissát*, mert egy régi prófécia szerint a várban el van rejtve 8 mágikus tárgy, amelyekből ha egy tiszta szívű "lány" főzetet készít, akkor az elkárhozott lélek megtalálja örök nyugodalmát. Ezt a főzetet kell Lissának elkészítenie; miután ezt megtette, a szellem elengedi. De a szellem hiába keresett-kutatott a várban, sehol sem találta a szükséges dolgokat. Olli ekkor lovagiasan ígéretet tett, hogy megszerzi a 8 tárgyat, csak hogy szerelme szabad lehessen. . .

Körülbelül ez a története az **Ionis Software International** egyik kedves kis programjának, az *Olli & Lissa I.*-nek, amit inkább a fiatalabb korosztálynak ajánlok, de az idősebbek is kipróbálhatják.

A játék kezdetekor a vár konyhájában találjuk magunkat, fönt egy nagy üstnél Lissa szorgoskodik, mellette a szellem bolyong. Ezen a képernyőn van az első tárgy, egy lámpa. Miután fölvettük, menjünk föl a lépcsőkön Lissához, és adjuk oda neki. Ezután mehetünk egy képernyővel tovább. Ezt a "szertartást" ezenkívül még hétszer kell megismételni, minden egyes tárgy megszerzése után visszamenni a konyhába. Ez idáig még egyszerű is lenne, ugyanis nem lehet eltévedni, mert mindig balról jobbra kell haladni és vissza. A tárgyak a vár különböző helyein vannak. Csakhogy mindenütt hemzsegnek a szörnyek, és van olyan hely, ahol igen nehezen lehet megtalálni a járható utat. Ezért inkább rövidre fogom a leírást, és közlök egy térképet az útvonalról.

A nyolc tárgy a következő:*lámpa, gyémánt, papírtekercs, gomba, béka, koponya, kulcs, varázsital*. Ezeket ilyen sorrendben lehet megtalálni az egymás utáni képernyőkön.

Néhány jótanács:

A 2. képernyőn, ahol a gyémánt van, a lépcsőzésekre vigyázzunk, mert ha egyszerre két lépcsőfokot lépünk, akkor a figuránk összetöri magát. Az alsó lépcsőnél várjuk meg, amíg elmegy a szörny jobbra, ugorjuk át, majd ide-oda cikázva óvatosan (!) menjünk le a lépcsőn. Ezzel legyünk gyorsak, mert ha a szörny eléri a fejünket, az egyenlő a klinikai halállal.

A 3. képernyőn egyik padkáról a másikra ugrálni kell, mert ha csak simán átmegyünk, akkor egy kissé belesüllyed Olli a padkába és nem tudunk átjutni. Ugyanez érvényes az 5. pályára is. Az ugrálást akkor kezdjük meg, amikor mindhárom denevér magasan a fejünk fölött jár, és legyünk gyorsak.

A 4. képernyőn a kukacot úgy tudjuk átugrani, ha egészen a padkák legszélére állunk. Vigyázzunk a pókokra!

A 6. képernyő alsó részén azután kezdjük meg az átkelést, mikor a medúza és a fölötte levő szörny pont egymás fölött jön visszafelé, majd újból eltávolodnak.

A 7. pályán szintán vigyázzunk a lépcsőzésre!

Az élet sajnos rohamosan fogy (még álldogálás közben is), ezért aki végig akarja játszani, az írja át a programot "örökéletesre". (L. ENTERPRESS, 1990. november-december)

Összességében véve jó ez a program, szép háttérgrafikával, de láttam nála már jobbat is. Mást nem tudok róla írni, mert könnyebb a térképen kiigazodni, mint a leíráson. Lehet, hogy sokan a leírás olvasása közben elhúzzák a szájukat, hogy milyen program lehet, mert ez az a program, amelyet könnyebb megfejteni, mint leírni. Normális ember nem képes egy ilyen leírásra (szerintem).

(Jaj, hogy sajnáltatja magát, csak azért, hogy ezért a zagyvaságért kevesebb kritikát kapjon!-EPY) Ami igaz, tényleg egy nagy zagyvaságot hordtam itt össze, aki ezen ki tud igazodni, őszinte tiszteletem előtte, aki meg nem, annak segít a térkép. Aki még így sem ért az egészből semmit, (teljesen megértem, hiszen én sem értem) az írjon nekem az újság címére, és megkísérlem neki elmagyarázni, miről is volt itt most szó.

Erről most többet nem tudok írni, ezért be is fejezem gyorsan, mielőtt lincshangulat tör ki az olvasók körében.

Lola

OLLI & LISSA	
Grafika:	8
Zene/FX:	5
Játszhatóság:	8
Összhatás:	7

CHEAT MIX

Dragon ninja 128K

[ESC] a főmenüben - végtelen élet, idő, energia.

Rock"n"roll

[A]+[S]+[D] töltés közben - a töltés után beírhatjuk, melyik pályán akarunk kezdeni.

Ghostbusters II.

[SPACE]+[M]+[INS] töltés közben - ha elvesztettük a játékot, tölthetjük a következő szintet.

Total recall 128K

A ponttáblázatba írjuk be:"THE END IS NIGH". Ezután a játék közben az [ENTER] pályát vált.

The Untochables

A ponttáblázatba írjuk be:"HUMPHREY BOGART". Játék közben a [STOP] megnyomására betöltődik a következő szint.

Licence to kill

Lőjük ki a kilőhető tereptárgyakat (pl. géppuskafészek) és a [STOP] billentyűvel lépjünk ki a menübe. Újrakezdéskor a kilőtt tárgyak nem jelennek meg.

Néhány általános helyettesítő billentyűkombináció az elromlott belső botkormány helyettesítéséhez.

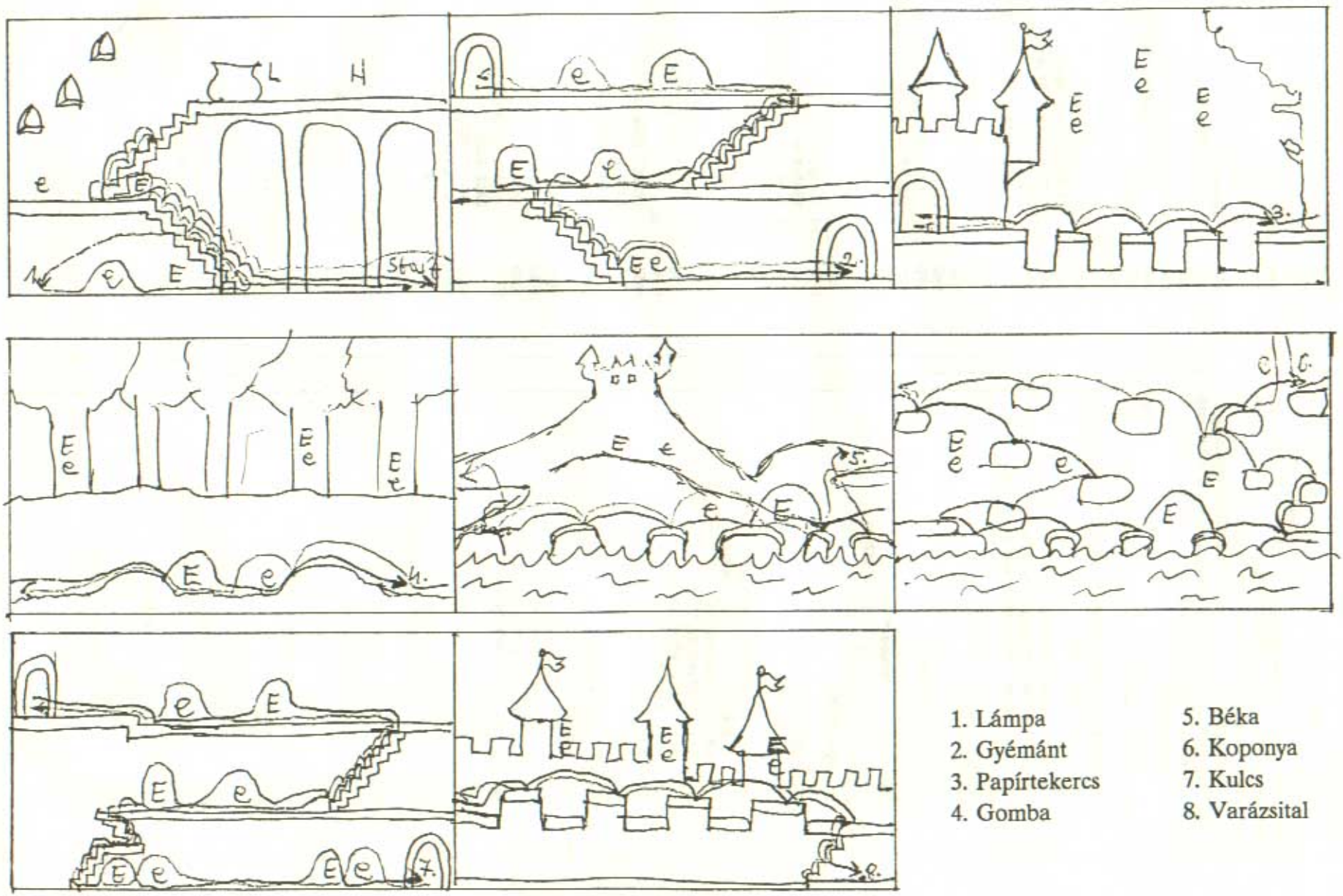
[V]+[B]+[ENTER] vagy bal [SHIFT]+[ALT]+[B] = joy jobbra
[Z]+[X]+[ENTER] vagy bal [SHIFT]+[ALT]+[X] = joy balra
[W]+[R]+[ENTER] vagy bal [SHIFT]+[ALT]+[C] = joy fel
[W]+[Q]+[ENTER] vagy bal [SHIFT]+[ALT]+[\] = joy le
[Z]+[N]+[M]=tűz

Ezek a kódok *angol nyelvű gépen* lettek kipróbálva!!!

Tellér András

E = Az ellenség tartózkodási helye — ahol ki lehet őt kerülni vagy átugrani — az odafelé vezető útnál
e = Ugyanaz, mint az „E", csak visszafelé
L = Lissa
H = Humphrey

OLLI & LISSA I.



1. Lámpa
2. Gyémánt
3. Papírtekercs
4. Gomba
5. Béka
6. Koponya
7. Kulcs
8. Varázsital

Labirintus

Ezt a cikket inkább a Programozástechnika rovatban kellene közölnünk. Az "elméleti rész" után látható program azonban nagyszerű játék!

Induláskor meghatározunk egy tetszőleges méretű téglalapot. Ezt (nem feltétlenül középen) kettéosztjuk egy fallal. Hogy a két terület mégse legyen egymástól elzárva, a falon kaput hagyunk. Most két téglalap alakú terület keletkezett, melyeket ugyanilyen módon tovább osztunk. Az eljárást addig folytatjuk amíg a terület szélessége le nem csökken a folyosó szélességére. Az algoritmus lényegén az sem változtat, hogyha véletlenszerűen, hol vízszintesen, hol pedig függőlegesen osztjuk ketté a területet. Jó szórakozást!

(Hsoft)

```
100 PROGRAM "LABIRINTUS"
110 ! (C) Hsoft 1990.
120 TEXT
130 INPUT PROMPT "Video X (5-42) ":VX
140 INPUT PROMPT "Video Y (4-27) ":VY
150 INPUT PROMPT "Bonyolultság (1-5) ":S
160 TEXT
170 LET S=8+8*2^(5-S)
180 SET VIDEO X VX
190 SET VIDEO Y VY
200 SET VIDEO MODE 1
210 SET VIDEO COLOUR 1
220 OPEN #101:"video:"
230 DISPLAY #101:AT 1+(27-VY)/2 FROM 1 TO VY
240 SET PALETTE 0,3,7,255
250 RANDOMIZE
260 DO
270   SET STATUS OFF
280   CLEAR #101
290   SET INK 1
300   LET MX=VX*16*2-1:LET MY=VY*9*4-1:
      LET K=RND(7)+1:LET A=RND(2)
310   LET MX=S*INT(MX/S)
320   LET MY=S*INT(MY/S)
330   PLOT 0,0;0,MY;MX,MY;MX,0;S,0
340   CALL XF(0,0,MX,MY)
350   LET V=0:CALL IR
360   WAIT 1
370 LOOP
380 DEF XF(X,Y,DX,DY)
390   NUMERIC O,L
400   IF DX>=2*S AND DY>=2*S THEN
410     LET O=X+S*RND(INT(DX/S)-1)+S:
            LET L=Y+S*RND(INT(DY/S))
420     LET L=Y+S*RND(INT(DY/S))
430     PLOT O,Y;O,L
440     PLOT O,L+S;O,Y+DY
450     IF A THEN
460       IF O-X>DY THEN
470         CALL XF(X,Y,O-X,DY)
480       ELSE
490         CALL YF(X,Y,O-X,DY)
500       END IF
510       IF X+DX-O>DY THEN
520         CALL XF(O,Y,X+DX-O,DY)
530       ELSE
540         CALL YF(O,Y,X+DX-O,DY)
550       END IF
560     ELSE
570       IF RND(10)>K THEN
580         CALL XF(X,Y,O-X,DY)
590       ELSE
600         CALL YF(X,Y,O-X,DY)
610       END IF
620       IF RND(10)>K THEN
630         CALL XF(O,Y,X+DX-O,DY)
640       ELSE
650         CALL YF(O,Y,X+DX-O,DY)
660       END IF
670     END IF
680   END IF
690 END DEF
700 DEF YF(X,Y,DX,DY)
710   NUMERIC O,L
720   IF DX>=2*S AND DY>=2*S THEN
730     LET O=Y+S*RND(INT(DY/S)-1)+S:
            LET L=X+S*RND(INT(DX/S))
740     LET L=X+S*RND(INT(DX/S))
750     PLOT X,O;L,O
760     PLOT L+S,O;X+DX,O
770     IF A THEN
780       IF DX<O-Y THEN
790         CALL YF(X,Y,DX,O-Y)
800       ELSE
810         CALL XF(X,Y,DX,O-Y)
820       END IF
830       IF DX<Y+DY-O THEN
840         CALL YF(X,O,DX,Y+DY-O)
850       ELSE
860         CALL XF(X,O,DX,Y+DY-O)
870       END IF
880     ELSE
890       IF RND(10)>K THEN
900         CALL YF(X,Y,DX,O-Y)
910       ELSE
920         CALL XF(X,Y,DX,O-Y)
930       END IF
940       IF RND(10)>K THEN
950         CALL YF(X,O,DX,Y+DY-O)
960       ELSE
970         CALL XF(X,O,DX,Y+DY-O)
980       END IF
990     END IF
1000  END IF
1010 END DEF
1020 DEF IR
1030   LET S2=S/2
1040   LET S3=S/4-4
1050   LET X=MX-S2
1060   LET Y=MY-S2
1070   SET KEY CLICK OFF
1080   SET LINE MODE 3
1090   SET INK 3
1100   PLOT X,Y,ELLIPSE S3,S3
1110   WHEN EXCEPTION USE EXIT
1120     DO
1130       SELECT CASE JOY(0)
1140       CASE 8
1150         CALL KZ(0,S2)
1160       CASE 4
1170         CALL KZ(0,-S2)
1180       CASE 2
1190         CALL KZ(-S2,0)
1200       CASE 1
1210         CALL KZ(S2,0)
1220       CASE ELSE
1230       END SELECT
1240     LOOP UNTIL V
1250   END WHEN
1260 END DEF
1270 DEF KZ(X1,Y1)
1280   PLOT X,Y,ELLIPSE S3,S3
1290   LOOK AT X+X1,Y+Y1:F
1300   IF F<>1 THEN LET X=X+X1+X1:
       LET Y=Y+Y1+Y1
1310   PLOT X,Y,ELLIPSE S3,S3
1320 END DEF
1330 HANDLER EXIT
1340   SET STATUS ON
1350   SET KEY CLICK ON
1360   IF EXTYPE<>9216 THEN TEXT :
       EXIT HANDLER
1370   SET LINE MODE 0
1380   PING
1390   PING
1400   PING
1410   PLOT MX-4,MY-4,PAINT
1420   LET V=1
1430   CONTINUE
1440 END HANDLER
```

Postafiók 334

Szmilkó József budapesti - az első számtól kezdve hűséges - olvasónk írja: *"A III/1-es számban van egy* LAKAT ALATT *című program. A programot begépeltem, sajnos, túl jól működik. Az lenne a kérésem, ha van egy olyan programjuk, amelyikkel a "lakatot" fel lehet nyitni, akkor azt a következő számban legyenek szívesek közölni, hogy a lakatra zárt programomat újra használhassam."*

Kedves Szmilkó József! Be kell vallanom, hogy kérésével igen nehéz helyzetbe hoz. Gondolja csak végig: a programot azért közöltük, hogy - mint ahogy a leírásban a szerző ki is fejti - a BASIC-ben programozóknak is legyen egy lehetőségük arra, hogy ha nem is százszázalékos biztonsággal, de megvédhessék programjaikat az illetéktelen beavatkozástól. Ha most közölnénk a "lakat" feltörésének módját, cserbenhagynánk azokat, akik esetleg már használják a megoldást. Gondolja csak el, ha a zárszaküzlet hátsó portáljánál álkulcsot árulnának az elöl eladott zárakhoz, lakatokhoz... Szerzőnk egyébként felhívta a bátor kísérletezők figyelmét, hogy őrizzék meg a program eredeti változatát, hogy később lehetőségük legyen a módosításra. Egyébként **mindig** érdemes minden programunkról biztonsági másolatot készíteni, különösen, ha valami durva beavatkozásra készülünk. (Idekívánkozik egy történet: A jelen sorok írója, valamikor a mikroszámítógépek - hazai - hőskorában, nagy merészen belekezdett egy *diszk editor* program elkészítésébe az egyik első itthon gyártott mikroszámítógépen. Akkor még ilyen program nem volt elérhető. Amikor végre hibátlanul lefordult az akkor már igencsak méretes programkezdemény újabb változata, amelyik már tartalmazta a közvetlen lemezre írást is, a program az első futtatásnál "kizárta önmagát": egy apró hiba miatt úgy módosította a lemez szerkezetét, hogy az operációs rendszer nem ismerte fel többé a floppyt. Egy apró programmódosítással a lemezt újra használhatóvá lehetett volna tenni, csakhogy az összes másolat a programról ugyanazon - az elrontott - lemezen volt... Sem lista, sem egyéb feljegyzés nem lévén, a teljes programot újra kellett alkotni a semmiből. Apró fintor Murphy részéről, hogy amint az új program működött, már nem volt rá szükség, hiszen "kizárta" a bezárt eredetit...) Tudom, hogy mindez nem csökkenti olvasónk lépcsőházi bosszúságát, aki sikeresen kizárta magát programjából. Több tanácsunk van:

1. Kísérelje meg megfejteni a "lakat" program által végrehajtott módosításokat (ehhez jó hasznát veszi a Flamingó könyvek sorozatban megjelent *ENTERPRISE, IS-BASIC ROM* című könyvnek, annak minden nyelvi és stilisztikai gyengesége ellenére (pl. a *belépés* (angolul eredetileg 'entry') szót következetesen *bejegyzés*nek kell olvasni, ha táblázatról van szó stb.).
2. Ahogy a lépcsőházba kiszorultak többsége lakatost hív, olvasónk is kérheti egy gyakorlott *programfeltörő* szakember (magyarul: *haker*) segítségét. Sajnos, a jelenlegi közállapotok szerint az ilyen szakemberek többsége nem *lakatosműhelyben* dolgozik, hanem *mackósként* keresi meg a betevőt; nem vállalva az ingyenreklám vádját, nem adunk meg címet.
3. Levelét továbbítjuk *Hsoft*-nak, aki talán vállal különmunkában *zárfelfeszítést* is...

Valyuch László 13 éves budapesti olvasónk igen nagy fába készül vágni fejszéjét: *"Az EP szövegszerkesztőjével nem vagyok megelégedve, ezért elhatároztam, hogy újat gyártok. Ehhez viszont kellene az Önök segítsége, annyoiban, hogy adnak néhány tippet, hogy egy új WP-nek mit kell tudnia. Várom válaszukat."*

Le a kalappal ez előtt a tettrekészség és lelkesedés előtt! Kedves Laci, nem akarunk elriasztani (tegeződjünk, jó?), de egy igazi jó szövegszerkesztő *egész* embert kíván, mégpedig nem is egyet (a programkészítés mai, ipari szintjén egy-egy szövegszerkesztőt 5-30 fős csapatok készítenek...). Azért néhány tippet adunk. Az első tippünkre ráéreztél, Laci: az első feladat egy pontos *specifikáció* készítése. ez tartalmazza, hogy mit is kell tudnia a programnak. Ez persze a lehetőségek, menet közben felmerülő nehézségek, újabb ötletek miatt változhat.

Egy korábbi számunkban már szerepelt egy zsörtölődés a WP hiányosságai miatt, ezt ajánljuk mint kiindulási alapot. Ehhez csak egy-két dolgot tudunk még hozzátenni. Tehát:

A szövegszerkesztőnek *teljes képernyősnek* kell lennie. Nem árt, ha gyorsabb, mint a WP! (Ez lehetséges.) Tudnia kell gyorsabban is mozognia, mint a WP "képernyő képernyő után" ballagása. Egyáltalán, ha a felhasználó tovább pöccintette a *SHIFT-kurzor-le* vagy *SHIFT-kurzor-fel* kombinációt, felesleges a megkezdett képernyőkiírást befejezni, azonnal tovább lehet menni a következő képernyőre. Ehhez igen virtuózan kell kezelni a kiírást és a billentyűzetet.

A program nem akadhat ki 16 kilobájtnyi szövegtől (sem a szöveg a programtól!), de az sem megoldás, ha ezt a korlátot megemeljük egy nagyobb értékre. A programnak meg kell tudnia oldani tetszőlegesen nagy szöveg kezelését, mégpedig úgy, hogy tetszés szerint lehessen előre-hátra "rohangálni" benne (ehhez valamilyen direkt lemezkezelésre lesz szükség).

Tudnia kell a magyar ékezetes karakterek kezelését. Ez azt is jelenti, hogy a szavak új sorba törésénél a magyar betűket is betűknek kell elfogadnia. Az Enterprise sajátosságai miatt meg kell engedni, hogy ezek az ékezetes betűk az egyes felhasználóknál eltérően legyenek definiálva, mind karakterkód, mind pedig billentyűkiosztás szempontjából (senki sem akarja az eddig megírt trilógiáját átgépelni, sem pedig újra megtanulni gépelni). Ehhez célszerűen valamiféle táblázatokat kell használni. Ezt a beállítást (és még sok más egyedi vonást) egy külön fájlban (úgynevezett *konfigurációs fájlban*) kell elmenteni, hogy ne kelljen minden betöltésnél újra installálni a programot.

Nagyon fontos a fájlkezelés: a minimum az, hogy a program mindig mentést készít a szöveg eredeti változatáról. Ha elszáll a program, vagy valami hülyeséget csinál a felhasználó, még mindig megvan az előző szövegváltozat. Ennek az eljárásnak a vázlata a következő: Amikor megnyitunk egy szöveget, a program abból csak olvas, írni egy úgynevezett *munkafájlba* ír. Ha el kell menteni a szöveget, először lezárjuk a munkafájlt, az eredeti fájlt átnevezzük (pl. a fájlnevet megtartjuk, a kiterjesztést pedig .BAK-re változtatjuk). Csak ezután kapja meg a munkafájl az eredeti fájlnevet és kiterjesztést. Ha így járunk el (*minden* szövegszerkesztő így csinálja), akkor gyakorlatilag lehetetlen, hogy elvesszen a szöveg; leszámítva, persze, a szándékos "rongálást". Az is fontos, hogy ne lehessen kilépni a programból úgy, hogy az meg ne kísérelné udvariasan felhívni a figyelmünket a megváltozott szöveg elmentésére.

Tudnia kell más forrásból származó szöveget (legalább sima ASCII fájlt) beolvasni és ilyen formában szöveget kitenni. Tudnia kell egy szövegfájlt az éppen feldolgozott szövegbe bemásolni, és esetleg a szövegnek csak egy részét kimenteni. Ezt persze nem kell ostoba módon a képernyőn is mutatnia, mert akkor ugyanolyan lassú lesz, mint a WP.

Elengedhetetlenek a kiegészítő funkciók, úgymint szövegkeresés és -csere, ezek nélkül egy szövegszerkesztő nem szövegszerkesztő. Tudni kell legalább a legfontosabb vezérlőkaraktereket vagy -funkciókat kezelni, minium a lapdobást.

A program kezelésére leginkább az úgynevezett *legördülő menüs* megoldás a megfelelő (lásd a III/1. és a mostani szám *A külső is számít* című cikkét). Nem merem meg sem említeni, hogy ugyanezt a menürendszert egérrel is kezelni kellene tudni. A jobb szövegszerkesztőkben van egy úgynevezett UNDO ("andú", magyarul talán VISSZA AZ EGÉSZ) funkció, amelyik az éppen elkövetett baklövést (a teljes szöveg letörlése stb.) képes visszacsinálni, esetleg több lépésre visszamenőleg...

Nagyjából ennyi, ami a legfontosabb. Természetesen az apró részletek szintjén még számtalan ötletet tudunk adni. Érdemes esetleg egy-két korszerű szövegszerkesztőt megnézni (Word, Wordperfect, Wordstar stb.).

Ja, az utóirat: *"Egy új WP-vel is be lehet nevezni a programozási versenybe?"* Azt hiszem, hogy egy színvonalas szövegszerkesztőt a zsűri versenyen kívül is értékelne. Leveled feladásakor ugyan az eredetileg kiírt programozási verseny beküldési határideje már lejárt. Azt hiszem azonban, hogy a magadnak kitűzött feladat megvalósítása nem egy-két nap, nem is egy-két hét, de még csak nem is egy-két hónap türelmes munkát igényel. Jó, ha a következő versenykiírásra elkészülsz... A szerkesztőség drukkol neked!

Az olvasóknak jó nyaralást és kellemes Enterprise-os perceket kíván:

A felelős szerkesztő

Az eddig beérkezett kisszámú pályamunka miatt folytatódik a . . .

PROGRAMOZÁSI VERSENY!!!

Szerkesztőségünk programozási versenyt hirdet az alábbi két kategóriában:

1.,**DEMO**

Olyan programokat várunk, amelyek

- minél jobban kihasználják a Nick és a Dave lehetőségeit,
- az Enterpress-t propagálják: szerepel az *újság neve*, a *postacímünk* (1399 Budapest, Pf. 701/334), a *kiadó neve* (Mátrix KFT.), *címe* (8000 Székesfehérvár, Honvéd utca 8.) és *telefonszáma* ((06-22) 29-888
- szellemes (magyar illetve angol nyelvű) mondatokat úsztatnak.

Demo programot egyénileg és csoportosan is lehet készíteni. A programnyelvet mindenki maga választhatja meg, és olyan segítséget használ, amilyet csak akar. A lényeg tehát a minél látványosabb program!

2., **BASIC JÁTÉKPROGRAM**

Olyan Basic nyelvű játékprogramokat várunk, amelyek

- a "PROGRAM 0"-ba befférnek,,
- minél gyorsabbak,
- alapötletük, történetük (lehetőleg) újszerű.

A programot egyének és csoportok is készíthetik. Nem nevezési feltétel, de nem baj, ha a program Zzzip-pel fordítható. A játékprogram fajtája bármilyen lehet: kalandjáték, logikai játék stb. Azok a programok számíthatnak sikerre, amelyek grafikát és zenét is használnak. A program kritikus részein gépi kódú rutinokat (CODE sorok, ROM hívás) is szabad használni, ezt azonban nem szabad túlzásba vinni! A legjobb programot közölni fogjuk.

Az új beküldési határidő: 1992. július 1.

A Mátrix KFT. szerkesztőségünkkel együtt kategóriánként az alábbi díjakkal jutalmazza az első három helyezettet:

1. díj: 3000 Ft
2. díj: 2000 Ft
3. díj: 1000 Ft

A programokat postacímünkre küldjék:
ENTERPRESS, 1399 Budapest, Pf. 701/334.

mikrovilág

A Mikrovilág minden számában két oldalnyi terjedelemben foglalkozik ENTERPRISE-os témákkal.

Az ENTERPRESS előző számai korlátozott példányszámban még megrendelhetők a kiadó címén (MÁTRIX Kft. 8000 Székesfehérvár, Zichy liget 10.), vagy megvásárolhatók a Műszaki Könyváruházban (Bp. VI. ker. Liszt F. tér 9.) és a Fókusz Könyváruházban (Bp. VII. ker. Rákóczi út 14.).

Tisztelt Olvasóink!

Arra kérjük Önöket, hogy utórendeléseiket és megrendeléseiket ne a szerkesztőség, hanem a kiadó (Mátrix Kft.) címére küldjék, mert így sokkal gyorsabban juthatnak hozzá kedvenc lapjukhoz. Köszönjük!

A szerkesztőség

HIRDETÉSFELVÉTEL

Az apróhirdetések ára: 1 Ft karakterenként. A szöveget és a befizetést igazoló nyugtát (rózsaszínű postautalványon) az alábbi címre kérjük feladni:

MÁTRIX Kft.
ENTERPRESS
8000 Székesfehérvár,
Zichy liget 10.

Megjegyzés: a nem saját fejlesztésű szoftverek másolásával foglalkozó üzletelők hirdetéseit nem áll módunkban elfogadni.

Apróhirdetések

Eladók ENTERPRISE-hoz joystick-illesztők (autofire, infra joystick is megy). Utánvéttellel 970 Ft/db.
Ábel Imre, 1116 Budapest, Fegyvernek u. 103.

Komplett gyári ENTERPRISE floppy drive eladó. Kontroller, tápegység, EXDOS beépítve, 16.000,-Ft.
Gitye György, 4027 Debrecen, Gyöngyösi u. 28. Tel.: (52) 11-067

ENTERPRISE turbósítás 6 MHz-re, működő perifériákkal, 3.500,-Ft; 320 KB-ra bővítés 3.000,-Ft. Együtt 6.000,-Ft!
Bozai Gábor, 8000 Székesfehérvár, József A. u. 70/A fsz. 1.
Tel.: (22) 10-665

AZ ENTERPRESS ALAPÍTVÁNY SZÁMLASZÁMA
Postabank 299-98922/042-01966/4004

LEVELEZÉS

A géppel kapcsolatos témákban levelezne:
Bognár Balázs, 9443 Petőháza, Bartók Béla u. 11.
Czibere Lajos, 4027 Debrecen, Füredi út 1. III/12.
Feczkó Krisztián, 8200 Veszprém, Ányos u. 1/3. Tel.: (80) 29-493
ifj. Márföldi Béla, 4400 Nyíregyháza, Új u. 28.
Tóth István, 7305 Mecsekpölöske, Szegfű u. 1.

KLUB

Budapesti Enterprise klub VSzM közösségi ház
Budapest, XI. ker. Fehérvári út 120.

* * * **ENTERPRESS** *Kéthavilap az Enterprise számítógépek felhasználóinak* * III. évfolyam 3. szám * 1992. május—június * * * Kiadja a MÁTRIX Kft., Székesfehérvár * Felelős kiadó: Juhász István ügyvezető * A kiadó címe: MÁTRIX Kft., 8000 Székesfehérvár, Zichy liget 10. Telefon:(22) 29-888 Telefax:(22) 29-888 * * * Felelős szerkesztő: Ujlaki László * A szerkesztőség tagjai: Hajnal Csaba főszerkesztő, Devilsoft, JOVI, Ari Sándor, Bozai Gábor, Haluska László, Loiasoft, Mészáros Gyula * A szerkesztőség csak levélben érhető el! A cím: **ENTERPRESS, 1399 Budapest, Pf. 701/334.** * Technikai szerkesztő: Szapper László * * * Nyomja a Duna Print Kft., Dunaújváros * Felelős vezető: Farkas István * * * **HU ISSN 0866-1820** * * * Terjeszti a Magyar Posta * Előfizethető a HELIR, 1900 Budapest, vagy a MÁTRIX Kft. címén * Előfizetési díj egy évre 294 Ft, fél évre 147 Ft. * * * Következő számunk júliusban jelenik meg. * * * Az ENTERPRESS-ben közreadott információk célja az, hogy segítsék, tudnivalókkal lássák el a gép felhasználóit. A közölt programokat, kapcsolási rajzokat, leírásokat mindenki szabadon felhasználhatja, de tilos azokat a kiadó írásbeli engedélye nélkül másolni, terjeszteni. * A szerkesztőség kéziratokat nem őriz meg, és nem küld vissza; továbbá külön levelezésre nincs módja. * * * **ENTERPRESS © 1992 MÁTRIX Kft.** * * *