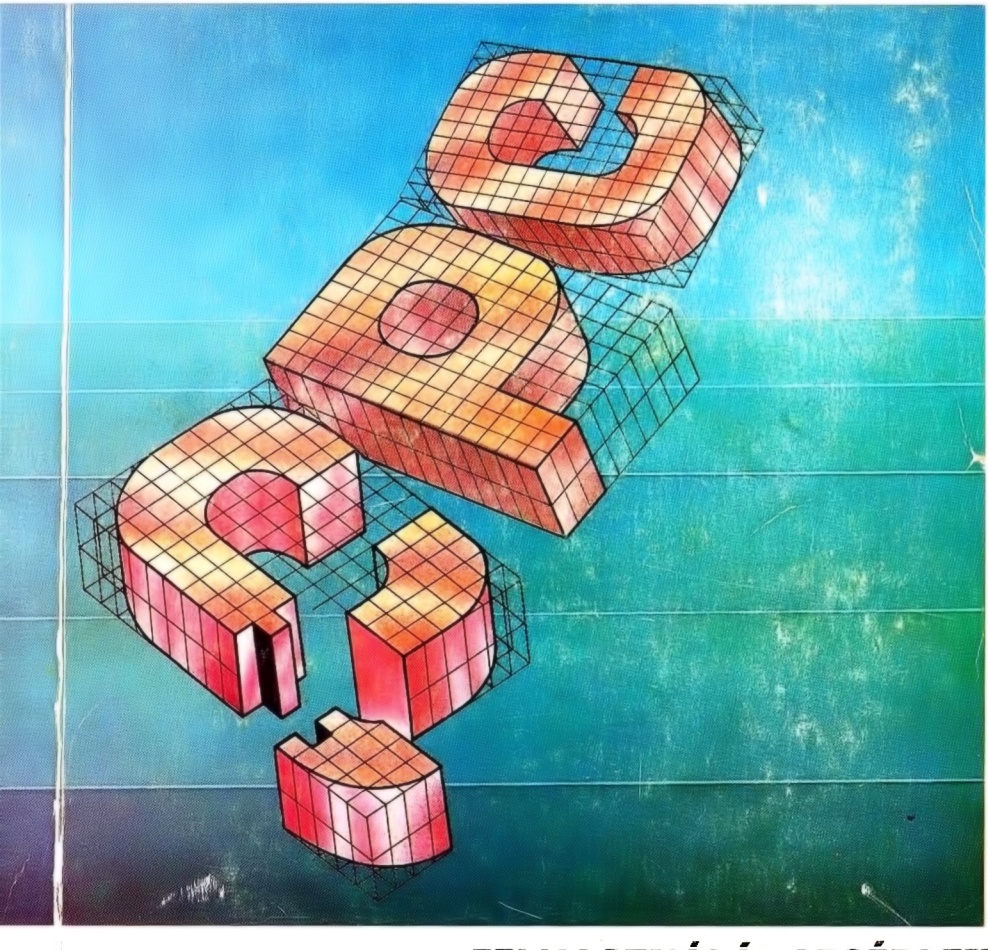
Dr. FUTÓ ISTVÁN

CPC BASIC
HÁROM SZINTEN



FELHASZNÁLÓI SEGÉDLET

198 Ft

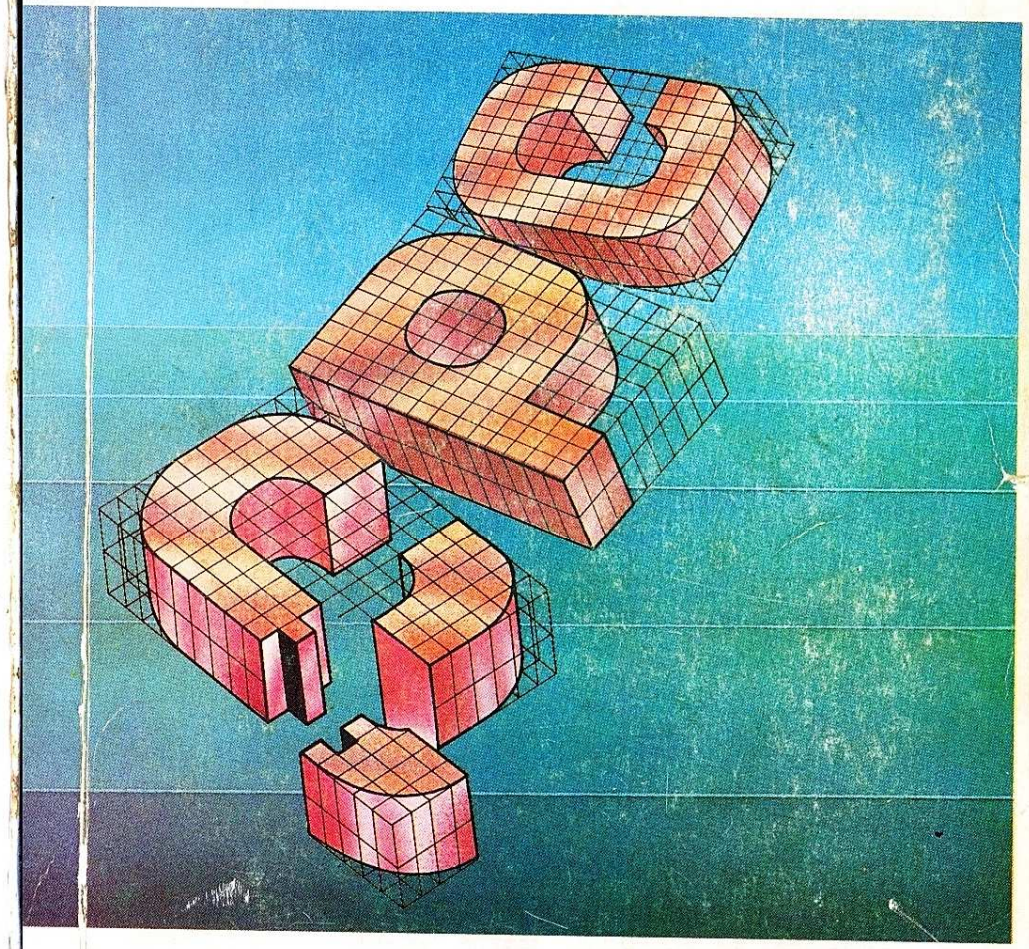
Ahány személyi számítógép, annyi BASIC nyelvjárás – ferdíthetnénk a közmondást. A sokféle BASIC-ből is kiemelkedik a Schneider/Amstrad CPC gépcsalád nyelvi gazdagsága és igényessége. Ez is lehetett annak az oka, hogy e gép nyerte el 1985-ben az ÉV GÉPE megtisztelő elnevezést a maga kategóriájában.
A hazai érdeklődők és géptulajdonosok, egyszóval az Olvasók e könyvből nemcsak a kitűnő gép tulajdonságaival, hanem egy szokatlanul közvetlen hangú és mégsem csak kezdőknek szóló BASIC leírással ismérkedhetnek meg.

BUDAPEST
MŰSZAKI KÖNYVKIADÓ

CPC BASIC HÁROM SZINTEN

Dr. FUTÓ ISTVÁN

CPC BASIC HÁROM SZINTEN



FELHASZNÁLÓI SEGÉDLET

198 Ft

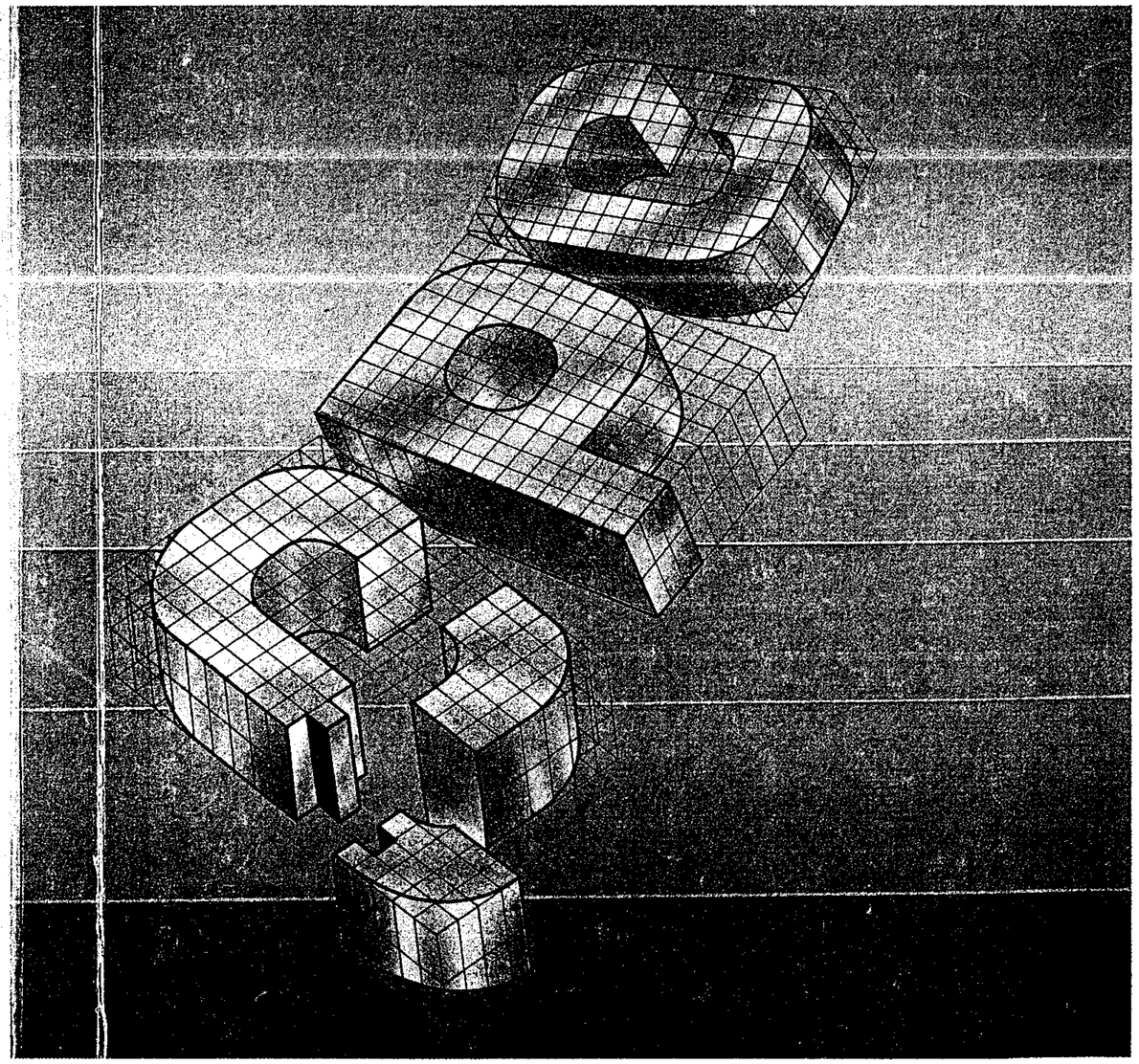
Ahány személyi számítógép, annyi BASIC nyelvjárás – ferdíthetnénk a közmondást. A sokféle BASIC-ből is kiemelkedik a Schneider/Amstrad CPC gépcsalád nyelvi gazdagsága és igényessége. Ez is lehetett annak az oka, hogy e gép nyerte el 1985-ben az ÉV GÉPE megtisztelő elnevezést a maga kategóriájában.
A hazai érdeklődők és géptulajdonosok, egyszóval az Olvasók e könyvből nemcsak a kitűnő gép tulajdonságaival, hanem egy szokatlanul közvetlen hangú és mégsem csak kezdőknek szóló BASIC leírással ismerkedhetnek meg.

BUDAPEST
MŰSZAKI KÖNYVKIADÓ

CPC BASIC HÁROM SZINTEN

Dr. FUTÓ ISTVÁN

CPC BASIC
HÁROM SZINTEN



FELHASZNÁLÓI SEGÉDLET

Dr. Futó István

CPC BASIC három szinten

Műszaki Könyvkiadó
Budapest, 1987

Lektorálta: Lengyel Tamás

ETO: 519.68

ISBN: 963 10 7279 7

T A R T A L O M

Második szint

Harmadik szint

E l ő s z ó

Az utóbbi évek sikertörténeteinek egyik főszereplője az angol Amstrad cég, amely, bár eredetileg csak híradástechnikai cikkek gyártásával foglalkozott, 1984-ben betört a számítógép-gyártó óriások közé. A **CPC 464, 664 és 6128-as** gépcsalád, megtörve a többéves COMMODORE egyeduralmat, 1985-ben elnyerte az év számítógépe címet az otthoni gépek kategóriájában. A gépeket német nyelvterületen a Schneider cég forgalmazza a saját neve alatt. A CPC-k a nyugateuropai országokban a sikerlista első helyeit mondhatják magukénak. Amstradék diadalútja tovább tart: magukba olvasztották a Sinclair céget gépeivel együtt, piacra dobták a JOYCE (PCW 256 és 512) nevű számítógép-csomagot, amely szövegszerkesztő programmal, lemezegységgel, monitorral és nyomtatóval együtt a legolcsóbb CP/M operációs rendszer alatt működő komplett rendszer. E könyv befejezésének pillanatában jelent meg az IBM kompatibilis Amstrad gép: ez olcsóbb a legolcsóbb hong-kongi másolatoknál és mégis olyan szolgáltatásokat nyújt, amelyek az eredeti PC-n csak többletköltségek árán érhetők el.

Mi a titka a CPC-k sikerének? Az otthoni gépek kategóriájában **egy alapgép áráért teljes rendszert** kap a vásárló:

- 64 vagy 128 Kbyte RAM kapacitás, korlátlanul bővíthető ROM-tár: memórialapozással 4 Mbyte címzése lehetséges,

- rendkívül kényelmes billentyűzet, különálló tízes számblokkal és 32 szabadon programozható funkcióbillentyűvel,

- beépített kazettás tároló vagy lemezmeghajtó. Az utóbbi kapacitása 2*180 Kbyte, sebessége egyedülálló,

- egyszínű zöld vagy RGB színes monitor, amely stabilan jeleníti meg a soronkénti 80 karaktert, megbirkózik a a gép 640*200 pontos grafikus felbontásával, 27 színével, és a 8 szabadon definiálható ablakkal,

- CPC 664 és 6128 esetében CP/M operációs rendszer: kulcs a gép professzionális alkalmazásához (külön lemez-

meghajtóval a CPC 464-en is fut a CP/M),

- a rendszerlemezen található Dr LOGO interpreter olyan strukturált programozási nyelv, amely grafikai parancsaival a legfiatalabb korosztállyal kedveltetheti meg a programozást és a logikus gondolkodást. A komolyabb korosztály sem becsülheti le például a LOGO 16 számjegy pontosságú számábrázolását,

- ellentétben más gépekkel (pl. COMMODORE), a CPC-n minden szabványos: CENTRONICS nyomtatóinterfész, karakterkódok az ASCII szabvány szerint, ipari szabványként is emlegetett Z80-as CPU, sztereó hangkimenet ...

A CPC interpretere a MICROSOFT BASIC mintájára készült, de alkotói néhány mesterfogással a kategória egyik leggyorsabb változatát hozták létre. A CPC BASIC-et nemcsak gyorsasága, hanem parancsokban való gazdagsága is jellemzi: a gép hardverének minden tulajdonságát egyedi parancsok, utasítások és függvények támogatják, PEEK-re POKE-ra gyakorlatilag nincs szükség. Az elmondottak ellenére a CPC BASIC a legmesszebbmenőkig szabványos, aki más gépen tanult programozni, nagyon gyorsan hozzászokik és megszereti. Mindez természetesen fordítva is érvényes: CPC-ről akár IBM-re is egyszerű az átállás.

Könyvünk célja, hogy a gép használóját végigvezesse a CPC BASIC rejtelmein. A három szintre való felosztás lehetővé teszi, hogy a különböző előképzettségű felhasználók tudásszintjük alapján találjanak érdeklődési körüknek megfelelő új ismereteket:

Az első szint a gép üzembehelyezésétől kezdve az első lépéseken keresztül elvezet a BASIC alapjainak elsajátításáig.

A második szint tartalmazza mindazt, amivel a CPC BASIC többet nyújt a hasonló kategóriájú gépek BASIC-jénél: grafika és hang, áttervezhető billentyűzet és karakterkészlet, multitasking és az ablakok programozása. CP/M és file-kezelés stb.

A harmadik szint már minőségi ugrás az első kettőhöz képest: a CPC gépi kódú programozásának lehetőségeivel ismerteti meg az Olvasót. Kitér a CPC tárfelosztására, képernyőszervezésére, bemutatja a fontosabb rendszerrutinokat stb.

A függelék mindhárom szinthez szolgáltat hasznos információt ábrákon és táblázatokon kívül a CPC BASIC összes kulcsszavának rövid ismertetése található benne.

A könyvben szereplő rövid programokban a BASIC kulcsszavak kisbetűvel állnak (pl. print), ezeket a programokat kötőjellel elválasztott - megjegyzések kísérik. A megjegyzéseket természetesen nem kell beírni a programmal együtt. A hosszabb programok nagybetűs kulcsszavakat tartalmaznak (pl. PRINT), ezek a programok egyenesen egy CPC-ből kerültek át a szövegszerkesztőbe, amellyel ez a könyv is készült, így garantáltan hibamentesek.

Azért, hogy a számítástechnikában már jártasabb Olvasókat ne untassák a kezdők számára ismeretlen szakkifejezések magyarázatai, a szövegben a kevésbé ismert szavakat * jel jelöli, magyarázatuk a függelékben található.

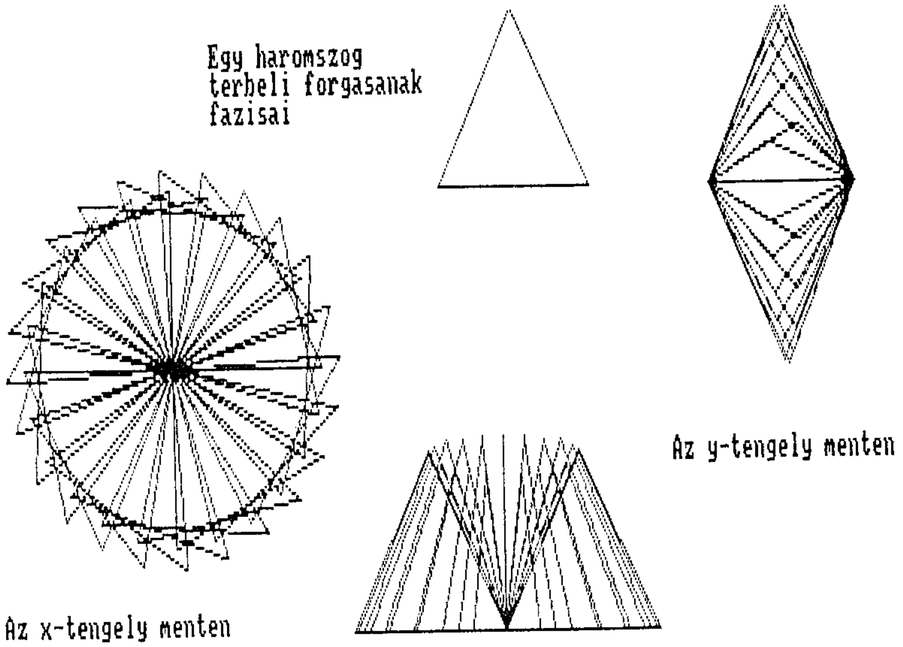
Mivel a CPC 664 és 6128 BASIC-je néhány utasítással többet nyújt, mint a 464-es BASIC-je, az eltérésekre a ♦ jel hívja fel a figyelmet.

A könyv szöveges részében a BASIC 'PARANCSSZAVAK' nagybetűvel, aposztrófok között állnak. Szintén aposztrófok közé kerülnek a 'változónevek' is. Az egyes billentyűk azonosítását szögletes zárójelbe foglalt [feliratuk] segíti elő.

▶▶▶ Ez a jelölés különösen fontos dolgokat emel ki. Feltétlenül olvassa el még akkor is, ha különben átugorná a fejezetet!

A könyv és a CPC használatához
sok sikert kíván

a szerző



E l s ő s z i n t

Bár az Első szint a kezdők szintje, a BASIC-ben már jártas Olvasó is feltétlenül lapozza át! Olyan dolgokat tudhat meg a CPC BASIC-járól, amelyek más gépen elképzelhetetlenek. Csak ízelítőnek: a változónevek **40** karakter hosszúak lehetnek és az interpreter minden karaktert figyelembe vesz.

Az első szint önállóan futtatható programjai:

SZAMOK ATLAGA	1.5.4
OSZTALYZAT SZAMLALAS	1.5.5
MAGYAR-ANGOL SZOTAR	1.5.8
PRIMSZAMOK	1.5.9.2
LEGNAGYOBB KOZOS OSZTO	1.5.9.2
TORPEDO	1.5.9.2
FENYUJSAG	1.5.9.4
KEVERT BETUK	1.5.9.4

1.1 A CPC ÜZEMBEHELYEZÉSE

1.1.1 Az első bekapcsolás

A számítógép - típusától függetlenül - két dobozban foglal helyet. A kisebbik, hosszúkás doboz tartalmazza a CPC központi egységét, billentyűzettel és beépített magnetofonnal (CPC 464) vagy lemezmeghajtóval (CPC 664 és CPC 6128). A nagyobbik doboz tartalmazza a színes vagy egyszínű zöld monitort*.

A gép üzembehelyezése a két doboz tartalmának kicsomagolásával kezdődik és a következő lépésekkel folytatódik:

- Helyezze a monitort a jövendő munkaasztalra. Az asztal helyét érdemes úgy kiválasztani, hogy direkt fény ne essen rá, mert az a monitorról visszatükrözve zavarhat a munkában.
- Helyezze a központi egységet a monitor elé úgy, hogy az kényelmesen a keze alá essen.
- A monitorból két kábel vezet ki: az egyiken egy kisebb csatlakozó van, ezen kapja a CPC az 5 V egyenáramot, a másik egy 6 pólusú DIN csatlakozóban végződik, ezen megy ki a kép a CPC-ből a monitorba. A csatlakozókat úgy alakították ki, hogy még csak véletlenül sem lehet felcserélni őket.
- Csatlakoztassa először az 5 V-ot adó kábelt a gép hátulján levő **5 V DC** feliratú hüvelybe, majd a 6 pólusú DIN dugaszt a mellette levő **MONITOR** feliratú hüvelybe. A CPC 464 összeállítása ezzel be is fejeződött.
- A CPC 664 és CPC 6128-as típusnál a központi egységből is vezet ki egy kábel a gép hátulján. Ezen kapja a beépített lemezmeghajtó a működéséhez szükséges 12 V egyenáramot. A kábelt a monitor elején levő **12 V DC** feliratú aljzatba kell csatlakoztatni.

- Az így összeállított rendszer a monitorból kapja a tápfeszültséget. Ehhez a monitor hátulján feltekercselt villásdugót tekerje le és csatlakoztassa egy hálózati 220 V-os konnektorba.
- Kapcsolja be a monitort: az elején található **POWER** feliratú kapcsolót kell ütközésig benyomni.

Kapcsolja be a központi egységet:

♦ CPC 464 és 664 esetében a gép jobb oldalán található csúszókapcsolót (felirata POWER) kell az **ON** állásba tolni.

```
Schneider 64K Microcomputer  (v1)

©1984 Amstrad Consumer Electronics plc
           and Locomotive Software Ltd.

BASIC 1.0

-------------------------------------

Amstrad 64K Microcomputer  (v2)

©1984 Amstrad Consumer Electronics plc
           and Locomotive Software Ltd.

BASIC 1.1

-------------------------------------

Schneider 128K Microcomputer  (v3)

©1985 Amstrad Consumer Electronics plc
           and Locomotive Software Ltd.

BASIC 1.1

-------------------------------------

Ready
■
```

♦ CPC 6128 esetében ez a kapcsoló a központi egység hátsó felén jobb oldalt található. A bekapcsolás természetesen ugyanúgy történik: **ON** állásba csúsztatással.

– Ha minden rendben ment, a központi egység tetején levő piros LED (fénykibocsátó dióda) jelzi, hogy a számítógép üzembeállít.

– Kényelmes munkához állítsa be a monitoron megjelenő kép fényességét a BRIGHTNESS feliratú gombbal. Zöld monitorokon a kép kontrasztjának (CONTRAST) a módosítására is van lehetőség.

1.1.2 A CPC bejelentkezik

A rendszer üzemel, a képernyőn az ábrán látható feliratok egyike jelenik meg: CPC 464-nél az első, CPC 664-nél a második, CPC 6128-nál a harmadik. A felirat az alábbiakat közli:

1. sor: a gyártó cég neve (Schneider vagy Amstrad), a CPC operatív tára*: 64 vagy 128 Kbyte RAM*,

a gép verziószáma: v1, v2, vagy v3.

2.,3.sor: a szerzői jog (copyright) az Amstrad cégé (ők tervezték a gépet) és a Locomotive szoftverházé (ők írták a gép operációs rendszerét* és a BASIC interpretert*).

4. sor: A gép BASIC interpreterének verziószáma: CPC 464 esetében 1.0, CPC 664 és CPC 6128 esetében 1.1.

5. sor: Ready = kész. A BASIC interpreter készen áll arra, hogy minden értelmes parancsunkat végrehajtsa.

6. sor: ■ (a kurzor, vagy magyarítva "helyőr") mutatja azt a helyet a képernyőn, ahol megjelenik majd a következő lenyomott billentyűnek megfelelő betű vagy jel.

A billentyűzetről részletesen később lesz szó, most talán próbálja beírni a nevét, egyelőre ékezetes betűk nélkül. A CPC ezt nyugodtan tűri, egészen addig, amíg Ön a tudomására nem hozza. Ha készen van a beírással, myomja meg az [ENTER] feliratú billentyűt! Az eredmény valószínűleg a következő lesz:

```
Ready                 – ez állt a BASIC felirat alatt
futo istvan           – ezt írta Ön és lenyomta az [ENTER]-t
Syntax error          – ezt válaszolta a CPC
Ready                 – újra készen áll
■                     – ide lehet írni
```

A lesújtó válasz: **'Syntax error'** azt jelenti, hogy a CPC nem értette meg, mit is akarunk tőle. Ön is mindig ezt a választ kapja majd, ha nem helyesen ad be egy parancsot, vagy értelmetlen dolgot ír be. A fenti cselekedetünk ezek szerint nem volt a CPC számára értelmezhető. Ha azt akartuk volna, hogy jegyezze meg a nevünket, azt másképpen kellett volna vele közölni, de ez egy későbbi fejezet témája.

1.1.3 A bemutató program

Hogy egyszerű módon meggyőződhessen gépe csodálatos képességeiről, vegye elő a géphez mellékelt kazettát vagy lemezt.

♦ CPC 464-hez kazetta jár. Nyissa ki a beépített magnetofon fedelét a STOP/EJECT feliratú gombbal. Tegye be a kazettát úgy, hogy az első oldal nézzen felfelé, majd ha szükséges, tekercselje vissza a szalagot a REW gombbal. Ezután nyomja le a [CTRL] feliratú billentyűt majd azt lenyomva tartva a kis [ENTER] billentyűt. A képernyőn az alábbi felirat jelenik meg:

```
RUN"                          - ezt írta ki a [CTRL] + kis [ENTER]
Press PLAY then any key: ■ - ezt válaszolta a CPC
```

Parancsunkkal utasítottuk a CPC-t, hogy töltse be és indítsa el (RUN) a szalagon található első programot.

A válasz fordítása: nyomja le a magnó gombjai közül a PLAY feliratút majd azután a billentyűzeten egy tetszőleges billentyűt. Amint ez bekövetkezik, a képernyőről eltűnik a kurzor és elindul a magnetofon. A program betöltése kb. 5 percig tart, de közben a képernyőn követhetjük a folyamatot: egy idő után megjelenik a következő felirat:

```
Loading WELCOME 1 block 1
```

A közlés értelme: éppen töltődik a WELCOME 1 nevű program első blokkja. A szalag a programot nem egyhuzamban, hanem darabokra bontva, blokk*-ok formájában tárolja, a CPC betöltés közben visszajelzi, hogy hányadik blokknál tart éppen. Hosszabb programoknál, ha egyszer megjegyeztük az utolsó blokk számát, elég a képernyőre pillantani, hogy megtudjuk, van-e még egy kávéra való időnk a betöltés végéig. De előbb-utóbb minden program betöltődik, így a WELCOME 1 is, és magától elindul. A magnetofon leáll, megnyomhatja a STOP/EJECT gombot. Ha elege volt a programból, az [ESC] feliratú billentyűt kell kétszer egymás után lenyomni, hogy az leálljon és Ön viszontláthassa a már ismerős 'Ready' feliratot. A kazettát megfordítva és visszatekercselve a fenti módszerrel betöltheti és elindíthatja a WELCOME 2 programot is.

♦ CPC 664-hoz egy, a CPC 6128-hoz pedig két lemez jár. A lemezegység működéséről később lesz szó, most elég megjegyezni, hogy a lemezek mindkét oldalán lehet programokat tárolni, és a meghajtóba* a lemezt mindig a kívánt programot tartalmazó

oldalával felfelé kell behelyezni. A behelyezés úgy történik, hogy a meghajtó vízszintes nyílásába addig csúsztatja befelé a lemezt, amíg azt a meghajtó mechanikája elkapja és egy kattanással a helyére teszi. A lemez vége így is kilátszik a nyílásból. CPC 664 esetében az egyetlen lemezt A-oldallal felfelé, CPC 6128 esetében pedig a **Side 4** feliratú lemezt (felirattal felfelé) helyezze a meghajtóba és billentyűzze be a következő parancsot:

```
run"disk" [ENTER]  - írja be a szöveget  majd nyomja le az
                     [ENTER] feliratú billentyűt! Az idéző-
                     jelet a  [SHIFT] és a  [2]  billentyű
                     együttes lenyomásával kapja meg.
```

A meghajtó zümmögni kezd, lámpája fel-felvillan, és szinte azonnal betöltődik és elindul a program. Ha megunta a végnélküli programot, nyomja le egyszerre a [SHIFT], [CTRL] és [ESC] billentyűket: a CPC a bekapcsoláshoz hasonló módon jelentkezik. Ha véletlenül másik lemezt helyezett a meghajtóba, vagy nem a megfelelő oldal van felfelé, a CPC nem találja a keresett programot és hibajelzést ad:

```
DISK .     not found      - a DISK .  nevű programot  nem
                            találja.
```

1.2 AZ ELSŐ LÉPÉSEK

1.2.1 Számoljunk a CPC-vel!

A bemutató programokat megtekintve ideje kipróbálni, hajlandó-e a CPC Önnek is engedelmeskedni. Csak tessék! Kapcsolja be a gépét. Ha már bekapcsolt állapotban van, akkor először kapcsolja ki, hogy tiszta lappal indulhasson! A CPC a már ismert módon jelentkezik:

```
Ready          - a BASIC készen áll parancsokat végrehajtani
■              - ide lehet írni
```

A billentyűzetről részletesen a következő fejezetben lesz szó, egyelőre használja úgy, ahogy egy írógépet használna, az esetleges elütéseket a **[DEL]** billentyű lenyomásásával javíthat-

ja: a kurzor visszalép és törli az utoljára beírt betűt vagy számjegyet.

▶▶▶ Az írógéptől eltérően, a nullának és az egyesnek is van saját billentyűje! A nulla billentyűje a [9]-estől jobbra van: a billentyűn áthúzott karikáról ismerhető fel, így jelenik meg a képernyőn is (ø). Nyomtatásban és így könyvünkben is a nulla keskenyebb mint az O-betű: O.

Írja be a következőket:

```
print 12 + 35
```

majd nyomja meg az [ENTER] billentyűt! Ahogy azt elvárta, gépe megadja a helyes eredmény:

```
 47
Ready
■
```

Nézzük meg még egyszer pontosan az előbbi beírást! Először a gépet utasítottuk arra, hogy mit tegyen: **'PRINT'** = írd ki azt, ami e szó (parancs*) után következik. Hogy ezt megtehesse, természetesen először ki kellett számítania az eredményt. A parancsot végrehajtva (az eredményt kiszámítva és megjelenítve) a gép új parancsra vár, amit a **'Ready'** üzenet és a kurzor is mutat. A gép számára az **[ENTER]** billentyű leütése jelezte, hogy most rajta a sor, hajtsa végra a parancsot. Amíg az [ENTER] billentyűt nem nyomjuk le, szabadon javíthatunk a parancsunkon, módosíthatjuk a beírást. Mivel ezt a billentyűt minden parancs végén le kell nyomni, a következőkben ezt nem fogjuk minden esetben külön említeni.

▶▶▶ Mivel a **'PRINT'** parancsot nagyon gyakran használják, egész egyszerűen egy **'?'** (kérdőjellel) helyettesíthető. A '?' a [SHIFT] billentyű és a [?] billentyű együttes lenyomásával érhető el, akárcsak az írógépen.

A következő két parancs tehát teljesen egyenértékű:

```
print 2 + 222        vagy          ? 2 + 222
```



```
 224
Ready
■
```

```
 224
Ready
■
```

Az összeadás után lássuk a kivonást. A kivonás választ ad ad arra is, miért nem a képernyő legszélére írta ki gépünk a fenti eredményeket. (Ne feledje: nulla a [ø] billentyűn!)

```
? 19 - 20
-1
Ready
■
```

Az eredmény előtt megjelent mínuszjel elfoglalja az előző eredmények előtti szóköz* helyét, tehát a gép a pozitív számokat nem jelzi plusszal, helyette egy szóközt jelenít meg, a negatív számoknál viszont kiteszi a mínuszjelet.

```
? 22 - -23          - mennyi lesz 22 mínusz -23 ?
 45
Ready
```

A szorzás a csillagjellel történik. Beadása: [SHIFT] [*].

```
print 1234 * 1234
 1522756
Ready
■
```

Az osztás a törtvonallal történik: [/].

```
print 20 / 8
 2.5
Ready
■
```

Megfigyelhetjük, hogy a fenti eredményben a tizedesvessző helyett pont szerepel. Ez az angol-amerikai írásmódot tükrözi. Ugyanez érvényes a tizedes törtek beírására is, itt is ponttal kell helyettesíteni a tizedesvesszőt:

```
print 123.45 / 12.345
 10
```

Az osztással kapcsolatban megjegyzendő, hogy a '/'-jel csak a vele közvetlenül szomszédos számokat érinti, nem helyettesíti a hosszú törtvonalat!

```
print 3 + 4 / 5 + 6       - mennyi lesz 3 plusz 4/5 plusz 6 ?
 9.8
Ready
■
```

Minden egyéb esetben zárójelet kell használni, hogy a CPC helyesen értelmezze a műveletek sorrendjét:

```
print (3 + 4) / (5 + 6)                  3+4
 0.636363637              - mennyi lesz  ---  ?
                                         5+6
```

Az eredmény egy végtelen tizedes tört, amit a CPC kerekítve, 9 számjegyes pontossággal ábrázolt. Ez a pontosság jellemző a CPC BASIC valós* típusú számaira. A valós típusú számokkal kapcsolatban kell megjegyezni, hogy a CPC csak a 9 számjeggyel ábrázolható számokat írja ki a megszokott alakban: ha a szám nagyobb mint 999999999, kisebb mint -999999999, vagy 0.000000001 és -0.000000001 közé esik, az ábrázolás (esetleg kerekítve) ún. normál alakban történik meg. Próbálja ki:

```
print 9999999999             tíz darab kilences!
 1E+10                     - ezt meg hogyan kell érteni?

print -0.0000000001        - kilenc nulla a tizedespont után!
-1E-10                     - 'E' mint exponens
```

Az ilyen ábrázolást <u>lebegőpontosnak</u> hívják, szemben a megszokottal, amelyet <u>fixpontosnak</u> neveznek. A lebegőpontos számábrázolás tartalmazza a mantisszát (a szám jegyeit, max. 9 jegy) és 'E' után az előjeles kitevőt (exponens, -39 és +38 között). Az 'E' után álló rész azt mutatja, hagy hány jellel kerül a tizedespont arrébb: jobbra kell eltolni, ha a kitevő pozitív, negatív kitevő esetén pedig balra. Mivel a CPC nemcsak kiírni tud exponenciális alakban, hanem az így beadott számokat el is fogadja, könnyen kipróbálhatjuk a megszerzett ismereteinket:


```
print 2.345E0       = 2.345 * 10 a nulladikon (vagyis egy)
 2.345
print 2.345E4       = 2.345 * 10 a negyediken (vagyis tízezer)
 23450
print 2.345E-2      = 2.345 * 10 a mínusz másodikon (= 0.01)
 0.02345
print -2.234E-3     = -2.345 * 10 a mínusz harmadikon (=0.001)
-0.002345
```

Tehát az exponenciális alakban a tíz egy hatványával kell megszorozni a mantisszát. A hatványozást szorzással is helyettesíthetjük, de erre nincs szükség, mivel van más lehetőség is: a hatványozás a felfelé mutató nyíl (↑) segítségével végezhető el. A kitevőt nem a szokásos feljebb írt számmal adjuk meg, mivel a képernyőn az nem ábrázolható, hanem a '↑' jelöli azt. Billentyűje a [CLR]-től balra található.

```
print 2 ↑ 16
 65536
```

A hatványozás segítségével lehet gyököt is vonni, bár a négyzetgyökre a CPC-nek külön függvénye is van.

```
print 27 ↑ (1/3)
 3
```

Az előbbi példában ismét zárójelet kellett használnunk a helyes eredmény eléréséhez, mert a hatványozás az osztásnál magasabb rangú művelet, tehát zárójel nélkül először az kerül végrehajtásra:

```
print 27 ↑ 1/3            - 27 az egyediken osztva hárommal !
 9
```

Látható, hogy a számok és a műveleti jelek között levő szóköz semmit sem mond a CPC-nek, a zárójelek viszont annál többet. Ilyen összefüggésben beszélhetünk a műveletek prioritásáról. Ez a szabály, amely megszabja, hogy egy képleten belül az egyes műveleteket milyen sorrendben kell elvégezni. A BASIC nyelvben

ez a következőképpen van:

- először a zárójelek között levő kifejezés* értéke számítódik ki, a legbelső zárójelben levő kifejezéssel kezdve,
- a műveletek rangsora: hatványozás, azután szorzás vagy osztás, végül összeadás vagy kivonás,
- azonos rangú kifejezések balról jobbra következnek.

▶▶▶ Ha biztosak akarunk lenni abban, hogy a műveleteket a CPC a kívánt sorrendben számítja ki, használjunk zárójelet. Mindig annyi bezáró zárójelnek kell lennie, mint ahány nyitó zárójel volt, különben a BASIC hibajelzést ad!

1.2.2 Változók és változónevek

A változók a BASIC programban nagyon fontos szerepet játszanak. Segítségükkel rengeteg munkát takaríthatunk meg, hiszen egy ismétlődő adatra annak 'nevével' hivatkozhatunk. A változót tekinthetjük a számítógép meghatározott tárolórekeszének (skatulya vagy fiók), amelybe betehetünk egy számot vagy szöveget. A rekeszt tetszes szerint elnevezhetjük és legközelebb már csak a nevét kell említeni, hogy hozzáférjünk a tartalmához. Azzal, hogy hol van ez a rekesz, nem kell törődnünk, ez a BASIC interpreter dolga. A rekeszt abban a pillanatban foglalja le a program, amikor először említjük egy változó nevét és a hozzárendelendő értéket. Ez az értékadó utasítással történik:

LET változónév = hozzáréndelendő érték, például

```
let cica = 100      - legyen a 'cica' nevű rekeszben száz!
Ready
■
```

Győződjünk meg róla, hogy tényleg száz van a 'cicában'!

```
print cica          - írd ki, mit tartalmaz a 'cica'!
 100                - tényleg száz van benne
Ready
print kutya         - vajon mi van a 'kutya' nevű rekeszben?
 0                  - semmi, ez várható volt
Ready
■
```


A 'kutya' nevű változót még egyszer sem említettük, a CPC mégsem adott hibajelzést, amikor lekérdeztük az értékét ... A magyarázat egyszerű: amikor a BASIC találkozik egy új változónévvel, automatikusan keres neki egy rekeszt és - ha számról van szó - nullát tesz bele. A 'LET' szó egyébként el is hagyható, az utasítás akkor is értékadás lesz, tehát

cica = 100 és let cica=100 egyenértékű.

Egy változóhoz természetesen egy számítás eredménye is hozzárendelhető:

```
kiscica = cica - 99
Ready
■
```

A fenti parancsra ez történt: a BASIC keresett egy szabad rekeszt, elnevezte 'kiscicá'-nak, megkereste a 'cica' nevű rekeszt, kiolvasta a tartalmát, kivont belőle 99-et és betette a már előkészített 'kiscica' rekeszbe, ami most 1-et tartalmaz. Győződjön meg róla!

```
print kiscica
 1
Ready
■
```

A változó egyszerre is szerepelhet az egyenlőségjel mindkét oldalán, így matematikailag ugyan helytelen, de BASIC-ben egy nagyon is hasznos értékadó szerkezetet kapunk:

```
cica = cica + kiscica       - új értéket adunk a 'cicá'-nak
Ready                       - elfogadta, tehát megértette!
print cica                  - akkor most mennyi a 'cica' ?
 101                        - új 'cica' = régi 'cica' + 'kiscica'
Ready
■
```

Tehát a 'cica = cica + kiscica' utasítás nem egyenlőség kifejezése (akkor nem lenne sok értelme), hanem értékadás. Arra kérjük a CPC-t, hogy vegye elő a 'cica' rekesz és a 'kiscica' rekesz tartalmát, adja össze őket, majd az eredményt rakja be a 'cica' rekeszbe. A 'cica' régi tartalma elvész!

▶▶▶ Amint láttuk, a változóknak hosszú és értelmes nevet adhatunk. A CPC BASIC megengedi, hogy egy változónév max. 40 karakter*-ből álljon! Használjon értelmes változó neveket, hogy később könnyebb legyen eligazodnia programjaiban.

A változók névadásakor a következő szabályokat kell betartani (különben 'Syntax error'!) :

- az első karakter az angol ábécé betűje legyen,
  a következő karakterek lehetnek betűk vagy számok,
  de a kis- és nagybetűk között a BASIC nem tesz különbséget,
- más BASIC-ektől eltérően a változónevek megkülönböztetésekor a CPC minden (max. 40) karaktert figyelembe vesz, és elfogadja a '.' (pontot) is változónévben.

1.2.3 A változók típusai

BASIC-ben a változóknak alapvetően két csoportja van:

1. Numerikus változók vagy számváltozók.
   Ha a CPC egy numerikus változót talál, annak értékét kikeresi a memóriában és ezzel az értékkel számításokat tud végezni. Eddig minden változónk numerikus volt.
2. Szöveges típusú változók vagy karakterláncok (stringek).
   Ezekkel a változókkal nem lehet számolni, bennük szöveget tárolhatunk, egyenként maximum 255 karakter hosszúságig. Ezekkel a változókkal a numerikus változók után foglalkozunk majd.

1.2.3.1 A numerikus változók

A numerikus változóknak két típusa van:

valós típusú változók (REAL változók)
egész típusú változók (INTEGER változók)

Eddig az összes használt változónk valós típusú volt. Normál esetben minden változó, amelynek neve után semmilyen más jel nincs, valós típusú változó. A valós típust azért hívják így, mert vele minden számot ábrázolhatunk: egészet, tizedes törtet, negatívot, pozitívot, kicsit, nagyot (bizonyos korlátozásokkal, ahogy már láttuk). Ezzel szemben az egész típusú változókkal csak egész számokat ábrázolhatunk, mégpedig a -32768 és +32767 közötti intervallumban. Az egész típusú változókat a nevük után álló '%' (százalékjel) különbözteti meg a többi változótól.

```
egesztipus% = 1234.56      - elfogadja a tizedes törtet egésznek?
Ready                      - elfogadta!
print egesztipus%          - vajon hogyan tárolja?
 1235                      - természetesen egészként (kerekítve)!
Ready
■
```

Valószínűleg felmerül az Olvasóban a kérdés: mire jók az egész típusú változók, hiszen lényegesen kisebb a tartományuk, mint a valósoknak! Az egész típusú változóknak megvan az az óriási előnyük, hogy kevesebb memóriaterületet igényelnek és a velük történő számolás lényegesen gyorsabban történik, mint a valós típusúakkal.

```
ezisvaltozo = 1234.5       - ez egy valós típusú változó
ezisvaltozo% = 5678        - ez pedig egy egész típusú
```

Két változó ugyanolyan név alatt? Igen, mert az CPC a változónak nemcsak a nevét írja fel a rekeszre, hanem a típusát is. Amelyiknek nincs típusjelzése, azt valósnak könyveli el. A valós típust jelölheti a neve után álló '!' (felkiáltójel) is, de erre általában nincs szükség.

```
print ezisvaltozo!         - ugyanaz mint az 'ezisvaltozo'
 1234.5                    - tehát az értéke is ugyanaz
print ezisvaltozo%         - ez már egész típusú!
 5678                      - vagyis saját rekesze van
```

1.2.3.2 A szöveges típusú változók

Hívják őket karakterfüzérnek, karakterláncnak és stringnek is. Bárhogy is nevezzük őket, megkülönböztető tulajdonságuk

az, hogy nem számok, hanem szövegek tárolására használhatók. A karakterlánc típusú változó neve után '$' (dollárjel) áll. Értékadásnál a szöveget idézőjelbe kell tenni, különben a BASIC azt hiszi, hogy egy változónévről van szó! A szöveges típusú változók értéke az a szöveg, amelyet tárolnak:

```
hogy.hivnak.engem$ = "Futo Istvan"   - szép hosszú változónév,
Ready                                - de elfogadta
■
print hogy.hivnak.engem$             - vajon meg is jegyezte?
Futo Istvan                          - természetesen
Ready
■
```

Van tehát egy hosszúnevű rekeszünk, amiben szöveget kívánunk tárolni, ezt jelöltük a neve után álló '$' jellel. A tárolandó szöveggel kapcsolatos egyetlen megkötés, hogy nem lehet 255 karakternél hosszabb. Természetesen a rekeszen ez a típusjelzés is szerepel:

```
cica = 101                           - csak akkor kell, ha közben
Ready                                  kikapcsolta a gépet!
■
cica$ = "Cirmos"                     - ez egy másik változó
Ready                                - ezt is elrakta
■
print cica, cica$                    - egy 'print'-tel két cica?
 101          Cirmos                 - a szám és a szöveg
Ready
■
```

Amint láttuk, egy 'PRINT' paranccsal több változó értékét kiírathatjuk, ha közéjük vesszőt teszünk. Ilyenkor a következő kiírás az előző után egy bizonyos távolságra folytatódik. Szövegek kiíratása a képernyőre ugyancsak idézőjellel történik:

```
print viszontlatasra!                - ez egy valós típusú változó
 0                                   - aminek még nem adtunk értéket
print "viszontlatasra!"              - ez egy szöveg, amit ki kell
viszontlatasra!                        írni
Ready
■
```


Írassuk ki a '123'-at számként és szövegként! Nem ugyanott jelennek meg, vajon miért? A számnak jár az előjel, vagy a pozitív előjelet helyettesítő szóköz, a szövegnek pedig nem lehet előjele! A szövegek nem számok, még ha számjegyekből is állnak.

```
print 123                            - írd ki ezt a számot!
 123                                 - az előjel helye ott van
print "123"                          - írd ki ezt a szöveget!
123                                  - szövegnek nem jár előjel
```

1.2.4 Programot írunk!

Eddig minden parancsunkat annak beadása után az okos CPC azonnal végre is hajtotta. Ezt az üzemmódot parancsmódnak is hívják. A parancsmódot általában akkor alkalmazzuk, ha valamilyen számítást gyorsan el akarunk végezni, vagy egy változó aktuális értékét akarjuk lekérdezni. Sokkal fontosabb üzemmód azonbam a program beírása és annak futtatása. A programban a parancsokat programsorokban helyezzük el, ekkor már utasításoknak hívják őket. Minden programsor egy sorszámmal kezdődik:

```
20print "program" [ENTER]      - ne írjon szóközt a szám után!
■
```

A kurzor megjelent a szokásos helyén, de hiányzik a 'Ready' felirat, tehát a CPC nincs kész a feladattal. Viszont hibát sem jelzett, vagyis megértette. Akkor tulajdonképpen mit csinált? Elkezdte olvasni a beadásunkat és számot talált az elején, automatikusan úgy döntött, hogy programot kívánunk írni, annak első sora pedig 20-as sorszámú lesz. Egy sorszám addig tart, amíg számjegyet tartalmaz. Az első nemszámjegy csak betű lehet: vagy egy parancsszó, vagy pedig egy változónév kezdődik vele. Minden másra a CPC hibát jelezne. Ezek szerint beírtuk programunk első sorát, a 20-as sorszámút. Ellenőrizzük, megvan-e még!

```
list [ENTER]                         - listázd ki a programot!
20 PRINT "program"                   - itt a 20-as sor
Ready                                - készen van a listázással
■
```

Az első sorunkat szépen rendberakta a CPC: elválasztotta

a sorszámot a 'print' parancsszótól, a kisbetűs 'print'-ből pedig nagybetűs 'PRINT'-et csinált. Az idézőjelben álló "program"-ot természetesen békén hagyta.

▶▶▶ A BASIC parancsszavakat tetszésünk szerint kisbetűvel vagy nagybetűvel adhatjuk be, a CPC mindkettőt egyformán megérti. Próbálkozó kedvűek kisérletezhetnek a 'pRiNt'-tel is. Listázáskor azonban mindig nagybetűvel látjuk viszont az értelmes parancsszavakat. Tehát tanácsos őket kisbetűvel beírni: amelyek listázáskor is kisbetűvel jelennek meg, azok a CPC számára értelmetlenek voltak. Más BASIC verzióktól eltérően a CPC csak akkor ismeri fel a BASIC parancsszavakat, ha azokat egy szóköz követi!

A BASIC program lényegében egy vagy több programsorból áll, amelyek sorszámmal kezdődnek és egy vagy több utasítást tartalmaznak. A sorok beadásának sorrendje tetszőleges, a CPC a sorszámuk alapján rendezi őket. Bővítsük első programunkat:

```
10 print "Ez az elso" [ENTER]     - ez a 20-as sor elé kerüljön
30 end               [ENTER]     - ennek a 20-as után kell jönni
1 rem program proba  [ENTER]     - ez legyen a legelején

list [ENTER]                     - vajon milyen a sorrend?
1 REM program proba              - természetesen a sorszámnak
10 PRINT "Ez az elso"              megfelelő!
20 PRINT "program"
30 END
Ready
■
```

Az 1-es sorba új BASIC utasítást írtunk (nagy betűvel listázta!): a **'REM'** után tetszőleges megjegyzéseket írhatunk a programba. A BASIC nem törődik a 'REM' utáni szöveggel, a következő sorra ugrik. Programjainkat lássuk el bőven megjegyzésekkel, mert egy idő után már arra sem emlékszünk majd, hogy mit csinál, még kevésbé arra, hogy azt hogyan csinálja. Mi itt a program célját írtuk a 'REM' után: ez a program arra valo, hogy kipróbáljuk vele, hogyan írhatunk majd bonyolultabb programokat.
A 30-as sorban is egy új BASIC utasítással találkozunk: **'END'**,

vége a programnak. Ha az interpreter ezzel az utasítással találkozik, befejezi a program futtatását és parancsmódba tér vissza. Természetesen az interpreter a program végét 'END' nélkül is felismeri, így általában (de nem mindig!) el is hagyható.

Ha egy olyan új sort adunk be, amelynek a sorszáma megegyezik egy már meglevő soréval, az új sor törli a már meglevőt. Ez a legegyszerűbb módja a javításnak is: egyszerűen írjuk újra a hibás vagy nem tetsző sort. Ha csak sorszámot adunk be, utasítás nélkül, az adott sorszámú sor törlődik a programból.

```
50 [ENTER]                       - töröljük az 50-es sort!
Line does not exist              - a sor nem létezik (mondja ő)
Ready
■
20 print "programom!"            - akkor írjuk át a 20-as sort!
list  [ENTER]                    - vajon átírta?
1 REM program proba
10 PRINT "Ez az elso"            - ezt nem bántottuk
20 PRINT "programom!"            - igen, ez az utolsó változat
30 END
```

Ideje lefuttatni a programot: írja be: **run** és nyomja le az [ENTER] billentyűt!

```
run [ENTER]                      - fuss! vagyis indulj el!
Ez az elso                       - ezt írta ki a 10-es sor
programom!                       - ezt írta ki a 20-as sor
Ready                            - készen van a futással
■
```

A **'LIST'** -tel kilistáztuk, a **'RUN'** -nal lefuttattuk első programunkat. Nem volt túl látványos, hiszen majdnem ugyanazt láttuk a képernyőn mindkét alkalommal. A CPC sem dolgozott valami sokat, rögtön készen lett a kiírással és még csak nem is számolt. Felejtsük el ezt a programot és írjunk egy másikat! Hogy a CPC is elfelejtse, írja be: **new** és nyomja le az [ENTER] billentyűt!

```
new [ENTER]                      - új! vagyis felejts el mindent!
```

```
Ready                          - készen van: elfelejtette
list  [ENTER]                  - hátha mégis maradt valami a régiből
Ready                          - nem maradt semmi
```

A **'NEW'** -val óvatosan kell bánni, hatására a program végleg elvész! De most tényleg nem volt már rá szükség, csak zavartak volna a bent maradó sorok. Adjuk be a következő rövid, ám a végtelenségig futó programot:

```
1 rem VEGTELEN PROGRAM
10 szam = 0                    - 'szam' változó értéke legyen nulla
20 print szam                  - írassuk ki az értékét!
30 szam = szam + 1             - növeld eggyel a 'szam' változót!
40 goto 20                     - menj a 20-as sorra, folytasd ott!
50 end                         - itt a program vége
```

A 30-as sor furcsaságával már megismerkedtünk: a változó régi értékéhez hozzáad 1-et és az eredményt elrakja a változó rekeszébe, tehát a változó új értéke eggyel növekszik. Most a **'GOTO'** utasítás hatását nézzük: a program nem a következő sorszámú sorral folytatódik (itt nem ér véget), hanem a 'GOTO' után álló számmal jelzett sor következik. Programunk tehát végtelen ciklusba kerül, de ezt inkább próbáljuk ki: **run** [ENTER]! Ha már elege van a képernyő bal szélén kígyózó számsorból, nyomja le kétszer egymásután az [ESC] billentyűt!

Amint láttuk, a 'PRINT' mindig új sorba írja ki a következő számot. Megismertük már a módját annak is, hogyan ne kezdjen új sort: a kiíratást ',' (vesszővel) kell lezárni. Nincs más hátra, újra kell írni a 20-as sort:

```
20 print szam,                 - írd ki és menj egy oszloppal jobbra
```

Most három oszlopba rendezve jelennek meg a számok, tehát a vessző három oszlopra tagolja a képernyőt. Ha még sűrűbben kívánjuk kiíratni a számokat, egy másik elválasztójelet kell használnunk: a ';' (pontosvessző) a szövegeket szorosan egymás mellé engedi, a számoknál viszont kihagy egy helyet a szám elején (az előjelnek) egy másikat pedig a szám végén. Még egyszer írjuk át a 20-as sort:


```
1   2   3   4   5   6   7   8   9   10  11  12
13  14  15  16  17  18  19  20  21  22
23  24  25  26  27  28  29  30  31  32
33  34  35  36  37  38  39  40  41  42
43  44  45  46  47  48  49  50  51  52
53  54  55  56  57  58  59  60  61  62
63  64  65  66  67  68  69  70  71  72
73  74  75  76  77  78  79  80  81  82
83  84  85  86  87  88  89  90  91  92
93  94  95  96  97  98  99  100 101
102 103 104 105 106 107 108 109
110 111 112 113 114 115 116 117
118 119 120 121 122 123 124 125
126 127 128 129 130 131 132 133
134 135 136 137 138 139 140 141
142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157
158 159 160 161 162 163 164 165
166 167 168 169 170 171 172 173
174 175 176 177 178 179 180 181
182 183 184 185 186 187 188 189
190 191 192 193 194 195 196 197
198 199 200 201 202 203 204 205
206 207 208 209 210 211 212 213
214 215 216 217 218 219 220 221
```

```
20 print szam;                 - írd ki, de ne menj messzire!
```

Az eredményt a fenti ábrán is láthatjuk: minden szám között két szóköz. Rövidke programunk tehát betölti az egész képernyőt és jól megdolgoztatja a CPC-t. Lehetne ugyanez még rövidebb? Igen, ugyanis a CPC BASIC-je megengedi, hogy egy sorba több utasítást is írjunk. Az egyes utasításokat ':' (kettősponttal) választjuk el egymástól. Egy sorba több utasítás elhelyezésének előnye is van, de ha túlzásba visszük, akkor inkább a hátrányait kell emlegetni. Előnye, hogy tárat takarítunk meg, mivel a CPC-nek kevesebb sorszámot kell megjegyeznie. Ha a szorosan összetartozó utasításokat egy sorba írjuk, programunk érthetőbb lesz.

▶▶▶ A CPC-nek elég nagy operatív tára" van, hogy ne kelljen vele mindenáron takarékoskodni. Bár egy programsor 255 karakter hosszú is lehet (több mint 6 képernyő sor!), ezt ne használjuk ki feltétlenül, inkább írjunk rövidebb, át-

tekinthető programsorokat.

Most pedig lássuk, milyen lenne a programunk egy sorba összezsúfolva:

```
new       [ENTER]      - felejtse el az előző programot!
Ready
10 print szam;: szam = szam + 1 : goto 10
```

Minden új numerikus változó nullát kap kezdőértéknek, ha mi nem adunk neki értéket. A 'szam' változó is ennyivel kezdi, ahogy az első utasítás meg is jeleníti. A második utasítás növeli a változót, a harmadik utasítás: 'goto 10', vagyis menj újra erre a sorra! 'Szam'-unk megint növekszik, újra megjelenik szorosan az 1-es mellett és megint növekszik és megint megjelenik ... Programunk ugyanúgy fut, ahogy az előbb, amikor az utasításokat még külön sorba írtuk. Láttuk, hogy a 'GOTO' utasítással nemcsak egyik sorból a másikba, de saját sorára is ugrathatunk. Nem lehet azonban egy olyan utasításra ugrani, amelyik egy sor belsejében van. Azt az utasítást, amelyre ugratni akarunk (amelynek át akarjuk adni a vezérlést), új sorba kell elhelyezni. Ez pedig még egy érv amellett, hogy ne írjunk mindent egy sorba.

1.3 A CPC BILLENTYOZETE

Mielőtt hosszabb programok írásához fognánk, részletesen meg kell ismerkednünk a számítógép billentyűzetével. Csak így tudjuk annak minden szolgáltatását kihasználni, amelyek lényegesen megkönnyíthetik a munkát.

A CPC 464-en és a CPC 664-en a billentyűzet három részből áll:

- alapbillentyűzet, tartalmazza az alfanumerikus billentyűket (betűk, számok, egyéb jelek), valamint az egyes speciális célú kontroll billentyűket, pl.[ESC].
- kurzorblokk, tartalmazza a négy nyílbillentyűt és a [COPY] feliratú billentyűt
- tízes számblokk, tartalmazza az összes számjegybillentyűt, egy [.] billentyűt a tizedespont beadásához és egy [ENTER] billentyűt a beadás lezárásához.

Bár a CPC 6128-on a három részt összevonták és egyes billentyűk is más helyet kaptak, a billentyűzetet mégis az előbbi felosztás alapján ismertetjük.

1.3.1 Az alapbillentyűzet

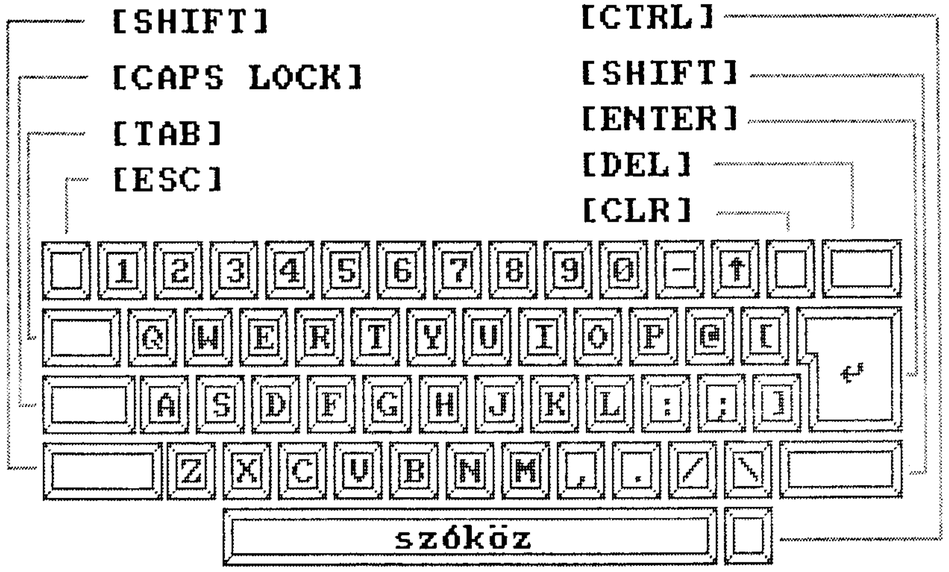
Az alapbillentyűzet - a speciális billentyűket kivéve - nagyrészt megegyezik az írógép billentyűzetével. Az ékezetes betűk hiányoznak (később meglátjuk, hogyan pótolhatjuk őket), egyes jelek is máshol találhatók, valamint a gépelni tudókat zavarhatja még, hogy a [Z] és az [Y] ki van cserélve. Alapállapotban a betűbillentyűk lenyomására a megfelelő kisbetű jelenik meg a képernyőn, a kétfeliratos billentyűk az alsó ábrának megfelelő számjegyet vagy jelet adják. A gépírni jól tudókat mégegyszer figyelmeztetjük, hogy a nullának és az o-betűnek külön billentyűje van, csakúgy mint az egyesnek és az l-betűnek. Amennyiben a billentyűzet elrendezése valamilyen oknál fogva nem nyerné el a tetszését, vagy hiányolja az ékezetes betűket, még idejében jegyezzük meg, hogy a CPC billentyűzete egyszerű módon (BASIC parancsokkal) tetszés szerint átprogramozható. Tehát van lehetőség az [Y] és a [Z] kicserélésére, az ékezetes betűk elhelyezésére stb.

1.3.1.1 A speciális célú billentyűk és funkcióik

[ESC] - A program megállítására szolgál.
A billentyű egyszeri lenyomása a programot megszakítja, de nem állítja le végleges: bármely más billentyű lenyomására a program folytatódik. Hasonlóképpen szüneteltethetjük a programok listázását is.
A billentyű másodszori lenyomására a program vagy a listázás végleg leáll. A billentyű csak BASIC programok futtatásakor hatásos, ilyenkor 'Break in sorszám' üzenettel áll le a program

[DEL] - A tévesen bevitt karakter törlésére szolgál.

A CPC alapbillentyüzete



A kurzortól balra álló karaktert törli és a kurzort annak a helyére állitja. Ha már az egész beadást töröltük és nincs hova visszalépni, sípoló hang figyelmeztet.

[CLR] – A beadás javitására, módositására szolgál. Hatása a [DEL] billentyűhöz hasonló, de nem hátrafelé töröl, hanem a kurzor alatti karaktert törli. Akkor használhatjuk, ha előbb a [←] billentyűvel visszalépünk a kitörlendő karakterre. Részletes leirása a programok szerkesztéséről szóló részben található.

[TAB] – BASIC-ben nincs funkciója. Lenyomására egy jobbra néző nyil kerül kiirásra, amelyet a BASIC szóköznek tekint. Egyes felhasználói programok* nevének megfelelően tabulálásra használják.

[CAPS LOCK] – A betűbillentyűket állandó nagybetűs üzemmódra

kapcsolja át. Ismételt lenyomása kisbetűkre kapcsol vissza. Csak a betűbillentyűkre van hatással, a többi billentyűt nem váltja át.

[ENTER] – Lenyomása parancs vagy programsor beadásának a végét jelzi: a BASIC interpreter csak ezután veszi át az üzenetet. Futó program közben szintén a kezelői beadás végét jelzi. A gép bekapcsolása után a CPC 464 és a CPC 664 mindkét [ENTER] billentyűjének ez a funkciója. A CPC 6128-on a nagy [ENTER] billentyű [RETURN] feliratot kapott.

♦CPC 6128: [RETURN]

[SHIFT] – A két [SHIFT] billentyűnek azonos a funkciója. A [SHIFT]-et lenyomva tartva a vele egyidőben lenyomott
1. betűbillentyű a megfelelő nagybetűt adja,
2. jelbillentyű a billentyűn ábrázolt jelek közül a felsőt jeleniti meg. Számbillentyűk 'siftelve' szintén jeleket adnak. A szögletes zárójelek '[' és ']'helyett '<' és '>' jelenik meg, ha a [SHIFT]-tel együtt nyomjuk le őket. [SHIFT] és [\] felső vesszőt jelenit meg: '^'.

[CTRL] – Magában, mint a [SHIFT]-nek, nincs funkciója. Lenyomva tartva egy betűbillentyűvel együttesen a betű ASCII* kódjánál 64-gyel kisebb kontroll kódot ad ki. A kontroll kódok különböző grafikus karakterek formájában jelennek meg és karakteres változókba beágyazhatók. Használatukról a Második szinten lesz szó. [CTRL] és [2] = felső hullámvonal '~'.

♦CPC 6128: [CONTROL]

[CTRL] és [CAPS LOCK] – A két billentyű együttes lenyomása a [SHIFT] – hez hasonló eredményt ad, hatása azonban tartós: a két billentyű újabb együttes lenyomásáig tart.

[CTRL] [TAB] – A két billentyű együttes lenyomás a 'beszúró' üzemmódból vált át a 'felülíró' üzemmódba és vissza. Alapállásban a 'beszúró' üzemmód van érvényben, ami azt jelenti, hogy ha egy elkez-

dett sorban a [←] billentyűvel visszalépünk és beírunk valamit, a már meglevő szöveg jobbra tolódik, hogy helyet csináljon. Használatára programsorok módosításakor lehet szükség.

1.3.2 A kurzorblokk

A CPC 464-en és a CPC 664-en a jól elkülönülő kurzorblokkból a nyílbillentyűk [←], [→], [↑], [↓] a CPC 6128-on a tízes blokk alá kerültek, a [COPY] feliratú billentyű pedig a szóköz billentyűtől balra található. Funkciójuk azonban változatlan maradt. Használatukat a programok szerkesztéséről szóló fejezet ismerteti.

1.3.3 A tízes számblokk

A tízes számblokk tartalmazza a numerikus adatok beadásához szükséges összes billentyűt. A CPC 6128-on az [ENTER] billentyű kiszorult a blokkból és a szóközbillentyű jobb oldalára került. A CPC 664 és a 6128 számbillentyűin a számokat egy 'f' betű előzi meg. Az 'f' betű a billentyűk funkcióbillentyűként való használatára utal, ami azonban érvényes a CPC 464-re is, bár azon ezt külön nem jelölik. Alapállásban, bekapcsolás után, csak egy előre beprogramozott funkcióbillentyű él. A kazettás programok betöltését és indítását megkönnyítő [CTRL] + kis [ENTER] használatát a bemutató program betöltésénél már megismertük. A funkcióbillentyűkhöz tetszőleges hosszú szöveget rendelhetünk, amely a billentyű lenyomásával beadásra kerül, mintha kézzel billentyűztük volna be a szöveget. A funkcióbillentyűk programozásáról a 2.7.3 pontban lesz szó.

1.4 BASIC PROGRAMOK SZERKESZTÉSE

Eddigi munkánk során, ha valami javítani valónk volt, akkor a [DEL] billentyűvel töröltük a hibás részt vagy az egész sort újraírtuk. Ez a módszer hosszabb programok írásakor eléggé fáradságosnak bizonyulhat, ezért a CPC többféle segítséget is nyújt a program szövegének szerkesztéséhez (beadásához, módosításához). Ezek megismeréséhez írjunk be egy rövid programot, természetesen hibásan. A program első sora egy új utasítást tartalmaz: a **'CLS'** törli a CPC képernyőjét.

```
1 rem MEGTANULUNK JAVITANI
10 clls                              - CLS =töröld a képernyőt!
20 kiir$ = "javitok"                 - ez tulajdonképpen helyes
30 pront kiir$; kiir$                - helyesen 'print'
40 got 10                            - helyesen 'goto'
```

1.4.1 Javítás 'Syntax error' után és az 'EDIT' parancs

Hibás programunkat megpróbálhatjuk futtatni, de az már az első sornál leáll az ismerős 'Syntax error' üzenettel. Ezúttal azonban megjelenik a hibás sor sorszáma, azt követi a teljes hibás sor, a kurzor pedig a sorszám első számjegyén jelenik meg:

```
run
Syntax error in 10                   - nem értem a 10-es sort!
■0 clls                              - kisbetűs maradt a parancs
```

Igen, két 'l'-betűvel írtuk a 'cls'-t. A [→] (jobbra nyíl) billentyűvel menjünk az egyik 'l' fölé majd nyomjuk le a [CLR] billentyűt! A kurzor alatti betű eltűnik és az utána álló 's' balra húzódik. Más javítanivaló nincs, nyomjuk le az [ENTER]-t. A CPC újra átveszi a sort, most már javított formában. Fusson újra a program! Az első sor hibátlan: törlődik a képernyő.

```
Syntax error in 30
■0 pront kiir$;kiir$                 - kisbetűs maradt a parancs
```

Az előző módszerrel kitöröljük a hibás 'o'-betűt, a sor többi része balra húzódik: 'prnt'. Nyugodtan írja be az 'i' betűt, az majd jobbra nyomja a sor végét. Gépünk beszúró üzemmódban van, amit most beírunk, nem törli az utánna álló betűt, hanem helyet csinál magának. Esetleg írjon be több 'i'-t is, majd törölje őket visszafelé a [DEL] billentyűvel, amíg csak egy marad, azután [ENTER]. Mivel rövid soraink vannak, csak a [←] és [→] billentyűket használjuk, hogy balról jobbra vagy visszafelé mozogjunk. Ha olyan hosszú a programsorunk, hogy több képernyősort foglal el, akkor a [↑] és [↓] billentyűket

is használhatjuk a hibás hely kikereséséhez.
Futtassuk a kijavított programot!

```
javitokjavitok
Syntax error in 40
■0 got 10
```

A szokásos módszerrel beszúrjuk a 'goto' második 'o'-ját, de a képernyőn látható eredmény nem az, amit akartunk. A két szöveg egymásba ér. Javítsuk át azt a sort is! Egy, különben helyes sort az **'EDIT'** paranccsal hívhatunk elő módosításra. A BASIC interpreter hiba esetén magától áll át editáló, javító üzemmódba, ha formai hibát nem talál, nekünk kell a kívánt sort módosításra kikérni:

```
edit 30  [ENTER]                    - javításra kérem a 30-ast!
■0 PRINT kiir$;kiir$                - ';' helyett ',' kell
```

Cseréljük ki a pontosvesszőt vesszőre és adjuk be a sort. Most már fut a program, ám elég furcsán: a két "javitok" alig jelenik meg, már el is tűnik! Állítsuk le, [ESC] [ESC], és listázzuk a programot. A lista a képernyő felső részén jelenik meg. Igen, a 10-es sorra küldtük vissza a programot, tehát minden kiírás után törli a képernyőt. At kell írni a sorszámot. Az 'EDIT' paranccsal megtehetnénk, de most inkább egy másik, kényelmesebb módszert alkalmazzunk!

1.4.2 Vegyük birtokba a képernyőt!

```
Break in 40                 - Önnél esetleg másik sorban állt meg
Ready
list
10 CLS                      - itt van minden a képernyőn,
20 kiir$="javitok"            csak le kellene másolni, ami jó
30 PRINT kiir$, kiir$         és átírni, ami nem tetszik!
40 GOTO 10                  - 'goto 30' és már nem töröl mindig
Ready
■
```

Nyomja le a [SHIFT] billentyűt és tartsa lenyomva! Indul-jon el felfelé a [↑] (felnyíl) billenytűvel: a kurzor kettévál-ik, az alapkurzor a helyén marad, az újat Ön mozgatja. A nyíl billentyűkkel nyugodtan elkalandozhat az egész képernyőn: amíg a [SHIFT]-et is lenyomva tartja, addig az alapkurzor nem mozdul. Ha most a [SHIFT]-nélkül mozog a nyilakkal, az alapkurzort irányítja és az új marad a helyén. Az alapkurzor az "igazi": ahol hagyja, ott fog írni, az új kurzor a copykurzor. Az alap-kurzort hagyja a képernyő bal szélén és a copykurzort vigye a lemásolandó sor elejére, a 4-es számjegyre. Nyomja le a **[COPY]** billentyűt: a copykurzor jobbra lép, az alatta álló karakter pedig megjelenik az alapkurzor alatt is, amely szintén jobbra indul el. Ezzel a módszerrel a képernyőről tetszőleges szöveg átmásolható abba a sorba, amelyet éppen írunk vagy módosítunk. Egy sor újraírása helyett másoljuk át a hibátlan részt, folytas-suk a módosítással, majd, ha kell [SHIFT] nyíllal megkeressük a már újra hibátlan részt és azt is átmásoljuk a [COPY] billen-tyűvel. Akár a képernyő több részéből is összeállíthatunk egy új sort, parancsot.

Amennyiben a képernyőn van még egy régi parancs vagy sor és módosítás nélkül újra be kívánja adni, az alapkurzorral is megkeresheti az elejét, majd a [COPY]-val végigmegy rajta. Csak az [ENTER]-t kell megnyomni és a CPC máris átveszi az egész szöveget.

Összefoglalva:

```
[←],[→],[↑],[↓]             - az alapkurzort mozgatja
[SHIFT] és nyílbillentyűk   - a copykurzort mozgatja
[COPY]                      - mindkét kurzort jobbra lépteti
                              és a copykurzor alatti karak-
                              tert az alapkurzor alá másolja
```

Mindeddig már beadott hibás sorok javításáról volt szó. Mit tehetünk, ha egy sor végén vesszük észre, hogy pl. a sorszám hibás és az így felülírna egy már létező sort? Nem szükséges visszafelé kitörölni az egész sort a [DEL] billentyűvel, két mód is van a javításra. Visszamehetünk a hibás sorszámig a [←] billentyűvel is, csak a sorszámot töröljük ki a [DEL]-lel vagy [CLR]-lel és beadjuk az immár kijavított sort. A másik módszer, hogy nem adjuk be a sort, hanem az [ESC] billentyűt nyomjuk le.

A sort egy 'Break' (megszakadt) felirat zárja majd le, egyébként rendelkezésünkre áll, hogy a másolós módszerrel felhasználjuk.

▶▶▶ A másolásos módszernél a CPC a képernyőről olvassa le a kurzor alatti szöveget. Ha a szövegre grafikát rajzolunk, legyen az egy alig látható pont, a gép nem képes azonosítani a karaktereket. Ilyenkor a [COPY] nem másol, hanem hangjelzéssel figyelmeztet!

1.4.3 A CPC BASIC programszerkesztő parancsai

Új parancsok, utasítások:

LIST
DELETE
RENUM
AUTO
STOP
CONT

A CPC több olyan parancsot ismer, amelyeket nem programokban használunk fel, hanem programírás közben. Ezek közül már megismertük a 'LIST' és az 'EDIT' parancsokat. A programírást megkönnyítő parancsokat is elhelyezhetjük programsorban, utasításként, de akkor végrehajtásuk után a BASIC parancsmódba tér vissza, a program futása tehát megszakad.

```
10 print "10-es sor"          - írjon valamit
20 print "20-as sor"          - megint írjon valamit
30 list                      - listázza ki a programot
40 print "40-es sor"          - írjon még valamit
```

A fenti program az első két sor végrehajtása után kilistázza önmagát és 'Ready'-vel jelentkezik, a 40-es sorig már nem jut el. A 'LIST' parancs utasításként való használata ezért általában nem célszerű. Programok tesztelésénél a program utolsó sorában (vagy a program logikai végén - az 'END' helyett - elhelyezett 'LIST' azonban meggyorsíthatja a munkánkat.

A 'LIST' parancsot eddig paraméter nélkül használtuk, így kilistázta az egész programot. Hosszabb program listázását az [ESC] billentyű egyszeri lenyomásával állíthatjuk le, bármely

más billentyű lenyomása a listázást folytatja, másodszori [ESC] 'Ready'-hez tér vissza. A parancs után sorszámok is megadhatók:

```
LIST 20          - csak a 20-as sort listázza ki
LIST 20 - 40     - a 20, 40 és a közöttük levő sorok
                   jelennek meg a képernyőn
LIST - 50        - a program elejétől az 50-es sorig
                   (azt is) listáz
LIST 50 -        - az 50-es sortól (azt is) listáz
                   a program végéig
```

Programsorok egyenkénti kitörlését megoldhatjuk a sorszám beadásával: a BASIC átveszi a beadott sorszámú üres sort, törli az azonos sorszámú már meglevő sort, de az üres sort nem iktatja be a programba, tehát egy programsort kitöröltünk. Több sor, egy egész programrészlet kitörlése ilyen módon hosszadalmas lenne. Programsorok, részletek kitörlésére szolgál a **'DELETE'** parancs:

```
DELETE 20        - törli a 20-as sort
DELETE 20 - 40   - törli 20, 40 és a közöttük levő
                   összes sort
DELETE - 50      - a program elejétől az összes
                   sort törli az 50-essel bezárólag
DELETE 50 -      - az 50-es sortól (azt is) a prog-
                   ram végéig törli az összes sort
```

Programsorok folyamatos beadását segíti elő a sorszámokat előállító **'AUTO'** parancs. Használata lehetővé teszi, hogy nem kell ügyelnünk a sorok helyes sorszámozására, teljes figyelmünket a "lényegre" fordíthatjuk. AZ 'AUTO' parancs önmagában, paraméter nélkül, 10-es sorral kezdődően 10-es lépésközzel állít elő sorszámokat. A kiírt sorszám után megjelenik a kurzor, ugyanúgy írhatunk, mintha a sorszámot magunk adtunk volna be. Ha nem kívánuk több sort beadni, az automatikusan kiírt sorszám után az [ESC] billentyű lenyomásával térhetünk vissza parancsmódba.

Az 'AUTO' parancsban megadhatjuk az első kívánt sorszámot valamint a lépésközt is:

AUTO első sor, növekmény

Például az 'AUTO 300,2' parancs a következő sorszámokat állítja elő: 300, 302, 304, 306 ... Ha már meglevő programba szeretnénk új sorokat beszúrni az 'AUTO' parancs segítségével, előfordulhat, hogy az automatikusan előállitott sorszám felülírna egy már meglevő sort. A CPC 464 ilyenkor a sorszám után egy '*' (csillagot) ir ki, ezzel jelezve, hogy ilyen sorszámú sor már létezik. [ESC]-vel kiszállhatunk parancsmódba anélkül, hogy felülirnánk a régi sort. A CPC 664 és 6128 ilyenkor másképpen reagál: editálásra kiadja a már meglevő sort, amelyet igy meghagyhatunk vagy átirhatunk anélkül, hogy parancsmódba térnénk vissza.

Programirás közben előfordulhat, hogy annyi új sort szúrunk be két másik sor közé, hogy már nem marad szabad sorszám még egy új sor beszúrásához. Bár ez nem vall jó programozói gyakorlatra, a CPC BASIC ilyenkor is segit: a **'RENUM'** parancs segitségével az egész programot, vagy annak egy részét újraszámoztathatjuk. A **'RENUM'** parancs önmagában, paraméter nélkül, az egész programot átszámozza, az első sor sorszáma 10-es lesz, a sorok közötti lépésköz szintén 10 lesz. A 'RENUM'-ban megadhatjuk az első átszámozott sor sorszámát, a régi program első átszámozandó sorának sorszámát és a lépésközt is:

RENUM első átszámozott sor sorszáma, régi sor, lépésköz

Számozzunk át egy szűkre sikerült programot:

```
1 rem RENUM proba                    - igy nem lehet új sort
2 goto 3                               beszúrni a régi sorok
3 list                                 közé
```

A 'RENUM' parancs kiadása után a program igy alakul:

```
10 REM RENUM proba                   - a sorok közé még
20 GOTO 30                             kilenc új sort
30 LIST                                beszúrhatunk
```

A 'RENUM 100,20,1' parancs kiadása után a program igy fest:


```
10 REM RENUM proba                   - ezt békén hagyta
100 GOTO 101                         - átszámozta: 100 ← 20
101 LIST                             - lépésköz: 1
```

A fenti átszámozásokban is megfigyelhettük, hogy a BASIC ügyel a 'GOTO' utáni sorszám megfelelő átszámozására is. Ez igy van más olyan utasitásokkal kapcsolatban is, amelyeket sorszám követ.

BASIC programok belövésekor, tesztelésekor szükség lehet arra, hogy a programot tetszőleges helyen megállithassuk, majd (például a kivánt változó értékének lekérdezése után) újra el tudjuk indítani. A program megállitása az [ESC] billentyűvel is lehetséges, de egy gyorsan futó program valószinűleg meghaladja kézügyességünket: az [ESC] billentyűt túl későn nyomnánk le. A programban utasitásként elhelyezett **'STOP'** megállitja a futást és a 'Break in sorszám' üzenettel parancsmódba tér vissza. A program futtatását a **'CONT'** paranccsal folytathatjuk:

```
10 rem STOP - CONT proba
20 a = 10                            - értéket adunk
30 stop                              - állj meg!
40 print a*a                         - csak 'CONT' után jön ide
```

A 'RUN' parancs kiadása után 'Break in 30' üzenettel megáll a program: megszakitva a 30-as sorban. Lekérdezhetjük az 'a' változó értékét: 'PRINT a', majd a 'CONT' (folytasd) paranccsal végrehajtathatjuk a program többi részét is.

```
run                                  - indítsuk el
Break in 30                          - megállt a 30-as sorban
Ready
print a                              - mennyi most 'a'?
 10
Ready
cont                                 - rendben: folytasd!
 100                                 - ezt a 40-es sor irta ki
Ready
```

A 'CONT' paranccsal folytathatjuk az [ESC] billentyűvel

megállitott programot is. Ha a megállitott programot módositjuk vagy új sort szúrunk be, a programot nem lehet folytatni. Esetleges próbálkozásunkra a 'Cannot CONTinue' (nem tudom folytatni) üzenetet kapjuk válaszul.

1.5 STANDARD BASIC

Ez a szakasz bemutatja a BASIC-ben még járatlan Olvasónak a nyelv legfontosabb elemeit. Ezek az elemek a legtöbb személyi számitógép BASIC-jében megtalálhatók. Ezért, ha Ön már túljutott a programozás alapjain, nyugodtan ugorja át ezt a részt.
A bemutatásra kerülő programok, programrészletek nem használják ki a CPC egyedi lehetőségeit: nem színesek, nem rajzolnak, nem zenélnek. Van viszont egy óriási előnyük: gyakorlatilag módositás nélkül futtathatók más, eltérő típusú gépeken is.

Mivel programjaink egyre hosszabbak és bonyolultabbak lesznek, az érdekesebbeket célszerű későbbi használatra a CPC beépitett külső tárába kimenteni. A programok kimentése és betöltése kazettára és lemezre egyformán történik:

SAVE "programnév" - a programot kimenti
LOAD "programnév" - a programot betölti
RUN "programnév" - betölti és azonnal indítja is

A "programnév" idézőjelek között álló 8 karakter hosszúságú szöveg lehet, amely csak betűkből és számjegyekből állhat. Kazettára való kivitelnél a CPC a következő üzenettel reagál a 'SAVE' parancsra:

Press REC and PLAY then any key: ■ - magnót felvételre!

A beépitett magnón nyomjuk le a 'REC' és 'PLAY' gombokat, majd egy tetszőleges billentyű lenyomására elindul a program kimentése. A betöltéskor ('LOAD') csak a 'PLAY' gombot kell lenyomni. Kivitelkor ügyeljünk arra, hogy nehogy felülirjunk egy fontos felvételt. Betöltéskor a szalagot a keresett program elé kell tekercselni: a CPC megkeresi a "programnév" nevű programot. Lemezre való kimentés és betöltés közben a CPC nem ir ki üzeneteket. Ha a kivánt műveletet elvégzi, 'Ready'-vel jelentkezik.

▶▶▶ Programjai tárolására ne használja a géphez kapott, fontos programokat tartalmazó rendszerlemezeket! Minden új lemezt használatbavétel előtt formattálni kell, ezért CPC 664 és 6128 esetében már most olvassa el a 2.9.1 pontot.

1.5.1 Az eddig megismert utasitások áttekintése

A BASIC nyelv parancsokra hallgat. Ha a gépet bekapcsoljuk, parancsüzemmódba kerül: a beirt parancsot azonnal végre is hajtja. A BASIC programot sorszámmal kezdődő sorok alkotják, amelyek egy vagy több utasitást tartalmaznak. Az utasitásokat ilyenkor ':' (kettőspont) választja el egymástól. A legtöbb parancsszó felhasználható úgy közvetlen parancsként, mint programsorban álló utasitásként. A már megismert utasitások:

PRINT [adat] [elválasztójel] [adat] [elválasztójel] ...
A képernyőn megjeleniti (kiirja) az utána álló adato(ka)t.
Az elválasztójel lehet
',' (vessző): az adatok a képernyőn három oszlopba rendezve jelennek meg, vagy
';' (pontosvessző): az adatok a képernyőn szorosan egymás mellett jelennek meg.

END A program logikai végét jelzi.

REM A program futása során figyelmen kivül hagyandó megjegyzés kezdetét jelzi. ': REM' helyettesithető aposztróffal is: '.

GOTO sorszám
A program a 'sorszám' sorszámú sor első utasitásával folytatódik függetlenül attól, hogy mi áll a 'GOTO'-t követő sorban, vagy a 'GOTO'-t tartalmazó sorban a 'sorszám' után.

CLS Törli a képernyőt. Csak a képernyő tartalma vész el, a program, minden adattal és változóval megmarad és fut tovább.

NEW Parancsként beadva törli a gépben levő BASIC programot. Utasitásként is használható, igy csak akkor lesz hatása, ha a program végrehajtja a 'NEW'-t tartalmazó sort: a program megsemmisiti önmagát.

▶▶▶ Példaprogramjaink beírása előtt ajánlatos mindig törölni az előző programot, különben a megmaradó sorok összekuszálhatják az új programot!

1.5.2 Képernyőkezelés

Új kulcsszavak: **TAB**
LOCATE

A képernyőkezelésre vonatkozó parancsok gyakran eltérnek a különböző BASIC változatokban, de szinte mindegyiknek van utasítása a képernyő törlésére ('CLS') és következő 'PRINT' helyének kijelölésére. A CPC BASIC-jében a 'PRINT TAB(x)' és a 'LOCATE x,y' utasításokkal jelölhetjük ki a következő kiírás helyét.

PRINT TAB (oszlop) [adatlista]

Egy adott soron belül a kiíratás oszlopát határozza meg. A CPC képernyőjén jelenleg 25 sort és 40 oszlopot használunk. (Később megismerjük a 20 és 80 karakter szélességű képernyőt is.) A 'TAB' után zárójelben megadott kifejezés a kurzort az aktuális soron belül a kívánt pozícióra állítja és a 'PRINT' utasítás adatlistáját ott kezdi kiírni.

```
print tab(20) "*"          - a sor közepére tesz ki egy csillagot
```

Amennyiben a kifejezés értéke meghaladja a 40-et (az utolsó oszlop sorszámát), a kurzor a következő sorra ugrik. Ugyancsak a következő sorra kerül a kurzor, ha a kívánt pozíción már túlhaladtunk:

```
print tab(10) "*" tab(20) "*" tab(1) "*"  - a harmadik csillag
                                            új sorba kerül
```

Amint látjuk, egy 'PRINT'-et több 'TAB' is követhet, esetleg ';' (pontosvesszővel) elválasztva:

```
print tab(1) "cica"; tab(10) "kutya";      - nem kezd új sort!
```


A képernyő tetszőleges pozíciójára helyezhetjük a kurzort a **'LOCATE'** utasítással. Két paraméter követi: a kívánt oszlop (x) és a kívánt sor (y) száma. Mindkettő számozása 1-től indul, a soroké 25-ig tart, az oszlopoké pedig 40-ig. (20 és 80 karakteres képernyőmódban természetesen 20-ig vagy 80-ig.) Mindkét számozás a képernyő bal felső sarkából indul.

```
locate 20,12 : print"kozep"              - a képernyő közepére ír
```

A 'LOCATE' önálló utasítás, csak a kurzor helyének kijelölésére szolgál: a következő 'PRINT' a kívánt helyen kezdi a kiírást. 'PRINT' nélkül a 'LOCATE'-nak nincs látható hatása.

```
locate 1,1 : locate 10,10 : ?"tiz/tiz"   - az első 'LOCATE'-et
                                           észre sem vettük!
```

1.5.3 Ciklusképző utasítások

Új kulcsszavak: **FOR ... TO ... STEP**
NEXT

Ciklusnak nevezzük azt a programrészletet, amelyet a program futása során egymás után többször is végre kell hajtani. Már láttuk, hogy a 'GOTO' utasítással is vissza tudunk ugratni egy korábbi utasításra, hogy az újra és újra végrehajtódjék, ám így végtelen ciklusba kerültünk, amiből csak a program leállítása árán szabadulhattunk. A ciklusképző utasításokkal előre meghatározhatjuk a kívánt utasítás vagy utasításcsoport végrehajtásának a számát.

```
10 for ennyiszer = 1 to 10          - 'ennyiszer'= ciklusváltozó
20 print "pont tizszer irok ";      - szóköz a szöveg végén!
30 next ennyiszer                   - ismételd ennyiszer!
```

A fenti program pontosan tizszer hajtja végre a 20-as sor 'PRINT' utasítását, majd magától leáll. A 20-as sor a ciklus magja. A ciklusmag itt egysoros, de tetszőleges számú sor kezelhető ciklusmagként. Elképzelhető ciklus ciklusmag nélkül is:

```
10 locate 10,12                     - lehetőleg középre
20 print "** Villogo felirat **"    - ird ki!
```

```
30 for varj= 1 to 500          - számolj el 500-ig!
40 next varj                   - addig vár a program
50 cls                         - töröld a képernyőt!
60 for varj= 1 to 500          - számolj el 500-ig!
70 next varj                   - addig vár a program
80 goto 10                     - kezd újra az egészet!
```

A fenti példában két ciklus is van, mindkettő ciklusmag nélkül: a 'FOR' és a 'NEXT' utasítás között nem áll más utasítás, tehát a ciklusok nem csinálnak semmit. Mégis szükség van rájuk! A BASIC ugyanis a következőképpen kezeli a ciklusokat: 'FOR ciklusváltozó = 1 TO 500' értéket ad a 'ciklusváltozónak' először 1-et, végrehajtja a ciklusmagot (ha van) és találkozik a 'NEXT cikluváltozó' utasítással. Ennek hatására növeli eggyel a 'ciklusváltozót' és összehasonlítja a 'TO' után álló számmal, a végértékkel: ha a 'ciklusváltozó' értéke túllépte a végértéket, akkor a 'NEXT' utasítás utáni, ha még nem, akkor a 'FOR' utasítás utáni utasítással folytatja a program végrehajtását. Esetünkben tehát 500 értékadás és összehasonlítás történt, ami a program futását egy kis időre lelassította. A lassításra pedig szükség van: nélküle a felirat túl gyorsan villogna. Ciklusmag nélküli ciklust tehát <u>időhúzásra</u> használhatunk fel, a végértéktől függ a várakozási idő hossza. Próbálja meg a feliratot gyorsabban és lassabban villogtatni!

A ciklusváltozó tetszőleges numerikus változó lehet, melynek aktuális értéke folyamatosan változik, eddigi példáinkban eggyel növekedett. A ciklusváltozó aktuális értékét a ciklusmagban is felhasználhatjuk:

```
10 for szam = 1 to 10          - 'szam' = ciklusváltozó
20 print szam                  - írd ki, hogy éppen mennyi!
30 next szam                   - növeld eggyel és újra
40 rem folytatjuk!
```

Kiírattuk a ciklusváltozó aktuális értékeit: 1-től 10-ig. Ezekkel az értékekkel azonban számolhatunk is. Módosítsa a 20-as sort a következő mintájára:

```
20 print szam; "*"; szam; "="; szam * szam
```


Mit is csinál a 20-as sor? Kiírja a 'szam' aktuális értékét, szorosan mellé egy csillagot, szorosan mellé újra a 'szam' aktuális értéke jön, majd egy egyenlőségjel, azt követi a 'szam * szam' kifejezés kiszámított értéke, vagyis a 'szam' négyzete. A 'PRINT' utasítást nem zárta elválasztójel, tehát a következő kiírás új sorban jelenik meg.

Eddig a ciklusváltozónk egyesével növekedett, pedig erre nem is utasítottuk a BASIC-et. Lehetőség van a ciklusváltozó más lépésközönkénti növelésére és csökkentésére is, de azt már jeleznünk kell a **'STEP'** (lépés) szó után írt növekménnyel. Mind a kezdőérték, a végérték és a növekmény lehetnek kifejezések is:

```
10 a = 12
20 for c = a+a+a to a*2 step -a/12      - lehetne pl. tört is!
30 print c                              -'a+a+a'-tól 'a*2'-ig,
40 next c                                '-a/12'-es lépésekben
```

A ciklus 36-tól (12+12+12) tart 24-ig (2*12) -1-es lépésekkel (-12/12): 36...35...34...33...32... ...27...26...25...24. Negatív növekmény segítségével tehát visszafelé is számolhatunk. Mi van akkor, ha a növekmény ugyan negatív, de a végérték nem kisebb mint a kezdőérték vagy fordítva: ha a pozitív növekmény esetén a végérték kisebb mint a kezdőérték?

▶▶▶ Más BASIC változatoktól eltérően a CPC egyszer sem hajt végre olyan ciklust, amelynek feltételei már a 'FOR' utasításban lehetetlenek. A program a 'NEXT' utasítás után folytatódik.

A ciklus magja is tartalmazhat egy újabb ciklust. Ilyenkor beszélünk **egymásba skatulyázott** ciklusokról. Az egymásba skatulyázott ciklusok ciklusváltozóinak természetesen különbözőeknek kell lenniük. A ciklusváltozók értékei bárhol felhasználhatók:

```
1 rem EGYMASBA SKATULYAZOTT CIKLUSOK
10 cls
20 for sor = 1 to 20                    - 20 sorban írunk csillagot
30   for egysorban = 1 to sor           - hány csillag lesz a sorban?
```

```
40       print "*";              - minden sorban eggyel több
50    next egysorban             - fejezd be a sort!
60       print                   - üres 'print': új sort kezd
70 next sor                      - amíg 20 sort el nem értél
```

A programsorokban alkalmazott szóközök a könnyebb áttekintést segítik elő, a BASIC interpretert nem zavarják, mi viszont azonnal látjuk, melyik 'FOR'-hoz melyik 'NEXT' tartozik.

A ciklusok használatával kapcsolatos főbb szabályok:

- 'NEXT' után a ciklusváltozó elhagyható, a BASIC nyilvántartja, hogy melyik ciklust kell éppen lezárni.
- A ciklus magjában a ciklusváltozónak nem adhatunk új értéket!
- Tilos egy ciklusba kívülről beugrani, vagy abból a 'NEXT'-et kikerülve kiugrani!
- Egymásba skatulyázott ciklusok esetében minden belső 'FOR' után egy belső 'NEXT' következik. Az "átlapolás" helytelen ciklust eredményez:

<u>Helyes ciklusképzés:</u>

```
for  a
 for  b
  for  c
  ...
  next c
 next b
next a
```

<u>Helytelen ciklusképzés:</u>

```
for  a
 for  b
 next b
 ...
 for  c
 next c
next a
```

<u>Helytelen ciklusképzés:</u>

```
for  a
 for   b
  for  c
  ...
 next b
 next  c
next a
```

A ciklusképző utasítások igazi hasznát a tömbbökkel kapcsolatban látjuk majd, amikor nagy mennyiségű adatot kell hasonló módon kezelni. A sok adatot pedig először be kell juttatnunk a CPC memóriájába.

1.5.4 Adatbevitel kintről és bentről

Új kulcsszavak: **INPUT**
DATA
READ
RESTORE

Ezidáig minden változónak a programon belül adtunk értéket a 'LET' értékadó utasítással. Ezzel a módszerrel azonban nagyon is korlátozott programunk használhatósága. Legyen mondjuk egy programunk, amely kiszámítja két szám átlagát:

```
1 rem ATLAGSZAMITAS
10 a = 100 : b = 200
20 c = (a+b)/2
30 print "A ket szam atlaga: "; c
40 rem folytatjuk!
```

Új átlag kiszámításához módosítani kellene a programot. Van azonban egyszerűbb megoldás is: az **'INPUT'** utasítással a program végrehajtása közben a billentyűzetről adhatunk értéket a változó(k)nak. Módosítsa a 10-es sort az alábbira:

```
10 input a,b                     - kérj értéket a-nak és b-nek!
```

A program futtatásakor megjelenik egy kérdőjel a képernyőn és a CPC türelmesen várakozik. Adjon be egy tetszőleges számot és nyomja le az [ENTER] billentyűt.

```
? 12?Redo from start             - csináld újra elölről!
```

A hibaüzenet azt jelzi, hogy a számot a gép nem fogadta el, meg kell ismételni a beadást! Érthető, hiszen egy 'INPUT'-tal két változónak is kérünk értéket, írjunk be tehát két számot, vesszővel elválasztva.

```
? 12, 45 [ENTER]                 - ezt a két számot adtuk be
A ket szam atlaga:  28.5         - ez az átlaguk
Ready                            - a program véget ért
```

Tetszőleges mennyiségű átlagot számíthatunk ki, ha nem hagyjuk lefutni a programot, hanem visszaküldjük a 10-es sorra.

```
40 goto 10
```

Az 'INPUT' utasításnál tehát a gép egy kérdőjelet ír ki és várja az adato(ka)t. Ha egyszerre több adatnak kívánunk értéket beolvastatni, azokat egymás után, vesszővel elválasztva kell beadni. Fontos, hogy a beírt adatok típusa megegyezzen a kért adatok típusával: ha numerikus változónak próbálunk szöveges értéket adni (például a fenti ?-re cica-t adunk be), a már megismert 'Redo from start' hibajelzést kapjuk. Szöveges változóba természetesen lehetséges számot is beadni, de az a továbbiakban már csak szövegként kezelhető.

A képernyőn megjelenő magányos kérdőjel nem sok információt szolgáltat afelől, hogy mit is kell beadni. Bár egy, az 'INPUT' előtt elhelyezett 'PRINT'-tel tájékoztathatjuk a felhasználót, de van egyszerűbb megoldás is a CPC-n:

```
10 input "adjon be 2 szamot, vesszovel elvalasztva:"; a,b
```

A tájékoztató szöveget tehát az 'INPUT'-tal is ki lehet íratni. A kiíratás helyét az előző 'PRINT' utasítás határozza meg, de a 'LOCATE' utasítással is kijelölhetjük.

▶▶▶ Ha a szöveget ';' (pontosvessző) választja el a változótól, akkor a szöveg után egy '?' kérdőjel is megjelenik. Elválasztójelként ',' (vesszőt) használva a kérdőjel nem jelenik meg, ami olykor előnyösebb.

Számítsuk ki tetszőleges sok szám átlagát! A már ismerős ciklusképző utasításokkal ez nem lesz nehéz:

```
1 rem SOK SZAM ATLAGA
10 input "Hany szam lesz"; darab          - 'darab' szám lesz
20 for t=1 to darab                       - 1-től darab-ig
30 print "Adja  be a"; t; ". szamot: ";   - magyarázat ; mellé
40 input "", szam                         - nem kell kérdőjel
50 osszeg = osszeg + szam                 - új összeg
```



```
60 next t                                 - az utolsó darab-ig
70 atlag = osszeg / darab                 - átlagot számít
80 print "A szamok atlaga:"; atlag        - és kiírja
90 end                                    - a rend kedvéért
```

Az első 'INPUT'-tal megtudjuk, hány szám átlagát kell kiszámítani: annyiszor kell bekérni egy számot és az 'osszeg'-hez hozzáadni (első futáskor az 'osssszeg' nullával indul). A magyarázó szöveget külön 'PRINT'-tel írtuk, mert csak így lehetséges változót is kiíratni: a 't'-edik számot kérjük. A 40-es sor megmutatja, hogyan kerülhetjük el a kérdőjelet. Az átlag kiszámításához sem kell kommentár.

▶▶▶ Az 'INPUT' utasítással nemcsak a billentyűzetről kérhetünk be adatot, hanem a külső tárból is. A külső táron a CPC-k beépített magnetofonját vagy lemezmeghajtóját értjük. A külső tárral való kommunikációra a 'PRINT' (kivitel) és az 'INPUT' (behozatal) utasítások módosított alakjai szolgálnak. A külső tár használatáról a második szinten, a file-kezelésről szóló fejezetben lesz szó.

Gyakran szükséges nagyobb mennyiségű adatot elhelyezni a programon belül. Ez megoldható a közönséges 'LET' utasítással is, úgy azonban kissé hosszadalmas lenne. Sokkal rugalmasabb megoldást kínál a **'DATA'** és **'READ'** utasításpár. A 'DATA' utasítás után elhelyezzük adatainkat, amelyeket a 'READ' utasítással egyszerre, csoportosan vagy egyenként kiolvasunk. A kiolvasás tulajdonképpen értékadás:

```
1 rem ADATOK KIOLVASASA DATA SOROKBOL
10 data hetfo, kedd, szerda, csutortok, pentek   - ezek adatok
20 read nap$                                     - kiolvasom
30 print nap$                                    - kiíratom
40 goto 20                                       - mindet!
50 rem folytatjuk!
```

A 10-es sorban elhelyezett adatok szövegek, ezért a 20-as sorban szöveges típusú változónak adjuk értékül az első adatot. A 20-as sor jelentése: olvasd ki a soron következő adatot és

helyezd el a 'nap$' számára fenntartott helyen. Tulajdonképpen egy 'LET nap$ = "hetfo"' került végrehajtásra. A 'nap$'-t ezután tetszőlegesen felhasználhatjuk, pl. kiíratjuk. A BASIC egy belső mutatóval jelzi, hogy melyik adatelem a következő, így másodszorra már egy 'LET nap$ = "kedd"' utasításnak megfelelő olvasás következik. Ez így megy tovább, amíg az adatok el nem fogynak, akkor, mint a fenti példában is, a gép hibát jelez:

```
DATA exhausted in 20          - az adatok elfogytak, a 20-as sor
                                'READ'-jének nincs mit olvasnia
```

Két mód is van arra, hogy ezt a hibajelzést elkerüljük: a kiolvasást egy 'FOR-NEXT' ciklussal oldjuk meg és pont annyi adatot olvasunk ki, amennyi a 'DATA' sor(ok)ban található, vagy legutolsó adatként elhelyezünk egy tetszőleges jelzőt, amellyel minden kiolvasott adatot összehasonlítunk: ha megegyeznek, akkor már kiolvastuk az utolsó adatot is. Az összehasonlításos módszerről az 'IF-THEN-ELSE' utasítások tárgyalásánál lesz szó. Módosítsuk fenti programunkat, hogy hibajelzés nélkül érjen véget!

```
15 for napok= 1 to 5          - öt adatunk van
40 next napok                 - csak ötször fut le a ciklus
```

A 'DATA' sorokat a programban bárhol elhelyezhetjük, nem kell megelőzniük az első 'READ' utasítást. Több adat esetén több 'DATA' sorra is szükségünk lehet. Ha az első sort 'READ'-del kiolvastuk, a BASIC belső mutatója automatikusan a következő 'DATA' sor első elemére áll, legyen az bárhol a programban. Próbáljuk ki! Helyezzük el egy új 'DATA' sorban a hiányzó két napot és módosítsuk a kiolvasó-kiíratót ciklust:

```
50 data szombat, vasárnap     - az adatokat bárhol elhelyezhetjük
15 for napok = 1 to 7         - már hét napot kell kiolvasni
```

Az 'INPUT'-hoz hasonlóan, egy 'READ'-del több adatot is be lehet olvasni. A változóneveket vesszővel kell elválasztani egymástól, akárcsak a 'DATA' sorba levő adatelemeket:

```
10 data 1, 2, 3, 4            - a vessző csak elválaszt
```



```
20 read a, b, c, d            - a vessző itt is csak elválaszt
30 print a, b, c, d           - ez a vessző új oszlopot kezd!
```

Akárcsak az 'INPUT' esetében, a 'READ'-nél is ügyelni kell arra, hogy a beolvasandó adat típusának megfelelő típusú változót használjunk a 'READ' utasításban. Ha numerikus változóba próbálunk szöveges típusút beolvasni, a CPC 'Syntax error'-t jelez és javításra adja ki a kérdéses 'DATA'-sort.

A 'DATA' sorokban álló szövegeket általában nem kell idézőjelbe tenni. Ha azok vesszőt tartalmaznak, akkor az idézőjel használata feltétlenül szükséges, különben a gép azt elválasztó jelként értelmezné. Ugyancsak szükség van az idézőjelekre, ha a szöveg szóközzel kezdődik vagy azzal végződik, különben a szóközöket a 'READ' nem adja át a változónak.

Előfordulhat, hogy a program futása közben elölről akarjuk kezdeni a 'DATA' sorok (az adatmező) beolvasását. Erre a célra szolgál a **'RESTORE'** utasítás, amely a BASIC belső adatmutatóját újra az első 'DATA' sor első elemére állítja.

```
10 data "Ez ","negy ","szoveges adat-elem "   - szóközök a
20 data "szokozokkel! "                       - szövegek után!
30 read a$: print a$;                         - első adat
40 restore                                    - mutatót vissza!
50 for t=1 to 4                               - négy adat van
60 read a$ : print a$;                        - olvasd és írd
70 next t                                     - mind a négyet
```

▶▶▶ Sok más BASIC változattól eltérően a CPC lehetővé teszi hogy a belső adatmutatót egy bizonyos 'DATA' sorra állítsuk rá. A kiolvasás így tetszőleges 'DATA' sorral kezdődhet vagy folytatódhat. Az utasítás alakja: **'RESTORE** sorszám'.

```
10 data ezt soha nem olvasom el   - ez egy szöveges adat
20 data mindig ezt olvasom        - csak egy adat!
30 restore 20                     - a 20-as sor következik
40 read a$ : print a$             - kiolvastuk az utolsót
50 goto 30                        - újra és újra
```

1.5.5 Vektorok*, tömbök, mátrixok*

Új kulcsszó: **DIM**

Nem kell megijedni, mind a három fogalom hasonlót jelent: több azonos típusú adat, amelyek valamiképpen kapcsolódnak egymáshoz. Ezeket az adatokat nem külön-külön helyezzük el egyedi változókban, hanem egy nagy tömbben tároljuk őket, és egy közös névvel, indexes változóval, hivatkozunk rájuk.

Egy konkrét példa: a padsorban hatan ülnek. Nem tudjuk a nevüket, azonosítani azonban egyszerű őket: az első, a második, a harmadik és így tovább. Megtudtuk, hogy a harmadikat Kiss Péternek hívják. Elraktározzuk a nevét egy szöveges változókat tartalmazó vektorban (egydimenziós tömbben):

```
padsor$(1) = "??????????"   - az első a padsorban
padsor$(2) = "??????????"   - a második a padsorban
padsor$(3) = "Kiss Peter"   - a harmadik, tudjuk a nevét
padsor$(4) = "??????????"   - a negyedik nevét sem tudjuk
```

A tömb egydimenziós, ahogy a padsor is csak hosszában terjed ki. A tömbelemekkel ugyanúgy dolgozhatunk, mint a változókkal: értéket adhatunk nekik, kiírathatjuk őket stb. A már megismert 'FOR-NEXT' és 'READ-DATA' utasításokkal azonban ezek a műveletek sokkal egyszerűbben elvégezhetők, mint ha egyedi változókról lenne szó:

```
1 rem TOMBOK KEZELESE
10 data Kovacs Jeno, Horvath Pal, Kiss Peter      - hatan ülnek
20 data Petra Dezso, Sebaj Tobias, Rozsa Sandor     a padsorban
30 for n=1 to 6                                   - sorra beolvas
40   read padsor$(n)                              - az n-ediket
50 next n                                         - 1-től 6-ig
60 for t=1 to 6                                   - sorra kiírja
70   print padsor$(t)                             - a t-ediket
80 next t                                         - 1-től 6-ig
90 rem folytatjuk!
```

Egy sorban is megoldható akár egy több száz elemes tömb

elemeinek értékadása (beolvasása) vagy kiíratása, ha például a 30-40-50-es sorokat összevonjuk:

```
for n=1 to elemszam : read tomb(n) : next n
```

Kezelhetjük a tömb elemeit egyenként is. Folytassuk fenti programunkat úgy, hogy az csak a kívánt nevet írja ki:

```
90 cls                                            - tiszta képet
100 input "hanyadik a padsorban "; ennyiedik      - melyik név?
110 print padsor$(ennyiedik)                      - tessék!
120 goto 100
```

Az 'ennyiedik'-et <u>indexnek</u> hívjuk. Az index mutatja a tömbelem helyét a tömbben. Eddig csak egydimenziós tömböt kezeltünk, minden index egy számot tartalmazott. Használhatunk azonban két- vagy többdimenziós tömböket is. Az osztályban három padsor van. Az elsőt már benépesítettük, tegyük ezt meg a többivel is! Programunkból csak a 'DATA' sorokat hagyja meg és folytassa így:

```
30 data Simon Kalman, Kristaly Adam, Kovacs Hugo - a második sor
40 data Toth Bela, Poloska Tihamer, Balog Gabor  - szintén fiú
50 data Barat Agnes, Szabo Zsuzsa, Toth Monika   - a harmadikban
60 data Papp Kornelia, Fecske Bea, Szalai Anna   - csupa lány
```

Van tehát 18 nevünk, 3 sorban soronként 6 tanuló, helyezzük őket egy kétdimenziós tömbbe! A tömböt már nem 'padsor$'-nak nevezzük, hiszen nem egy padsorról van szó, hanem egy egész osztályról, amelynek a 3 padsor tulajdonképpen az egyik dimenziója: azok adják meg a szélességét. A padsorok egységes hossza (6 tanuló) adja meg az osztály hosszát. Az 'osztaly$' tömbünk egyik dimenziója tehát a 3 padsor, másik dimenziója pedig a 6 tanuló:

```
70 for padsor = 1 to 3                            - első padsor
80     for tanulo = 1 to 6                        - mind a hat
90         read osztály$(padsor,tanulo)           - tanulóját
100    next tanulo                                - olvasd be!
110 next padsor                                   - következő sor
```

Az egymásba skatulyázott ciklusokkal tehát egy kétdimenziós tömböt olvastunk be 'DATA' sorokból. Először az első padsor összes tanulóját, majd a másodikét, végül a harmadikét. Hasonló módon ki is írathatnánk a nevüket, de most inkább kérdezzünk rájuk szúrópróbaszerűen:

```
120 input "melyik padsorban, hanyadik tanulo "; p,t
130 print osztály$(p,t)
140 goto 120
```

A 130-as sor kiírja a p-edik padsor t-edik tanulóját. A kétdimenziós tömb elemeire hivatkozva az indexnek tehát már két számot, két koordinátát kell tartalmaznia. Mehetnénk magasabb dimenziókba is: egy emeleten 5 osztály van (harmadik dimenzió) és az iskolának 4 emelete van (negyedik dimenzió). Ha azt hiszi, hogy ezzel már vége a dimenzióknak, nagyon téved! A kerületben 17 iskola van (ötödik dimenzió), a városnak 10 kerülete van (hatodik dimenzió), az országban 111 város van ... A hetedik dimenziónál abbahagyjuk, elég ha tudatosítja: egy szép nagy tömbben elhelyezhető az ország összes tanulója és a megfelelő index-szel azonnal kikereshető az a-adik város, b-edik kerületének az c-edik iskolájának az d-edik emeletének e-edik tantermének f-edik padsorában a g-edik tanuló! Ekkora tömb azonban a CPC-ben valószínűleg már nem férne el, de még ha el is férne, a gép 'Subscript out of range' hibajelzéssel leállna, amikor találkozik a kerületben levő iskolák számával. A tömböt ugyanis nem dimenzináltuk

Amikor egy tömb elemeinek értéket adunk anélkül, hogy előzőleg közöltük volna a CPC-vel a tömb méretét, gépünk automatikusan egy 11 elemes tömböt tételez fel. Az elemek számozása nullától indul: 0, 1, ... 9, 10. Az osztály ebbe a tömbbe bőven belefért: a nulladik elemet az egyik dimenzióban sem használtuk fel és egyik dimenziónak sem használtuk ki mind a 11 elemét. Ez pedig helypazarlás. Célszerű tehát a kis tömböket is dimenzionálni, 10-nél nagyobb indexelés esetén pedig BASIC-ben kötelező a dimenzionálás a **'DIM'** utasítással:

```
5 dim osztaly$(3,6)          - ennyi eleme lesz a tömbünknek
```


A 'DIM' utasítás értelmezése: készülj fel egy akkora tömb fogadására, amelynek 'x' sora és 'y' oszlopa van (dimenzióktól függően kell egy vagy több paramétert megadni). A 'DIM' utasítás a tömb méretének meghatározásán kívül a tömbváltozó típusát is közli a géppel: esetünkben szöveges változóról volt szó. A 'DIM' utasításban szereplő tömbök indexei nemcsak konkrét számok (állandók), hanem változók, illetve aritmetikai kifejezések is lehetnek. Az aktuális értéknek természetesen pozitív számnak kell lennie akkor, amikor a 'DIM' utasítás hajtódik végre:

```
5 input "hany padsor,soronként hany tanulo van "; p,t
6 dim osztaly$(p,t)          - dinamikus dimenzionálás
```

A tömbök használatával kapcsolatban tudni kell a még:

- Tömböt újradimenzionálni nem lehet, a próbálkozás eredménye: 'Array already dimensioned' (már dimenzionált tömb) hibajelzés. Amennyiben a tömböt növelni vagy csökkenteni szeretnénk, vagy már egyáltalán nincs is rá szükségünk és csak a tárat foglalja, használjuk az **'ERASE'** utasítást.
- Indexelt változó (tömb) és nemindexelt egyszerű változó viselheti ugyanazt a nevet! Tehát a 'tomb' nevű egyszerű változónak semmi köze nincs a 'tomb(x,y)' elemeihez! Egy- vagy többdimenziós tömbök viszont ugyanazt a nevet nem kaphatják: 'tomb(x,y)' mellett már nem lehet 'tomb(z)'.

Lássunk még egy példát a tömbök hasznosításásra. Néha kis tömbbel is sok munkát takaríthatunk meg. Az iskolánál maradva, azt a feladatot kaptuk, hogy számoljuk meg a naplóban levő osztályzatokat, és készítsünk kimutatást arról, hogy melyikből hány van. Kézenfekvő megoldásnak kínálkozik minden osztályzatzathoz egy-egy változót rendelni és azokat növelni, amikor a megfelelő osztályzatból bejön egy darab. De hogyan mondjuk meg a gépnek, hogy mikor melyiket növelje? Elképzelhető így is:

```
      egyes volt ? - igen, akkor 'egyes'  = 'egyes'  + 1
nem:  kettes volt ? - igen, akkor 'kettes' = 'kettes' + 1
nem:  hármas volt ? - igen, akkor 'hármas' = 'hármas' + 1  stb.
```

Hosszadalmas, nehézkes megoldás, bár a BASIC-nek megvannak

a lehetőségei, hogy a fentiekhez hasonló szerkezetet alkossunk. Sokkal egyszerűbb egy tömböt képezni, amelynek minden eleme tárolja az indexének megfelelő osztályzatok számát. Egydimenziós tömbünk az 'osztalyzat' hat elemből áll: a nulladik elemben, mivel "nullás" osztályzat nincs, az összes osztályzat számát tároljuk, a többi elemben pedig az azonos sorszámú osztályzatokat gyűjtjük össze:

```
1 rem TOMBELEMEK mint SZAMLALOK
10 dim osztalyzat(5)                                  - 0-tól 5-ig
20 input "Az osztalyzat "; hanyas                     - hányas volt?
30 osztalyzat(hanyas) = osztalyzat(hanyas) + 1        - az növekszik
40 osztalyzat(0) = osztalyzat(0) + 1                  - összesen
50 goto 20                                            - a következő
60 rem folytatjuk!
```

Azért, hogy programunknak jelezzük, mikor fogynak el az osztályzatok, egy új utasítást kell használni, amelyet részletesen később ismertetünk:

```
22 if hanyas = 0 then goto 60          - ha 'hanyas' = 0 akkor
                                         ugorj az 60-es sorra!
```

Az osztályzatok végét tehát egy nulla beadásával jelezzük. Nem marad más hátra, mint kiíratni az egyes osztályzatok darabszámát:

```
60 print "Osszesen "; osztalyzat(0); "darab osztalyzat volt"
70 print "ezekbol"
80 for t = 1 to 5
90 print osztalyzat(t); "darab "; t; "-es"
100 next t
```

1.5.6 Változók, tömbök törlése

Új kulcsszavak: **CLEAR**
ERASE

A BASIC interpreter minden 'RUN' parancs végrehajtását a változók törlésével, nullára állításával kezdi. A numerikus változók értéke valóban nulla lesz, a szöveges típusú változók hossza lesz nulla: semmi$ = "" (az idézőjelek között nincs semmi, a szóköz nem semmi, annak már van hossza!). Változókat mi is nullázhatunk egyszerű értékadással, gyakran van szükség azonban a program összes változójának törlésére, erre szolgál a **'CLEAR'** utasítás. Végrehajtása a programot nem törli, az folytatódik tovább, azonban a korábbi változók értékére nem emlékszik. Tömbök újradimenzionálására is van lehetőség.

▶▶▶ A CPC BASIC lehetőséget ad arra, hogy a nemindexelt változók értékének meghagyásával csak a tömböket töröljük. Sok helyet már feleslegesen foglaló tömbök törlésén kívül hasznos ez az utasítás tömbök újradimenzionálása előtt is. Az utasítás: **'ERASE'** tömbváltozó neve dimenziók nélkül.

```
1 rem VALTOZOK ES TOMBOK TORLESE
10 dim tomb(10,10)                    - a 'tomb' maximális méretei
20 a = 111 : tomb(10,10) = 222        - értéket kapnak a változók
20 clear                              - nullázz mindent!
30 print a, tomb(10,10)               - mindent elfelejtett
40 dim tomb(10,10)                    - újra dimenzionálunk
40 a = 111 : tomb(10,10) = 222        - és értéket adunk
50 erase tomb                         - csak a 'tomb'-ot töröld!
60 print a, tomb(10,10)               - az 'a'-ra még emlékszik
```

1.5.7 Vezérlésátadás, szubrutinok

Új kulcsszavak: **GOSUB**
RETURN
ON ... GOTO
ON ... GOSUB

Mivel a BASIC programban szereplő utasítások a sorszámuknak megfelelő sorrendben hajtódnak végre, szükség lehet olyan utasításokra, amelyek lehetőséget adnak arra, hogy ettől eltérjünk. A 'GOTO' utasítást már megismertük és felhasználtuk végtelen hurok képzésére.

A 'GOTO' utasítás hatására a program futása a 'GOTO' után álló számnak megfelelő sorra adódik át, ha nincs ilyen sor, a 'Line does not exist' (a sor nem létezik) hibajelzést kapjuk.

```
10 print "*";                          - írj ki egy csillagot!
20 goto 10ez mintha itt se lenne        - nem szép, de elmegy!
```

A 'GOTO' utasítást általában döntéshozatal esetén használjuk, ezért az 'IF-THEN-ELSE' utasításokkal kapcsolatban lesz még szó róla. Nem jó programozói gyakorlat összevissza ugrálni egy programban 'GOTO'-k segítségével, mert akkor nem áttekinthető programunk logikája. Vannak olyan programozási nyelvek amelyek egyáltalán nem ismernek a 'GOTO'-hoz hasonló utasítást, ilyen például a FORTH.

A legtöbb program tartalmaz olyan utasítássorozatokat, amelyeket több alkalommal is végre kell hajtani. Természetesen minden alkalommal beírhatjuk őket a megfelelő helyre, de ez egyrészt kusza és nehezen áttekinthető programot eredményez, másrészt felesleges erő- és tárpazarlás. A BASIC lehetőséget ad arra, hogy a programot több kisebb, előre elkészített elemből állítsuk össze. Ezeket a szubrutinnak nevezett programrészeket csak egyszer kell megírnunk és amikor szükség van rájuk, egy speciális vezérlésátadó utasítással hívhatjuk meg őket. Amíg a 'GOTO' elugrik a jelzett sorra és úgy folytatja ott a programot, mintha mi sem történt volna, addig a szubrutint hívó **'GOSUB'** megjegyzi, hogy honnan ment el és oda is tér majd vissza, ha találkozik egy **'RETURN'** -nel.

Rajzoljunk négyzeteket a CPC képernyőjére! Egyelőre durva felbontással, '*' (csillagokkal) húzzuk a vonalakat. Egy vízszintes vonalat x1-től x2-ig a következő szubrutin húzhat meg:

```
99 rem VIZSZINTES VONALAT RAJZOLO SZUBRUTIN
100 for t = x1 to x2              - 't' = ciklusváltozó, növekszik
110 locate t,y : print "*";       - 'y' koordináta változatlan
120 next t                        - következő csillag
130 return                        - térj vissza, ahonnan jöttél
140 rem folytatjuk!
```

Szubrutinunknak 3 paramétert kell átadni, hogy megfelelően működjön: mettől (x1) meddig (x2) kell a vonalat húzni, milyen függőleges koordinátával (y). Ezeket a paramétereket a szubrutin felhasználja, de nem változtatja meg. A 't' ciklusváltozóval

azonban vigyázni kell: ha valahol a főprogramban is előfordul ugyanilyen nevű változó, annak értéke a szubrutin hatására megváltozhat!

```
199 rem FUGGOLEGES VONALAT RAJZOLO SZUBRUTIN
200 for t = y1 to y2
210 locate x,t : print "*";       - ';' hogy ne kezdjen új sort!
220 next t
230 return
```

A függőleges vonalat hasonló módon rajzoltatjuk meg. Itt is a 't' ciklusváltozót használjuk: ha már az előző rutin úgyis lefoglalja, felesleges újabb változót bevezetni.

Próbáljuk ki a szubrutinokat:

```
10 cls
20 x1 = 10 : x2 = 30 : y = 10     - paraméterek a vízszinteshez
30 gosub 100                      - rajzold ki és gyere vissza
40 y1 = 10 : y2 = 20 : x = 20     - paraméterek a függőlegeshez
50 gosub 200                      - rajzold ki és gyere vissza
90 end                            - vége: ne menj tovább!
```

A 20-as és 40-es sor feltölti adatokkal a szubrutin által várt változókat, a 30-as és 50-es sor pedig meghívja a szubrutint. A szubrutin 'RETURN' utasítása visszaadja a vezérlést a 'GOSUB' után álló utasításnak. Ha a 'GOSUB' után a sor folytatódik, vagyis ':' (kettőspont) után egy újabb utasítás áll, a program ott folytatódik, nem a következő soron! Példánkban azonban a 'GOSUB'-okat nem követi újabb utasítás, tehát a következő sorra tér vissza a 'RETURN'. Figyeljük meg, hogy az 'END' utasítás használata feltétlenül szükséges: az 50-es sor 'GOSUB 200'-ának vissza kell térnie valahová. A 90-es sor nélkül a következőre, a 100-asra térne vissza, az pedig egy szubrutin része. Erről viszont még nem tud a program, tehát végrehatja a sorokat egészen a 'RETURN'-ig. Ilyenkor az 'Unexpected RETURN' hibajelzéssel áll le: váratlan 'RETURN'-nel találkozik a 130-as sorban.

A programot lefuttatva egy széles kalapú 'T' alak jelenik meg. Használjuk fel másképpen az immár kipróbált és jól működő szubrutinokat. Ahhoz hogy négyzetet rajzoljunk, két vízszintes és két függőleges vonalat kell húzni: tudnunk kell például a bal felső pont koordinátáit (bx,by) és a négyzet oldalának a hosszát (hossz).

```
299 rem NEGYZETET RAJZOLO SZUBRUTIN
300 x1 = bx: x2 = bx + hossz - 1        - a vízszintesek
310 y = by : gosub 100                  - egy vonal fent
320 y = by + hossz - 1 : gosub 100      - egy másik lejjebb
330 y1 = by: y2 = by + hossz - 1        - a függőlegesek
340 x = bx : gosub 200                  - egyik vonal baloldalt
350 x = bx + hossz - 1 : gosub 200      - a másik tőle jobbra
360 return                              - térj vissza!
```

A négyzetrajzoló szubrutin tehát két másik szubrutint hív, összesen négy alkalommal. Szubrutin is hívhat másik szubrutint, itt is csak a visszatérésre kell ügyelnünk: az mindig 'RETURN' segítségével történjék. Ha esetleg 'GOTO'-val küldjük el szubrutinból a programot, az nem hibajelzést, hanem álmatlan éjszakákat okozhat, amíg a furcsán viselkedő programot kibogarásszuk!

Van tehát egy teljesen kész négyzetet rajzoló szubrutinunk, amelyet parancsmódban is kipróbálhatunk:

```
bx = 10: by = 10: hossz = 10: gosub 300 [ENTER]
```

A CPC a négyzet kirajzolása után parancsmódba tér vissza. Módosítsuk előbbi programunkat hogy sok-sok négyzetet rajzoljon:

```
20 bx = 1 : by = 1                          - mindig itt kezd
30 for egyrenagyobb = 1 to 25 step 2        - oldalhossz növekszik
40    hossz = egyrenagyobb : gosub 300      - és kirajzolja
50 next egyrenagyobb                        - most nagyobbat
```

Ha tovább kíván gyönyörködni az alkotásában és zavarja a képet feljebb toló 'Ready' felirat. segít egy '90 GOTO 90'!

Egy másik példa a szubrutin felhasználására. Itt két ciklus is fut: az egyik lefelé viszi a kis négyzetek kinduló pontját,

a másik pedig jobbra. A ciklusváltozók neve még csak véletlenül sem lehet 't'!

```
20 hossz = 4                                    - sok kis négyzet
30 for v = 1 to 36 step 5                       - külső hurok: jobbra
40     for f = 1 to 21 step 5                   - belső hurok: lefelé
50         bx = v: by = f: gosub 300            - itt a helye, rajz
60 next f,v                                     - így is lezárhatunk
                                                    két ciklust!
```

A 'GOTO' és a 'GOSUB' utasításnak van egy különleges használata, amely a program többirányú elágaztatását teszi lehetővé.

ON kapcsoló **GOTO** sorszám, sorszám, sorszám ...
ON kapcsoló **GOSUB** sorszám, sorszám, sorszám ...

A 'kapcsoló' lehet egy változó, vagy olyan aritmetikai kifejezés, amelynek értéke az utasítás végrehajtásának pillanatában 0 és 255 közötti. Ha az érték nagyobb vagy negatív, a CPC 'Improper argument' (helytelen argumentum) hibajelzéssel leáll. Az utasítás a kapcsoló aktuális értékét kiszámítva (legyen 'n'), kikeresi a sorszámlistából az n-edik elemet és arra a sorra próbálja átadni a vezérlést. Ha 'GOTO'-val megy oda, akkor nem tér vissza, 'GOSUB' esetén a szubrutint záró 'RETURN' a sorszámlistát követő sorra küldi vissza a programot. Ha 'n' nulla vagy nagyobb, mint a megadott sorszámok száma, akkor a gép a következő sorral folytatja a programot.

```
1 rem PELDA ON GOTO-ra
10 input "Melyik sorra menjek "; kapcs          - válasszon sort
20 on kapcs/100 goto 100,200,300,400            - 1-től 4-ig jó
30 print "Vagy nagyobb mint 400 vagy 0 volt!"   - különben baj
40 goto 10                                          van
100 print "100-as sor.": goto 10
200 print "200-as sor.": goto 10
300 print "300-as sor.": goto 10
400 print "400-as sor.": goto 10
```

Próbálja ki a programot különböző számok beadásával. Ha nem kerek százast ad válaszul a kérdésre, a CPC a 'kapcs/100'

kifejezés kiszámítása után az eredményt kerekítve használja fel, hogy a sprszámlista megfelelő elemét kikeresse. Cserélje ki a a 20-as sor 'GOTO'-ját 'GOSUB'-ra, és természetesen a 100-400-as sorokból is csináljon szubrutint: a 'GOTO 10' helyett 'RETURN' lesz. Figyelje meg, hogy a program a szubrutinokból mindig a 30-as sorra tér vissza! A visszatérés ez esetben arra a sorra történik, ahova a nullás vagy túl nagy kapcsoló juttatja a programot, ez pedig hibához vezethet! 'ON kapcs GOSUB' esetén ajánlatos az utasítás előtt ellenőrizni a kapcsoló értékét az alább ismertetett 'IF-THEN-ELSE' utasításokkal!

1.5.8 Feltételes utasítások, döntéshozatal

Új kulcsszavak: **IF ... THEN ... ELSE**
AND
OR

A számítógép intelligenciájának kulcsa abban rejlik, hogy a program végrehajtásának természetes menetét valamely feltétel teljesülésétől vagy nem teljesülésétől függően befolyásolni tudjuk. Az erre szolgáló utasítás általános alakja:

IF	feltétel	**THEN**	igaz ág	**ELSE** hamis ág
ha	ez-így-van	akkor	ezt-tedd	különben ezt-tedd

Több BASIC változatban az 'ELSE' utasítás és vele együtt a hamis ág hiányzik: a feltétel nem teljesülése esetén a program a következő sorral folytatódik. Milyen is a feltételes utasítás a gyakorlatban?

```
1 rem PELDA IF-THEN SZERKEZETRE
10 input "Adj be egy szamot"; szam
20 if szam < 0 then print "A szam negativ!": goto 10
30 if szam > 0 then print "A szam pozitiv!": goto 10
40 print "A szam nulla!": goto 10
```

Ha a bekért szám kisebb mint nulla, akkor a 20-as sorban levő feltétel igaz [szam < 0], tehát a 'THEN' utáni ágon megy tovább a program. Mivel 'ELSE' ág nincs, a feltétel nem teljesülése esetén a gép a következő programsorra ugrik. Ez a sor is hasonló, de itt a feltétel [szam > 0] az, hogy a szám nagyobb legyen, mint nulla. A feltétel nem teljesülése esetén tovább megy a program. A 40-es sorban már nincs mit vizsgálni: a szám egészen biztosan nulla. Figyeljük meg, hogy az 'igaz ág' a megfelelő 'PRINT' utasításon kívül még egy 'GOTO'-t is tartalmaz. Amennyiben nem akarjuk, hogy a gép újra kérdezzen, írjuk át őket 'END'-re vagy 'STOP'-ra, de mindenképpen gondoskodjunk arról, hogy a program természetes módon ne mehessen tovább, hiszen akkor végül is belefolyik a 40-es sorba, ahol feltétel nélkül nullának nyilvánítjuk a számot. Ezt a sort mindenképpen el kell kerülnünk.

'ELSE' ág beiktatásával és az 'IF'-ek egymásba skatulyázásával a fenti programot rövidebbé tehetjük, ám ettől programunk kevésbé áttekinthetővé válik:

```
10 input "Adj be egy szamot"; szam
20 if szam < 0 then print "A szam negativ!" : goto 10 else if
szam > 0 then print "A szam pozitiv!" : goto 10 else print "A
szam nulla!": goto 10
```

A 20-as sor értelmezése: ha a szám kisebb mint nulla, akkor írd ki ..., majd menj a 10-es sorra, különben ha a szám nagyobb mint nulla, akkor írd ki ..., majd menj a 10-es sorra, különben írd ki ..., majd menj a 10-es sorra.

Úgy a 'THEN' ágon, mint az 'ELSE' ágon az összes lehetséges BASIC utasítás megengedett. A 'THEN' vagy 'ELSE' kulcsszó után álló 'GOTO' elhagyható, a BASIC a sorszámból megérti, hogy az adott sorra kell ugornia: pl. 'THEN 90'.

A feltételeket relációs (összehasonlító) kifejezésekkel adhatjuk meg, amelyek segítségével két értéket hasonlítunk össze. A CPC relációs műveletei a következők (feltételezzük, hogy a műveletek végrehajtása előtt 'LET x = 5' és 'LET a$="c" értéket állítottuk be):

Műveleti jel	Vizsgált reláció	Példa igaz-ra
=	egyenlő	x = 5
<>	nem egyenlő	a$ <> "cc"
<	kisebb	x < 6
<=	kisebb vagy egyenlő	x <= 5
>	nagyobb	a$ > "b"
>=	nagyobb vagy egyenlő	x >= 5

A relációs jelek szövegekkel kapcsolatban is használhatók, ez esetben a BASIC karakterről-karakterre egybeveti a két szöveget és azok ASCII kódját hasonlítja össze. A Függelékben megtalálható a teljes ASCII kódtáblázat. Megfigyelhető, hogy az lényegében a latin ábécé sorrendjében tartalmazza a betűket, ám a kisbetűk a nagybetűk után következnek. Az azonos helyen magasabb kódszámú karaktert tartalmazó szöveg tehát nagyobb:

```
"BB" > "BA"          - a második karakter kölönbözik
"B " > "B"           - "B " hoszabb: a szóköz is számít
"cica" > "Cica"      - kisbetűk kódszáma nagyobb!
```

A szöveges változók összehasonlíthatóságára épül az alábbi kis szótárprogram.

```
1 rem MAGYAR-ANGOL SZOTAR
10 restore                                  - elölről olvas
20 input "magyar szo "; szo$                - mit keressen
30 read magyar$, angol$                     - új szópár
40 if magyar$="*" then 100                  - szótár vége?
50 if magyar$<>szo$ then 30                 - nem egyezik!
60 print magyar$; "angolul: "; angol$       - megtalálta
70 goto 10                                  - új kérdés
100 print "Ezt a szot nem ismerem!"         - ilyen szó nincs
110 goto 10
200 data asztal, table, szek, chair, konyv, book, szoba, room
210 data ablak, window, ing, shirt, apa, father, anya, mother
220 data zokni, socks, kutya, dog, macska, cat, ajto, door
1000 data *,*
```

A 10-es sor 'RESTORE' utasitását minden keresés előtt végre kell hajtani, hogy mindig a szótár elejéről indulhassunk el.

A 30-as sor egy szópárt olvas be, az első szót magyarnak tekinti, a másodikat angolnak. A beolvasás után első teendő ellenőrizni, hogy elértünk-e már a szótár végére, ezt két '*' jelöli. Ha elértünk, akkor mindkét szó helyett csillagot olvastunk be, tehát a [magyar$ = "*"] igaz. Ez esetben irány a 100-as sor. Ha még nem vagyunk a szótár végén, akkor van egy szópárunk, aminek a 'magyar$'-a megegyezhez a keresett 'szo$'-val. Kényelmesebb inkább azt tekinteni igaznak, ha a két szó nem egyezik [magyar$ <> szo$], mert így egyszerűbb megfogalmazni az utasítást: olvass be új szópárt! Ha ez nem igaz, akkor a 'magyar$' és a 'szo$' megegyeznek, az 'angol$'-ban pedig a fordítás van.

A feltételek az összehasonlításokon kívül gyakran tartalmaznak logikai műveleteket is. A logikai műveletek a Második szint anyagát alkotják, itt csak a feltételekben használt **'AND'** (és) valamint **'OR'** (vagy) operátorokkal" foglalkozunk. A két kulcsszóval feltételeket kapcsolhatunk össze, fogalmazhatunk meg röviden és egyértelműen. Lássunk két példát:

```
10 input "valaszolj: igen vagy nem "; valasz$
20 if valasz$="igen" or valasz$="IGEN" then ...
30 if valasz$="nem" or valasz$="NEM" then ...
40 print "Nem ertem a valaszt!"
```

A válasz bekérésekor nem tudhatjuk, hogy a kezelő azt kis- vagy nagybetűkkel adja be. A gép számára pedig ez egyáltalán nem mindegy. Amint említettük, a kisbetűk kódszáma nagyobb, mint a nagybetűké, vagyis ["igen" > "IGEN"] és ["NEM" < "nem"]. Fel kell készülnünk mindkétféle válasz fogadására. Az 'OR' kulcsszó pontosan erre szolgál: vagy ez, vagy az; elég ha az egyik feltétel igaz és a 'THEN' ágon folytatódik a program.

Az 'AND' kulcsszó csak akkor tekinti igaznak az összekapcsolt feltételeket, ha mindkét feltétel igaz. Ellenkező esetben nem a 'THEN' ágon folytatódik a program. A következő példa ezt illusztrálja:

```
10 input "Adj be egy betűt! ", betu$
20 if betu$ >= "a" and betu$ <= "z" then ...
30 if betu$ >= "A" and betu$ <= "Z" then ...
```

```
40 print "Ez nem betu! "
```

Ha a 20-as sor mindkét feltétele teljesül, akkor a kapott karakter kisbetű: "a" és "z" között van, esetleg ezek egyike. Ha a 30-as sor mindkét feltétele teljesül, és csak akkor, a kapott karakter nagybetű. Amennyiben egyik soron sem folytatódott a 'THEN' ág, nem betű volt a kapott karakter, hanem valami más (számjegy vagy egyéb jel, pl. .,?!"#$ stb.).

1.5.9 Függvények és használatuk

Gyakran előfordul, hogy programunkban a matematikai képletekben használatos függvényeket (pl. színusz, négyzetgyök) kell alkalmaznunk. A legtöbb BASIC változat ún. belső függvényei tartalmazzák az alább ismertetett függvényeket. A CPC-n még több beépített függvény áll rendelkezésünkre, róluk, és a felhasználó által definiálható ún. külső függvényekről a Második szinten lesz szó.

Függvénynek nevezzük általában azt a BASIC kulcsszót, amely egy argumentumon* valamilyen műveletet végez és annak eredményét adja vissza. A függvényeket mindenhol használhatjuk, ahol konstansok, változók vagy kifejezések előfordulhatnak. Például a 'SQR(x)' függvény az 'x' argumentum négyzetgyökét adja eredményül. Amint az alábbi felhasználási lehetőségekből is látható, a függvény argumentuma lehet konstans, változó vagy kifejezés, de mindenképpen zárójelbe kerül:

```
print sqr(81)
a = sqr(b * b + c * c)
x = 39 : a = sqr(x - 3)
x = 100: y = 144 : locate sqr(x),sqr(y)
if sqr(x) > sqr(y * (z1 + z2)) then ...
on sqr(kapcs) goto ...
```

A BASIC belső (előre definiált) függvényeit azok argumentuma és eredménye, valamint az általuk végrehajtott műveletek alapján három csoportra oszthatjuk:

1. Aritmetikai függvények. Ezek argumentuma numerikus érték és eredményként is számértéket adnak. Közéjük sorolunk több olyan függvényt is, amelyeknek nincs matematikai megfelelőjük.

2. Szöveges függvények. Argumentumuk vagy eredményük (legtöbbször mindkettő) karakterlánc. Segítségükkel szövegeket szabdalhatunk szét, állíthatunk össze stb.

3. Speciális függvények, amelyek a számítógép vagy a BASIC interpreter működésével vannak kapcsolatban. Ezekre a CPC-n csak korlátozott mértékben van szükség, mivel számítógépünk fejlett BASIC-je mellett nagyon ritkán kell hozzájuk folyamodni. A speciális függvényekről a Harmadik szinten lesz szó.

1.5.9.1 Aritmetikai függvények

A numerikus kifejezésekben használatos aritmetikai függvények a következők:

ABS (x) - kiszámítja 'x' abszolút értékét.
Például abs(-3) = 3

ATN (x) ↓ - kiszámítja 'x' arkusz tangensét.

CINT (x) - 'x'-et az 'INT' függvényhez hasonlóan (1. ott) egész számmá alakítja. Ha a kapott eredmény nem esik a -32768 ... 32767 intervallumba, 'Overflow' (túlcsodulás) hibajelzés keletkezik.

COS (x) ↓ - kiszámítja 'x' koszinuszát.

EXP (x) - kiszámítja az 'e' alapú exponenciális függvény értékét az 'x' helyen.

FIX (x) - leválasztja az 'x' argumentum tizedespont utáni részét és a maradó egész részt adja eredményül.
Például fix(4.44) = 4, fix(-3.33) = -3.

INT (x) - kiszámítja az 'x'-nél nem nagyobb egész számot és azt adja eredményül. Vigyázat: lefele kerekít!
Például int(8.11) = 8, int(8.88) = 8,
int(-8.11) = -9, int(-8.88) = -9.

LOG (x) - kiszámítja az 'x' természetes alapú logaritmusát.

LOG10 (x) - kiszámítja 'x' tízes alapú logaritmusát.

RND - a 0 - 1 intervallumba eső álvéletlen számot ad eredményként argumentum nélkül vagy pozitív argumentummal.

Például szam = rnd, vagy szam = rnd(1).

Ha az argumentum értéke nulla, az előző véletlen számot adja vissza.

Például a = rnd: b = rnd(0) : rem a = b

Ha az argumentum értéke negatív, egy új véletlen sorozat első számát adja eredményül.

SGN (x) - az 'x' kifejezés előjele, eredménye:

-1, ha 'x' negatív, pl. sgn(-2.22) = -1

0, ha 'x' nulla, pl. sgn(2 - 2) = 0

+1, ha 'x' pozitív, pl. sgn(23.44) = 1

SIN (x) ↓ - kiszámítja 'x' szinuszát.

SQR (x) - kiszámítja 'x' négyzetgyökét. Ha 'x' aktuális értéke nem pozitív, 'Improper argument' (helytelen argumentum) hibajelzést kapunk.

TAN (x) ↓ - kiszámítja 'x' tangensét.

▶▶▶ A '↓' jellel megjelölt szögfüggvények alapállapotban, a CPC bekapcsolása után, radiánban kifejezett értéket adnak eredményül. Más BASIC változatoktól eltérően a CPC-nek két utasítása is van a szögfüggvények működésének befolyásolására:

DEG - minden további számitás szögfokban történik

RAD - minden további számítás radiánban történik

Például rad: print sin(180) eredménye -0.801152633

deg: print sin(180) eredménye 0

1.5.9.2 Néhány példa aritmetikai függvényalkalmazásra

Az alábbi példákban nem a geometriai függvények alkalmazását mutatjuk be (azokra nagyobb szükség van a nagyfelbontású grafika területén), hanem az általános problémák megoldásához szükséges fogásokra térünk ki. Első programunkkal írassuk ki az összes prímszámot 1-től 100-ig. A prímszámok azok a számok, amelyek csak önmagukkal és 1-gyel oszthatók maradék nélkül. Ha 2 után keressük a további prímszámokat, csak a páratlanokat

kell megvizsgálni. A program úgy dönti el, hogy egy szám prímszám vagy sem, hogy a számot elosztja az összes, már megtalált prímszámmal, amelyek kisebbek mint a szám négyzetgyöke. Ez a tesztelő programrész a 80-astól a 110-es sorig tart, 'x' a vizsgált szám (kettesével növekszik), 'n' az aktuális prímszámok száma. A megtalált prímszámokat a 'p(100)' tömbben gyűjtjük össze, mivel a "teszteléskor" szükség van rájuk.

```
10 CLS
20 PRINT "Az elso szaz prim szam"
30 PRINT "======================":PRINT
40 DIM p(100)
50 p(1)=2 : p(2)=3 : x=5 : n=3
60 PRINT "az 1 . prim szam:"; TAB(22); p(1)
70 PRINT "a  2 . prim szam:"; TAB(22); p(2)
80 FOR j = 2 TO n
90 IF x/p(j) - INT(x/p(j)) < 1E-10 THEN 150
100 IF SQR(x) < p(j) THEN 120
110 NEXT j
120 p(n) = x
130 PRINT "a "; n; ". prim szam:"; TAB(22); p(n)
140 n = n+1
150 x = x+2
160 IF n<101 THEN 80
170 END
```

Következő programunk két szám legnagyobb közös osztóját keresi meg, vagyis azt a legnagyobb egész számot, amellyel mindkét szám maradék nélkül osztható. Itt is szükség van az 'INT' függvényre:

```
10 REM ezt persze maskeppen is meg lehet oldani
20 CLS
30 PRINT " Legnagyobb kozos oszto"
40 PRINT "========================"
50 PRINT
60 INPUT "Adja be az elso szamot  :",a
70 INPUT "Adja be a masodik szamot:",b
80 IF a<1 OR b<1 THEN 60
90 c = b : b =a-b*INT(a/b) : a = c
```

```
100 IF b<>0 THEN 90
110 PRINT : PRINT "A legnagyobb kozos oszto: ";c
115 PRINT"------------------------------"
120 GOTO 50
```

Elágazás a szám előjelétől függően

Az 'SGN' függvény leggyakrabban az 'ON ... GOTO/GOSUB' utasításokkal kapcsolatban fordul elő és lehetővé teszi, hogy a program futását egy szám előjelétől tegyük függővé:

```
10 input "Adjon be egy számot: ", szam
20 on sgn(szam)+2 goto 100,200,300        - sgn() = -1, 0, +1
30 rem ide soha sem johet!!                 igy  1, 2, 3 lesz
100 print "A szam negativ!": stop
200 print "A szam nulla!": stop
300 print "A szam pozitiv!": stop
```

Véletlen számok előállítása

A CPC véletlenszám-generátorát a **'RANDOMIZE'** utasítással állíthatjuk új kezdőértékre. Ha az utasításnak nem adunk paramétert, a program INPUT-szerűen lekérdezi azt: 'Random number seed ?' (véletlen szám mag?). A beadott válasz egy új véletlenszám-sorozatot indít el. Ugyanezt programból is megtehetjük a 'RANDOMIZE' utasítás után beírt paraméterrel. A véletlenszám-sorozat elemei mindig ugyanazok a számok lesznek. Ezt a lehetőséget programok belövésekor használhatjuk ki. Legtöbbször programunk minden futtatásakor más és más véletlenszám-sorozatra van szüségünk, ezért paraméterként a CPC belső órájának aktuális értékét adjuk meg, ezt a **'TIME'** változó tárolja. Mivel a gép bekapcsolásától az utasítás végrehajtásáig eltelt idő alkalmanként eltérő lesz, a véletlenszám-generátor is eltérő kezdőértékkel fog indulni. Véletlen számokat használó programunk első sora legyen tehát:

```
1 rem PELDA VELETLEN SZAMOKRA
10 randomize time
```


Az 'RND' függvény által szolgáltatott szám nullától egyig terjedő (azt el nem érő) tizedes tört. Felhasználásának egyik lehetősége:

```
20 if rnd<0.5 then 100 else 200      - a program véletlenszerűen
100 print "*"; : goto 20              folytatódik a 100-as vagy
200 print " "; : goto 20               200-as soron
```

Gyakrabban van szükség 1 és egy felső határ közötti véletlen egészekre. A kiszámításukhoz szükséges formula:

véletlenszám = int(rnd * felsőhatár) + 1

Szimuláljunk kockadobást!

```
1 rem KOCKADOBAS SZIMULACIO
10 randomize time                    - teljesen véletlen sorozat
20 vszam = int(rnd * 6) + 1          - 1-től 6-ig, de melyik?
30 print vszam;                      - kiírjuk
40 goto 20                           - jöhet a következő
```

Ellenőrizzük a véletleszám-generátort! Ehhez minnél több véletlen számra van szükség, ezer már elég lesz. Ha 1-től 10-ig állítunk elő véletlen számokat, ezerből mindegyikre kb. 100-nak kell jutnia, ha a generátor jól működik.

```
10 randomize time                    - minden futásnál új!
20 dim szam(10)                      - a tíz véletlen számnak
30 for t= 1 to 1000                  - ezerszer talál ki
40     vszam = int(rnd * 10) + 1
50     szam(vszam) = szam(vszam) + 1
60 next t
70 for t= 1 to 10                    - kiíratjuk az eredményt
80 print t; "-bol "; szam(t); "darab"
90 next t
```

Az eredmény eléggé egyenletes megoszlású. Azonban ha tovább növeljük a 30-as sor ciklusát, még egyenletesebb válik a véletlen számok eloszlása. Próbálja ki 'for t= 1 to 10000' -rel is!

Az 'INT' függvényre a CPC-n különlegesen gyakran van szükség. Mindenhol tanácsos használni, ahol egész számot kell megadni paraméterként. Ha egy utasítás paramétere vagy egy tömb indexe olyan kifejezés, amely értéke törtszám, a BASIC interpreter egész számot állit elő belőle. Az előállitás módja azonban gépenként változik.

▶▶▶ A CPC BASIC interpretere belső típuskonverzió esetén általában szabályos kerekitéssel dolgozik, amikor az egésztípusú paraméterek értékét kiszámolja. Más BASIC interpreterek gyakran az 'INT' funkciót használják fel ilyen esetben.

	CPC BASIC	sok más BASIC
szam(5.1) =	szam(5)	szam(5)
szam(5.6) =	szam(6)	szam(5)

Ezt a különbséget figyelembe kell venni. Az 'INT' funkció alkalmazása kizár minden hibalehetőséget:
szam(int(5.1)) = szam(int(5.6)) minden BASIC változatban.

A számitógép véletlenszám-generáló képességét leginkább szimulációs és játékprogramokban használhatjuk fel. Szolgáljon például az ismert TORPEDÓ játék CPC-re irt változata. A játék célja: egy 10*10-es négyzetrácson elrejtőzött tengeralattjárót kell eltalálnunk. A program minden leadott lövés után kiszámitja a lövés becsapódása és a tengeralattjáró közötti távolságot (200-as sor) és azt visszajelzi, ennek alapján irányithatjuk további lövéseinket. Ha majdnem találtunk (a távolság csak 1), a tengeralattjáró véletlenszerűen új helyet választ magának, ezt jelzi a "Hopla..." felirat, vagyis kezdhetünk mindent elölről.

```
10 REM ***********
20 REM *         *
30 REM * TORPEDO *
40 REM *         *
50 REM ***********
60 RANDOMIZE TIME
70 CLS
80 PRINT "   0  1  2  3  4  5  6  7  8  9":PRINT
90 FOR i=0 TO 9
100 PRINT  i;". . . . . ."
110 PRINT
120 NEXT i
130 s=1
140 x=INT (RND*10)
150 y=INT (RND*10)
160 LOCATE 1,23:PRINT "                                        "
170 LOCATE 1,23:INPUT"sor: ", h
180 LOCATE 24,23:INPUT"oszlop: ", v
190 LOCATE 3*v+3,2*h+3: PRINT s
200 d=INT(SQR((x-h)*(x-h)+(y-v)*(y-v)))
210 IF d=0 THEN 250
220 s=s+1
230 IF d=1 THEN LOCATE 1,24:PRINT "Hopla ...                ":GOT
O 140
240 LOCATE 1,24:PRINT "tavolsag: ";d:GOTO 160
250 CLS : LOCATE 4,12
260 PRINT "Gratulalok a";s;". loves talalt!"
270 END
```

A felhasznált változók: s - a lövések száma (1-ről indul)
x,y - a hajó véletlen koordinátái
h,v - a leadott lövés koordinátái
d - a kettő közötti távolság

1.5.9.3 Karakterláncok és szövegkezelő függvények

Az adatfeldolgozásban a szövegeknek igen fontos a szerepük. Az a számitógép, amelyik szövegek feldolgozására nem képes, nem több, mint egy óriási zsebszámológép. Ezért minden valamirevaló BASIC változatnak sokféle szövegkezelő függvénye van.

Az alábbi függvények és utasitások közös vonása, hogy karakterrekkel, karakterláncokkal kapcsolatban használjuk őket. Ennek ellenére néhány függvény numerikus argumentumot kíván, vagy az általa szolgáltatott eredmény lesz szám. Ha egy függvény eredményül karakter(lánco)t szolgáltat, a neve '$' (dollárjellel) végződik, ebben is hasonlit a szöveges típusú változók jelölésére. A numerikus eredményt adó függvényeknek, amint láttuk, nincs külön azonositójuk.

Karakterláncokkal használatos fontosabb függvények abécé sorrendben. ('A$' tetszőleges szöveges változó vagy konstans, 'n' és 'k' tetszőleges numerikus kifejezés, változó vagy konstans.)

ASC (A$) - megadja a szöveg első karakterének ASCII kódját.

CHR$ (n) - előállítja az 'n' értékének megfelelő ASCII kódú karaktert.

INKEY$ - az utoljára lenyomott billentyűt adja értékül, ha ilyen nem volt, üres szöveget kapunk vissza: "".

LEN (A$) - a szöveg hosszát adja eredményül

**LEFT$** (A$,n) - értékül adja a szöveg első 'n' karakterét (a bal oldalát).

**MID$** (A$,k,n) - értékül adja a szöveg 'n' karakter hosszúságú részét a 'k' pozíciótól kezdve (a közepét).

**RIGHT$** (A$,n) - értékül adja a szöveg utolsó 'n' karakterét (a jobb oldalát).

STR$ (n) - 'n' numerikus változót vagy konstanst karakterlánccá alakítja. Az első karakterpozíció az előjel számára lesz fenntartva, pozitív előjel esetén szóköz jelenik meg.

STRING$ (n,k) - 'n' hosszúságú karakterláncot állít elő 'k' ASCII kódszámú karakterekből.

VAL (A$) - az 'A$' karakterláncot numerikus értékké alakítja át. Amennyiben 'A$' számként nem értelmezhető, a visszaadott érték nulla.

1.5.9.4 Példák a szövegkezelő függvények használatára

A karakterláncok (szöveges típusú változók) összehasonlításával kapcsolatban előfordult az ASCII kód fogalma. A karakterláncok minden elemének, karakterének, egy 0-255 közötti kódszám felel meg. Ezt a számot adja meg az **'ASC'** függvény.

```
10 input "Adj be egy karaktert! ", betu$
20 kod = asc(betu$)
```



```
30 print "Az "; betu$; " karakter kodja: "; kod
40 goto 10
```

Ha a 10-es sor 'INPUT'-ja a betu$-ba nem egy karaktert, hanem egy hosszabb szöveget kap, a függvény akkor is működik: a szöveg első karakterének ASCII kódját adja vissza.

A 'ASC' függvény párja a **'CHR$'** függvény, amely egy kódszámból a hozzátartozó karaktert állítja elő: 'PRINT CHR$(65)' egy 'A' betűt ír ki a képernyőre, mivel az 'A' ASCII kódja 65. Három betűből (függetlenül attól, hogy azok magánhangzók vagy mássalhangzók) 26*26*26= 17576 "szót" lehet alkotni, mert mindhárom betűhelyen az angol ábécé 26 betűje állhat. A nagybetűk ASCII kódja 65-től 90-ig terjed ('A'-'Z'). Készítsünk egy programot, amely az összes lehetséges hárombetűs "szót" kiírja a képernyőre!

```
1 rem 17576 HAROMBETUS SZO
10 for a = 65 to 90                          - első betű 'A'-'Z'
20     for b = 65 to 90                      - második betű 'A'-'Z'
30         for c = 65 to 90                  - harmadik betű 'A'-'Z'
40 print chr$(a);chr$(b);chr$(c);" ";        - egy kombináció + " "
50 next c,b,a                                - 26 * 26 * 26 -szor
```

A 40-es sorban a 'PRINT'-en belüli ';' (pontosvesszők) elhagyhatók, csak a sort záró ';' szükséges ahhoz, hogy a kiírás a szóköz (" ") után folytatódjék. A szavakat nem jegyeztük meg, csak kiírattuk őket. Nincs azonban akadálya, hogy a karakterekből összeállított szót egy változónak adjuk át. Ehhez az egyetlen stringművelet, a konkatenáció vagyis összekapcsolás lesz a segítségünkre, a művelet jele '+'.

```
10 szo$= chr$(65) + chr$(66) + chr$(67)      - "A" + "B" + "C"
20 print szo$                                - "ABC"
```

A konkatenációnak semmi köze a számértékek összeadásához, a művelet tulajdonképpen az összevonandó szövegekből egy új szöveges típusú változót hoz létre.

```
10 a=111 : b=222                        - numerikus változók
20 c=a+b : print c                      - 111 + 222 = 333
30 a$="111" : b$="222"                  - szöveges változók
40 c$=a$+b$ : print c$                  - "111" + "222" = "111222"
```

A 'CHR$' függvény tulajdonképpen egykarakternyi szöveget állit elő. A **'STRING$'** függvény pedig tetszőleges hosszúságú, max. 255 karakterből álló szöveget ad eredményül. Első argumentumának a szöveg hosszát, másodiknak pedig a kívánt karaktert kell megadni. A második paraméter lehet egy idézőjelben álló karakter vagy annak ASCII kódja. A következő két parancs ugyanazt a cselekvést hajtja végre: kirajzol egy sor csillagot.

```
print string$(40, "*")                  - 40 darab "*"
print string$(40, 42)                   - 40 darab chr$(42)
```

A szöveges változók legnagyobb előnye a számokkal szemben az, hogy tetszés szerint szabdalhatjuk őket: levághatjuk az első vagy utolsó 'n' számú karaktert, vagy kivághatunk egy darabot a szöveg közepéből is. Az már csak tőlünk függ, hogy a kivágott darabot értékül adjuk egy másik szöveges változónak, vagy azonnal felhasználjuk. A szövegek bal oldalát a **'LEFT$'** függvény segítségével vághatjuk le:

```
1 rem SZOVEGEK SZABDALASA
10 input "Adj be egy szot határozott nevelovel! ", szoveg$
20 if left$(szoveg$,2)= "a " or left$(szoveg$,3)= "az " then 40
30 print "Nem jot adtal be!" : goto 10
40 rem folytatjuk!
```

Futtatáskor a 'szoveg$'-ba kapunk egy szöveget, amely lehet felhasználható, pl. "a kutya", "az asztal", vagy rossz, téves beadás, pl. "na mi ez?", "nem irok be semmit". Meg kell tudnunk, hogy a szöveg határozott névelővel kezdődik-e, és ha nem, akkor újra kérdezünk. A 20-as sor ezt elvégzi. Ha a beadott szöveg (szoveg$) első két betűje "a " vagy az első három betűje "az ", akkor a szöveg megfelel a kívánalmaknak: a 'THEN'-ágon haladunk tovább a 40-es sorra. Ellenkező esetben ('ELSE'-ág híján) a program a következő sorra lép, ami figyelmeztet és visszaküld az 'INPUT'-hoz.

Kaptunk tehát egy szót névelővel, de mit csináljunk vele? Vágjuk le a névelőt! Ehhez szükségünk lesz a szöveg hosszára is. A **'LEN'** függvény segítségével megkaphatjuk. A függvény eredményül adja a szöveg karaktereinek számát, beleértve a szóközöket is. Üres szöveg hossza nulla: len("")=0.

```
40 hossz = len(szoveg$)                 - 'hossz' = a szöveg hossza
```

Le kell vágnunk az első két vagy három karaktert egy szövegből úgy, hogy a maradékra van szükségünk. A szöveg jobb oldalát adja vissza a **'RIGHT$'** függvény, ahol a jobb oldalon levágandó karakterek száma a második paraméter:

```
50 if left$(szoveg$,2)="a " then szo$=right$(szoveg$,hossz-2)
   else szo$=right$(szoveg$,hossz-3)
```

De mekkora lesz a hossza a levágandó jobb oldalnak? Nem nehéz kiszámitani a 'hossz' és a névelős bal oldal hosszából. Sajnos nem jegyeztük meg még korábban, hogy milyen hosszú a névelős rész (2 vagy 3 karakter), így újra hasonlítgatnunk kell. Az 50-es sor értelmezése: ha a szöveg$ "a "-val kezdődik, akkor állits elő belőle egy új karakterláncot (szo$), amely a szoveg$ jobb oldalából 'hossz-2' karaktert tartalmaz. Ha a szoveg$ nem "a "-val kezdődik, akkor csak "az "-zal kezdődhet, tehát a szo$ a szoveg$ hosszánál 3 karakterrel lesz rövidebb.

```
60 print "A szo névelo nélkul: "; szo$
```

Programunk nem készült fel a nagybetűs beadásra, így pl. nem fogadja el "A kutya"-t. A szükséges változtatás elvégzését az Olvasóra bizzuk. Az értelmetlen, de formailag helyes beadások pl. "az brbrbrbr" kiszűrése sajnos lehetetlen.

A 'LEFT$' és 'RIGHT$' függvények könnyen helyettesithetők a szövegek közepét kivágó **'MID$'** függvénnyel. A 'MID$' az egyik leghatásosabb szövegkezelő függvény, segitségével a szöveg bármely karakterpozíciójától kezdve tetszőleges hosszan vághatunk ki darabokat:

```
  A$="ABCDEFGH"
  print mid$(A$,3,2)                    - a 3.-tól 2 karakter:  "CD"
```

```
print mid$(A$,3)              - a 3.-tól a végéig:    "CDEFGH"
                                = right$(A$,len(A$)-3)
print mid$(A$,1,2)            - az 1.-től 2 karakter: "AB"
                                = left$(A$,2)
```

▶▶▶ A CPC BASIC-je megengedi a 'MID$' használatát az egyenlőségjel bal oldalán is. Segítségével szövegek belsejébe helyezhetünk más szövegeket, felülírva azok előző tartalmát:

```
nev$="Kovacs Karcsi"                  - átírjuk a "Kar"-t
mas$="Jan"                            - erre
mid$(nev$,8,3)= mas$ : print nev$     - a 8. pozíciótól
                                        3 karaktert cserélt
```

Gyakran szükség van arra, hogy számokat szöveggé alakítsunk át és fordítva. A számoknak az az előnyük, hogy számolhatunk velük, a szövegeket viszont tetszés szerint szabdalhatjuk, és ezt jó lenne a számokkal is megtenni. Két függvény szolgálja a konverziót a numerikus és a szöveges változók között. A **'STR$'** függvénnyel bármely számot karakterlánccá alakíthatunk át.

```
1 rem SZAMBOL SZOVEGET CSINALUNK
10 input "Adj be egy szamot! ", szam          - most nem akarunk
20 print szam; "szamjegyei: ";                  vele számolni
30 szam$= str$(szam)                          - szöveget csinál
40 for t= 1 to len(szam$)                     - elejétől végéig
50        print mid$(szam$,t,1); " ";         - egy-egy karakter
60 next t                                       utána szóköz
```

A 'STR$' segítségével létrejött karakterlánc mindig eggyel hosszabb lesz, mint ahány számjegyből állt a szám, mivel negatív számok esetén egy mínuszjelet, pozitív számok esetén pedig egy szóközt is tartalmaz. Tehát 'len(str$(123)) = 4' csakúgy mint 'len(str$(-123)) = 4'

Szövegek számmá való átalakítása a **'VAL'** függvénnyel végezhető el. A BASIC interpreter addig próbál számot csinálni a karakterláncból, amíg egy nemszámjegy karakterrel találkozik, kivéve a szöveg kezdetén álló mínuszjelet. a számjegy karakterek között álló "e" vagy "E" karaktert (normál alak!), és a

tizedespontként értelmezhető első pontot:

```
print val("123")          - 123 , végig jó
print val("12a")          - 12  , az "a" már nem számjegy
print val("a12")          - 0   , nem kezdődik számjeggyel
print val("-12")          - -12 , a minusz-jelet elfogadja
print val("1-2")          - 1   , műveletet nem hajt végre
print val("1E2"           - 100 , mantissza E kitevő
print val("1 e 2")        - 100 , így is elfogadja !
print val("1.2")          - 1.2 , a tizedespont rendben van
print val("1.2.3")        - 1.2 , a második már értelmetlen
```

Tulajdonképpen az adatbevitelről szóló fejezetben lenne a helye az **'INKEY$'** függvénynek, hiszen a program futását meg nem szakító adatbevitelt teszi lehetővé. Az 'INKEY$' függvény eredménye egy egykarakter hosszúságú szöveg, amely az utasítás végrehajtásának pillanatában lenyomott billentyűnek megfelelő karakterből áll. Ha az adott pillanatban egy billentyű sem volt lenyomva, a szöveg hossza nulla: "".

```
10 a$=inkey$ : if a$="" then 10       - van lenyomott billentyű?
20 ?a$; "  kodja: "; asc(a$)          - írd ki, a kódját is!
30 goto 10
```

A 10-es sor felhasználható időhúzásként is: addig vár itt a program, amíg a$-ban egy lenyomott billentyűnek megfelelő karaktert el tud helyezni. Amíg nincs lenyomott billentyű, 'a$' hossza nulla, vagyis a$="". A "semminek" nincs ASCII kódja, ha tehát egy üres 'a$'-t ráeresztenénk a 20-as sor 'asc(a$)' függvényére, az hibajelzést okozna. Az 'INKEY$' használata sok esetben kényelmesebb, mint az 'INPUT', hiszen a beadást nem kell [ENTER]-rel lezárni. Az a hátránya, hogy csak egykarakter hosszúságú beadást kezel egyszerre, előnyként is felfogható: a beadott karakter azonnal megvizsgálható, a nemkívánt kiszűrhető vagy módosítható. És a beadott karakter nem jelenik meg magától a képernyőn! Ezt használja ki az alábbi, jelszót ellenőrző program:

```
1 rem JELSZO.BEVETEL ALCAZASSAL
10 jelszo$="csiga" : h=len(jelszo$)          - tetszőleges jelszó
20 cls
30 ?"Kerem a jelszot!"
40 for t=1 to h                              - ahány betű a jelszó
50   a$= inkey$: if a$="" then 50            - a$-ba a billentyű
60   if a$ <> mid$(jelszo$,t,1) then new     - azonos a jelszóval?
70   print chr$(int(rnd*96)+32);             - álcázás !
70 next t                                    - következő betű
80 rem minden rendben, tudta a jelszót!      - mehet tovább
```

A jelszót bevehetnénk 'INPUT'-tal is, de akkor azt bárki megláthatja! 'INKEY$' használatával minden lenyomott billentyűnek megfelelő karaktert sorra egybevetünk a 'jelszo$' karaktereivel ('MID$' függvény) és ha csak egy is nem egyezik meg, azonnal 'NEW', vagyis a program kitörli önmagát. A 70-es sor az álcázás céljából került be a programba. Ha valaki a hátunk mögött leselkedik, miközben a jelszót beírjuk, hadd olvassa el, amit a képernyőn lát! A 'CHR$' függvénnyel kiíratunk egy véletlen számnak megfelelő karaktert, aminek természetesen semmi köze a jelszóhoz és a lenyomott billentyűhöz. A véletlent egy kicsit befolyásoljuk: a legkisebb kiírható karakter ASCII kódja 32, (a szóközről van szó: " "), tehát gondoskodnuk kell róla, hogy a 'CHR$' argumentuma 32-nél ne legyen kisebb. A szabvány ASCII karakterkészlet legnagyobb kódszámú karaktere a 127-es, véletlen számunk kódjának így 32 és 127 közé kell esnie, hogy az "álcázás" megfelelő legyen. A 'CHR$' függvény elfogad 32-nél kisebb (0 - 31) és 127-nél nagyobb (128 - 255) argumentumot is, azonban az első esetben nem kiírható, hanem géptől függő ún. kontrol karakterekről" van szó, a 127-nél nagyobb karakterek pedig - szintén géptől függő - grafikus ábraként jelennek meg, amelyeket a CPC-n nem lehet a billentyűzetről beadni.

▶▶▶ A CPC-n az 'INKEY$' függvény nem a végrehajtás pillanatában lenyomott billentyűt adja vissza, hanem egy puffer"-ből veszi elő az utoljára lenyomott billentyű kódját, ez pedig egy korábban lenyomott billentyűnek is megfelelhet. A billentyűzetpuffer működését részletesen a 2.7.1 pontban ismertetjük.

A fejezet utolsó előtti programja a legtöbb szövegkezelő függvényt felhasználja. A FENYUJSAG segítségével tetszőleges (max. 255 karakter hosszú) szöveget mozgathatunk a képernyőn. A szöveget az 'INKEY$' függvény segítségével vesszük be és 'b$'-ban tároljuk. A mozgatást a valódi fényújsághoz hasonlóan ismételt, de már kissé eltolt kiíratással hozzuk létre. Mindig egy újabb darabot íratunk ki a szövegből a 'MID$' függvénnyel, majd az iránynak megfelelően új szöveget állítunk össze a 'LEFT$' és a 'RIGHT$' függvényt felhasználva. A szöveg mozgatásának sebességét egy időhúzó hurok szabályozza.

```
10 REM ************
20 REM *          *
30 REM * FENYUJSAG *
40 REM *          *
50 REM ************
60 CLS
70 b$="" : disz$=STRING$(38,"=")
80 PRINT "Adja be a szoveget:
90 PRINT
100 a$=INKEY$:IF a$="" THEN 100
110 IF a$=CHR$(13) THEN b$=b$+" ":GOTO 140
120 b$=b$+a$
130 PRINT a$;:GOTO 100
140 PRINT:PRINT:INPUT "Balra (B) vagy jobbra (J) menjen "; irany
$
150 PRINT
160 INPUT "Sebesseg (1 - 10) "; seb
170 seb = 10 - seb
180 CLS
190 LOCATE 8,6: PRINT"******** FENYUJSAG ********"
200 LOCATE 2,10: PRINT disz$: LOCATE 2,14:PRINT disz$
210 t=LEN(b$)
220 IF t<38 THEN b$=b$+" ": GOTO 210
230 LOCATE 1,12:PRINT " "+MID$(b$,1,38)+" "
240 IF LEFT$(irany$,1)="B" OR LEFT$(irany$,1)="b"  THEN b$=RIGHT
$(b$,t-1)+LEFT$(b$,1)
250 IF LEFT$(irany$,1)="J" OR LEFT$(irany$,1)="j"  THEN b$=RIGHT
$(b$,1)+LEFT$(b$,t-1)
260 FOR J=0 TO 20 * seb
```

```
270 IF INKEY$<>"" THEN RUN
280 NEXT J
290 GOTO 230
```

Játsszunk a betűkkel! A program a 250-es sorral kezdődő 'DATA' sor(ok)ból találomra kiválaszt egy szót, összekeveri a betűit, kiírja azokat, és várja a helyes megoldást. Figyelemre méltó a betűk összekeverésének módja a 140 - 190-es sorokban: a 'kerdes$'-ban épül fel az új szó a 'talald.ki$'-ból. A program szókincse könnyen bővíthető, csak arra kell ügyelni, hogy az első 'DATA' sor első adata a szavak száma legyen!

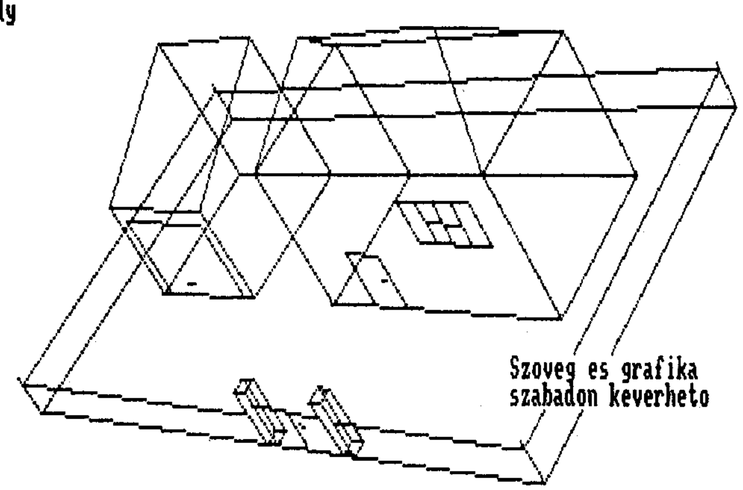
```
10 REM ****************
20 REM *              *
30 REM * KEVERT BETUK *
40 REM *              *
50 REM ****************
60 CLS
70 PRINT "Talald ki a szot!"
80 kerdes$ = "": valasz$ = ""                    'nullazni a valtozokat!
90 RESTORE: READ szavak.szama                    'ennyi szo van osszesen
100 FOR j= 1 TO RND * szavak.szama + 1           'veletlen szot kivalaszt
110 READ talald.ki$                              'az utoljara beolvasott
120 NEXT j                                       'szo lesz osszekeverve
130 hossz = LEN(talald.ki$) : megoldas$ = talald.ki$
140 FOR j= 1 TO hossz                            'elso betutol az utolsoig
150 n = INT(1 + RND * hossz)                     'veletlen betu
160 x$=MID$(talald.ki$,n,1): IF x$="*" THEN 150  ' * = mar volt!
170 MID$(talald.ki$,n,1) = "*"                   'bejelolni, hogy megvan
180 kerdes$ = kerdes$ + x$                       'epul a kevert szo
190 NEXT j                                       'az utolso betuig
200 PRINT
210 PRINT kerdes$
220 INPUT "", valasz$
230 IF valasz$ = megoldas$ THEN PRINT "Eltalaltad!": GOTO 80
240 PRINT "Probald meg ujra!": GOTO 200
250 DATA 6, cica, krokodil, elefant, computer, asztal, iskola
```


(második szint cimkép:)

```
MODE 2:  640 * 200 pont a felbontas,  egy sorban 80 karakter fer el.

A karakterkeszlet:  ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = >
 ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ↑ _ ` a b c d e f
 g h i j k l m n o p q r s t u v w x y z { | } ~ ▒

Ready
```



M á s o d i k s z i n t

A Második szint a CPC egyedi lehetőségeit ismerteti. Színes, grafikus, zenés programjai példát mutatnak arra, hogyan aknázhatók ki a gép szinte korlátlan lehetőségei. A CPC perifériáinak programozása is a Második szint témája.

Néhány önállóan futtatható program:

KARAKTERTERVEZŐ	2.2.4	MINI ORGONA	2.6.1
KÍGYÓ	2.3.4	NÉGY DAL SZTEREÓBAN	2.6.3
MOZGÓ EMBERKÉK	2.3.6	RAKÉTÁKAT KERÜLGETŐS	2.7.1
3-DIMENZIÓS HISZTOGRAM	2.3.8	MAGYAR KARAKTERKÉSZ-	
3-DIMENZIÓS KALAP	2.3.8	LET ÉS BILLENTYŰZET	2.7.2
SOKSZÖG RAJZOLÓ	2.3.8	REMTELENÍTŐ: BASIC	
FORGÓ GÖMB	2.3.8	PROGRAM MÓDOSÍTÁS	2.10.2.2
OVÁLIS MINTÁK	2.3.8	MINI ADATBANK	2.10.2.3
ÖRÖKNAPTÁR	2.4.2	SZOFTVER ÓRA	2.5.2
REAKCIÓIDŐ MÉRÉS	2.5.1		

2.1 A CPC BASIC ALTALANOS UTASITASAI ÉS FÜGGVÉNYEI

Ez a szakasz ismerteti azokat az általános utasításokat és függvényeket, amelyek nem találhatók meg minden BASIC változatban. Az új kulcsszavak többsége a programozás megkönnyítésére szolgál, más gépeken szubrutinnal, körülírással kell helyettesíteni őket.

2.1.1 Változók típusának deklarálása

Új kulcsszavak: **DEFINT**
DEFREAL
DEFSTRING

A változók típusát eddig a nevük után álló '%', '!' és '$' jelekkel határoztuk meg. A CPC BASIC lehetővé teszi, hogy egyes változónevek típusát előre deklaráljuk (bejelentsük). Az így deklarált változónevek után nem szükséges a fent felsorolt jeleket kitenni, a BASIC nélkülük is felismeri a változók típusát. A változók típusának deklarálása a változónevek első betűjére vonatkozik: például szöveges típusúnak deklarálhatunk minden 'S' betűvel kezdődő nevű változót a 'DEFSTR S' utasítással. A kívánt kezdőbetűket kétféle formában is megadhatjuk:

- a kezdőbetűket egymástól ',' (vesszővel) választjuk el
- kezdőbetű tartományt adunk meg: a kezdő- és végkaraktert '-' (kötőjellel) kapcsoljuk össze
- a két módszert keverhetjük is:

```
DEFINT I,J,K        - az I,J és K betűvel kezdődő változók egészek
DEFREAL  M-P        - az M,N,O és P betűvel kezdődő változók valós
                      típusúak
DEFSTR A,D,X-Z      - az A,D,X,Y és Z betűvel kezdődő változók
                      szöveges típusúak
```

Az előre deklarált típusú változók neve után a típusjelzés már nem szükséges, ha mégis kiteszünk egy típusazonosító jelet, az felülbírálja az előre deklarált típust:

```
10 defint a-z                - az összes változó egész típusú
20 a$="ez egy szöveg"        - a$ azonban szöveges: $-jel
30 a =1.111                  - 'a'-t egésznek deklaráltuk
40 print a$; a
```

Mivel az egész típusú változók kevesebb helyet igényelnek a gép operatív tárában és a velük való számolás is gyorsabban történik, egy 'DEFINT a-z' programjaink elején azok hasznára válhat. A szükséges valós típusú változók neve után ez esetben mindig ki kell tenni a '!' jelet! Próbálja ki az egész számok használata által okozott sebességnövekedést:

```
20 for t=1 to 10000                  - 10000-szer hajtsa végre
30   csinaljon.valamit =t/t          - számoljon is!
40 next t
```

A program több mint 29 másodperc alatt fut le. Deklaráljunk minden változót egész típusúnak a 10-es sorban:

```
10 defint a-z                        - így semmi sem marad ki!
```

Most már csak 21 másodperc kellett, hogy megjelenjen a 'Ready'.

2.1.2 Felhasználói függvények

Új kulcsszó: **DEF FN**

A CPC BASIC gazdag belső függvénytárát tetszésünk szerint bővíthetjük saját függvények definiálásával: aritmetikai és karakterláncot kezelő függvényeket alkothatunk, amelyekre majd az **'FN'** segítségével hivatkozhatunk. Egyedi függvények alkalmazásával lényegében szubrutinokat helyettesítünk, programunk áttekinthetőbbé válik. A függvénydefiníció szintaxisa:

DEF FN függvénynév (formális paraméterek) = kifejezés

A 'függvénynév'-re vonatkozó szabályok megegyeznek a BASIC változónevekre vonatkozó szabályokkal: maximum 40 karakter hosszúságú függvényneveket használhatunk. A 'formális paraméterek' listája zárójelben tartalmazza azokat a paramétereket, amelyek

értékét a függvény kiszámításához át kell adni az egyenlőségjel jobb oldalán álló 'kifejezésnek'. A 'kifejezés' a kiszámítás módját irja le.

```
1 rem PELDA FUGGVENYDEFINICIORA
10 def fn atlag (x,y) = (x+y)/2      - 'atlag' nevű függvény
20 input "Kerek ket szamot ", a,b      két szám átlagát számolja
30 print "Az atlaguk: ";              ki a 'kifejezés'-ben meg-
40 print fn atlag (a,b)               adott minta alapján
50 rem folytatjuk!
```

Függvények definiálása csak programban lehetséges, parancsmódban történő próbálkozásra 'Invalid direct command' (érvénytelen közvetlen parancs) hibajelzést kapunk. A 'formális paraméterek' listájában valamint a 'kifejezés'-ben szereplő változóknak semmi közük a program hasonló nevű változóihoz, sőt a függvény neve is használható eltérő változónévként. Bővítse ki a fenti programot a következő sorokkal:

```
11 x=1: y=2: atlag=3                - ezek önálló változók
50 print x, y, atlag                - értékük megmaradt
```

A 'formális paraméterek' el is maradhatnak, ha a függvény kiszámításához nincs szükség paraméterekre. A CPC belső órája a gép bekapcsolásától kezdve méri az eltelt időt, lekérdezése a 'PRINT TIME' utasitással lehetséges. A kapott válasz azonban háromszázad másodpercekben adja meg az eltelt időt, amit minden alkalommal emberközelibb időegységekre kell átszámitanunk. Ezt az átszámitgatást megteszi helyettünk egy-két függvény is:

```
10 def fn idomp = int(time/300)      - eltelt másodpercek
20 def fn idop  = int(fn idopm/60)   - eltelt percek száma
30 def fn idoo  = int(fn idop/60)    - eltelt órák száma
```

A program lefutása után például 'PRINT FN idoo' paranccsal lekérdezhetjük a gép bekapcsolása óta eltelt órák számát. Kis programunk egyébként azt is hivatott illusztrálni, hogy egy függvény definiciója már korábban definiált függvényeket is tartalmazhat.


```
1 rem PELDA SZOVEGES FUGGVENY DEFINICIORA
10 def fn kozepre$(szoveg$,szeles) =string$(szeles/2-len(szoveg$
)/2," ") + szoveg$
20  input "Mit irjak ki kozepre "; a$
30  print fn kozepre$(a$, 40)
```

A függvény a 'szoveg$'-ban megadott szöveget baloldalt kipárnázza szóközökkel úgy, hogy ha azt kiiratjuk a 'szeles'-sel megadott oszlopnyi szélességű képernyőre, a szöveg pontosan a sor közepére kerül.

```
1 rem A HET NAPJAINAK NEVE SORSZAMUK ALAPJAN
10 def fn nap$(melyik) =mid$("hetfo   kedd    szerda  csutorto
kpentek  szombat  vasarnap ", (melyik-1)* 9 + 1, 9)
20 input "Hanyadik nap "; h
30 nap$= fn nap$(h)
40 print "A"; h; ". nap: "; nap$
```

A függvény megadja a nap nevét, a nap sorszáma ('melyik') alapján. A 10-es sorban ügyelni kell a napneveket elválasztó közök számára: minden napnév a következő szóközökkel együtt 9 karakter hosszúságú legyen.

2.1.3 Decimális számok formattált kiiratása

Új kulcsszó: **PRINT USING**

Számok kiiratásakor gyakran szükséges felülbirálni a BASIC számmegjelenitési szokásait. Ha számoszlopokat egymás alá kívánunk irni, lehetőleg egyforma szélességgel, nem hagyatkozhatunk a BASIC automatikus számábrázolására. A 'PRINT USING' utasitás segitségével tetszőleges formátumban irhatunk ki adatokat. Az adatok lehetnek szövegek is, de leggyakrabban számokkal kapcsolatban használjuk a formattált kiiratást, amelynek általános szintaxisa a következő:

PRINT USING formátum ; adat-lista

A 'formátum'-ot szöveges változóként és konstansként egyaránt megadhatjuk, az alábbi két parancs ugyanazt a képet ered-

ményezi:

```
print using "###.#"; 12.78
f$="###.#": print using f$; 12.78
```

Numerikus értékek megjelenitésekor a 'formátum'-ban a következő megjelenitést vezérlő karaktereket használhatjuk:

'#' A kettőskereszt számjegypoziciót képvisel. A kiiratandó szám a számjegypoziciókon jobbra igazodva jelenik meg, a baloldalt esetleg fel nem használt pozíciókat szóköz tölti ki. Amennyiben a megjelenitésre kerülő szám nem férne el a 'formátum' által kijelölt területen, formattálás nélkül jelenik meg a teljes szám, erre a szám elé kiirt '%' jel hivja fel a figyelmet. A tizedesponttól jobbra (az utolsó jegy után) 0-k jelennek meg. Ha a szám több értékes tizedes jegyet tartalmaz, mint a 'formátum', a gép szabályosan kerekit. Ha az első jegy és a tizedespont közé valahova egy vesszőt teszünk, akkor a kiiráskor a tizedespont előtt minden harmadik számjegy elé egy vessző kerül.

Az alábbi programmal tetszőleges 'formátum'-okat próbálhatunk ki:

```
10 input "formatum"; f$                  - igy kell kiirni
20 if f$="" then end                     - ki is lehet szállni
30 input "szam"; szam                    - ezt kell kiirni
40 print using f$; szam                  - lássuk csak!
50 print                                 - egy üres sor
60 goto 10                               - és újra
```

Indítsuk el a programot egy próbafutamra:

```
run
formatum? ##.##
szam? 12.34
12.34                                    - nem hagyott szóközt a
                                           szám előtt

formatum? ###.##
szam? 12.34
 12.34                                   - van egy üres pozíció,
                                           az szóköz lesz

formatum? ##.##
szam? 123.45
%123.45                                  - nem fért el a 'formátum'-ban

formatum? ##.##
szam? 12.345
12.35                                    - felkerekitette, hogy
                                           elférjen

formatum? ##.##
szam? 12.3
12.30                                    - két tizedes kell

formatum? "##,###.##"                    - idézőjelben kell beadni
szam? 12345.67                             mert vessző van benne!
12,345.67
```

Az utolsó példa vesszője az angol-amerikai használatnak megfelelően három számjegyenként (ezresekként) teszi olvashatóbbá a nagy számokat. A 'formátum'-ban felhasználható további vezérlő karakterek:

'**' A 'formátum' elején álló két csillag azt eredményezi, hogy az üres helyek csillagokkal töltődnek fel. A két csillag beleszámit a formátummezőbe: két számjegyet képvisel.

'$$' Ha a formátummező elejére két dollárjelet irunk, a gép a legnagyobb helyiértékű szám elé egy '$'-t ir ki. A '$'-jel szintén beleszámit a formátummezőbe.

'**$' Kombinálja a '**' és '$$' hatását. A szám előtt minden üres pozicióba csillag, a legnagyobb helyiértékű számjegy elé '$' kerül.

'+' Ha a 'formátum' elejére (vagy végére) '+'-jelet irunk, a pozitiv számok elé (vagy után) '+' (pluszjel), negativ számok elé (vagy után) '-' (minuszjel) kerül.

'-' Ha a 'formátum' végén '-' (minuszjel) áll, a számitógép

pozitív számok után szóköz karaktert, negatív számok után '-'(mínuszjelet) ír ki.

```
formatum? ****.##                    - négy számjegy a tizedes
szam? 12.3                             előtt!
**12.30

formatum? $$##.##                    - itt is négy számjegynek
szam? 12.341                           van hely
 $12.34

formatum? **$###.##                  - kombináljuk az előzőket
szam? 1234.56
*$1234.56

formatum? +##.##                     - az előjelet rakja ki!
szam? 12.34
+12.34

formatum? ##.##-                     - előjel a szám után
szam? -12.34
12.34-
```

'↑↑↑↑' A 'formátum' végén álló négy '↑' (felnyíl) jel exponenciális kiírást eredményez. A megjelenítendő szám mantisszája a '↑↑↑↑' előtt álló 'formátum'-nak megfelelően fog megjelenni, a kitevő pedig 'E±xx' alakban. Szükség esetén a 'formátum'-ban megadott tizedespontnak megfelelően változik a kitevő értéke.

```
formatum? ##.##↑↑↑↑                  - exponenciálisan
szam? 1234.56                          egy ilyen kis számot?
 1.23E+03                            - hát érdemes volt?

formatum? ##.##↑↑↑↑-                 - kombinálni is lehet
szam? -123456789
12.35E+07-
```

Szövegek kiíratásakor a 'formátum'-ban a következő vezérlő karaktereket akalmazhatjuk:

'!' A felkiáltójel hatására az adatlista szöveges típusú adatainak csak az első karaktere jelenik meg.

```
print using "!"; "elso","betuk"
eb                                   - csak az első betűk!
Ready
```

'\n db. szóköz\' A fordított perjelek között elhelyezett szóközök számánál kettővel több (n+2) karaktert jelenít meg minden szöveges típusú adatból:

```
print using "\ \"; "csak","harom","betu"      - egy szóköz =
csaharbet                                       az első 3
Ready                                           karakter
```

A 'formátum'-ban használt minden egyéb karaktert a BASIC interpreter karakteres konstansnak tekint és változtatás nélkül megjelenít. Tetszőleges 'formátum'-okat kombinálhatunk így össze szövegekkel:

```
f$="ar: ###.##  db: ###   osszeg: ####.##"    - egy formátum-
Ready                                           ban 3 adat és
print using f$; 12.5, 123, 12.5*123             sok szöveg!
ar:  12.5  db: 123   osszeg: 1537.50
Ready
```

2.1.4 Numerikus függvények és konstansok

Új kulcsszavak: **MIN**
MAX
MOD
****
ROUND
PI

Aritmetikai feladatok könnyebb megoldását segítik elő a CPC alábbi numerikus függvényei:

MIN (adatlista) - az 'adatlista' vesszővel elválasztott elemei közül a legkisebbet adja eredményül.

MAX (adatlista) - az 'adatlista' vesszővel elválasztott elemei közül a legnagyobbat adja eredményül.

```
10 input "Adj be 5 számot! ", a,b,c,d,e
20 print "A legkisebb:"; min(a,b,c,d,e)
30 print "A legnagyobb:"; max(a,b,c,d,e)
```

```
x MOD y          - az 'x/y' osztás maradékát adja eredményül
x \ y            - az 'x/y' osztás eredményének egész részét adja
                   értékül
```

```
10 input "Adj be ket szamot! ", a,b
20 print a; "-ban"; b; "megvan"; a\b; "-szer"
30 print "maradek:"; a mod b
```

Számok kerekítését egyszerűen elvégezhetjük a **'ROUND'** függvény segítségével. Első argumentuma a kerekítendő szám, második argumentuma a tizedespont utáni számjegyek száma. A második argumentum hiánya esetén egészre kerekít. Ha a második argumentum negatív szám, a függvény tízesekre, százasokra, ezresekre stb. kerekít:

```
1 rem SZAMOK KEREKITESE
10 szam=1234.567
20 for jegy=3 to -3 step -1
30 print szam; jegy; "tizedessel:";
40 print round(szam, jegy)
50 next jegy
```

A CPC BASIC-je felismeri a **PI** állandót. Értékét 3.14159265-nek veszi, ami a kör kerülete és átmérője közötti aránynak felel meg.

```
10 input "A kor sugara "; sugar
20 print "A kor terulete:"; sugar * sugar * pi
30 print "A kor kerulete:"; 2 * sugar * pi
```

2.1.5 Bináris és hexadecimális számok

Új jelölések és kulcsszavak: **&H** vagy **&**

&X
HEX$
BIN$
DEC$ ♦ nincs CPC 464-en

A bináris és hexadecimális számok használata nemcsak a gépi kódú" programok írásakor lehetséges és célszerű. BASIC-ben is találunk sok olyan feladatot, amely a bináris vagy hexadecimális számokkal egyszerűbben oldható meg, mint a megszokott tízes számrendszerben. Bár a gép belső élete, csakúgy mint a gépi kódú programozás a Harmadik szint témája, már most szükséges megemlíteni a bináris és hexadecimális számok jelölésével és átváltásával kapcsolatos konvenciókat, mert egyes programokban ezen a szinten is használni fogjuk őket.

A hexadecimális konstans jelölése a szám előtt álló **'&H'** vagy csak **'&'** . Az utóbbi jelölést kapjuk vissza a programok listázásakor, még akkor is, ha '&H'-val vezettük be a hexadecimális számainkat. A hexadecimális számokkal minden szokásos és szokatlan művelet elvégezhető, az eredményt azonban decimálisan jeleníti meg a gép:

```
print &HC000           - a képernyő kezdőcíme, hexában egyszerű
-16384                 - így már kevésbe áttekinthető

print &HC000 + &800    - a második pixel"-sor kezdete
-14336                   = kezdőcím (&C000) + 2048 (&800)
```

A bináris számok jelölése **'&X'** . A bináris számokat szintén ugyanúgy kezelhetjük, mint a decimális vagy hexadecimális számokat, bitekkel dolgozó logikai műveletekben valamint karakterek tervezésénél használatuk azonban sokkal kényelmesebb.

```
print &X10000001   - a két szélső bit van beállítva: látszik is
 129               - ezen már kevésbé látszik

print &X10 or &X01 - 2 vagy 1 (az 'OR' bitenként egybeveti a két
 3                   számot és meghagyja az '1'-et: &H11)
```

Mivel a CPC megérti a bináris és hexadecimális számokat,

szükségünk lehet arra is, hogy bináris vagy hexadecimális alakban jelenítsen meg adatokat. A **'BIN$'** és **'HEX$'** függvények ezt egyszerű módon lehetővé is teszik. A megjelenítendő numerikus kifejezés mindkét függvény kötelező argumentuma:

```
print bin$(2)          - 10          elég két számjegy
print bin$(128+1)      - 10000001    nyolc számjegy kell
print hex$(14)         - E           elég egy számjegy
print hex$(128+127)    - FF          két számjegy kell
print hex$(31899)      - 7C9B        ehhez 4 számjegy kell
```

A példákból is látható, hogy a függvények bináris vagy hexadecimális számjegyekből álló nem formattált szöveget adnak eredményül. A szövegek formátumát a második argumentumként opcionálisan megadható minimális karakterszám határozza meg:

```
print bin$(2, 8)       - 00000010    nyolc karakter széles
print bin$(129, 8)     - 10000001    ez különben is annyi
print hex$(255, 4)     - 00FF        4 számjegyet kértünk,
print hex$(31899, 4)   - 7C9B        hogy passzoljon ehhez
```

Természetesen decimális szám helyet megadhatunk bináris és hexadecimális konstansokat is:

```
print bin$(&H81,16)      - 0000000010000001
print hex$(&X10000001)   - 81
```

A CPC 664 és 6128 képes tízes számrendszerbeli számjegyekből álló formattált szöveget is létrehozni a **'DEC$'** függvény segítségével. A második operandusként megadott formátumminta teljesen megegyezik a 'PRINT USING'-nál leírt 'formátum'-mal.

```
print dec$(10^6, "########,.##")      - 1,000,000.00
```

2.1.6 Szövegkezelő függvények

Új kulcsszavak: **INSTR**
LOWER$
UPPER$

A CPC sokoldalú szövegkezelő függvényei szubrutinok tucatjait teszik feleslegessé. Az **'INSTR'** függvénnyel hosszabb szövegekben kereshetjük kisebb szövegrészek vagy egy bizonyos karakter előfordulási helyét. Az eredményül kapott numerikus érték 0, ha a keresett szöveget nem találta meg, vagy a keresett szöveg kezdetének pozíciója a hosszabb szövegben. A függvény általános szintaxisa:

INSTR (indulo.pos, szoveg1$, szoveg2$)

A 'szoveg1$'-ban keressük a 'szoveg2$' első előfordulási helyét az 'indulo.pos' karaktertől kezdve. Ha az 'indulo.pos'-t nem adjuk meg, a keresés a 'szoveg1$' első karakterétől indul.

```
10 a$="abcdefghij abcde"
20 print instr(a$, "bc")          - 2 , az elejétől kereste
30 print instr(3, a$, "bc")       - 13, a 3. karaktertől kereste
40 print instr(14, a$, "bc")      - 0 , a 14. után már nincs!
50 print instr(a$, "qw")          - 0 , ez sehol sincs
```

A szöveges típusú változókban előforduló angol nagybetűket kisbetűre cseréli a **'LOWER$'** függvény, a kisbetűkből pedig nagybetűt csinálhatunk az **'UPPER$'** függvény segítségével.

```
print upper$("csupa kisbetu")
print lower$("CSUPA NAGYBETU")
```

A két függvény felhasználása adatok abécé sorrendbe történő rendezésénél gyakori, amikor is a "Kutya" a "cica" elé kerülne, hiszen a nagybetűk ASCII kódszáma kisebb mint a kisbetűké. Ha a 'LOWER$' vagy az 'UPPER$' függvénnyel minden adatot nagybetűsre vagy kisbetűsre konvertálunk, ilyen hibától nem kell tartanunk. A billentyűzetről történő beolvasáskor sem lehet biztosan tudni, hogy a felhasználó kisbetűs "igen"-t vagy nagybetűs "IGEN"-t, esetleg csak "i"-t vagy 'I'-t ad válaszul a feltett kérdésre.

```
100 input "Ujra (igen/nem) "; valasz$
110 if upper$(left$(valasz$,1)) = "I" then goto ...  - igen
120 if upper$(left$(valasz$,1)) = "N" then goto ...  - nem
130 goto 100 : rem  nem ertelmes valasz
```

2.1.7 Adatok bevitele és kiíratása

Új kulcsszavak: **LINE INPUT**
WRITE
ZONE
SPACE$
SPC

Az ismert 'INPUT' utasításnak szövegekkel kapcsolatban két hátránya is van: ha a szöveg vesszőt is tartalmaz, akkor idézőjelbe zárva kell beadni, idézőjelet tartalmazó szöveget pedig lehetetlen 'INPUT'-tal bevenni. Mindkét feladatot megoldja a kizárólag egy szöveges változót beolvasó **'LINE INPUT'** . Használata megegyezik az 'INPUT'-éval, de csak egy szöveges változónak kérhetünk vele értéket:

```
10 line input "Irj be barmit! "; barmi$          - nincs '?'  !
20 print barmi$
```

A 10-es sor 'LINE INPUT' szövege és a 'barmi$' változó között használhatjuk elválasztójelként a vesszőt is: kérdőjel akkor sem jelenik meg az "Irj be barmit! " után. A 'barmi$'-t komolyan kell érteni: tartalmazhat vesszőket és idézőjeleket is. Olyan szöveget, amelynek a tartalmát nem ismerjük, csak ezzel az utasítással érdemes beolvasni.

A vesszőket és idézőjeleket nemcsak beolvasni nehéz, hanem a kiíratásuk sem egyszerű. A **'WRITE'** utasítás úgy jeleníti meg a kívánt adatlistát, hogy a szövegeket idézőjelbe zárja, az egyes adatelemeket pedig vesszővel választja el egymástól.

```
10 a$="cica": a=123
20 write a$,"kutya",a,456     - "cica","kutya",123,456
```

A 'PRINT'-től eltérően a 'WRITE' utasításban az adatelemek közötti vesszők tehát nem a kiíratás távolságát szabályozzák, hanem egyszerűen megjelennek a képernyőn.

A 'PRINT' utasítás elválasztó vesszője alapállásban a képernyő 3, egyenként 13 karakter széles zónái közül a következőre ugratja a kurzort. A vesszővel elválasztott adatok tehát három oszlopba, zónába rendezetten jelennek meg. Gyakran több oszlopra van szükségünk. Ezt megvalósíthatjuk a 'TAB' esetleg a 'LOCATE' utasítással is, de a CPC lehetőséget ad arra is, hogy a **'ZONE'** utasítással meghatározzuk a kiíratási zónák, oszlopok szélességét.

```
1 rem PELDA ZONE-ra
10 cls
20 input "Milyen szeles legyen az oszlop "; sz
30 zone sz
40 esorban=int(40/sz)
50 for magas=1 to 10
60     for szeles= 1 to esorban
70         print int(rnd*99),
80     next szeles
90 next magas
100 print
110 input "Uj tablazathoz ENTER!",v$
120 goto 10
```

A program véletlen számokból (0 - 99) készít egy 10 sor 'magas' táblázatot. A táblázat oszlopainak száma a választott zóna szélességétől függ. Figyeljük meg, hogy maximálisan kétszámjegyű számainkhoz is legalább 5-ös zónaszélességet kell megadni, hogy az oszlopok függőlegesek legyenek.

A kiíratás formáttálását segíti a következő két függvény is: mindkettővel szóközöket iktathatunk be a kiíratandó adatlista elemei közé. Az **'SPC(n)'** elválasztó 'n' számú szóközt jelenít meg az általa elválasztott két adatelem között, így a következő elem kiíratási pozícióját az előzőhöz viszonyítva adhatjuk meg, ellentétben a 'TAB'-nál megadható abszolút pozícióval.

```
print "ot" spc(5) "szokoz"      - tényleg 5 szóköz van közöttük
print 12+1 spc(1) 10+10         - 3 szóköz, vajon miért?
```

A második példa 3 szóközét egyszerű megmagyarázni: egy szóköz zárja az első számot, egyet kértünk az 'SPC(1)'-gyel, a harmadik szóköz pedig a második szám pozitív előjele helyett áll. Számok kiiratásakor a 'PRINT USING'-ot használva hasonló meglepetések nem érhetnek.

Szóközökből álló karakterláncot a **'SPACE$(n)'** függvénnyel képezhetünk, a karakterlánc hosszúsága 'n' lesz. Ez a függvény tulajdonképpen megfelel a 'STRING$(n, 32)'-nek, ahol a '32' a szóköz karakter ASCII kódja,

```
10 abra$=string$(rnd*10+1,rnd*128+127)    - a grafikus karakterek
20 lyuk$=space$(rnd*10+1)                 - egy kis üres hely
30 ?abra$; lyuk$; : goto 10
```

2.1.8 Ciklusképzés másképpen

Új kulcsszavak: **WHILE**
WEND

A már megismert ciklusképző utasításokon kívül a CPC BASIC ismeri az 'amíg a feltétel igaz, addig csináld ezt ...' típusú utasítást is. A 'feltétel' megfogalmazása azonos az 'IF-THEN' utasításpárral kapcsolatban leírtakkal. A 'WHILE' utasítást követi a feltétel, amíg az 'igaz', egy 'WEND' újra visszaküldi a vezérlést 'WHILE'-t követő utasításra, ha már nem 'igaz', akkor a 'WEND'-et követő utasítással folytatódik a program:

```
WHILE  (amíg a feltétel igaz)
     ezt csináld
     ezt csináld
     ...
WEND   (vége a feltételes résznek)
     különben ide jön
```

```
10 while a<101                    - amíg 'a' kissebb mint 100:
20 print a; : a=a+1               - addig írd ki és növeld eggyel
30 wend                           - menj vissza a 'WHILE'-hoz!
40 print "Elerte a szazat!"       - a feltétel már nem igaz
```


A 'WHILE-WEND' szerkezetet általában könnyű helyettesíteni 'IF-THEN' utasításpárral, azonban a program áttekinthetősége, rövidsége inkább az előbbi mellett szól. Néhány példa az "üres" hurok használatára:

```
while inkey$="" : wend          - egy billentyű lenyomására vár
ido=time: while time<ido+300*x : wend   - 'x' másodpercig vár
```

A 'WHILE-WEND' hurkok egymásba skatulyázhatók, bár a 'WEND' után nem áll azonosító, a BASIC minden 'WEND'-nek megtalálja a 'WHILE'-ját. Ha a programozó jóvoltából ez nem sikerülne, 'Unexpected WEND' (váratlan WEND) hibajelzést ad. Ha egy meglévő 'WHILE'-hoz nem talál 'WEND'-et, 'WEND missing' (WEND hiányzik) hibajelzéssel áll le a program.

▶▶▶ Valós számokkal történő számolásnál a belső számábrázolás és a kerekítések miatt sohasem bízhatunk abban, hogy két, számunkra egyenlő számot a gép is egyenlőnek tekint! Próbálja ki az alábbi programot, hogy rejtélyes hibáktól kímélje meg magát a jövőben.

```
10 rem KEREKITESI HIBAK
20 a=0                          - nulláról indulunk
30 while a<>5                   - amíg 'a' nem egyenlő 5-tel
40 a= a+0.02                    - addig növeljük 0.02-vel
50 wend                         - de csak addig!
60 print "a>5"
```

Indítsa el a programot és várjon egy kis ideig. Ha már türelmetlen, mert lassúnak találja a CPC-t, állítsa le a programot és kérdezze le az 'a' értékét! Az pedig jóval több lesz, mint öt ... Mi is történt valójában? Rosszul szerveztük meg a ciklust? Esetleg átírhatja 'IF-THEN'-re, de ez sem segít. Adja be a következő sort, lássunk is valamit:

```
42 print a                      - mikor lesz egyenlő 5-tel?
```

A látvány elkeserítő: 0.64 után nem 0.66 jön, hanem egy ici-picivel több: 0.660000001. A különbség olyan csekély, hogy 1 után ismét helyre áll a rend, de 2.28 után nem a várt 2.30

következik, hanem 2.29999999, vagyis egy kicsit kevesebb és a csökkenés folytatódik: 5 helyett már csak 4.99999993 az eredmény. Valójában a különbség kevésbé igényes számításoknál még most is figyelmen kívül hagyható, azonban a feltételünk 'a<>5' akkor is 'IGAZ' marad, ha 'a' értéke meghaladja 5-öt. Javítsuk ki a 30-as sort:

```
30 while a<5                    - amíg 'a' kisebb 5-nél
```

Most már 5 után tényleg leáll a program, de a kiírás a régi marad. Ezen is segíthetünk, a gép kerekítését kerekíttessük újra a 'PRINT USING' segítségével:

```
42 print using "#.##"; a        - csak 2 tizedes kell, így növeljük
```

Most már a kiírás is elfogadható; azt is tudjuk, hogy miért fut túl a program az 5.00-án (az neki csak 4.99999993 volt). A tanulság: amikor csak lehet, használjunk egész számokat!

2.1.9 Hibakezelés BASIC-ben

Új kulcsszavak:

ERR
ERL
ON ERROR GOTO
RESUME
ERROR
TRON
TROFF

Hibákra mindig fel kell készülni! A legjobb program is leállhat hibajelzéssel, ha nem a programozó, akkor a felhasználó jóvoltából. Ha például egy 'INPUT' nem azt kapja, amit várunk, a BASIC egyből jelez. Néhány hibajelzéssel már találkoztunk, a CPC BASIC interpreter összes hibajelzése a leggyakoribb kiváltó okokkal együtt megtalálható a Függelékben. Gondos munkával készíthetünk olyan programot, amelyet még a legelvetemültebb felhasználó sem tud hibára csalni, de általában célszerűbb a CPC sokoldalú hibakezelő utasításait használni. Nézzünk egy példát:


```
100 rem PELDA HIBAKEZELESRE
110 cls
120 defint n                            - n-nel kezdődő: egész szám
200 rem *** egesz szamot ker be ***
210 input "Adj be egy pozitiv egesz szamot!", n
220 print "Elszamolok 1-tol"; n; "-ig!"
230 for i=1 to n: next i
240 print "Kesz!"
250 goto 210
260 rem folytatjuk!
```

A programot futtatva nem kapunk hibajelzést akkor sem, ha a beadott szám negatív (legfeljebb nem számol a CPC), és akkor sem, ha a beadott szám nem egész: a CPC a szabályosan egészre kerekített 'n'-nel számol. Próbáljunk meg azonban 32767-nél nagyobb számot beadni, például 40000-t!

```
Adj be egy pozitiv egesz szamot! 40000
Overflow in 210
Ready
```

'Overflow', vagyis túlcsordulás történt a 210-es sorban. A túlcsordulás itt azt jelenti, hogy a beadott szám nem fér el a változónak fenntartott helyen. Emlékezzünk vissza, hogy az egész számok megengedett tartománya -32768..32767 között van! A beadott 40000 pedig jócskán kilóg belőle, tehát jogos a hibajelzés. Mit lehet tenni, hogy a program ne álljon le hibával? Bővítsük ki a programot a következő sorokkal:

```
130 on error goto 1000                  - hiba esetén menj 1000-re!

1000 rem *** hibakezeles ***
1010 print "A szam tul nagy!"
1020 print "32767-nel kisebbet adj be!"
1030 resume                             - folytasd a hibás soron!
```

A 130-as sor arra utasítja a BASIC-et, hogy az 1000-es sorra adja a vezérlést, ha a program futása során bárhol hiba fordul elő: **'ON ERROR GOTO'** = hiba esetén menj ... A hibakezelő rutin (amely természetesen az 1000-es sorral kezdődik), kiírja

a megfelelő, általunk megszerkesztett hibaüzenetet és visszaadja a vezérlést arra az utasításra, ahol a hiba előfordult, erre szolgál a **'RESUME'** utasítás.

A fenti kis programunkban más hibalehetőség nincs is, így a hibakezelés ezen egyszerű módja kielégítő. Nagyobb programokban a lehetséges hibahelyek száma arányosan növekszik. Sokféle hibát nem orvosolhatunk egyazon módon! Szükségünk van a hiba helyére (melyik sorban fordult elő) és a hiba típusára (a BASIC hibajelzései a hiba típusára utalnak). Nézzük, hogyan oldható meg többféle hiba kezelése egy programon belül:

```
100 rem HIBAKEZELES 2
110 cls
200 rem *** tombfeltoltes ***
210 print "Adatbevetel": print
220 input "Adja be az adat sorszamat: ", n
230 print n; : line input ".  adat: "; a$(n)
240 a$(n)=a$(n) + " adatfelveteli proba"
250 print
260 goto 220
270 rem folytatjuk!
```

A program egy szöveges tömböt tölt fel a felhasználó által beadott adatokkal, mégpedig úgy, hogy minden szövege végére odabiggyeszti az ' adatfelveteli proba' karakterláncot. Próbálja ki a programot!

```
Adja be az adat sorszamat: 12                    - 12. adat
 12 .  adat: Na, milyen hiba lesz most?          - egy szöveg
Subscript out of range in 230                    - és egy hiba
```

Igen, az 'a$' tömböt nem dimenzionáltuk, tehát a CPC csak 11 elemre készül fel (0 - 10), így a 12. elemet már nincs hova tenni: 'Subscript out of range', vagyis a tömb indexe meghaladja a maximális értéket. Ez eddig egy hibaforrás, van még más is? Próbáljon meg beadni - persze most elfogadható index-szel - egy jó hosszú szöveget, mondjuk annyi "a" betűt, amennyit a CPC elfogad. Ez pontosan 255 darab lesz. Aki bead legalább 236 karaktert, megszolgálja a következő hibajelzést:


```
String too long in 240                           - újabb hiba
```

A 240-es sorban létrehozandó karakterlánc túl hosszú lesz. Amint a karakterláncokkal kapcsolatban megtudtunk, egy szöveges típusú változó maximum 255 karakter hosszú szöveget tud tárolni. Ha 'a$(n)' már legalább 236 karakter hosszú, nem adhatjuk hozzá a 20 karakter hosszúságú ' adatfelveteli proba'-t. Megismerve a lehetséges hibaforrásokat, készüljünk fel az elhárításukra! A hibák előfordulási helyét, vagyis a hibás sor sorszámát adja meg az 'ERL' függvény, a hibák típusáról egy másik függvény, az 'ERR' tájékoztat. Az, hogy a kettő közül melyiket használjuk, az adott programtól függ: egy több utasítást tartalmazó programsorral kapcsolatban nem leszünk sokkal okosabbak, ha megtudjuk, hogy hiba van benne, inkább a hiba típusára van szükségünk. Mint mindig, a rövidebb programsorok a hibakezeléshez is előnyösebbek. Kezeljük hibáinkat mind a kétféle módon:

```
10 on error goto 1000
1000 if erl<>230 then 1100                       - következő hiba ...
1010 print "Az adat sorszama tul nagy."         - 230-as sorban lévő
1020 print "Csak 0-10 kozott lehet!"              hibára reagálunk
1030 print "Ismetelje meg a beadast!"            és a 220-es sorra
1040 resume 220                                  küldjük vissza!

1100 if erl<>240 then 1200                       - következő hiba ...
1110 print "Az adat tul hosszu."                 - 240-es sorban lévő
1120 print "Max. 235 karaktert adhat be!"         hibára reagálunk
1130 print "Ismetelje meg a beadast!"            és a 230-as sorra
1140 resume 230                                  küldjük vissza!

1200 rem ismeretlen hibahely                     - nem az előző két
1210 print "Ismeretlen hiba tortent!"             hibahelyünkről jött
1220 print "Talan sikerul folytatni..."          ide a program...
1230 resume                                      - hátha sikerül...
```

Az 1000-es sorban megnéztük, hogy a hiba helye a 230-as sorban található-e, ha nem, akkor tovább küldjük a programot, ha igen, kiírjuk hibaüzenetünket a felhasználónak és a 220-as sorra adjuk a vezérlést: 'RESUME 220'. Hasonlóan járunk el a 1100-es sorral kezdődő hibakezelő rutinban is. Ha a hiba nem a

240-es sorban fordult elő, továbbküldjük a programot az 1200-as sorra. A legutolsó hibakezelő rutinunk nem tudhatja, hol fordult elő a hiba, ezért kiír egy semleges üzenetet és megkockáztatja a lehetetlent: 'RESUME', vagyis a hibát okozó utasításra küldi vissza a programot. Orvosuljuk a hibákat az 'ERR' függvény által szolgáltatott hibakódokat felhasználva, ehhez azonban fel kell lapozni a Függelékben található táblázatot! A programunkban csak két sort kell átírni:

```
1000 if err<>9 then 1100
1100 if err<>15 then 1200
```

Ha a hibakód nem 9 (Subscript out of range), akkor tovább megyünk a 1100-es sorra, ahol a 15-ös kódszámú hibát kezeljük (String too long). Ha a hibakód nem e kettő közül való, ismeretlen típusú hibával van dolgunk: 1200-es sor. A hibakezelésben gyakran kell kombinálni az 'ERL' és 'ERR' függvényeket, hiszen egy hosszabb programban több helyen is előfordulhatnak azonos típusú hibák. Egy másik megoldás: minden hibalehetőség elé egy saját 'ON ERROR GOTO'-t írni.

Minden 'ON ERROR GOTO' addig érvényes, amíg egy másikkal nem találkozik a program. Ha a hibakezelést vissza kívánjuk adni a BASIC interpreternek, az utasítás következő formáját használjuk: **'ON ERROR GOTO 0'** , vagyis hiba esetén állítsd le a programot és írj ki hibaüzenetet.

A 'RESUME' utasításban, mint láttuk, sorszám is megadható. Az utasításnak három formája van, mindegyik máshová küldi vissza a programot:

RESUME	- a hibát tartalmazó utasításra küldi vissza
RESUME sorszám	- a 'sorszám'-ú sorra küldi
RESUME NEXT	- a hiba helyét követő sorra küldi

A CPC hibakezelése lehetővé teszi, hogy szimuláljuk a várható hibákat, vagyis mesterségesen állítsunk elő hibákat, és segítségükkel tesztelhessük hibakezelő rutinjainkat. Hibát okozni az **'ERROR'** utasítással lehet, egy kötelező paramétere

van: a kívánt hiba hibakódja.

error [ENTER]	- adja be csak így, paraméter nélkül
Operand missing	- ez valódi hibajelzés, tényleg hiányzik az operandus (paraméter)
error 17	- milyen hiba a 17-es kódszámú?
Cannot CONTinue	- 'Nem tudom folytatni'
error 28	- és a 28-as?
Unknown command	- 'Ismeretlen parancs'

Az 'ERROR' utasítást természetesen nem közvetlenül, hanem programok kipróbálásakor célszerű alkalmazni. Ott kell elhelyezni, ahol egy bizonyos hiba előre várható és a hibakezelő rutin tesztelése után el kell távolítani.

A program tesztelése során megkönnyíti a hiba kiszűrését, a hibás programág megtalálását, ha nyomonkövetjük a program futását. A nyomonkövetés elárulja, hogy merre jár, mely sorokat hajtja éppen végre a program akkor is, amikor a képernyőn már hosszabb ideje semmi sem történik. A CPC BASIC nyomonkövetése a **'TRON'** paranccsal kapcsolható be és a **'TROFF'** paranccsal állítható le. Egy tetszés szerinti program indítása előtt adja be a 'TRON' parancsot és lásson csodát: a képernyőn [szögletes] zárójelekbe foglalva jelennek meg a végrehajtásra kerülő sorok sorszámai. Ez a képernyőfelépítést sajnos elrontja, de például egy végtelen hurkot azonnal felismerhetünk az ismétlődő sorszámokról.

TRON	- nyomonkövetést bekapcsol
TROFF	- nyomonkövetést kikapcsol

A két kulcsszó utasításként is használható: ha programsorban helyezzük el őket, akkor módunk van csak a gyanús programrészlet sorairól kérni sorszámlistát. A CPC ablakkezelése (a képernyőablakokról később lesz szó), lehetővé teszi azt is, hogy a sorszámlistát egy semleges területre tereljük, vagyis hogy az ne zavarja a képernyő többi részét.

2.2 A CPC KÉPERNYŐKEZELÉSE SZÖVEGES ÜZEMMÓDBAN

2.2.1 Képernyő-üzemmódok

Új BASIC utasítás: MODE

A CPC képernyője több mint 16 ezer byte*-ot használ fel a rajta levő információ tárolásához. Ilyen nagy tár felhasználása lehetővé teszi, hogy a képernyőn tetszőlegesen jelenítsünk meg, akár keverve is, karaktereket és grafikát. A megjelenítés vízszintes felbontása (hány grafikus pont különböztethető meg) fordítottan arányos a felhasználható színek számával.

A CPC háromféle üzemmódban tudja a képernyőt kezelni. Az üzemmódok közötti átváltásra a **'MODE'** utasítás szolgál. A gép bekapcsolás után automatikusan 'MODE 1'-et hajt végre.

A különböző képernyő-üzemmódok legfontosabb jellemzői:

	Karakter egy sorban	Színek száma	Vízszintes pontok száma
MODE 0	20	16	160
MODE 1	40	4	320
MODE 2	80	2	640

A függőleges felbontás minden üzemmódban azonos:
25 karaktersor = 200 grafikus pont felbontással.

Minden képernyő-üzemmódnak megvan a maga előnye: a nullás üzemmódban egyszerre 16 szín is lehet a képernyőn, játékokhoz ez ideális, "komolyabb" programokhoz kevésbé, hiszen csak 20 karakter fér el egy sorban. Az egyes üzemmódot használjuk leggyakrabban: 4 szín általában elég a különböző effektusokhoz, a 40 karakteres képernyősorban pedig már elég adat jeleníthető meg. A grafikus felbontás is itt a legarányosabb: 320 pont vízszintesen, 200 pont függőlegesen. A kettes üzemmód a gép professzionális felhasználhatóságának a kulcsa: 80 karakter egy sorban elegendő a legkomolyabb szövegszerkesztő* vagy táblázatszámító programokhoz is, amelyek színeket nemigen használnak.

Tekintsük meg a három képernyő-üzemmód hatását:

```
10 for m=0 to 2                                        - három MODE
20 mode m                                              - MODE 0,1,2
30 print "MODE"; m                                     - lássuk, melyik
40   for k=33 to 255                                   - hogyan néznek
50       print chr$(k);                                  ki a betük?
60   next k
70 print: input "Kovetkezo MODE-hoz ENTER! ", v$       - vár
80 next m
```

A 'MODE' utasítás a kívánt képernyőmód beállításán kívül törli is a képernyőt ('CLS'-t hajt végre), ezért a 20-as sor eltünteti a képernyő előző tartalmát.

Megtalálva a programíráshoz megfelelő képernyő-üzemmódot, a színes monitorok eddig boldog tulajdonosai meglepődve veszik tudomásul, hogy 'MODE 2'-ben a mélykék alapon megjelenő sárga betük elég nehezen olvashatók. Az egyszínű zöld monitoron a sötét alapon világos betük 'MODE 2'-ben is stabilan élesek. Megfelelő színeket összepárosítva a színes monitoron is könnyen olvasható képet kapunk 80-karakteres üzemmódban is.

2.2.2 Színek: tinták, tollak, papírok, keretek

Új BASIC utasítások: **PEN**
PAPER
BORDER
INK
SPEED INK

A képernyőn megjelenő színek száma fordított arányban áll a vízszintes felbontással, de akárhány szín is legyen egyidőben a képernyőn, azt a CPC 27 színű palettájáról BASIC parancsokkal kiválaszthatjuk. A palettáról kiválasztott színből tinta ('INK') lesz, amibe tollakat ('PEN') mártogatunk és papírra ('PAPER') írunk vagy rajzolunk velük. A papírunkat keret ('BORDER') veszi körül, színe független a tinták színétől: a paletta 27 színe közül választhatjuk ki. A CPC palettáján a színeknek sorszámuk van (0 - 26), amely sorrend a színek világosságát is tükrözi:

a 0-ás szín a legsötétebb (fekete), a 26-os szín a legvilágosabb (vakító fehér). A palettán fellelhető színeket a sorszámukkal együtt a Függelék tartalmazza.

Bekapcsolás után a keret színe egybeesik a papír színével, így nem is látható, meddig tart a papírunk. Egy **'BORDER'** parancs a megfelelő paraméterrel átváltja a keret színét. A paraméter a paletta színeinek megfelelően 0 és 26 között lehet, bár a CPC 31-ig nem ad hibajelzést (de új színt se: a régieket ismétli).

```
border 26          - a vakitóan fehér keret elválik a papírtól
border 1           - visszaállítja az előző kék keretet
```

A mindenkori 'PRINT' utasítás az előzőleg meghatározott tollal ír ki. Bekapcsolás után az egyes toll aktív ('PEN 1'). Tollaink száma az aktuális képernyő-üzemmódtól függ:

```
MODE 0 -ban   16 tollat használhatunk:  PEN 0 ... PEN 15
MODE 1 -ben    4 tollat használhatunk:  PEN 0 ... PEN 3
MODE 2 -ben    2 tollat használhatunk:  PEN 0 ... PEN 1
```

A gép nem jelez hibát akkor sem, ha az üzemmódnak megfelelő tollsorszámnál nagyobbat választunk (max. 15-ig), de nem is ad újabb színeket:

```
1 rem TOLLPROBA
10 for m= 0 to 2                          - minden MODE-ban
20 mode m                                 - MODE 0 - 1
30   for toll= 0 to 15                    - minden tollal
40       pen toll                         - PEN 0 - 15
50       print "PEN"; toll; "(toll)"      - kiirunk valamit
60   next toll
70   print: input "tovabb = ENTER ", v$
80 next m
```

Amint a lefuttatott programból megtudhatjuk, 'PEN 0 (toll)' felirat sehol sem jelenik meg, mivel a nullás toll színe egybeesik a papír színével: kékre kékkel írtunk. 'MODE 0'-ban mind a 15 toll más színű, a 14-es és a 15-ös villognak is. 'MODE 1'-ben csak 3 színben látjuk a feliratot, mivel az 5-ös toll ugyanolyan

színű, mint az 1-es stb. 'MODE 2'-ben minden második felirat esik egybe a papír színével, csak a páratlan számú tollakkal írt feliratok látszanak: 'PEN 0' a papír színe, 'PEN 1' az író szín, 'PEN 2' újra a papír színe stb.

Térjünk vissza a 'MODE 1'-hez. Kék papíron sárgán ír az egyes sorszámú tollunk. Tollcserével átválthatunk világos kékre (PEN 2) és pirosra (PEN 3), de hogyan érhetjük el a többi színt? Természetesen tintacserével. Tollaink sorszámának semmi köze a palettán levő színek sorszámához. A tinták ('INK') a közvetítők.

Minden MODE-ban annyi tintatartót használhatunk, ahány tollunk van, 'MODE 1'-ben tehát négyet. A tintatartók feltöltése, a tinták meghatározása, az **'INK'** utasítással történik:

INK tollsorszám, palettaszín

Az 1-es tollunk eddig sárga volt, mert a CPC bekapcsoláskor az 1-es tinta(tartó)hoz a 24-es színt rendelte hozzá. Öntsünk bele zöld színt: 'INK 1,18'. Ha ezután az egyes tollat választjuk ('PEN 1'), az már zölden ír. A papírunk színe kék, mert a 0-ás tintához a paletta 1-es színét társította a CPC: 'INK 0,1'. Legyen inkább piros, jól mutat rajta a zöld írás: 'INK 0,6'. Rikítóbb színkombinációt nehéz elképzelni, bár egy kis próbálkozással biztos találnánk, hiszen 27 szín közül lehet választani!

A papír színét a nullás tinta adta eddig. Ez nem szükségszerű, a **'PAPER'** utasítással a rendelkezésre álló tintákból bármelyiket kiönthetjük a képernyőre:

PAPER tintasorszám

A 'PAPER' utasítás végrehajtása után a megjelenő karakterek alatti új papír már a kívánt színű lesz. Hogy az egész képernyő átvegye az új papírszínt, egy 'CLS' utasítás szükséges.

```
10 for p= 0 to 3                          - a négy papírszín
20   paper p: cls                         - eláraszta a képet
30   pen p+1: print string$(40,"*")       - ne a papír színével
```

```
40       for var= 1 to 500: next var        -irjunk!
50 next p
```

A gép bekapcsolásakor a különböző képernyő-üzemmódokban a lehetséges tintákhoz hozzárendelt palettaszínek a következők:

Sorszám	MODE 0	MODE 1	MODE 2
0	1	1	1
1	24	24	24
2	20	20	1
3	6	6	24
4	26	1	1
5	0	24	24
6	2	20	1
7	8	6	24
8	10	1	1
9	12	24	24
10	14	20	1
11	16	6	24
12	18	1	1
13	22	24	24
14	1/24	20	1
15	16/11	6	24

'MODE 0'-ban egyidőben használható 16 INK közül az utolsó kettőhöz egyszerre két palettaszín is tartozik, ez a két szín váltakozását, villogását eredményezi: 'MODE 0: PEN 14'. Hasonló effektus a másik két üzemmódban is elérhető, ha egy tintatartóba két színt töltünk. Ugyanígy villogtatható a keret színe is:

INK tollsorszám, palettaszín1, palettaszín2
BORDER palettaszín1, palettaszín2

A villogás sebességét a **'SPEED INK'** utasítással állíthatjuk be. Két paramétert kell megadni: mindegyik szín időtartamának 0.02 (egy ötvened) másodpercben kifejezett hosszát. Az alábbi programmal kék-sárga villogó keretben zöld-piros villogó papíron fekete-fehér villogó feliratot állítunk elő. Az első színt egy másodperc (50 * 0.02 mp) múlva váltja fel a második, amely fél másodperc (25 * 0.02 mp) után vált vissza újra az első színre:


```
1 rem VILLOGO SZINEK
10 mode 0                                  - a széles betűk miatt
20 border 1,24                             - a keret színei
30 ink 0,18,6                              - nullás tinta színei
40 ink 1,0,26                              - egyes tinta színei
50 paper 0: pen 1                          - papír és toll választás
60 speed ink 50,25                         - az első és második szín
                                             időtartama
70 locate 6,12: print "TOKELETES"
80 goto 80                                 - gyönyörködjünk benne!
```

2.2.3 Ablakok a képernyőn

Új kulcsszavak és jelölések: **WINDOW**
#
POS
VPOS
WINDOW SWAP

A CPC irodalma az adatok be- és kivitelére szolgáló eszközöket angol szóhasználattal 'stream'-nek nevezi, ez magyarra 'áramlat'-nak fordítható, de könyvünkben a szokásos **'csatorna'** szót használjuk ezen eszközök megnevezésére. A CPC BASIC-je 10 csatornát kezel, ezek közül az első nyolc (#0-#7) a képernyőre irányul, a kilencedik (#8) a nyomtató, a tizedik (#9) pedig a külső tár (a kazettán vagy floppy-lemezen levő adatfile*). Minden csatornának azonosító száma van, amelyet a '#' (kettőskereszt) vezet be. Az első nyolc csatornát, amelyek a képernyőre irányulnak, ablaknak (window) nevezzük.

Az ablak a képernyő teszőleges nagyságú része, amelynek helyét és nagyságát előzőleg a 'WINDOW' utasítással meghatároztunk. A gép bekapcsolásakor a nullás ablakot (#0) választja és az összes ablak paramétereit egyformára állítja: minden ablak az egész képernyőre kiterjed.

```
print #0, "0.ablak"            - rendesen kiírja, mint #0 nélkül
Ready
print #1, "1.ablak"            - a bal felső sarokba írta!
Ready
```

Az első paranccsal a nullás ablakba írtunk, az eredmény ugyanaz, mintha a '#0' csatornaszám nélkül írtunk volna. Minden parancs és utasítás, ha nem adunk meg csatornaszámot, a nullás ablakba irányul. A nullás ablak pedig a jól ismert teljes képernyő. Ezért például a 'LIST #0' sem csinál mást, mint a 'LIST'. Amikor második parancsunkkal az egyes ablakba írattunk, az a képernyő bal felső sarkában jelent meg. Mint már említettünk, a gép bekapcsolásakor minden ablak a teljes képernyőre terjed ki. Az egyes ablakba eddig még nem írtunk, tehát az egyes ablak saját kurzorja eddig az 1,1 pozíción állt, az ablakba irányuló első szöveg is ott jelent meg.

Minden ablaknak megvannak a saját paraméterei, ezeket a következő utasításokkal adhatjuk meg (ha az ablak koordinátáin kívül nem adunk meg más paramétert, azokat a CPC a bekapcsolás után beállított értékekkel kezeli):

WINDOW #n, x.bal, x.jobb, y.fent, y.lent

Az 'n' sorszámú (0-7) ablak helyét jelöli ki a képernyőn, a vízszintes koordináták az aktuális 'MODE'-tól függenek. A koordinátákkal a kívánt képernyőpozíciót kell megadni: 'WINDOW #3,1,10,1,10' a 3-as számú ablakot a képernyő bal felső sarkában helyezi el, vízszintesen és függőlegesen egyaránt az 1-es karakterpozíciótól a 10-esig tart, azokat is beleértve.

PAPER #n,tinta

Az 'n' sorszámú ablak papírszínéhez a 'tinta' sorszámú tintaszínt rendeli hozzá.

PEN #n,tinta

Az 'n' sorszámú ablakba legközelebb író tollat választja ki.

Minden eddig megismert, a képernyőre irányuló utasítást egy ablakba irányíthatunk, ha az utasítás egyéb paraméterei előtt megadjuk az ablak sorszámát **'#n,'** is. Néhány példa:

PRINT #n, adatlista (hasonlóan: WRITE #n,)

Csak az 'n' sorszámú ablakot használja a kiíráshoz, a képernyő többi részét szabadon hagyja. Szükség szerint felfelé görgeti a gyorsan betelő ablak tartalmát.

INPUT #n,"szöveg"; változó (hasonlóan: LINE INPUT #n,)

A "szöveg"-et és a kérdőjelet az 'n' sorszámú ablakban jeleníti meg, oda várja a választ is.

CLS #n

Az 'n' sorszámú ablakot törli az ablak papírszínére.

LOCATE #n,x,y

Egy ablak koordinátái annak bal felső sarkából indulnak és az ablak nagysága határozza meg a maximális értékeket. A CPC minden ablakra vonatkozóan nyilvántartja az aktuális kurzorpozíciót.

POS (#n)

Az 'n' ablak kurzorjának vízszintes pozícióját eredményező függvény. A zárójelben akkor is kötelező megadni az ablak sorszámát, ha a nullás (#0) ablakról van szó.

VPOS (#n)

Az 'n' ablak kurzorjának függőleges (vertikális) pozícióját eredményező függvény. Használata a 'POS'-hoz hasonló:

```
10 locate 20,20 :print "A";      - a 0-ás ablakba irunk
20 print pos(#0); vpos(#0)       - hol áll most a kurzor?
```

Mire is használhatók valójában az ablakok? Mindenek előtt adatok tagolt megjelenítésére. A rendszer gondoskodik róla, hogy az adataink csakis a kijelölt területen jelenjenek meg, arról ki nem lóghatnak, más területet véletlenül sem írhatnak felül.

Ezenkívül többféle, nem kevésbé fontos felhasználási lehetőséget találhatunk az ablakok számára. Egy ablak megnyitásával, papírszínének megadásával és letörlésével a lehető leggyorsabb módon festhetünk be egy területet. A következő program egy magyar zászlót varázsol azonnal a képernyőre:

```
1 rem MAGYAR ZASZLO
10 mode 1                        - elég lesz négy szín
20 ink 1,6                       - egyes szín a piros
30 ink 2,26                      - kettes szín a világos fehér
40 ink 3,9                       - hármas szín a zöld
50 window #1,1,40,1,8            - az első nyolc sor
60 window #2,1,40,9,16           - a második nyolc sor
70 window #3,1,40,17,24          - a harmadik nyolc sor
80 paper #1,1                    - fent piros papír
90 paper #2,2                    - középen fehér papír
100 paper #3,3                   - lent zöld papír
110 cls #1: cls #2: cls #3       - papírszínre törölni mindhármat
120 while inkey$="": wend        - ne zavarjon a 'Ready'
```

A tinták kiválasztása után (20-40-es sor), meghatároztuk az ablakok méreteit. Mindhárom ablakunk a képernyő bal szélétől a jobb széléig tart (1,40), a különbség a függőleges koordinátákban van: ahol az egyik ablak végződik, ott kezdődik a következő. Az ablakok papírszínének kiválasztása (80-100-as sor) nem okoz látható változást, amíg nem töröljük az ablakokat, ezt a 110-es sor végzi el.

Ablakokkal létrehozható effektust illusztrál a következő program is. Mindössze egy ablakot nyitunk meg újra és újra, de a paramétereit véletlenszerűen állítjuk be. 'MODE 0' garantálja a lehető legtarkább képernyőt.

```
1 rem SOK SZINES ABLAK
10 mode 0                        - minél több szín legyen
20 x= int(rnd*11)+1              - bal széle: 1-től 11-ig lehet
30 y= int(rnd*14)+1              - teteje: 1-től 14-ig lehet
40 xj=int(rnd*19)+x              - jobb széle = bal széle + véletlen
50 yj=int(rnd*12)+y              - alja = teteje + véletlen
60 window #1,x,xj,y,yj           - megnyitjuk egyes ablakként
```



```
70 alap= int(rnd*16)             - véletlen szín a papírnak
80 paper #1,alap : cls #1        - ablakot papírszínre törölni
200 goto 10                      - és így tovább
210 rem folytatjuk!
```

Egy ablak újradefiniálása semmi hatással nincs a képernyőn láthatókra. Az ablakba történő írással és az ablak törlésével azonban megsemmisítjük az alatta levő információt. A CPC BASIC sajnos nem rendelkezik olyan utasítással, amellyel egy ablak alatti képernyőrészletet ideiglenesen elraktározhatnánk. A Harmadik szint egyik gépi kódú programja ezt a hiányt próbálja pótolni. Most azonban írjuk tele ablakainkat olyan információval, amiért nem kár. Bővítsük az előző programot:

```
90 iroszin= int(rnd*16)          - véletlen szín a toll számára
100 if iroszin=alap then 90      - ne legyen azonos a papíréval!
110 pen #1,iroszin               - ezzel írunk majd, de mit?
120 betu= int(rnd*222)+33        - bármit, de nem szóközt
130 darab= (xj-x+1)*(yj-y+1)     - ennyi betű fér el az ablakban
140 for t=1 to darab             - egy hurokkal teleírjuk
150    print #1,chr$(betu);      - a véletlen karakterünkkel
160 next t                       - az egész ablakot
```

Az ablakokban nemcsak betűk jelennek meg, hanem különböző grafikus ábrák is, ezekről nemsokára szó lesz. De először az ablakok még egy hasznos felhasználási módját vizsgáljuk meg. Eddig nem bántottuk a nullás ablakot, hiszen a BASIC ott jeleníti meg a 'Ready' feliratot és az esetleges hibaüzenetet is. Adja be a következő parancsot, ami az események természetes folyamát a képernyő alsó öt sorába tereli:

```
window #0, 1, 40, 20, 25         - ugyanaz mint 'window 1,40,20,25'
```

A rendszer minden üzenete ezentúl csak az alsó öt sorban jelenik meg, a képernyő többi része változatlan marad. Listázni természetesen elég kényelmetlen ezen a kis területen és a kurzor sem képes kitörni belőle. Azért, hogy visszaálljon a régi rend, nem szükséges újradefiniálni a 0-ás ablakot, egy 'MODE' parancs a teljes képernyőre állítja az összes ablakot.

Ablakok kicserélését teszi lehetővé a **'WINDOW SWAP'** utasítás. A paraméterként megadott két ablak természetesen a helyén marad, de az információ, ami előzőleg az egyikbe irányult, az utasítás után a másik ablakban jelenik meg. A BASIC által produkált (hiba)üzenetek elirányítására is módot ad ez az utasítás:

```
10 rem pl. rajzolo program
20 window #1, 1, 40, 20, 25                    - alsó öt sor
30 window swap 1,0                             - oda megy, ami
40 rem most mar nyugodtan rajzolhatunk,                 eddig a
50 rem ha hibaval leall, nem rontja el a rajzot!        nullásba
                                                        ment
```

Ha a fenti programrészlet 30-as sora után a képernyőre nem csak rajzolni, hanem írni is akarunk, akkor az egyes ablakba kell azt irányítani ('PRINT #1,'), különben a saját csapdánkba esünk ... . Ablakokkal egyszerűen megoldható a függőleges írás is, csak megfelelően keskeny ablakot kell definiálnunk, mondjuk egy karakter széleset:

```
1 rem FUGGOLEGES OSZLOPBA IR
10 cls
20 window #0,2,40,1,25                 - hogy a 'Ready' ne zavarjon
30 window #1,1,1,1,25                  - csak a bal szélső oszlop
40 print #1, "Ez egy fuggoleges szoveg!";
```

2.2.4 A CPC karakterkészlete, saját tervezésű karakterek

Új kulcsszavak: **SYMBOL AFTER**
SYMBOL

A CPC 256 különféle karaktert képes megjeleníteni. Ezek egy része a billentyűzeten is elérhető, a többit csak a 'CHR$' függvény segítségével lehet a képernyőre varázsolni. A CPC karakterkészlete a legmesszebbmenőkig alkalmazkodik az ASCII kódhoz. A gép összes karaktere felnagyított formában megtalálható a Függelékben. Most csoportosítsuk a 256 karaktert:

PRINT #n, adatlista (hasonlóan: WRITE #n,)

Csak az 'n' sorszámú ablakot használja a kiíráshoz, a képernyő többi részét szabadon hagyja. Szükség szerint felfelé görgeti a gyorsan betelő ablak tartalmát.

INPUT #n,"szöveg"; változó (hasonlóan: LINE INPUT #n,)

A "szöveg"-et és a kérdőjelet az 'n' sorszámú ablakban jeleníti meg, oda várja a választ is.

CLS #n

Az 'n' sorszámú ablakot törli az ablak papírszínére.

LOCATE #n,x,y

Egy ablak koordinátái annak bal felső sarkából indulnak és az ablak nagysága határozza meg a maximális értékeket. A CPC minden ablakra vonatkozóan nyilvántartja az aktuális kurzorpozíciót.

POS (#n)

Az 'n' ablak kurzorjának vízszintes pozícióját eredményező függvény. A zárójelben akkor is kötelező megadni az ablak sorszámát, ha a nullás (#0) ablakról van szó.

VPOS (#n)

Az 'n' ablak kurzorjának függőleges (vertikális) pozícióját eredményező függvény. Használata a 'POS'-hoz hasonló:

```
10 locate 20,20 :print "A";        - a 0-ás ablakba irunk
20 print pos(#0); vpos(#0)         - hol áll most a kurzor?
```

Mire is használhatók valójában az ablakok? Mindenek előtt adatok tagolt megjelenítésére. A rendszer gondoskodik róla, hogy az adataink csakis a kijelölt területen jelenjenek meg, arról ki nem lóghatnak, más területet véletlenül sem írhatnak felül.

Ezenkívül többféle, nem kevésbé fontos felhasználási lehetőséget találhatunk az ablakok számára. Egy ablak megnyitásával, papírszínének megadásával és letörlésével a lehető leggyorsabb módon festhetünk be egy területet. A következő program egy magyar zászlót varázsol azonnal a képernyőre:

```
1 rem MAGYAR ZASZLO
10 mode 1                        - elég lesz négy szín
20 ink 1,6                       - egyes szín a piros
30 ink 2,26                      - kettes szín a világos fehér
40 ink 3,9                       - hármas szín a zöld
50 window #1,1,40,1,8            - az első nyolc sor
60 window #2,1,40,9,16           - a második nyolc sor
70 window #3,1,40,17,24          - a harmadik nyolc sor
80 paper #1,1                    - fent piros papír
90 paper #2,2                    - középen fehér papír
100 paper #3,3                   - lent zöld papír
110 cls #1: cls #2: cls #3       - papírszínre törölni mindhármat
120 while inkey$="": wend        - ne zavarjon a 'Ready'
```

A tinták kiválasztása után (20-40-es sor), meghatároztuk az ablakok méreteit. Mindhárom ablakunk a képernyő bal szélétől a jobb széléig tart (1,40), a különbség a függőleges koordinátákban van: ahol az egyik ablak végződik, ott kezdődik a következő. Az ablakok papírszínének kiválasztása (80-100-as sor) nem okoz látható változást, amíg nem töröljük az ablakokat, ezt a 110-es sor végzi el.

Ablakokkal létrehozható effektust illusztrál a következő program is. Mindössze egy ablakot nyitunk meg újra és újra, de a paramétereit véletlenszerűen állítjuk be. 'MODE 0' garantálja a lehető legtarkább képernyőt.

```
1 rem SOK SZINES ABLAK
10 mode 0                        - minél több szín legyen
20 x= int(rnd*11)+1              - bal széle: 1-től 11-ig lehet
30 y= int(rnd*14)+1              - teteje: 1-től 14-ig lehet
40 xj=int(rnd*19)+x              - jobb széle = bal széle + véletlen
50 yj=int(rnd*12)+y              - alja = teteje + véletlen
60 window #1,x,xj,y,yj           - megnyitjuk egyes ablakként
```


A számolás közben feltűnhet, hogy a karakter függőleges vonalait két tollszínű ponttal adtuk meg. Miért van erre szükség? A színes monitor tökéletlensége miatt az egy pontból álló függőleges vonal már 'MODE 1'-ben is alig látszik, 'MODE 2'-ben pedig teljesen elnyomja a háttér színe, vagyis a 80 karakteres képernyő-üzemmódban színes monitoron olvashatatlan lenne a kép. A megoldás: minden függőleges vonalat meg kell duplázni.

Kis 'é'-betűnknek mind a nyolc sorát kiszámítottuk, nem marad más hátra, definiáljuk a karaktert a 'SYMBOL' utasítás segítségével. De melyik kódszámú karakter helyére? Egyelőre válasszuk az utolsót, a 255-öst:

```
symbol 255, 12, 24, 60, 102, 126, 96, 60, 0      - definiáljuk
Ready
print chr$(255)                                  - kipróbáljuk
é                                                - íme ő
Ready
```

Egy karakter mintájának kiszámítása tízes számrendszerben egy kicsit nehézkes és, mivel a CPC megérti a bináris számokat is, feleslegesnek tűnhet. A kis 'é'-betű alakját közölhetjük így is:

```
10 a=&x00001100                  - '&x' = bináris szám jön,
20 b=&x00011000                    minden sor egy-egy byte,
30 c=&x00111100                    minden byte 8 bit,
40 d=&x01100110                    minden bit '0' vagy '1',
50 d=&x01111110                    ha '1' akkor tollszínű
60 e=&x01100000                    lesz a pont, ha '0' akkor
70 f=&x00111100                    papírszínű.
80 g=&x00000000
90 symbol 255, a,b,c,d,e,f,g     - saját számaival terveztünk
```

Ennek a módszernek csak egy hátránya van: sok helyet foglal el. Tehetjük a bináris számokat 'DATA' sorokba, de igazán sokat akkor sem nyerünk. Két megoldás is kínálkozik: készítsünk egy egyszerű karaktertervező programot, amely kiszámítja helyettünk a sorok adatait. A másik megoldás szintén egy program, amely nemcsak számol, hanem el is helyezi a karaktert a memóriában,

egyszerre megjeleniti az összes áttervezhető karaktert és kívánságra külső tárba viszi ki őket, ahonnan az egész áttervezett karakterkészletet azonnal betölthetjük az operatív tárba. Egy ilyen, minden igényt kielégitő karaktergenerátor program található a Harmadik szinten. Egyelőre készitsük el az egyszerűbbet:

```
10 REM ********************
20 REM *                  *
30 REM * KARAKTER TERVEZO *
40 REM *                  *
50 REM ********************
60 DIM matr(8,8),sor(8)
70 MODE 1
80 PRINT STRING$(40,"-");
90 PRINT"            Karakter tervezo"
100 PRINT STRING$(40,"-")
110 PRINT TAB(24)"Kezeles: nyilak"
120 PRINT TAB(24)"      es [COPY]"
130 PRINT TAB(24)"      billentyu"
140 FOR i=8 TO 1 STEP -1
150 PRINT i;CHR$(159)
160 NEXT
170 PRINT"   ";CHR$(147);
180 FOR i=1 TO 8
190 PRINT CHR$(159);
200 NEXT
210 PRINT:PRINT"    12345678"
220 PRINT:PRINT:PRINT"Ha kesz: [ENTER] !"
230 x=1:y=1
240 b$=INKEY$
250 IF matr(x,y)<>1 THEN LOCATE 4+x,16-y:PRINT CHR$(32)
260 IF b$=CHR$(240) THEN IF y<8 THEN y=y+1 ELSE PRINT CHR$(7)
270 IF b$=CHR$(241) THEN IF y>1 THEN y=y-1 ELSE PRINT CHR$(7)
280 IF b$=CHR$(242) THEN IF x>1 THEN x=x-1 ELSE PRINT CHR$(7)
290 IF b$=CHR$(243) THEN IF x<8 THEN x=x+1 ELSE PRINT CHR$(7)
300 IF b$=CHR$(224) AND matr(x,y)=0 THEN matr(x,y)=1:GOTO 340
310 IF b$=CHR$(224) AND matr(x,y)=1 THEN matr(x,y)=0
320 IF b$=CHR$(13) THEN 350
330 LOCATE 4+x,16-y:PRINT CHR$(143)
340 GOTO 240
```



```
350 rem ------------  kiszamolja a bitmintat
360 FOR y=8 TO 1 STEP -1
370 FOR x=1 TO 8
380 IF matr(x,y)=1 THEN sor(9-y)=sor(9-y)+2↑(8-x)
390 NEXT x
400 NEXT y
402 FOR i=1 to 8
404 LOCATE 16,7+i
406 PRINT sor(i)
408 NEXT i
410 SYMBOL 255,sor(1),sor(2),sor(3),sor(4),sor(5),sor(6),sor(7),
sor(8)
420 LOCATE 24,10
430 FOR i=1 TO 16:PRINT CHR$(255);:NEXT
440 LOCATE 24,14
450 INPUT"ujra: [ENTER]!",v$
460 RUN
```

A program fontosabb elemei:

60: dimenzionálja a karaktermátrix 8*8-as tömbjét 'matr(8,8)' és a nyolc sor adatait tároló 'sor(8)' vektort.

70-130: címfelirat, használati utasitás

140-220: a tervezőhely, a keret kirajzolása grafikus karakterekkel

230: kiinduló koordináták a kereten belül: x=1, y=1

240-340: a fő hurok, lenyomott billentyűre vár 'b$', közben villogtatja a saját kurzort: 'chr$(32)' és 'chr$(143)'. A lenyomott billentyű lehet nyil 'chr$(240)-chr$(243)' vagy [COPY] 'chr$(224)', esetleg [ENTER] 'chr$(13)'. Nyilak esetén a kereten belül mozgunk, kerethatár túllépése helyett csöngetünk: print chr$(7). Ha a [COPY]-t nyomták le, ki kell fordítani a mátrix aktuális elemét: 0-ból 1 lesz, 1-ből 0. [ENTER] kilép a hurokból.

350-400: a mátrix elemei alapján kiszámitja a karakter nyolc soranak adatait.

402-408: a keret mellé kiírja minden sor kiszámított adatát

410: áttervezi a 255-ös karaktert

420-460: kiír egy sort az áttervezett karakterből. Újra indítja a programot az [ENTER] lenyomása után.

A program kezelése egyszerű: a nyílbillentyűk segítségével mozoghatunk a 8*8-as kereten belül és a [COPY] billentyűvel beállíthatjuk az üres pontot vagy törölhetjük a beállított pontot. Az [ENTER] billentyűvel jelezzük a tervezés végét: a keret mellett megjelennek a kiszámított karaktersorok adatai, ezeket felhasználhatjuk a 'SYMBOL' utasításban. Ráadásul a program a kívánt mintára definiálja a 255-ös karaktert és kiír belőle egy sort, hogy valódi nagyságában is megtekinthessük újonnan tervezett karakterünket.

Karakterek áttervezésével a képernyő urai lehetünk, hiszen egy karakter kiíratásával egyszerre 64 pontot jelenítünk meg a képernyőn. Tetszőleges típusú betűkészletek mellett tervezhetünk több karakterből összeállítható alakzatokat is, ezeket szöveges változóban tárolhatjuk és egy 'PRINT'-tel a képernyőre vihetjük. Természetesen az ékezetes magyar betűknek sincs már akadálya, de mivel azokhoz a billentyűzetet is át kell tervezni, későbbre hagyjuk őket.

2.2.5 A vezérlő karakterek

A 0-31 kódszámú karaktereket nem jeleníthetük meg a 'PRINT CHR$(n)' utasítással, mert ahelyett, hogy látható ábrát produkálnának a képernyőn, különböző cselekvést hajtanak végre. Ezek a cselekvések egyrészt a kurzor mozgatását (vezérlését) végzik, de van közöttük olyan is, amelyre a CPC-nek megvan a külön BASIC utasítása. Az a tény, hogy ezeket a cselekvéseket a 'CHR$' függvénnyel hívhatjuk meg, lehetővé teszi, hogy karakterláncokba ágyazzuk őket. Például a 'CHR$(7)' kiíratása csönget egyet:

```
print chr$(7)                  - nincs a végén pontosvessző!
                               - csöngetett és egy üres sort adott
Ready
f$="FIGYELEM!"+chr$(7)         - figyelemfelkeltés képpel, hanggal
Ready
print f$
FIGYELEM!                      - kiírja és csönget is
```


Az alábbiakban áttekintjük a vezérlő karaktereket és az általuk végrehajtott cselekvéseket. Némely vezérlő karakter paramétert kíván, a paramétereket ugyancsak karakterek formájában kell beadni.

Vezérlő karakter	Paraméter értéke (vagy példa)	A karakter hatása
CHR$(0)		- nincs hatása
CHR$(1)	0 - 255	- megjeleníti a paraméterként megadott karaktert, még akkor is, ha az vezérlő karakter
	`print chr$(7)`	- csönget
	`print chr$(1);chr$(7)`	- csörgő órát jelenít meg
CHR$(2)		- a kurzort kikapcsolja, hasznos például 'INPUT' utasítás előtt
CHR$(3)		- a kurzort újra bekapcsolja
CHR$(4)	0 - 2	- a 'MODE' parancsnak felel meg
	`print chr$(4);chr$(0)`	- 'MODE 0'-ba vált át
CHR$(5)	0 - 255	- a paraméterben megadott karakter képét a grafikus kurzorpozíción jeleníti meg (vö. TAG utasítás)
	`plot 100,100: ? chr$(5);"A"`	- a 100,100 grafikus pozíción kezdődik az 'A'
CHR$(6)		- engedélyezi a képernyőt, vö. CHR$(21)
CHR$(7)		- csönget egyet
CHR$(8)		- a kurzort egy karakterrel balra

mozgatja

```
print " a";chr$(8);chr$(8);"b"    - "ba"-t ir ki
```

CHR$(9) - a kurzort egy karakterrel jobbra mozgatja

CHR$(10) - a kurzort egy sorral lejjebb viszi

```
print "a";chr$(10);"b"            - "à
                                      b"-t ir ki
```

CHR$(11) - a kurzort egy sorral feljebb viszi

CHR$(12) - a 'CLS' utasitás megfelelője

CHR$(13) - a kurzort az aktuális sor elejére állitja

CHR$(14) 0 - 15 - a 'PAPER' utasitás megfelelője

CHR$(15) 0 - 15 - a 'PEN' utasitás megfelelője

```
a$=chr$(15)+chr$(3)+"AAA"+chr$(15)+chr$(1)
- ezek után PRINT a$ = PEN 3: PRINT "AAA": PEN 1
```

CHR$(16) - a kurzor alatti karaktert kitörli az aktuális papir színére

```
locate 1,1: print "a"          - ott az 'a'-betű
locate 1,1: print chr$(16)     - már nincs ott
```

CHR$(17) - a sor kezdetétől a kurzorpozicióig (azt is beleértve) töröl

```
print "aaaaaaaaaa"; chr$(17)  - kiirja és le is törli
```

CHR$(18) - a kurzorpozíciótól (azt is beleértve) a sor végéig töröl


```
locate 1,1: print string$(40,"*")   - egy sor csillag
locate 1,20: print chr$(18)         - a felét kitörli
```

CHR$(19) - az aktuális ablakot annak bal felső sarkától a kurzorpozicióval bezárólag a papír színére törli

CHR$(20) - az aktuális ablakot a kurzorpozíciótól a jobb alsó sarokig a papír színére törli

CHR$(21) - a képernyőt letiltja: az csak a következő CHR$(6)-ra reagál újra

CHR$(22) 0 - 1 - átlátszó megjelenitést kapcsol be (1), vagy kapcsol ki (0)

```
locate 1,1: print "=";                      - egyenlő
locate 1,1: print chr$(22); chr$(1); "/"    - áthúzzuk!
-ezek után gyorsan kapcsoljon vissza nemátlátszóra!
```

CHR$(23) 0 - 3 - a grafikus megjelenités módusát határozza meg: 0 - normális
1 - XOR
2 - AND
3 - OR
(ismertetése a 2.3.7 pontban)

CHR$(24) - felcseréli a toll és a papír színét: inverz megjelenités

```
a$=chr$(24)+"Ez mindig inverz lesz"+chr$(24)
print a$
```

CHR$(25) (9 paraméter) - a kötelező 9 paraméterével egy karaktert definiál, a 'SYMBOL' utasitás megfelelője

CHR$(26) (4 paraméter) - a kötelező 4 paraméterével abla-

kot definiál: 'WINDOW #0'

CHR$(27)		- nincs hatása
CHR$(28)	(3 paraméter)	- a kötelező 3 paraméterével egy tinta színét állítja be, az 'INK' utasításnak megfelelően
CHR$(29)	(2 paraméter)	- a kötelező 2 paraméterével a 'BORDER' utasítás megfelelője
CHR$(30)		- a kurzort az aktuális ablak bal felső sarkába állítja
CHR$(31)	1 - 80 1 - 25	- a kurzort az aktuális ablakban a kötelezően megadott paraméterek által jelzett helyre állítja, a 'LOCATE' utasítás megfelelője

A vezérlő karakterek közül ki kell emelni a 24-est (inverz ki-be), a 22-est (átlátszó megjelenítés ki-be) és a kurzormozgató karaktereket. A kurzormozgató karakterekkel kombinált karaktergrafikával nagyobb ábrákat is egyszerűen megjeleníthetünk:

```
1 rem ABRA TOBB KARAKTERBOL
10 mode 1
20 teteje$=chr$(214)+chr$(215)                  - két grafikus ábra
30 alja$=chr$(213)+chr$(212)                    - két grafikus ábra
40 le2vissza$=chr$(10)+chr$(8)+chr$(8)          - alsó részhez beáll
50 fel$=chr$(11)                                - folytatáshoz vissza
60 abra$= teteje$+le2vissza$+alja$+fel$         - összeáll az egész
70 for t=1 to 10: print abra$; :next t          - lássuk csak!
80 print: print                                 - különben ráír
90 rem folytatjuk!
```

A 'le2vissza$'-val a kurzort egy sorral le és két pozícióval balra léptetjük, hogy a 'teteje$' alá álljon, mert oda kerül az 'alja$'. Kirajzolva az 'alja$'-t, egy sorral feljebb lépünk, hogy az 'abra$'-k egymás mellé kerüljenek. Módositsuk úgy a programot, hogy minden második 'abra$' inverz legyen:


```
65 abra$=chr$(24)+abra$                         - bekapcsol-kikapcsol
```

▶▶▶ A vezérlő karaktereket a 'CHR$(n)' függvény helyett a billentyűzetről is beadhatjuk. A [CTRL] billentyűt lenyomva tartva, a lenyomott betűbillentyűk saját kódjuknál 64-gyel kisebb kódszámot adnak: például [CTRL] [G] használható a 'CHR$(7)' helyett (71 - 64 = 7). A kapott grafikus karaktereket karakterláncokba ágyazhatjuk, kiíratáskor már mint a kívánt cselekvés jelenik meg. A programot, amely grafikus formában tartalmazza a vezérlő karaktereket, nyomtatóra nem lehet kilistázni, mivel a vezérlő karakterek a nyomtatót is különböző cselekvésekre fogják ösztönözni ...

A vezérlő kódokkal a CPC karaktergrafikáját maximálisan kihasználhatjuk. Igazán finom felbontáshoz azonban ez sem elég, valódi, pontonként címezhető pixel*-grafikára van szükség.

2.3 NAGYFELBONTASÚ GRAFIKA

2.3.1 A CPC grafikus koordinátarendszere

A CPC nagyfelbontású grafikája a gép egyik erőssége. A képernyő bármelyik pontja a rendelkezésekre álló színek bármelyikét felveheti, szöveg és grafika tetszés szerint keverhető, BASIC utasítások tucatja segíti elő a grafika könnyű és gyors kezelését. Mint már szó volt róla, a képernyőn vízszintes irányban megcímezhető pontok száma a 'MODE' utasítás paramétere szerint változik, vele együtt változik az egyszerre felhasználható színek száma is. A legnagyobb felbontás 'MODE 2'-ben érhető el: 640*200 = 128000 pont, minden pont színe két szín közül választható ki. A "legrosszabb" felbontást 'MODE 0'-ban kapjuk: 160*200 vagyis 32000 pontot ábrázolhatunk a képernyőn, de ekkor 16 szín áll rendelkezésunkre. Grafikában is a 'MODE 1' az arany középút, 320*200 = 64000 pontot állíthatunk be négy szín bármelyikére.

A CPC grafikája nem ismeri a más gépekre jellemző korlátozásokat: egy karakterpozíción belül 'MODE 0'-ban akár 16 színt is használhatunk. A képernyő nem oszlik grafikus és szöveges részre, az egész képernyőre írhatunk és rajzolhatunk, egyiket a másik hátára is. A legtöbb képernyőt és színeket kezelő utasítás

(pl. CLS, INK, SPEED INK stb.) az egész képernyőt (vagy a kívánt ablakot) befolyásolja, függetlenül attól, hogy ott éppen szöveg vagy grafika van.

Amint látható, a különböző képernyő-üzemmódokban csak a vízszintesen megkülönböztethető pontok száma változik, függőlegesen a gép mindig 200 pontos felbontással dolgozik. Hogy a BASIC grafikus utasításai és azok paraméterei ne függjenek a mindenkori 'MODE'-tól, a CPC minden üzemmódban egységes grafikus koordinátarendszerrel dolgozik, ennek főbb jellemzői:

- a 0,0 koordináta a képernyő bal alsó sarkában van
- vízszintesen 640 koordinátapozíció: 0 <= x <= 639.
- függőlegesen 400 koordinátapozíció: 0 <= y <= 399.

A koordinátarendszer előnye a 'MODE'-októl való függetlenségen kívül a megfelelő x/y arány: négyzetek, körök rajzolásánál minden 'n' lépés vízszintes irányban ugyanakkora távolság lesz, mint 'n' lépés függőleges irányban, más szóval a rajzolt idomok oldalarányosak lesznek. A koordinátarendszer így függőlegesen kétszer annyi koordinátapozíciót tartalmaz, mint amennyi pont valójában található a képernyőn. Vízszintes irányban 'MODE 2'-ben minden koordinátapozíciónak valóban megfelel egy-egy pont a képernyőn, 'MODE 1'-ben minden második pozícióra esik egy pont, 'MODE 0'-ban minden negyedikre. A rendszer a kívánt koordinátapozíciót szükség esetén lefelé kerekíti a legközelebbi képernyőpont pozíciójára. Ez a kerekítés legjobban 'MODE 0'-ban figyelhető meg (a 'PLOT' utasítás egy grafikus pontot állít be a megadott koordinátapozíción a toll színére):

```
mode 0                          - minden negyedik pozíció új pont
plot 320,200                    - középen egy "lapos" pont
plot 321,200                    - semmi hatás (320,200-ra tette)
plot 322,200                    - semmi hatás (ezt is oda tette)
plot 323,200                    - még mindig nem látni semmit
plot 324,200                    - új pont a régitől jobbra!
Ready
```

A koordinátarendszer használata egyszerű és kényelmes, a hibalehetőség csekély, mivel a BASIC elfogadja a lehető legképtelenebb (a képernyőn kívül eső) koordinátapozícókat is, legfeljebb nem csinál semmit. Próbálja ki: 'plot -1000, 2222'. Bár ez a koordinátarendszer a legtöbb feladat megoldásához ideális, a CPC azt is lehetővé teszi, hogy a felhasználó tetszőlegesen eltolja a koordinátarendszert: például a rendszer 0,0 pontját (az origót) a képernyő közepére helyezze.

2.3.2 Grafikus kurzor, pontok és vonalak

Új kulcsszavak: **PLOT**
DRAW
MOVE

A CPC nemcsak a nyolc szöveges ablak kurzorpozícióját tartja nyilván, hanem egy grafikus kurzort is kezel. A grafikus kurzor ugyan láthatatlan, de a pozíciója a további utasítások kiindulópontjaként szolgálhat. A legelemibb grafikus utasítás egy grafikus pontot tesz a paraméterekben megadott pozícióra, a megadott tollszínben. Ha 'toll'-at elhagyjuk, a CPC az utoljára felhasznált grafikus 'toll'-lal teszi ki a pontot. (Alapállapotban ennek színe megegyezik a szöveges toll színével.)

```
PLOT x,y,toll                   - a 'toll' elhagyható
```

Nézzük meg a különböző 'MODE'-okban a legkisebb képernyőpont méretét és alakját:

```
1 rem EGY GRAFIKUS PONT MERETE
10 ink 0,0: ink 1,26                    - feketén fehérrel
20 for m=0 to 2
30 mode m
40 print "MODE"; m
50  for p=1 to 300                      - 300 pontot teszünk ki
60   x=rnd*639: y=rnd*399               - véletlen pozícióra
70   plot x,y,1                         - fehér színben
80  next p
90 input "tovabb [ENTER]", v$
100 next m                              - következő 'MODE'
```

A pontok aránytalansága 'MODE 0'-ban a legfeltünőbb. A fekete-fehér színösszeállitásra azért van szükség, mert a 'MODE 2' finom felbontásában színes monitoron egy magányos pont egyébként beleolvadna a háttérbe.

A grafikus kurzort háromféle módon is mozgathatjuk: a most megismert 'PLOT' utasitással a kurzor az utoljára kitett pont pozicióján található. A 'MOVE' utasitással a grafikus kurzort az utasitás paramétereként megadott új pozición helyezhetjük el:

MOVE x,y — grafikus kurzort x,y-ra

A 'MOVE' utasitással csak a grafikus kurzort helyezzük el, más, látható cselekvés nem történik. A kurzortól vonalat húz a 'DRAW' utasitás a paramétereiben megadott pozicióig, az opcionálisan megadott tintával. A vonal meghúzása után a vonal végpontja lesz az új grafikus kurzorpozició.

DRAW x,y,toll — x,y-ig vonalat húz

A következő program a 320,0 koordinátájú pontot köti össze a 399-es sor összes pontjával:

```
10 mode 1                        - minden második pozició
20 for x=0 to 639 step 2           külön képernyőpont!
30 move 320,0                    - alsó sor közepéből
40 draw x,399,1                  - vonalat húz a felső sor
50 next x                          'x'-poziciójába
```

Ha a lépésközt 1-re módositjuk, az ábra nem változik, csak a megjelenités lesz lassabb: minden második vonal egybeesik az előzővel. A lépésközt növelve (pl. 'STEP 4') legyezőmintát kapunk, érdemes kipróbálni! Rajzoljunk egy sokszinben csillogó gyémántot, természetesen 'MODE 0'-ban:

```
1 rem SZINES GYEMANT
10 mode 0                        - 16 szin
20 toll =1                       - első szin
30 for x=0 to 639 step 4         - elég minden 4. pozició
40 move 320,399                  - felső kiinduló pont
```



```
50 draw x,200, toll : draw 320,0     - két vonalat húzunk
60 if x mod 16=0 then toll =toll+1   - minden 4 vonal után
70 if toll =16 then toll =1            új tollat veszünk elő
80 next x
```

A 'Ready' felirat az 1,1 szöveges kurzorpozición jelenik meg, mivel a 'MODE' utasitás után nem irtunk ki semmit. Az 50-es sor második 'DRAW' utasitásában nem szükséges a 'toll' számát újra megadni, ha az előző grafikus tollat kivánjuk használni.

2.3.3 Minden relativ

Új kulcsszavak: **PLOTR**
DRAWR
MOVER

A CPC BASIC nem csak az eddig használt <u>abszolút</u> pozicómegadást teszi lehetővé, megadhatjuk a grafikus kurzor új helyét - például egy új pont pozicióját - az aktuális grafikus kurzor pozicójához viszonyitva, <u>relative</u> is. A 'PLOTR','DRAWR','MOVER' utasitások paramétereivel tulajdonképpen az előző pozicótól koordinátapontokban mért távolságot kell megadni (a 'toll'-at itt is elhagyhatjuk):

PLOTR x-távolság, y-távolság, toll
DRAWR x-távolság, y-távolság, toll
MOVER x-távolság, y-távolság

Az 'x-távolság' pozitiv, ha ez előző ponttól jobbra lesz az új pont, negativ, ha balra. Az 'y-távolság' pozitiv, ha az előző ponttól felfelé lesz az új pont, negativ, ha lefelé.

```
move 320,200 : drawr 100, -200
```

A fenti parancsokkal a képernyő közepét a tőle jobbra 100 és lefelé 200 pontnyi távolságra levő koordinátaponttal kötjük össze. A grafikus kurzor új abszolút pozicója igy 420,0 lesz.

A relativ pozicómegadás segitségével rendkivüli módon leegyszerűsödik a különböző idomok rajzolása, hiszen például

egy 25 pontnyi oldalhosszúságú négyzetet a képernyő bármely részén ugyanazzal az utasítással helyezhetünk el:

```
99 rem NEGYZETRAJZOLO SZUBRUTIN
100 drawr 25,0,toll: drawr 0,25: drawr -25,0: drawr 0,-25:
    return
```

Ez a szubrutin a kiindulóponttól először jobbra húz egy 25 pont hosszúságú vonalat, majd onnan felfelé, annak a végétől balra és végül lefelé: kirajzol egy négyzetet. Helyezzük el egy programban:

```
10 mode 1                                        - négy szín
20 toll=1                                        - első szín
30 x=rnd*639: y=rnd*399                          - véletlen pozíció
40 move x,y                                      - gr. kurzort elhelyez
50 gosub 100                                     - egy kis négyzet
60 toll=toll+1: if toll=4 then toll=1            - új szín
70 goto 30                                       - sok-sok kis négyzet
```

A CPC BASIC-jéből hiányzó körrajzoló utasítást többféleképpen is pótolhatjuk. Elrettentő például szolgáljon a következő program:

```
1 rem LASSU KOR
10 deg                                           - szögfokban számolunk
20 sugar=100: kx=320: ky=200                     - sugár, középpont
30 for i=0 to 360                                - minden pontot
40 plot kx+sugar*sin(i), ky+sugar*cos(i)           egyenként kiszámol
50 next i                                          és kirajzol
```

Ha a 'PLOTR' utasítást használjuk, egy számítás alapján egyszerre négy pontot is kirajzoltathatunk, ami jelentősen felgyorsítja a kör kirajzolását:

```
1 rem GYORSABB KOR
10 deg
20 sugar=100: kx=320: ky=200                     - sugár és középpont
30 for t=0 to 90                                 - negyed kör
40 x=sugar*sin(t): y=sugar*cos(t)                - egy pontot kiszámol
```



```
50 move kx,ky: plotr x,y                         - a négy körív egy-egy
60 move kx,ky: plotr x,-y                          pontját kirajzolja
70 move kx,ky: plotr -x,-y
80 move kx,ky: plotr -x,y
90 next t
```

Ha beérjük egy kicsit szögletesebb körrel, összeszedjük rég elfelejtett matematikai ismereteinket és a számítások egy részét még a program elején elvégezzük, a kör sokkal gyorsabban készül el:

```
1 rem BONYOLULT KOR
10 deg
20 sugar=100: kx=320: ky=200
30 n=int(pi*sqr(sugar)+1): sn=sin(360/n): cs=cos(360/n)
40 x=sugar: y=0: move kx+sugar,ky
50 for i%=1 to n
60 pont=cs*x-sn*y: y=sn*x+cs*y: x=pont
70 draw kx+x, ky+y
80 next i%
```

2.3.4 Hol vagyunk és mi van alattunk?

Új kulcsszavak: **XPOS**
YPOS
TEST
TESTR

Gyakrak van szükség arra, hogy tájékozódjunk a grafikus kurzor pillanatnyi helyzetéről. A vízszintes és a függőleges pozíció lekérdezésére egy-egy függvény szolgál:

XPOS megadja a grafikus kurzor vízszintes pozícióját
YPOS megadja a grafikus kurzor függőleges pozícióját

```
move 444,333                                     - nem látunk semmit, de ott van!
Ready                                            - a szöveges kurzor marad a régi
print xpos, ypos                                 - lekérdezzük a grafikus kurzort
 444          333
Ready
```

Ha ezek után egy 'DRAW' utasítást adunk ki, az természetesen a 444,333 koordinátájú pontból indulna el. A láthatatlan grafikus kurzorral kapcsolatban szükségünk lehet a pillanatnyi pozíción kívül annak a pontnak a színére is, amelyik az adott pozíción (a kurzor alatt) található. Játékprogramokban például a legegyszerűbben így állapíthatjuk meg, hogy "ütköztünk-e" vagy még szabad az út. A 'TEST' függvény adja meg egy abszolút pozíción található pont színét. A 'TESTR' függvény pedig a többi relatív utasításhoz hasonlóan a pillanatnyi grafikus kurzortól a kívánt x-távolságra és y-távolságra eső pont színét közli velünk. A színen természetesen annak a tollnak a sorszámát kell érteni, amellyel az adott pontot kitettük. Ha a megadott pozíción nincs világító pont, akkor az papírszínű, vagyis a 'TEST' függvény eredménye nulla lesz.

TEST (x,y) megadja a x,y koordinátájú pont színét
TESTR (x-távolság,y-távolság) megadja a grafikus kurzortól x,y-távolságra eső pont színét

```
mode 1
move 0,0: draw 399,399,2                              - kettes toll!
for v=20 to 40: print test(v,30);: next
```

Az eredmény: tíz nulla, két kettes és újra kilenc nulla. A nullákkal nincs baj: 'PEN 0' a papír színe, de miért van két kettes is? Nem szabad elfelejtenünk, hogy 'MODE 1'-ben vagyunk, minden képernyőpont vízszintesen és függőlegesen egységesen két koordinátapozíciót foglal el. Ezért esetünkben a 'TEST(30,30)' és a 'TEST(31,30)' ugyanazt a képernyőpontot ellenőrizte, így kaptunk két kettest.

▶▶▶ A 'TEST' és 'TESTR' függvények nemcsak egy pont színét adják eredményül, hanem a grafikus kurzort is elhelyezik az argumentumként megadott pozíción!

Az új függvények használatát legjobban egy játékprogrammal lehet bemutatni. A KÍGYÓ egyszerű, de élvezetes játék: a képernyő közepén elinduló kígyót a négy nyílbillentyűvel úgy kell irányítani, hogy az nehogy nekiütközzék a keretnek, a sokasodó pöttyöknek vagy saját magának. A játék nyilvántartja a leghosszabb kígyó hosszát, pontokban kifejezve.

```
10 REM ***********
20 REM *         *
30 REM *  KIGYO   *
40 REM *         *
50 REM ***********
60 MODE 1 : RANDOMIZE TIME
70 PRINT "**** KIGYO ****": PRINT
80 INPUT "Valassz fokozatot 1-99: ",fok
90 IF fok<1 OR fok>99 THEN 80
100 fok = fok/100
110 legjobb = 0
120 INK 0,0 : INK 1,14 : INK 2,24 : INK 3,26
130 SPEED INK 5,5:BORDER 0 : PAPER 0
140 CLS: PRINT "Az eddigi leghosszabb kigyo:"; legjobb ;"pont"
150 hossz=0
160 PLOT 0,0
170 DRAW 639,0 : DRAW 639,380 : DRAW 0,380 : DRAW 0,0
180 x = 300 : y = 190 : b = INT(RND*4) : nyil = 3-b
190 b$ = INKEY$ : IF b$<>"" THEN nyil = ASC(b$)-240
200 IF nyil >= 0 AND b <4 THEN b = nyil
210 IF b = 0 THEN y = y+2
220 IF b = 1 THEN y = y-2
230 IF b = 2 THEN x = x-2
240 IF b = 3 THEN x = x+2
250 IF TEST(x,y)<>0 THEN 310
260 PLOT x,y,2
270 hossz=hossz + 1
280 p = INT(RND*638+1) : q = INT(RND*378+1)
290 IF TEST(p,q) = 0 AND RND<fok THEN PLOT p,q,3: PLOTR 2,0: PLO
TR 0,2: PLOTR -2,0
300 GOTO 190
310 REM utkozes
320 IF hossz>legjobb THEN legjobb = hossz
330 INK 0,6,1: BORDER 1,6
340 IF INKEY$ <> "" THEN 340
350 IF INKEY$ = "" THEN 350
360 GOTO 120
```

A program ismertetése

```
60:       4 szín; a véletlenszám-generátort állítja be
80-100:   'fok': nehézségi fokozat, az akadálypontok sűrűsége
110:      'legjobb': a leghosszabb kígyó hossza
120-130:  színeket beállít, villogás sebessége (ütközés után)
150:      induló kígyó kezdő 'hossz'-a nulla
160-170:  keretet rajzol
180:      kígyó vége: 'x','y'; 'b': új irány, 'nyil': régi irány
190:      van billentyű lenyomva? Ha igen, akkor 'nyil' = billen-
          tyű kódja - 240
200:      ha nyílbillentyű volt lenyomva (kódja >=240), akkor
          'b' = új irány
210-240:  az iránynak megfelelően a kígyó új koordinátáit számít-
          ja ki, 'MODE 1'-ben két koordinátapontot kell lépni,
          hogy valóban lépjünk egyet!
250:      a kígyó új pozíciója üres? Ha igen, akkor 'TEST(x,y)=0'
          Ha nem, akkor a kígyó akadályba ütközött: 310-es sor.
260:      kígyó tovább lép
270:      és a 'hossz'-a növekszik
280:      új akadálypontnak véletlen hely a kereten belül: 'p,q'
290:      ha 'TEST(p,q)=0' (papír) akkor a 'fok'-ozatnak megfele-
          lően esetleg kirajzol egy akadálypontot, nagysága 2*2
          képernyőpont
300:      vissza a főhurokba
320:      átveszi a 'legjobb'-ba a kígyó 'hossz'-át, ha új rekord
          volt
330:      papír és keret ellenkező színekben kezd villogni
340:      a billentyűzet-puffert kiüríti
350-360:  billentyűre vár majd új játékot kezd
```

2.3.5 Saját koordinátarendszer, grafikus ablak

Új kulcsszavak: **ORIGIN**
CLG
GRAPHICS PEN ♦ csak CPC 664-en
GRAPHICS PAPER ♦ és CPC 6128-on

A CPC grafikus lehetőségeit egyedülállóvá teszi az a tény, hogy a felhasználónak lehetőséget nyújt saját koordinátarendszer

bevezetésére. Mivel a koordinátarendszer kezdőpontját tetszésünk szerint elhelyezhetjük a képernyőn, a 0,0 pont lehet akár a képernyő közepén is:

```
origin 320,200          - új 0,0 pont a képernyő közepén
```

Az **'ORIGIN'** utasítás paraméterei az új koordinátarendszer 0,0 pontjának koordinátáit adják meg. Az utasítás hatására a CPC minden 'PLOT', 'MOVE' és 'DRAW' utasítás paraméterét az új 0,0 ponthoz viszonyítva értelmezi:

```
plot 0,0                - a képernyő közepén jelenik meg
plot 100,100            - tőle jobbra fel 100 pontnyira
plot -100,-100          - az első ponttól balra le
```

Láthatjuk, hogy új koordinátarendszerünkben már negatív abszolút pozíciójú pontok is ráférnek a képernyőre, ellentétben a gép alap-koordinátarendszerével, amikor ezek a pontok a képernyőn kívül estek. Ez a tény egy egész sor felesleges számítást takarít meg nekünk. Például a 'SIN' és 'COS' függvények eredménye -1 és +1 között van. Ha negatív értékek is megjeleníthetők 'PLOT'-tal, egy kör kirajzolása egysoros programmal is megoldható, feltéve, hogy az előző 'ORIGIN 320,200' még érvényben van:

```
deg: s=100: for t=0 to 360: plot sin(t)*s, cos(t)*s : next t
```

A képernyő közepén lassan, de biztosan kirajzolódik egy 100-as sugarú kör. Ugyanezzel az egy sorral elhelyezhetjük a kört a képernyő tetszőleges pontján, ha előtte beállítjuk az új 0,0 pontot, természetesen az 'ORIGIN' utasítással:

```
1 rem KOROK RAJZOLASA ORIGO ATHELYEZESEVEL
10 locate 1,1: input "Uj origo koordinatai "; x,y
20 origin x,y
30 input "A kor sugara: "; s
40 deg
50 for t=1 to 360: plot sin(t)*s, cos(t)*s: next t
60 goto 10
```

Próbáljunk lehetséges és lehetetlen paramétereket adni az 'ORIGIN' utasításnak, a CPC egy bizonyos határon belül mindent eltűr és gondoskodik róla, hogy a képernyőn kívül eső pontok valóban ne jelenjenek meg. Egy 100-as sugarú körnek csak a harmadik negyede látható a képernyő jobb felső sarkában, ha a 10-es sor 'INPUT'-jára 639,399-et válaszolunk.

Az 'ORIGIN' utasításnak eddig két paraméterét ismertük meg. Az utasítást azonban még további négy paraméter is követheti, amelyek egy grafikus ablak helyét határozzák meg a 'WINDOW'-hoz hasonlóan:

ORIGIN x0,y0, balszél, jobbszél, teteje, alja

Az utasítás további paramétereit is koordinátapontokkal kell megadni. A definiált grafikus ablaknak a többi ablakhoz hasonló tulajdonságai vannak: saját papírszíne és tollszíne van. A következő grafikus utasításoknak csak az ablak területén van hatása. A rendszer gondoskodik róla, hogy a grafikus ablakon kívülre eső képernyőre még véletlenül se juthasson ki a grafikus kurzor.

```
1 rem GRAFIKUS ABLAK DEMO
10 mode 1                                  - képernyő alapállapot
20 origin 200,200, 200,400, 200,300        - grafikus ablak
30 x=rnd*1000-500 : y=rnd*1000-500         - "vad" koordináták
40 draw x,y                                - meddig húz vonalat?
50 goto 30
```

A definiált ablak 0,0 pontja a régi koordinátarendszer szerinti 200,200-as ponton van. Az ablak 200 pont széles és 100 pont magas, a saját koordinátái tehát az x-tengelyen a 0-200, az y-tengelyen pedig a 0-100 intervallumba esnek. A 30-as sorban -500 és +500 közötti véletlen számokat generálunk, ezek többsége kívül esik a grafikus ablak határain. Ennek ellenére a 'DRAW' utasítást csak az ablakon belül eső képernyőtartományban hajtja végre a CPC, a képernyő többi része érintetlen marad.

A grafikus ablakban is a már megismert utasításokat használhatjuk. A grafikus toll és papír színéről el kell még mondani

a következőket:

- Alapállapotban a grafikus toll színe megegyezik az egyes tintáéval, a grafikus papír színét pedig a nullás tinta adja.
- A grafikus toll színét a 'PLOT', 'PLOTR', 'DRAW','DRAWR' utasítások harmadik paramétere váltja át másik színre, az új szín a következő átváltásig hatásos. Ameddig nem kívánjuk változtatni a grafikus toll színét, a felsorolt utasításokban nem kell harmadik paramétert megadni.
- A grafikus toll színét a CPC 664-en és 6128-on egy külön utasítással is beállíthatjuk: **'GRAPHICS PEN** tinta'.
- A grafikus papír színét a CPC 664-en és 6128-on egyszerű utasítással beállíthatjuk: **'GRAPHICS PAPER** tinta'. A CPC 464-en a grafikus papír színét csak egy rendszerrutin" hívásával lehet beállítani, a paraméterként megadott nullák száma határozza meg a papírhoz rendelendő tinta sorszámát:

  így pl. **CALL &BBE4,** 0,0,0 megfelel a nagyobb CPC-k GRAPHICS PAPER 3 utasításának.

- A grafikus ablakot a **'CLG'** utasítással törölhetjük a papír színére.

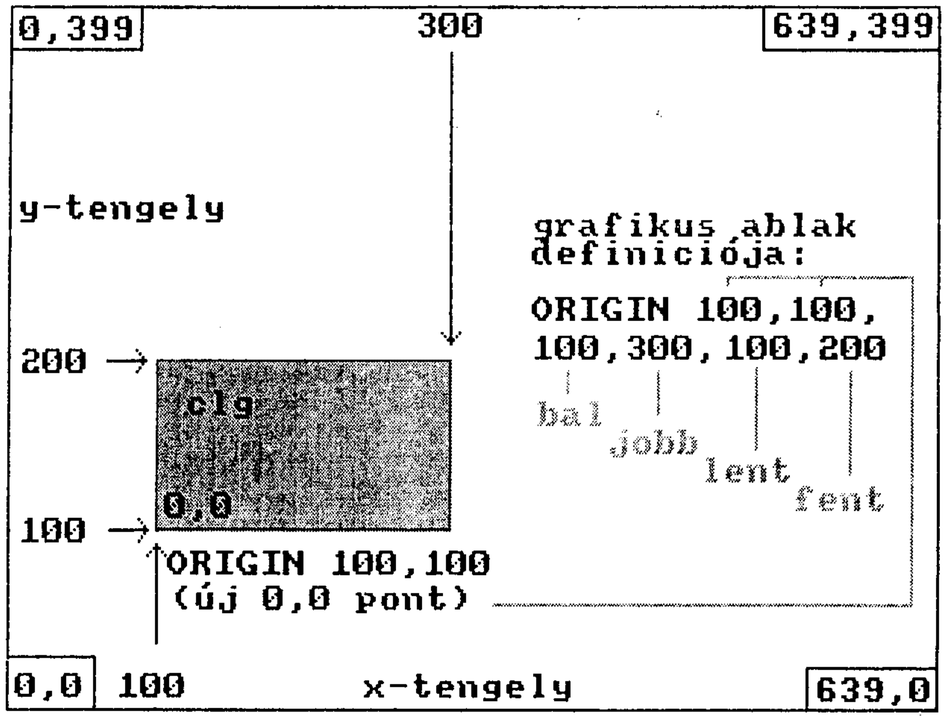
Próbáljuk ki a grafikus ablakot kezelő utasításokat:

```
mode 1          - minden ablakot elfelejteni!
cls             - törli a képernyőt, a szöveges kurzor haza-
Ready             megy a bal felső sarokba
clg             - törli a képernyőt, a szöveges kurzor marad
Ready             a régi pozición
```

'MODE' utasítás után a CPC elfelejtkezik minden ablakról, vagyis az összes ablak, köztük a grafikus is, az egész képernyőre terjed ki, ezért törölte a 'CLG' a teljes képernyőt. Definiáljunk egy új grafikus ablakot:

```
origin 100,100, 100,300, 100,200        - nem látunk semmit
```

Az ablak ott van, de papírjának színe megegyezik a képernyő



színével. Géptípustól függően adja be a következő utasítást:

```
graphics paper 3    - ♦ CPC 664-en és 6128-on: hármas színű papír
call &bbe4,0,0,0    - ♦ CPC 464-en: hármas színű papír
clg                 - most töröld a grafikus ablakot !
```

És megjelenik piros színű ablakunk a képernyő közepén. Egy világos kék pontot az ablak közepébe a 'PLOT 50,50,2' utasítás tesz ki mindhárom géptípuson. CPC 664-en és 6128-on az utasítást felbonthatjuk: 'GRAPHICS PEN 2: PLOT 50,50'.

2.3.6 Karakterek grafikus koordinátákon, animáció

Új kulcsszavak: **TAG**
TAGOFF
FRAME ♦ csak CPC 664 és 6128

Karaktereket eddig az aktuális 'MODE'-tól függő karakterpozíciókon helyeztünk el a 'LOCATE' utasítás segítségével. Mivel

a CPC képernyőjén nem válik külön a szöveges és a grafikus terület, nincs semmi akadálya annak, hogy a karakterek tetszőleges grafikus koordinátapozíción jelenjenek meg. Két utasítással válthatunk át a szöveges kurzorpozíción és a grafikus kurzorpozíción történő karaktermegjelenítés között:

TAG minden további karakter az aktuális grafikus kurzor pozícióján jelenik meg a grafikus toll színében

TAGOFF minden további karakter az aktuális szöveges kurzor pozícióján jelenik meg a szöveges toll színében

```
1 rem SZOVEGEK GRAFIKUS POZICION
10 mode 1                                  - 4 szín elég lesz
20 tag                                     - grafikus pozícióra ír
30 x=rnd*640: y=rnd*400                    - véletlen koordináták
40 move x,y : print "FELIRAT";             - grafikus kurzort el-
50 goto 30                                   helyezi és odaír
60 rem folytatjuk!
```

A "FELIRAT"-ok elárasztják a képernyőt. Megfigyelhető, hogy valóban grafikus pozíciókra kerülnek: egymástól nem teljes karakterekben mérhető távolságra vannak. Figyeljük meg a 40-es sor 'PRINT'-jét lezáró ';' (pontosvesszőt). 'TAG' utasítás után minden 'PRINT'-nél használni kell, mert a 'TAG' megjelenítené a 'kocsivissza' és 'újsor' karaktereket is, amelyeket a BASIC az elválasztó jel nélküli 'PRINT'-ek után kiad. A 'kocsivissza' karakter ASCII kódja 13, az 'újsor' vagy 'soremelés' kódja 10. A két elnevezés azokból az időkből származik, amikor még nyomtatón történt minden kiíratás: nyomtatókocsit (fejet) állítsd vissza a sor elejére és egy sorral továbbítsd a papírt! A 'TAG' a karakterek grafikus képét jeleníti meg, így a vezérlő karakterek is megjelennek, ahelyett, hogy egy cselekvést hajtanának végre!

'TAG' után nem érvényes a 'PRINT chr$(22);chr$(1);', amely az átlátszó módust kapcsolja szöveges üzemmódban. "FELIRAT"-ainak azonban elhelyezhetők egymás hegyén-hátán is, anélkül, hogy a későbbjövő törölné az előzőt. Erre szolgál a chr$(23);chr$(3)-as

vezérlő kód, amely a megjelenő grafikát és az annak a helyén levő hátteret logikai 'vagy'-gyal (OR) kapcsolja össze és az így kapott kombinációt jeleníti meg. A gyakorlatban ez azt jelenti, hogy az új "FELIRAT" nem törli a régit. Bővítsük ki a programot két sorral:

```
12 print chr$(23);chr$(3);              - nem törli a hátteret
32 plot 700,700, int(rnd*3)+1           - a grafikus toll
                                          véletlen színű lesz
```

A képernyőn levő és a megjelenítendő grafikus pontok között a logikai 'vagy'-on kívül lehet még 'és', valamint 'kizáró vagy' kapcsolat is. Ezekről a következő pontban lesz szó. Most inkább nézzük meg, hogyan használhatjuk a 'TAG'-et animáció (életrekeltés), vagyis mozgáseffektusok létrehozására.

```
1 rem MOZGO CSILLAG
10 a$=" *"                      - szóköz csillag
20 tag                          - grafikus pozícióra fogunk írni
30 for x=0 to 640 step 2        - minden képernyőpont
40 move x,200                   - grafikus kurzort az új helyre
50 print a$;                    - 'a$'-t megjelenít az új helyen
60 next x                       - a következő hely jobbra lesz
```

A fenti kis program egy csillagot visz át a képernyő bal oldaláról a jobb oldalra egészen lágy mozgással: az elmozdulás csak egy képernyőpont minden új megjelenítés között. Saját tervezésű karakterekkel tetszőleges rajzfilmet alkothatunk (a gép gyorsaságának korlátain belül), de a CPC beépített karaktereit is felhasználhatjuk:

```
1 rem MOZGO EMBERKEK
10 mode 1
20 tag                                      - grafikus kurzorpoz.
30 plot 0,183: drawr 639,0,1                - alapvonal
40 for i=0 to 3: read alak(i): next i       - négy emberke alak
50 for v=0 to 260 step 2                    - képernyőpontonként
60   move v,200                             - a bal oldali alak
70   print " "; chr$(alak(s));              - eltüntet, kitesz
80   move 620-v, 200                        - a jobb oldali alak
```



```
90   print chr$(alak(s)); " ";              - kitesz, eltüntet
100     s=s+1: if s=4 then s=0              - következő alak (0-3)
110     for ido=1 to 30: next ido           - egy kicsit várunk
120 next v                                  - mehetnek tovább
130 plot 315,210,3: print chr$(228);        - piros szív közéjük
140 data 248,250,249,251                    - chr$() = alakok
```

Grafikus ábrák mozgatásánál szükség lehet az ábra ismételt kirajzolását egybehangolni a képernyőn végigfutó katódsugár útjával. A kép villódzását okozhatja, ha pont akkor viszünk a képernyőre egy ábrát, amikor a katódsugár az adott területen halad át. Előnyösebb megvárni, amíg a katódsugár kirajzolja a teljes képernyőt és visszafut a bal felső sarokba, a visszafutási idő alatt nyugodtan hozzányúlhatunk a képernyőhöz. A CPC 664-en és 6128-on külön BASIC utasítás várakozik a katódsugár visszafutására, a CPC 464-en egy rendszerrutint kell meghívni e célból:

```
FRAME              - csak CPC 664 és 6128!
CALL &BD19         - mindegyik CPC megérti
```

A 'FRAME' vagy a 'CALL &BD19' utasítást tehát a 'PRINT'-ek és 'DRAW'-k előtt célszerű elhelyezni. Fenti kis programunkban a mozgás egyenletesebb lesz, ha a 70-es és 90-es sort átírjuk a következők szerint:

```
70 call &bd19: print " "; chr$(alak(s));
90 call &bd19: print chr$(alak(s)); " ";
```

Mivel 'TAG' után szövegek kiíratásához is a grafikus koordinátákat kell megadni, hasznos segítségül szolgál a következő néhány függvény. A függvények csak 'MODE 1'-ben érvényesek, de könnyű az átszámítás bármelyik 'MODE'-ra. A függvények feltételezik, hogy a gép eredeti koordinátarendszere van érvényben:

```
GX.bal  = (TX - 1) * 16          - a szöveg kezdetének grafikus
                                   x-koordinátája
GY.lent = (25 - TY)* 16          - a szöveg aljának grafikus
                                   y-koordinátája
GY.fent = (25 - TY)* 16 + 15     - a szöveg tetejének grafikus
```

'GX', 'GY' : grafikus koordináták, pl. 'MOVE GX,GY'
'TX', 'TY' : szöveges koordináták, pl. 'LOCATE TX,TY'

Egy átszámitási példa: mi lesz a 'LOCATE 20,10: PRINT "OK"' megfelelője 'TAG' után? A szövegek első karakterének bal felső pozíciójára kell állítani a grafikus kurzort: GX = (20-1) * 16, vagyis 304; GY.fent = (25 - 10) * 16 + 15, vagyis 255. Tehát a 'TAG: MOVE 304,255: PRINT"OK";' parancsra ugyanott jelenik meg az "OK", ahová a 'LOCATE 20,10' tenné. Más 'MODE'-ra való átszámításnál a 'koordinátapont / grafikus pont' arányt kell figyelembe venni. Függőlegesen mindig 2 koordinátapont esik egy grafikus pontra, a vízszintes arány, mint ismeretes, 'MODE'-tól függő. Egy karakter pedig mindig 8*8 grafikus pontból áll.

2.3.7 Színek egymásrahatása

Minden eddig kitett grafikus pontunk és vonalunk felülírta azokat a képernyőpontokat, amelyekre a pontot rátettük, vagy amelyeken a vonal áthaladt. A CPC azonban arra is lehetőséget biztosít, hogy a kitett pont vagy kirajzolt vonal valamiféle kapcsolatba lépjen az alatta elhelyezkedő pontok színével. A színek ilyen egymásrahatása egy bonyolult mechanizmus eredménye, amelyhez nemcsak a logikai műveleteket, de a CPC képernyőjének felépitését is ismernünk kellene, ezért most hosszadalmas magyarázgatások helyett nézzük meg inkább a gyakorlatban, mi történhet akkor, ha egy korábban rajzolt pontot (vanalat) keresztez egy újabb vonal?

A CPC BASIC a keresztezési pont színének kialakítására a vezérlő karakterek használatával ad lehetőséget. A felhasznált vezérlő karakterek a következők:

```
print chr$(23);chr$(0);        - normál mód (felülír)
print chr$(23);chr$(1);        - XOR mód
print chr$(23);chr$(2);        - AND mód
print chr$(23);chr$(3);        - OR  mód
```

Amint megfigyelhetjük, a 'CHR$(23)'-as vezérlő karakter paramétereként megadott 'CHR$(0)-CHR$(3)'-mal kapcsolhatjuk be a különböző keresztezési módokat. A vezérlő karakter aktiválása a 'PRINT' utasitással történik, az utasitást lezáró ';' (pontos-

vessző) arról gondoskodik, hogy ne történjen soremelés.

Mit is jelentenek tulajdonképpen a keresztezési módok? A legegyszerűbben megfogalmazva: a kitett pont vagy vonal színe nem feltétlenül az aktuális rajzoló szín lesz, hanem a keresztezési módnak és a régi pontoknak megfelelően egy bizonyos másik szín is létrejöhet. Lássunk egy drasztikus példát! Hozza gépét alapállapotba: [ESC] [SHIFT] [CTRL], megjelenik az ismert bejelentkezési szöveg, húzzunk rá egy vonalat 'XOR' módban:

```
print chr$(23);chr$(1); : draw 400,400,1
```

A betűk foghijasak lettek: azok a pontjaik, ahol vonalunk keresztezi őket, a papír színére változtak át. Vonalunk tulajdonképpen kifordította az alatta levő pontok színét: ami eddig kék volt, az sárga lett, ami sárga volt, az kékre változott. Ennek az oka az előzőleg kiadott vezérlő karakter, a 'XOR' mód. Minden keresztezési mód addig van érvényben, amig egy újabb fel nem váltja, tehát most is 'XOR' módban vagyunk. Ha a vonalunkra egy újabb vonalat húzunk, annak újra ki kell fordítania az alatta lévő pontok színét, vagyis ...

```
move 0,0: draw 400,400,1                    - visszaállunk 0,0 -ra!
```

Igen, az előző vonal eltűnt, a bejelentkezési felirat betűi újra épek és egészek, mintha mi sem történt volna. Egy kis szépséghibát azért észrevehetünk: amikor a második vonalunk az új parancson haladt át, útközben kifordította a 'w' betű pontjait.

A továbbiakban ismertetett színátváltások négyszínű képernyőmódra érvényesek. Kétszínű üzemmódban a helyzet jóval egyszerűbb, mivel új színek nem keletkezhetnek. Tizenhatszínű üzemmódban viszont egyszerűbb a gyakorlatban kipróbálni, hogy két szín kombinációja mit eredményez egy bizonyos keresztezési mód hatására. Az alább bemutatásra kerülő négyszínű üzemmódban a képernyőmemória minden byte-ja egyszerre négy grafikus pont színét határozza meg, egy-egy pontra tehát két bit jut. Két bittel négy különféle kombinációt fejezhetünk ki: 00, 01, 10, 11. Ezek a kombinációk felelnek meg egy-egy 'PEN' színének. A biteket szoftver, vagyis program helyezi el (állítja be vagy oltja ki),

de az már a hardver (a gép áramköreinek a) dolga, hogy egy bitkombinációhoz milyen palettaszínt rendeltünk hozzá az 'INK' utasítással. A különböző keresztezési módokkal a grafikus pontnak megfelelő bitkombinációt hozzuk logikai 'XOR', 'AND' vagy 'OR' kapcsolatba a kiteendő pont bitkombinációjával. A logikai műveletekről a következő szakaszban lesz szó.

NORMAL keresztezési mód (print chr$(23);chr$(0);)

Ebben az esetben a későbbi pont vagy vonal felülírja a régi pontot, a keresztezési pont színe az aktuális rajzoló szín lesz. A gép bekapcsolásától egy másik keresztezési mód választásáig ez az állapot van érvényben.

XOR keresztezési mód (print chr$(23);chr$(1);)

A keresztezési pont színe bitenkénti 'XOR' (kizáró vagy) művelettel képződik, az eredő színek:

	szín0	szín1	szín2	szín3
szín0	szín0	szín1	szín2	szín3
szín1	szín1	szín0	szín3	szín2
szín2	szín2	szín3	szín0	szín1
szín3	szín3	szín2	szín1	szín0

AND keresztezési mód (print chr$(23);chr$(2);)

A keresztezési pont színe bitenkénti 'AND' (és) művelettel képződik, az eredő színek:

	szín0	szín1	szín2	szín3
szín0	szín0	szín0	szín0	szín0
szín1	szín0	szín1	szín0	szín1
szín2	szín0	szín0	szín2	szín2
szín3	szín0	szín1	szín2	szín2

Figyeljük meg: ha a grafikus papirunk színe 0 (és ez általában így van), akkor bármilyen színnel is rajzolunk rá, 'AND' keresztezési módban nem látunk semmit: az eredő szín mindig 0 lesz, amint az első oszlop és az első sort bizonyítja.

OR keresztezési mód (print chr$(23);chr$(3);)

A keresztezési pont színe bitenkénti 'OR' (vagy) művelettel

képződik, az eredő színek:

	szín0	szín1	szín2	szín3
szín0	szín0	szín1	szín2	szín3
szín1	szín1	szín1	szín3	szín3
szín2	szín2	szín3	szín2	szín3
szín3	szín3	szín3	szín3	szín3

A következő program színuszgörbékkel határolt, félig egymásra tolódó felületeket rajzol. A papír színe 0, a két rajzoló szín: 1 és 2. Futtassa le a programot négyszer úgy, hogy a 'p' változó értékét (keresztezési mód) a négy lehetséges módnak megfelelően állítsa be a 80-as sorban!

```
10 REM ****************************
20 REM *                          *
30 REM * NORMAL/XOR/AND/OR DEMO *
40 REM *                          *
50 REM ****************************
60 MODE 1
70 REM ------------------------------
80 p = 0 ' megprobalni 1,2,3-mal is!
90 REM ------------------------------
100 PRINT CHR$(23);CHR$(p);
110 FOR j = 0 TO 15*EI STEP PI/25
120 x = j*50/PI
130 y1 = 100*SIN(j)
140 y2 = 100*COS(j)
150 PLOT x,150 : DRAWR 0,y1,2
160 PLOT x,150 : DRAWR 0,y2,1
170 NEXT j
```

Megfigyelhetjük, hogy 'XOR' és 'OR' módban egy újabb szín is megjelenik ott, ahol 'NORMAL' módban az egyik szín a másikat felülírná. 'AND' módban pedig nem láttunk semmit, bár a CPC szorgalmasan dolgozott. Futtassa le még egyszer a programot úgy, hogy először azt kilistázza, kitörli a 60-as sort, hogy a lista a képernyőn maradjon, majd a 'p'-t 1-re állítja. RUN! A táblázatok alapján próbálja meg ellenőrizni, hogy helyesen forgatta-e ki a CPC a lista betűinek színeit!

Az előbbi betűátszínezést a gyakorlatban is felhasználhatjuk: ahhoz, hogy egy már megjelenített szöveg betűinek színét kicseréljük, nem szükséges azt újra írni, elég ha egy bizonyos keresztezési módban sűrűn kitett vonalakkal beborítjuk! Bár az 'INK' utasítással is váltogathatjuk a színeket, az elért effektus lényegesen különböző! Ha például 'PEN 3'-mal írunk ki egy szöveget, az 'INK'-csere után is 'PEN 3'-mal kiírt szöveg marad, legfeljebb más színű lesz. A keresztezési módok használatával azonban a 'PEN'-szín változik meg, amint egy vonal keresztülhúzza a szöveget. Lássunk egy példát! A következő programban az 'INK 0,0' és 'INK 3,0' utasításokkal egyformára állítjuk a papír (0) és a hármas toll színét. Ha most a hármas tollal írunk, az természetesen láthatatlan marad. Amennyiben különböző keresztezési módokat választva, vonalakkal borítjuk be az egész képernyőt (180 - 210-es sorok), a színek megváltoznak, az írás megjelenik, majd színt vált, inverzzé válik, végül a papír színe változik meg, esetleg mindez egyszerre történik.

```
10 REM ************************
20 REM *                      *
30 REM * KERESZTEZESI MODOK   *
40 REM *                      *
50 REM ************************
60 DEFINT a-z
70 MODE 1:INK 0,0:INK 1,18:INK 2,6:INK 3,0:BORDER 0
80 PEN 3
90 FOR t=1 TO 24:PRINT STRING$(40,CHR$(64+t));: NEXT t
100 PRINT CHR$(23)CHR$(2);: s=2:GOSUB 170
110 PRINT CHR$(7);
120 PRINT CHR$(23)CHR$(1);: s=2:GOSUB 170
130 PRINT CHR$(7);
140 PRINT CHR$(23)CHR$(3);: s=1:GOSUB 170
150 PRINT CHR$(7);
160 PEN 1: WHILE INKEY$="":WEND:LIST
170 ' vonalak megjelenitik a szoveget
180 FOR v=640 TO 0 STEP -2
190 MOVE v,0: DRAWR 0,400,s
200 NEXT v
210 RETURN
```


A 90-es sorban egy tetszőleges szöveggel írjuk tele a képernyőt, ami egyenlőre még nem jelenik meg. Egy kis idő múlva jobbról balra, nem karakterméretű, hanem grafikus pontnyi nagyságú lépésekben jelenik meg a szöveg egyszerre egy függőleges csíkban. Ha átírjuk a vonalakat kirajzoló szubrutint, tetszés szerint jeleníthetjük meg szövegünket akár háromszög vagy körkörös alakban is! A megjelenítendő színeket a 100, 120, 140-es sorok befolyásolják: itt határozzuk meg a keresztezés módját és a meghúzandó vonal színét ('s').

A keresztezési módokat minden grafikus utasítás figyelembe veszi, így a 'TAG' után 'PRINT'-tel a grafikus kurzorpozíción kiíratott szövegek esetében is érvényben van az előzőleg kiválasztott keresztezési mód.

▶▶▶ CPC 664-en és 6128-on a keresztezési mód választása egyszerűbben is lehetséges: a PLOT, PLOTR, DRAW, DRAWR utasítások negyedik paramétereként megadott szám (0-3) hatása egyenértékű az eddig használt vezérlő karakterek hatásával. Tehát mindhárom gépen lehetséges:

```
print chr$(23);chr$(3);: plot 0,0,1
```

csak 664-en és 6128-on lehetséges: `: plot 0,0,1,3`

2.3.8 Néhány grafikus ötlet

Az alábbi rövid programok a CPC grafikus lehetőségeit hivatottak illusztrálni. Bár mindegyikre jellemző, hogy a szemet gyönyörködteti, nem lesz nehéz gyakorlati felhasználást is találni számukra, például az üzleti grafika területén.

Látványosan ábrázolhatjuk egy időszak termelésének alakulását, ha a termelés mennyiségének megfelelő magasságú oszlopokat egymás mellé állítjuk. Ezt megvalósíthatjuk akár 'x'-ek egymás mellé vagy fölé írásával, de mennyivel meggyőzőbb egy olyan grafikon, amelyen valódi térbeli oszlopok közlik ugyanazt. Az alábbi programmal maximum 18 egységre eső tetszőleges adatokat ábrázolhatunk háromdimenziós formában.

```
10 REM ****************************
11 REM *                          *
20 REM * 3-DIMENZIOS HISZTOGRAM   *
```

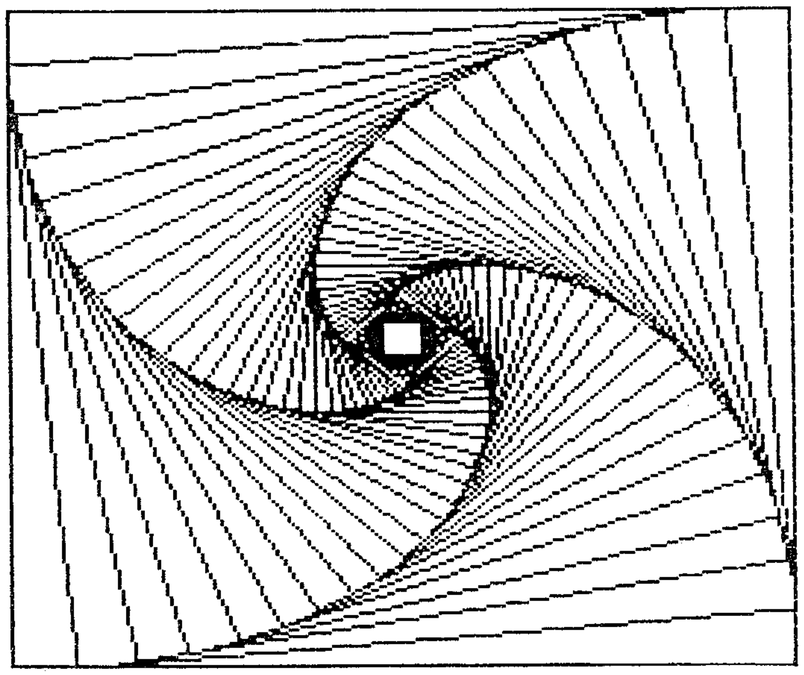
```
22 REM *                          *
30 REM ****************************
40 INK 1,24: INK 2,6: INK 3,9
50 MODE 1
60 FOR m= 350 TO 25 STEP -25
70 PLOT 20,m,3: DRAWR 640,0: NEXT m
80 ORIGIN 0,25
90 DIM adat(20)
100 READ honap$
110 LOCATE 3,25 : PRINT honap$
120 FOR j = 1 TO 18
130 READ v : IF v = -1 THEN 170
140 n = j : adat(j) = v
150 IF v>maximum THEN maximum = v
160 NEXT j
170 FOR q = 1 TO n
180 y = adat(q)/maximum * 300
190 GOSUB 220
200 NEXT q
210 GOTO 210
220 FOR p = 0 TO y STEP 2
230 PLOT 32*q,p
240 DRAWR 30,0,2
250 DRAWR 10,10,3
260 NEXT p
270 FOR p = 1 TO 10 STEP 2
280 PLOT 32*q+p,y+p
290 DRAWR 32,0,1
300 NEXT p
310 RETURN
320 DATA "J F M A M J J A S O N D        1986"
330 DATA 77,155,100,23,56,77,111,99,145,177,100,200
340 DATA -1
```

Az adatokat (max. 18 darab lehet!) a 330-as 'DATA' sorban kell elhelyezni. Az utolsó adatot egy -1 kövesse, ez jelzi a beolvasó huroknak (120 - 160-as sor), hogy nem lesz több adat. A beolvasó hurok a következőket végzi el: feltölti az 'adat()' tömböt, megállapítja az adatok tényleges számát ('n') és nyilvántartja közülük a legnagyobbat ('maximum'), ami az ábrázolás

méretarányának kiszámításához feltétlenül szükséges. Nem marad más hátra, mint kirajzolni a megfelelő méretű oszlopokat, ezt végzi el a 170 - 200-as sorokban levő ciklus. Egy-egy oszlopot a 220-as soron kezdődő szubrutin rajzol ki. A szubrutin két ciklusból áll: az első rajzolja ki az oszlop törzsét szemből és oldalról, a másik rajzolja meg az oszlop tetejét, hogy teljes legyen a térbeliség illúziója. Figyeljük meg a ciklusok kettes lépésközét: 'MODE 1'-ben vagyunk, így gyorsabban megy a kirajzolás és nem veszítünk semmit, hiszen csak minden második koordinátapontnak felel meg egy újabb grafikus pont.

Háromdimenziós ábrákról szólva nem hagyhatjuk ki a térbeli kalapképet létrehozó függvényábrázolást sem. Az alábbi program a térbeliség illúziójához még azzal is hozzájárul, hogy egy-egy pont színét a pont magasságának ('c') megfelelően választja meg. A háromdimenziós kép kirajzolását jelentősen meggyorsítja, hogy a 90-es sorban előre kiszámítjuk és azután konstansként használjuk a nézőpontnak megfelelő tengelyarányokat. A térbeli három tengelynek ('a','b','c') megfelelő kétdimenziós képernyő koordinátákat ('x','y') a 140 - 150-es sorok számítják ki.

```
10 REM ************************
20 REM *                      *
30 REM * HAROMDIMENZIOS KALAP *
40 REM *                      *
50 REM ************************
60 MODE 1 : ORIGIN 320,200
70 INK 1,21: INK 2,24: INK 3,19 :INK 0,0: BORDER 0
80 s=1 :  v=0.5 :  w=0.3
90 sinv=SIN(v): cosv=COS(v): sinw=SIN(w): cosw=COS(w)
100 FOR a=-3 TO 3 STEP 0.2
110 FOR b=-3 TO 3 STEP 0.04
120 c= 5 * ((SIN(a*a+b*b))/(a*a+b*b))
130 IF c>=0  THEN s= 2
140 IF c>=2  THEN s= 3
150 x= cosv * a + sinv * b
160 y= cosw * (cosv * b - sinv * a) + sinw * c
170 PLOT x * 60, y * 40, s
180 s=1
190 NEXT b,a
200 GOTO 200
```



Térbeliként ható ábra keletkezik akkor is, ha például négyzeteket rajzolunk egy középpont köré úgy, hogy mindegyik kisebb lesz, mint az előző és még kissé el is fordul képzeletbeli tengelye körül. Az alábbi programot nem kell kommentálni mivel beszédes változónevei magukat magyarázzák. Próbálja meg változtatni a 'szog'-et és a 'faktor'-t, érdemes!

```
10 REM ************************************
20 REM *                                  *
30 REM * ELFORGATOTT. CSOKKENO NEGYZETEK *
40 REM *                                  *
50 REM ************************************
60 MODE 1:DEG: ORIGIN 320,200
70 sugar=199 : szog=5 : faktor=0.92 : szam=37
80 cs=faktor*COS(szog): sn=faktor*SIN(szog)
90 x=sugar:y=sugar
100 CLS
```



```
110 FOR i%=1 TO szam
120 PLOT x,y,1
130 DRAW -y,x
140 DRAW -x,-y
150 DRAW y,-x
160 DRAW x,y
170 t=x*cs-y*sn:y=x*sn+y*cs:x=t
180 NEXT
```

A tetszőleges oldalszámú sokszögeket rajzol a következő program. Futtatásával meggyőződhetünk róla, hogy egy bizonyos oldalszám után a sokszöget nem tudjuk többé megkülönböztetni egy körtől, ami egyben a gyors körrajzolásnak is a titka. Egyébként a program felkészült a legmegátalkodottabb felhasználóra is: ha valaki tört oldalszámú vagy háromnál kevesebb oldalú sokszöget akar vele rajzoltatni, csak újra indítja a programot.

```
10 '**********************
20 '*                    *
30 '*  SOKSZOG-RAJZOLO   *
40 '*                    *
50 '**********************
60 MODE 1
70 LOCATE 1,1: PRINT CHR$(18);
80 INPUT"Hany oldala legyen";oldalszam
90 IF oldalszam < 3 THEN RUN
100 IF INT(oldalszam) <> oldalszam THEN RUN
110 ORIGIN 320,200
120 PLOT 150,0,3
130 FOR i=1 TO oldalszam
140 x= COS(2*PI*i/oldalszam)*150
150 y= SIN(2*PI*I/oldalszam)*150
160 DRAW x,y
170 NEXT i
180 GOTO 70
```

Animációs effektust nem csak az ábrák valódi, fizikai áthelyezésével érhetünk el. Különösen a tizenhatszínű üzemmódban használhatjuk ki a színek cseréjét életrekeltés céljából: minden mozgásfázist más színnel rajzolunk ki úgy, hogy előzőleg az

összes 'INK'-et a papír színére állítottunk. Abránkat életre-kelthetjük, ha egyszerre egy-egy 'INK'-et állítunk a papírtól eltérő színűre, majd az így megjelenő ábrát egy kis idő múltán újra eltűntetjük: visszaállítjuk papírszínűre. Végighaladva a teljes 'INK'-skálán, képünk valóban megelevenedik.

A következő program 'MODE 1'-ben működik, ahol csak három, a papírtól eltérő színt használhatunk fel az egyes mozgásfázisok ábrázolására. Mégis be kell érnünk ennyivel, mert egy valódi forgó gömb 'MODE 0'-ban eléggé darabos lenne. A program indítása után egy kicsit várni kell, amíg a 80 - 140-es sorok kirajzolják a gömb íveit. A kirajzolás 'i' színnel történik, a 130-as sor biztosítja, hogy minden ív színének sorszáma eggyel nagyobb lesz mint az előzőé volt, de a hármast soha sem éri el: csak 0,1 vagy 2 lehet! Mivel a 20-as sorban minden 'INK'-et 0-ra állítottunk, a körívek kirajzolását természetesen nem látjuk. A 150 - 200-as sorok gondoskodnak arról, hogy a nullás, egyes, kettes tintához hozzárendelődjön a 40-es sorban megadott 'i()' tömbben tárolt színek közül a megfelelő. És mert itt ciklus fut le, a színek egyre cserélődnek. A 210-es sor a gömb végtelen forgásának a záloga, nélküle csak három ívnyit fordulna a gömb.

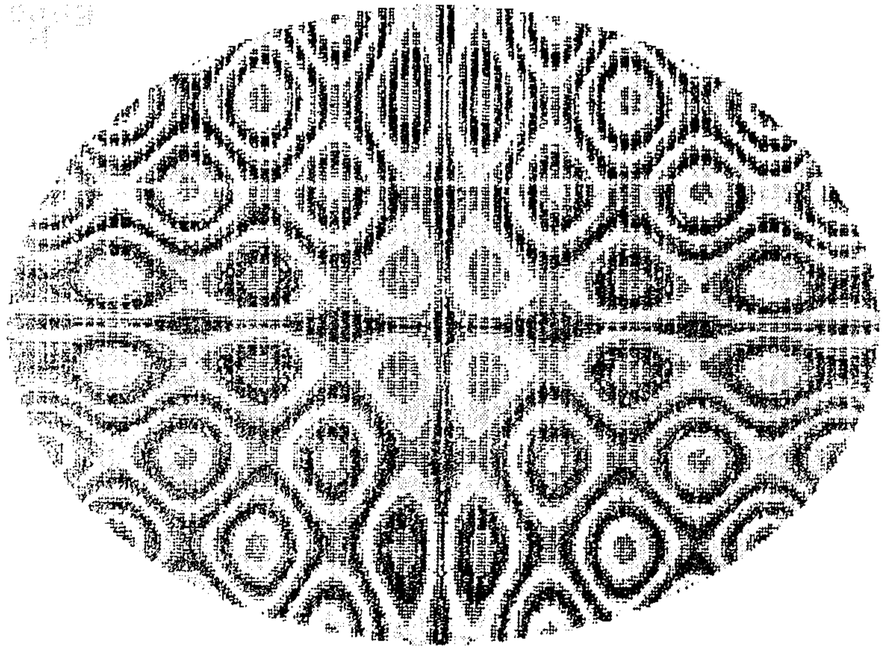
```
1 REM **************
2 REM *            *
3 REM * FORGO GOMB *
4 REM *            *
5 REM **************
10 DEG
20 INK 0,0 : INK 1,0 : INK 2,0 : INK 3,0
30 PAPER 3 : BORDER 0 : CLS
40 i(0) = 14 : i(1) = 6 : i(2) = 5
50 b = 100
60 i = 0
70 ORIGIN 320,200
80 FOR a = -b TO b STEP 5
90 MOVE 0,b
100 FOR t = 0 TO 180 STEP 18
110 DRAW a*SIN(t),b*COS(t),i
120 NEXT t
130 i =(i+1) MOD 3
```



```
140 NEXT a
150 FOR a = 0 TO 2
160 INK 0,i(a)
170 INK 1,i((a+1) MOD 3)
180 INK 2,i((a+2) MOD 3)
190 FOR b = 0 TO 100 : NEXT b
200 NEXT a
210 GOTO 150
```

A CPC grafikájának tökéletességét csak úgy értékelhetjük igazából, ha az egész képernyőt minnél finomabb rajzolatú ábra borítja. Az eddigi példákból leginkább a háromdimenziós kalap közelítette meg ezt a kritériumot. Lássunk most egy olyan programot, amely sűrűn borítja be finoman csipkés mintával az egész képernyőt. Természetesen, az egyszerűség kedvéért, a mintát most is egy vagy több függvény fogja szolgáltatni. Am a kalaptól eltérően, a minta itt nem a vonalak irányát jelenti, hanem a pontok színét: a képernyőn szinte alig marad szabad, papírszínű pont. A program lényegében egyetlen nagy hurokból áll, amely a képernyőre egy nagy tojást rajzol (fekvő csökkenő koncentrikus ellipszisekből). A létrejövő mintát a kitett pontok színének rendszeres változása okozza. Az aktuális színt a 120 - 140-es sorokban számítjuk ki. A kiszámításhoz felhasznált függvény tetszőleges lehet, csak arra kell vigyázni, hogy az általa szolgáltatott végérték ('r') 0 és 3 közötti legyen. Példánkban elég bonyolult módon számítjuk ki az 'r'-t, de ez senkit se riasszon el a próbálkozástól: egyszerűbb függvényekkel is szép mintákat állíthatunk elő.

```
10 REM ******************
20 REM *                *
30 REM * OVALIS MINTAK *
40 REM *                *
50 REM ******************
60 MODE 1: ORIGIN 320,200
70 rx=300: ry=180
80 WHILE rx>1
90 FOR i=0 TO PI/2 STEP 2/rx
100 x=rx*COS(i)
110 y=ry*SIN(i)
```



```
120 r1=SQR((x*SIN(y/14)-10)↑2+(y*SIN(x/14)-10)↑2)
130 r2=SQR((x*SIN(y/14)+10)↑2+(y*SIN(x/14)+10)↑2)
140 r= SGN(COS(r1-r2))+2
150 PLOT x,y,r
160 PLOT x,-y,r
170 PLOT -x,y,r
180 PLOT -x,-y,r
190 NEXT
200 rx=rx-2:ry=ry-2
210 WEND
220 GOTO 220   ' nem kell a Ready !
230 '---- ha tetszik a kep, akkor
240 '---- a 220-as sor helyere
250 '---- ujrainditas elott irja be
260 ' 220 save"!kep-neve",b,&c000,&4000
```

A program rendkívül lassan dolgozik, ha a kialakuló minta nem nyeri el a tetszését, nyugodtan lője ki és próbálkozzék egy másik függvénnyel. Amennyiben úgy látja, hogy megalkotta

élete nagy grafikáját, módosítsa a 220-as sort a program végén megadottak szerint, hogy a kész képet szalagon vagy lemezen megörökíthesse. A '!kep-neve' helyett természetesen tetszőleges nevet adhat be, a felkiáltójel csak akkor szükséges, ha a kimentés kazettára történik: így a CPC nem rontja el üzeneteivel a képet. A kép betöltése a 'MODE 1: LOAD "!kep-neve"' paranccsal történik, a 'MODE 1'-re azért van szükség, hogy a kép valóban a helyére kerüljön, de ez már a Harmadik szint témája.

2.4 EGY KIS LOGIKA

A logikai műveletek a számítástechnika alapját alkotják, így elkerülhetetlen, hogy előbb vagy utóbb ne foglalkozzunk velük. Bár lehet, hogy kezdetben bonyolultnak tűnnek a bináris számok és a velük végzett műveletek, elsajátításuk megéri a fáradtságot: segítségükkel programjaink gyorsabbak, áttekinthetőbbek, logikusabbak lesznek.

2.4.1 Logikai műveletek

Kulcsszavak új értelmezésben:	**AND**
	OR
Új kulcsszavak:	**XOR**
	NOT

Az 'AND' és 'OR' logikai műveletekkel már megismerkedtünk az 'IF-THEN-ELSE' utasításokkal kapcsolatban, bár akkor még nem tekintettük őket logikai műveleteknek. A grafikus pontok keresztezési módjainál csatlakozott hozzájuk a 'XOR' is, de a művelet mibenlétéről ott sem volt szó. Itt az ideje, hogy megismerjük a logikai műveletek mechanizmusát, mivel segítségükkel sok feladatot egyszerűbben lehet megoldani és a továbbiakban más BASIC utasításokkal kapcsolatban is szükség lesz rájuk.

Ha műveletekről van szó, akkor ez azt jelenti, hogy ezeket a kulcsszavakat ugyanúgy használhatjuk, mint a többi BASIC operátort: '*', '+', '-', 'MOD', stb. Próbáljuk ki!

```
print 12 and 5
 4
print 12 or 5
```

```
 13
print 12 xor 5
 9
```

Első látásra nem sok összefüggést fedezhetünk fel a két szám és a műveletek eredményei között. Ez azért van, mert mi a tízes számrendszerben adtuk be a számokat és a CPC is így írta ki az eredményt. A gépen belül azonban minden számábrázolás binárisan történik: a tízes számrendszer csak a felhasználó kényelmét szolgálja, a CPC azonnal binárisra vált át, ha a számokat tárolni kell, vagy ha azokkal műveletet kell végeznie. Ez magyarázza a számításbeli pontatlanságokat is! Ahogy nem lehet például az 1/3-ot pontosan ábrázolni tizedes törttel, úgy nem lehet egyes tizedes törteket pontosan ábrázolni bináris módon, a kerekítések sorozata pedig pontatlanságokhoz vezet. De térjünk vissza a bináris számokhoz: a decimális számok binárissá való átalakítása nem boszorkányság, ám a CPC megcsinálja helyettünk a már megismert 'BIN$' függvény segítségével. Végezzük el újra a fenti műveleteket, de úgy, hogy a számok bináris ábrázolását vetjük egybe. A 'BIN$' második argumentumaként megadott '8' azért szükséges, hogy a megjelenítendő bináris szám elejéről ne maradjanak le a bevezető nullák. A számokat valójában 16 biten tárolja a CPC, de 255-ig elég nyolc is. A logikai műveletek a bináris számok azonos helyiértékű bitjeinek összehasonlítását végzik:

```
?bin$(12,8):?bin$(5,8):?bin$(12 and 5,8)
00001100                                  - 12
00000101                                  -  5
00000100                                  -  4
```

Az 'AND' művelet eredményeképpen kapott bináris számban ott kaptunk '1'-et, ahol mindkét operandusban '1' állt, minden más kombináció '0'-t eredményez. A 'logikai és' műveletet hajtottuk végre.

```
?bin$(12,8):?bin$(5,8):?bin$(12 or 5,8)
00001100                                  - 12
00000101                                  -  5
00001101                                  - 13
```


AZ 'OR' művelet eredményeképpen kapott bináris számban ott kaptunk '1'-et, ahol bármelyik operandusban '1' állt, csak akkor kaptunk '0'-t, ha mindkét operandusban '0' állt azon a helyen. A 'logikai vagy' műveletet hajtottuk végre.

```
?bin$(12,8):?bin$(5,8):?bin$(12 xor 5,8)
00001100                                  - 12
00000101                                  -  5
00001001                                  -  9
```

A 'XOR' művelet eredményeképpen kapott bináris számban csak akkor kaptunk '1'-et, ha az egyik operandusban '1', a másikban pedig '0' állt, vagyis a két operandus adott bitje különbözött. Ha a két operandus adott bitje egyforma (akár '0', akár '1') az eredő bit '0' lesz. A 'logikai kizáró vagy' műveletet hajtottuk végre.

A logikai műveletek szabadon keverhetők a többi művelettel, a műveletek hierarchiájában azonban az utolsó helyet foglalják el: a gép utoljára hajtja őket végre. A végrehajtás sorrendjét természetesen zárójelek használatával tetszőlegesen befolyásolhatjuk.

```
print 12 and 3 + 2                        - vagyis 12 and 5
 4

print (12 and 3) + 2                      - vagyis  0 + 2
 2
```

Lehetséges több logikai műveletet is, zárójellel vagy anélkül, egyszerre kiszámítani:

```
print 25 and 12 or 3 xor 6                - ha nem hiszi,
 13                                         ellenőrizze!
```

A CPC a logikai műveleteket mindig egész típusú számokon végzi. Ha a szám nem egész típusú, amint eddigi példáinkban sem az volt, a BASIC automatikusan egészre kerekíti őket:

```
print 2.6 and 3                          - vagyis 3 and 3
 3
print 2.1 and 3                          - vagyis 2 and 3
 2
```

A kerekítésre azért van szükség, hogy mindkét szám azonos számú bitből álljon: csak így lehet őket bitenként egybevetni. Megfigyelhette, hogy eddig csak pozitív számokat használtunk logikai műveletekben. A jövőben is így teszünk majd, mivel a negatív számokat kissé bonyolultabb binárisan ábrázolni, és nem is lesz rájuk szükség. Az egyetlen negatív szám, amellyel dolgunk lesz, a -1. Nézze meg a bináris képét!

```
print bin$(-1)
1111111111111111                         - 16 darab beállitott bit
```

Hátramaradt az utolsó logikai művelet: a **'NOT'** (nem) csak egy operandust kíván maga után, és annak minden bitjét negálja, vagyis kifordítja: '1'-ből '0' lesz, '0'-ból pedig '1'. Negáljuk a mínusz egyet!

```
print not -1                             - mind a 16 darab '1'-ből
 0                                         '0' lesz, tehát: 0

print not 0                              - ennek minden bitje '0'
-1                                       - mindből '1'-et csinált!
```

Ha más számokkal is kipróbálja a 'NOT'-ot, bizonyára rájön, hogy a kapott eredmény = (operandus * -1) - 1. Egyelőre azonban elegendő, hogy: 'NOT'-tal nullából minusz egyet csinálhatunk és fordítva. A többi logikai művelettől eltérően a 'NOT' előkelő helyet foglal el a műveletek hierarchiájában: rögtön a zárójelek után következik.

A logikai műveleteket igazságtáblázatokba foglalva szokták ábrázolni. Ha a '0'-t HAMIS-nak tekintjük, az '1'-et pedig IGAZ-nak, a velük végzett logikai műveleteket az alábbi táblázat tartalmazza:

		AND	**OR**	**XOR**	**NOT**	
másik		↓ ↓	↓ ↓	↓ ↓	↓	
operandus		0 1	0 1	0 1		
egyik	→ 0	0 0	0 1	0 1	1	eredő
operandus →	1	0 1	1 1	1 0	0	bit

2.4.2 Ha nulla, akkor hamis

A logikai műveleteknek akkor látjuk igazi hasznát, ha az 'IF-THEN-ELSE' szerkezetben megismert relációs műveletekkel kombináljuk őket. A BASIC ugyanis kiértékeli a relációs műveleteket: a HAMIS feltételnek nullát ad értékül, az IGAZ-nak pedig 'NOT 0'-t, vagyis mínusz egyet.

```
print 12 < 1                             - ez HAMIS,
 0                                         tehát 0
print 12 > 1                             - ez viszont IGAZ,
-1                                         tehát -1
```

Ha változók szerepelnek a relációs műveletekben, azoknak az aktuális értékét veti egybe és az eredménynek megfelelően halad tovább a 'THEN' (igaz), vagy az 'ELSE' (hamis) ágon, vagy annak híján továbblép a következő programsorra.

```
a= 2
if a>0 then print "nagyobb mint nulla"
nagyobb mint nulla
```

Kettő nagyobb mint nulla, ez igaz, tehát -1. Végezzük el a CPC helyett a relációs műveletet és helyettesítsük be az eredményt a művelet helyére:

```
if -1 then print "nagyobb mint nulla"
nagyobb mint nulla
```

Hasonló módon parancsmódban is előállíthatunk végtelen hurkot, nem kell programsorba ágyazni például a következő sort:

```
while -1: print a;: a=a+1: wend          - mindig igaz: ismételd!
```

A '-1' helyett tetszőleges számot beírhatunk, a CPC mindent IGAZ-nak tekint, ami nem nulla, bár ő maga a '-1'-et használja az IGAZ jelölésére. Lássunk néhány gyakorlati példát ismereteink hasznosítására! Állapítsuk meg egy karakterről, hogy számjegy-e: ha a karakterünk valóban számjegy, a 'szamjegy' változó értéke legyen '-1', különben legyen '0'! A számjegyek ASCII kódja 48 és 57 között van:

```
szamjegy = (a>=48) and (a<=57)
```

Ha a továbbiakban programunknak aszerint kell elágaznia, hogy számjegyet találtunk vagy sem, az 'IF' után már nem kell összehasonlítgatnuk, hiszen a 'szamjegy' változó értéke IGAZ vagy HAMIS:

```
1  rem SZAMJEGY  -  NEM SZAMJEGY DEMO
10 input a$: a=asc(a$)
20 szamjegy = (a>=48) and (a<=57)
30 if szamjegy then print "szamjegy": goto 10
40 print "nem szamjegy": goto 10
```

Programjainkban gyakran használunk fel változókat zászlóként: csupán egy bizonyos állapot meglétét vagy meg nem létét jelöljük velük, sokszor kell átkapcsolni IGAZ-ról HAMIS-ra és fordítva. Célszerű ilyenkor a CPC által használt értékekhez tartani magunkat: -1 vagy 0. A 'PRINT CHR$(24);'-gyel kapcsolhatunk át inverz írásmódra és vissza. Többszöri átkapcsolás után magunk sem tudjuk már, hogy inverzben vagyunk-e vagy sem. Tároljuk az inverz állapotot az 'inv' változóban: -1, ha éppen inverz a kiírás, és 0, ha normál. A következő két szubrutinnal biztonsággal kapcsolhatunk ide-oda:

```
999  rem MINDIG INVERZRE KAPCSOL
1000 if inv then return                    - már inverz!
1010 print chr$(24);                       - átkapcsol
1020 inv = not inv: return                 - zászlót állít
1999 rem MINDIG NORMALRA KAPCSOL
2000 if not inv then return                - már normál!
2010 print chr$(24);                       - átkapcsol
2020 inv = not inv: return                 - zászlót állít
```


A logikai és relációs műveleteket kombinálva bonyolult programrészleteket helyettesíthetünk egyetlen kifejezéssel. Tegyük fel, hogy a következő problémát kell megoldanunk:

Ha 'a' értéke 100 és 300 között van, 'b' legyen 1.
Ha 'a' értéke 301 és 600 közé esik, 'b' legyen 2.
Ha 'a' értéke nagyobb, mint 600, 'b' legyen 3.
Különben 'b' legyen 0.

Első látásra hihetetlennek tűnhet, de a következő egysoros logikai kifejezés megoldja a problémát. Számoljon utána különböző értéket adva 'a'-nak:

```
b= -(a>=100) * -( (a>=100) + (a>=300) + (a>=600) )
```

Az ilyen kifejezések programsorban nemigen segítik elő a program olvashatóságát, ezért érdemes kipróbálás után felhasználói függvényként definiálni őket, a rájuk való hivatkozás már valóban rövid lesz.

A következő program egy öröknaptár szolgáltatásait nyújtja: azonnal megtudhatjuk belőle, hogy milyen napra esett születésnapunk vagy például milyen napra esnek jövőre az ünnepek.

A program ugyancsak kihasználja a logikai műveletek adta lehetőségeket, hogy működését megértse, a megfejtést kezdje a 90-es soron. Ez a sor számolja ki, hogy a kívánt hónap hány napos. Fel fog tűnni, hogy szökőévről szó sincs, a februárt mindig 31-3 naposnak veszi a program! Ez csak szépséghiba, mert a program a szökőévekre is helyes prognózist ad, erről gondoskodnak a 60-70-es sorokban szereplő törtszámok.

```
10 REM ****************
11 REM *               *
12 REM *  OROKNAPTAR   *
13 REM *               *
14 REM ****************
20 CLS
30 INPUT "Melyik ev ";ev
40 PRINT
50 INPUT "Melyik honap ";honap
```

```
55 IF honap<1 OR honap>12 THEN 50
60 J=INT(365.25*(ev+(honap<3)))+1
70 J=J+INT(30.6*(honap+1-(honap<3)*12))-INT((INT((ev+(honap<3))/
   100)-7)*0.75)
80 J=J-7*INT(J/7)
90 nap=31+(honap=4 OR honap=6 OR honap=9 OR honap=11)+(honap=2)
   *3
100 PRINT:PRINT:PRINT
120 PRINT "vasn hetf kedd szer csut pent szom"
130 PRINT
140 ZONE 5
150 LOCATE 5*J+1,10
160 FOR ev=1 TO nap
170 PRINT ev,
180 IF J+ev-1=6 THEN PRINT:PRINT:J=J-7
190 NEXT ev
200 LOCATE 4,24: INPUT"Ujra = ENTER!", U$: RUN
```

2.5 MULTITASKING BASIC-BEN

A címben szereplő számitástechnikai szakkifejezés a kívülállók kétkedését válthatja ki, ugyanis azt jelenti, hogy a CPC képes egyszerre több feladatot is végrehajtani. És ezt ráadásul BASIC-ben is megteszi, nincs szükség a bonyolult gépi kódú programozáshoz fordulni. A félreértések elkerülése végett tisztázzuk a kezdet kezdetén: minden számítógép, amelyben csak egy CPU* van, egyszerre valójában csak egy dologgal képes foglalkozni, ez alól a CPC sem kivétel. A CPU azonban olyan óriási sebességgel hajtja végre primitiv műveleteit, hogy a közönséges halandónak fel sem tűnik, ha időnként mást is csinál. Ahhoz, hogy rendszeresen megszakitsa a főprogram végrehajtását és más munka után nézzen, egy külső jelre van szüksége. Ez a jel a megszakitás, angolul interrupt. A CPC másodpercenként 300-szor szakitja meg a pillanatnyi program futását, bár ezt eddig észre sem vettük.

Mit csinál a CPC akkor, amikor a megszakitó jel hatására félreteszi aktuális teendőit? Mindenek előtt egy időegységgel előbbre állitja belső óráját, amelyet a gép bekapcsolása indított el és állitott nullára. A belső óra időegysége 1/300 másodperc, tehát minden megszakitásnál állitani kell rajta egyet.

A többi teendő nem ilyen sürgős, ezért ritkábban is hajtja őket végre. Másodpercenként 100-szor nézi meg, kell-e új információt közölni a hanggenerátorral. Másodpercenként 50-szer kérdezi le a billentyűzetet, van-e lenyomott billentyű, ha igen, kell-e ismételni. Ugyanilyen gyakran nézi át az 'INK'-kel meghatározott színeket: nem kell-e valamelyiket egy másikra kicserélni? Ezek a rendszeres rutinfeladatai. Ezek után, mielőtt visszatérne a megszakitott program végrehajtásához, még azt is megnézi, hogy Ön, a felhasználó, nem adott-e neki valami különleges feladatot, amelyet időnként végre kell hajtania. A CPC BASIC lehetőséget nyújt arra, hogy egyszerű módon adjunk a gépnek ilyen időnként végrehajtandó feladatokat.

2.5.1 A CPC belső órája

Új kulcsszó: **TIME**

A CPC belső órája a gép bekapcsolásakor vagy a újraindításakor ([ESC] [SHIFT] [CTRL]) állitódik nullára. A nullázástól eltelt időt a 'TIME' rendszerváltozó tárolja, 1/300 másodperces időegységekben mérve. A változó pillanatnyi állapotát szokásos módon lekérdezhetjük vagy más változónak értékül adhatjuk. Sajnos a belső óra BASIC paranccsal nem nullázható és nem állitható tetszőleges időpontra. Időtartamok mérésénél ezért a következő módon kell eljárni:

```
1 rem PELDA IDOMERESRE
10 kezdet = time
20 ' itt történik az, aminek az időtartamát mérjük
30 ' ..............
999  veg = time
1000 idotartam = veg - kezdet
1010 masodperc = int(idotartam / 300)              - egész rész
1020 tortmasodperc = idotartam - masodperc*300     - maradék
```

A kiszámított 'masodperc'-et természetesen tovább oszthatjuk, ha percekben esetleg órákban van szükségünk az eredményre. A 'tortmasodperc'-et a kivánt pontosságra kerekithetjük a 'PRINT USING' utasitással vagy a 'ROUND' függvény használatával.

A következő program a játékos reflexeit hivatott kiértékelni. A játékos az [ENTER] billentyű lenyomásával tudatja a géppel, hogy készen áll a teszthez. A 100 - 120-as sorokban található véletlen hosszúságú időhúzó ciklus közben a 110-es sor figyeli, hogy nem jött-e be egy idő előtt lenyomott billentyű, mert az csalást jelent. A 130-as sor hívja fel a játékost, arra, hogy nyomja le a szóköz billentyűt, a felhívást támogatja egy sípoló hang 'chr$(7)', és a villogó keret 'border 0,26' is. A kezdő időpontot az 'i1' változóban tároljuk. A program a 170-es sorban bolyong, amíg a játékos a szóköz billentyűt lenyomja. Ekkor az 'i2'-ben megjegyezzük az időpontot. A különbséget a szokásos módon számítjuk ki, bár a tisztesség megkívánná, hogy még egy kicsit levonjunk belőle: a 170-es sor végrehajtásához is idő kell.

```
10 REM **********************
20 REM *                    *
30 REM * REAKCIO IDO MERES *
40 REM *                    *
50 REM **********************
60 MODE 1: BORDER 1
70 LOCATE 8,10 :PRINT "Ha kezdhetjuk: [ENTER] !";
80 INPUT "",a$
90 CLS
100 FOR x = 0 TO 3000*RND
110 IF INKEY$ <> "" THEN CLS : PRINT "CSALTAL !!!" :PRINT:PRINT
    "Veled nem jatszom tobbet." : END
120 NEXT x
130 LOCATE 10,10 : PRINT "Nyomd le a [SZOKOZ]-t !"
140 BORDER 0,26
150 i1 = TIME
160 PRINT CHR$(7)
170 IF INKEY$ <> "" THEN 170
180 i2 = TIME
190 BORDER 1
200 LOCATE 8,13
210 PRINT "Reakcio ido:" ; ROUND ((i2-i1)/ 300,2);"masodperc."
220 FOR x = 0 TO 3000 : NEXT x
230 GOTO 60
```


Próbálja meg futtatni a programot a 170-es sor nélkül! Az eredmény 0.01 másodperc lesz, vagyis ennyi kellett a BASIC-nek, hogy ellenőrizze a szóközbillentyű lenyomását. A korrekt reakcióidőt tehát úgy kapjuk meg, ha a különbségből még 3 időegységet levonunk: i2-i1-3.

2.5.2 Csináljunk időnként mást is!

Új kulcsszavak: **AFTER ... GOSUB**
EVERY ... GOSUB
REMAIN
DI
EI

A CPC lehetőséget ad a felhasználónak, hogy BASIC programjaiba időnként végrehajtandó szubrutinokat építsen be. Minden szubrutint egy időmérőhöz lehet hozzárendelni. Az időmérők sorszáma 0 - 3. Amennyiben nem határozzuk meg az időmérőt, automatikusan a nullást választottuk. Az időmérők 1/50 másodperces időegységekben számlálnak. Egy szubrutin egyszeri meghívását végzi el a következő utasítás, amennyiben az utasítás kiadása után az 'időtartam'-mal megadott idő letelt:

AFTER időtartam [,időmérő] **GOSUB** sorszám

```
10 cls
20 after 50 gosub 100          - egy másodperc múlva gosub 100 !
30 print "*"; : goto 30
100 print "itt voltam": return
```

A 20-as sor végrehajtása egy ideig nem okoz látható eredményt: a CPC vígan írja ki a csillagokat a 30-as sor végtelen ciklusában. Am egy másodperc múlva a végtelen ciklus megszakad, és a sok csillag után a gép tudatja velünk, hogy tényleg ott volt a 100-as soron. A megszakító szubrutint szokásos módon 'RETURN' zárja le, ami a programot visszatéríti csillagjaihoz.

Az 'AFTER GOSUB' utasításnál is érdekesebb az a lehetőség, hogy egy szubrutint ne csak egyszer, hanem rendszeres időközönként hajtassunk végre, az erre szolgáló utasítás szintaxisa

hasonló:

EVERY időtartam [,időmérő] **GOSUB** sorszám

```
10 cls
20 every 50 gosub 100          - másodpercenként gosub 100 !
30 print "*";: goto 30
100 print "itt voltam";: return
```

Kígyózó csillagjainkat most már másodpercenként szakítja meg az "itt voltam" felirat. Ez valódi multitasking: a 30-as sor végtelen ciklusa rendszeresen megszakad, hogy egy másik utasítás kerüljön végrehajtásra! Természetesen a BASIC sebessége nem engedi meg, hogy például 1/50 másodpercenként gyököt vonjunk a főprogram futásának lényeges lelassulása nélkül, de a megszakításos szubrutinokat mégis sok mindenre felhasználhatjuk. Most először figyeljük meg a BASIC megszakítás-kezelésének egy furcsa vonását. Allítsa le a programot az [ESC] billentyű egyszer lenyomásával, majd egy kis idő, mondjuk 10 másodperc leteltével bármely billentyű lenyomásával indítsa újra! A képernyőn először egy sor "itt voltam" jelenik meg, csak azután folytatódik a program az elvárt módon. Az időmérő a program leállása idején tovább számolt, a program futásának újraengedélyezése után pedig először bepótolta a kimaradt szubrutinokat!

A CPC négy időmérője nem egyenrangú: a 0-ás prioritása a legalacsonyobb, a 4-esé a legmagasabb. A magasabb prioritású időmérő megszakíthatja az alacsonyabb prioritásút:

```
10 rem IDOMERO PRIORITAS DEMO
20 cls
30 every 50,0 gosub 70
40 every 50,1 gosub 100
60 goto 60
70 t=t+1: locate 1,1: print t
80 while inkey$="": wend
90 return
100 x=x+1: locate 1,5: print x
110 return
120 rem folytatjuk !
```


A két időmérő egyszerre indul és egy másodpercenként kell kiírniuk az időt a képernyő bal szélére: felül az alacsonyabb prioritású 0-ás, alul a magasabb prioritású 1-es időmérő jelenít meg egy-egy számot. A 0-ás időmérő szubrutinja azonban sok időt vesz igénybe, billentyűre vár, az 1-es tehát megszakítja, kiírja a saját változójának aktuális értékét és visszatér a 0-áshoz. Ott még mindig nem történt billentyűlenyomás, tehát egy másodperc letelte után újra megszakítja az 1-es időmérő ... Az alsó szám másodpercenként növekszik, a felső megakadt az egyesnél. Segítsen neki! Nyomjon le egy billentyűt és tartsa lenyomva, amíg a 0-ás időmérő behozza lemaradását.

Gyakran van szükségünk arra, hogy az időmérők munkájába beleavatkozzunk, - ideiglenesen - letiltsuk őket, majd újra engedélyezzük működésüket. Erre két új BASIC utasítás szolgál:

DI	- letiltja az időmérőket
EI	- újra engedélyezi az időmérőket
RETURN	- szintén újra engedélyezi az időmérőket!

▶▶▶ Gépi kódban is programozók számára megjegyezzük, hogy a két utasítás nem azonos a Z80-as CPU 'DI' és 'EI' utasításaival. A két BASIC utasítás csak a BASIC szintű megszakításokat kezeli.

Bővítsük ki előző programunk 70-es sorát egy 'DI'-vel:

```
70 di: t=t+1: locate 1,1: print t
```

A 'DI' utasítás nem engedélyezi, hogy a nagyobb prioritású 1-es időmérő megszakítsa a billentyűre váró 0-ás időmérő szubrutinját. A program két egyes kiírása után befagy. Várjunk egy kis időt, majd nyomjunk le egy billentyűt! A lenyomva tartott billentyű hatására a program továbblép a 80-as sorra, ahol a 'RETURN' feloldja a 'DI' hatását. Az időmérők gyorsan bepótolják az elvesztegetett időt, ám most is először a nagyobb prioritású 1-es időmérő pörgeti fel az alsó számokat.

A BASIC megszakításokkal kapcsolatos utolsó kulcsszó egy függvény: a **'REMAIN'** azt az időtartamot adja értékül, amely

az argumentummal megadott időmérőn a következő megszakitásig még megmaradt. Egyúttal leállitja a szóban forgó időmérőt! Törölje ki programunk 70-es sorából a 'DI' utasitást, a 80-as sort pedig irja át a következőképpen:

```
80 print remain(0)
```

A 0-ás időmérő most csak egy számot irt ki, máris leállt. A következő megszakitásig még 48/50 másodperce volt hátra. Az 1-es időmérő pedig vigan fut tovább.

A megszakitásos szubrutinok felhasználására sokféle lehetőség nyilik, a legkézenfekvőbb: irjunk egy olyan szubrutint, amely BASIC programok futása közben állandóan megjeleniti a pontos időt a képernyő jobb felső sarkában.

```
1 REM ****************
2 REM *              *
3 REM * SZOFTVER ORA *
4 REM *              *
5 REM ****************
10 WINDOW #1,32,39,1,1 :PEN #1,3
12 WINDOW #0,1,40,2,25 :PEN #0,1
20 PRINT "Ora beallitas.": PRINT
30 INPUT "Adja be az orak szamat:  ",h
40 IF h<0 OR h>23 THEN 20
50 INPUT "Adja be a percek szamat: ",m
60 IF m<0 OR m>59 THEN 20
70 INPUT "[ENTER] = nulla  masodperc!",q$
80 CLS
90 EVERY 50 GOSUB 1000
100 ' tetszoleges program .....................................
105 k=32
110 k=k+1: IF k>255 THEN k=32
120 PRINT CHR$(k);: GOTO 110
999 ' idaig tarthat a program .................................
1000 s = s+1
1010 IF s>59 THEN s = 0 : m = m+1
1020 IF m>59 THEN m = 0 : h = h+1 : PRINT CHR$(7);
1030 IF h>23 THEN h = 0
```



```
1040 IF s MOD 2 = 0 THEN c$ = ":" ELSE c$ = " "
1050 LOCATE #1,1,1
1060 PRINT #1, USING "##!";h;c$;
1070 PRINT #1, USING "##!";m;c$;
1080 PRINT #1, USING "##!";s;
1090 RETURN
```

A program legelőször is ablakot hoz létre az óra számára a legfelső képernyősorban, az általánosan használt nullás ablakot pedig leszűkiti az óra által nem használt területre. Hasznos képernyőnk igy 24 sorra csökken, viszont biztositottuk, hogy az órát egy sima 'PRINT' nem irhatja többé felül, de még a hasznos képernyő felfelé történő görgetése sem mozditja el a pontos időt. A pontos idő beadását a 20 - 70-es sorok fogadják. A 90-es sorral pedig elindul az óra, ettől kezdve az órát megjelenitő szubrutinig tetszőleges BASIC program helyezhető el, példánkban megelégszük a CPC karakterkészletének kiiratásával. Az órát kezelő és megjelenitő szubrutin az 1000-es sorral kezdődik: növeljük a másodpercszámlálót: 's=s+1'. Ha 's' elérte a 60-at, eltelt egy perc, a másodpercszámlálót le kell nullázni és növelni kell a percszámlálót: 'm=m+1'. Hasonló módon történik az óraszámláló karbantartása. A 1040-es sor az órákat, perceket és másodperceket elválasztó karaktert határozza meg: páratlan számú másodpercek esetén ' ' (szóköz), páros számú másodpercek esetén pedig ':' (kettőspont) lesz. A látvány igy egy valódi kvarcórával vetekszik. A kiiratás a 'PRINT USING' segitségével történik, hogy a számjegyek mindig a kivánt helyre kerüljenek.

2.5.3 Kiszállni nem is olyan egyszerű

Új kulcsszavak: **ON BREAK GOSUB**
ON BREAK STOP
ON BREAK CONT ♦ csak 664 és 6128

Az összes megszakitás közül a legmagasabb prioritású a felhasználó által kétszer lenyomott [ESC] billentyű: a program futása megszakad, a képernyőn megjelenik a 'Break in sorszám' felirat és a rendszer parancsmódba tér vissza. Ez azonban nem szükségszerű: az **'ON BREAK GOSUB** sorszám' utasitás hatására a

program futása nem szakad meg, hanem a 'sorszám'-mal kezdődő szubrutinnal folytatódik.

```
10 rem MEGALLITHATATLAN PROGRAM
20 on break gosub 100                    - [ESC]-re gosub 100
30 print "*"; : goto 30
100 return                               - folytasd tovább!
```

A fenti megállíthatatlan programból csak újraindítás árán lehet kiszállni. Ez azonban sajnos nem jelenti azt, hogy megoldódott programunk biztos védelme a kívülállóktól, ugyanis az 'INPUT' utasítást követő várakozás közben a programot le lehet állítani. Az 'ON BREAK GOSUB' utasítás igazi hasznát akkor látjuk, ha a program akaratlan leállítása nagy mennyiségű adatunk elvesztését okozná. Ilyenkor célszerű megkérdezni a felhasználótól, hogy nem akarja-e kimenteni az adatokat.

```
1 rem VELETLENUL LENYOMOTT [ESC] KEZELESE
10 on break gosub 1000
20 rem itt vesszuk be a sok-sok adatot
30 rem --------------------------------
1000 cls
1010 print "Folytatni a programot: F"
1020 print "Kimenteni az adatokat: K"
1030 print "Unom az egeszet, vege: V"
1040 a$=inkey$: if a$="" then 1040
1050 if upper$(a$)="F" then return
1060 if upper$(a$)="V" then end
1070 if upper$(a$)<>"K" then 1040
1080 rem adatok kimentese itt indul
```

Az 'ON BREAK GOSUB' utasítást néha célszerű letiltani, erre szolgál az **'ON BREAK STOP'** utasítás, amely visszaállítja az [ESC] billentyű eredeti funkcióját: a program újra megállítható.

A programvédelmet a CPC 664-en és 6128-on még egy utasítás támogatja: az **'ON BREAK CONT'** utasítás hatástalanítja az [ESC] billentyűt. Az alábbi szerkezettel CPC 464-en is ugyanezt a hatást érjük el:


```
1 on break gosub 2: goto 10              - [ESC]-t hatálytalanít
2 return                                 - ez a két sor
10 rem itt kezdodik a program
```

Végezetül egy ötlet, amely a programfejlesztést könnyiti meg. Programírás közben gyakran kell erőszakkal megszakítanunk a rakoncátlan programot, hogy egy-egy hiba okát a programlista bogarászásával kiderítsük. Ha programunk színeket és 'MODE'-okat váltogat, előfordulhat, hogy az [ESC] billentyűvel olyankor szakítjuk meg a programot, amikor a képernyőméret és a toll-papír színkombináció nem alkalmas listázásra. Három-négy parancsot is be kell ilyenkor adnunk, hogy átváltsunk 'MODE 2'-re és előállítsuk a megfelelő színkombinációt. Az 'ON BREAK GOSUB' ezt feleslegessé teszi:

```
1 rem SEGITSEG PROGRAMFELJESZTESHEZ
2 on break gosub 3:  goto 10
3 mode 2: ink 0,13: ink 1,0: border 13: paper 0: pen 1: list
10 rem itt kezdodhet a valodi program
```

Az [ESC] billentyű kétszeri lenyomása a 3-as sorra tereli programunk futását. Ez a sor elvégzi a szükséges beállításokat és azonnal elkezdi listázni a programot.

2.6 A MUZIKALIS CPC

A CPC beépített nagytudású hanggenerátorral rendelkezik, amellyel a legkülönfélébb hangeffektusokat lehet előállítani, kezdve az egyszerű pittyegésen, egészen a kristálytiszta hangszerszólókig. A hangokat a központi egységbe épített parányi hangszóró szólaltatja meg, amelynek hangminősége még a botfülűek számára is távol áll az ideálistól. Amennyiben Ön komolyabban szándékozik a CPC zenei képességeit hasznosítani, feltétlenül csatlakoztasson a géphez egy erősítőt. A gép hátoldalán 3,5 mm-es sztereó (!) jack-dugó csatlakoztatására szolgáló hüvely található. Az ott megjelenő feszültség elegendő bármely erősítő meghajtásához. Hangszóró vagy fejhallgató közvetlen rákapcsolása a gépet is károsíthatja! A sztereó csatlakozón megjelenő hang valóban sztereó hatású: a hangcsatornák programozásának megfelelően baloldalt, jobboldalt vagy középen szólal meg. Az erősítő

és a CPC összehangolásához a következő kis program szólaltatja meg a nemzetközi A hangot tíz másodpercig a bal csatornán, tíz másodpercig középen, majd tíz másodpercig a jobb csatornán:

```
10 sound 1,142,1000,15                  - bal csatorna
15 while sq(1)>127 : wend               - várj, amíg az szól
20 sound 2,142,1000,15                  - középen szólal meg
25 while sq(2)>127 : wend               - várj, amíg az szól
30 sound 4,142,1000,15                  - jobb csatorna
```

A beépitett hangszóró természetesen csak mono hangot ad ki.

▶▶▶ Azok számára, akik programozták már a HT vagy MSX gépek hanggenerátorát, már itt eláruljuk, hogy a CPC-ben ugyanaz a hanggenerátor található, mint az említett gépekben. A CPC hangparancsai a hanggenerátor rugalmas használatát teszik lehetővé, de akinek kész muzsikája van például a HT gépre, biztosan szívesen kezelné közvetlenül a hanggenerátor regisztereit. Ehhez az Olvasó a Harmadik szinten találhat segédprogramot.

2.6.1 A hangprogramozás alapjai

Új kulcsszó: **SOUND**

A CPC hangelőállítása teljes egészében a megszakításokra épül. Ha On kihagyta az előző szakaszt, most még nem késő, pótolja be az elmulasztottakat, csak akkor lesz érthető a hangcsatornák időzítése.

A hang nem más mint a levegő hullámmozgása. Két egymást követő hullámcsúcs között eltelt idő egy periódus. Minél kisebb egy hang periódusa, annál magasabb hangot hallunk és fordítva: minél nagyobb a periódus, annál mélyebb a hang. A periódus reciprok értéke a frekvencia. Bár a hang frekvenciája gyakrabban használt fogalom, a CPC által előállított hangok magasságát azok periódusával határozhatjuk meg. A hang frekvenciája alapján a periódust a következő képlettel számíthatjuk ki:

periódus = 62500 / frekvencia

A zeneileg képzettek a kívánt oktáv (-2-től +4-ig) és hangjegy (C= 1, C#= 2, D= 3, E= 4 stb.) alapján számíthatják ki egy hang periódusát az alábbi képlettel.

periodus = 62500 /(440 * (2↑(oktáv + (hangjegy-10) /12)))

A legegyszerűbb hang megszólaltatásához a **'SOUND'** utasítás után két paramétert kell megadni: a hangcsatorna számát és a hang periódusát:

```
sound 1,15                  - magas hang    az 'A' csatornán
sound 2,239                 - középső C      a 'B' csatornán
sound 4,800                 - mély hang      a 'C' csatornán
```

A hangcsatorna jelölése binárisan történik. Minden bitnek saját funkciója van, a bitek kombinálhatók. Most csak az első három bit jelentését vizsgáljuk meg:

0. bit (értéke 1) funkciója: 'A' hangcsatorna (baloldalt)
1. bit (értéke 2) funkciója: 'B' hangcsatorna (középen)
2. bit (értéke 4) funkciója: 'C' hangcsatorna (jobboldalt)

Az értékek összeadásával a hang egyszerre több hangcsatornára is kiadható:

```
sound 3,20                  - magas hang 'A' és 'B' csatornán
sound 7,600                 - mély hang mindhárom csatornán
```

Minden hang bizonyos hangerővel megadott ideig szól. Mivel parancsainkban egyiket sem határoztuk meg, a CPC feltételezte, hogy 0.2 másodpercig akarjuk a hangokat hallani, a hangerőt pedig középértékre állította. Természetesen mindkettő szabályozható a 'SOUND' utasítással:

SOUND hangcsatorna, periódus, időtartam, hangerő

Az időtartamot 1/100 másodperces egységekben adhatjuk meg, a hangerő 0 - 15 között lehet. A CPC 664 és 6128 16 fokozatban szabályozza a hangerőt, a CPC 464 azonban csak 8 fokozatban kezeli a hangerő paraméterét: 0 - 7 (ez a maximum). A paramé-

ter 8-as értéke újra 0-t jelent, a 9-es érték 1-et, a 10-es 2-t stb. Mindhárom géptípusra érvényes, hogy a hangerőparaméter 15-ös értéke a legerősebb hangot adja.

Már a jelenlegi ismereteinket felhasználva készíthetünk mini orgonát. Az orgonát a billentyűsor számgombjai szólaltatják meg.

```
10 REM ***************
20 REM *               *
30 REM * MINI ORGONA *
40 REM *               *
50 REM ***************
60 DIM n(10)
70 ON BREAK GOSUB 160
80 SPEED KEY 5,1
90 FOR j = 1 TO 10 : READ n(j) : NEXT j
100 MODE 1
110 LOCATE 10,10: PRINT"  zenelo billentyuk"
120 LOCATE 10,12: PRINT" 1 2 3 4 5 6 7 8 9 0"
130 k$ = INKEY$ : IF k$ = "" THEN 130 ELSE k = ASC(k$)-47
140 IF k>0 AND k<11 THEN SOUND 1,n(k),10,15 ELSE 130
150 GOTO 130
160 SPEED KEY 10,2
170 END
180 DATA 142,239,225,213,201,190,179,169,159,150
```

A 80-as sor felgyorsítja á billentyűzetet. Az utasítást részletesen a következő pontban mutatjuk be. Mivel a program leállítása után ilyen gyors billentyűzeten nehezen lehet bármit is beírni, [ESC] esetén a BREAK-szubrutin a 160-as sorra tereli a programot, ahol visszaáll az eredeti sebesség. A 90-es sor az 'n()' tömbbe olvassa be a középső oktáv hangjait a középső C-től a nemzetközi A hangig. Az utóbbit szólaltatja meg a [0] billentyű, ezért kerül 'n(0)'-ba. Az oktáv, a periódus és a frekvencia összefüggése a Függelékben található táblázatból ellenőrizhető. Hogy a hangok váltakozása gyorsan kövesse a gombokat, csak 10/100 másodpercre állítottuk egy-egy hang hosszát, a hangok maximális (15-ös) hangerővel szólalnak meg.

A 'SOUND' utasítást ciklusban elhelyezve skáláztathatjuk a CPC-t. A hangeffektusok hatását a ciklus kiinduló és végértéke, a lépésköz, valamint a hangok időtartama határozza meg. Az alábbi program rendőrautó szirénáját utánozza:

```
10 rem RENDORAUTO SZIRENA
20 rem
30 for n=100 to 200 step 10                    - oda
40     sound 1, n, 2
50 next n
60 for n=200 to 100 step -10                   - vissza
70     sound 1, n, 2
80 next n
90 goto 30
```

2.6.2 Hangok időzítése

Új kulcsszavak: **RELEASE**
SQ
ON SQ GOSUB

Már a mini orgonán való játék közben is feltünhetett, hogy a CPC emlékszik a hangokra. A következő program ezt egyértelműen be is bizonyítja:

```
10 rem HANGOK IDOZITESE 1
20 sound 1,100,100,15                  - 1 másodpercig szól
30 sound 1,200,100,15                  - 1 másodpercig szól
40 sound 1,300,100,15                  - 1 másodpercig szól
50 sound 1,400,100,15                  - 1 másodpercig szól
60 sound 1,500,100,15                  - 1 másodpercig szól
100 print "keszen vagyok"
110 end
120 rem folytatjuk!

run
Ready                                  - és még mindig szól!
```

A "keszen vagyok" felirat azonnal megjelenik a képernyőn, bár még az első hang sem ért véget! A program lefutott, de

a hangok szép sorjában megszólalnak. Bővitse ki a programot még két hangparanccsal és futtassa le újra:

```
70 sound 1,600,100,15                    - ez a hatodik hang
80 sound 1,700,100,15                    - ez a hetedik hang
```

Most mar nem azonnal, hanem két hang lecsengése után jelenik meg a felirat. 7 - 2 = 5, a CPC tehát öt hangra vonatkozó információt tud tárolni egyszerre. A példa nem mutatta ugyan, de **csatornánként öt hangról** van szó. Ha öt hangnál nem vár több a megszólaltatásra, a program fut tovább. A hangok szinkronizálására, többszólamú zeneművek programozására a CPC-nek gazdag utasitáskészlete van.

A 'SOUND' utasitás első paraméterével a hangcsatorna számán kivül még sok minden beállitható. A paraméter a hangcsatorna státuszát határozza meg. Mint ismeretes, minden bitnek önálló fukciója van:

Bit	Érték	Funkció
0	1	'A' csatorna kiválasztása
1	2	'B' csatorna kiválasztása
2	4	'C' csatorna kiválasztása
3	8	randevú az 'A' csatornával
4	16	randevú a 'B' csatornával
5	32	randevú a 'C' csatornával
6	64	HALT (hangot elraktároz, de nem szól!)
7	128	FLUSH (hangot megszakit és elfelejt)

A bitek tetszés szerint kombinálhatók. Az értékeket össze kell adni, hogy speciális effektusokat érjünk el. Néhány példa:

```
10 rem RANDEVU DEMO
20 sound 1, 400,500              - első hang, 5 mp az 'A' csatornán
25 print "itt vagyok"
30 sound 1+16, 100,200           - kiad 'A'-ra, randevú 'B'-vel
40 sound 2+8, 200,400            - kiad 'B'-re, randevú 'A'-val
```

Az első hang 5 másodpercig szól, mindkét következő hang türelmesen várakozik: randevújuk van egymással. Ha simán adnánk

ki hangot a 'B' csatornára, az azonnal megszólalna, mivel a 'B' csatorna üres. A 40-es sorban kiadott hang azonban a 30-as sor hangjára vár, az pedig csak akkor indulhat, ha az 'A' csatorna éppen szóló hangja véget ér.

```
10 rem FLUSH DEMO
20 sound 1,2000, 32000, 15               - hosszú és csúnya hang
30 print "Ha elég volt: [ENTER]"
40 while inkey$<>chr$(13): wend
50 sound 1+128, 0                        - ez megszabadit tőle
```

A 20-sor hangja több, mint 5 percig is szólna, ha az 50-es sor meg nem szakitaná. Ugye, jobb a csend? Ha a programot [ESC] billentyűvel szakitotta meg, azt nagyon rosszul tette! A hang ugyanis megszakadt, de a gép nem felejtette el. A **'RELEASE'** parancs újra elindítja. A parancsot az engedélyezendő csatorna száma követi: 1, 2, 4 vagy több csatorna esetén ezek összege. Hangok szinkronizálásához tehát a randevún kívül a 'RELEASE' is felhasználható:

```
sound 1+64, 300,200,7                    - miért nem szól?
release 1                                - most szólalt meg!
```

Összefoglalva: a CPC minden csatornán egyszerre öt hang adatait tudja tárolni, öt hang várakozhat a sorban. A csatornák szinkronizálása a randevú-bitekkel lehetséges. Egy hangot úgy is beállithatunk a várakozó sorba, hogy az ne szólaljon meg; ez a HALT-bit segitségével történik. Az ilyen hangot a 'RELEASE' parancs szólaltatja meg.

Ha zeneműveinket programjainkban háttérzörejként kívánjuk használni, a hangokat nemcsak egymással, hanem a program többi részével is szinkronizálni kell. A legegyszerűbb példa, dallam nélkül: kiírunk egy sort, eljátszunk egy hangot, kiírunk egy sort, eljátszunk egy hosszabb hangot és igy tovább, amíg a képernyő meg nem telik.

```
10 rem HANGOK IDOZITESE 2
20 rem ahogy nem lehet csinalni
30 mode 1
```

```
40 for t= 1 to 24
50   print string$(40,rnd*127+127);              <- egy sor jel
60   sound 1, t*10, t*2, 15                       - egy hang hozzá
70 next t
80 rem folytatjuk!
```

A 60-as sor változtatja a hang periódusát és időtartamát. A program lefutott, a zene tovább szól. A szinkronizáció nem mondható sikeresnek. Ahhoz, hogy a programot a zenéhez illeszthessük, tudnunk kell, szól-e még egy hang, mert ha igen, akkor nem megyünk tovább a program többi részével sem. A hangcsatornák pillanatnyi állapotáról tájékoztat az **'SQ'** függvény. A függvény zárójelben megadandó argumentuma a kérdéses csatorna száma: 1, 2, vagy 4. A függvény eredményének minden bitje jelent valamit:

0., 1., 2. bit - a várakozó sorban levő szabad helyek száma
3., 4., 5. bit - a sorban első hang randevú állapota
6. bit - a csatorna HALT állapotban van
7. bit - a csatorna aktiv

A fenti bitek közül igazából az utolsóra van szükség, ez jelzi hogy a csatornán éppen szól egy hang. Ha tehát a függvény eredménye nagyobb, mint 127, akkor a kérdéses csatorna hangot ad ki. Várakozni, amíg egy hang véget ér, a következő programsorral lehet:

```
while sq(csatorna) > 127 : wend
```

Pótolja ki az előző programot a következő sorral és a szinkronizáció tökéletes lesz:

```
65 while sq(1) > 127 : wend                  - várj, amíg szól az 'A'
```

Az előző példánkban sikerült szikronizálni a program futását az aláfestő zenével. Ez az időzítés azonban a főprogram rovására ment: amíg egy hang le nem csengett, megállt a program és csak a következő hang kezdete engedélyezte a folytatást. Az igazi szinkronizáció pedig azt jelenti, hogy a főprogram és az aláfestő zene egymástól függetlenül fut, egyidőben. Mivel

a CPC képes BASIC-ben is megszakításkezelésre és a hangkezelés a megszakításokon alapszik, nem csoda, hogy rendelkezésünkre áll a megfelelő utasítás: **'ON SQ() GOSUB'**. Az utasítást abban a programrészben kell elhelyezni, amely a főprogram magja, vagyis állandóan ismétlődik. Az 'ON SQ() GOSUB' nem tesz mást, minthogy végrehajtásának pillantában lekérdezi az 'SQ' változó értékét: ha szól még a hang, a program a következő utasítással folytatódik, ha azonban szabad a hangcsatorna, a szubrutinra ugrik. A szubrutin természetesen ellátja a szabad hangcsatornát a következő hanggal, majd egy 'RETURN'-nel visszaadja a vezérlést a főprogramnak.

ON SQ(hangcsatorna) GOSUB sorszám

Az alábbi program megvalósitja a kép és a hang maximális szinkronizációját és az előzőtől eltérően még elfogadható zenét is produkál. A zenét később még tökéletesitjük, a program 'DATA' sorait ezért őrizze meg!

```
10 REM ********************
20 REM *                  *
30 REM * ON SQ GOSUB DEMO *
40 REM *                  *
50 REM ********************
60 MODE 1
70 REM ========= fo program ==================
90 x=RND*39+1: y=RND*24+1: LOCATE x,y
100 toll= INT(RND*3)+1
110 ON SQ(1) GOSUB 150                                  ez a lenyeg!
120 PEN toll: PRINT "*"; :GOTO 90
130 REM ======================================
140 REM itt kezdodik az ON SQ GOSUB rutinja
150 READ hang, hossz:  IF hossz=0 THEN RESTORE 190 : RETURN
160 SOUND 1,hang,hossz*15,7
170 RETURN
180 REM ************* katyusa ****************
190 DATA 107, 3, 95, 1, 90, 3, 107, 1, 90, 1, 90, 1, 95, 1, 107,
 1, 95, 2, 142, 2, 95, 3, 90, 1, 80, 3, 95, 1, 80, 1, 80, 1, 90,
 1, 95, 1, 107, 2,  0, 2, 71, 2, 53, 2, 60, 2, 53, 1, 60, 1, 67,
 1, 67, 1, 71, 1, 80, 1, 71, 2, 107, 2, 0, 1, 67, 2
```

```
210 DATA 80, 1, 71, 3, 90, 1, 95, 1, 80, 1, 90, 1, 95, 1, 107, 4
, 0, 0
```

A dallam kódolása a következőket tartalmazza: a hangjegynek megfelelő periódusérték, a hang hossza időegységben. A periódusértékek ellenőrizhetők a Függelékben található táblázatból, a hang hosszát a 160-as sorban álló 'SOUND' utasításban számítjuk ki. Tetszés szerint lehet gyorsítani és lassítani, csak a 15-ös szorzót kell átírni.

Hogyan működik tulajdonképpen az 'ON SQ() GOSUB'? A program magját a 90-120-as sorok által behatárolt végtelen hurok alkotja: véletlen színű csillagokat tesz ki véletlen koordinátákra. A 110-es soron minden csillag megjelenítése előtt keresztülmegy a program, de csak akkor hajtja végre a 'GOSUB'-ot, ha az A csatornán van szabad hely egy újabb hang számára. A zenélő szubrutin lényegében egy ciklusmagot helyettesít: főprogram nélkül, ha nem kellene ügyelni a szinkronizálásra, ciklussal lehetne kiolvasni az adatokat a 'DATA' sorokból. Itt azonban egy hang megszólaltatása után vissza kell térni a csillagokhoz: 'RETURN'. Mivel a 'DATA' sorokból előbb-utóbb elfogynak az adatok, minden beolvasásnál ellenőrizni kell, hogy maradt-e még. A nullával jelzett utolsó adat megszólaltatása helyett egy 'RESTORE'-t hajtunk végre és 'RETURN'-nal távozunk a szubrutinból. A zene így vég nélkül szól. (És a vége egybeolvad az elejével.)

2.6.3 Görbék és borítékok

Új kulcsszavak: **ENV**
ENT

A KATYUSA dallama kissé egyhangúan szólt. Ennek az okát nem kell sokáig keresni: minden megszólaltatott hang tiszta színuszos rezgés volt, egyforma hangerővel és frekvenciával elejétől a végéig. A zenei hang hangereje pedig attól kezdve változik, hogy meghalljuk, egészen addig, amíg el nem hal. A hangerőnek ezt a változását egy szögletes görbeként is elképzelhetjük: lassabban vagy gyorsabban felfut a csúcsértékre, ott egy ideig kitart, majd végül a nullára esik vissza. A szimulálandó hangszertől függ, hogy a három fázis hogyan aránylik egymáshoz: az ütőhangszereknél azonnali a felfutás, amelyet

kitartás nélkül követ egy hosszabb lecsengés. A másik végletet a vonóshangszerek képviselik: a hangerő lassan nő a csúcsértékig, majd egy közbenső szintre süllyed, ott egy ideig kitart, végül lassan elhalkul. Ennyi bevezetés után eláruljuk, hogy a CPC lehetőséget ad tetszőleges hangerőgörbék beállítására, sőt a hangszín is módosítható egy-egy hang megszólaltatása alatt. És mivel mindez nem a beépített hanggenerátor gyári görbéinek felhasználásával történik, tetszés szerinti görbékbe burkolható úgy a hangerő, mint a hangszín.

15 különböző hangerő- és hangszíngörbét lehet definiálni egyszerre. Egy-egy görbe definícióját borítéknak nevezzük. A hangerőborítékot az **'ENV'** utasítással, a hangszínborítékot pedig az **'ENT'** utasítással adjuk át a CPC-nek.

ENV száma, lépésszám, lépésnagyság, időegység, ...
ENT száma, lépésszám, lépésnagyság, időegység, ...

A borítékok 'száma' 1 és 15 között lehet. A következő három paraméter 5-ször ismételhető meg, tehát maximálisan 15 paraméter állhat a boríték 'száma' után. Ha a két utasítást csak a boríték 'száma' követi, további paraméterek nélkül, az előző boríték-definíció érvényét veszti. Az alábbiakban az **'ENV'** utasítás paramétereit ismertetjük:

lépésszám: a hangerő kívánt módosítása hány lépésben történjék? Minnél nagyobb a 'lépésszám', annál finomabb az átmenet a két végpont között. Értéke 0 - 127 között lehet.

lépésnagyság: a hangerő mekkora egységekben módosuljon egy-egy lépésben? Értéke -128 és 127 között lehet, negatív érték csökkenti, pozitív érték növeli a hangerőt. Vigyázat: a CPC az értéket MOD 15 veszi figyelembe, hiszen az abszolút hangerő csak 15-ig terjed!

időegység: egy-egy lépés időtartama 1/100 másodpercben mérve. A hang időtartama = lépésszám * időegység.

```
env 1, 15, 1, 10
```
- 1-es hangerőboríték: tizenöt lépésben fog a hangerő módosulni, minden lépésben egy egységet növekszik a hangerő, minden lépés

10/100 másodpercig tart. A teljes hang igy
15 * 10/100 = 1.5 másodpercig szól.

Definiáltunk egy hangerőborítékot, egyelőre semmi hatása. A hangot most is a 'SOUND' paranccsal kell megszólaltatni. Eddig négy paramétert adtunk meg, az ötödik paraméter: a használandó 'ENV' sorszáma. Mivel a hangerőboríték a nevéhez illően szabályozza a hangerőt, a 'SOUND'-ban nullát adunk meg, hogy a teljes csendből induljon. A boríték meghatározza a hang idejét is, így azt is nullával helyettesítjük.

```
sound 1,100,150,15          - sima hang, 1.5 mp, max. hangerő
sound 1,100,0,0,1           - ugyanaz a periódus, de az 1-es
                              számú 'ENV' borítékot használva
```

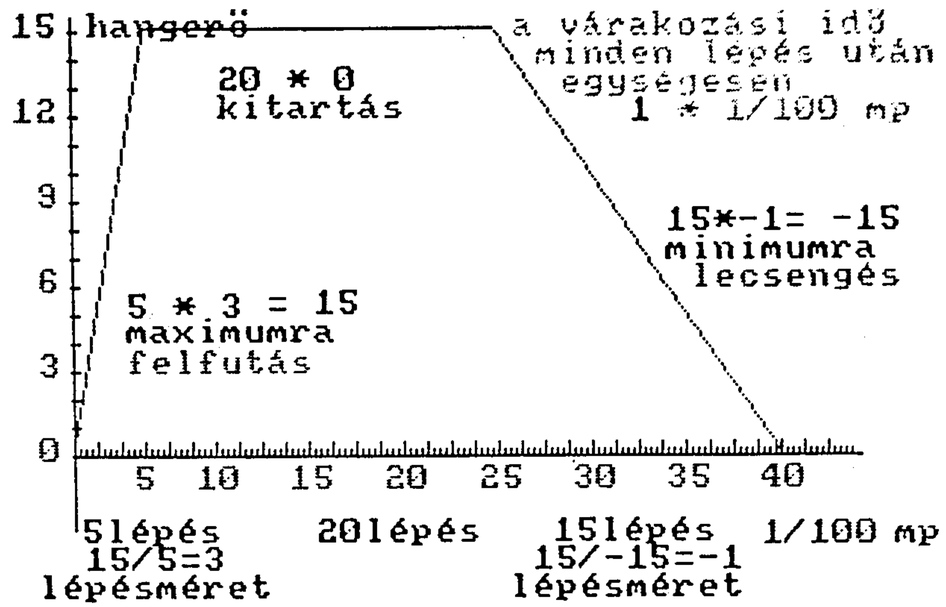
A hangerő fokozatosan emelkedik nulla szintről a maximumig. A borítékot természetesen más 'SOUND' parancsokban is felhasználhatjuk, ha az ötödik paraméterrel a sorszámára hivatkozunk. Készítsük el a boríték fordítottját: csökkenjen a hangerő a maximumról nullára:

```
env 2, 15, -1, 10           - 2. boríték: negatív lépésnagyság
sound 1,100,0,15,2          - 15-ös hangerőről csökkenjen!
```

Hangerőgörbénk eddig mindössze egyetlen szakaszból áll, de a legtöbb valósághű hang előállításához legalább három-négy fázisra van szükség. Az 'ENV' utasításban maximálisan 5 szakasz definiálható, így a legkifinomultabb hangot is lehet szimulálni. A következő boríték három szakaszból áll: a hangerő gyorsan növekszik, hosszan kitart és lassan csökken.

```
env 3, 15,1,1,  1,0,135,  15,-1,8    - 3. boríték: 3 szakasz
sound 1,100,0,0,3                    - használd fel!
```

A hang 15*1 + 1*135 + 15*8 = 270/100 másodpercig szól. A hangerő tizenöt lépésben a felmegy a maximumig, ott kitart 135/100 másodpercig (0 lépésköz = nincs változás), majd újra tizenöt lépésben csökken a minimumra, de ezúttal egy-egy lépés nyolcszor annyi ideig tart, mint a felfutásnál.



ENV 1, 5,3,1, 20,0,1, 15,-1,1

A boríték tervezésénél ügyelni kell arra, hogy a hangerő maximális értéke csak 15 lehet. Amennyiben ezt túllépnénk, az érték/15 maradékára áll be a hangerő, vagyis újra indul nullától. Hosszabb hangnál ez többször is megismétlődhet, ami érdekes effektusokhoz vezet:

```
env 4, 127,1,4                       - 15-ig, újra 15-ig ...
sound 1,100,0,0,4                      127 * 4 / 100 mp-ig
```

A hangerőboríték átveszi a hang időtartamának vezérlését is. Ez viszont azt jelenti, hogy minden hanghosszúsághoz külön borítékot kell tervezni. Még bonyolultabb zeneművekhez is elegendő a CPC 15 hangerőboríték, egyszerűbb esetekben kevesebbel is kijövünk. A következő programban négy borítékot tervezünk meg, mindegyik időtartama az előző kétszerese. A borítékok az időtartamtól eltekintve egységesek: gyors felfutás, lassabb lecsengés.

```
10 REM ************************
```

```
20 REM *                        *
30 REM * NEGY DAL SZTEREOBAN     *
40 REM *                        *
50 REM **************************
60 REM
70 ENV 1,1,14,1,7,-1,2
80 ENV 2,1,14,2,7,-1,4
90 ENV 3,1,14,3,7,-1,6
100 ENV 4,1,14,4,7,-1,8
110 REM
112 REM
120 READ hang, boritek:  IF boritek=9 THEN 190
130 REM
140 SOUND 1, hang*2,  0,          0, boritek
150 SOUND 2, hang,    boritek*15, 15
160 SOUND 4, 239,     0,          0, boritek
170 REM
180 WHILE SQ(2) > 127: WEND:  GOTO 120
190 IF hang = 0 THEN END
200 FOR var=1 TO 400:NEXT:    GOTO 120
210 REM ************** cifra palota **********
220 DATA 71, 2, 71, 2, 71, 1, 90, 1, 107, 2, 90, 1, 90, 1, 80, 1
, 80, 1, 71, 2, 0, 2, 71, 1, 71, 1, 71, 1, 71, 1, 71, 1, 90, 1,
95, 1, 107, 1, 90, 1, 90, 1, 95, 1, 95, 1, 107, 2, 1, 9
230 REM ************* katyusa *****************
240 DATA 107, 3, 95, 1, 90, 3, 107, 1, 90, 1, 90, 1, 95, 1, 107,
 1, 95, 2, 142, 2, 95, 3, 90, 1, 80, 3, 95, 1, 80, 1, 80, 1, 90,
 1, 95, 1, 107, 2,  0, 2, 71, 2, 53, 2, 60, 2, 53, 1, 60, 1, 67,
 1, 67, 1, 71, 1, 80, 1, 71, 2, 107, 2, 0, 1, 67, 2
250 DATA 80, 1, 71, 3, 90, 1, 95, 1, 80, 1, 90, 1, 95, 1, 107, 4
, 1, 9
260 REM ************* ten green bottles ******
270 DATA 120, 2, 120, 2, 120, 1, 95, 3, 107, 1, 120, 1, 107, 1,
95, 1, 120, 2, 0, 2, 95, 2, 95, 2, 95, 1, 80, 3, 90, 1, 95, 1, 9
0, 1, 80, 1, 95, 2, 120, 1, 120, 1, 71, 2, 71, 2, 80, 1, 95, 2,
120, 1, 107, 1, 95, 1, 107, 1, 120, 1, 142, 2
280 DATA 160, 1, 160, 1, 120, 2, 120, 2, 120, 1, 95, 3, 107, 1,
120, 1, 107, 1, 95, 1, 120, 2, 1, 9
290 REM ************* oh, susanna *************
300 DATA 107, 1, 95, 1, 85, 2, 71, 2, 71, 2, 64, 2, 71, 2, 85, 2
```



```
, 107, 2, 95, 2, 85, 2, 85, 2, 95, 2, 107, 2, 95, 2, 0, 2, 0, 2,
 107, 1, 95, 1, 85, 2, 71, 2, 71, 2, 64, 2, 71, 2, 85, 2, 107, 2
, 95, 2, 85, 2, 85, 2, 95, 2, 95, 2, 107, 4, 0, 9
```

A dalok kódolása megegyezik az előző programban leírtakkal, azzal a különbséggel, hogy a hangjegyet itt a hozzá tartozó boritek száma követi. A borítékok a hang hosszát is képviselik, ezért az előző KATYUSA 'DATA' sorát minden további nélkül át lehet venni, csak a dalvég jelölése különbözik. Mivel itt több dal is szerepel, meg kellett különböztetni a dalokat elválasztó dalvégeket az utolsó dal végét jelölőtől: 1,9 = egy dalnak van vége, kis várakozás után jöhet a következő. 0,9 = vége az utolsó dalnak is.

A program mind a három hangcsatornát megszólaltatja. Az A és a C csatorna a hang hosszúságának megfelelő borítékot használja fel. A B csatorna teljes hangerővel szinuszos rezgéssel szól. A hangok szinkronizálása (180-as sor) a B csatornára épül. Próbálja megváltoztatni az 'SQ' argumentumát és azonnal rájön, hogy miért! A hangok úgy eltolódnak, mint egy összehangolatlan vonósnégyes. Bár az A és a C csatorna ugyanazt az 'ENV' borítékot használja, hatásuk különböző: az A csatorna a beolvasott hangnál eggyel mélyebb oktávban szól, a C csatorna állandóan a középső C-t szólaltatja meg. A borítékok alapján működő mindkét csatorna tulajdonképpen alápenget az alaphangot adó, furulyához hasonló hangú C csatornának.

Az **'ENT'** utasítással a megszólaltatott hang frekvenciáját lehet menet közben változtatni. Az utasítás paraméterei közül a 'lépésnagyság' természetesen a hangmagasság relatív változását határozza meg. Pozitív és negatív értékekkel használható: -128 és 127 között. Pozitív értékek csökkentik a hang magasságát, negatív értékek pedig növelik azt. Az **'ENT'** utasítással definiált hangszínborítékra a 'SOUND' utasítás hatodik paraméterével hivatkozhatunk. Amennyiben a 'SOUND' utasításban 'ENV' borítékra nem kívánunk hivatkozni, az ötödik paraméter elhagyható, az elválasztó ',' (vessző) kitétele azonban kötelező.

```
ent 1, 100, 2, 1          - 100 lépésben, 2 egységenként,
                            0.01 mp-es időegységekkel
sound 1,200,100,15,,1     - a hangerő most is 0 - 15 közötti
```

A 'SOUND' parancsban megadott időtartamot nem bírálja felül az 'ENT'. Ha a hangszínboritéк ideje lejárt, de a 'SOUND'-ban megadott időtartam még nem ért véget, a hang tovább szól, állandó periódussal.

```
sound 1,200,200,15,,1          - 1 mp-ig mélyül, azután állandó
```

Az 'ENT' borítékban is öt szakasz határozható meg összesen 15 paraméterrel. Amig a hangerőborítékban általában elegendő két-három szakaszt meghatározni, addig a hangszín vibrálásához minél több szakaszt adunk meg, annál jobb. Vibrátóeffektus érhető el úgy is, ha a rövid hangszíngörbe egy hosszabb hang időtartama alatt állandóan ismétlődik. Az 'ENV' boriték akkor ismétlődik automatikusan, ha a boriték sorszáma előtt mínuszjel áll és legalább két ellentétes fázis paramétereit definiáltuk.

```
ent -2, 4,1,2                  - 2. boriték: rövid hangszíngörbe
sound 1,200,200,15,,2          - nem vibrál
ent -2, 4,1,2, 4,-1,2          - egyszer fel, egyszer le, és újra
sound 1,200,200,15,,2          - most már vibrál
```

Amint a példákból is látható, a 'SOUND' parancsban mínuszjel nélkül kell az ismétlődő 'ENT' borítékra hivatkozni. Vibráló effektussal nemcsak egyes hangszerek hangját lehet imitálni, de egyetlen borítékkal több 'SOUND' utasítás hatása érhető el:

```
ent -3, 20,1,1, 20,-1,1        - ebből sziréna lesz, ha megszólal
sound 1,200,1000,15,,3
```

Hátramaradt a 'SOUND' parancs hetedik paramétere. A hanggenerátor nemcsak zenei jellegű hangok megszólaltatására képes, hanem ún. fehér zaj (szabálytalan hullámforma) előállitására is. A hetedik paraméter a zaj periódusát állitja be, értéke 0 - 31 között lehet. Előtte az elhagyott borítékszámok helyett elég a vesszőt kitenni:

```
sound 1,0,200,15,,,31          - magas zajperiódus = durva zaj
                                 (nulla hangperiódus = nincs hang)
sound 1,200,200,15,,,1         - igy a zenei hang is szól mellette
```


A fejezet utolsó programja egy bomba becsapódásának hangját idézi fel, az utána következő robbanással együtt. A becsapódás hangját egy ciklus állitja elő, bár 'ENT' boriték használatával egyetlen 'SOUND' utasitással is megoldható lenne. A zaj változó periódusa egyre tompább hangot eredményez. A robbanás mind a három csatornán szól, igy erősebb hangot eredményez.

```
1 rem BECSAPODAS ROBBANSSAL
10 for futty = 50 to 150
20     sound 1,futty,3,15
30 next futty
40 for hangero = 7 to 0 step -0.1
50     sound 7,0,6,hangero,,,31-4*hangero
60 next hangero
```

2.7 A BILLENTYOZET ES A BOTKORMANY PROGRAMOZASA

A CPC-k önmagában is rendkivül kényelmes billentyűzete a szokásosnál jóval több komfortot nyújt: az egész billentyűzet szabadon átprogramozható. A karakterkészlet áttervezésével kombinálva lehetőség nyilik igy tetszőleges, akár a magyar szabvány szerinti, billentyűzet kialakitására, de létrehozhatunk például cirillbetűs vagy arabul iró tasztatúrát is. Akik gyakran programoznak gépi kódban, jogosan keveslik a tizes számblokkot és szivesen látnának egy hexadecimális blokkot a billentyűzeten. Ennek sincs akadálya. E szakaszt átolvasva bárki képes lesz egyéni izlésének és szükségletének megfelelően átalakitani a CPC leggyakrabban használt perifériáját, a tasztatúrát.

A billentyűzet ilyen könnyű átalakithatóságának egyszerű a titka: a CPC minden billentyűt annak sorszáma szerint tart nyilván. Ha egy billentyűt lenyomunk, a CPC ezt érzékeli és a billentyű sorszáma alapján egy táblázatból kikeresi a billentyűnek tulajdonitandó kódot. A táblázatot csak át kell irnunk, hogy például a 'z'-ből 'y' legyen. Minden billentyűnek három kód felel meg: egy a magában lenyomott billentyűé, egy másik a [SHIFT]-tel együtt lenyomott billentyűé, a harmadik a [CRTL]-lal együtt lenyomott billentyű kódja. Ezért a billentyűzet dekódolása is három táblázat alapján történik. A felhasználó egyszerű BASIC utasitásokkal bármelyik táblázatot módosithatja.

Más számitógépektől eltérően a CPC a botkormányt is a billentyűzet részének tekinti. Ebből következik, hogy minden olyan utasitás, amely a billentyűzetre vonatkozik, egyaránt érvényes a botkormányra is. Ez az oka annak, hogy a botkormánnyal kapcsolatos tudnivalók is ebbe a szakaszba kerültek.

2.7.1 Figyeljük a billentyűzetet

Új kulcsszavak: **CLEAR INPUT** ♦ csak CPC 664 és 6128
INKEY

A már megismert és gyakran használt 'INKEY$' függvénynek van egy óriási hátránya: nem a lekérdezés pillanatában lenyomott billentyűről ad tájékoztatást, hanem egy pufferből veszi elő az abban található első karakter kódját. Erről könnyen meggyőződhetünk a következő példa segitségével:

```
1 rem BILLENTYUTEZ-PUFFER DEMO
10 for t=1 to 2000: next t                    - időhurok
20 print inkey$                               - mi volt lenyomva?
30 if inkey$<>"" then 30                      - kiürítjük a puffert
```

Amig a 10-es sor időhúzó ciklusa folyik, nyomjon le néhány billentyűt! A 20-as sor a legelőször lenyomott billentyűt adja vissza. A 30-as sorra azért van szükség, hogy a parancsmódba való visszatéréskor a puffer üres legyen, különben a benne levő karakterek megjelennek a képernyőn. Törölje ki a 30-as sort és futtassa újra a programot! A 30-as sor trükkjét gyakran alkalmazzuk, hogy a billentyűzetpuffert kiürítsük és tiszta lappal indulhassunk egy újabb 'INKEY$' vagy 'INPUT' előtt. A CPC 664 és 6128 BASIC-je egy speciális utasitással feleslegessé teszi ezt a trükköt: a **'CLEAR INPUT'** utasitás kiüriti a puffert.

Gyors reakciót követelő programokban (amilyenek a játékprogramok) nehézkes az 'INKEY$' függvény használata. Olyan utasitásra van szükségünk, amely a végrehajtás pillantában vizsgálja meg a kérdéses billentyű állapotát. Erre szolgál az 'INKEY' függvény, amely a lekérdezett billentyű állapotán kívül a [SHIFT] és [CTRL] billyentyű pillanatnyi helyzetét is eredményül adja. Az 'INKEY' függvény argumentuma a kérdéses

billentyű sorszáma. A CPC billentyűi 0-tól 79-ig vannak megszámozva. A billentyűk sorszáma a CPC 664-en és 6128-on könnyen kiolvasható a lemezmeghajtó tetején található ábráról. Könyvünk Függelékében hasonló ábra tájékoztatja a CPC 464 tulajdonosait.

▶▶▶ Bár a CPC 6128-as billentyűzetének elrendezése kissé eltér a két kisebb testvér billentyűzetétől, az azonos funkciójú (feliratú) billentyűk sorszáma mind a három gépen megegyezik. A [COPY] billentyű sorszáma például 9, függetlenül attól, hogy az baloldalt lent, vagy jobboldalt fent található a billentyűzeten.

Az 'INKEY' függvény eredménye bináris formában tükrözi a lekérdezett billentyű, a [SHIFT] és a [CTRL] állapotát:

Érték	[billentyű]	[SHIFT]	[CTRL]
-1	nincs lenyomva	?	?
0	le van nyomva	nincs lenyomva	nincs lenyomva
32	le van nyomva	le van nyomva	nincs lenyomva
128	le van nyomva	nincs lenyomva	le van nyomva
160	le van nyomva	le van nyomva	le van nyomva

A következő program a [COPY] billentyűt figyeli, ennek 9-es a sorszáma:

```
1 rem PELDA INKEY-RE
10 b = inkey (9)
20 if b=-1 then print "SEMMI": goto 10
30 if b=0  then print "[COPY]": goto 10
40 if b=32 then print "[COPY] [SHIFT]": goto 10
50 if b=129 then print"[COPY] [CTRL]" : goto 10
60 if b=160 then print"[COPY] [SHIFT] [CTRL]" : goto 10
70 rem ide sohase johet
```

Az 'INKEY' függvény értéke -1, ha a billentyű nincs lenyomva, ez pedig a logikai IGAZ-nak felel meg. Amennyiben a [SHIFT] és a [CTRL] állapota minket nem érdekel, egyszerrűbb módon is megvizsgálhatunk egy billentyűt:

```
1 rem PELDA EGYSZERU BILLENTYU FIGYELESRE
10 rem a [COPY] billentyu allapotara van szuksegunk
20 '
100 if inkey (9) then print "SEMMI" else print "[COPY]"
110 goto 100
```

A nyílbillentyűk vizsgálatával egyszerű és látványos menüválasztás oldható meg. A felhasználó a [↑] és [↓] billentyűkkel kiválasztja a kívánt menüpontot és a [COPY] billentyűvel jelzi, hogy a választás megtörtént. A 'p' változó tartalmazza a mutató pozícióját. A 90-es és 100-as sor ügyel arra, hogy a felhasználó csak a menüpontok magasságában mozgathassa a mutatót.

```
1 REM ********************************
2 REM *                              *
3 REM * BARATSAGOS MENU VALASZTAS    *
4 REM *                              *
5 REM ********************************
10 CLS
20 LOCATE 11,6 : PRINT "**** VALASSZON ****"
30 RESTORE : READ n
40 FOR j = 1 TO n
50 READ a$
60 LOCATE 10,7+j : PRINT j;a$
70 NEXT j
80 p = 1
90 IF p>n THEN p = 1
100 IF p<1 THEN p = n
110 pp = p
120 LOCATE 9,7+p : PRINT ">"
130 FOR j = 1 TO 200 : NEXT j
140 IF INKEY(9) <>-1 THEN 190      ' copy
150 IF INKEY(0) <>-1 THEN p = p-1 ' fel
160 IF INKEY(2) <>-1 THEN p = p+1 ' le
170 IF p<>pp THEN LOCATE 9,7+pp : PRINT " " : GOTO 90
180 GOTO 140
190 ON p GOSUB 1000,2000,3000,4000,5000
200 GOTO 10
210 DATA 5
220 DATA "Elso menupont"
```



```
230 DATA "Masodik menupont"
240 DATA "Harmadik menupont"
250 DATA "Negyedik menupont"
260 DATA "Otodik menupont"
1000 END
2000 RETURN
3000 RETURN
4000 RETURN
5000 RETURN
```

Az 'INKEY' függvény nélkül elképzelhetetlen egy gyors reakciót igénylő játékprogram, amilyen például az alábbi. A feladat egyszerű: a [←] és [→] billentyűkkel irányítjuk emberkénket a képernyő legfelső sorában jobbra vagy balra úgy, hogy a felfelé igyekvő rakétákat kikerülje.

```
1 REM ********************************
2 REM *                              *
3 REM * RAKETAKAT KERULGETOS JATEK   *
4 REM *                              *
5 REM ********************************
10 DEFINT a-z: DEFREAL i: DIM sor(25) : RANDOMIZE TIME
20 MODE 1: BORDER 0: INK 0,0: INK 1,24: INK 2,20: INK 3,15
30 hely = 20 : t = 100 : idokezd = TIME
40 p = 40*RND+1
50 LOCATE p,25 : PEN 1
60 x = x+1 : IF x>25 THEN x = 1
70 sor(x) = p : v = 0
80 PRINT CHR$(239) : PRINT
90 IF sor((x+3) MOD 25) = hely+1 THEN 170
100 IF INKEY(8) <>-1 AND hely>1 THEN hely = hely-1 : v = 3
110 IF INKEY(1) <>-1 AND hely<38 THEN hely = hely+1 : v = 2
120 LOCATE hely,1 : PEN 2
130 PRINT " ";CHR$(248+v);" ";
140 FOR j = 1 TO t : NEXT j
160 IF sor((x+2) MOD 25) <> hely+1 THEN 40
170 ido = ROUND ((TIME-idokezd)/300,2)
190 LOCATE hely+1,1 : PEN 3: PRINT CHR$(238);
200 FOR j = 1 TO 300: BORDER RND*4: NEXT j
210 IF ido>legjobb OR legjobb = 0 THEN legjobb = ido
```

```
220 CLS : PEN 2
230 LOCATE 10,10 : PRINT "A te idod:";ido
240 LOCATE 10,12 : PRINT "Az eddigi legjobb:";legjobb
250 LOCATE 10,15 : PRINT "Uj jatekot I/N?"
260 valasz$=INKEY$: IF valasz$="" THEN 260
270 IF UPPER$(valasz$)="I" THEN ERASE sor: GOTO 10 ELSE IF UPPER
$(valasz$)<>"N" THEN 260
280 CLS: END
```

A program főbb részei:

10: a sebesség növelése érdekében minden változónk egész típusú lesz, kivéve az 'i'-vel kezdődőket: 'ido', 'idokezd'; 'sor()'-ban tároljuk a felfelé jövő rakáták vízszintes pozícióját
20: beállítja a képernyőt és a színeket
30: 'hely' az emberke vízszintes pozíciója, 't' időhúzó hurok változója (csökkentésével gyorsabb lesz a játék)
40: 'p' a soronkövetkező rakéta vízszintes pozíciója
50: beállítja a kurzort a rakéta kitevéséhez
60: 'x' a soronkövetkező rakéta sorszáma (1 - 25)
70: megjegyzi a pozíciót, 'v' az emberke mozgásának iránya
80: kiteszi a rakétát és kiír egy üres sort: a képernyő felfelé gördül el
90: ha emberkénket rakéta találta el, irány a 170-es sor
100-120: billentyűfigyelés és kiértékelés, 'hely' az emberke pozíciója, 'v' a mozgás iránya
130: kiteszi a mozgás irányának megfelelő alakot, törli az előző emberkét
140: időhúzás: itt könnyen gyorsítható vagy lassítható a játék
160: ha az emberke nem áll rakéta előtt, akkor folytatódik a játék
170: az emberkét találat érte, kiszámítja az eltelt időt
190-200: robbanás, villogó keret
210: ha az eltelt 'ido' nagyobb, mint az eddigi 'legjobb' idő, akkor az lesz a 'legjobb'
200-280: eredmény kiíratás, kell-e új játék, ha igen, akkor törölni kell a 'sor()' tömböt!

2.7.2 Tervezzük át a billentyűzetet!

Új kulcsszavak: **SPEED KEY**
KEY DEF

A CPC egyes billentyűi önismétlő (auto-repeat) üzemmódban működnek: a billentyűt lenyomva tartva bizonyos idő elteltével gyors ütemben jelenik meg a kívánt karakter vagy mozdul el a kurzor. Mint a teljes billentyűzet, az önismétlő üzemmód is átprogramozható, ki- és bekapcsolható, gyorsítható és lassítható. Az önismétlésre beállított billentyűk sebességét módosítja a 'SPEED KEY' utasítás:

SPEED KEY késleltetési idő, ismétlés üteme

A 'késleltetési idő' az első ismétlésig eltelt időt határozza meg, az 'ismétlés üteme' pedig az ismétlések közötti intervallumot állítja be. Mindkét paramétert 0.02 (egy ötvened) másodperces időegységekben kell megadni.

```
speed key 50,1      - egy másodpercig nem ismétel, de azután
                      a lehető leggyorsabb lesz az ismétlés üteme
```

A 'SPEED KEY' utasításnak hasznát láthatjuk akkor, ha gyors billentyűzetre van szükségünk. De vigyázat: egy 'SPEED KEY 1,1' után nem vagyunk többé urai a gépnek, a billentyűzet olyan gyors lesz, hogy értelmes parancsot többé nem tudunk beadni rajta! Ilyenkor csak a teljes újraindítás segít: [ESC] [SHIFT] [CTRL].

A billentyűk gyorsításánál sokkal fontosabb és hasznosabb az átprogramozásuk. A 'SYMBOL' utasítással megtervezhetjük magyar, orosz vagy arab karakterkészletünket, de milyen kényelmetlen a használata, ha a billentyűk máshol vannak, mint ahogy azt egy írógépen megszoktuk. A CPC természetesen itt is segít: minden billentyűhöz tetszőleges karakter rendelhető a 'KEY DEF' utasítással.

KEY DEF billentyűsorszám, ismétlés, normál, [SHIFT]-tel, [CTRL]-lal

A 'billentyűsorszám' megegyezik az 'INKEY'-nél ismertetett sorszámmal, általa határozzuk meg, hogy melyik gombot kívánjuk átprogramozni. Az 'ismétlés' értéke 1, ha azt kívánjuk, hogy a gomb önismétlő legyen, vagy 0, ha nincs szükségünk önismétlésre. A lenyomott gomb által szolgáltatandó ASCII kódot határozza meg a 'normál' elnevezésű paraméter, az önmagában lenyomott billentyű adja majd ezt az értéket. A két következő paraméter azt határozza meg, hogy milyen értéket szolgáltasson a billentyű, ha azzal egyidőben a [SHIFT] vagy a [CTRL] gombot is lenyomjuk. A utolsó paraméterek elhagyhatók. Ha például a nagy [ENTER] billentyűt kívánjuk önismétlődővé tenni, elég a következő parancsot beadni:

```
key def 18,1                    - [ENTER] sorszáma 18
```

Próbaképpen cseréljük ki a 'Z' és az 'Y' billentyűt! Most a helyzet még a következő:

```
[Z] billentyű sorszáma: 71,  kis 'z' ASCII kódja 122
                             nagy 'Z' ASCII kódja 90
[Y] billentyű sorszáma: 43,  kis 'y' ASCII kódja 121
                             nagy 'Y' ASCII kódja 89
```

A billentyűcseréhez adjuk be a következő két parancsot:

```
key def 71,1,121,89             - 71: önismétlő, y, Y
key def 43,1,122,90             - 43: önismétlő, z, Z
```

Figyeljük meg, hogy az utolsó paramétert lehagytuk: így a billentyűk által szolgáltatott kód a [CTRL] gomb egyidejű lenyomása mellett a régi marad! BASIC szinten ez többnyire nem lényeges, de egyes felhasználói programoknál zavart okozhat, ha [CTRL] [Z] helyett [CTRL] [Y]-t kell lenyomnunk.

▶▶▶ A 'KEY DEF' utolsó három paramétere 0 - 255 közötti értéket fogad el. Nem tanácsos azonban 127-nél nagyobb értéket beadni, mert a BASIC interpreter azokat sajátos módon értelmezi. Egyes grafikus karakterek az elvárt módon megjelennek, mások azonban a képernyőszerkesztő funkcióit hajtják végre (224, 225, 240-254), vagy funkcióbillentyűket

jelölnek ki (128-159). A 255-ös kódot a rendszer figyelmen kívül hagyja, ezért a 'KEY DEF 66,0,255,255,255' utasítás hatálytalanítja az [ESC] billentyűt.

Használjuk fel új ismereteinket és készítsünk egy olyan programot, amely megtervezi az ékezetes magyar betűket és egyúttal el is helyezi őket a magyar írógépen megszokott billentyűkre. A program több követelménynek tesz eleget, ezért sajnos kompromisszumokra is kényszerül. A követelmények és a velük járó kompromisszumok a következők:

- A magyar betűket ki is lehessen nyomtatni. Ha a nyomtató karakterkészlete áttervezhető, vagy már rendelkezik magyar betűkkel, ez a követelmény kielégíthető. A CPC azonban csak 0 - 127 közötti kódokat képes a nyomtatóra küldeni, azért a magyar betűket is ebben a tartományban kell elhelyezni.

- A magyar betűk helye a legmesszebbmenőkig feleljen meg a szabványos írógépen elfoglalt helyüknek. Ekkor azonban le kell mondanunk a [CLR] billentyűről, hiszen az írógépen ott a hosszú [ó] található, valamint egyes írásjeleket is át kell helyezni, hogy a helyükre magyar betűket tehessünk.

- A magyar betűk legyenek átlátszóak a CPC BASIC (esetleg más programnyelv) számára. Ez azt jelenti, hogy a rendszer ne is tudjon arról, hogy egyes jeleit mi másra használjuk. A magyar karakterkészlet és billentyűzet, amennyire lehet, ne nehezítse a programírást.

Mindent mérlegelve, a többi nemzeti ábécéhez hasonlóan, mi is a ritkábban használt jelek (pl. [](){|'↑~ stb.) helyére, azok kódszámát felhasználva helyezzük el a magyar betűket. Mivel ékezetes betűből több van, mint a könnyen elhagyható jelekből, le kell mondanunk a hosszú í-ről (ez sok írógépen szintén hiányzik), és a nagybetűkből is csak az Á-t és az É-t engedhetjük meg magunknak. Miért pont ezt a kettőt? E két nagybetű alakilag is eltér a megfelelő kisbetűtől, amíg a többiek tulajdonképpen felnagyított kisbetűk. A betűk elhelyezése könnyen megoldható hála a CPC temérdek billentyűjének. Azokat a karaktereket, amelyeket a magyar betűk kiszorítanak, tehetnénk a jobb oldali

tizes blokkra is, de akkor lemondanánk annak előnyeiről. Ezért a kiszoruló karaktereket inkább a felső számsorra tesszük: azokat a billentyűket csak normál és shift-módusban használjuk a ctrl-módus igy a rendelkezésünkre áll. Ennyi bevezető után lássuk a programot!

```
10 REM ***************************
20 REM *                         *
30 REM * MAGYAR KARAKTERKESZLET *
40 REM *                         *
50 REM ***************************
60 SYMBOL AFTER 32
70 REM -----------------------------------------
80 FOR terv = 1 TO 10
90 READ karnum,t1,t2,t3,t4,t5,t6,t7,t8
100 SYMBOL karnum,t1,t2,t3,t4,t5,t6,t7,t8
110 NEXT terv
120 REM -----------------------------------------
130 DATA &5b, &36, &00, &3c, &66, &66, &66, &3c, &0
140 DATA &5c, &36, &00, &66, &66, &66, &66, &3e, &0
150 DATA &5d, &0c, &18, &3c, &66, &66, &66, &3c, &0
160 DATA &5e, &33, &66, &3c, &66, &66, &66, &3c, &0
170 DATA &5f, &0c, &18, &66, &66, &66, &66, &3e, &0
180 DATA &60, &0c, &18, &3c, &66, &7e, &60, &3e, &0
190 DATA &7b, &0c, &7e, &6c, &60, &78, &60, &7e, &0
200 DATA &7c, &18, &30, &78, &0c, &7c, &cc, &76, &0
210 DATA &7d, &06, &1e, &2c, &66, &7e, &66, &66, &0
220 DATA &7e, &33, &66, &00, &66, &66, &66, &3e, &0
230 REM -----------------------------------------
400 REM ************************
410 REM *                      *
420 REM * MAGYAR BILLENTYUZET *
430 REM *                      *
440 REM ************************
450 ' ---------------------------------
460 ' felso sor [-]-szal kezdve
470 KEY DEF &19,1,&5B,&5B,&1B 'kis oe
480 KEY DEF &18,1,&5C,&5C,&1C 'kis ue
490 KEY DEF &10,1,&5D,&5D,&1D 'kis oo
500 ' ---------------------------------
```



```
510 ' eggyel lejjebb [@]-cal kezdve
520 KEY DEF &1A,1,&5E,&5E,&1E 'kis oeoe
530 KEY DEF &11,1,&5F,&5F,&1F 'kis uu
540 ' ---------------------------------
550 ' eggyel lejjebb [*]-gal kezdve
560 KEY DEF &1D,1,&60,&7B,&1B 'kis es nagy ee
570 KEY DEF &1C,1,&7C,&7D,&1C 'kis es nagy aa
580 KEY DEF &13,1,&7E,&7E,&1E 'kis ueue
590 ' ---------------------------------
600 ' y - z csere
610 KEY DEF &2B,1,&7A,&5A,&1A 'z
620 KEY DEF &47,1,&79,&59,&19 'y
630 ' ---------------------------------
640 ' hianyzo jeleket 1,2,3,4 + CTRL-ra
650 KEY DEF &40,1,&31,&21,&3D '[1]-re =
660 KEY DEF &41,1,&32,&22,&2D '[2]-re -
670 KEY DEF &39,1,&33,&23,&2A '[3]-ra *
680 KEY DEF &38,1,&34,&24,&3A '[4]-re :
690 ' ---------------------------------
700 ' [\]-ra  ; es +
710 KEY DEF &16,1,&3B,&2B     '; es +
720 ' ---------------------------------
730 ' [0]-ra  @ (kukac)
740 KEY DEF &20,1,&30,&40 ' 0 es kukac
```

A programban nem véletlenül használtuk a hexadecimális számokat: ez megkönnyíti a program adaptálását, átírását gépi kódra vagy más programnyelvre. Ez az oka annak is, hogy nem hagytuk el a billentyűk [CTRL]-lal együtt szolgáltatott kódját. A program egyszeri futás után kitörölhető: 'NEW', többé nincs rá szükség. A magyar billentyűzet és a karakterkészlet a gép újraindításáig érvényben marad. Az új billentyűzeten való eligazodást segíti elő a következő táblázat:

CPC magyar billentyűzet

(kontroll karakterek hexadecimálisan)

ctrl	=	-	*	:										DEL
shift	!	"	#	$	%	&	'	(	)	@	ö	ü	ó	DEL
normál	1	2	3	4	5	6	7	8	9	0	ö	ü	ó	DEL
ctrl		11	17	05	12	14	1A	15	09	0F	10	1B	1C	ENTER
shift		Q	W	E	R	T	Z	U	I	O	P	ő	ú	ENTER
normál		q	w	e	r	t	z	u	i	o	p	ő	ú	ENTER
														ENTER
ctrl		01	13	04	06	07	08	0A	0B	0C	1D	1E	1F	ENTER
shift		A	S	D	F	G	H	J	K	L	É	Á	Ű	ENTER
normál		a	s	d	f	g	h	j	k	l	é	á	ű	ENTER
ctrl		19	18	03	16	02	0E	0D						SHIFT
shift		Y	X	C	V	B	N	M	<	>	?	+		SHIFT
normál		y	x	c	v	b	n	m	,	.	/	;		SHIFT

Az alábbi táblázat az elveszett ASCII karaktereket és a helyettük megjelenő magyar betűket veti egybe. Az új ékezetes betűk ASCII kódjára szükségünk lehet például az 'INKEY$' függvénnyel vagy a relációs operátorokkal (<=>) kapcsolatban. Bánjunk óvatosan a magyar betűkkel: csak szövegekben, szöveges típusú változókban fordulhatnak elő. Semmiképpen ne használjuk őket változónevekben, az 'á'-t kerüljük még 'REM' után is, és 'DATA' sorokban tegyük idézőjelbe az ékezetes betűket is tartalmazó szövegeket!

ASCII karakter	Kód dec.	Kód hex.	Magyar karakter
[	91	5B	ö
\	92	5C	ü
]	93	5D	ó
↑	94	5E	ő
_	95	5F	ú
{	123	7B	É
}	125	7D	Á
`	96	60	é
¦	124	7C	á
~	126	7E	ű

2.7.3 A funkcióbillentyűk programozása

Új kulcsszó: **KEY**

A funkcióbillentyűk nem mások, mint tetszőleges szöveggel feltölthető billentyűk. A kívánt szöveg betűről-betűre való beadása helyett elég a megfelelő funkcióbillentyűt lenyomni és a szöveget máris elhelyeztük a billentyűzetpufferben, ahonnan az általában a képernyőre kerül. A CPC 664 és 6128 tízes számblokkján a számjegyek előtt álló 'f'-betű utal arra, hogy ezek a billentyűk funkcióbillentyűként használhatók. Ez azonban így félrevezető! Az 'f'-betű csupán azt jelzi, hogy ezek a gombok már a gép bekapcsolásakor funkcióbillentyűként jelennek meg. De mivel a CPC billentyűzete szabadon átprogramozható, gyakorlatilag a 80 gomb bármelyike lehet funkcióbillentyű. Összesen 32 funkcióbillentyűt jelölhetünk ki egyidejűleg. A funkcióbillentyűk kódszáma 128-tól 159-ig terjed, ezt a számot kell megadni a 'KEY DEF' utasítás második, harmadik vagy negyedik paramétereként. A gép bekapcsolásakor a tízes számblokk a következő értékeket kapja:

Billentyűsorszám	Funkció kódszám	Hozzárendelt karakter(lánc)
15	128	0
13	129	1
14	130	2
5	131	3
20	132	4
12	133	5
4	134	6
10	135	7
11	136	8
3	137	9
7	138	.
6	139	[ENTER]
6	140	RUN" [ENTER]

Az utolsó, 140-es kódszámú funkció a [CTRL] + kis [ENTER] együttes lenyomásával érhető el. Amint látjuk, a számblokk számjegyfeliratú billentyűihez egykarakter hosszúságú szöveg

rendelődik hozzá a gép bekapcsolásakor. Változtassunk ezen! BASIC-ben a következő utasítással adhatunk értékül szöveget egy funkcióbillentyűnek:

KEY funkciókódszám, szöveg

A 'funkciókódszám' 128 - 159 között legyen, a 'szöveg' teszőleges karakterlánc lehet, vezérlő karaktereket a 'CHR$' függvény segítségével ágyazhatunk bele és természetesen BASIC kulcsszavakat is tartalmazhat. Rendeljük például a számblokk [.] billentyűjéhez a "cls" szöveget:

```
key 138,"cls"                    - funkciókódszáma 138
```

A [.] billentyűt lenyomva megjelenik a 'cls', mintha betűről betűre adtuk volna be. Ha utána az egyik [ENTER] gombot is lenyomjuk, a CPC azonnal végrehajtja a kapott parancsot. Mindez azonban egyszerűbben is megoldható: ágyazzuk be az [ENTER]-t is a 'szöveg'-be!

```
key 138,"cls"+chr$(13)          - chr$(13) = [ENTER] !
```

Most már elegendő lenyomni a [.] billentyűt, hogy kitisztuljon a képernyő. Hosszabb szöveget is értékül adhatunk a funkcióbillentyűknek. Az egyetlen megkötés, hogy a szövegek együttes hossza nem haladhatja meg a 120 karaktert.

```
key 138,"mode 2: ink 0,13: ink 1,0: border 13: pen 1" + chr(13)
```

[.] gombunk így nemcsak képernyőt tisztít, hanem a megfelelő színkombinációt és a képernyő-üzemmódot is beállítja. És mindezt egyetlen gombnyomásra! Ne feledjük, 32 ilyen csodagombot programozhatunk be!

A gép bekapcsolásakor csak a 128 - 140-es funkciókódok rendelődnek hozzá a tízes számblokk gombjaihoz. A 141 - 159-es funkciókódok még szabadok, tetszés szerint felhasználhatjuk őket.

Lehetőségünk van így megőrizni a tízes számblokk előnyeit, és bármely más gombot, gomb + [SHIFT] vagy [CTRL] kombinációt funkciókóddal ellátni. Felhasználva a [CTRL] + [betűgomb] kombinációt, könnyen megjegyezhető funkcióbillentyűket hozhatunk létre.

```
10 rem PELDA FUNKCIOBILLENTYUZETRE
20 rem
30 data 50, 1, 114, 82, 141          - 50-es gomb: r, R, 141
40 data 62, 1,  99, 67, 142          - 62-es gomb: c, C, 142
50 data 36, 1, 108, 76, 143          - 36-os gomb: l, L, 143
60 data 60, 1, 115, 83, 144          - 60-as gomb: s, S, 144
65 data 58, 1, 101, 69, 145          - 58-as gomb: e, E, 145
70 rem es igy tovabb .............
80 for ennyivan= 1 to 5              - csak 5-öt definiálunk
90 read sorszam, ism, normal, shift, ctrl
100 key def sorszam, ism, normal, shift, ctrl
110 next ennyivan
120 rem
130 rem a 141-145-es funkciokodokat elhelyeztuk
140 rem  most fel kell tolteni oket szoveggel !
150 rem
160 data "run",    "return"
170 data "cls",    "return"
180 data "load ",  "idezojel"
190 data "save ",  "idezojel"
195 data "edit ",  "semmi"
200 rem es igy tovabb ...........
210 for ennyivan = 1 to 5
220 read fnkcio$, vege$
230 if vege$="return"   then vege$=chr$(13)
240 if vege$="idezojel" then vege$=chr$(34)
250 if vege$="semmi"    then vege$=""
260 key 140+ennyivan, funkcio$+vege$
270 next ennyivan
```

A programból könnyen látható, hogy mely billentyűkre milyen szöveget helyeztünk: [CTRL] [R] "run"+chr$(13)-at eredményez, [CTRL] [C] "cls"+chr$(13)-at ad stb. A szövegek végét lezáró 'chr$(13)' és 'chr$(34)' beadásához egy kis trükkhöz kellett

folyamodnunk, mivel ezek a karakterek nem helyezhetők el 'DATA' sorban. A program tetszés szerint bővithető. Hosszabb programok begépelésekor kifizetődő a gyakran használt szövegeket funkció-billentyűre tenni.

2.7.4 A botkormány és programozása

Új kulcsszó: **JOY**

A CPC két botkormány használatát teszi lehetővé, azonban csak egy csatlakozó található a gépen. A csatlakozó a szokásos 9 pólusú ATARI-típusú, ezért gyakorlatilag a legtöbb botkormány használható a géppel. Ha egyszerre kettőt is használni kívánunk, akkor az eredeti AMSTRAD/SCHNEIDER botkormányt (típusszáma: JY2) kell beszereznünk: ezen egy újabb csatlakozó található, amely a második botkormányt (típusszáma JY1) szállásolja el.

▶▶▶ Ne vásároljon olyan botkormányt, amelyen a tüzelő gomb önismétlését (auto fire) nem lehet kikapcsolni! Ezek a botkormányok a gépből kapnak tápfeszültséget, a CPC-n pedig ez a tápfeszültség nem áll rendelkezésre. Az ilyen botkormány tüzelő gombja a CPC-n hatástalan marad.

A botkormányt a CPC a billentyűzet részeként kezeli, ez azt jelenti, hogy mindkét botkormány állapotát az 'INKEY' függvénnyel is lekérdezhetjük. A két botkormány és a kurzorblokk billentyűinek sorszámait tartalmazza a következő táblázat:

Első botkormány	Második botkormány	Kurzorblokk
[↑]	[↑]	[↑]
72	48	0
[←] [tűz 2] [→]	[←] [tűz 2] [→]	[←] [COPY] [→]
74 76 75	50 52 51	8 9 1
[tűz 1]	[tűz 1]	
77	53	
[↓]	[↓]	[↓]
73	49	2

Amint a táblázatból kiderül, a második botkormány a billentyűzet egyes gombjainak a sorszámát viseli, ezért például a [G] billentyű lenyomására ugyanúgy reagál a BASIC, mintha a második botkormány [tűz 2] gombját nyomtuk volna le.

A botkormányok közvetlen lekérdezésére szolgál a CPC BASIC 'JOY' függvénye. A függvény argumentumaként zárójelben a kívánt botkormány sorszámát kell megadni: (0) első botkormány, (1) második botkormány. A függvény eredménye binárisan tükrözi a botkormányok állapotát:

Bit-sorszám	Decimális érték	Jelentése
0	1	előre (fel)
1	2	hátra (le)
2	4	balra
3	8	jobbra
4	16	tűz 2
5	32	tűz 1

A következő kis programmal kipróbálhatjuk botkormányuk működését:

```
1 rem BOTKORMANY LEKERDEZES
10 b = joy(0)                    - első botkormány
20 print b                       - le volt valami nyomva?
30 goto 10
```

A program futtatása során fel fog tűnni, hogy a CPC a botkormány fő tüzelő gombját tekinti [tűz 2]-nek. A 'JOY' függvény eredményéből kiolvasható a botkormány egyszerre több irányban történő elmozdítása is:

```
JOY(0) = 9    a jobbra-előre irányt jelenti (= 8 + 1)
JOY(0) = 26   a balra-hátra irányt jelenti, és a [tűz 2]
              gomb is le van nyomva (= 4 + 2 + 16)
```

Mivel az első botkormány önálló billentyűsorszámokkal rendelkezik, használhatjuk vele kapcsolatban a 'KEY' és 'KEY DEF' utasításokat is. Próbálja meg botkormányát funkcióbillentyűkké átalakítani! Segítségül egy példa:

```
key def 76, 0, 141, 142, 143          - [tűz 2]: 141, 142, 143
key 141, "cls" + chr$(13)             - sima tűz:  cls
key 142, "run" + chr$(13)             - shift tűz: run
key 143, "list"+ chr$(13)             - ctrl tűz:  list
```

2.8 A NYOMTATÓ KEZELÉSE BASIC-BEN

Új kulcsszó: **WIDTH**

A CPC-vel bármely CENTRONICS típusú csatlakozóval ellátott nyomtató használható, mindössze egy kábel szükséges, amely a a gépet a nyomtatóval összeköti. A Függelékben megtalálható a CPC nyomtatócsatlakozó portjának bekötése, így a kábel akár házilag is elkészíthető.

A BASIC a nyomtatót a gép 8-as számú csatornájaként kezeli, ezért, az ablakokhoz hasonlóan, a 'PRINT', 'WRITE' stb. utasítás után álló **#8** csatornakijelölés küldi a kiíratandó adatokat a nyomtatóra. Például a 'LIST #8' a nyomtatóra listázza ki a CPC operatív memóriájában található BASIC programot. Természetesen egy programrészlet kilistázásához használhatunk opcionális paramétereket is:

```
list #8                 - az egész programot a nyomtatóra küldi
list 100-200,#8         - csak a 100-200-as sorszámú sorok
                          közötti programrész megy a nyomtatóra
list 400-, #8           - a 400-as sortól a program végéig listáz
list -400, #8           - a program elejétől a 400-as sorig, stb.
```

Programok nyomtatóra való listázásakor előfordulhat, hogy a CPC nagylelkűen kezeli a papírt: minden sor után egy üres sort is beiktat a listába. Ennek a magyarázata egyszerű, a CPC ugyanúgy kezeli a nyomtatót, mint a képernyőt, vagyis minden kiírt sor után kiküld egy kocsivissza-soremelés kódkombinációt is: chr$(13) chr$(10). A legtöbb nyomtató alapbeállításban azonban a chr$(13)-ra is emel sort, tehát így két soremelés is történik egymás után. Ezen általában könnyű segíteni: minden nyomtatón van több kétállású kapcsoló, ezek leírása megtalálható a nyomtató kézikönyvében. Keresse meg az 'Automatic line feed' (automatikus soremelés) feliratú kapcsolót és állítsa 'OFF' állásba. Ezután a CPC-ből kiküldött chr$(13) csak visszaküldi a nyomtató fejet a sor elejére de nem továbbítja a papírt, a soremelést a következő chr$(10) végzi el.

Küldjünk ki a nyomtatóra néhány próbaszöveget:

```
print #8, "Ez most a nyomtatora megy."      - akárcsak a kép-
print #8, "111 + 222 = "; 111 + 222           ernyőre menne!
```

Vesse össze nyomtatója karakterkészletét a CPC beépített karakterkészletével:

```
for t=32 to 126: print #8, chr$(t);: next: print #8
```

A nyomtatóra küldendő karaktereket ';' (pontosvesszővel) választottuk el egymástól, különben mindegyik új sorba kerülne. A ciklus után kiadott üres 'PRINT #8' gondoskodik róla, hogy a nyomtató azonnal kinyomtassa a karaktereket. Mivel a legtöbb nyomtató rendelkezik kisebb-nagyobb pufferrel az átvett szöveg tárolására, csak akkor kezd el nyomtatni, ha a puffer betelt, vagy egy kocsivissza-soremelés kódsorozatot kapott. Ezt küldte ki az üres 'PRINT #8' utasítás. A ciklust csak 126-ig futtattuk, mert a chr$(127) arra utasítaná a nyomtatót, hogy felejtse el a közvetlenül előtte kapott karaktert, tehát a chr$(126) sem jelent volna meg a papíron.

▶▶▶ A CPC csak 7-bites kódot tud a nyomtatóra küldeni, így nem nyomtathatjuk ki a CPC grafikus karaktereit. Ha mégis megpróbálkozik például egy 'PRINT #8, chr$(142)'-vel, meglepetésben lesz része: nyomtatója valószínűleg duplaszéles betűkkel ír tovább. A CPC ugyanis levon 128-at a kiküldendő karakter kódjából, ha az meghaladja a 127-et. 142-128 = 14, a chr$(14) pedig a legtöbb nyomtató számára a széles írásra való átkapcsolás kontroll kódja.

A szöveges kurzor pozíciójához hasonlóan ad tájékoztatást a nyomtatófej pillanatnyi vízszintes helyzetéről a 'POS (#8)' függvény. Használata megkönnyíti táblázatok formattált kinyomtatását. Nyomtassuk ki újra a karakterkészletet, tíz karakter

széles oszlopba rendezve:

```
10 for t=32 to 126                      - karakterkészlet
20 if pos(#8) = 11 then print #8        - ha volt 10, új sor!
30 print #8, chr$(t);                   - nyomtasd ki az
40 next t: print #8                       összes karaktert!
```

Hosszabb programsorok vagy szövegek nyomtatóra történő kiíratásakor előfordulhat, hogy a nyomtató váratlanul új sort kezd a szöveg közepén. Ezt is a CPC okozza. A BASIC nyilvántartja a legutolsó kocsivissza-soremelés óta kiadott karakterek számát és egy bizonyos számú karakter után önkényesen kocsivissza-soremelést iktat be a szövegbe. Alapállapotban ez minden 132. karakter után következik be. Egyszerű BASIC utasítással magunk is beállíthatjuk a nyomtató sorszélességét, erre szolgál a 'WIDTH' utasítás, paramétere az 1 - 254 intervallumba esik:

WIDTH maximális sorszélesség

Ha például programunkat a 'MODE 1' képernyőnek megfelelő 40 karakter/sor formátumban kívánjuk nyomtatóra listázni, adjuk be a következő parancsokat:

```
width 40
list #8
```

Gyakran lehet szükség arra, hogy a BASIC ne szóljon bele a nyomtatóra küldendő sorok szélességébe, hagyja ezt a nyomtatóra. Az utasítás 'WIDTH 255' alakja arra utasítja az interpretert, hogy ne törődjön többé a nyomtatóra küldött karakterek számával és ne iktasson közbe kocsivissza-soremelést.

2.9 AMSDOS és CP/M

Ez a fejezet a beépített lemezmeghajtóval rendelkező CPC-k és a DDI-1 típusjelzésű lemezmeghajtóval kibővített CPC 464-es rendszer használóihoz szól. A file-kezelés a következő szakasz témája; mivel a CPC lemezes és kazettás file-kezelése messzemenően hasonló, a csak kazettát használó CPC 464-es tulajdonosok

átugorhatják ezt a részt.

Az **AMSDOS** (Amstrad Disc Operating System = Amstrad lemezoperációs rendszer) és a **CP/M** (Control Program for Microcomputers = mikroszámítógépek kontrolprogramja) egymást kiegészítő operációs rendszerek. A lemezoperációs rendszer fő feladata, hogy programjainkat és adatainkat file-okba szervezve tárolja. A file-t magyarul adatállománynak vagy állománynak is nevezik, egy file lényegében egymáshoz tartozó adatok halmaza, hasonlóan a BASIC-ben megismert tömbhöz. A file-ok lehetnek egyrészt ún. ASCII file-ok, ezek ASCII kód formájában tartalmaznak adatot, szöveget (akár programszöveget is), másrészt bináris file-ok, amelyek kódolt formában tárolnak memóriatartalmat. A lemezoperációs rendszer lehetővé teszi, hogy bármely típusú file-ra névvel hivatkozzunk, azokat lemezre kimentsük, onnan betöltsük, egyik lemezről a másikra másoljuk, átnevezzük, kitöröljük őket, stb. A világon sokfelé lemezoperációs rendszer terjedt el, közöttük a CP/M előkelő helyet foglal el. Óriási előnye, hogy az alatta futó programok szabadon átvihetők egyik gépről a másikra. Így a CPC-ken is futtathatók az immár klasszikussá vált programok, mint például a WORDSTAR szövegszerkesztő. Aki kinőtte a CPC BASIC-jét, CP/M alatt programozási nyelvek tucatját használhatja. A kompatibilitásért azonban nagy árat kell fizetni: a CP/M alatt futó általános programok, programnyelvek nem használják ki a CPC egyedi jó tulajdonságait, mint például a grafika, hang, multitasking, áttervezhető karakterkészlet.

A CPC-hez egyszerre két lemezoperációs rendszer is jár, ezek dinamikusan kiegészítik egymást: amit az egyik nyújt, azt nem mindig találjuk meg a másikban és fordítva. A gép bekapcsolásakor, bár erről felirat nem tájékoztat, az AMSDOS parancsaira hallgat a gép. Az AMSDOS parancsai és utasításai tehát BASIC-ből is elérhetők, mivel az AMSDOS ROM-ban foglal helyet. Más a helyzet a CP/M-mel: ahhoz, hogy ez a rendszer vegye át a gép vezérlését, a lemezmeghajtóba kell helyezni az operációs rendszert tartalmazó ún. rendszerlemezt és be kell adni a '|CPM' parancsot. A '|' jellel még gyakran találkozunk, a [SHIFT] [@] billentyűkombináció állítja elő. Ezzel a jellel kezdődnek az AMSDOS parancsai. A '|CPM' AMSDOS parancs betölti a CP/M operációs rendszert a meghajtóban levő lemezről és átadja

neki a gép vezérlését.

2.9.1 Mire használhatjuk a CP/M-et?

2.9.1.1 Lemezek formattálása

A CPC meghajtója 3-inch méretű CF-2 típusjelzésű lemezek (Compact Floppy Disk) befogadására képes. A tulajdonképpeni mágneses lemez biztonságos fém tokban foglal helyet, amelynek ablaka csak akkor nyílik ki, ha a lemezt a meghajtóba helyezzük. A lemezek mindkét oldalát fel lehet használni, egy-egy oldal 180 Kbyte hasznos információ tárolására szolgál. Mindkét lemezoldal rendelkezik írásgátló réssel, amely a lemezen lévő információ véletlen felülírását hivatott megakadályozni. Az írásgátló rés a lemez elülső felén baloldalt található: amennyiben az ott lévő lyukon átlátunk, a lemezre nem lehet írni. A rést egy műanyag csúszkával eltakarhatjuk: a lemez így írhatóvá válik.

▶▶▶ A géphez kapott rendszerlemezt soha ne tegye írhatóvá! Ne helyezzük a lemezeket erős mágneses tér közelébe, ilyen a gép monitora is, mert ez megsemmisítheti a lemezen lévő információt. Mindig csak bekapcsolt meghajtóba tegyünk be lemezt és még a rendszer kikapcsolása előtt vegyük azt ki onnan.

A nyers lemezt használat előtt formattálni kell. A formattálás során a lemezre jelek kerülnek, amelyek alapján az operációs rendszer megtalálja a sávokat és szektorokat. A lemezen az információt ugyanis koncentrikus körökön rögzítjük, amelyeket sávoknak (angolul track) hívunk, a sávok pedig a gyorsabb elérés érdekében egyenlő hosszúságú részekre oszlanak, ezeket nevezzük szektornak. A meghajtóban levő író-olvasó fejet szinte azonnal bármely sáv tetszőleges szektorára lehet állítani. A CPC negyven sávos lemezeket használ, a sávokat 9 szektorra osztja fel. A lemez formattálása csak CP/M alatt végezhető el, ezért helyezzük be meghajtóba a rendszerlemezt az 1-es oldallal felfelé és adjuk be a már megismert '|CPM' parancsot.

▶▶▶ A CPC 6128-on a CP/M operációs rendszer két verziója is fut: a többi CPC-n is futó CP/M 2.2 és a kibővített

CP/M 3.0, másnéven CP/M Plus. Mivel a következőkben a CP/M 2.2 verziót mutatjuk be, CPC 6128-on használja a 4. lemezoldalt, ott található a CP/M 2.2.

```
|CPM [ENTER]                                          - ezt Ön írta be
(világosra vált a képernyő)
CP/M 2.2 - Amstrad Consumer Electronics plc          - ez a CPC
A>                                                     válasza!
```

Az 'A>' jelzés a CP/M ún. promptja, ezzel tudatja, hogy készen áll parancsaink fogadására. A promptban szereplő 'A' jelzi, hogy a továbbiakban az 'A' nevű (első) meghajtó áll a rendelkezésünkre. Akik második lemezegységgel is rendelkeznek (típusszáma FD1), 'B:' paranccsal kérhetik a rendszertől, hogy a továbbiakban azt kezelje. Ekkor a prompt is 'B>'-re vált át. Mostantól kezdve hatástalan minden BASIC parancs. Próbáljon meg például egy 'CLS'-t kérni a géptől!

```
cls  [ENTER]
CLS?                                  - CLS? hát ez meg micsoda?
A>
```

A CP/M csak saját parancsait érti. A parancsai belső (rezidens) és külső (tranziens) parancsokra oszthatók fel. A belső parancsokat azonnal végrehatja, a külső parancsok végrehajtásához szükséges programok a lemezen file-okban foglalnak helyet, ezeket először be kell tölteni az operatív tárba. Ezért indult el a meghajtó a 'CLS' beadása után: megnézte, van-e a lemezen CLS nevű parancs. A lemezen levő összes file neve és a lemezen elfoglalt helye a gyors hozzáférés érdekében az ún. tartalomjegyzékben (angolul directory) van nyilvántartva, ami szintén a lemezen található. Hogy megtudjuk, létezik-e a keresett nevű program, elég a tartalomjegyzéket bekérni és azt végignézni. A külső parancsok nevét .COM kiterjesztés" zárja le. Hogy egy külső parancsot végrehajtassunk, elég a kiterjesztés nélküli nevét beadni. Az előbb tehát a CLS.COM nevű file-t kereste a rendszer. Adjunk be egy azonnal végrehajtható belső parancsot:

```
dir  [ENTER]                          - listázd ki a tartalomjegyzéket!
```

A képernyőn megjelennek a rendszerlemezen levő file-ok nevei, figyelje meg, milyen sok után áll .COM kiterjesztés. Ezek mind külső parancsok vagy segédprogramok, ún. utility-k. Ott van közöttük a formattáláshoz szükséges FORMAT.COM is. A CPC 6128-on nem jelenik meg a FORMAT.COM, ott a DISCKIT2.COM nevű segédprogrammal formattálhatunk nyers lemezt.

▶▶▶ CP/M-ben nem használhatjuk a szokásos kurzorbillentyűket, csak a [DEL] gombbal léphetünk vissza betűt törölni. A [CLR] billentyűtől különösen óvakodjon, lenyomása nyomtatóra irányítja az összes kiíratást, annak hiányában pedig befagy a rendszer, csak az újraindítás segít!

Formattálás a FORMAT segítségével

Adja be a FORMAT külső parancsot, a program betöltődik és bejelentkezik, majd a következő felirat jelenik meg a képernyőn:

```
Please insert disc to be formatted into drive A
then press any key:_
```

Kövesse az utasítást: helyezze be a formattálandó nyers lemezt az 'A' meghajtóba, majd nyomjon le egy billentyűt!

A lemez formattálásának menetét a képernyőn követheti, a 39. sáv után (számozásuk 0 - 39), a program megkérdezi:

```
Do you want to format another disc (Y/N):_
```

Akar egy másik lemezt is formattálni? Természetesen, hiszen a lemeznek mindkét oldalát fel lehet használni. Fordítsa meg a lemezt és nyomja le az [Y] billentyűt: yes, vagyis igen. Ha azzal is készen van, az újabb kérdésre már válaszoljon nemet: [N], hacsak nem kívánja összes nyers lemezét egyszerre formattálni. Nem rossz ötlet! Sohasem árt, ha van kéznél egy formattált lemez. A rendszer válasza:

```
Please insert a CP/M system disc into drive A
then press any key:_
```

Helyezzen be egy CP/M rendszerlemezt az 'A' meghajtóba és

nyomjon le egy billentyűt. Ne siessen kicserélni a lemezt! A frissen formattált lemeze is megfelelő rendszerlemez: tartalmazza az operációs rendszert és a belső parancsokat. Próbálja ki! Nyomjon le egy billentyűt, megjelenik a 'A>' prompt. Kérje a tartalomjegyzéket!

```
dir                    - tartalomjegyzéket kérek
NO FILE                - nincs egy file sem! (jogos)
```

A FORMAT segítségével rendszerlemezt formattáltunk. A lemez így kevesebb adat befogadására képes, mert a rajta levő CP/M is helyet igényel. A CPC lehetőséget ad arra, hogy a FORMAT parancs után írt paraméterrel (egy karakter) másképpen is formattáljunk.

```
format                 - a rendszert is felviszi a lemezre
format d               - adatlemez, a rendszer nem lesz
                         rajta a lemezen, több hely marad
                         programok, adatok számára
format v               - a rendszert nem viszi a lemezre,
                         de a helyét üresen hagyja
```

Az utóbbi módon akkor formattáljuk a lemezt, ha CP/M alatt futó programunkat értékesíteni kívánjuk. A rendszert ugyanis tilos továbbadni! A vevő a SYSGEN külső parancs segítségével a szabadon hagyott helyre bemásolhatja a CP/M rendszert saját rendszerlemezéről.

Formattálás a DISCKIT2 segítségével

A DISCKIT2 többcélú program: lemezek formattálásán kívül lemezek másolását (copy) és ellenőrzését (verify) is elvégezhetjük vele. Adja be a DISCKIT2 külső parancsot, a program bejelentkezik és közli, hogy egy meghajtót talált: One drive found. A képernyő alsó felében található menüből válassza ki a Format parancsot, a tízes számblokk [4] gombját kell ehhez lenyomni. Az újabb menü a formátum típusát pontosítja, hasonlóan a fentebb elmondottakhoz:

```
System format      [9]      - format
Data format        [6]      - format d
```

Vendor format	[3]	- format v
Exit menu	[.]	- vissza a menühöz

Válassza a System format-ot! A program most egy rendszerlemezt kér, hogy beolvashassa róla az első két sávot, amelyek az az operációs rendszert tartalmazzák és amelyeket rá kell másolni a formatálandó lemezre: Insert a System disc. Mivel a meghajtóban lévő lemez maga is rendszerlemez, nyugodtan nyomjon le egy billentyűt. Két sáv (0-1) beolvasása után vegye ki a rendszerlemezt (Remove disc) és nyomjon le egy gombot. Most még egyszer igazolnia kell, hogy rendszerlemezt kíván formattálni: nyomja le a [Y] billentyűt, bármely más billentyű lenyomására a formattálás félbeszakad. A következő üzenet:

Insert disc to format Press any key to continue■	- tegye be a formattálandó lemezt és nyomjon le egy billentyűt

A formattálás befejeztével válaszoljon igennel [Y] a 'Do you want to format another disc' kérdésre és csinálja végig az előző procedúrát a nyers lemez másik oldalával is.

2.9.1.2 Lemezmásolás CP/M alatt

File-ok, lemezek másolása mindennapos feladat, ezért a rendszer több külső parancsa, segédprogramja könnyiti meg ezt a műveletet. Legelőször is a rendszerlemezről kell egy másolatot készíteni, hogy az eredeti megmaradjon biztonsági tartaléknak. Az ehhez szükséges segédprogram neve a CPC 464 és 664-hez adott rendszerlemezen DISCCOPY.COM. A CPC 6128-on ezt a feladatot is a DISCKIT2.COM látja el. Ezek a programok nem file-onként másolják a lemezt, az túl sokáig tartana, hanem sávonként. Az így kapott másolat teljesen megegyezik az eredetivel.

Lemezmásolás a DISCCOPY segítségével

Helyezze be a meghajtóba a rendszerlemezt és adja be a parancsot: DISCCOPY. A program bejelentkezik és a következő üzenet jelenik meg a képernyőn:

Plase insert **source disc** into drive A
then press any key_

Helyezze be a **forráslemezt** a meghajtóba és nyomjon le egy gombot. A forráslemez az a lemez, amelyről másolunk. Éppen az van a meghajtóban. Nyugodtan nyomjon le egy billentyűt.

Copying started	- a másolás elindult
Reading track 0 to 7	- beolvasom a 0 - 7 sávot

A program egyszerre 8 sávot olvas be, majd lemezcserére szólít fel:

Please insert **destination disc** into drive A
then press any key_

Helyezze be a **céllemezt** a meghajtóba és nyomjon le egy gombot. A céllemez az a lemez, amelyre másolunk. Kövesse az utasítást, vegye ki az eredetit és tegye be a frissen formattált lemezét. Nyomjon le egy gombot és megindul a másolás második fázisa:

Writing track 0 to 7	- kiírom a 0 - 7 sávot

A lemezcserét még néhányszor meg kell ismételni, amíg a rendszerlemez negyven sávja átkerül az új lemezre. A program búcsúzóul megkérdi, hogy akar-e másolni még egy lemezt. Másolja át a másik oldalt is. Végül a már ismerős üzenet kéri fel, hogy tegyen egy rendszerlemezt az 'A' meghajtóba:

Please insert a CP/M system disc into drive A
then press any key_

Az elkészült munkalemezének a második oldala megfelel, még sok hely van rajta a programok számára.

Lemezmásolás a DISCKIT2 segítségével

A DISCKIT2 behívása a már szokásos módon történik. Most azonban a menüből a Copy opciót válassza ki. Egy [Y]-nal még igazolnia kell a választását, amire azonnal megindul a meghajtóban levő lemez beolvasása. A program egyszerre olyan sok sávot olvas be, hogy még a képernyőt is felhasználja az információ tárolására. Majd a következő üzenet jelenik meg:

Disc is system format	- a lemez System formátumú
Insert disc **to Write**	- tegye be **írásra** a cél-
Press any key to continue■	lemezt és nyomjon le egy billentyűt

Helyezze be a frissen formattált lemezét a meghajtóba. A kiírás után újabb beolvasás következik, ami természetesen újabb lemezcserét von maga után:

Insert disc **to Read**	- tegye be **olvasásra** a forráslemezt

A lemezcserét még néhányszor meg kell ismételni. Amikor az egyik oldal készen van, válaszoljon igennel [Y] arra a kérdésre, hogy akar-e még másolni és másolja át a másik lemezoldalt is.

▶▶▶ A DISCCOPY és a DISCKIT2 tud nyers lemezre is másolni: ha a céllemezen nem talál sávokat és szektorokat, először formattálja azt. Erről üzenettel is tájékoztat.

Lemezmásolás két meghajtóval

Két meghajtóval a másolás gyorsabb és kényelmesebb, mivel elmarad a gyakori lemezcsere. A DISCKIT2 önmagától felismeri a második meghajtó jelenlétét. CPC 664-en és 464-en a COPYDISC programot kell használni. Mindkét esetben a forráslemezt az 'A' (első), a céllemezt pedig a 'B' (második) meghajtóba kell helyezni.

2.9.2 File-nevek és kiterjesztések

Mivel a CP/M és az AMSDOS egymással kompatibilis operációs rendszerek, egyazon szabályok vonatkoznak a file-ok névadására is. A file neve legfeljebb nyolc karakterből állhat. Célszerű csak betűket és számokat használni.

A file-nevet '.' (ponttal) elválasztott hárombetűs kiterjesztés követheti. A kiterjesztés a file típusára utal. Két azonos nevű, de más kiterjesztésű file nem zárja ki egymást. Így például a KONYVTAR.BAS és KONYVTAR.DAT file egymás mellett létezhet egy lemezen, az első a BASIC programot tartalmazza, a másik a hozzátartozó adatfile. Az AMSDOS nem várja el, hogy minden file-névhez kiterjesztés járuljon. Ellenkezőleg, maga állít elő kiterjesztéseket:

.BAS	kiterjesztést kapnak a SAVE "program" paranccsal kimentett BASIC programok.
.BIN	kiterjesztést kapnak a SAVE "program",B,cím,hossz paranccsal kimentett gépi kódú programok vagy memóriatartományok.
.BAK	kiterjesztést kapja az összes olyan file, amelynek egy újabb változatát kimentettük a lemezre, típustól függetlenül. A régi változat, mint biztonsági másolat (backup) továbbra is rendelkezésünkre áll .BAK kiterjesztéssel. Az AMSDOS ilyen nagyvonalú gesztusa más operációs rendszerekben ritkaság.
.ooo	nem kapnak kiterjesztést az ASCII formában kivitt szöveg- és adatfile-ok. Ezek kiterjesztését a felhasználó szabadon megadhatja, szinte szabványosak a következő kiterjesztések:
.DAT	adatfile
.TXT	szövegfile
.SOR	assemblerforrásfile (source)

A CP/M külső parancsainak .COM kiterjesztésével már találkoztunk. E programok elindításához elegendő a nevüket beadni, természetesen a .COM kiterjesztés nélkül. A CP/M és az AMSDOS rendelkezik olyan praktikus tulajdonsággal, hogy egyszerre több file-ra is hivatkozhatunk. Ez előnyös lehet akkor, amikor ha-

sonló műveletet kívánunk végeztetni a file-ok egész csoportjával. Két ún. joker karakter helyettesíthet a file-névben egy vagy több karaktert:

?	a file-névben egy karaktert helyettesít
*	a file-névben több karaktert helyettesít vagy helyettesítheti az egész file-nevet és kiterjesztést

A következő példa azt illusztrálja, hogyan használhatjuk a joker karaktereket csoportos file-hivatkozásra:

Tartalomjegyzék	File-név CI??.BA?	File-név *.BAS	File.név K*.*
CICA.BAS	megfelel	megfelel	-
CILI.BAK	megfelel	-	-
KUTYA1.BAS	-	megfelel	megfelel
KUTYA2.BIN	-	-	megfelel
KELEMEN.BAK	-	-	megfelel

A *.* természetesen a tartalomjegyzékben szereplő összes file-ra vonatkozik.

2.9.3 A CP/M gyakran használt parancsai

Könyvünk nem tekinti feladatának, hogy kimerítő leírást nyújtson a CP/M operációs rendszerről. Az érdeklődőt a Függelék irodalomjegyzékéhez utaljuk, ahol a témába vágó magyar nyelvű szakirodalmat is talál. A CPC-re implementált CP/M a lehető legmesszebbmenően megfelel az immár normává vált operációs rendszer szabványának, így gyakorlatilag minden ide vonatkozó információt biztonsággal használhatunk a CPC-vel kapcsolatban is. A továbbiakban a CP/M azon parancsait ismertetjük, amelyek feltétlenül szükségesek ahhoz, hogy kihasználjuk a CPC BASIC által nyújtott összes lehetőséget.

2.9.3.1 Belső (rezidens) parancsok

DIR (directory = tartalomjegyzék)

A leggyakrabban használt belső parancs. Segítségével gyorsan tájékozódhatunk a lemez tartalmáról. A joker karakterek segítségével többféle módon használhatjuk:

DIR	- önmagában az összes file nevét kilistázza abban a sorrendben, ahogy azok a lemezre kerültek
DIR EX1.BAS	- ha az adott file a lemezen van, a nevét írja ki, ha nincs, NO FILE a válasz
DIR *.BAS	- az összes .BAS kiterjesztésű file nevét kilistázza: a lemezen lévő BASIC programokról kapunk így áttekintést
DIR P*.*	- az összes 'P'-vel kezdődő program neve megjelenik a képernyőn

ERA (erase = kitörölni)

A feleslegessé vált file-okat törli ki a lemezről. Használatára előbb-utóbb szükség lesz, hogy hely szabaduljon fel az új file-ok számára. A csak olvasható file-okat nem lehet vele kitörölni (l. STAT).

ERA CICA.BAS	- kitörli a CICA.BAS nevű file-t
ERA *.BAK	- minden .BAK kiterjesztésű file-t kitöröl (hála az AMSDOS nagylelkűségének, ezek gyorsan szaporodnak)
ERA *.*	- a lemezen lévő összes file-t kitörli (ha csak olvasható file-ra bukkan, megszakítja a működését)

REN (rename = átnevezni)

File-ok átnevezésére szolgál. A file-neveket kiterjesztésükkel együtt kell megadni, joker karakterek nélkül. Az ERA-val kombinálva egy biztonsági tartalék file-t újra elővehetünk a feledésből:

```
REN ujnév=réginév
ERA CICA.BAS              - megmaradt a CICA.BAK
REN CICA.BAS=CICA.BAK     - és .BAS file-lá lépett elő
```

TYPE (type = gépirni)

Segítségével ASCII file-okat írathatunk ki a képernyőre.

.BAS, .BIN, .COM file-ok kiíratása katasztrofális eredménynyel járhat, mivel a vezérlő karaktereket is kiadja a képernyőre!

TYPE EX1.BAS — ez a file kivételesen ASCII formátumban lett kimentve, tehát kiíratható

2.9.3.2 Külső (tranziens) parancsok

A külső parancsok valójában .COM kiterjesztésű file-ok, de a belső parancsokhoz hasonlóan elegendő a nevüket beadni, hogy betöltődjenek és átvegyék a vezérlést. A CP/M rendszerlemezen sok tranziens parancs található. Ezek közül már megismertük a FORMAT, DISCCOPY, COPYDISC parancsokat, valamint a CPC 6128-hoz adott DISCKIT2-t. Az alapvető CP/M tranziens parancsok közül a következőket fontos megismerni:

STAT (statistics = statisztika)

A lemezen lévő file-ok számáról, azok méretéről, típusáról, a még fennmaradó szabad helyről tájékoztat. Segítségével file-okat nyilvánithatunk csak olvashatónak, így meggátolhatjuk azok akaratlan módositását, kitörlését.
Többféle formában használható:

STAT — a lemez hozzáférhetőségét (R/W = irható és olvasható, R/O = csak olvasható) és a szabad lemezkapacitást közli

STAT dsk: — a lemezformátummal kapcsolatos információkat közöl

STAT file-név — egyedi vagy csoportos file-név hivatkozás esetén a file(-ok) méretét és hozzáférhetőségét közli

STAT *.* — példa az előző parancsbeadási módra

STAT file-név $R/O — csak olvashatónak nyilvánit egy file-t

STAT file-név $R/W — irhatónak és olvashatónak nyilvánit egy file-t

STAT file-név $SYS — a file többé nem jelenik meg a DIR parancsra kiirt tartalomjegyzékben

STAT file-név $DIR — a file újra megjelenik a tartalomjegyzékben

PIP (peripheral interchange program = perifériák közötti cserét lebonyolító program)

A PIP rendkívül sokoldalú program, segítségével file-okat vihetünk át az egyik perifériáról a másikra. Periférián értjük itt az összes adatbevitelre és kiadásra szolgáló eszközt: lemez, billentyűzet, képernyő, nyomtató, soros interfész stb. File-ok lemezről lemezre történő másolása PIP-pel sajnos csak két meghajtó esetében lehetséges, így CPC-n a FILECOPY parancs használata célszerűbb (l. ott).
A parancs általános szintaxisa: **PIP cél=forrás.** A 'cél' és a 'forrás' egyaránt lehet file-név, egyéb perifériák jelölésére az alábbi rövidítések szolgálnak:

forrás:	CON:	(konzol)	billentyűzet
	RDR:	(reader)	soros interfész
cél:	CON:	(konzol)	képernyő
	PUN:	(puncher)	soros interfész
	LST:	(lister)	nyomtató

PIP LST:=EX1.BAS — nyomtatóra listázza az EX1.BAS nevű file-t (csak ASCII file-okkal használható!)

PIP CON:=EX1.BAS — a képernyőre listázza az EX1.BAS nevű ASCII file-t

PIP SZOVEG.TXT=CON: — a billentyűzetről beadott szöveget a SZOVEG.TXT nevű file-ba irányítja (ha szükséges, akkor létrehoz egy ilyen nevű file-t). A szöveg beadását [CRTL][Z]-vel kell befejezni.

SYSGEN (system generation = rendszer generálás)

'V' (Vendor) opcióval formattált lemezre egy rendszerlemezről átmásolja a CP/M indításához szükséges rendszersávokat. Erre a parancsra akkor van szükség, ha a CP/M alatt futó programokat tartalmazó készen vásárolt lemezről nem tudjuk a rendszert és igy a programot indítani. Ezek a lemezek

általában - jogi okokból - nem tartalmazzák az operációs rendszert.

FILECOPY (file copy = file másolás)

Egy lemezmeghajtóval használható file-másoló program. Úgy egyéni, mint csoportos file-név hivatkozással használható. Az egyes file-ok beolvasása után a program lemezcserére szólít fel, üzenetei a következők:

Please insert SOURCE disc into drive A then press any key:
(helyezze be a forráslemezt és nyomjon le egy gombot)

Copying started (a másolás elkezdődött)

Please insert DESTINATION disc into drive A then press ..:
(helyezze be a céllemezt és nyomjon le egy gomgot)

file-név Copied. (a file másolása megtörtént)

A FILECOPY csoportos file-hivatkozásnál megkérdezi, hogy kívánja-e igazolni az egyes file-neveket, igenlő [Y] válasz esetén egyenként kilistázza a file-neveket és mindegyiknél [Y]-nal jelezhetjük, hogy kérjük a másolást, [N]-nel pedig, lemondhatjuk az adott file másolását. Példák a FILECOPY használatára:

FILECOPY FILECOPY.COM ↵	- átmásolja a FILECOPY.COM nevű file-t (vagyis saját magát) a céllemezre.
FILECOPY *.BAS	- minden .BAS kiterjesztésű file-ot átmásol.
FILECOPY *.*	- a lemezen található összes file átmásolható ezzel a módszerrel. Ha nincs mindre szüksége, akkor is célszerű így beadni a parancsot, legfeljebb a visszaigazolásnál nemmel [N] felel.

AMSDOS

CP/M-ből az AMSDOS-ba térhetünk vissza a paranccsal. Hatása megegyezik a [CTRL][ESC][SHIFT] billentyűkombinációval elérhető újraindítással, így gyakorlatilag felesleges.

DISCCHK (disc check = lemez ellenőrzés)

Egy meghajtó esetén a DISCCOPY-val átmásolt lemezt egybeveti az eredetivel. A DISCCOPY-hoz hasonlóan szólít fel lemezcserére. A DISCKIT2-n található Verify opció ugyanezt végzi el. Ha a két lemez között eltérést talál, kijelzi az első eltérő sávot (track) és szektort (sector) a következő üzenettel:

Failed to verify destination disc correctly:
track x sector y

CHKDISC (check disc = ellenőrizd a lemezt)

Két meghajtó esetén lemezcsere nélküli összehasonlítást végez. A DISCKIT2 Verify opciója automatikusan érzékeli a második meghajtó jelenlétét. A forráslemezt az 'A', a céllemezt pedig a 'B' meghajtóba kell helyezni.

2.9.4 A CP/M billentyűzete és hibajelzései

A CP/M operációs rendszer a CPC-n megszokottól eltérően kezeli a billentyűztet. A kurzormozgató billentyűk ASCII karakterekként jelennek meg a képernyőn, tehát nem használhatók. Parancsbeadáskor csak a visszalépésre van lehetőség a [DEL] billentyűvel. További vezérlő funkciók adhatók be a [CTRL] és egy betűbillentyű kombinációjával:

[CTRL] [C]	- megszakítja a parancsbeadást vagy a futó programot és ún. meleg indítást hajt végre. A meleg indítás a rendszer részleges újrabetöltését jelenti, tehát a meghajtóban rendszerlemeznek kell lenni! CP/M 2.2 verzió alatt meleg indítást kell végezni minden lemezcsere után, CP/M Plus alatt

erre nincs szükség.

[CTRL] [P] - a képernyő mellett a nyomtatóra is irányitja a kiiratást. Másodszori lenyomása után újra csak a képernyőn folytatódik a kiiratás.

[CTRL] [S] - a kiiratást befagyasztja. Másodszori lenyomása a kiiratást folytatja. Hasznos hosszabb file-ok TYPE-pal való kilistázásakor.

[CTRL] [Z] - szövegvégejel. Szövegbeadásnál jelöli a szöveg végét. Felhasználási példát lásd a PIP parancsnál.

A CP/M a lemezekkel kapcsolatban többféle hibajelzést használ, ezek egy részét átvette az AMSDOS is. A CP/M hibajelzései általában annak a meghajtónak a nevével kezdődnek, ahol a hibát észlelte: Drive A: ('A' meghajtó). A gyakoribb hibajelzések:

Disc missing - a meghajtóból hiányzik a lemez (nem létező meghajtóra is ezt mondja!)

Failed to load CP/M - nem tudta a CP/M rendszert betölteni (nincs bent lemez, vagy nem rendszerlemez)

Disc is write protected - a lemez írásvédett (csak olvasható: a felülírást gátló rés nincs eltakarva, vagy a lemez R/O-nak lett nyilvánítva, lásd a STAT parancsnál.)

Bdos error on Drive A - a CP/M részét alkotó BDOS hibajelzése

Disc full - a lemez megtelt

Read/Write fail - olvasási/irási hiba

file-név not found - a file nincs a tartalomjegyzékben

Bad command - (=rossz parancs) az AMSDOS parancsot hibásan adta be

Hibajelzés után a rendszer általában felteszi a kérdést:

Retry, Ignore or Cancel? - újra próbálja, hagyja figyelmen kivül vagy lemondja a műveletet?

Az éppen folyó lemezművelettől függően válaszolhatunk a kívánt opció kezdőbetűjével. Leggyakoribb a [C] válasz, de ha például nincs lemez a meghajtóban és meleg indítást kívánunk végezni, akkor ez sem segít.

2.9.5 Az AMSDOS lemezes parancsai

A CP/M-től eltérően az AMSDOS a gép bekapcsolásakor azonnal a felhasználó rendelkezésére áll. Mivel a CPC BASIC legtöbb file-kezelő parancsa és utasítása független attól, hogy kazettás vagy lemezes rendszerről van szó, itt csak a lemezes rendszeren megtalálható többletparancsokat tekintjük át. A többi parancs a következő szakasz témája. A CPC tervezői igen ötletesen oldották meg a kazettás és lemezes rendszerek kompatibilitását: ha bekapcsoláskor az interpreter érzékeli, hogy a géphez lemezmeghajtót csatlakoztattak (ami CPC 664 és 6128 esetében mindig igy van), minden kazettával kapcsolatos parancsot a lemezegységre irányit át. Ezeken kivül az AMSDOS új parancsok formájában többletszolgáltatásokat is nyújt, amelyek a kazettás rendszeren hiányoznak. Az új parancsok egyaránt használhatók parancsmódban és programsorba ágyazva utasításként. Minden AMSDOS parancs a [SHIFT] [@] billentyűkombinációval elérhető függőleges vonallal kezdődik: '|'. (A [@] billentyűn látható megszakitott vonal megtévesztő, a képernyőn összefüggő vonal jelenik meg.)

▶▶▶ Az AMSDOS parancsok karakterlánc tipusú paramétereit a CPC 664-en és 6128-on idézőjelek között kell beadni, a CPC 464-en először egy változónak kell értékül adni őket, a parancsba '@' és a változó neve kerül. Mindhárom gépen a parancsot vesszővel kell elválasztani az utána következő paraméterektől. Egy példa egyszerűbbé teszi az eljárást:

A PARANCS két szöveges paramétert kíván: CICA, KUTYA.

CPC 664-on és 6128-en így lesz: `|PARANCS,"CICA","KUTYA"`

CPC 464-en bonyolultabb a dolog:
```
a$="CICA": b$="KUTYA":
|PARANCS,@a$,@b$
```

Az új AMSDOS parancsok a következők:

|A - az első lemezmeghajtót választja a további lemezműveletekhez (ez a bekapcsolás utáni alapállapot)

!B - a második lemezmeghajtót választja a további lemezműveletekhez (ha ilyen nincs: Disc missing üzenet)

**!DRIVE,meghajtó$** - az előző parancsokhoz hasonló, de rugalmasabb, mert változóval adhatjuk meg a kívánt meghajtó$ nevét: "A" vagy "B"

!CPM - az 'A' meghajtóban levő CP/M rendszerlemezről betölti a CP/M operációs rendszert és átadja neki a vezérlést

!DIR - a CP/M DIR parancsának felel meg: a tartalomjegyzék kilistázására szolgál. A CP/M DIR-hez hasonlóan szöveges paraméterrel is beadható, például csak a .BAS kiterjesztésű file-okat listázza ki a következő formában:

```
CPC 664 és 6128:        !DIR,"*.BAS"
CPC 464:   a$="*.BAS": !DIR,@a$
```

!ERA - a CP/M ERA parancsának felel meg: file(ok) törlésére szolgál, a file-hivatkozást szöveges paraméterként kell megadni. A CICA.BAS nevű file-t törölhetjük

```
CPC 664-en és 6128-on:    !ERA,"CICA.BAS"
CPC 464-en:  a$="CICA.BAS": ERA,@a$
```

Csoportos file-hivatkozást is elfogad, például egy !ERA,"*.*" törli a lemezen található összes file-t.

!REN - a CP/M REN parancsának felel meg: file-ok átnevezését teszi lehetővé. Két szöveges paramétere közül az első a file új neve, a második a régi név.

```
CPC 664 és 6128:  !REN,"UJNEV.BAS","REGINEV.BAS"
CPC 464:  u$="UJNEV.BAS": r$="REGINEV.BAS":
          !REN,@u$,@r$
```

A további AMSDOS parancsok lehetővé teszik hogy lemezes rendszerrel is folytathassunk kazettás file-műveleteket. Mivel alapállapotban a gép író és olvasó file-kezelő parancsai a lemezegységre irányulnak, kazettás programok betöltése, file-ok kazettára való kivitele csak a következő parancsok után lehetséges:

!TAPE - minden további író-olvasó file-művelet a kazettára irányul. Betölthetünk kazettán lévő programokat, mintha lemezmeghajtó nem is lenne a géphez kapcsolva. Programok, file-ok kimentése is a kazettára történik.

!TAPE.IN - csak az olvasó file-műveletek céljára választja a kazettát. Lehetőség nyílik így programot kazettáról beolvasni és lemezre kivinni.

!TAPE.OUT - csak az író file-műveletek céljára választja a kazettát. Lemezről tudunk így file-okat kazettára másolani.

!DISC - minden további író-olvasó file-művelet újra a lemezre irányul. Megszünteti az összes !TAPE parancs hatását.

!DISC.IN - csak az olvasó file-műveletekhez állítja vissza a lemezt.

!DISC.OUT - csak az író file-műveletekhez állítja vissza a lemezt.

2.10 FILE-KEZELÉS BASIC-BEN

A CPC fejlett file-kezelő parancsai egyaránt használhatók kazettás és lemezes rendszeren. A kazettás felhasználóknak csak két hátránnyal kell szembenézniük, az egyik a műveletek lassabb végrehajtása, a másik pedig az, hogy a kazetta gyors előre-hátra tekercselését nekik maguknak kell elvégezni. A beépített kazettás magnó számlálója lesz ebben a segítségükre. A két rendszer közötti kisebb eltérésekre minden alkalommal kitérünk. Mivel a továbbiakban egyszerre szólunk lemezről és kazettáról, a kiviteli eszközt külső tárnak nevezzük.

2.10.1 Programfile-ok

Új kulcsszavak:

SAVE
LOAD
MERGE
CHAIN
CHAIN MERGE
SPEED WRITE
CAT

A CPC kétféle programfile-t kezel: BASIC programot tartalmazó file-t és gépi kódú programot tartalmazó bináris file-t. Minden file kimenetésekor először egy ún. fejlécet ír ki, ez tartalmazza a file-lal kapcsolatos legfontosabb adatokat: nevét, típusát, hosszát, betöltési címét stb. Betöltéskor ez a fejléc árulja el, hogy mit is kell a bejövő adatokkal kezdeni.

▶▶▶ A kazettás file-ok neve maximum 16 karakterből állhat. Ha azonban a file-t később esetleg lemezre kívánja átvinni, csak nyolckarakteres nevet használjon. Nézzen utána a CP/M file-nevekkel kapcsolatos megkötéseinek! A kazettás rendszer üzenetekkel szólít fel a magnó bekapcsolására, kiírja a megtalált program nevét, stb. Az üzenetek kiadását kikapcsolhatja, ha a file-nevet '!' (felkiáltójellel) kezdi (ez nem számít be a név hosszába), a lemezes rendszer figyelmen kívül hagyja a file-nevet kezdő '!'-t.

A kazettára való kiíratás sebességét a **'SPEED WRITE 1'** parancs a duplájára növeli. Visszaállni a lassabb, de biztosabb sebességre a **'SPEED WRITE 0'** paranccsal lehet. Beolvasáskor nincs szükség sebességállításra, mert a CPC magától is felismeri az aktuális sebességet.

Programfile-ok kimentését a 'SAVE' parancs végzi el. A parancsot többféle alakban adhatjuk be a file típusától függően. A példákban szereplő "file-név" szöveges típusú változó vagy konstans lehet.

SAVE "file-név" — az operatív tárban található BASIC programot a külső tárba menti. A programot tokenizált (sűrített) formában viszi ki. A kiterjesztés nélküli file-nevet az AMSDOS automatikusan a .BAS kiterjesztéssel látja el.

SAVE "file-név",P — az előző formában menti ki a BASIC programot, de azt védettnek (protected) nyilvánítja: a programot csak 'RUN'-nal lehet betölteni és indítani. A leállított program megsemmisíti önmagát, kívülálló így nem tudja listázni. (De többé Ön sem!)

SAVE "file-név",A — az operatív tárban lévő BASIC programot ASCII-file formájában menti ki a külső tárba. Az így kiírt program hosszabb, mint a tokenizált program, viszont megvan az az előnye, hogy adatfile-ként kezelhető: szöveges változókba be lehet olvasni, stb. A kiterjesztés nélküli file-név AMSDOS alatt sem kap kiterjesztést.

SAVE "file-név",B, kezdőcím, hossz [, belépési cím]
Az operatív tár (RAM) egy részét tárolhatjuk bináris file-ként ezzel parancsmóddal. A 'kezdőcím'-től 'hossz' mennyiségű byte-ot ír ki a külső tárba. 'RUN'-nal indítandó gépi kódú programoknál azok 'belépési címét' is meg kell adni. Az AMSDOS a file-nevet automatikusan a .BIN kiterjesztéssel látja el. A CPC teljes képernyőjét a SAVE "kepernyo",b,&c000,&4000 paranccsal menthetjük ki.

Programfile-ok betöltésére szolgál a 'LOAD' parancs. A betöltendő file fejlécéből felismeri, hogy sűrített BASIC, ASCII-formátumú BASIC vagy bináris file-ról van szó. Az utóbbinak a fejlécéből a betöltési címet is megtudja: a 'kezdőcím'-re tölti be. Lehetőség van azonban a parancsban új betöltési címet meghatározni:

LOAD "file-név" — BASIC programfile-t az operatív tárba tölt. Bináris file-t a kimentéskor megadott 'kezdőcím'-re tölt.

LOAD "file-név", betöltési cím — bináris file-t a kivitelkor megadott 'kezdőcím'-től függetlenül a 'betöltési cím'-re tölt be.

▶▶▶ Kazettán lehetséges file-ok kimentése és betöltése file-név megadása nélkül is. Beolvasáskor a kazettán található első file jön be. Kivitelkor inkább adjon neki nevet!

RUN "file-név" BASIC vagy bináris file-t a külső tárból betölt és azonnal indít. 'P' opcióval kimentett védett programok csak így tölthetők be.

▶▶▶ Lemezről való betöltéskor kiterjesztés nélküli file-nevek esetén az AMSDOS a következő módon keresi a file-t:

1. kiterjesztés nélkül, ha ilyen nincs, akkor
2. .BAS kiterjesztéssel, ha így sem találja, akkor
3. .BIN kiterjesztéssel próbálja meg.

Gyakran fordul elő, hogy BASIC programunkban már kidogozott programrészleteket, szubrutinokat kívánunk felhasználni. Erre a 'LOAD' parancs nem alkamas, mivel az törli a tárban levő régi programot. Megtörténhet az is, hogy programunk nem fér el egyszerre a CPC operatív tárában. Mindkét esetben sikeresen használhatjuk a következő utasításokat, amelyek az operatív tárban levő programhoz a külső tárból fűznek hozzá újabb programo(ka)t.

MERGE "file-név" az operatív tárban lévő BASIC programhoz fésüli a "file-név" nevű BASIC programot. A két program esetleg megegyező sorszámú sorai közül a bejövő program sorai felülírják a tárban lévő program sorait.

CHAIN "file-név" [, sorszám] törli a tárban levő programot, betölti és az első sortól vagy az opcionálisan megadott 'sorszám'-tól indítja a "file-név" nevű BASIC programot. Az új program nem veszi át a régi változóit!

CHAIN MERGE "file-név" [,sorszám [,DELETE tól-ig]]
a 'MERGE'-hez hasonlóan a tárban lévő programhoz fésüli a "file-név" nevű BASIC programot. Előzőleg végrehajtja az opcionálisan megadott 'DELETE' parancsot. Az így előállt program az első sortól vagy az opcionálisan megadott 'sorszám'-tól indul.
Az új program a régi összes változójával rendelkezik!

▶▶▶ Becsüljük meg a 'CHAIN MERGE' utasítást! Segítségével gyakorlatilag korlátlan méretű BASIC programokat kezelhetünk. Az operatív tárban mindig csak az a programrészlet foglal helyet, amelyre éppen szükség van, a többit utána töltjük, előzőleg kitörölve a már feleslegessé vált sorokat.

A CPC 464-es AMSDOS alatt csak ASCII-formátumban kivitt programfile-okat tud kezelni a 'MERGE', 'CHAIN MERGE' utasításokkal, sűrített BASIC file-ok esetén 'EOF found' hibajelzéssel leáll.

Utoljára maradt az egyik leggyakrabban használt parancs a:

CAT

A 'CAT' parancs a külső tárolón lévő file-okról nyújt információt, AMSDOS alatt kilistázza a lemez tartalomjegyzékében lévő file-ok nevét és kiterjesztését, azok hosszát Kbyte-ban, valamint megjeleníti a még szabad lemezkapacitást is. Kazettás rendszerben a 'CAT'-nak fizikailag végig kell néznie az egész szalagot, ezért soká kell rá várni, de megéri. Az előállított katalógus a következőképpen épül fel:

```
file-név    blokkszám    file-típusjel    OK
```

A legutolsó blokkszámból megtudhatjuk a file-méretét: egy blokk 2 Kbyte hosszú. A 'OK' természetesen azt jelenti, hogy a file rendben van, minden blokkja olvasható. Ha nem így van, azt a 'Read error A vagy B' (olvasási hiba) üzenettel jelzi. A file-típusjelek a következők:

- $ BASIC program
- % védett ('P' opcióval kimentett) BASIC program
- * ASCII adatfile
- & bináris file

2.10.2 Adatfile-ok és programozásuk

A CPC BASIC csak soros file-ok használatát teszi lehetővé. Nem változtat ezen a tényen az AMSDOS alatt működő lemezegység jelenléte sem. Ha Ön már járatos a file-programozás területén és úgy érzi, hogy közvetlen hozzáférésű file-ok nélkül nem

tud meglenni, beszerezhet olyan CP/M alatt futó BASIC értelmezőt vagy fordítót, amely a file-kezelés tekintetében minden igényét kielégíti. Mivel azonban a CPC lemezmeghajtója igen gyors, a soros file-ok az esetek nagy többségében nem maradnak le más gép, például a Commodore 64 körülményesen programozható 'relative' file-jai mögött.

2.10.2.1 Az elemi file-műveletek

Új kulcsszavak: **OPENIN**
OPENOUT
CLOSEIN
CLOSEOUT
EOF

Mit jelent a soros file fogalma? Egész egyszerűen egymás után írt adatok halmazát. Mivel egy file adatait csak a számítógépben lehet feldolgozni, az adatokat be kell olvasni, majd feldolgozás után vissza kell írni a külső tárba. A file adatainak a beolvasása úgy történik, hogy egy-egy változóhoz rendeljük őket hozzá, olyan sorrendben, ahogy a külső tárba kerültek. Egy adategységet blokknak nevezünk (például egy személy neve, címe, telefonszáma). Ha egy meghatározott blokk adataira van szükségünk, az összes előtte levő blokkot be kell olvasni és az operatív tárban végigvizsgálni, megtaláltuk-e a keresettet. Minden adat után egy elválasztójel áll, amelyről a gép az adat végét felismeri. A file végét az ún. file-végjel zárja le.

A soros file-ban való keresést ahhoz lehet hasonlítani, mint amikor egy regényben keresünk egy epizódot, de nem tudjuk hol olvastuk: végig kell lapozni a könyvet. Ezzel szemben az ún. közvetlen hozzáférésű file egy szótárhoz hasonlítható: ha a 'kutyá'-ra van szükség, a 'K' betűnél ütjük fel, nem lapozzuk végig az 'A'-tól. A CPC BASIC ezt sajnos nem tudja, viszont gyorsan lapoz!

Mielőtt egy file-hoz hozzáférhetünk, meg kell nyitni. Ha befejeztük a file feldolgozását, a file-t le kell zárni. A következő parancsok ezt a célt szolgálják:

OPENOUT "file-név" - írásra file-t megnyit
OPENIN "file-név" - olvasásra file-t megnyit
CLOSEOUT - az írásra megnyitott file-t lezárja
CLOSEIN - az olvasásra megnyitott file-t lezárja

▶▶▶ Kazettás rendszeren a file megnyitását megelőzőleg elő kell készíteni a magnetofont felvételre vagy lejátszásra. Ha a file-név nem '!' (felkiáltójellel) kezdődik, a CPC is felszólít erre.

A megnyitott file-t a CPC a 9-es számú adatcsatornának tekinti, tehát a bele való íráshoz ugyanolyan szabályokat kell betartani, mint amikor az ablakokba (#0 - #7), vagy a nyomtatóra (#8) küldtünk ki szöveget. Mivel a file a 9-es csatorna, minden rá vonatkozó 'PRINT', 'WRITE', 'INPUT', 'LINE INPUT' utasítást #9-cel kell jelölni. Hozzunk létre egy próbafile-t, amely ismerőseink nevét, címét és telefonszámát tartalmazza.

```
10 rem PROBAFILE KIIRAS 1
20 rem
30 openout "probfil1.dat"                - íráshoz megnyit
40 print #9,"Kovacs Janos"                - #9 : file-ba ír
50 print #9,"Budapest I. Cica u. 1"       - #9 : file-ba ír
60 print #9,"111-111"                     - #9 : file-ba ír
70 rem elso blokk megvolt
80 print #9,"Horvat Dezso"
90 print #9,"Miskolc I. Kutya u. 2"
100 print#9,"222-222"
110 rem masodik blokk megvolt
120 print#9,"Fekete Piroska"
130 print#9,"Szeged I. Kacsa u. 3"
140 print#9,"333-333"
150 rem harmadik blokk is megvolt
160 closeout                              - írás után lezár
170 end                                   - a rend kedvéért
```

Az adatokat egyenként írtunk ki a file-ba. A BASIC közéjük elválasztójelet tett, jelen esetben ez egy kocsivissza-újsor karakterkombináció. Ezt láttuk volna akkor is, ha a kiíratás a képernyőre vagy a nyomtatóra megy: minden 'PRINT' után új sort

kezd a gép. A CPC egy másik elválasztójelet is ismer: ez a ',' (vessző). A két jel egyenrangú. Beolvashatnánk a file-unkat ugyanúgy, mint ahogy kiírtuk: minden adatot egyenként. Kilenc 'INPUT #9, a$' megtenné. De most inkább egy egész blokkot egyszerre olvasunk be: elválasztójelként az 'INPUT #9' változólistájában a vesszőt használjuk. Mivel egy 'INPUT #9'-cel három összetartozó adatot olvasunk be egyszerre, programunk nemcsak áttekinthetőbb lesz, hanem rövidebb is. Tovább rövidíthetjük a programot, ha a beolvasáshoz ciklust használunk:

```
10 rem PROBAFILE BEOLVASAS 1
20 rem
30 openin "probfil1.dat"                  - olvasásra megnyit
40 for blokk = 1 to 3                     - 3 blokkunk van
50   input #9, nev$, cim$, telefon$       - egy blokkot beolvas
60   print nev$, cim$, telefon$           - ezt írtuk ki?
70 next blokk                             - lássuk a következőt!
80 closein                                - olvasás után lezárni
90 end
100 rem folytatjuk!
```

A képernyőn megjelennek a file-ból beolvasott adatok, programunk működik és sokkal rövidebb, mint a file-ba író program volt. Valószínűleg felmerül az Olvasóban a gondolat: a kiírást is ilyen röviden kellene elintézni. Egyelőre hagyjuk meg próba-file-unkat és foglalkozzunk a beolvasással, a kiírásra később visszatérünk.

A beolvasó program ugyan jól működik, de használhatósága korlátozott: három adatblokkra van felkészítve, se többre, se kevesebbre. Mit tehetünk annak érdekében, hogy file-unk tetszőleges hosszúságú legyen? Több megoldás is kínálkozik:

Az első kivitt adat ne adatblokk legyen, hanem egy szám, amely a beolvasandó adatblokkok számát adja meg. Valahogy így:

```
input #9, blokkokszama
for i= 1 to blokkokszama
    input #9, a$,b$,c$
next i
```


A módszert később még felhasználjuk. Nemcsak a file elejére tehetünk jelet, hanem a végére is. A 'DATA' sorok beolvasásához hasonlóan az utolsó valódi adatblokk után álblokk kerül a file-ba, amelyről felismerhetjük, hogy itt a vége. Az álblokk adatszerkezete meg kell hogy egyezzen a valódi blokk szerkezetével. Ha három adatból állnak blokkjaink, az utolsónak kivitt álblokkot például

```
for i=1 to 3: print #9,"*": next i
```

utasítással hozhatjuk létre. A beolvasás és ellenőrzés így történhet:

```
vege = 0
while not vege
  input #9, a$,b$,c$
  if a$="*" and b$="*" and c$="*" then vege=-1
wend
```

Amíg a 'vege' értéke nulla marad, a 'WEND' visszaküldi a a programot egy újabb 'INPUT #9'-hez. Ha megtaláltuk az álblokkot, a 'vege'-t -1 -re állítjuk és kilépünk a ciklusból.

De valójában mit csinál a BASIC akkor, ha megpróbálunk a file végén túl is olvasni? Próbálja ki! Módosítsa a beolvasó program ciklusát úgy, hogy az négy blokkot olvasson be:

```
40 for blokk= 1 to 4                      - nincs is ennyi blokk!

run
.... .... ...                             - ezek a meglévő blokkok
EOF met in 50                             - az EOF-fal találkoztam!
```

EOF = End Of File, vagyis a file vége. Az utolsó blokkunk után tehát ott áll az **EOF** jelzés, amely láttán az interpreter hibajezéssel leáll. Fizikailag az EOF nem más, mint egy vezérlő karakter: CHR$(26). Ezt a karaktert sohasem írhatjuk ki file-ba még szöveges változóba ágyazva sem, mert a BASIC file-végként fogja értelmezni és nem lesz hajlandó tovább olvasni! Az **EOF** azonban nemcsak egy karakter, amely a file végét jelzi, hanem

egy BASIC rendszerváltozó neve is. Az 'EOF' változó értéke -1, amíg a BASIC nem találkozott a file végével, ha találkozott vele, az 'EOF' értéke nullára változik.

```
print eof                                  - írd ki az értékét!
 0                                         - már találkozott vele
```

Újra módosítsa a 40-es sort úgy, hogy a ciklus csak két blokkot olvasson be. Futtassa le a programot és kérdezzen rá az 'EOF' értékére!

```
print eof                                  - vajon most mennyi lesz?
-1                                         - még nem volt file-vég
```

Nem kell álblokkal jelölnünk tehát a file végét, elég az 'EOF' változó értékét figyelni ahhoz, hogy megtudjuk, van-e még olvasnivalónk. A következő beolvasó program tetszőleges blokkszámmal használható:

```
10 rem probafile beolvasas 1.1
20 rem
30 openin "probfil1.dat"
40  while not eof                          - amíg nem látsz EOF-ot
50   input #9, nev$, cim$, telefon$        - addig olvass be blokkot
60   print nev$, cim$, telefon$
70  wend                                   - újra és újra
80 end
```

▶▶▶ Ezzel a módszerrel tetszőleges blokkszámú file-t olvashatunk be, de csak akkor, ha a blokkok szerkezetét - hány elemből áll, azok milyen típusúak stb. - ismerjük. Célszerű ezért első adatként a blokkok felépítésére vonatkozó információt kiírni a file-ba. Így évek múltán is elég lesz ezt beolvasni, hogy a rég elfelejtett file-t újra használatba vehessük.

2.10.2.2 Adatelválasztó-jelek

Eddig egy adatelválasztó-jelet használtunk: 'PRINT #9, a$' az adat után kocsivissza-újsor jelet küldött ki a file-ba. Az így kiírt adatokat beolvashatjuk egyenként 'INPUT #9, a$'-ral, vagy blokkonként az 'INPUT #9' után vesszővel elválasztott adatlistába. Próbáljuk meg az adatokat is blokkonként kivinni:

```
10 rem PROBAFILE KIIRAS 2
20 rem
30 openout "probfil2.dat"
40 print #9, "elso adat","masodik adat","harmadik adat"
50 closeout
60 end
```

Olvassuk be a file-t a szokásos módon egy blokkban:

```
10 rem PROBAFILE BEOLVASAS 2
20 rem
30 openin "probfil2.dat"
40 input #9, a$, b$, c$
50 print a$, b$, c$
60 closein
70 end

run
EOF met in 40
```

Elértük volna a file végét? Három adatot ment ki a file-ba, annyit is vár el a 40-es sorban az 'INPUT #9'. Egyáltalán beolvasott valamit?

```
print a$                                                    - mi van a$-ban
elso adat     masodik adat harmadik adat                    - három adat!
```

A 'PRINT #9' ugyanúgy ír ki a file-ba, ahogy a 'PRINT' ír a képernyőre: a vessző után a következő tabulátorpozícióig szóközöket ír, majd a következő adatot és így tovább. Ez ment ki a file-ba is. Az utolsó adatot nem zárta le semmi, tehát egy 'CHR$(13) CHR$(10)' követte. Az 'INPUT #9' adatelválasztó-jelig

olvas, szöveges változóknál pedig a szóköz nem elválasztójel. File-unk tehát nem tartalmazott egyetlen adatelválasztó-jelet sem, beolvasáskor így az egész file (az egy sor szöveg) az első változóba került. A többi változóba már nem volt mit olvasni, elértük a file végét. Tegyünk elválasztójelet a 'PRINT' lista elemei közé, módosítsuk a PROBAFILE KIIRAS 2 programot:

```
40 print #9,"elso adat"; "," ; "masodik adat"; "," ; "harmadik
adat"
```

Nehézkes, igaz, de így már működik a beolvasás is. Sok más gépen csak így oldható meg az adatelválasztó-jelek elhelyezése. A CPC BASIC azonban egyszerűbb megoldást is kínál: a 'WRITE' utasítás, akárcsak a képernyőn, a file-ban is vesszővel választja el egymástól a kiírandó adatokat. A szövegeket idézőjelek közé zárja, de azokat az 'INPUT' úgyis lehámozza róluk. Előző módosításunkat írjuk át újra, egyszerűbben:

```
40 write #9,"elso adat","masodik adat","harmadik adat"
```

Hogy végleg tisztázzuk a 'PRINT' és 'WRITE' közötti különbséget, hasonlítsa össze a kétfajta kiíratást a képernyőn és ne feledje: a file-ba az adatok ugyanígy kerülnek ki. Ahhoz pedig, hogy az adatokat egyenként be tudjuk olvasni, el kell őket választani vesszővel vagy kocsivissza-újsor jellel.

```
print "egy","ketto",harom"
egy           ketto       harom                    - nincs vessző!
Ready
write "egy","ketto","harom"
"egy","ketto","harom"                              - van vessző!
Ready
```

Az a tény, hogy az 'INPUT #9' számára a vessző elválasztó jelnek számít és a szövegekről lehámozza az idézőjelet, alkalmatlanná teszi ezt az utasítást valódi szöveges file-ok beolvasására. Egy szöveges file-ban, legyen az egy levél vagy egy 'SAVE "program",A' opcióval kivitt BASIC program, minden írható karakter előfordulhat, beleértve a vesszőt és az idézőjelet is. Olyan utasításra van szükség, amely csak a sor végét, a kocsi-

vissza-újsor karaktert, tekinti elválasztó jelnek. A CPC BASIC ilyen utasítást is ismer, ez a 'LINE INPUT'.

Állítson elő egy szövegfile-t! Nem kell sokat programoznia, elég, ha az éppen a gépben lévő programot ASCII-file formában kimenti a külső tárba:

```
save "szoveg.txt",a
```

Ez a parancs ugyanolyan formában menti ki a programot, ahogy azt a 'LIST' a képernyőre, vagy a 'LIST #8' a nyomtatóra kiírja: egy sor szövege kocsivissza-újsor egy sor szövege kocsivissza-újsor egy sor szövege ... EOF. Az EOF természetesen csak a külső tárba kiírt file-ra vonatkozik, képernyőn és nyomtatón nem lenne értelme.

A létrehozott szövegfile-t, amelyben hemzsegnek a vesszők és az idézőjelek, a következő programmal olvassuk be és íratjuk ki a képernyőre:

```
10 rem SZOVEGFILE BEOLVASAS
20 rem
30 openin "szoveg.txt"                  - szokásos megnyitás
40 while not eof                        - csak a file végéig
50   line input sor$                    - olvass be egy sort
60   print sor$                         - lássuk a képernyőn
70 wend                                 - újra, ha nem EOF
80 closein
90 end
```

Az így beolvasott BASIC programfile képernyőre történő kiíratása csak példa volt a szövegfile-ok használatára. Sokkal hasznosabb a programsorokat egy nagy tömbbe beolvasni, amelyből a sorokat egyenként elővehetjük és feldolgozhatjuk szövegkezelő függvények segítségével. Miféle feldolgozás végezhető egy BASIC programon? Csak egy pár ötlet: nyomtatóra kilistázhatjuk úgy a programot, hogy minden utasítás új sorba kerüljön, a 'FOR-NEXT', 'WHILE-WEND' ciklusok magjai pedig beljebb kezdődjenek. Jóval olvashatóbb, áttekinthetőbb listát kapunk így, mint egy közönséges 'LIST #8' paranccsal. A tömbbe beolvasott program-

sorokon tetszőleges változtatást végezhetünk - változónevek módosítása, gyakran ismétlődő szövegek megváltoztatása stb. Ha a tömböt visszaírjuk a külső tárba, a BASIC 'LOAD' paranccsal betölthető majd futtatható módosított programunk. Nagyon hosszú programok nem férnek el egyszerre egy tömbben, ezért célszerű őket soronként feldolgozni. Ez csak két file megnyitásával lehetséges: az egyikből beolvasunk egy sort, azt feldolgozzuk és a másikba írjuk ki. Erre szolgál például a következő program, amely az ASCII-file-ként kivitt BASIC programból kiírtja a megjegyzéseket, amelyeket REM vagy ' (felső vessző) vezet be. Próbaképpen maga a REM-TELENITO program is remtelenithető!

```
10 REM ********************
15 REM *                  *
20 REM *   REM-TELENITO   *
25 REM *                  *
30 REM ********************
40 INPUT"filename";fbe$          'ASCII file: ezt kell REMteleniteni
50 OPENIN fbe$                   'beolvasasra megnyit
60   fki$=fbe$+".BAS"            'fki$ lesz a neve az uj file-nak
70 OPENOUT fki$                  'megnyitjuk kiirashoz
80 WHILE NOT EOF                 'csak a file vegeig olvas be
90 LINE INPUT #9,a$              'teljes sor, sorvegjel: chr$(13)
100 hely=INSTR(1,a$,"REM")       'van benne REM?
110 IF hely=0 THEN 130           'ha nincs, akkor '-t keres
120 a$=LEFT$(a$,hely-1)+":"      'ha van: attol kezdve nem kell
130 hely=INSTR(1,a$,"'")         'felso vesszot keres
140 IF hely=0 THEN 160           'ha nincs, akkor kiirhato
150 a$=LEFT$(a$,hely-1)+":"      'ha van: attol kezdve nem kell
160 PRINT #9,a$                  'feldolgozva, kiirhato
170 WEND                         'van meg beolvasnivalo?
180 CLOSEOUT                     'nincs: lezar file-okat
190 CLOSEIN
```


2.10.2.3 File-kezelés: tippek és trükkök

Egy file feldolgozása, adatok, adatblokkok módosítása, törlése, beszúrása úgy végezhető el a legegyszerűbben, ha az egész file-t egy vagy több tömbbe beolvassuk, majd a feldolgozás után újra kiírjuk a külső tárba.

▶▶▶ Ha egy file-t írásra megnyitunk, az felülírja a lemezen található azonos nevű file-t. Az AMSDOS jóvoltából a régi file .BAK kiterjesztéssel továbbra is rendelkezésre áll. A kazettás magnó természetesen oda ír, ahova a szalagot beállítottuk. Ne írjuk felül régi file-unkat, szükség lehet még rá!

Rendkívüli méretű file-ok esetén előfordulhat, hogy még a CPC operatív tárába sem fér be egyszerre az összes adat. Többféle megoldás is kínálkozik ilyenkor. A legegyszerűbb és leghatékonyabb megoldás a megelőzés: ne hozzunk létre gigantikus file-okat, hanem több kisebb file-ba csoportosítsuk az adatokat. Ezek egyenként beolvashatók tömbökbe, a feldolgozás után pedig a feleslegessé vált tömbök 'ERASE' utasítással törölhetők, hogy helyüket a következő file-t elszállásoló tömbök foglalhassák el. Gyakran nincs is szükség egyszerre minden adatra, csak egy bizonyos adatcsoportot szeretnénk beolvasni. Ezt saját használatú elválasztójelekkel tehetjük meg. Például, az összetartozó adatblokkokat "*1*", "*2*", "*3*" stb. jelöléssel választjuk el. Ha tudjuk, hogy a minket érdeklő adatok a "*3*" jel után következnek, a beolvasás így történhet:

```
jel$=""
while jel$ <> "*3*"
  input #9, jel$
wend
rem most már a keresett adatblokkok jönnek,
rem tömbbe lehet olvasni őket
```

Tegyük fel, hogy feldolgoztuk a "*3*" jel után következő adatokat és most a "*2*" jelet követő adatokra van szükség. A soros file-ban azonban csak előre lehet lapozni. Ahhoz, hogy a file adatmutatóját újra az első adatra állítsuk, le kell

zárni a file-t, majd újra meg kell nyitni. Ezzel az adatmutató újra a file elejére áll be. Kazetta esetében ez a szalag mechanikus visszatekercselését igényli.

▶▶▶ Megnyitott file mellett nem lehet egy újabb, azonos jellegű file-t megnyitni, ez a 'File already open' (file már nyitva) hibajelzéshez vezet. Ha megnyitás nélkül próbálunk file-ba írni, a 'File not open' (file nincs nyitva) hibaüzenetet kapjuk. Az AMSDOS hibajelzései a CPC 464-en minden esetben megszakítják a program futását. CPC 664-en és 6128-on az 'ON ERROR GOTO' hibakezelés lemezhibák esetén is alkalmazható. A hiba kódszámát a **'DERR'** rendszerváltozó tartalmazza. Ismertetése a Függelék A CPC BASIC kulcsszavainak áttekintése című részben található.

A CPC-n egyszerre két file-t lehet megnyitni: egyet az 'OPENIN', egyet pedig az 'OPENOUT' utasítással. Kazettán ez csak teoretikus lehetőség, gyakorlatilag nem kivitelezhető a kazetta cseréje vagy a szalag állandó tekercselgetése miatt. AMSDOS alatt azonban rugalmassá válik a file-kezelés: az egyik file-ból beolvasunk, az adatok közül az éppen szükségeseket a másik file-ba írjuk át, elmarad a közbenső tárolás. Ezen kívül még sok más felhasználás is elképzelhető egy olvasásra és egy írásra megnyitott file egyidejű alkalmazásával.

Nagy mennyiségű szöveges adat esetén előfordulhat, hogy egy file megnyitása hosszú másodpercekre befagyasztja a program futását. Ez a jelenség főként a CPC 464-en gyakori és a szakzsargonban 'szemétgyűjtés'-nek nevezik. Ahhoz, hogy megértsük, miféle szemetet gyűjt a CPC, meg kell ismerkedni a szöveges típusú változók tárolásával és a file-megnyitás mechanizmusával. A CPC BASIC, más BASIC verziókhoz hasonlóan, a numerikus változókat a programszöveg után, a szöveges típusú változókat pedig a BASIC terület felső részén tárolja. File-megnyitáskor a gép 4 Kbyte szabad területet keres a file-pufferek számára. A file-ba való kiírás és beolvasás 2 Kbyte-os egységekben történik, minden adatkivitel és beolvasás ezekbe a pufferekbe irányul. A külső tárba csak akkor kerülnek ki az adatok, ha a puffer megtelt vagy a file-t lezártuk. Beolvasáskor is egyszerre két kilobyte-nyi adat jön be a pufferbe, az 'INPUT #9' tulajdonképpen onnan kapja az olvasnivalót. Ha a puffert olvasással kiürítettük, egy újabb puffernyi adatot vesz át a gép a külső tárból. Ha csak egy file-t nyitunk meg, a CPC akkor is két puffert hoz létre. A 4 Kbyte-os puffernek mindig a BASIC terület legtetején keres helyet, ott pedig a szöveges változók vannak. Keres számukra egy szabad területet, átpakolja őket, közben ellenőrzi, hogy mindegyikre szükség van-e, átállítja az összes velük kapcsolatos mutatót ... ez pedig eltart egy ideig. A megoldás egyszerű: létre kell hozni egy file-pufferterületet a szöveges változók bevezetése előtt és be kell állítani a BASIC terület tetejét úgy, hogy az általunk létrehozott puffer alá mutasson. A következő programsort célszerű minden file-kezelő program első soraként alkalmazni:

```
openout "cica": memory himem-1: closeout cica
```

A "cica" helyett természetesen tetszőleges nevű álfile-t is meg lehet nyitni, annak semmiféle maradandó hatása nem lesz. A két új kulcsszóval a Harmadik szinten ismerkedünk meg részletesen.

▶▶▶ A szemétgyűjtéssel függ össze az is, hogy a CPC néha elfelejti az 'INPUT'-tal bevett file-nevet. Nem mindig működik például a következő két sor:

```
INPUT file$
OPENIN file$
```

Itt is egyszerű a megoldás: a 'file$'-t adjuk hozzá egy üres szöveghez és a CPC-nek mindjárt eszébe jut a file-név:

```
INPUT file$
OPENIN ""+file$
```

A file-kezelés legegyszerűbben gyakorlat útján sajátítható el. Gyakorlati például szolgáljon a következő mini adatbank, amelyben ismerőseink adatait tárolhatjuk. A program rendkívül egyszerű, kommentárt nem igényel, a változónevek önmagukat magyarázzák. Egyedül a 70-90-es sorok kis trükkjét ismertetjük. Azért, hogy a program kazettás és lemezes rendszeren egyaránt fusson - és ne a felhasználót kelljen zavarni a külső tárra vonatkozó kérdésekkel - a 80-as sor megpróbál lemezre kapcsolni. Ha ez sikerül, akkor a 'disk' változó értéke -1 lesz. Ka-

zetta esetén a parancs szintaxis hibához vezet, a 70-es sorban álló 'ON ERROR GOTO 90' a 90-es sorra irányitja a programot, ahol a 'disk'-et 0-ra állitjuk. Mindkét esetben a program a 100-as soron folytatódik. A maximálisan használható adatblokkok számát a 'maxadat' változó tartalmazza, ezt a 140-es sorban lehet módositani. A program több file-t is tud kezelni: amennyiben a "File nev ?" kérdésre csak az [ENTER]-t nyomja le, a file neve a "miniadat.dat" lesz, ezt a 'file$' változó tartalmazza, szintén a 140-es sorban módositható. A mini adatban egyszerűen bővithető, további szolgáltatások (keresés, rendezés stb.) bevezetését az Olvasóra bizzuk.

```
10 REM ******************
20 REM *                  *
30 REM * MINI ADATBANK *
40 REM *                  *
50 REM ******************
60 OPENOUT "semmi":MEMORY HIMEM-1: CLOSEOUT
70 ON ERROR GOTO 90
80 !DISC: disk = -1: GOTO 100
90 disk = 0: RESUME 100
100 ON ERROR GOTO 1200
110 MODE 1: INK 3,0,26
120 ON BREAK GOSUB 270
130 nev=1 : varos=2 : utca=3 : iranyszam=4 : telefon=5 : megjegy
zes=6
140 maxadat = 100 : DIM adat$(maxadat,6) : ez = 1 : file$="minia
dat.dat"
150 MODE 1:PRINT:PRINT: PRINT TAB(6) "** MINI ADATBANK **": PRIN
T
160 PRINT TAB(6)"Valasszon:" : PRINT
170 PRINT TAB(6)"1...Adatokat beolvasni"
180 PRINT TAB(6)"2...Adatokat kimenteni"
190 PRINT TAB(6)"3...Uj adatot felvenni"
200 PRINT TAB(6)"4...Adatot eltavolitani"
210 PRINT TAB(6)"5...Adatot megjeleniteni"
220 PRINT TAB(6)"6...Neveket megjeleniteni"
230 be$ = INKEY$ : IF be$ = "" THEN 230 ELSE k = ASC(be$)-48
240 IF k<1 OR k>6 THEN uzenet$ = "Rossz valasztas!" : GOSUB 1150
 : GOTO 230
```



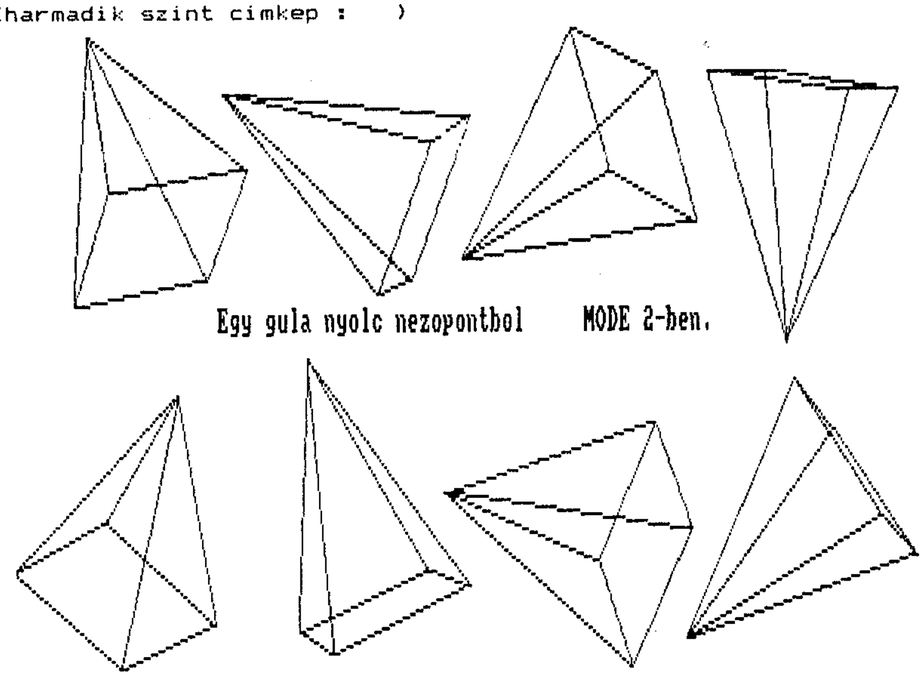
```
250 ON k GOSUB 320,460,650,860,950,1080
260 GOTO 150
270 CLS : PRINT "[ESC]-et nyomta le!"
280 PRINT "Ki akarja menteni az adatokat?"
290 be$ = INKEY$ : IF be$ = "" THEN 290 ELSE be$ = LOWER$(be$)
300 IF be$ <> "n" THEN GOSUB 460
310 END
320 CLS : PRINT "Adatokot beolvasni":PRINT
330 INPUT "File nev ";f$
340 IF f$="" THEN f$=file$
350 PRINT
360 OPENIN ""+f$
370 FOR ez = 1 TO maxadat
380 FOR j = 1 TO 6
390 INPUT #9,adat$(ez,j)
400 NEXT j
410 IF EOF THEN 440
420 NEXT ez
430 uzenet$ = "Az adatbank megtelt!" : GOSUB 1150
440 CLOSEIN
450 RETURN
460 CLS : PRINT "Adatokat kimenteni"
470 IF disk THEN 530
480 PRINT : PRINT "Adja be a kazetta sebesseget!"
490 PRINT "0... biztonsagos"
500 PRINT "1..... gyors"
510 be$ = INKEY$ : IF be$ = "" THEN 510
520 IF be$ = "0" THEN SPEED WRITE 0 ELSE SPEED WRITE 1
530 PRINT
540 INPUT "File nev "; f$
550 IF f$="" THEN f$=file$
560 OPENOUT ""+f$
570 FOR ez = 1 TO maxadat
580 IF adat$(ez,nev) = "" THEN 620
590 FOR j = 1 TO 6
600 WRITE #9,adat$(ez,j)
610 NEXT j
620 NEXT ez
630 CLOSEOUT
640 RETURN
```

```
650 CLS : ez = 0
660 PRINT "Uj adatot felvenni" : PRINT
670 PRINT "Valaszoljon:" :PRINT
680 INPUT "NEV";n$
690 IF n$ = "" THEN uzenet$ = "Nevet meg kell adni!" : GOSUB 115
0 : GOTO 850
700 FOR j = maxadat TO 1 STEP -1
710 IF adat$(j,nev) = "" THEN ez = j
720 IF adat$(j,nev) = n$ THEN uzenet$ = "Ilyen nev mar van!" :GO
SUB 1150 : GOTO 850
730 NEXT j
740 IF ez = 0 THEN uzenet$ = "Adatbank megtelt! : GOSUB 7000 : G
OTO 3190
750 adat$(ez,nev) = n$
760 LOCATE 1,6 :PRINT CHR$(20);
770 INPUT "VAROS";adat$(ez,varos)
780 INPUT "UTCA";adat$(ez,utca)
790 INPUT "IRANY.SZAM";adat$(ez,iranyszam)
800 INPUT "TELEFON";adat$(ez,telefon)
810 INPUT "MEGJEGYZES";adat$(ez,megjegyzes)
820 PRINT : PRINT "Rendben van?"
830 be$ = INKEY$ : IF be$ = "" THEN 830 ELSE be$ = LOWER$(be$)
840 IF be$ = "n" THEN PRINT : GOTO 760
850 RETURN
860 CLS : PRINT "Adatot eltavolitani" : PRINT
870 INPUT "Kerem a nevet ";n$ : PRINT
880 FOR ez =1 TO maxadat
890 IF adat$(ez,nev) = n$ THEN adat$(ez,nev) = "" : GOTO 920
900 NEXT ez
910 uzenet$ = "Ilyen nev nem is volt!" : GOSUB 1150 : GOTO 940
920 PRINT : PEN 3:PRINT n$;: PEN 1: PRINT" eltavolitva"
930 FOR j = 1 TO 2500 : NEXT j
940 RETURN
950 CLS : PRINT "Adatot megjeleniteni" : PRINT
960 INPUT "Kerem a nevet ";n$ : PRINT
970 FOR ez = 1 TO maxadat
980 IF adat$(ez,nev)= n$ THEN 1010
990 NEXT ez
1000 uzenet$ = "Ilyen nev nincs!" : GOSUB 1150 : GOTO 1070
1010 PRINT "VAROS",adat$(ez,varos)
```



```
1020 PRINT "UTCA",adat$(ez,utca)
1030 PRINT "IRANY.SZAM", adat$(ez,iranyszam,
1040 PRINT "TELEFON",adat$(ez,telefon)
1050 PRINT "MEGJEGYZES",adat$(ez,megjegyzes)
1060 IF INKEY$ = "" THEN 1060
1070 RETURN
1080 CLS
1090 FOR ez = 1 TO maxadat
1100 IF adat$(ez,nev) <> "" THEN WRITE adat$(ez,nev)
1110 NEXT ez
1120 PRINT : PRINT "EOF"
1130 FOR j = 1 TO 2500 : NEXT j
1140 RETURN
1150 LOCATE 2,24: PEN 3:PRINT"*** "; uzenet$ ; " ***"; : PEN 1
1160 IF INKEY$<>"" THEN 1160
1170 WHILE INKEY$="": WEND
1180 LOCATE 2,24:PRINT CHR$(20);
1190 RETURN
1200 uzenet$="Hiba ... probalja meg ujra!": GOSUB 1150: RESUME
```

(harmadik szint cimkep :)



Egy gula nyolc nezopontbol MODE 2-ben.

H a r m a d i k s z i n t

A Harmadik szint a CPC belső életével foglalkozik. Feltételezi, hogy az Olvasó már járatos a Z80-as mikroprocesszor programozásában. Ha ez nem így van, a Függelékben található irodalomjegyzék magyar nyelvű könyveket is felsorol, amelyek közérthetően vezetnek be a gépi kódú programozás rejtelmeibe.

A Harmadik szint gépi kódú programjai természetesen előképzettség nélkül is használhatók: assembler híján a gépi kódú rutinok a BASIC program DATA soraiból kerülnek be a memóriába.

Az önállóan futtatható programok:

3.1 GÉPI KÓDÚ PROGRAMOZÁS A CPC-N

A CPC BASIC-je egyike a leggyorsabbaknak, sebesség tekintetében még az IBM mögött sem marad el. A BASIC gyorsasága nemcsak a 4 MHz órajellel működő Z80A CPU-nak köszönhető, hanem elsősorban a jól megírt, parancsokban gazdag BASIC interpreternek. Az elmondottakból következik, hogy a CPC-n csak korlátozott mértékben van szükség gépi kódú programok használatára a végrehajtás sebességének fokozására. Gépi kódban főként az olyan programrészeket érdemes megírni, amelyek még a CPC-n is lassan futnának: ilyen a közvetlen képernyőkezelés, rendező rutinok stb. Csak gépi kódban oldhatók meg olyan feladatok, amelyek a CPC hardverjét nem rendeltetésszerűen kívánják használni: a hanggenerátort beszéltetni akarják vagy egynél több képernyőüzemmódot kívánnak alkalmazni. A CPC gépi kódú programozás tekintetében rendkívül nyitott gép, a FIRMWARE® több száz elemi rutinja hívható meg akár közvetlenül BASIC-ből is. Ezek közül a legfontosabbak megtalálhatók a következő pontban.

3.1.1 Gépi kódú programok elhelyezése

Új kulcsszavak:

FRE
HIMEM
MEMORY
PEEK
POKE

A CPC 64 Kbyte RAM-mal és 32 Kbyte ROM-mal rendelkezik. A Z80-as CPU egyszerre csak 64 Kbyte memóriát tud megcímezni; a RAM és ROM közötti átkapcsolás bonyolult - bár a felhasználó számára átlátszó - módon történik a Z80-as RST utasításaival. BASIC-ből a 64 Kbyte RAM címezhető közvetlenül. A BASIC program számára szabad terület nagyságát a **'FRE'** függvénnyel lehet lekérdezni. A függvénynek egy álargumentumot kell megadni, ez egyaránt lehet numerikus és szöveges típusú. A szöveges típusú álargumentum esetében a szabad tárterület kiszámítása előtt az interpreter a szöveges típusú változókat összefüggő területre tömöríti és a feleslegeseket kidobálja, vagyis elvégzi a szemétgyűjtést is:

```
print fre(0)
 42249                          - lemezes rendszeren bekapcsolás után
                                  (kazettás rendszeren: 43533)

a$="cica": a$="kutya": a$="teve": a$="nyul"
print fre(numerikus.valtozo)    - mennyi a szabad tárterület?
 42225                          - felesleges 'a$'-ok megvannak
                                  és helyet foglalnak
print fre("szoveges.valtozo")   - és így mennyi a szabad byte?
 42238                          - felesleges 'a$'-ok kidobálva
                                  (szemétgyűjtés történt)
```

A BASIC terület felső határa a **'HIMEM'** rendszerváltozóval kérdezhető le:

```
print himem, hex$(himem)
 42619       A67B               - lemezes rendszeren
```

A megkapott cím a BASIC által használt legmagasabb cím. Felette kezdődik a 'SYMBOL AFTER' utasítással az áttervezhető karakterek számára fenntartott terület, karakterenként 8 byte. Amint ismeretes, alapállapotban a gép 240 - 255-ös kódszámú karakterei tervezhetők át. A HIMEM+1 cimen tehát a 240-es karakter felső sora található. Ha a 'SYMBOL AFTER' utasítással több karakter mátrixát* másoltatjuk át a RAM-ba, hogy azok áttervezhetők legyenek, a HIMEM is automatikusan lejjebb megy:

```
symbol after 32                 - mindent át akarok tervezni
```

Most a HIMEM+1 címem a 32-es karakter felső sorának mintája található. RAM-ba írni a **'POKE'** utasítással lehet, a RAM cella tartalmát pedig a **'PEEK'** függvénnyel kérdezhetjük le:

```
        POKE cím, érték          cím: 0 - 65535
        tartalom = PEEK(cím)     érték: 0 - 255

print peek(himem+1)             - szóköz felső sorának mintája
 0                              - a szóköz felső sora üres
poke himem+1, &X01010101        - szaggatott vonal mintája
Ready
```


Nyomja le néhányszor a szóközbillentyűt, hogy meggyőződjék róla: valóban szaggatott vonal lett-e a szóközkarakter felső sorából. Az eredeti állapot visszaállításához elég az előbbi címre (HIMEM+1) egy nullát 'POKE'-olni.

Gépi kódú programok elhelyezésére olyan helyet kell találni, ahova a BASIC nem nyúlhat. Ez a hely csak a HIMEM felett, de egyben az áttervezhető karakterek kezdete alatt lehet. Jelen pillanatban a kettő egymás mellett van. A HIMEM-et a **'MEMORY'** utasítás állitja lejjebb, az utasítás paramétere a BASIC által használható legfelső memóriacím.

```
print himem
 40955                                  - SYMBOL AFTER 32 után
memory 39999                            - 40000-től  40955-ig a miénk
Ready
print himem
 39999                                  - ez még a BASIC területe
```

▶▶▶ A HIMEM beállitása után nem lehetséges a 'SYMBOL AFTER' utasítás újbóli használata, a gép az 'Improper argument' üzenettel tagadja meg a végrehajtását. A HIMEM-et nem állíthatjuk olyan alacsonyra vagy magasra, hogy az hibát okozhasson a rendszer működésében. Az interpreter a 'Memory full' hibaüzenettel akadályozza meg a merényletet.

Gépi kódú programjainkat tehát a 'MEMORY' utasítással beállított HIMEM fölött helyezhetjük el. A legegyszerűbb megoldás a gépi kódú utasításokat hexadecimális formában 'DATA' sorokban elhelyezni, majd onnan egy ciklussal a HIMEM fölé 'POKE'-olni.

```
10 rem PELDA GEPI KOD ELHELYEZESERE
20 rem
30 memory &9fff                         - &a000-tól a miénk
40 restore 1000: szamlalo = 0           - ott vannak a kódok
50 read kod$: if kod$="vege" then 90    - egyet beolvas, utolsó?
60 poke &a000+szamlalo, val("&"+kod$)   - számmá alakit, elhelyez
70 szamlalo = szamlalo+1 : goto 50      - növeli a memóriamutatót
80 rem
90 call &a000                           - meghivja a rutint
```

```
100 end
999 rem a gépi kódú program kódjai:      - Z80-as mnemonikokkal:
1000 data 21,01,01                      - ld hl,&0101   (x,y pos.)
1010 data cd,75,bb                      - call &bb75    (kurzor)
1020 data 3e,41                         - ld a,&41      (="A")
1030 data cd,5d,bb                      - call &bb5d    (kiírja)
1040 data c9, vege                      - ret           (vissza!)
```

A program egy gépi kódú rutint hív meg, amelyet előzőleg az &A000 címtől helyezett el. A rutin a maga részéről két FIRMWARE rutint hív meg: &BB75 pozicionálja a kurzort a H és L regiszterekkel megadott koordinátára, &BB5D karakterként megjeleníti az A regiszter tartalmát. Egy 'A' betű jelenik meg tehát a képernyő bal felső sarkában.

Rövidke gépi kódú programunk jól működik. Hasonlóképpen bármikor betölthető és meghívható, azonban hosszabb programoknál ez nem célszerű. Mentsük ki a memóriában található gépi kódú programrészt önállóan úgy, hogy az bármikor közvetlenül betölthető legyen! A kezdőcímet magunk határoztuk meg (&A000), a byte-ok számát egy híján még tartalmazza a 'szamlalo':

```
SAVE "gepikod1.bin", b, &a000, szamlalo+1
```

Ahhoz, hogy gépi kódú rutinunkat más BASIC programokban is felhasználhassuk, elegendő a HIMEM-et a kezdő cím alá állítani, betölteni a programot és meghívni a rutint:

```
10 rem PELDA GEPI KODU RUTIN BETOLTESERE
20 rem
30 memory &9fff                         - HIMEM-et &A000 alá!
40 load "gepikod1.bin", &a000           - töltsd be ezt, ide!
50 rem bent van, hivhato
60 call &a000                           - meghívja a rutint
70 end
```

Az ismertetett módszerrel elhelyezhető, kipróbálható, kimenthető és újra beolvasható a könyvben szereplő összes gépi kódú program. A betöltéshez nem szükséges megadni a töltőcímet, a CPC ezt a betöltendő program fejlécéből* megtudja és töltőcím

megadása nélkül is oda kerül a program. Ha a kód relokálható*, azt tetszőleges címre tölthetjük a töltőcím megadásával.

3.1.2 A CPC tárfelosztása

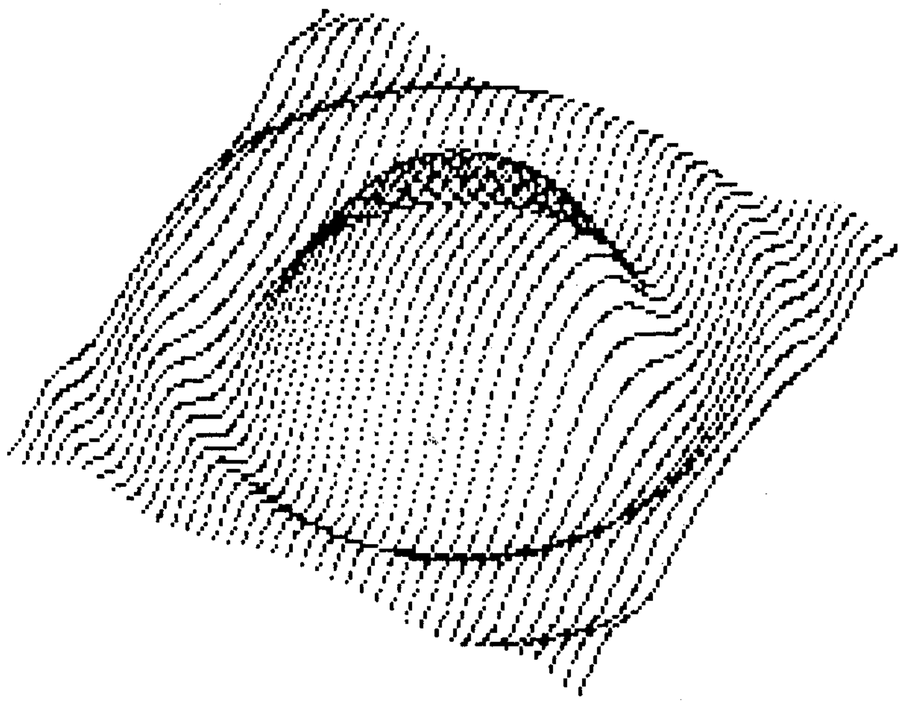
Amint már szó volt róla, a CPC 64 Kbyte RAM memóriája a BASIC programból mindig hozzáférhető: minden BASIC-ben kiadott 'PEEK' és 'POKE' a RAM-ra irányul, oda ír vagy onnan olvas be. A CPC ezen kívül 32 Kbyte csak olvasható tárral is rendelkezik. 16 Kbyte ROM található a &0000-&3FFF tartományban, ez az alsó vagy FIRMWARE ROM, itt találhatók a rendszer működéséhez szükséges alapvető rutinok, a karakterkészlet és a valós számokkal dolgozó aritmetikai rutinok. A FIRMWARE rutinjait a felhasználó egy ún. ugrótáblán keresztül hívhatja meg. Ugyancsak 16 Kbyte ROM található a &C000-&FFFF tartományban, ez a felső, vagy BASIC ROM, a BASIC interpretert tartalmazza. A BASIC interpreter rutinjai a felhasználó számára egyszerű módon nem hozzáférhetők.

A CPC felső ROM-jával párhuzamba, megfelelő hardver segítségével, még 252 (!) ún. bővítő ROM kapcsolható. A rendszer képes ezek kezelésére, így egyszerre 252*16 Kbyte, azaz 4 Mbyte ROM kapacitású gép is kiépíthető. A lemezes rendszerek már tartalmaznak egy ilyen bővítő ROM-ot: az AMSDOS, a CP/M egyes részei, valamint a LOGO* interpreter egyes részei találhatók benne. A memória ilyen bonyolult felépítésénél gépi kódú programozás szempontjából különlegesen fontos a középső 32 Kbyte RAM (&4000-&BFFF), mivel ez a terület biztosan RAM, sohasem lapozódik el. E terület felső részét a rendszer használja fel olyan adatok és címek tárolására, amelyeket mindig látnia kell. Az alsó és felső ROM-ba kukucskálhatunk egy olyan gépi kódú monitor vagy disassembler segítségével, amely a középső 32 Kbyte RAM-ba tölthető. Az eljárás a következő:

- monitort vagy disassemblert &4000-&BFFF közé tölteni
- a középső RAM egy szabad területére beírni vele az alábbi utasításokat:

```
ide: CD 00 B9       CALL &B900          - felső ROM be
     CD 06 B9       CALL &B906          - alsó  ROM be
     C3 .. ..       JP monitor          - indits újra!
```

- a monitorból az 'ide' címre ugratni (JUMP) és a program utolsó utasítása újra beugrik a monitorba vagy disassemblerbe, amely ezután már nem RAM-ot, hanem ROM-ot lát a &0000-&3FFF és &C000-&FFFF tartományban.



HAROMDIMENZIOS KALAP a grafikus ötletekből

A C P C **memóriatérképe**

R A M		ROM1	ROM2
==========	&FFFF	==========	==========
képernyő (16 Kbyte)		**BASIC INTERPRETER**	**AMSDOS ROM**
==========	&C000	==========	==========
rendszer verem			

FIRMWARE rutinok ugrótáblája			
----------	&BB00		
rendszer RAM			
----------	&AC00		
AMSDOS RAM			
----------	&A700		
áttervezhető karakterek (SYMBOL AFTER-rel beállítva)	--		
***** szabad terület *****			
MEMORY-val beállított	**HIMEM**		

↓ (OPENIN,OPENOUT puffer)			
↓ BASIC szöveges típusú			
↓ változók			
.....			
	&4000	==========	
↑ BASIC numerikus változók		**FIRMWARE R O M**	
↑			
↑ BASIC programszöveg			
----------	&0170	(alapvető rendszer-rutinok)	
RENDSZER RAM RST vektorok			
==========	&0000	==========	

3.1.3 Kommunikáció a BASIC és a gépi kód között

Új kulcsszavak: **CALL**
@

A gépi kódu rutinoknak BASIC-ből paramétereket lehet átadni, a rutinok maguk is módosíthatják a BASIC egyes változóit. Hogy a paraméterátadást eredményesen használhassuk, először meg kell ismerkedni azzal a móddal, ahogy a BASIC tárolja az egész típusú és szöveges változóit. Ezek lesznek ugyanis az átadandó paraméterek. A változók memóriában elfoglalt címét a '@' függvény adja meg, a függvény argumentumát (a változó nevét) nem szabad zárójelbe tenni. A '@' más BASIC verziók 'VARPTR' függvényének felel meg, segítségével BASIC-ben is közvetlenül hozzáférhetünk az egyes változók tartalmához:

```
a%=1: print peek(@a%)        - itt egy változó, mi van a címén?
 1                           - természetesen az értéke
poke @a%,2 : print a%        - írjunk be a címére 2-t!
 2                           - megváltozott az értéke
```

Az egész típusú változókat a BASIC 2 byte-on tárolja, LSB", MSB" sorrendben. Az előbb az LSB-t módosítottuk, próbáljuk meg ugyanezt az MSB-vel:

```
poke @a%+1,2 : print a%      - pozitív egész típusú változó
 514                           értéke = LSB + 256 * MSB
```

A szöveges típusú változók a memóriában két helyen is szerepelnek: a leírójuk első byte-ja tartalmazza a változó hosszát, a következő két byte (LSB, MSB sorrendben) a változó címét. Esetükben a '@' a leíró első byte-jának a címét adja meg:

```
a$="A" : print peek(@a$)     - itt egy szöveg, mi a hossza?
 1
cime = peek(@a$+1)+256*peek(@a$+2)    - a címét így számítjuk ki
poke cime,66                          - 'B' kódját tesszük oda
print a$
B
```


Gépi kódú rutint a 'CALL' utasítással hívhatunk meg BASIC-ből. A rutint záró Z80-as RET utasítás a 'CALL'-t követő BASIC utasításnak adja vissza a vezérlést. A gépi kódú rutinnak maximálisan 32 paraméter adható át a 'CALL' segítségével.

CALL cím, paraméter, paraméter, paraméter, paraméter,...

A 'cím' egész típusú numerikus kifejezés lehet, a 'paraméter' tetszőleges típusú változó vagy konstans. A gépi kódú rutin belépéskor az A regiszterben kapja meg a paraméterek számát. Az IX regiszterpár egy veremterületre mutat, ahol a paraméterek találhatók. Minden paraméter két byte-on kerül be az átadó verembe LSB, MSB sorrendben. A paraméterek átadására a következő szabályok vonatkoznak:

- egész típusú konstansoknak és változóknak az értéke kerül az átadó verembe,
- valós típusú konstansok és változók egész típusúra konvertált értéke kerül az átadó verembe,
- tetszőleges típusú változó címe adható át, ha a változó nevét '@' előzi meg. A numerikus változók valódi címe kerül átadásra, a szöveges típusú változók leírójának címét adja meg a '@'.

Mivel LIFO (utoljára be - elsőként ki) rendszerű veremről van szó, az IX regiszterpár az utoljára átvett paraméterre mutat. A paraméterek átadását és átvételét a következő két kis program mutatja be:

```
10 rem A HANGGENERATOR REGISZTEREI
20 rem közvetlenul kezelhetok a rutinnal
30 rem
40 memory &9fff
50 read a$: if a$="vege" then 90
60 poke &a000+szamlalo, val("&"+a$)
70 szamlalo=szamlalo+1: goto 50
80 rem
90 print "HANGREGISZTER rutin betoltve"
100 print "hivhato: call &a000,regiszter,ertek"
110 end
```

```
1000 data dd,4e,00                    - ld c,(ix+0)
1010 data dd,7e,02                    - ld a,(ix+2)
1020 data cd,34,bd                    - call &bd34
1030 data c9, vege                    - ret
```

A hanggenerátor regiszterei közvetlenül nehezen érhetők el, ezért a &BD34 című FIRMWARE rutint hívjuk segítségül. E rutin a kívánt regiszterbe a megadott értéket tölti. A hanggenerátor regiszterének sorszámát a Z80-as A regiszterében kell átadni, a töltendő értéket pedig a C regiszter tartalmazza. A 'CALL' utasítás végrehajtása során két paraméter kerül be az átadó verembe, amelyre az IX regiszterpár mutat:

```
'ertek' LSB        -->   (IX+0)
'ertek' MSB        -->   (IX+1)
'regiszter' LSB    -->   (IX+2)
'regiszter' MSB    -->   (IX+3)
```

Figyeljük meg a paraméterek sorrendjét! Mivel mindkét paraméter egy byte-on ábrázolható, elég csak az LSB-jüket betölteni az A és a C regiszterbe és hívható a FIRMWARE rutin. Ezúttal nem használtuk ki a CPC paraméterátadásának azt a szolgáltatását, hogy a rutinba való belépéskor az A regiszter tartalmazza a paraméterek számát. Ez a szolgáltatás egyrészt rugalmassá teszi az átveendő paraméterek számát, másrészt lehetőséget nyújt elemi hibák kiküszöbölésére: ha például a rutin nem kapta meg a kellő számú paramétert, akkor azonnal visszatérhet a hívó programba.

Előző programunkat ki lehet törölni: a rutin a helyén van és ott is marad, amíg rá nem töltünk valami mást. A BASIC 'NEW' nem bántja a védett területet. Az alábbi programmal kipróbálhatjuk a HANGREGISZTER rutin működését:

```
10 hreg= &a000                        - így egyszerűbb!
20 call hreg,1,0                      - CALL cím,param1,param2
30 call hreg,0,120
40 call hrag,8,16
50 call hreg,7,190
60 call hreg,13,8
70 call hreg,12,16
```

hez. A memóriába töltött RSX parancsokat először inicializálni kell: CALL ini-rutin, könyvünkben található bővítéseknél ez 'CALL &A000' lesz. A rendszer csak ezután hajlandó megérteni a az új parancsokat. Ismeretlen nevű RSX parancsokra 'Unknown command' (ismeretlen parancs) hibajelzéssel reagál. Az inicializált új parancsok a következőképpen hívhatók:

|PARANCSNEV, paraméter, paraméter, ...

A parancsnevet vessző választja el az első paramétertől. A paraméterek átadására a 'CALL' utasításnál ismertetett szabályok érvényesek: szövegek közvetlenül nem adhatók át, változónak kell őket értékül adni, a változó leírójának a címe adható meg csak paraméterként. (Így már érthető, miért olyan nehézkes az AMSDOS parancsok használata CPC 464-en!)

A továbbiakban egy program illusztrálja az RSX parancsok programozását és használatát. A program lényegében egy BASIC rendező szubrutint helyettesít. Rendezésről eddig nem volt szó a CPC BASIC egyik szintjén sem, pedig sok különböző rendező algoritmus található meg egyes szakkönyvekben. Vannak egyszerű és lassú, valamint bonyolult, de gyorsabb algoritmusok. Szöveges változók (tömbök) ábécé szerinti rendezése azonban a leggyorsabb és legbonyolultabb algoritmussal is időpazarlás BASIC-ben. A szöveges változók új értékadása esetén a BASIC ugyanis meghagyja a régi szövegeket is, ami egy idő múlva a már ismert szemétgyűjtéshez vezet. Ez pedig az esetek többségében hosszabb ideig tart, mint maga a rendezés. Itt valójában csak a gépi kód segít. Az alább leírt rutin a legprimitívebb algoritmus alapján működik: az ún. buborékos rendezés módszerével két közvetlenül egymás mellett álló elemet vet egybe, szükség esetén felcseréli őket. Ezt ismételve a tömb egyszeri végigfutása után a legkisebb elem buborékként a tömb tetejére emelkedik: az utolsó elem lesz. Az egybevetések következő sorozata már eggyel kevesebb elemmel dolgozik, az egyre rövidülő végigfutások után a tömb rendezetté válik. Gépi kódban lehetőség van közvetlenül a szöveges változók leíróját kicserélni, ami jóval gyorsabb, mintha a teljes szövegeket kellene fizikailag áthelyezni.

Az alábbi lista assembly nyelven tartalmazza a programot,

ennek alapján könnyen elkészíthető a 'DATA' sorokba ágyazott kódlista is: a második oszlop hexadecimális számait kell csak átvenni. A számok 2 számjegyenként, vesszővel elválasztva kerülnek 'DATA' sorokba.

```
; RSX parancs: |SORT,@szöveges.tömb (első), ennyit
; szöveges tömböt abécé sorrendbe rendez (ASCII kód!)
; első paraméter: az első rendezendő elem leírója
; második paraméter: a rendezendő elemek száma -1
;                    (a megadott elemtől számítva)
;
A000              10          ORG  #A000
; az inicializáló rutint BASIC-ből meghívni: CALL &A000
A000  010AA0      30 RUTINI:  LD   BC,RSXTAB
A003  2113A0      40          LD   HL,CPCNEK
A006  CDD1BC      50          CALL #BCD1
A009  C9          60          RET
; RSX-tábla:
A00A  0FA0        70 RSXTAB:  DEFW PNEVEK
A00C  C31BA0      80          JP   SORT
A00F  534F52      90 PNEVEK:  DEFM "SOR"
A012  D4         100          DEFB "T"+#80
A013             110 CPCNEK:  DEFS 4
; 'DATA' sorokba itt 00,00,00,00 -t kell beírni!
; itt kezdődik az RSX parancs:
A01B  DD4E00     120 SORT:    LD   C,(IX+0)
A01E  DD4601     130          LD   B,(IX+1)
; BC-ben az elemek száma
A021  DD6E02     140          LD   L,(IX+2)
A024  DD6603     150          LD   H,(IX+3)
; HL-ben az első elem leírójának címe
A027  E5         160          PUSH HL
A028  DDE1       170          POP  IX
; a leíró címe az IX-be került, a nagyhurok:
A02A  C5         180 HUROK:   PUSH BC
A02B  DDE5       190          PUSH IX
A02D  FDE1       200          POP  IY
; a következő elem leírója 3 byte-tal nagyobb címen van:
A02F  C5         210 EGYSTR:  PUSH BC
A030  FD23       220          INC  IY
```



```
A032  FD23       230          INC  IY
A034  FD23       240          INC  IY
; IY mutat a következő elem leírójára
A036  DD4600     250          LD   B,(IX+0)
A039  FD4E00     260          LD   C,(IY+0)
A03C  DD6E01     270          LD   L,(IX+1)
A03F  DD6602     280          LD   H,(IX+2)
A042  FD5E01     290          LD   E,(IY+1)
A045  FD5602     300          LD   D,(IY+2)
; B,C : a stringek hossza, HL,DE : a stringek címe
; a két elem összehasonlítása:
A048  1A         310 HASONL:  LD   A,(DE)
A049  BE         320          CP   (HL)
A04A  380B       330          JR   C,CSERE
A04C  202D       340          JR   NZ,TOVABB
A04E  13         350          INC  DE
A04F  0D         360          DEC  C
A050  2805       370          JR   Z,CSERE
A052  23         380          INC  HL
A053  10F3       390          DJNZ HASONL
A055  1824       400          JR   TOVABB
; stringcseréhez elég a leírók tartalmát kicserélni:
A057  DD4600     410 CSERE:   LD   B,(IX+0)
A05A  DD6E01     420          LD   L,(IX+1)
A05D  DD6602     430          LD   H,(IX+2)
A060  FD4E00     440          LD   C,(IY+0)
A063  FD5E01     450          LD   E,(IY+1)
A066  FD5602     460          LD   D,(IY+2)
A069  DD7100     470          LD   (IX+0),C
A06C  DD7301     480          LD   (IX+1),E
A06F  DD7202     490          LD   (IX+2),D
A072  FD7000     500          LD   (IY+0),B
A075  FD7501     510          LD   (IY+1),L
A078  FD7402     520          LD   (IY+2),H
A07B  C1         530 TOVABB:  POP  BC
A07C  0B         540          DEC  BC
A07D  78         550          LD   A,B
A07E  B1         560          OR   C
A07F  20AE       570          JR   NZ,EGYSTR
A081  C1         580          POP  BC
```

```
; IX felkészül a következő elemre:
A082  DD23        590          INC   IX
A084  DD23        600          INC   IX
A086  DD23        610          INC   IX
; van még végigfutás? Igen, ha BC > 0
A088  0B          620          DEC   BC
A089  78          630          LD    A,B
A08A  B1          640          OR    C
A08B  209D        650          JR    NZ,HUROK
; nincs több: vissza a hívó programba:
A08D  C9          660          RET
```

Az új parancs használatához tételezzük fel, hogy van egy 201 elemes szöveges tömbünk 'tomb$(200)'. Az egész tömb a következő paranccsal rendezhető: |sort,@tomb$(0),200. Ha csak az első száz elemet kívánjuk rendezni: |sort,@tomb$(0),99.

▶▶▶ Gépi kódban nem számíthat a BASIC barátságos hibajelzéseire. Ha több elemet próbál meg rendezni, mint amennyi valójában van, a rendszer elszáll: nem segít más, csak a teljes újraindítás. Ha az új parancsra az 'Unknown command' hibajelzést kapja, akkor elfelejtette azt inicializálni: CALL &A000.

3.2 A CPC FIRMWARE fontosabb rutinjai

A CPC FIRMWARE az alsó ROM-ban helyezkedik el a &0000-&3FFF memóriaterületen. A FIRMWARE tartalmazza a gép működéséhez szükséges elemi rutinokat. Mivel a ROM-ban lévő rutinok közvetlenül nem érhetők el, a rutinok egy ugrótábla segítségével hívhatók. Az ugrótábla a &BB00 címen kezdődik, tehát RAM-ban található. Ez a megoldás lehetőséget nyújt egyrészt a rutinok hasznosítására úgy BASIC-ben, mint gépi kódú programokban, másrészt az ugrótábla egyes tételeinek átírásával megoldható egyedi perifériák kezelése is. Az ugrótábla egyben a CPC-k közötti kompatibilitás záloga is: a CPC 464, 664 és 6128 FIRMWARE-e és BASIC interpretere között óriási a különbség, azonban ha a rendszer rutinjait az ugrótáblán keresztül hívjuk, ez a különbség figyelmen kívül hagyható.

A rutinok leírásában a CPC BASIC hexadecimális jelölésének megfelelően a hexadecimális számokat '&' jel előzi meg. Minden rutin leírása a rutin angol elnevezésével és a beugrási címével kezdődik, ezt követi a rutin által végrehajtott cselekvés rövid ismertetése. Ha a rutinnak megfelel egy BASIC utasítás, ez az ismertetés után zárójelben található. A 'belépés' és 'kilépés' rovat a rutin által igényelt, ill. az általa megváltoztatott regisztereket tünteti fel a Z80-as CPU-nál megszokott jelöléssel.

3.2.1 A billentyűzettel kapcsolatos FIRMWARE rutinok

KM INITIALISE **&BB00**

A billentyűzet teljes inicializálását végzi. A gép bekapcsolása utáni állapotot hozza létre.

Belépés: -
Kilépés: AF, BC, DE, HL regisztereket megváltoztatja.

KM WAIT CHAR **&BB06**

A billentyűzetpufferből egy karaktert olvas ki. Ha a puffer üres, megvárja, amíg egy karakter beérkezik.

Belépés: -
Kilépés: A : beolvasott karakter, C-flag beállítva,
a többi regiszter változatlan.

KM READ CHAR **&BB09**

A billentyűzetpufferből egy karaktert kisérel meg kiolvasni. Akkor is visszatér, ha a puffer üres. (INKEY$)

Belépés: -
Kilépés: Ha talált karaktert: A : beolvasott karakter,
C-flag beállítva, többi regiszter változatlan.
Ha nem talált karaktert: C-flaget törli,
A megváltozik, a többi regiszter változatlan.

KM CHAR RETURN **&BB0C**

A KM READ CHAR és KM WAIT CHAR számára megadja a következő olvasandó karaktert. Ez a karakter prioritást élvez a billentyűzetpufferben található karakterek előtt.

Belépés: A : a visszaadandó karakter.
Kilépés: minden regiszter változatlan.

KM SET EXPAND **&BB0F**

Funkcióbillentyűhöz bővítő szöveget rendel. (KEY)

Belépés: B : a funkcióbillentyű kódszáma,
C : a bővítő szöveg hossza,
HL : a bővítő szöveg helye.
Kilépés: Ha a művelet rendben végbement: C-flag beállítva,
AF, BC, DE, HL megváltozik.
Érvénytelen kódszám és túl hosszú szöveg esetében
C-flag törölve, regiszterek mint fent.

KM EXP BUFFER **&BB15**

Funkcióbillentyűk szövegeinek tárolásához puffert hoz létre. Az eredeti puffer csak 120 karaktert tud tárolni.

Belépés: DE : puffercím,
HL : pufferhossz.
Kilépés: Ha a művelet rendben végbement: C-flag beállítva.
Ha puffer túl kicsi: C-flag törölve. Mindkét esetben AF, BC, DE, HL megváltozik.

KM TEST KEY **&BB1E**

Egy tetszőleges billetyű állapotáról informál. (INKEY)

Belépés: A : billentyű sorszáma.
Kilépés: Ha a billentyű le van nyomva: Z-flag törölve.
Ha a billentyű nincs lenyomva: Z-flag beállítva.
Mindkát esetben AF, HL megváltozik.

KM GET STATE **&BB21**

A [CAPS LOCK] és [SHIFT LOCK] billentyűk állapotáról ad tájékoztatást. (INKEY)

Belépés: -
Kilépés: L : [SHIFT LOCK] állapota: &00 = bekapcsolva,
H : [CAPS LOCK] állapota: &FF = kikapcsolva.
AF megváltozik, a többi regiszter változatlan.

KM GET JOYSTICK **&BB24**

A botkormány pillanatnyi állapotáról tájékoztat. A visszaadott értékeknek minden bitje a 'JOY' utasításnál leírt módon értelmezendő.

Belépés: -
Kilépés: A, H : 0. botkormány állapota,
L : 1. botkormány állapota,
A többi regiszter változatlan.

KM SET TRANSLATE **&BB27**
KM SET SHIFT **&BB2D**
KM SET CONTROL **&BB33**

A 'KEY DEF' utasításhoz hasonlóan e három rutinnal egy-egy billentyűhöz, annak sorszáma alapján, tetszőleges ASCII kód rendelhető hozzá. Az első rutin a magányosan lenyomott billentyű, a második a [SHIFT]-tel együtt lenyomott gomb, a harmadik pedig a [CTRL]-lal együtt lenyomott gomb által szolgáltatandó kódot határozza meg.

Belépés: A : átdefiniálandó billentyű sorszáma,
B : a hozzárendelendő kódszám.
Kilépés: minden regiszter változatlan.

KM SET REPEAT **&BB39**

A 'KEY DEF' utasítás első paraméteréhez hasonlóan egy billentyű önismétlő funkcióját kapcsolja ki vagy be.

Belépés: A : a billentyű sorszáma,
B : &FF = ismétlés engedélyezve,
&00 = ismétlés tilos.

KM SET DELAY **&BB3F**

A 'SPEED KEY' utasításhoz hasonlóan az önismétlő billentyűk késleltetési és ismétlési idejét állitja be.

Belépés: H : késleltetési idő,
L : ismétlési idő, mindkettő 1/50 másodpercben.
Kilépés: AF megváltozik, minden más regiszter változatlan.

3.2.2 A szöveges képernyővel kapcsolatos FIRMWARE rutinok

TXT INITIALIZE **&BB4E**

A szöveges képernyő minden paraméterét inicializálja. Nem módosít a 'MODE' és 'INK' utasítással beállított állapoton, nem törli a képernyőt, de a kurzort az 1,1 pozícióba állitja.

Belépés: -
Kilépés: minden regiszter változatlan.

TXT OUTPUT **&BB5A**

A szöveges képernyőn az aktuális kurzorpozíción megjeleníti az írható karaktert. A vezérlő karakternek megfelelő cselekvést végrehajtja. A megjelenítés ill. cselekvés a pillanatnyilag kiválasztott ablakra vonatkozik, a kurzorpozíció megfelelően módosul. (PRINT CHR$())

Belépés: A : kiküldendő karakter.
Kilépés: az összes regiszter változatlan.

TXT WR CHAR **&BB5D**

A szöveges képernyőre karaktert ír ki. A vezérlő karakterek is grafikusan jelennek meg. Vö. TXT OUTPUT.

Belépés: A : a kiküldendő karakter.
Kilépés: az összes regiszter megváltozik.

TXT RD CHAR **&BB60**

Az aktuális kurzorpozíción található karaktert megkisérli azonosítani és beolvasni. A CPC 664 és 6128-on megtalálható 'COPYCHR$' függvény megfelelője. A rutin az aktuális 'PEN'-nel irt normál és inverz karaktereket ismeri fel.

Belépés: -
Kilépés: Ha sikerült egy karaktert felismerni és beolvasni,
C-flag beállítva, A : beolvasott karakter.
Ha nem sikerült a karaktert azonosítani,
C-flag törölve, A : &00.
Mindkét esetben a többi regiszter változatlan.

TXT WIN ENABLE **&BB66**

A további szöveges megjelenítéshez ablakot határoz meg. Az ablak határait két szemben levő sarok pozíciójával lehet megadni. A sor- és oszloppozíciók 0-tól indulnak! A kurzor pozíciója a megnyitott ablak bal felső sarka lesz. (WINDOW)

Belépés: D : egyik sarok oszlopa, E : egyik sarok sora,
H : másik sarok oszlopa, L : másik sarok sora.
Kilépés: AF, BC, DE, HL megváltozik.

TXT CLEAR WINDOW **&BB6C**

Az aktuális szöveges ablakot a hozzátartozó papír színére törli. (CLS)

Belépés: -
Kilépés: AF, BC, DE, HL megváltozik.

TXT SET CURSOR **&BB75**

Az aktuálisan kiválasztott ablakban a kurzort a kívánt pozícióra állítja. (LOCATE)

Belépés: H : a kívánt oszlop száma
Kilépés: L : a kívánt sor száma

TXT GET CURSOR **&BB78**

A pillanatnyi ablak aktuális kurzorpozíciójáról tájékoztat. (POS, VPOS)

Belépés: -
Kilépés: H : a kurzor y-koordinátája (sor)
L : a kurzor x-koordinátája (oszlop)

TXT CUR ON **&BB81**

A szöveges kurzort bekapcsolja, ha ezt a rendszer engedélyezi. A regisztereket változatlanul hagyja.

TXT CUR OFF **&BB84**

Az előző rutinnal bekapcsolt kurzort kikapcsolja.

TXT SET PEN **&BB90**

A további kiíráshoz használandó tollat választja ki. (PEN)

Belépés: A : a tollhoz rendelendő tinta sorszáma.
Kilépés: AF, HL megváltozik, a többi regiszter változatlan.

TXT SET PAPER **&BB96**

A kiírandó szöveg hátteréül papírt választ. (PAPER)

Belépés: A : a papírhoz rendelendő tinta színe.
Kilépés: AF, HL megváltozik, a többi regiszter változatlan.

TXT INVERSE **&BB9C**

A toll és a papír színét kicseréli. Az AF és HL regisztereket megváltoztatja. (PRINT CHR$(24))

TXT SET BACK **&BB9F**

A = 0 felülíró szöveges megjelenítésre kapcsol,
A <> 0 átlátszó szöveges megjelenítésre kapcsol.
Mindkét esetben megváltoztatja az AF, HL regisztereket.

3.2.3 A grafikus képernyővel kapcsolatos rutinok

GRA INITIALISE **&BBBA**

A grafikus képernyőt inicializálja, de nem törli. Grafikus toll és papír színét, a koordinátarendszert, a grafikus kurzor helyét stb. a kezdeti értékre állítja.

Belépés: -
Kilépés: AF, BC, DE, HL regiszterek megváltoznak.

GRA MOVE ABSOLUTE **&BBC0**

A grafikus kurzort a kívánt abszolút pozícióra állítja.

Belépés: DE : a felhasználói x-koordináta,
HL : a felhasználói y-koordináta.
KIlépés: AF, BC, DE, HL regiszterek megváltoznak.

GRA MOVE RELATIVE **&BBC3**

A grafikus kurzort aktuális pozíciójához képest x,y távolságra állítja. (MOVER)

Belépés: DE : előjeles távolság az x-tengelyen,
HL : előjeles távolság az y-tengelyen.
Kilépés: AF, BC, DE, HL regiszterek megváltoznak.

GRA SET PEN **&BBDE**

A grafikus toll színét állítja be. (GRAPHICS PEN)

Belépés: A : a tollhoz rendelendő tinta sorszáma.
Kilépés: AF megváltozik, a többi regiszter változatlan.

GRA SET PAPER **&BBE4**

A grafikus papír színét állítja be. (GRAPHICS PAPER)

Belépés: A : a papírhoz rendelendő tinta sorszáma.
Kilépés: AF megváltozik, a többi regiszter változatlan.

GRA PLOT ABSOLUTE **&BBEA**

Egy abszolút koordinátapontra pontot tesz ki. (PLOT)

Belépés: DE : x-koordináta, HL : y-koordináta.
Kilépés: AF, BC, DE, HL regiszterek megváltoznak.

GRA PLOT RELATIVE **&BBED**

A kurzor pillanatnyi pozíciójához képest x,y távolságra pontot tesz ki. (PLOTR)

Belépés: DE : előjeles távolság az x-tengelyen,
HL : előjeles távolság az y-tengelyen.
Kilépés: AF, BC, DE, HL regiszterek megváltoznak.

GRA TEST ABSOLUTE **&BBF0**

Egy abszolút koordinátapont színéről tájékoztat. (TEST)

Belépés: DE : a tesztelendő pont x-koordinátája,
HL : a tesztelendő pont y-koordinátája.
Kilépés: A : a pont színe (tintasorszám)
BC, DE, HL regiszterek megváltoznak.

GRA TEST RELATIVE **&BBF3**

A pillanatnyi grafikus kurzorpozícióhoz képest x,y távolságra levő pont színéről tájékoztat. (TESTR)

Belépés: DE : a pont előjeles távolsága az x-tengelyen,
HL : a pont előjeles távolsága az y-tengelyen.
Kilépés: A : a pont színe (tintasorszám),

BC, DE, HL regiszterek megváltoznak.

GRA LINE ABSOLUTE **&BBF6**

A pillanatnyi grafikus kurzorpozícióból kiinduló vonalat húz a megadott végpontig. (DRAW)

Belépés: DE : a végpont x-koordinátája,
HL : a végpont y-koordinátája.
Kilépés: AF, BC, DE, HL regiszterek megváltoznak.

GRA LINE RELATIVE **&BBF9**

A pillanatnyi grafikus kurzorpozícióból kiinduló vonalat húz az attól x,y távolságra eső végpontig. (DRAW)

Belépés: DE : a végpont előjeles távolsága az x-tengelyen,
HL : a végpont előjeles távolsága az y-tengelyen.
Kilépés: AF, BC, DE, HL regiszterek megváltoznak.

GRA WRITE CHAR **&BBFC**

Karaktert jelenít meg a pillanatnyi grafikus kurzorpozíción. A karakter színét a grafikus toll aktuális színe, a karakter hátterét a grafikus papír aktuális színe határozza meg. Minden karaktert megjelenít, a 0 - 31 kódszámúakat is.

Belépés: A : a megjelenítendő karakter kódszáma.
Kilépés: AF, BC, DE, HL regiszterek megváltoznak.

3.2.4 Képernyőkezeléssel kapcsolatos rutinok

SCR INITIALISE **&BBFF**

A képernyőt a gép bekapcsolása utáni állapotba állítja. A képernyő minden paraméterét inicializálja. Törli a képernyőt, de a szöveges kurzor pozíciója nem módosul. Az összes regiszter megváltozik.

SCR SET BASE &BC08

A képernyő kezdőcímét állítja be. A képernyő 16 Kbyte RAM területet foglal el, alapállapotban a &C000 címtől kezdve. Lehetőség van a képernyőt egy másik 16 Kbyte-os blokkban elhelyezni. A rendszer által használt RAM területek megsértése nélkül csak a &4000-től kezdődő blokk jöhet számításba. A rutin által áthelyezett képernyőt a rendszer szabályosan kezeli tovább.

Belépés: A : a képernyő számára kijelölt RAM terület MSB-je, &C0 vagy &40.
Kilépés: AF, HL megváltozik, a többi regiszter változatlan.

SCR GET LOCATION &BC0B

A képernyő aktuális báziscímét és eltolását adja meg. A báziscím &C000 vagy &4000 lehet, az eltolás a képernyő bal felső pontjának címét jelenti a báziscímhez képest. Az eltolásra a hardver görgetéskor kerül sor. A képernyő pillanatnyi kezdőcímét a báziscím + eltolás adja meg.

Belépés: -
Kilépés: A : a képernyő báziscímének MSB-je (&C0 vagy &40),
HL : a pillanatnyi eltolás értéke

SCR SET MODE &BC0E

A képernyő-üzemmódot állítja be a BASIC 'MODE' parancshoz hasonlóan. A paramétert (0, 1, 2) az A regiszterben kell megadni a rutin az összes regisztert megváltoztatja.

SCR CLEAR &BC14

A képernyőt a nullás tinta színére törli: a képernyő memóriát &00-val írja tele.

SCR CHAR POSITION &BC1A

A kívánt karakterpozíciónak megfelelő aktuális memóriacímet adja meg.

Belépés: H : fizikai y-pozíció (0-tól indul!)
L : fizikai x-pozíció (0-tól indul!)
Kilépés: HL : a megadott karakterpozíciónak megfelelő memóriacím (a karakter bal felső pontjának címe).
B : a karakterpozíció szélessége byte-okban (MODE 0-ban 4, MODE 1-ben 2, MODE 2-ben 1).

SCR DOT POSITION &BC1D

A kívánt abszolút grafikus koordinátapont (origó a bal alsó sarokban) memóriacímét adja meg. Tájékoztat a MODE-nak megfelelő maszkról is, amelyet egy pont megjelenítéséhez kell felhasználni. Paraméterként a pont fizikai koordinátáit kell megadni: minden egység egy képernyőpontnak felel meg.

Belépés: DE : a pont x-koordinátája (0 - 159, 319, 639)
HL : a pont y-koordinátája (0 - 199 !)
Kilépés: HL : a pont memóriacíme,
C : a képernyőpont maszkja.

SCR NEXT LINE &BC26

A megadott képernyőcím alatti pont memóriacímét adja vissza.

Belépés: HL : képernyőcím.
Kilépés: HL : a kiszámított memóriacím,
AF megváltozik, a többi regiszter változatlan.

SCR PREV LINE &BC29

A megadott képernyőcím feletti pont memóriacímét adja vissza.

Belépés: HL : képernyőcím.
Kilépés: HL : a kiszámított memóriacím,
AF megváltozik, a többi regiszter változatlan.

SCR SET INK &BC23

A megadott tintához két palettaszínt rendel. (INK)

Belépés: A : tintasorszám,
B : egyik palettaszín, C : másik palettaszín,
ha B = C, a tinta színe nem fog váltakozni.
Kilépés: AF, BC, DE, HL regiszterek megváltoznak.

SCR SET BORDER **&BC38**

A képernyő keretéhez két palettaszínt rendel. Paraméterei, az A regiszter kivételével, megegyeznek az előző rutin paramétereivel. (BORDER)

SCR SET FLASHING **&BC3E**

A tintákhoz rendelt két palettaszín váltakozásának idejét állítja be. (SPEED INK)

Belépés: H : az első szín időtartama 1/50 másodpercben,
L : a második szín időtartama 1/50 másodpercben.
Kilépés: AF, HL megváltozik, a többi regiszter változatlan.

SCR HW ROLL **&BC4D**

A teljes képernyőt egy karaktersorral felfelé vagy lefelé görgeti. A görgetést a képernyőkontroller egyes regisztereinek beállításával éri el, így a képernyőmemória kezdő címe megváltozik.

Belépés: B = 0 lefelé görget, B <> 0 felfelé görget.
Kilépés: AF, BC, DE, HL megváltozik.

SCR SW ROLL **&BC50**

A képernyő egy részét egy karaktersorral felfelé vagy lefelé görgeti. A görgetést a kívánt memóriatartomány fizikai átmásolásával végzi el. A paraméterként megadott koordináták számozása 0-tól indul.

Belépés: H : a képernyőrész bal oszlopa,
D : a képernyőrész jobb oszlopa,
L : a képernyőrész felső sora,

E : a képernyőrész alsó sora.
Kilépés: AF, BC, DE, HL megváltozik.

SCR ACCESS **&BC59**

A képernyőn megjelenítendő pontok és a háttér keresztezési módját állítja be. Alapállapotban a felülíró mód van érvényben. Az alábbiakban 'NEW' jelöli a megjelenő színt, 'OLD' az előzőleg ott lévő pixel színét, 'INK' pedig a megjelenítéshez használt tinta színét:

Belépés: A = 0 felülíró mód: NEW = INK
A = 1 XOR-mód: NEW = INK xor OLD
A = 2 AND-mód: NEW = INK and OLD
A = 3 OR-mód: NEW = INK or OLD
Kilépés: AF, BC, DE, HL megváltozik.

3.2.5 A ROM-ok kezelésével kapcsolatos rutinok

KL U ROM ENABLE **&B900**

A felső (&C000-&FFFF) ROM-ot engedélyezi. Az ott található BASIC interpreter rutinjai hívhatók meg ezután.

Belépés: -
Kilépés: A : tartalmazza az előző ROM-állapotot, annak visszaállításához ajánlatos megőrizni.

KL U ROM DISABLE **&B903**

Letiltja a felső ROM-ot. A &C000-&FFFF tartomány ezután a (képernyő-) RAM területe lesz. A regiszterhasználatot l. az előző rutinnál.

KL L ROM ENABLE **&B906**

Engedélyezi az alsó (&0000-&3FFF) ROM-ot. Az ott található FIRMWARE rutinjai hívhatók meg így közvetlenül. Nagyon nem ajánlatos a rutinokat közvetlenül hívni! Az ott található karakterkészletet viszont minden további nélkül ki lehet olvasni. A re-

giszterhasználatot l. a KL U ROM ENABLE rutinnál.

KL L ROM DISABLE **&B909**

Letiltja az alsó ROM-ot. A &0000-&3FFF tartomány ezután RAM terület lesz. A regiszterhasználatot l. a KL U ROM ENABLE rutinnál.

KL ROM RESTORE **&B90C**

Az előző ROM-állapotot visszaállítja. A fenti rutinok használata után tanácsos mindig igénybe venni.

Belépés: A : az előző ROM-állapot, l. a KL U ROM ENABLE-nél.

KL LDIR **&B91B**

Z80-as LDIR utasítást hajt végre. RAM-ból RAM-ba másol, függetlenül a ROM-ok pillanatnyi állapotától.

Belépés: megfelel az LDIR utasítás regiszterhasználatának.

3.2.6 Gyakrabban használt vegyes rendeltetésű rutinok

CAS START MOTOR **&BC6E**

Elindítja a beépitett (CPC 664 és 6128 esetén a géphez kapcsolt) magnó motorját.

CAS STOP MOTOR **&BC71**

Megállitja a beépitett vagy a géphez kapcsolt magnó motorját.

CAS CATALOG **&BC9B**

Katalógust készit a kazettán található programokról és a képernyőn megjeleniti azt. Lemezes rendszernél ugyanezt végzi el a lemezzel kapcsolatban. (CAT)

Belépés: DE : 2 Kbyte nagyságú pufferterületre mutat, amelyet a rutin szabadon felhasználhat.
Kilépés: az összes regiszter megváltozik.

SOUND RESET **&BCA7**

Inicializálja a hanggenerátort. Minden hangkiadást megszüntet. Az összes regisztert megváltoztatja.

MC SOUND REGISTER **&BD34**

A hanggenerátor kívánt regiszterébe a megadott értéket tölti. A gép hardverfelépitése miatt ez közvetlenül nehezen lenne megoldható.

Belépés: A : a hanggenerátor regiszterének sorszáma.
C : a regiszterbe töltendő érték.
Kilépés: AF, BC megváltozik, a többi regiszter változatlan.

KL TIME PLEASE **&BDOD**

A gép bekapcsolása óta eltelt időt adja meg 1/300 másodperces időegységekben. A gép belső órája 14.316.557 másodperc, azaz 166 nap eltelte után csordul túl. (TIME)

Belépés: -
Kilépés: DEHL : tartalmazza a bekapcsolás óta eltelt időt, D az MSB, L az LSB.

KL TIME SET **&BD10**

A gép belső óráját állitja be a DEHL regiszterpárokkal megadott értékre. Lehetőséget ad arra, hogy a belső órát tetszőleges időponttal inicializáljuk. A paraméterként átadandó DE és HL regiszterek használatát l. az előző rutinnál.

MC WAIT FLYBACK **&BD19**

A képernyőtartalmat megjelenitő katódsugár visszafutásának kezdetére vár. (FRAME)

MC PRINT CHAR **&BD2B**

A CPC CENTRONICS típusú nyomtató portján egy karaktert ad ki. Ha a nyomtató foglalt, egy ideig várakozik és újra megkísérli a karakter kiküldését.

Belépés: A : a nyomtatóra küldendő karakter (a 7. bit-et nem veszi figyelembe!).
Kilépés: C-flag beállítva, ha a karakter azonnal kiment,
C-flag törölve, ha várakozni kellett.

MC BUSY PRINTER **&BD2E**

Megvizsgálja, hogy a géphez kapcsolt nyomtató foglalt-e.

Belépés: -
Kilépés: C-flag beállítva, ha a nyomtató foglalt,
C-flag törölve, ha a nyomtató készen áll.

JUMP RESTORE **&BD37**

Visszaállítja a FIRMWARE ugrótábla eredeti állapotát. Használata akkor célszerű, ha egyes rutinokat valamilyen célból máshová irányítottunk át.

3.2.7 FIRMWARE rutinok hívása BASIC-ből

A cím csalóka, hiszen a BASIC 'CALL' utasítással nem lehet feltölteni a Z80-as regisztereit, ez pedig a FIRMWARE rutinok meghívásához feltétlenül szükséges. Az egyes rutinok által szolgáltatott adatok is a Z80-as CPU regisztereiben találhatók, így ezek átvétele sem lehetséges. Vannak azonban kivételek.

Az olyan FIRMWARE rutinok, amelyek nem várnak paramétereket a regiszterekben és nem is adnak vissza adatokat, minden további nélkül meghívhatók BASIC-ből is. Néhány példa:

CALL &BD19 megfelel a CPC 664 és 6128 'FRAME' utasításának,
de a 464-esen is használható.

CALL &BB9C felcseréli a toll és papir szinét, ahogy azt a BASIC 'PRINT CHR$(24)' is teszi.

Amint a 'CALL' utasítás paraméterátadásánál megtudtuk, az átadandó paraméterek száma az A regiszterbe kerül. Maximálisan 32 paraméter adható át, így lehetőség nyílik az A regisztert kisebb értékekkel feltölteni: annyi álparamétert kell megadni, amilyen értéket az A regiszternek átadni kívánunk. Szimulálható így a nagyobb CPC-k 'GRAPHICS PEN' vagy 'GRAPHICS PAPER' utasítása, amely a 464-esen hiányzik:

```
CALL &BBDE,0,0,0  = GRAPHICS PEN 3
CALL &BBE4,0,0    = GRAPHICS PAPER 2
```

Az összes regisztert feltölti a következő programban található gépi kódú rutin. A rutin hívása előtt az 'AF', 'BC', 'DE', 'HL' egész típusú BASIC változóknak kell értékül adni a kívánt paramétereket. A rutin nemcsak feltölti a Z80-as regisztereit és meghívja a FIRMVARE rutint, hanem visszatérés előtt a regeszterek tartalmát az említett változókba tölti, amelyek értéke azután egyszerűen lekérdezhető. A program a FIRMWARE rutinok kényelmes kipróbálását teszi lehetővé: a regiszterek pillanatnyi állapota állandóan a képernyőn látható, a kurzorbillentyűkkel bármely regiszter azonnal elérhető és hexadecimális számjegyekkel feltölthető. A FIRMWARE rutint a [CTRL][R] billentyűkkel hívjuk meg. Visszatérés után a képernyő egy gomb lenyomásáig változatlan marad, hogy az esetleges kiíró, rajzoló rutinok eredményét ne rontsa el az újra megjelenő regiszter-táblázat. Egy gomb lenyomára újra kirajzolódik a képernyő, amin most már a FIRMWARE rutin által visszaadott regiszterek tartalma is látható.

```
10 REM ************************
20 REM *                      *
30 REM * FIRMWARE RUTIN HIVO  *
40 REM *                      *
50 REM ************************
60 MEMORY &9FFF: GOSUB 840
70 '------------------------------------------------------------
80 DEFINT a-z: DIM reg$(4,5), reg(4)
```

```
90 bal =242: jobb =243: fel =240: le =241
100 alahuz$=STRING$(18,"_")+SPACE$(4)+STRING$(16,"_")+CHR$(10)
110 vk=11: vm=vk+4: fk=5: fm=fk+4*3
120  GOSUB 640 '                          nullaz
130 '-----------------------------------fo hurok
140 GOSUB 230 '                           fejlec, tabla
150 GOSUB 670 '                           regisztereket megjelenit
160 GOSUB 710 '                           visszakapott regiszterek
170 GOSUB 410 '                           beadas
180 GOSUB 580 '                           reg.feltoltes
190 CALL &A000,cim,@af,@bc,@de,@hl '      meghivja a rutint
200 GOSUB 790 '                           folytatni?
210 GOTO 140
220 '-----------------------------------tablakep
230 INK 0,13: INK 1,0: BORDER 13 : PEN 1
240 INK 2,0,26: INK 3,26,0: SPEED INK 20,20
250 MODE 1: PRINT"           CPC firmware rutin hivo"
260 PRINT: PRINT"         belepes        " CHR$(24)"  kilepes  "
CHR$(24)
270 LOCATE 1,5: PRINT"cim ="
280 PRINT alahuz$
290 PRINT"A F ="
300 PRINT alahuz$
310 PRINT"B C ="
320 PRINT alahuz$
330 PRINT"D E ="
340 PRINT alahuz$
350 PRINT"H L ="
360 PRINT alahuz$
370 LOCATE 1,20: PRINT"rutint meghivni         : [CTRL][R]"
380 LOCATE 1,22: PRINT"regisztereket nullazni : [CTRL][C]"
390 RETURN
400 '-----------------------------------beado szubrutin
410 x=vk: y=fk
420 LOCATE x,y : PEN 2:CALL &BB81
430 a$=INKEY$: IF a$="" THEN 430 ELSE CALL &BB84: PEN 1
440 a=ASC(UPPER$(a$)): IF a=&12 THEN RETURN
450 IF (a>47 AND a<58) OR (a>64 AND a<71) THEN 540 'hexa szam
460 IF a=13 THEN x=vk: a=le ELSE IF a=32 THEN a=jobb ELSE IF a=1
27 THEN a=bal
```



```
470 IF a=3 THEN GOSUB 640: GOSUB 670: GOTO 410
480 IF a>244 OR a<239 THEN PRINT CHR$(7);: GOTO 420
490 IF a=bal  AND x>vk THEN x=x-1 :IF x-vk=2 THEN x=vk+1
500 IF a=jobb AND x<vm THEN x=x+1 :IF x-vk=2 THEN x=vk+3
510 IF a=fel  AND y>fk THEN y=y-3
520 IF a=le   AND y<fm THEN y=y+3
530 GOTO 420
540 '-----------------------------------hexaszam
550 reg$(x-vk,(y-fk)/3)=CHR$(a)
560 PRINT CHR$(a);
570 a=jobb: GOTO 500
580 '-----------------------------------reg()-t feltolt
590 FOR i=0 TO 4
600 reg(i)=VAL("&h"+reg$(0,i)+reg$(1,i)+reg$(3,i)+reg$(4,i))
610 NEXT i
620 cim=reg(0): af=reg(1): bc=reg(2): de=reg(3): hl=reg(4)
630 RETURN
640 '-----------------------------------reg$()="0"
650 FOR i=0 TO 4: FOR j=0 TO 4: reg$(i,j)="0": NEXT j,i: RETURN
660 '
670 '-----------------------------------reg$-t kiir
680 FOR i=0 TO 4
690 LOCATE vk, fk+3*i: PRINT reg$(0,i) reg$(1,i) " " reg$(3,i) r
eg$(4,i)
700 NEXT i: RETURN
710 '-----------------------------------visszakapott reg.-eket kiir
720 PRINT CHR$(24);
730 LOCATE vk+15,fk+3: PRINT "  "LEFT$(HEX$(af,4),2) " " RIGHT$(
HEX$(af,4),2) "  "
740 LOCATE vk+15,fk+6: PRINT "  "LEFT$(HEX$(bc,4),2) " " RIGHT$(
HEX$(bc,4),2) "  "
750 LOCATE vk+15,fk+9: PRINT "  "LEFT$(HEX$(de,4),2) " " RIGHT$(
HEX$(de,4),2) "  "
760 LOCATE vk+15,fk+12:PRINT "  "LEFT$(HEX$(hl,4),2) " " RIGHT$(
HEX$(hl,4),2) "  "
770 PRINT CHR$(24);
780 RETURN
790 '-----------------------------------folytatni ?
800 LOCATE 1,25: PEN 3 :PRINT CHR$(24);
810 PRINT " RUTINBOL VISSZA. TOVABB: [ ENTER ]    ";
```

```
820 PRINT CHR$(24);:PEN 1: CALL &BB06
830 RETURN
840 '------------------------------ hivo rutint betolt &a000-tol
850 checksum=0: cim=&A000-1: RESTORE 900
860 READ a$: IF a$="vege" THEN 880 ELSE a=VAL("&h"+a$): cim=cim+
1
870 checksum=checksum + a: POKE cim,a: GOTO 860
880 IF checksum=15077 THEN RETURN
890 PRINT"Hiba van a HEX szamokat tartalmazo DATA sorokban!": ST
OP
900 DATA F3, DD, 22, 71, A0, ED, 73, 73, A0, FD, 21, 75, A0, 06,
 04, DD, 6E, 00, DD, 66, 01, 7E, FD, 77, 00, 23, FD, 23, 7E, FD,
 77, 00, FD, 23, DD, 23, DD, 23
910 DATA 10, E7, DD, 6E, 00, DD, 66, 01, 22, 3E, A0, 31, 75, A0,
 E1, D1, C1, F1, ED, 7B, 73, A0, FB, CD, 18, BB, F3, 31, 7D, A0,
 F5, C5, D5, E5, ED, 7B, 73, A0
920 DATA DD, 2A, 71, A0, FD, 21, 75, A0, 06, 04, DD, 6E, 00, DD,
 66, 01, FD, 7E, 00, 77, 23, FD, 23, FD, 7E, 00, 77, FD, 23, DD,
 23, DD, 23, 10, E7, FB, C9, vege
```

Ha saját programjaiban is használni kívánja a regiszter-feltöltő rutinhívó rutint, a 840-es sorral kezdődő részt kell átvennie. Ne feledkezzen meg a regisztervváltozókat egész típusúnak deklarálni és a HIMEM-et &9FFF-re állítani. A regiszterek feltöltése LSB, MSB alapon történik, tehát ha az A regiszterbe &40-et kívánunk tölteni, akkor AF%=&40*256. H-ba 10, L-be 20: HL%= 10*256+20. A rutin hívására példát mutat a 190-es sor: CALL &a000, cím%,af%,bc%,de%,hl% (ott '%' nélkül, mivel minden változó egész típusúnak lett deklarálva).

3.3 A KÉPERNYŐMEMÓRIA ÉS KEZELÉSE

A CPC képernyője a felső ROM-mal párhuzamban a &C000-&FFFF RAM tartományban helyezkedik el. A képernyő kezdőcíme minden görgetésnél (scroll) eltolódhat, csak a 'CLS' és 'MODE' utasítások után áll újra vissza a &C000-ás kezdő cím. Amennyiben a képernyőmemóriát közvetlenül kívánjuk megcímezni, gondoskodni kell róla, hogy az utolsó 'CLS' vagy 'MODE' parancs kiadása után görgetés ne történjék. Ez különösen képernyőtartalom kimentésénél majd visszatöltésénél fontos. Elgörgetett képernyőre betöltve egy régi képet, annak széttöredezett mását látjuk viszont.

3.3.1 A képernyőmemória szervezése

A CPC képernyője bitszervezésű. Ez azt jelenti, hogy minden megjelenítendő pontnak a képernyőmemória egy vagy több bitje felel meg. Ha a képernyőre 'POKE'-olunk, nem karakter jelenik meg, mint a HT vagy COMMODORE 64 gépeken, hanem egy vagy több különböző színű pont. A pontok és bitek közötti összefüggés a pillanatnyi képernyő-üzemmódtól függ. Legegyszerűbb a helyzet 'MODE 2'-ben:

```
10 rem KEPERNYOMEMORIA KIOLVASAS
20 rem
30 mode 2                                  - kezdő címet &C000-ra!
40 print "A"                               - bal felső sarokba: A
50 for cim=&C000 to &FFFF step &800        - erről még szó lesz
60   print bin$(peek(cim),8)               - egy-egy byte bitmintája
70 next cim
80 rem folytatjuk !
```

Az 'A' betű alatt megjelenő '0'-ákból és '1'-esekből álló alakzatban könnyen felismerhető a betű képe: a betű minden pontjának '1', a háttér pontjainak '0' felel meg. A betűt tehát 8 egymás feletti byte jeleníti meg. Minden beállított bit 'PEN 1' színű pontot eredményez, minden törölt bit 'PEN 0' színű pontot ad. A tollak tintáihoz rendelt palettaszín beállítása és megjelenítése a hardver dolga. Mivel 'MODE 2'-ben csak két szín lehet egyszerre a képernyőn, egy bit elegendő egy pont színének a meghatározásához: 0 vagy 1. A bitmintának tehát semmi köze a valóban megjelenő színekhez.

Egy byte 8 bitje elegendő egy karakter egy sorának ábrázolásához, 'MODE 2'-ben pedig 80 karakter fér el egy sorban, egyszerűen adódik a képernyő szélessége byte-okban: 80 byte, azaz 80*8 = 640 bit. A karakter következő sora azonban nem az &C000+80 címen kezdődik, ahogy az elvárható lenne, hanem az &C000+&800 címen! Ez magyarázza meg előző programunk ciklusváltozóját: a karakter második pixelsora 2048 (=&800) byte-tal magasabb címen található, a harmadik pixelsor a másodikhoz

képest szintén &800 byte-tal tolódik el, stb. Egy karakter bitmintája tehát az egész 16 Kbyte-os memória területén van szétdobálva. Az &C000+80 címen a második karaktersor első pixelsora kezdődik. Módosítsa az előző programot a következő sorral:

```
60 poke cim+80, peek(cim)          - 80 byte-tal feljebb teszi
```

A program átmásolja az 'A'-t egy karaktersorral lejjebb. A tanulság így szól: a 16 Kbyte-os képernyőmemória 8 darab 2 Kbyte-os blokkra osztható fel, egy-egy blokk tartalmazza az összes karaktersor egy-egy pixelsorát. Blokkonként 2000 byte szükséges így, vagyis 48 byte nincs kihasználva. Ez a terület sajnos nem hasznosítható, mivel a képernyő görgetése módosítja a kezdőcímét és az addig kihasználatlan területre is kerülhet hasznos információ.

CPC kepernyőmemória MODE parancs után

1.	2. byte	79.	80.		
C000	C001	C04E	C04F	1. pixelsor	
C800	C801	C84E	C84F	2. pixelsor	**1.**
D000	D001	D04E	D04F	3. pixelsor	ka-
D800	D801	D84E	D84F	4. pixelsor	rak-
E000	E001	E04E	E04F	5. pixelsor	ter
E800	E801	E84E	E84F	6. pixelsor	
F000	F001	F04E	F04F	7. pixelsor	sor
F800	F801	F84E	F84F	8. pixelsor	
C050	C051	C09E	C09F	1. pixelsor	
C850	C851	C89E	C89F	2. pixelsor	2.
.	.	.	.	.	kar.
F050	F051	F09E	F09F	7. pixelsor	
F850	F851	F89E	F89F	8. pixelsor	sor
C0A0	C0A1	C14E	C14F	1. pixelsor	3.
.	.	.	.	.	kar.
F8A0	F8A1	F8EE	F8EF	8. pixelsor	sor
C6E0	C6E1	C72E	C72F	1. pixelsor	23.
FED0	FED1	FF2E	FF2F	8. pixelsor	kar. sor
C730	C731	C77E	C77F	1. pixelsor	**24.**
.	.	.	.	.	kar.
FF30	FF31	FF7E	FF7F	8. pixelsor	sor
C780	C781	C7CE	C7CF	1. pixelsor	
CF80	CF81	CFCE	CECF	2. pixelsor	**25.**
D780	D781	D7CE	D7CF	3. pixelsor	ka-
DF80	DF81	DFCE	DFCF	4. pixelsor	rak-
E780	E781	E7CE	E7CF	5. pixelsor	ter-
EF80	EF81	EFCE	EFCF	6. pixelsor	
F780	F781	F7CE	F7CF	7. pixelsor	sor
FF80	FF81	FFCE	FFCF	8. pixelsor	

A képernyőpontok és bitek közötti összefüggés aránylag egyszerű 'MODE 2'-ben, ahol egy pont állapotát egy bit állapota határozza meg. A képernyő minden 'MODE'-ban 80 byte széles, azonban 'MODE 1'-ben csak 320, 'MODE 0'-ban pedig csak 160 pont a vízszintes felbontás. Ebből adódik, hogy 'MODE 1'-ben 640/320, vagyis 2 bit, 'MODE 0'-ban pedig 640/160 = 4 bit határozza meg egy-egy pont színét. Két bittel négy, négy bittel pedig 16 szín különböztethető meg, ami megfelel az egyes 'MODE'-okben használható színek számának. Mivel egy karakter mindig 8 pixelpont széles, csak 'MODE 2'-ben elég egy byte a karakter egy pixelsorának az ábrázolásához:

	Karakter szélessége byte-okban	Egy pixelpont színét meghatározó bitek száma	
MODE 2	1	1	(2 szín)
MODE 1	2	2	(4 szín)
MODE 0	4	4	(16 szín)

A pont színére vonatkozó információt nem a szomszédos bitek tartalmazzák, hanem a byte kettő vagy négy részre oszlik és a részeken belül azonos pozíción álló bitek határoznak meg egy-egy színt. 'MODE 1'-ben a helyzet a következő:

	1. pont	2. pont	3. pont	4. pont	
bitpárok:	3.	2.	1.	0.	(függőlegesen
	7.	6.	5.	4.	állnak párban!)

A bitpárok által képviselt kombináció a következő információt hordozza:

```
0 0 ..... 'PEN 0'
0 1 ..... 'PEN 1'
1 0 ..... 'PEN 2'
1 1 ..... 'PEN 3'
```

```
mode 1
poke &c000, &x10000000                  - 'PEN 1' pont baloldalt
poke &c000, &x00000001                  - 'PEN 2' pont jobbra
```

A következő program a képernyő bal felső sarkából kiinduló ferde vonalat húz 'PEN 3'-mal:

```
1 rem KOZVETLEN KEPERNYO MANIPULACIO
2 rem ======= MODE 1-ben =========
5 defint a - z                          - igy gyorsabb lesz
10 mode 1                               - kezdő cim: &C000
20 data &x10001000                      - bal szélső pont maszkja
30 data &x01000100                      - tőle jobbra eső pont
40 data &x00100010                      - még eggyel jobbra
50 data &x00010001                      - jobb szélső pont
60 for t=0 to 3                         - 1 byte: 4 pont, minden
70 read maszk(t): next t                  pont: 2 bit, de melyik?
80 rem
90 for karsor=0 to 24                   - nagyhurok: fentről le
100  for egysor=0 to 7                  - kishurok: 8 pixelsor
110    psorcim= &c000 + egysor*&800 + karsor * 80
120    jobbra=jobbra+1                  - egy ponttal jobbra
130    byte = jobbra \ 4: bitek= jobbra mod 4
140    poke psorcim+byte, peek(psorcim+byte) or maszk(bitek)
150  next egysor
160 next karsor
```

A 'psorcim' tartalmazza a kivánt pixelsor kezdő címét, de mivel ferde vonalat akarunk húzni, minden következő pont eggyel 'jobbra' kerül az előzőtől. Minden negyedik pont után egy byte-

ot kell lépni, addig viszont ugyanabba a byte-ba kell elhelyezni egymás után a négy 'maszk'-ot. A 'byte' és 'bitek' kiszámitását a 130-as sor végzi el, a 140-es sor a kivánt memóriacím tartalmához hozzáadja az új pontnak megfelelő 'maszk'-ot.

'MODE 0'-ban a bitek és a pontok színe közötti összefüggés hasonló elvekre épül. Az elmondottakból következik, hogy bátor ember az, aki a CPC képernyőjéhez közvetlenül kíván hozzányúlni. Mielőtt egy ilyen vállalkozásra szánnánk rá magunkat, magyarázattal tartozunk a képernyő görgetésekor bekövetkező kezdőcím eltolódásra. Görgetéskor a felső vagy alsó karaktersor tartalma elvész, minden sort eggyel arrébb kell másolni. A CPC nehézkes képernyőszervezése miatt ez időtrabló tevékenység. A FIRMWARE megalkotói a gyorsabb görgetés elérésére egy trükköt alkalmaztak: a teljes képernyő görgetésekor nem pakolják át a 16 Kbyte-ot, hanem az intelligens képernyő-processzort egy új kezdő címre állitják, amely igy a kivánt karaktersornál kezdi a képet. A sor valódi memóriacíme természetesen nem változik meg. A rendszer nyilvántartja az igy adódó eltolást, hiszen minden képernyőművelethez szükséges a valódi kezdőcím ismerete. Az &C000 címhez viszonyitott eltolást a SCR GET LOCATION FIRMWARE rutin adja meg a HL regiszterpárban.

A CPC képernyője a 64 Kbyte RAM bármely 16 Kbyte-os blokkjában elhelyezhető a SCR SET BASE FIRMWARE rutin segitségével. Gyakorlatilag csak az &4000 cimen kezdődő blokk használható ki, mivel a másik kettőben a rendszer számára fontos információ található (vö. a CPC memóriatérképe). A következő rövid program két képernyőt biztosit a felhasználónak. A képernyőket egy 'CALL'-lal lehet váltogatni.

```
10 rem KET KEPERNYO
20 rem
30 memory &3fff                         - &4000-től az új képernyő
40 read a$: if a$="vege" then 80
50 poke &8000+szamlalo,val("&"+a$)      - az új képernyő fölé
60 szamlalo = szamlalo + 1: goto 40
70 rem
80 print "KET KEPERNYO rutin inicializalva."
90 print "also kepernyore kapcsol: call &8000"
```

```
100 print"felso kepernyore kapcsol: call &8006"
110 end
999 rem
1000 data 3e,40                    - ld a,&40    (&4000-hez)
1010 data cd,08,bc                 - call &bc08 (FIRMWARE)
1020 data c9                       - ret
1030 data 3e,c0                    - ld a,&c0    (&C000-hoz)
1040 data cd,08,bc                 - call &bc08 (FIRMWARE)
1050 data c9, vege                 - ret
```

A gépi kódú rutinok vége (&800C) és az áttervezhető karakterek közötti kb. 9 Kbyte-nyi memóriatartomány szabadon áll adatok vagy más gépi kódú programok részére.

3.3.2 Gyors képernyőmanipuláció

Az egyébként parancsokban gazdag CPC BASIC-ból hiányoznak a képernyő egy darabkájának elraktározását és újbóli megjelenítését támogató utasítások. Ilyen utasítások segítségével lehetővé válna az ablakok alatti képernyőterület elmentése, majd az ablak bezárását szimuláló azonnali visszamásolása. A CPC-nél hiányzó hardver sprite-ok is szimulálhatók ilyen szoftverrel: képernyődarabkát (hátteret) elmenteni, sprite-ot kirakni, kis ideig várakozni, hátteret visszarakni, a sprite mozgásának megfelelő újabb hátteret elmenteni, sprite-ot arrébb kirakni ... és a sprite végigvonul a képernyőn.

A továbbiakban egy olyan BASIC bővítést ismertetünk, amely mindezt lehetővé teszi két RSX parancs bevezetésével. A bővítés 'MODE 1'-ben használható, ahol 3*3 karakternyi képernyőterületek (24*24 pixel) azonnali elmentését és megjelenítését végzi el. Az új parancsok a következők:

|GET, sprite-szám, x-koordináta, y-koordináta
|PUT, sprite-szám, x-koordináta, y-koordináta [,mód]

sprite-szám: (1-20) 20 darab egyenként 3*3 karakternyi képernyőterület, a továbbiakban 'sprite', tárolása lehetséges. Ha valóban mozgó sprite-okat kívánuk létrehozni az új parancsokkal, akkor ennek a számnak csak a fele

használható ki: tárolni kell a sprite-ok alatti hátteret is, mivel azt a sprite áthelyezése előtt újra meg kell jeleníteni.

koordináták: (x: 1-80) (y: 1-50) a finomabb mozgás érdekében a szöveges koordinátarendszernél finomabb beosztásokra van szükség. A képernyőszervezés ismertetésekor szó volt arról, hogy 'MODE 1'-ben egy karakter szélessége 2 byte, így könnyen lehetséges (bitek tologatása nélkül is) az x-tengely beosztásait a duplájára növelni. Az arányosság kedvéért az y-tengely beosztásai is megkétszereződtek. A szöveges képernyő koordinátarendszeréhez hasonlóan az 1,1 pont a bal felső sarokban van.

mód: (0-3) a sprite megjelenítési módja teljesen megegyezik a 2.3.7 pontban ismertetett keresztezési módokkal. Nem megadott 'mód' a nullás módot jelenti: a sprite egész területe felülírja az alatta található képernyőterületet. A színek és módok megfelelő kombinációjával elérhető, hogy a sprite egy háttér előtt vagy mögött jelenjen meg.

Assembler hiányában az alábbi programlista második oszlopának hexadecimális számait kell 'DATA' sorokba írni és a szokásos módon, &A000 címtől a memóriába 'POKE'-olni. A HIMEM-et azonban nem a program alá, hanem &8FFF-re kell állítani: 'MEMORY &8FFF', mivel a sprite-oknak is kell hely. Egy sprite 6*24 = 144 byte helyet igényel. (Vízszintesen 6 byte = 6*4, vagyis 24 pixel.)

```
 ; RSX parancsok: |GET,sprnum,x,y
 ;                |PUT,sprnum,x,y,mód
 ;
                  10 sprmem: equ  #9000
 ; itt kezdődnek a sprite-ok, 20 darab á 144 byte
A000              20         org  #a000
 ; az RSX parancsok inicializálása
A000  010DA0      40         ld   bc,rsxtab
A003  2109A0      50         ld   hl,cpcnek
A006  C3D1BC      60         jp   #bcd1
A009              70 cpcnek: defs 4
 ; 'DATA' sorokba itt 00,00,00,00 jön!!!!!!
```

```
A00D  15A0        80 rsxtab  defw pnevek
A00F  C33DA0      90         jp   put
A012  C31CA0     100         jp   get
A015  5055       110 pnevek: defm "PU"
A017  D4         120         defb "T"+#80
A018  4745       130         defm "GE"
A01A  D4         140         defb "T"+#80
A01B  00         150         defb 0
 ; itt kezdődik a tulajdonképpeni bővítés
 ; paraméterátvétel szigorú ellenőrzéssel
 ; rossz paraméter esetében visszatér és nem hajtja végre
A01C  FE03       160 get:    cp   3
 ; ha nincs 3 paraméter vissza a BASIC-be
A01E  C0         170         ret  nz
A01F  DD7E00     180         ld   a,(ix+0)
 ; (ix+0) tartalmazza az y-koordinátát
A022  FE2E       190         cp   46
 ; y irányban a határ 45, hogy lent ki ne lógjon!
A024  D0         200         ret  nc
A025  321EA1     210         ld   (yps),a
A028  DD7E02     220         ld   a,(ix+2)
 ; (ix+2)-ben az x-koordináta
A02B  FE4C       230         cp   76
A02D  D0         240         ret  nc
 ; ha 75-nél nagyobb: vissza BASIC-be (kilógna jobboldalt)
A02E  321FA1     250         ld   (iks),a
A031  DD7E04     260         ld   a,(ix+4)
 ; (ix+4) -en a sprite-szám  (1 - 20 lehet)
A034  FE15       270         cp   21
A036  D0         280         ret  nc
A037  3220A1     290         ld   (sprsz),a
A03A  C3B2A0     300         jp   get1
 ; PUT parancsnál 3 vagy 4 paraméter lehet
A03D  FE04       310 put:    cp   4
A03F  280A       320         jr   z,jo1
A041  FE03       330         cp   3
A043  C0         340         ret  nz
 ; ha A-ban 3 van: nincs módparaméter, vagyis mód=0
A044  AF         350         xor  a
 ; (ix+0)-ra kell állni
```



```
A045  DD2B       360         dec  ix
A047  DD2B       370         dec  ix
A049  1806       380         jr   jo2
 ; ha A-ban 4 van: van módparaméter
A04B  DD7E00     390 jo1:    ld   a,(ix+0)
 ; (ix+0)-an a mód száma, csak  0,1,2,3 érvényes!
A04E  FE04       400         cp   4
A050  D0         410         ret  nc
A051  3221A1     420 jo2:    ld   (mod),a
A054  DD7E02     430         ld   a,(ix+2)
A057  FE2E       440         cp   46
A059  D0         450         ret  nc
A05A  321EA1     460         ld   (yps),a
A05D  DD7E04     470         ld   a,(ix+4)
A060  FE4C       480         cp   76
A062  D0         490         ret  nc
A063  321FA1     500         ld   (iks),a
A066  DD7E06     510         ld   a,(ix+6)
A069  FE15       520         cp   21
A06B  D0         530         ret  nc
A06C  3220A1     540         ld   (sprsz),a
A06F  C3DBA0     550         jp   put1
 ; a következő rész a paraméterek alapján kiszámítja
 ; a képernyődarabka bal felső pontjának címét (scrcim)
 ; valamint az aktuális sprite helyét a sprite-táblában
 ; sprmem-től (sprlok)
A072  CD0BBC     590 cimek:  call #bc0b
 ; SRC GET LOC: hl-ben a képernyőkezdet eltolása &C000-tól
A075  1100C0     600         ld   de,#c000
A078  19         610         add  hl,de
A079  1600       620         ld   d,0
A07B  3A1FA1     630         ld   a,(iks)
A07E  3D         640         dec  a
A07F  5F         650         ld   e,a
A080  19         660         add  hl,de
 ; hl a sprite x-koordinátájára mutat
A081  3A1EA1     670         ld   a,(yps)
A084  FE01       680         cp   1
 ; ha y = 1, akkor függőleges koord. OK !
A086  2816       690         jr   z,foke
```

```
A08B  3D          700          dec  a
A089  CB27        710          sla  a
A08B  CB27        720          sla  a
A08D  47          730          ld   b,a
A08E  110008      740          ld   de,#800
 ; eltolás: két karaktersor közötti különbség: &800
A091  C5          750 h1:      push bc
A092  19          760          add  hl,de
 ; egy karaktersort lefelé
A093  3006        770          jr   nc,h1v
A095  01B03F      780          ld   bc,#3fb0
 ; ha HL átbukik &ffff-en, akkor baj van!
 ; le kell vonni belőle &4000-80-at, hogy jó legyen a cím
 ; az átbukást a képernyő kezdőcímének eltolódása miatt
 ; kell figyelni
A098  A7          790          and  a
A099  ED42        800          sbc  hl,bc
A09B  C1          810 h1v:     pop  bc
A09C  10F3        820          djnz h1
A09E  2222A1      830 foke:    ld   (scrcim),hl
 ; képernyődarabka címe megvan, jöhet a sprite címe
A0A1  3A20A1      850 sphely:  ld   a,(sprsz)
A0A4  47          860          ld   b,a
A0A5  21707F      870          ld   hl,sprmem-144
A0A8  119000      880          ld   de,144
A0AB  19          890 h2:      add  hl,de
A0AC  10FD        900          djnz h2
A0AE  2224A1      910          ld   (sprlok),hl
A0B1  C9          920          ret
 ; GET1: scrcim-től átmásolni sprlok-ba. Csakhogy amíg
 ; a sprite adatai összefüggő 144 byte-on helyezkednek el,
 ; a képernyőn csak 6 byte van egymás mellett, a következő
 ; sor pedig &800-zal magasabb címen van!
A0B2  F3          960 get1:    di
A0B3  CD72A0      970          call cimek
A0B6  2A22A1      980          ld   hl,(scrcim)
A0B9  ED5B24A1    990          ld   de,(sprlok)
A0BD  0618       1000          ld   b,3*8
 ; 3 karaktersor, 3*8  = 24 pixelsor
A0BF  C5         1010 loop1:   push bc
```



```
A0C0  E5         1020          push hl
A0C1  0606       1030          ld   b,6
 ; 6 byte szélesség = 3 karakter MODE 1-ben
A0C3  7E         1040 loop2:   ld   a,(hl)
A0C4  12         1050          ld   (de),a
A0C5  13         1060          inc  de
A0C6  23         1070          inc  hl
A0C7  10FA       1080          djnz loop2
A0C9  E1         1090          pop  hl
A0CA  010008     1100          ld   bc,#0800
 ; a képernyőn minden pixelsor között &800 a különbség
A0CD  09         1110          add  hl,bc
 ; ha a 15. bit-en átvitel: CY=1, átbukott &FFFF-en
A0CE  3006       1120          jr   nc,lopend
A0D0  01B03F     1130          ld   bc,#3fb0
A0D3  A7         1140          and  a
 ; C-flaget törölni a kivonáshoz!
A0D4  ED42       1150          sbc  hl,bc
A0D6  C1         1160 lopend:  pop  bc
A0D7  10E6       1170          djnz loop1
A0D9  FB         1180          ei
A0DA  C9         1190          ret
 ; PUT1: minden azonos a GET1-gyel, csak az átrakás iránya
 ; különbözik: a scrcim-re kell ugrásokkal kitenni a sprlok-
 ; nál kezdődő összefüggő 144 byte-ot.
A0DB  F3         1220 put1:    di
A0DC  CD0BA1     1230          call mode
 ; milyen módban kell kitenni?
A0DF  CD72A0     1240          call cimek
A0E2  CD19BD     1250          call #bd19
 ; MC WAIT FLYBACK rutin a FIRMWARE-ből, hogy ne villogjon!
A0E5  2A22A1     1260          ld   hl,(scrcim)
A0E8  ED5B24A1   1270          ld   de,(sprlok)
A0EC  0618       1280          ld   b,3*8
A0EE  C5         1290 lop1:    push bc
A0EF  E5         1300          push hl
A0F0  0606       1310          ld   b,6
A0F2  1A         1320 lop2:    ld   a,(de)
A0F3  00         1330 chmod:   nop
 ; a mode rutin már átírta az előző byte-ot!
```

```
A0F4  77        1340          ld   (hl),a
A0F5  13        1350          inc  de
A0F6  23        1360          inc  hl
A0F7  10F9      1370          djnz lop2
A0F9  E1        1380          pop  hl
A0FA  010008    1390          ld   bc,#0800
A0FD  09        1400          add  hl,bc
A0FE  3006      1410          jr   nc,lend
A100  01B03F    1420          ld   bc,#3fb0
A103  A7        1430          and  a
A104  ED42      1440          sbc  hl,bc
A106  C1        1450 lend:    pop  bc
A107  10E5      1460          djnz lop1
A109  FB        1470          ei
A10A  C9        1480          ret
A10B  3A21A1    1490 mode:    ld   a,(mod)
A10E  2600      1500          ld   h,0
A110  6F        1510          ld   l,a
A111  111AA1    1520          ld   de,modtab
A114  19        1530          add  hl,de
 ; hl-ben modtab + A címe
A115  7E        1540          ld   a,(hl)
 ; a-ban: 00,AE,A6 vagy B6
A116  32F3A0    1550          ld   (chmod),a
 ; a megfelelő beíródik a chmod címre
A119  C9        1560          ret
A11A  00        1570 modtab:  nop
A11B  AE        1580          xor  (hl)
A11C  A6        1590          and  (hl)
A11D  B6        1600          or   (hl)
 ; paraméterek tárolói:
A11E  00        1630 yps:     defb 00
A11F  00        1640 iks:     defb 00
A120  00        1650 sprsz:   defb 00
A121  00        1660 mod:     defb 00
A122  0000      1670 scrcim:  defw 0000
A124  0000      1680 sprlok:  defw 0000
```

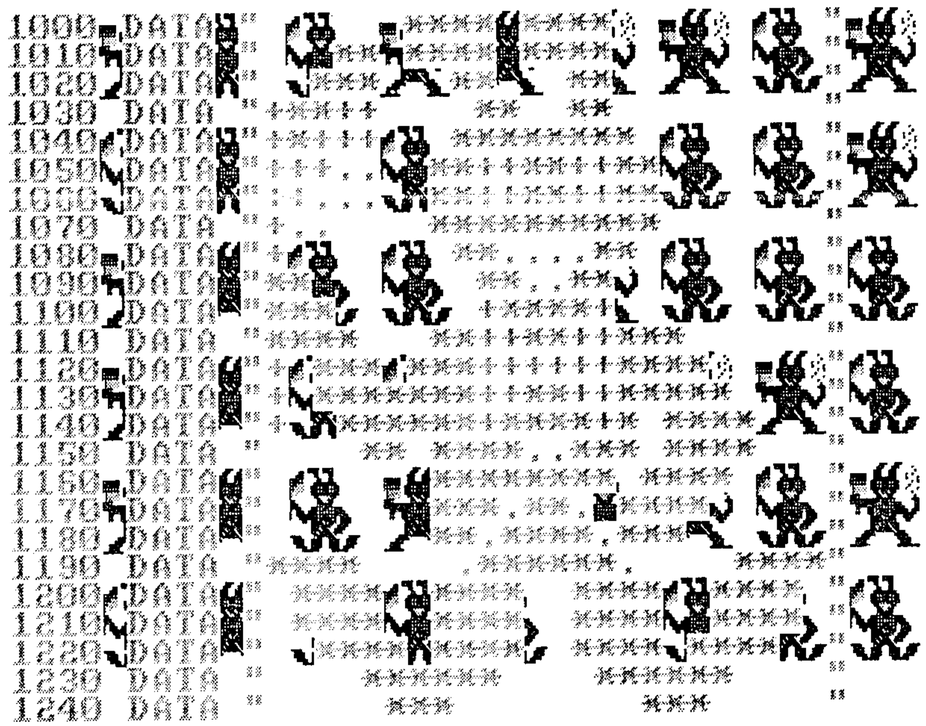
Bárhogy is kerüljön be a bővítés a CPC RAM-jába, használatbavétel előtt inicializálni kell: 'CALL &A000'. Ezután már

megérti a rendszer a két új parancsot. Kezdetnek próbálja meg a képernyő darabkáit elraktározni majd máshová visszamásolni. Ablakok elmentése nem lesz nehéz, csak a koordináták átszámítására kell ügyelni. Minden szöveges koordinátának páratlan sprite-koordináta felel meg: spritekoord = (szövkoord * 2) - 1. Valódi mozgatható sprite-okat először meg kell tervezni, majd kirajzolni a képernyőre és onnan '!GET'-tel elmenteni a 20 hely egyikére. Az összes sprite külső tárba menthető és onnan betölthető a következő parancsokkal:

```
SAVE "file-név",&9000,20*144      - sprite-okat kiment
LOAD "file-név",&9000             - sprite-okat beolvas
```

A sprite-ok megtervezésére és átvételére mutat példát a következő program. A huszonnégy pixelsor egy-egy 'DATA' sorba kerül, így a készülő kép azonnal áttekinthető. Az egyes pontok színét a következő karakterek jelölik: PEN 0 (szóköz)
PEN 1 . (pont)
PEN 2 + (pluszjel)
PEN 3 * (csillag).
A 30-70 sorok közötti kettős ciklus a 'PLOT' utasítással az 1,1 sprite-koordinátára rajzolja ki az alakzatot. Onnan már egyszerű elmenteni a '!GET, száma,1,1' paranccsal. A sprite sorszáma természetesen 1 és 20 között lehet.

```
1 rem ****************************
2 rem *                          *
3 rem * EGYSZERU SPRITE TERVEZO *
4 rem *                          *
5 rem ****************************
8 MEMORY &8FFF              ' GET-PUT rutin inicializalva lett?
10 MODE 1 : DEFINT a-z
15 toll$=" .+*"             ' pen 0,1,2,3-nak megfelelo karakter
20 RESTORE 1000             ' elso sprite kepe itt kezdodik
30 FOR f= 1 TO 24: READ a$  ' egy pixelsor
40 FOR v= 1 TO 24:          ' egy pont
50     pont$=MID$(a$,v,1): szin=INSTR(toll$,pont$)-1
60     PLOT v*2-2,400-2*f+1, szin
70 NEXT v,f                 ' sprite kirajzolodik 1,1 pozicion
80 !GET 1,1,1               ' 1-es szamu sprite-kent elmenti
```



```
999 DATA  "123456789012345678901234"   [ez a vizszintes merce]
1000 DATA "       **** ****          "
1010 DATA "    **  **** ****         "
1020 DATA "   ***    **   **         "
1030 DATA "+****+     **  **         "
1040 DATA "+*++++    ********        "
1050 DATA "+++..    **+******+**     "
1060 DATA "++...    **+****+**      "
1070 DATA "+..      **********      "
1080 DATA "+         **....**       "
1090 DATA "**        **..**         "
1100 DATA "***       +****+         "
1110 DATA "****    **+*****+**      "
1120 DATA "+ *** ***+*********      "
1130 DATA "+ ********+*********     "
1140 DATA "+  ******+*****+ ****    "
1150 DATA "     ** ****..*** ****   "
1160 DATA "        ******** ****    "
1170 DATA "        ***.**. ****     "
```



```
1180 DATA "        **.****.***       "
1190 DATA "****     .******.     ****"
1200 DATA " ****  ****  ****  **** "
1210 DATA " ****  ****  ****  **** "
1220 DATA "  **** ****  **** ****  "
1230 DATA "    ******    ******    "
1240 DATA "     ***        ***     "
1999 rem es igy tovabb
```

3.4 A BASIC PROGRAMOK SZERKEZETE

A CPC BASIC interpretere a felső ROM-ban található a &C000--&FFFF tartományban. Rutinjaihoz egyszerű módon nem lehet hozzáférni, de nem is érdemes, mivel a FIRMWARE rutinokkal a legtöbb feladat megoldható. A BASIC program szerkezetét megismerve azonban választ kapunk arra a kérdésre is, hogy miért ilyen gyors a CPC BASIC interpretere. A sebesség lemérhető az alábbi próbaprogram segítségével:

```
100 REM BENCHMARK 7: SZUPER BIT-LET 50. old
110 PRINT"s"
120 k=0
130 DIM m(5)
140 k=k+1
150 a=k/2*3+4-5
160 GOSUB 230
170 FOR 1=1 TO 5
180 m(1)=a
190 NEXT 1
200 IF k<1000 THEN 140
210 PRINT"e"
220 STOP
230 RETURN
240 END
```

A CPC 30.3 másodperc alatt hajtja végre a programot.
Soknak tűnik? Akkor vesse össze az alábbi gépek adataival:

COMMODORE 64: 49.5 s	ATARI 400/800: 61.5 s
COMMODORE 16: 55.6 s	APPLE III : 42.5 s
HT 1080Z : 80.0 s	Sinclair QL : 42.6 s

IBM PC : 37.4 s " Spectrum : 80.7 s

3.4.1 Programsorok és változók tárolása

A BASIC program szövege általában az &170 címen kezdődik. Az alábbi kis programmal megvizsgáljuk a sorok szerkezetét. Pontosan úgy írja be, ahogy a lista szól, hogy ne legyen eltérés a memóriában! A kétsoros program utáni közvetlen parancs hexadecimális formában megjeleníti a programszöveget tartalmazó tár egyes byte-jait:

```
10 a$ = "Jozsi bacsi:":print a$
1000 xy=sin(20):print xy

for t=0 to 60: print hex$(peek(&170+t),2); " ";: next t
```

Az &170 címtől található tártartalom értelmezése

```
21 00     - az első sor hossza,
            a következő sor kezdőcíme: &170 + &21+&00*&100 = &191
0A 00     - az első sor sorszáma: &A0 + &00*&100 = &A0, vagyis 10
03 00 00  - szöveges típusú változó következik
E1        - változó neve: ASC ("a") + &80, ami a név utolsó
            karakterét jelzi: &61 + &80 = &E1
20 EF 20  - szóköz,   '=' tokenje,     szóköz
22        - nyitó idézőjel
4A 6F 7A 73 69 20 62 61 63 73 69 3A - Jozsi bacsi:  (ASCII kód)
22        - záró idézőjel
01        - ':' tokenje
BF        - 'PRINT' tokenje
20        - szóköz
03 00 00 E1       - szöveges változó a$ (mint fent)
00        - programsor vége
19 00     - a második sor hossza &19 byte,
            következő sor címe: e sor címe (lásd az előző sornál)
            + &19 + &00*&100 =  &191 + &19 = &1AA
E8 03     - a sor sorszáma: &E8 + &03*&100 = &3E8, vagyis 1000
0D 00 00  - valós típusú változó következik
78 F9     - ASC("x"), ASC("y")+ &80, a változónév utolsó karaktere megjelölve: 7. bit beállítva
```



```
EF        - "=" tokenje
FF 15     - függvény következik, 'SIN' tokenje
28        - nyitó zárójel
19 14     - egybyte-os szám következik, &14, vagyis 20
29        - záró zárójel
01 BF 20  - ':' tokenje,   'PRINT' tokenje,  szóköz
0D 00 00  - valós típusú változó következik
78 F9     - 'xy' változónév, utolsó karakter 7. bitje beállítva
00 00 00  - program vége: sor végét jelölő &00 és a következő
            sor hossza helyett &00 &00
```

A programszöveg memóriaképe a következőket árulja el:

Minden programsor a sor hosszával kezdődik. A következő sor kezdőcíme = az adott sor kezdőcíme + az adott sor hossza.

Ezt követi a sor sorszáma LSB, MSB sorrendben, binárisan.

A BASIC parancsok és utasítások helyett egybyte-os tokenek állnak. A tokenek használata lerövidíti a programszöveget és meggyorsítja a program értelmezését. Az utasítástokenek kódja 127 (&7F) fölött kezdődik, pl. &80 = 'AFTER'.

A BASIC függvények tokenje kétbyte-os: &FF vezeti be az &80-nál kisebb kódszámú függvénytokeneket, pl. &FF &15 = 'SIN'.

Az utasításokat elválasztó ':' tokenje &01, a programsor végét &00 jelzi.

A változók nevének végét a karakter ASCII kódszámához hozzáadott &80 jelzi.

Az interpreter minden operátort tokenizál, nem tartja meg a műveleti jelek ASCII kódját sem, pl. "=" tokenje &EF.

Az eddig felsorolt tulajdonságok a legtöbb BASIC interpreterre jellemzőek. A következők azonban a CPC sajátosságai. Az alábbiakban olyan tulajdonságokat is felsorolunk, amelyekre kis programunk nem szolgáltatott példát.

A változó nevét annak típusjelzése vezeti be:
&02 vezeti be a '%' jellel jelzett egész típusú változót,
&03 vezeti be a '$' jellel jelzett szöveges típusú változót,
&04 vezeti be a '!' jellel jelzett valós típusú változót,
&0B vezeti be a deklarált egész típusút változót,
&0C vezeti be a deklarált szöveges típusú változót,

%0D vezeti be a deklarált valós típusú változót.

A típusjelzést még két %00 byte követi, a program futásakor ide írja be az interpreter a változók címét a programszöveg után elhelyezett változótábla kezdetéhez viszonyítva. A változók keresése ily módon jelentősen felgyorsul.

A CPC a numerikus konstansokat nem ASCII karakterláncként veszi be a programszövegbe, hanem már a programsor átvételekor binárisan tárolja azokat. Hogy listázáskor a konstansok a beírt alakban jelenjenek meg, a BASIC a következő jelölést használja:
%19 vezeti be az egy byte-on tárolható decimális konstanst,
%1A vezeti be a két byte-on tárolható decimális konstanst,
%1B vezeti be a két byte-on tárolható bináris konstanst,
%1C vezeti be a két byte-on tárolható hexadecimális konstanst,
%1F vezeti be a két byte-on már nem tárolható, tehát valós típusú konstanst, ezt a következő 5 byte-on tárolja.

A CPC egy egész sor rövidítést használ numerikus konstansok helyett, pl. %0E - %17 jelöli a 0 - 9 közötti számokat.

A 'GOTO' és 'GOSUB' utáni sorszám szintén bináris formában kerül be a programsorba, a sorszámot %1E vezeti be. A program végrehajtása során az interpreter csak egyszer keresi meg sorszám alapján a keresett programsort, ha megtalálta, annak memóriacímével írja felül a 'GOTO' és 'GOSUB' utáni sorszámot, majd %1D-re változtatja a jelölést. Ezzel a módszerrel a BASIC program végrehajtási sebessége ugrásszerűen megjavul.

3.4.2 Fontosabb BASIC rendszerváltozók

Az alábbi rendszerváltozók mindhárom CPC-n az &AC00 címtől kezdődő tartományban a FIRMWARE ugrótábla alatt helyezkednek el. A felsorolásban két címet is megadunk, mivel a CPC 464 interpreterének felépítése - így a rendszerváltozók címe is - jelentősen eltér a két nagyobb CPC-ben található interpretertől. Egyes, az adott programban felesleges rendszerváltozók helye - megfelelő körültekintéssel - kisebb gépi kódú rutinok számára igénybe vehető. Ilyen például az 'ENV' és 'ENT' borítékoknak fenntartott több mint 200 byte. Így elkerülhető a HIMEM állítá-

sa. A rendszerváltozók használatakor nem lehetünk elég óvatosak: egy rossz 'POKE' a program elvesztését okozhatja!

CPC 464	CPC 664 és 6128	
&ACA4	&AC8A	256 byte: sorbeadó puffer, parancsmódban a szöveg először ide kerül (tokenizálás &40-től)
&ADAA	&AD90	utoljára előfordult hiba kódja
---	&AD91	utojára előfordult AMSDOS hiba kódja
&ADD0	&ADB7	26 * 2 byte, a skaláris változók kezdőbetű szerint láncolt táblájára mutat
&AE0C	&ADF3	26 byte: típusdefiníciók A-Z
&AE45	&AE2C	védett (protected) programot jelző zászló
&AE46	&AE2D	46 byte: karakterlánccá való átalakításhoz
&AE7B	&AE5E	2 byte: mutató a HIMEM-re
&AE81	&AE64	2 byte: BASIC programszöveg eleje
&AE83	&AE66	2 byte: BASIC programszöveg vége
&AE85	&AE68	2 byte: változók kezdete
&AE87	&AE6A	2 byte: tömbök kezdete
&AE89	&AE6C	2 byte: tömbök vége
&B08D	&B071	2 byte: szöveges változók kezdete
&B08F	&B073	2 byte: szöveges változók vége
&B1C8	&B7C3	képernyőüzemmód
&B20D	&B6B6	120 byte: az ablakok paraméterei
&B294	&B734	az első átdefiniálható karakter
&B34C	&B496	80 byte: billentyűzettábla, normál mód
&B39C	&B4E6	80 byte: billentyűzettábla, shift mód
&B3EC	&B536	80 byte: billentyűzettábla, ctrl mód
&B43C	&B586	10 byte: billentyűzettábla, auto-repeat
&B446	&B590	152 byte: funkcióbillentyűk bővítő szövege
&B61A	&B2B6	240 byte: ENV borítékok
&B70A	&B3A6	240 byte: ENT borítékok

3.4.3 Három segédprogram

Az első segédprogram nagyon hasznos szolgáltatásokat nyújt: más BASIC programok változóiról készít statisztikát. Megmondja, milyen nevű változók szerepelnek a programban, melyik program-sor(ok)ban fordulnak ezek elő és hányszor, mindezt természete-

sen ábécé sorrendbe rendezetten. A program felhasználja a BASIC programszöveg belső tárolásáról megismert tényeket, így bővebb magyarázatot nem igényel. Használata egyszerű: a kívánt BASIC programhoz 'MERGE' utasítással hozzátöltjük és 'RUN 64000'-rel elindítjuk. A tárban lévő program természetesen nem tartalmazhat 63999-nél nagyobb sorszámokat. A változókról készített statisztika tetszőlegesen küldhető a képernyőre vagy a nyomtatóra.

A program fontosabb változói:

a - programszöveg-mutató (&170-ről indul)
va$() - változónevet és az előfordulási sorszámot tárolja
p() - segédtömb a változónevek rendezéséhez
v - változónevek előfordulási száma

```
64000 REM ****************************
64001 REM *                          *
64002 REM * STATISZTIKA VALTOZOKROL *
64003 REM *                          *
64004 REM ****************************
64005 a=368: margo=10: WIDTH 80
64006 MODE 2: INK 0,13: BORDER 13: INK 1,0: PAPER 0: PEN 1
64007 INPUT "A program neve ";pn$
64008 INPUT "Nyomtatora (I/N) " ;v$:IF UPPER$(v$)="I" THEN d=8 E
LSE d=0
64009 DIM va$(1000,1),p(1000)
64010 LOCATE 1,4: PRINT "Feldolgozas alatt a";
64011 FOR i=0 TO 1000:p(i)=i:NEXT
64012 v=0
64013 kovsor = PEEK(a) + PEEK(a+1) * 256
64014 a=a+2
64015 sorszam = PEEK(a) + PEEK(a+1) * 256
64016 LOCATE 20,4:PRINT sorszam; "-es sor"
64017 IF sorszam > 63999 THEN 64030
64018 a=a+2
64019 FOR i = a TO a+kovsor-6
64020 x = PEEK(i)
64021 IF x =  2 AND PEEK(i+1) = 0 THEN  v$="%":i=i+3: GOTO 64025
64022 IF x =  3 AND PEEK(i+1) = 0 THEN  v$="$":i=i+3: GOTO 64025
64023 IF x = 13 AND PEEK(i+1) = 0 THEN  v$="": i=i+3: GOTO 64025
```



```
64024 GOTO 64027
64025 x=PEEK(I): IF x>127 THEN x=x-128: sor$=sor$+CHR$(x)+v$:GOT
O 64026 ELSE sor$=sor$+CHR$(x): i=i+1: GOTO 64025
64026 v=v+1: va$(v,0)=sor$: va$(v,1)=STR$(sorszam): sor$=""
64027 NEXT i
64028 a=a+kovsor-4: GOTO 64013
64029 REM ------------------------ rendezes indul
64030 PRINT:PRINT"Rendezes, egy pillanat turelmet ..."
64031 ww=2:n=v:n2=n/2
64032 z=0
64033 ww=ww*2
64034 IF ww<n2 THEN 64033
64035 a=1
64036 b=a
64037 IF va$(p(b),0) > va$(p(b+ww),0)    THEN h=p(b):p(b)=p(b+ww)
:p(b+ww)=h:GOTO 64040
64038 a=a+1: IF a<=n-ww THEN 64036 ELSE ww=INT(ww/2)
64039 IF ww>0 THEN 64035 ELSE 64041
64040 b=b-ww:IF b>=1 THEN 64037 ELSE 64038
64041 REM ------------------------ valtozokneveket egymas melle
64042 FOR i=0 TO n-1
64043 IF va$(p(i),0) = ""  THEN 64046
64044 j=i+1
64045 IF va$(p(i),0)=va$(p(j),0) THEN va$(p(j),0)="":va$(p(i),1)
=va$(p(i),1)+va$(p(j),1):j=j+1: IF j>n THEN 64046 ELSE 64045
64046 NEXT i
64047 REM ------------------------ valtozok rendezve, kiiratas
64048 CLS
64049 PRINT #d,SPC(margo);"**  ";pn$;"  **": PRINT #d,SPC(margo)
;STRING$(80-margo,"-");
64050 PRINT #d,SPC(margo); "VALTOZOK   SORSZAM":PRINT #d,SPC(mar
go);STRING$(80-margo,"-")
64051 FOR i=0 TO n
64052 IF va$(p(i),0)="" THEN 64054 ELSE IF ASC(va$(p(i),0))< 65
THEN 64054
64053 PRINT #d,SPC(margo);va$(p(i),0); TAB(20);va$(p(i),1)
64054 NEXT
```

A második program, bár kazetta kezelésével kapcsolatos, lemezes rendszerek tulajdonosainak is jó szolgálatokat tehet

kazettás programok lemezre történő átmásolásakor. A rendszerrel elvégeztetett 'CAT' után a program kiolvassa az ún. fejlécet: tájékoztat a kazettás program típusáról, töltőcíméről, hosszáról, stb. Mivel a 'CAT' géptípusonként máshová olvassa be a kazettás fejlécet, a 80-as sor állapítja meg a gép típusát:

```
10 REM ************************
20 REM *                      *
30 REM * PROGRAM FEJLEC OLVASO *
40 REM *                      *
50 REM ************************
60 ON BREAK GOSUB 190
70 MODE 1
80 IF PEEK (6)=128 THEN puff=&B88C ELSE puff =&B1A4 '
   CAT puffer 464-en es a ket nagyobb gepen
90 ON ERROR GOTO 100 : |TAPE : GOTO 110 'van floppy?
100 RESUME 110 'nincs
110 ' most mar biztos, hogy kazettat olvas be a CAT
120 inv$=CHR$(24)
130 PRINT "TEGYE BE A KAZETTAT A MAGNOBA! ";: PRINT:PRINT
140 PRINT "Ha az "; inv$; "'OK'"; inv$; "  megjelenik,"
150 PRINT "Nyomja le ketszer az "; inv$;"[ESC]"; inv$; " billent
yut!"
160 PRINT:PRINT
170 CAT
180 '----------------------------------------
190 MODE 2
200 LOCATE 5,2:PRINT inv$;" KAZETTAS PROGRAM FEJLEC OLVASO "; in
v$
210 filenev$="": FOR i=puff TO puff+15
220 filenev$=filenev$+CHR$(PEEK(i)) : NEXT i
230 '----------------------------------------
250 tipus=PEEK(puff+18)
260 bloksz=PEEK(puff+16) : bloksz1=bloksz
270 blokhossz=PEEK(puff+20)*256+PEEK(puff+19)
280 proghossz=PEEK(puff+25)*256+PEEK(puff+24)
290 toltocim=PEEK(puff+22)*256+PEEK(puff+21)
300 tipus=PEEK(puff+18)
310 blokkok=proghossz/2048
320 IF blokkok<>INT(blokkok) THEN blokkok=INT(blokkok)+1
```



```
330 IF tipus =0 OR tipus =1 THEN progtoltcim=&170: GOTO 350
340 progtoltcim=PEEK(puff+27*256)+PEEK(puff+26)
350 vegcim=progtoltcim+proghossz
360 '----------------------------------------
370 LOCATE 5,4:PRINT inv$;" A PROGRAM NEVE   ";inv$;"       "; f
ilenev$
380 LOCATE 5,6:PRINT inv$;" A PROGRAM TIPUSA ";inv$;"       ";
390 IF tipus=0 THEN PRINT"STANDARD BASIC"
400 IF tipus=1 THEN PRINT"VEDETT BASIC"
410 IF tipus=2 THEN PRINT"GEPI KODU"
420 IF tipus=22 THEN PRINT"ASCII FILE"
430 LOCATE 5,8:PRINT inv$;" AKTUALIS BLOKK-SZAM   ";
440 PRINT inv$;"    ";bloksz
450 LOCATE 5,10:PRINT inv$;" OSSZES BLOKKOK SZAMA  ";
460 PRINT inv$;"    ";blokkok
470 LOCATE 5,12:PRINT inv$;" A BLOKK HOSSZA        ";
480 PRINT inv$;"    ";blokhossz;" byte "
490 LOCATE 5,14:PRINT inv$;" A BLOKK TOLTO CIME    ";
500 PRINT inv$;"    ";toltocim
510 LOCATE 5,16:PRINT inv$;" A PROGRAM TOLTO CIME  ";
520 PRINT inv$;"    ";progtoltcim
530 LOCATE 5,18:PRINT inv$;" A PROGRAM VEGCIME     ";
540 PRINT inv$;"    ";vegcim
550 LOCATE 5,20:PRINT inv$;" A PROGRAM HOSSZA      ";
560 PRINT inv$;"    ";proghossz;"byte"
570 PRINT STRING$(76,95)
580 '-----------------------------------------------
590 LOCATE 10,23
600 PRINT"Megnezzuk a kovetkezo programot is?";
610 PRINT"    (i / n)  ";inv$;"-";inv$;CHR$(8);
620 k$=INKEY$:IF k$=""THEN 620
630 IF k$="i" THEN RUN ELSE IF k$="n" THEN END
640 PRINT CHR$(7);:GOTO 590
```

Az utolsó segédprogram egy karaktertergenerátor, amellyel egyszerű módon áttervezhető a CPC teljes karakterkészlete. A program az ablakhasználat következtében rendkívül barátságos és áttekinthető: a jobb oldali ablakban látható mind a 256 karakter aktuális állapota. A tervezőablakban a nyílbillentyűkkel lehet mozogni, a [COPY] beállítja a törölt pontot és törli a

beállitott pontot. Az áttervezendő karaktert közvetlenül is ki lehet választani a megfelelő billentyűvel, vagy az ASCII kód alapján kérhető ki a billentyűzetről el nem érhető karakter. Egy karakter tervezésének befejezése az [E] billetyű lenyomásával jelezhető, az áttervezett karakter természetes nagyságban is megjelenik a többiek között a jobb oldali ablakban. A karakterkészlet kimentése [CTRL][E]-vel kérhető, az 1020-as sor a megadott néven viszi ki a készletet. A kimentett karakterek a karaktergenerátorba újra betölthetők,így a munkát később is lehet folytatni. A karakterkészlet betöltését tetszőleges BASIC programba a következő programsor végzi el:

```
1 symbol after 0: h=himem: load"file-név",h+1
2 rem ---  Ez legyen az első sor! Utána már állitható
3 rem ---  lejjebb a HIMEM a 'MEMORY' utasitással.
```

```
10 REM ************************
20 REM *                      *
30 REM * KARAKTER GENERATOR *
40 REM *                      *
50 REM ************************
60 MODE 1:INK 0,13:INK 1,0:  GOSUB 670
70 CLS:PRINT"* CPC karakter-tervezo *****************"
80 GOSUB 1040 : GOSUB 1090
90 GOSUB 510: GOSUB 910
100 IF jel=5 THEN GOTO 960
110 LOCATE #4,3,19:PRINT #4,"Akt. karakter: ";CHR$(1)CHR$(jel)
120 LOCATE #4,6,22:PRINT #4,"CHR$ (";:PRINT #4,USING "###";jel;:
PRINT #4,")";
130 CLS #2: CLS #3
140 sp=jel*8+karkezd
150 FOR i=7 TO 0 STEP -1
160 z(i+1)=PEEK(sp+i):a$=BIN$(z(i+1),8)
170 FOR j=1 TO 8:IF MID$(a$,j,1)="0" THEN c$(j,i+1)="-" ELSE c$(
j,i+1)="*"
180 NEXT: NEXT
190 WINDOW #3,2,22,16,24
200 GOSUB 210:GOTO 280
210 FOR y=1 TO 8
220 FOR x=1 TO 8
```



```
230 LOCATE #2,x,y: PRINT #2, c$(x,y);
240 NEXT
250 NEXT
260 FOR y=1 TO 8:LOCATE #1,1,y:PRINT #1, USING "####";z(y);:NEXT
:RETURN
270 LOCATE #2,x,y: PRINT #2,CHR$(22); CHR$(1);CHR$(1);CHR$(0);
280 x=1:y=1:x1=x:y1=y
290 GOSUB 740
300 GOSUB 370
310 GOSUB 590: GOSUB 910
320 IF jel = 5 THEN jel=0: GOTO 90
330 sp=jel*8+karkezd
340 FOR i=1 TO 8:POKE sp-1+i,z(i):NEXT
350 ky=INT(jel/16): kx=jel MOD 16: LOCATE #4,kx+1,ky+1:PRINT #4,
CHR$(1);CHR$(jel);
360 GOTO 90
370 LOCATE #2,x,y: PRINT #2,CHR$(22);CHR$(1);CHR$(1);CHR$(0);
380 a$=INKEY$:IF a$="" THEN 380
390 a=ASC(a$)
400 IF a=240 THEN y1=y-1:IF y1<1 THEN y1=8: GOTO 470
410 IF a=241 THEN y1=y+1:IF y1>8 THEN y1=1: GOTO 470
420 IF a=242 THEN x1=x-1:IF x1<1 THEN x1=8: GOTO 470
430 IF a=243 THEN x1=x+1:IF x1>8 THEN x1=1: GOTO 470
440 IF a=224 THEN IF c$(x,y) = "*" THEN c$(x,y)="-" :z(y)=z(y)-c
(x) ELSE c$(x,y)="*": z(y)=z(y)+c(x)
450 IF a$="e" OR a$="E" THEN RETURN
460 IF a$="L" OR a$="l" THEN GOSUB 810: GOTO 370
470 LOCATE #2,x,y:PRINT #2,CHR$(22);CHR$(0);c$(x,y);
480 LOCATE #1,1,y:PRINT #1,USING "####";z(y);
490 x=x1:y=y1
500 GOTO 370
510 CLS #3
520 PRINT #3,"Karakter valasztas:"
530 PRINT #3,"--------------------"
540 PRINT #3,"billentyuvel  vagy "
550 PRINT #3,"0 + ASC kod beadas "
560 PRINT #3,"--------------------"
570 PRINT #3,"[CTRL]+E: kiszall!"
580 RETURN
590 CLS #3
```

```
600 PRINT #3,"Karaktert elhelyez:"
610 PRINT #3,"--------------------"
620 PRINT #3,"billentyu      vagy "
630 PRINT #3,"0 + ASC kod beadasa"
640 PRINT #3,"--------------------"
650 PRINT #3,"[CTRL]+E: kiszall! "
660 RETURN
670 DIM c$(8,8),z(8),c(8)
680 c(1)=128:c(2)=64:c(3)=32:c(4)=16:c(5)=8:c(6)=4:c(7)=2:c(8)=1
690 SYMBOL AFTER 0
700 karkezd=HIMEM+1
710 PRINT"Betoltendo file neve":PRINT"-- vagy csak [ENTER]";: LO
CATE 22,1:INPUT n$
720 IF n$="" THEN RETURN
730 n$=""+n$:LOAD n$:RETURN
740 CLS #3
750 PRINT #3,"[COPY] + nyilak     "
760 PRINT #3,"--------------------"
770 PRINT #3,"E = vege, uj chr$   "
780 PRINT #3,"L = kitorolni!      "
790 PRINT #3,"--------------------"
800 RETURN
810 CLS #2
820 FOR x=1 TO 8
830 FOR y=1 TO 8
840 c$(x,y)="-"
850 NEXT
860 z(x)=0
870 NEXT
880 GOSUB 210
890 x=1:y=1:x1=x:y1=y:RETURN
900 REM --- beadasok ---
910 PRINT#3:INPUT #3,"melyik karakter ",jel$
920 IF LEFT$(jel$,1) = "0" AND LEN(jel$)>1 THEN jel=VAL(jel$)
930 IF LEN(jel$)=1 THEN jel=ASC(jel$)
940 RETURN
950 REM --- kimentes ---
960 CLS #1: CLS #2
970 CLS #3:PRINT#3,"Uj karakterkeszletet kimenteni ...": PRINT#3
: INPUT #3,"File-nev ";n$
```



```
980 IF n$="" THEN 970
990 PRINT #3,CHR$(24) n$ " OK? (i/n)" CHR$(24);
1000 a$=INKEY$: IF a$="" THEN 1000
1010 IF UPPER$(a$) <> "I" THEN 90
1020 SAVE n$,b,karkezd,2048
1030 END ' ------------------ atirhato RUN-ra, akkor ujra kezd
1040 WINDOW #1,15,19,4,11
1050 WINDOW #2,3,10,4,11
1060 WINDOW #3,2,21,17,24
1070 WINDOW #4,24,39,3,24
1080 RETURN
1090 FOR t=0 TO 255: PRINT #4, CHR$(1);CHR$(t);: NEXT t
1100 PLOT 23*16-8,372:DRAWR 17*16,0: DRAWR 0,-17*16: DRAWR -17*1
6,0: DRAWR 0,17*16
1110 PLOT 8,8: DRAWR 0,10*16: DRAWR 22*16-4,0: DRAWR 0,-10*16: D
RAWR -22*16+4,0
1120 RETURN
```

3.5 A CPC 6128 memórialapozó RSX parancsai

A CPC 6128 dupla annyi RAM-mal rendelkezik, mint a két kisebb CPC: 128 Kbyte írható-olvasható memória van a gépbe építve. A gép teljes RAM-kapacitását kihasználja a CP/M Plus operációs rendszer, amely alatt így korlátozás nélkül fut az összes CP/M-re írt felhasználói program. A beépített BASIC interpreter azonban nem tartalmaz parancsokat a második 64 Kbyte-os RAM-lap kezelésére. A CPC 6128-hoz mellékelt rendszerlemez 1-es oldalán található BANKMAN program RSX parancsok formájában segíti elő a második 64 Kbyte hasznosítását. A programot a RUN "BANKMAN" parancs tölti be és indítja el. A program által inicializált RSX parancsok két csoportba sorolhatók: képernyőmásoló és filekezelő parancsok.

Képernyőmásoló parancsok

A második 64 Kbyte-os RAM-lap 4 darab 16 Kbyte-os blokkra osztható fel. Minden blokkban elfér egy teljes képernyő tartalma. Az első lapon található (valódi) képernyőre az 1-es sorszámmal, a második lapon található blokkokra a 2, 3, 4 és 5-ös sorszámmal lehet hivatkozni a következő parancsokban:

!SCREENCOPY, [részlet,] hova, honnan

A 'honnan' sorszámú képernyő tartalmát másolja át a 'hova' sorszámúra. Például a **!SCREENCOPY,2,1** parancs a második lap első blokkjába másolja a valódi képernyő pillanatnyi tartalmát. A képernyőtartalmat a **!SCREENCOPY,1,2** parancs másolja vissza. A második lap blokkjai egymásra is másolhatók, bár ennek nincs sok értelme. Egy képernyőmásolás ideje kb. fél másodperc, ami pontos időzítést kívánó programoknál eltolódásoz (zenénél nyávogáshoz) vezethet. Ez kiküszöbölhető a képernyőtartalom részletekben történő átmásolásával. A 'részlet'-ek számozása: 0-63, tehát egy képernyő átmásolása 64 részletben történik. Részletekben történő másolásnál ügyelni kell arra, hogy minden részlet átmásolódjék, különben foghíjas lesz az újra megjelenő kép.

!SCREENSWAP, [részlet,] egyik, másik

Az előző parancstól eltérően kétirányú másolást végez: kicseréli az 'egyik' blokk tartalmát a 'másik'-éval. Így az egyik blokk tartalma sem semmisül meg. A **!SCREENSWAP,1,5** parancs pl. a pillanatnyi képernyőtartalmat a második lap 4. blokkjába írja, miközben a blokk tartalma megjelenik a képernyőn. Egy újabb !SCREENSWAP,1,5 parancs után helyreáll a kiinduló állapot.

▶▶▶ Képernyőtartalmak másolása csak el nem görgetett képernyő esetében lehetséges a képernyőtartalom széttöredezése nélkül. A 'MODE' utasítás alapállapotba hozza a képernyőt és semlegesíti az előző görgetések hatását.

File-kezelő parancsok

A második 64 Kbyte-os lap RAM-lemezként is felhasználható: megnyitható egy file, amely nem a lemezen foglal helyet, hanem a memóriában. A file-műveletek ez esetben gyakorlatilag azonnal végrehajtódnak, tehát a RAM-lemez használata gyakori hozzáférést igénylő file-ok esetében előnyös. A RAM-lemezen megnyitott file bármelyik adatblokkjához közvetlenül hozzá lehet férni, ez kiküszöböli az AMSDOS soros file-kezelésének hátrányait is. A RAM-lemezt kezelő RSX parancsok azonnali hozzáférést, keresést és egyéb szolgáltatást nyújtanak, amelyek rendkívül kényelmes

file-kezelést tesznek lehetővé. Érdemes ezért a szükséges adatfile-t először lemezről a RAM-lemezre másolni, ott feldolgozni, majd a lemezre visszaírni. Mivel a közvetlen hozzáférésű RAM-file adatelemei között nincs elválasztójel, és a file végén sem áll EOF jelzés, csak egységes hosszúságú adatblokkokkal dolgozhatunk. Egyszerre egy teljes blokkhoz lehet hozzáférni, ezért célszerű a blokkon belüli adatelemeket is egységes hosszúságúra tervezni, így azok a szövegkezelő függvények segítségével egyszerűen elhelyezhetők a blokkban vagy kihámozhatók belőle.

!BANKOPEN, blokkhossz

A blokkmutatót nullára állítja, és a 'blokkhossz'-szal megadott karakternyi hosszúságú blokkok kezelésére készül fel. A blokkmutatót minden író-olvasó művelet eggyel növeli, így soros típusú kezelésnél nem kell törődni vele. Lehetőség van azonban a blokkmutatót tetszőleges blokkra állítani és a kiválasztott adatblokkhoz közvetlenül hozzáférni. Az adatblokkoknak egységes hosszúságúnak kell lenniük, ezt állítja be a 'blokkhossz'. A !BANKOPEN,100 parancs például 100 karakter hosszúságú adatblokkok kezelésére utasítja a rendszert és a 0. adatblokkra állítja a blokkmutatót.

!BANKWRITE, @visszatérti.kód%, szöveges.változó$ [,blokkszám]

A RAM-lemez használatakor a rendszer (hiba)jelzéseit egy egész típusú változón keresztül közli, ennek a változónak a címét kell átadni az író-olvasó parancsok első paramétereként: @visszatérti.kód%. A !BANKWRITE parancs a 'szöveges.változó$' tartalmát az opcionális blokkszám hiányában abba az adatblokkba írja, amelyre a blokkmutató éppen mutat. 'Blokkszám' megadásakor a 'szöveges.változó$' tartalma a kívánt blokkba kerül. Mindkét esetben a blokkmutató a kiírás után eggyel növekszik. Ha a 'szöveges.változó$' hossza rövidebb, mint a !BANPOPEN parancsban megadott 'blokkhossz', a fennmaradó helyen a blokk régi tartalma változatlan marad, ezért célszerű a változót a blokk hosszára szóközökkel kipárnázni. Ha a változó nem fér be a blokkba, a kilógó részt a rendszer egyszerűen levágja, nem írja felül a következő blokk tartalmát.

!BANKREAD, @visszatérti.kód%, szöveges.változó$ [,blokkszám]

A 'szöveges.változó$'-ba olvassa be azon blokk tartalmát, amelyre a blokkmutató éppen mutat. A kiíráshoz hasonlóan itt is tetszőleges blokkra állítható be a blokkmutató a 'blokkszám' megadásával. A 'visszatérti.kód%'-ból kiolvasható jelzés mindkét parancsnál azonos: ha nem történt hiba, a változó a blokkmutató értékét tartalmazza, hiba esetén tartalma negatív szám:

- 1 : 'EOF', a blokkmutató túlmutat a második RAM-lap területén, a lap megtelt.
- 2 : rendszerhiba memórialapozás közben.

A következő kis program három adatblokkot helyez el a RAM-lemezen. Figyelje meg az adatelemek egységes hosszúságának előállítási módját: vágás vagy párnázás. Az adatelemek elhelyezése a file-ba irandó adatblokkba egyszerűen és biztonságosan elvégezhető a 'MID$' utasítás segítségével.

```
10 rem RAM-LEMEZEN KOZVETLEN HOZZAFERESU FILE KIIRAS
20 rem az adatokat peldakeppen DATA sorokbol olvassuk ki
30 data "Kovacs Jozsef","Budapest IX. Ulloi u. 3","123-456"
40 data "Kecskes Istvan","Debrecen II. Kocsis u. 2","nincs"
50 data "Cica Mica","Csajagarocsoge Poros u. 13","987-654"
60 rem --- nev: max. 20 karakter, cim: max. 30 karakter
70 rem --- telefon: 7 karakter, osszesen 20+30+7=57 karakter
80 rem --- file-nyitas kovetkezik:
90 blokk$= string$(57,32)                    ' ezt toltjuk majd fel
100 !BANKOPEN,57                             ' egy adatblokk hossza
110 vk%=0                                    ' visszaterti.kod%
120 rem adatelemeket egyseges hosszura vagni vagy parnazni
130 for blokkszam = 0 to 2                   ' 0. blokk is van!
140     read nev$                            ' pontosan 20 hosszura
150      if len(nev$)>20 then nev$=left$(nev$,20)
160      if len(nev$)<20 then nev$=nev$+" ": goto 160
170     read cim$                            ' pontosan 30 hosszura
180      if len(cim$)>30 then cim$=left$(cim$,30)
190      if len(cim$)<30 then cim$=cim$+" ": goto 190
200     read telefon$                        ' itt is lehetne vagni
210      if len(telefon$)<7 then telefon$=telefon$+" ": goto 210
```



```
220 rem egyseges hosszusagu adatelemeket elhelyezni a blokk$-ban
230   mid$(blokk$,1,20)=nev$                 ' elso 20 karakter
240   mid$(blokk$,21,30)=cim$                ' kovetkezo 30 kar.
250   mid$(blokk$,51,7)=telefon$             ' utolso 7 karakter
260 rem blokk$ osszeallitva, file-ba irhato
270   !BANKWRITE, @vk%, blokk$, blokkszam    ' kiirja a RAM-lemezre
280   if vk%<0 then goto 1000                ' hiba tortent !!!!!
290 next blokkszam
999 stop
1000 rem itt lehet a hibakezelo rutin
1111 rem folytatjuk !
```

A RAM-lemezen nem szükséges, és nem is lehet file-t lezárni. A megnyitás is csak egyféle, mivel tetszőlegesen keverhetők az író és olvasó műveletek. Egy újabb megnyitás sem törölné a RAM-lemezen található adatokat, csak a blokkmutatót állítaná nullára. A következő programrész az előzőleg kivitt adatokat olvassa be tetszés szerinti sorrendben. Figyelje meg az adatblokkok felszabdalását adatelemekre:

```
500 rem BEOLVASAS KOZVETLEN HOZZAFERESU FILE-BOL
510 cls
520 input "Melyik blokk adatait keri "; blokkszam
530 !BANKREAD, vk%, blokk$, blokkszam
540 if vk%<0 then 1000                       ' hibalehetoseg pl. EOF
545  print "A blokk tartalma: "; blokk$      ' a teljes blokk
550  nev$=mid$(blokk$,1,20)                  ' elso 20 karakter
560  cim$=mid$(blokk$,21,30)                 ' kovetkezo 30 karakter
570  telefon$=mid$(blokk$,51,7)              ' utolso 7 karakter
580 print "Az adatok egyenkent:"
590 print "Nev:      "; nev$
600 print "Cim:      "; cim$
610 print "Telefon:  "; telefon$
620 print: goto 520
```

Közvetlen hozzáférésű file-ban is csak sorosan lehet keresni: egy blokkot beolvasunk, ez az? Ha nem, akkor jöhet a következő blokk és így tovább. A BANKMAN utolsó file-kezelő parancsa a RAM-lemezre írt file azonnali végigkeresését teszi lehetővé az egyes adatblokkok beolvasása nélkül. A műveletek sebességét

össze sem lehet hasonlítani a lemezen levő soros file-ban történő kereséssel. Gyakorlatilag azonnal visszajelzést ad a keresett szöveg blokkszámáról a !BANKFIND parancs.

!BANKFIND, @visszatérti.kód%, keresett.szöveg$
[, első.blokk [, utolsó.blokk]]

Az opcionális 'első.blokk' paraméter hiján a 'keresett.szöveg' keresése a blokkmutatóval kijelölt blokktól indul. Az esetek többségében így sohasem találja meg a 'keresett.szöveg'-et, ezért ne hagyja el az 'első.blokk'-ot! A 'visszatérti.kód%' adja meg azon blokk sorszámát, ahol a 'keresett.szöveg$' elsőként előfordul. Ha tovább kívánunk keresni, a következő blokk számát kell megadni 'első.blokk'-ként. Amennyiben nem kívánjuk végigkerestetni a teljes 64 Kbyte-os második lapot, adjuk meg az 'utolsó.blokk'-ot is. A megtalált 'keresett.szöveg$' helyét nemcsak a 'visszatérti.kód%' tartalmából tudhatjuk meg, hanem a file blokkmutatója is felveszi ezt az értéket, így a következő olvasás vagy írás 'blokkszám' megadása nélkül is a keresett blokkra irányul. A 'visszatérti.kód' negatív értéke a már megismert hibajelzéseken kívül -3-mal jelzi, hogy nem találta meg a 'keresett.szöveg$'-et.

```
!bankfind,vk%,"Kovacs",0,2  - vk%=0, a nulladik blokkban megvan
!bankfind,vk%,"Cica",0,2    - vk%=2, a második blokkban van
!bankfind,vk%,"Hugo",0      - vk%=-3, végigkereste a teljes 64K
                              második lapot, mégsem találta
```

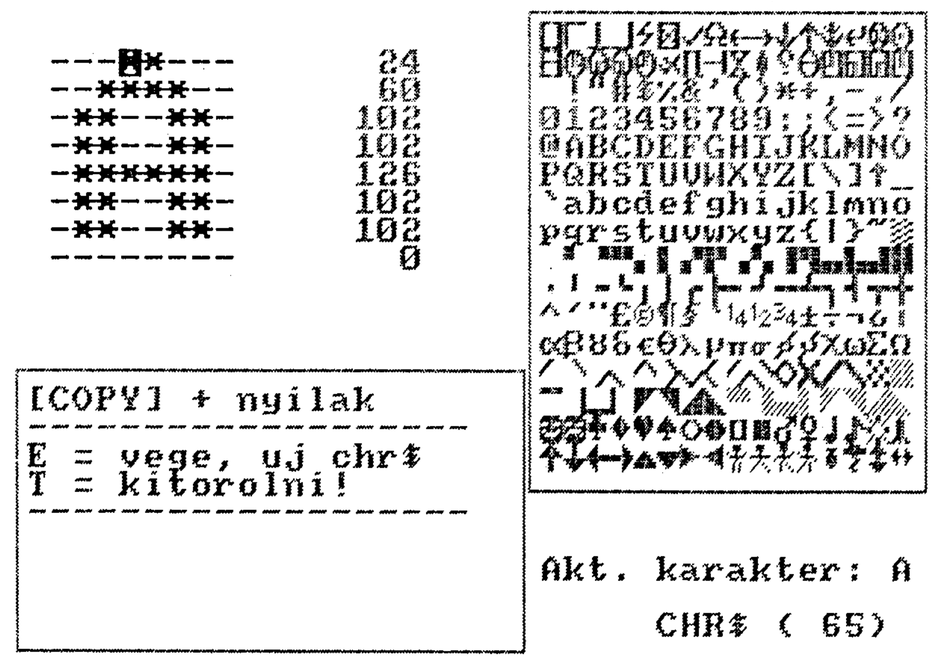
Ha a 'keresett.szöveg$' nem az adatblokk legelején áll, joker karakterekkel kell kipárnázni az összes megelőző karakter helyét. A joker karakter, a file-nevekben megadható '?' (kérdőjelhez) hasonlóan, bármely más karaktert helyettesíthet. Adatblokkok esetében ilyen joker karakter a 'CHR$(0)'. Keressük meg azt az adatblokkot, amelyikben a "Debrecen" előfordul!

```
joker$=string$(20,0)          - az első 20 karakter bármi lehet
keresett.szoveg$=joker$+"Debrecen"
!bankfind,vk%,keresett.szoveg$,0,2     - vk%=1, az első blokk
```

Ha azt az adatblokkot keressük, amelyben a telefonszám

második három számjegye "654", a következőképpen kell eljárni: az első két adatelem együttes hossza 50 karakter, a 'telefon$' első négy karaktere szintén bármi lehet, vagyis az első 54 karakter után kezdődik a minket érdeklő rész:

```
keresett.szoveg$=string$(54,0)+"654"   - az első 54 karakter ...
!bankfind,vk%,keresett.szoveg$,0,2     - vk%=2
```



A KARAKTER GENERATOR program képernyője

Függelék 1. Hangperiódusok

Az alábbi táblázat a CPC 'SOUND' utasítás második paramé-terének, a hang periódusának lehetséges értékeit tartalmazza a hang frekvenciájának függvényében. (A régebbi gépkönyvekben a hangok egy oktávval eltolódtak!)

HANG	FREKVENCIA	PERIODUS	
C	32.703	1911	
C#	34.648	1804	
D	36.708	1703	
D#	38.891	1607	
E	41.203	1517	
F	43.654	1432	oktáv - 3
F#	46.249	1351	
G	48.999	1276	
G#	51.913	1204	
A	55.000	1136	
A#	58.270	1073	
B	61.735	1012	

HANG	FREKVENCIA	PERIODUS	
C	65.406	956	
C#	69.296	902	
D	73.416	851	
D#	77.782	804	
E	82.407	758	
F	87.307	716	oktáv - 2
F#	92.499	676	
G	97.999	638	
G#	103.826	602	
A	110.000	568	
A#	116.541	536	
B	123.471	506	

HANG	FREKVENCIA	PERIODUS
C	130.813	478
C#	138.591	451
D	146.832	426

D#	155.563	402	
E	164.814	379	
F	174.614	358	oktáv - 1
F#	184.997	338	
G	195.998	319	
G#	207.652	301	
A	220.000	284	
A#	233.082	268	
B	246.942	253	

HANG	FREKVENCIA	PERIODUS	
C	261.626	239	középső C
C#	277.183	225	
D	293.665	213	
D#	311.127	201	
E	329.628	190	
F	349.228	179	oktáv 0
F#	369.994	169	
G	391.995	159	
G#	415.305	150	
A	440.000	142	nemzetközi A
A#	466.164	134	
B	493.883	127	

HANG	FREKVENCIA	PERIODUS	
C	523.251	119	
C#	554.365	113	
D	587.330	106	
D#	622.254	100	
E	659.255	95	
F	698.456	89	oktáv + 1
F#	739.989	84	
G	783.991	80	
G#	830.609	75	
A	880.000	71	
A#	932.328	67	
B	987.767	63	

HANG	FREKVENCIA	PERIODUS
C	1046.502	60

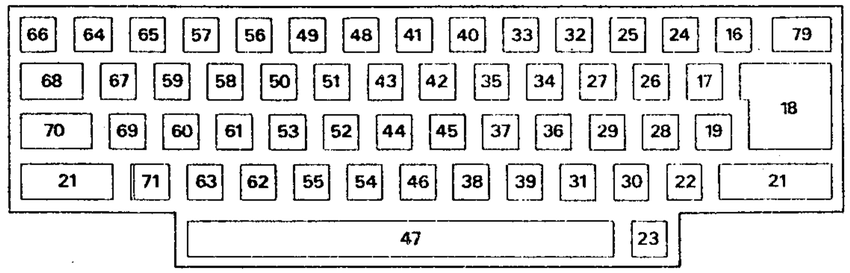
C#	1108.731	56	
D	1174.659	53	
D#	1244.508	50	
E	1318.510	47	
F	1396.913	45	oktáv + 2
F#	1479.978	42	
G	1567.982	40	
G#	1661.219	38	
A	1760.000	36	
A#	1864.655	34	
B	1975.533	32	

HANG	FREKVENCIA	PERIODUS	
C	2093.005	30	
C#	2217.461	28	
D	2349.318	27	
D#	2489.016	25	
E	2637.020	24	
F	2793.826	22	oktáv + 3
F#	2959.955	21	
G	3135.963	20	
G#	3322.438	19	
A	3520.000	18	
A#	3729.310	17	
B	3951.066	16	

Függelék 2. Palettaszínek

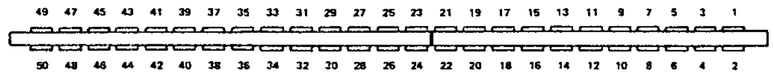
0	fekete	10	szürkéskék	20	világoszöldeskék
1	sötétkék	11	égkék	21	citromzöld
2	világoskék	12	sötétsárga	22	pasztellzöld
3	sötétvörös	13	piszkosfehér	23	pasztellkékeszöld
4	magenta (lila)	14	pasztellkék	24	világossárga
5	világosviolett	15	narancssárga	25	pasztellsárga
6	vörös	16	rózsaszín	26	világítófehér
7	bíbor	17	pasztellmagenta		
8	világosmagenta	18	világoszöld		
9	sötétzöld	19	tengerzöld		

Függelék 3. Billentyűsorszámok



A kurzorbillentyűk és a botkormányok gombjainak sorszámai a 2.7.4 pont alatt találhatók. A számblokk billentyűinek sorszámát a hozzájuk rendelt funkciószámokkal együtt a 2.7.3 pontban ismertettük.

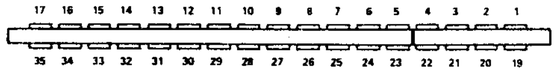
Függelék 4. A bővítőport bekötése



PIN 1	SOUND	PIN 18	A0	PIN 35	$\overline{INT}$
PIN 2	GND	PIN 19	D7	PIN 36	$\overline{NMI}$
PIN 3	A15	PIN 20	D6	PIN 37	$\overline{BUSR2}$
PIN 4	A14	PIN 21	D5	PIN 38	$\overline{BUSAK}$
PIN 5	A13	PIN 22	D4	PIN 39	READY
PIN 6	A12	PIN 23	D3	PIN 40	$\overline{BUS\ RESET}$
PIN 7	A11	PIN 24	D2	PIN 41	$\overline{RESET}$
PIN 8	A10	PIN 25	D1	PIN 42	$\overline{ROMEN}$
PIN 9	A9	PIN 26	D0	PIN 43	$\overline{ROMDIS}$
PIN 10	A8	PIN 27	+5v	PIN 44	$\overline{RAMRD}$
PIN 11	A7	PIN 28	$\overline{MREQ}$	PIN 45	$\overline{RAMDIS}$
PIN 12	A6	PIN 29	$\overline{M1}$	PIN 46	CURSOR
PIN 13	A5	PIN 30	$\overline{RFSH}$	PIN 47	L. PEN
PIN 14	A4	PIN 31	$\overline{IORQ}$	PIN 48	$\overline{EXP}$
PIN 15	A3	PIN 32	$\overline{RD}$	PIN 49	GND
PIN 16	A2	PIN 33	$\overline{WR}$	PIN 50	°
PIN 17	A1	PIN 34	$\overline{HALT}$		

A Z80-as CPU összes vonala hozzáférhető a csatlakozón. Hardver bővítések építésekor figyelembe kell venni, hogy a vonalak nincsenek pufferálva. Külső periféria megcímzése mind a 16 (!) címvonal dekódolásával történik a következő címeken: a felső 8 címvonal &F8 - &FB közötti címet tartalmazhat, az alsó 8 címvonal &E0 - &FE közötti értéket vehet fel. Felhasználható utasítások: 'LD A,adat', 'LD BC,cím', 'OUT (C),A'.

Függelék 5. A nyomtatóport bekötése



PIN	1	$\overline{\text{STROBE}}$	PIN	19	GND
PIN	2	D0	PIN	20	GND
PIN	3	D1	PIN	21	GND
PIN	4	D2	PIN	22	GND
PIN	5	D3	PIN	23	GND
PIN	6	D4	PIN	24	GND
PIN	7	D5	PIN	25	GND
PIN	8	D6	PIN	26	GND
PIN	9	GND	PIN	27	GND
PIN	11	BUSY	PIN	28	GND
PIN	14	GND	PIN	33	GND
PIN	16	GND			

A D7-es vonal a földre van kötve, így csak 7-bites adatátvitel lehetséges. Egyes nyomtatóknál előfordulhat, hogy csak a 14-es vonal elvágásával lehet megakadályozni a kettős soremelést.

Függelék 6. A CPC BASIC hibakódjai és hibajelzései

A lehetséges hibaforrásokat ◆▶ jel választja el egymástól. A lemezegységgel kapcsolatos hibajelzések a 2.9.4 pontban, a hibakódok pedig a Függelék BASIC kulcsszavakat ismertető részében a 'DERR' kulcsszó alatt találhatók.

1 **Unexpected NEXT** (váratlan NEXT)
Az interpreter 'NEXT' utasítással találkozott, egy megelőző 'FOR' nélkül. ◆▶ A 'NEXT' változója nem egyezik meg az utolsó 'FOR'-ban szereplő ciklusváltozóval.

2 **Syntax Error** (szintaxis hiba)
A beadott sort a BASIC valamiért nem tudja értelmezni: hibásan beadott vagy nemlétező parancs, elválasztójelek, zárójelek hiánya vagy felesleges használata, stb.

3 **Unexpected RETURN** (váratlan RETURN)
Nem 'GOSUB' utasítás végrehatása közben találkozott a BASIC egy 'RETURN'-nel.

4 **DATA exhausted** (DATA kimerült)
Egy 'READ' utasítás az utolsó 'DATA' sorban lévő utolsó adatelem után is megpróbál adatelemet beolvasni.

5 **Improper argument** (helytelen argumentum)
Egy utasítás paramétere vagy egy függvény argumentuma nem esik a lehetséges határok közé.

6 **Overflow** (túlcsordulás)
Egy aritmetikai művelet eredménye túlcsorduláshoz vezet: az eredmény már nem ábrázolható a kívánt típusú változóval. Leggyakrabban az egész típusú változókkal kapcsolatban fordul elő, hiszen azok értéke csak -32768 és + 32767 között lehet.

7 **Memory full** (a tár megtelt)
Egy program túl nagy, vagy túl sok változót, nagy tömböket használ. ◆▶ Egymásba ágyazott 'GOSUB', 'WHILE', 'FOR-NEXT' ciklusok is eredményezhetik: 10 a=a+1: print a;: gosub 10. ◆▶ Ha a rendszer file-műveletekhez nem tud 4 Kbyte pufferterületet felszabadítani, szintén ezt a hibajelzést adja ki.

8 **Line does not exist** (nemlétező sorszám)
Nemlétező sorszámú programsorra történt hivatkozás.

9 **Subscript out of range** (nem megfelelő index)
Egy tömb indexe nagyobb, mint ahogy azt előzőleg dimenzionálta, vagy dimenzionálás híján nagyobb 10-nél.

10 **Array already dimensioned** (már dimenzionált tömb)
Megpróbált újradimenzionálni egy már dimenzionált tömböt. Használja először az 'ERASE' utasítást!

11 **Division by zero** (nullával történő osztás)
Ezt természetesen lehetetlen végrehajtani. Ellenőrizze a művelet argumentumait: melyik és miért egyenlő nullával.

12 **Invalid direct command** (érvénytelen közvetlen parancs)
Csak programsorban használható utasítást próbált meg parancsmódban beadni.

13 **Type mismatch** (téves típus)
Valamely változóhoz nem a megfelelő típusú értéket próbálta hozzárendelni. Előfordulhat 'READ' utasítás végrehajtása közben is.

14 **String space full** (szöveges változók helye betelt)
A program túl sok szöveges változót vagy túl nagy szöveges tömböt használ.

15 **String too long** (a szöveg túl hosszú)
Egy szöveges változó hossza meghaladná a 255 karaktert. Szöveges típusú változók összeadása (konkatenációja) közben fordulhat elő.

16 **String expression too complex** (túl bonyolult szöveges művelet) Egyetlen utasításba túl sok szöveges műveletet zsúfolt be: bontsa fel azt kisebb műveletcsoportokra.

17 **Cannot CONTinue** (nem tudom folytatni)
A leállított programot módosítás után próbálta meg 'CONT' paranccsal folytatni! ◀▶ A program fizikai végén túl sem lehet folytatni a végrehajtást.

18 **Unknown user function** (ismeretlen felhasználói függvény)
Nem definiált függvényre történt hivatkozás. 'DEF FN'-nel először definálni kell a felhasználói függvényt!

19 **RESUME missing** (RESUME hiányzik)
'ON ERROR GOTO' hibakezelés végrehajtása közben a program véget ért. A hibakezelő rutint 'RESUME'-mal kell lezárni.

20 **Unexpected RESUME** (váratlan RESUME)
Az interpreter 'ON ERROR GOTO' hibakezelő rutinon kívül találkozott a 'RESUME' utasítással.

21 **Direct command found** (közvetlen paranccsal találkoztam)
Sorszám nélküli programsort próbált beolvastatni a külső tárból. ASC-file-ként kivitt BASIC programot csak 'LINE INPUT'-tal lehel beolvasni!

22 **Operand missing** (hiányzó operandus)
Utasítás paramétere, művelet operandusa, vagy egy függvény argumentuma hiányzik.

23 **Line too long** (túl hosszú sor)
A programsor túl hosszúra sikerült.

24 **EOF met** (a file végét elérte)
A külső tárban lévő file-ból annak végén túl próbált meg adatot beolvasni.

25 **File type error** (téves file típus)
Beolvasásra nem ASCII-file-t próbált megnyitni. ◀▶ Betölteni és futtatni csak a 'SAVE' parancs valamilyen formájával kimentett file-t lehet.

26 **NEXT missing** (hiányzó NEXT)
Egy 'FOR' utasításnak nincs meg a 'NEXT' párja. Okozhatja tévesen írt ciklusváltozó is.

27 **File already open** (a file már meg van nyitva)
'OPENIN' vagy 'OPENOUT' utasítással megnyitott file mellett egy új file-t próbált megnyitni hasonló művelet céljából.

28 **Unknown command** (ismeretlen parancs)
Nem inicializált RSX parancsot próbált meg használni: a rendszer még nem ismeri.

29 **WEND missing** (hiányzó WEND)
'WHILE' utasítással kezdődő ciklus végéről lemaradt a kötelező 'WEND'.

30 **Unexpected WEND** (váratlan WEND)
Az interpreter 'WEND' utasítással találkozott, anélkül hogy egy 'WHILE' ciklus meg lenne nyitva.

Függelék 7. Ajánlott szakirodalom

Dr. Székely Jenő: **Sorvezető** (a Szuper BIT-LET-ben: Z80-as gépi kód) Az Ötlet számítástechnikai különkiadása, Budapest, 1986.

Sztrókay Kálmán: **A Z80 Assembler**
Műszaki Könyvkiadó, Budapest, 1985.

CP/M operációs rendszer (szerkesztette Szenes Katalin)
LSI Alkalmazástechnikai Tanácsadó Szolgálat, Budapest, 1984.

FIRMWARE HANDBUCH Soft 258 (Az Amstrad és Schneider cég által kiadott FIRMWARE leírást több nyelvre is lefordították. Az összes FIRMWARE rutinon kívül a gép hardverének felépítését is részletesen ismerteti.)

DATA BECKER könyvek német nyelven:

CPC 464 INTERN és **CPC 664/6128 INTERN**
(A gép belső működésének kimerítő leírása, különösen ajánlható gépi kódú programozáshoz, hardver bővítéshez stb.)

Das Grosse FLOPPY BUCH
(A lemezegység hardverének leírása, elemi lemezműveletek és az AMSDOS rutinjainak ismertetése, segédprogramok, pl. lemezmonitor, közvetlen hozzáférésű file-ok szimulálása stb.)

Rendszeresen megjelenő folyóiratok (gyakran adnak ki különszámokat is: Sonderheft)
CPC Schneider International (havonta jelenik meg németül);
Schneider Aktiv (havonta jelenik meg német nyelven);
CPC Schneider Magazin (kéthavonta jelenik meg németül);
AMSTRAD USER (havonta jelenik meg angol nyelven).

Az OMIKK és a SZAMALK könyvtáraiban is megtalálható folyóiratok, amelyekben gyakran van CPC-vel foglalkozó cikk:

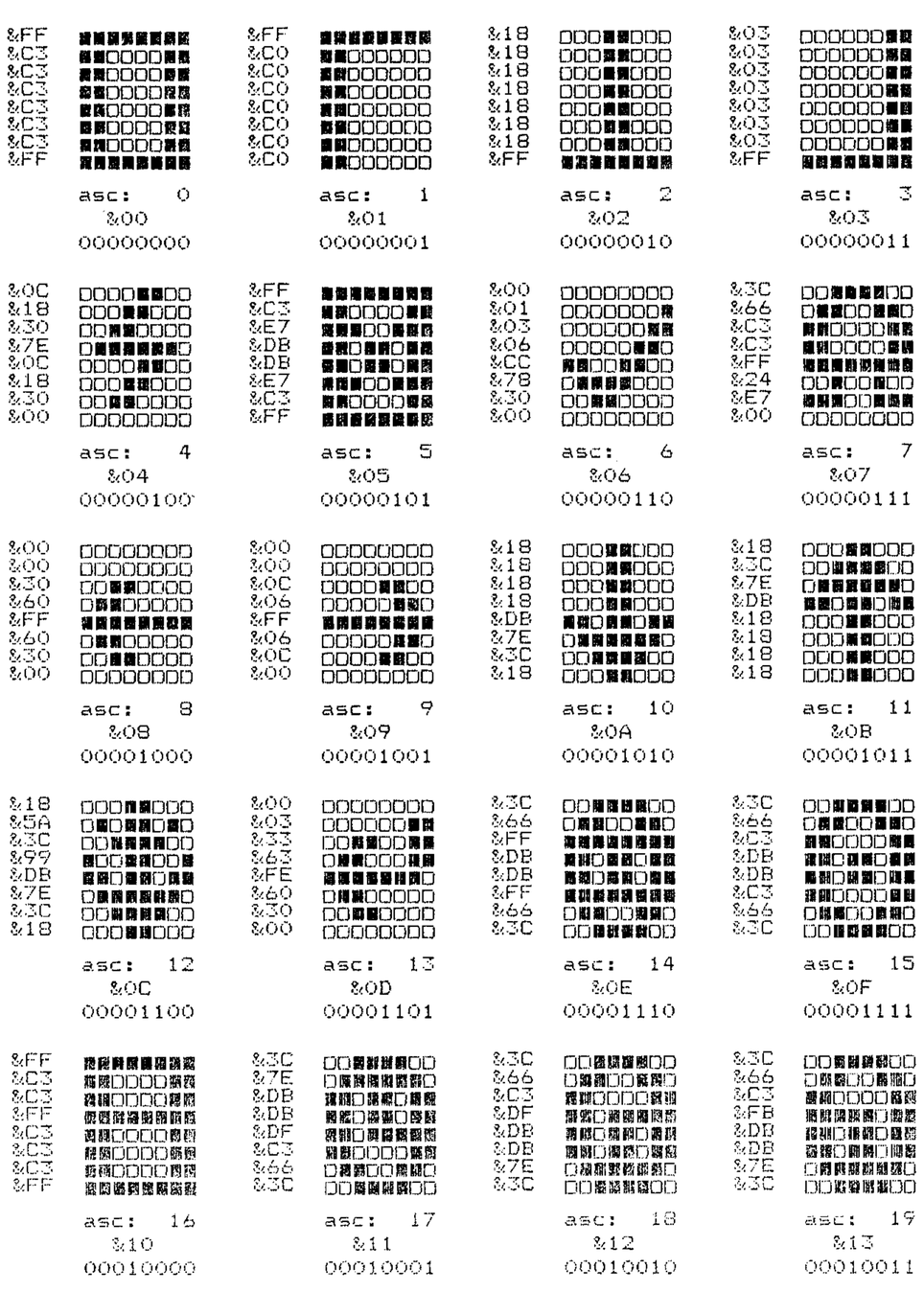
Happy Computer (német nyelvű);
YOUR COMPUTER (angol nyelvű);
Computer Persönlich (német nyelvű).

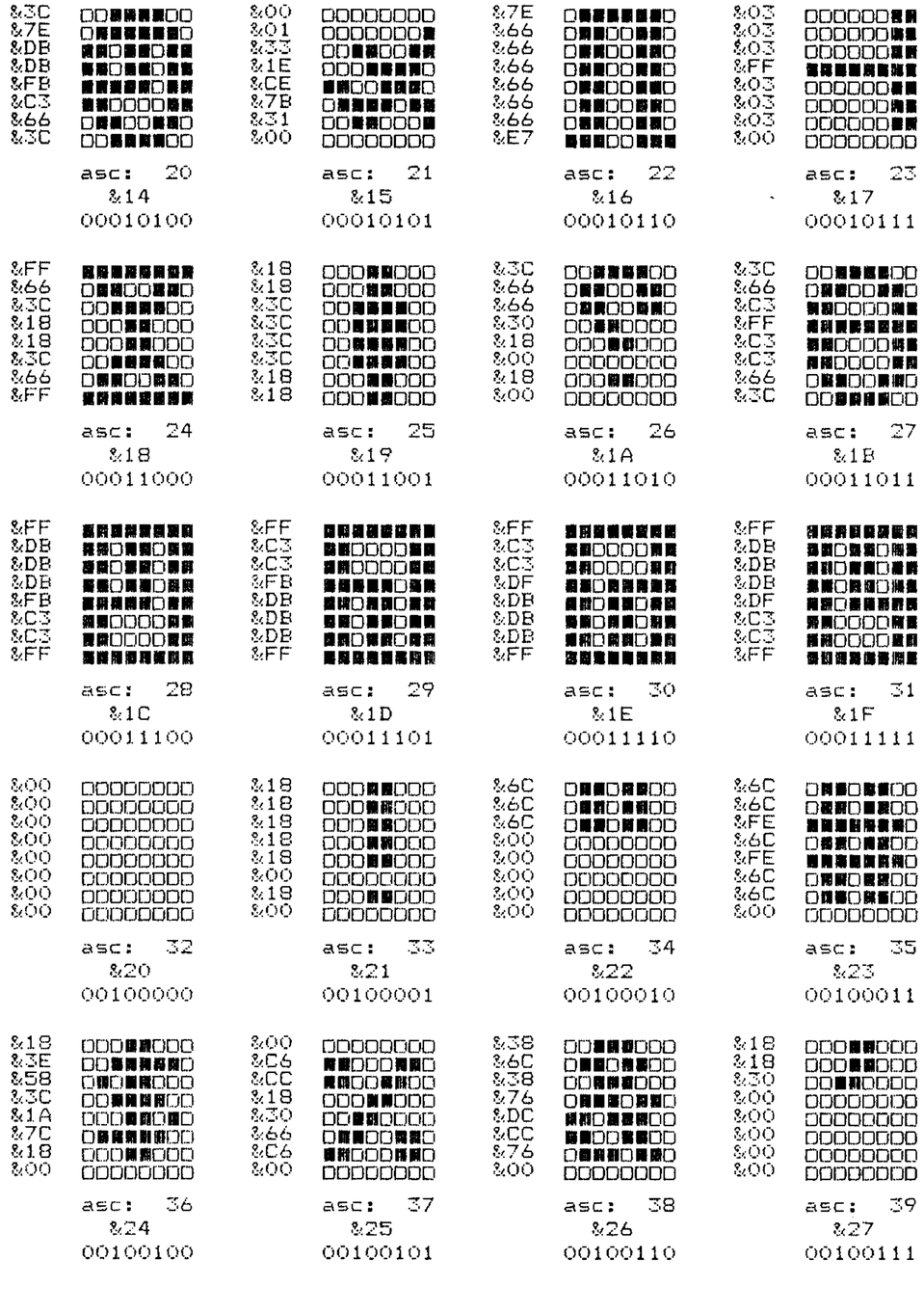
Függelék 8. ASCII kódtáblázat és a CPC karakterkészlete

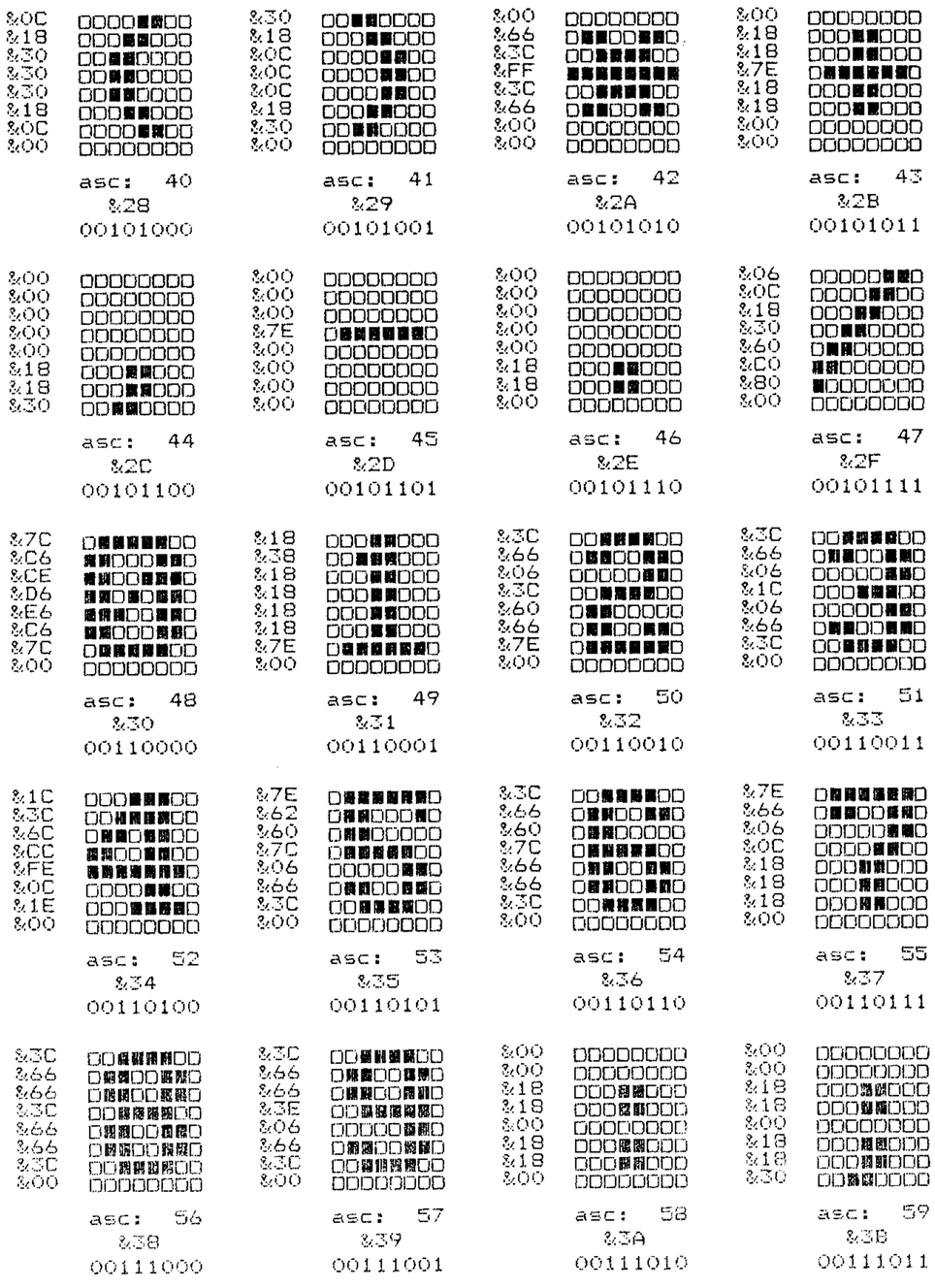
DEC	HEX	CHR$	DEC	HEX	CHR$	DEC	HEX	CHR$	DEC	HEX	CHR$
32	&20:	szóköz	33	&21:	!	34	&22:	"	35	&23:	#
36	&24:	$	37	&25:	%	38	&26:	&	39	&27:	'
40	&28:	(	41	&29:	)	42	&2A:	*	43	&2B:	+
44	&2C:	,	45	&2D:	-	46	&2E:	.	47	&2F:	/
48	&30:	0	49	&31:	1	50	&32:	2	51	&33:	3
52	&34:	4	53	&35:	5	54	&36:	6	55	&37:	7
56	&38:	8	57	&39:	9	58	&3A:	:	59	&3B:	;
60	&3C:	<	61	&3D:	=	62	&3E:	>	63	&3F:	?
64	&40:	@	65	&41:	A	66	&42:	B	67	&43:	C
68	&44:	D	69	&45:	E	70	&46:	F	71	&47:	G
72	&48:	H	73	&49:	I	74	&4A:	J	75	&4B:	K
76	&4C:	L	77	&4D:	M	78	&4E:	N	79	&4F:	O
80	&50:	P	81	&51:	Q	82	&52:	R	83	&53:	S
84	&54:	T	85	&55:	U	86	&56:	V	87	&57:	W
88	&58:	X	89	&59:	Y	90	&5A:	Z	91	&5B:	[
92	&5C:	\	93	&5D:	]	94	&5E:	^	95	&5F:	_
96	&60:	`	97	&61:	a	98	&62:	b	99	&63:	c
100	&64:	d	101	&65:	e	102	&66:	f	103	&67:	g
104	&68:	h	105	&69:	i	106	&6A:	j	107	&6B:	k
108	&6C:	l	109	&6D:	m	110	&6E:	n	111	&6F:	o
112	&70:	p	113	&71:	q	114	&72:	r	115	&73:	s
116	&74:	t	117	&75:	u	118	&76:	v	119	&77:	w
120	&78:	x	121	&79:	y	122	&7A:	z	123	&7B:	{
124	&7C:	¦	125	&7D:	}	126	&7E:	~	127	&7F:	DEL

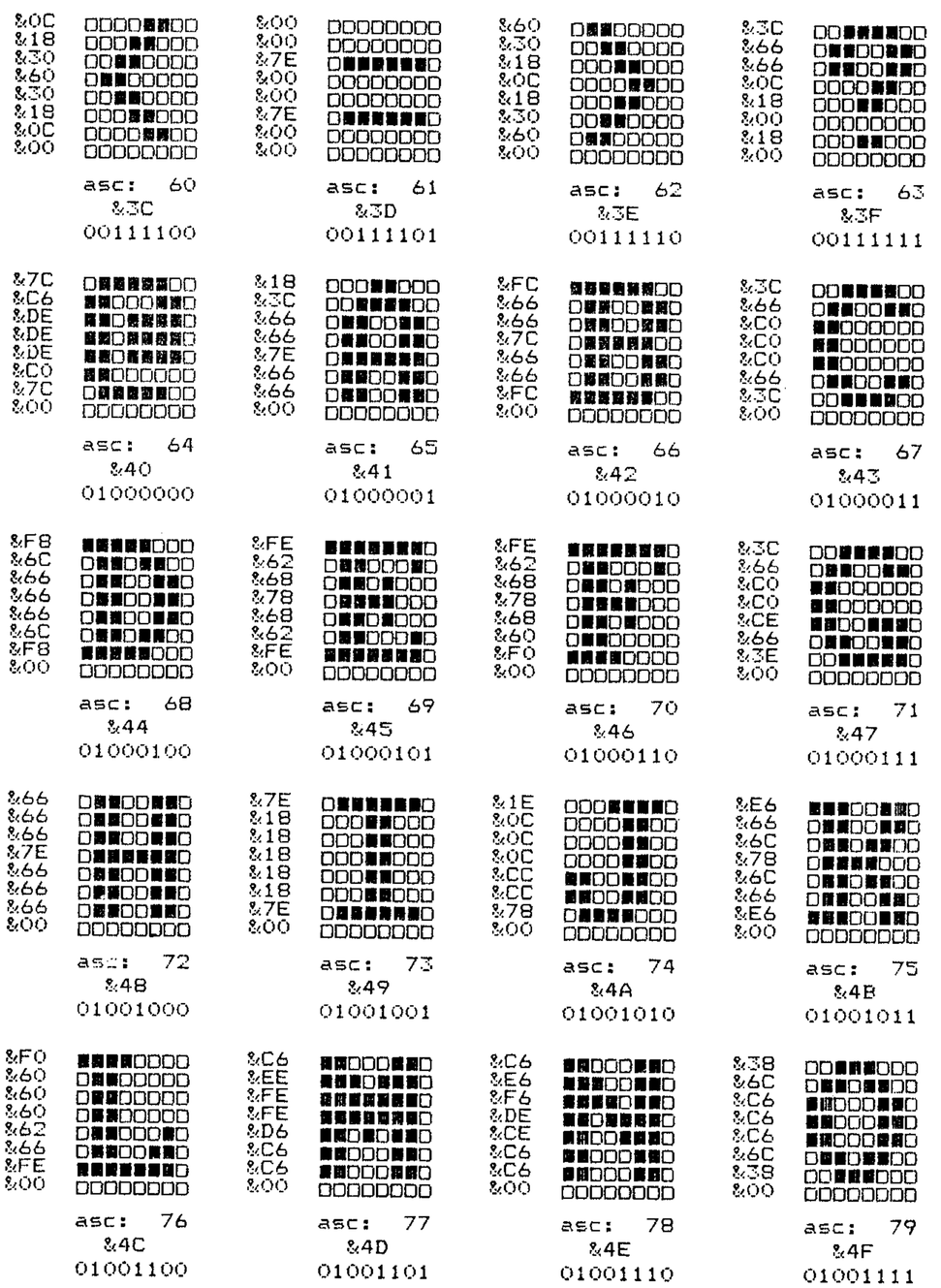
Megjegyzés: A 0-31 kódszámú vezérlő karakterek ismertetése a 2.2.5 pontban található. A CPC-n a 94-es kódszámú '^' helyett '↑' (felfelé mutató nyíl) jelenik meg, a 127-es kódszámú karakter pedig egy pepita négyzet. Ez utóbbi karakter az ASCII szabvány szerint vezérlő funkciót tölt be: kitörli az utoljára kapott karaktert. A legtöbb nyomtatón is így viselkedik.

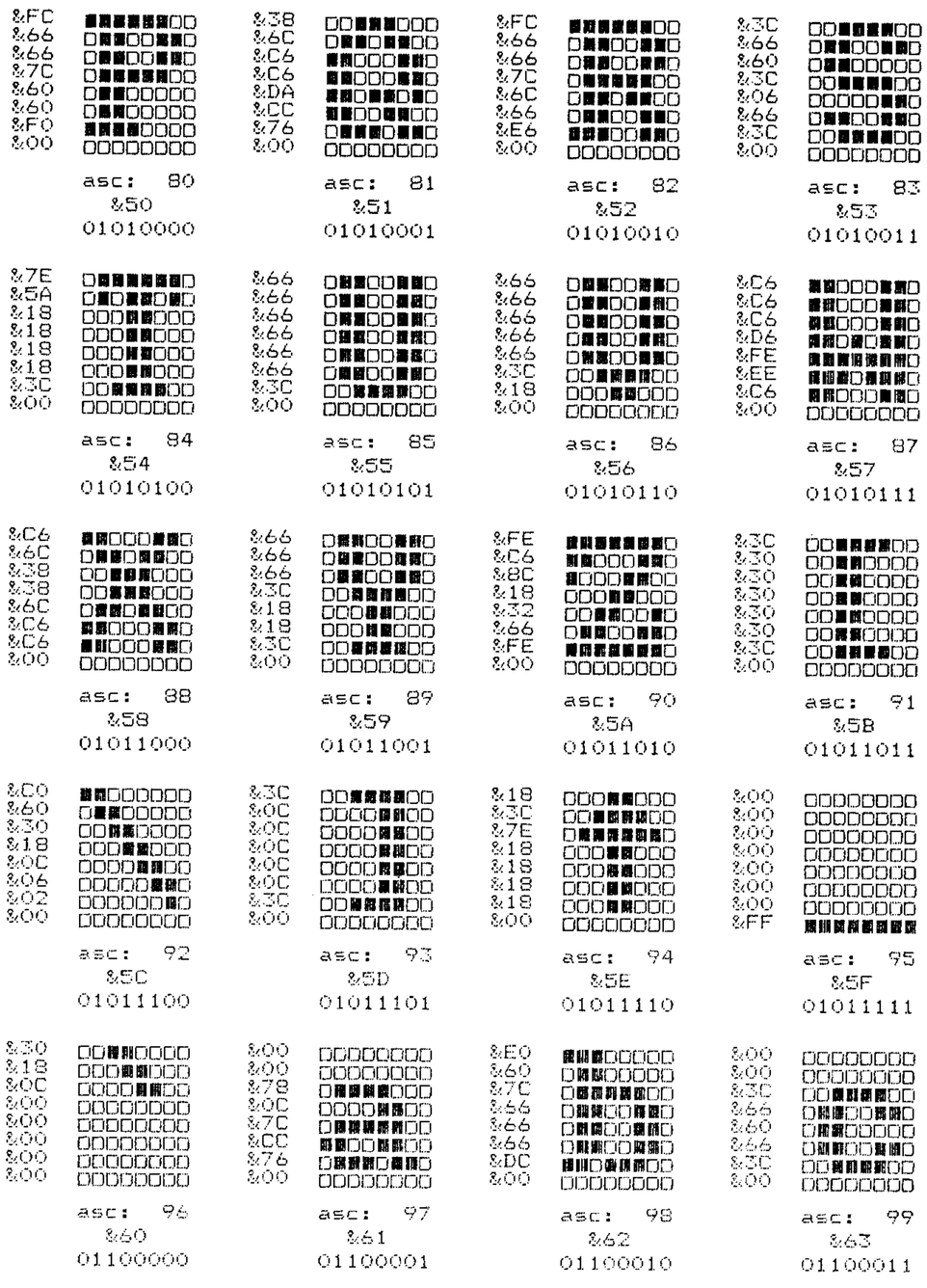
A következő oldalakon a CPC karaktergenerátora által előállított összes karakter képe megtalálható. A karakterek 8*8-as mátrixképe minden sorához megadjuk a tollszínű pontok értékének összegét, amely a 'SYMBOL' parancsban közvetlenül felhasználható hasonló alakú karakterek tervezésekor.

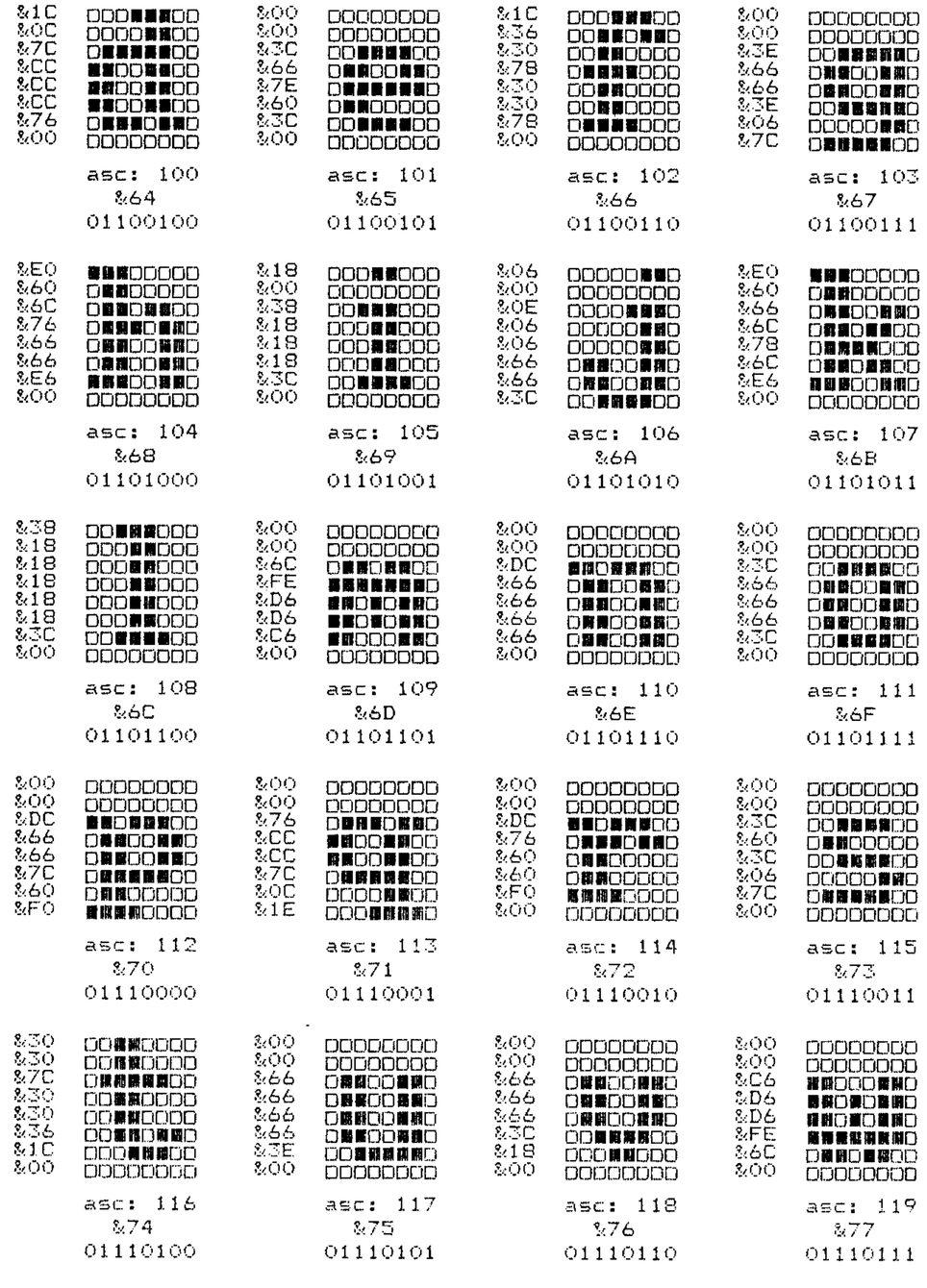

&FF &FF &18 &03
&C3 &C0 &18 &03
&C3 &C0 &18 &03
&C3 &C0 &18 &03
&C3 &C0 &18 &03
&C3 &C0 &18 &03
&C3 &C0 &18 &03
&FF &C0 &FF &FF
asc: 0 asc: 1 asc: 2 asc: 3
&00 &01 &02 &03
00000000 00000001 00000010 00000011
&0C &FF &00 &3C
&18 &C3 &01 &66
&30 &E7 &03 &C3
&7E &DB &06 &FF
&0C &DB &CC &24
&18 &E7 &78 &E7
&30 &C3 &30 &00
&00 &FF &00
asc: 4 asc: 5 asc: 6 asc: 7
&04 &05 &06 &07
00000100 00000101 00000110 00000111
&00 &00 &18 &18
&00 &00 &18 &3C
&30 &0C &18 &7E
&60 &06 &18 &DB
&FF &FF &DB &18
&60 &06 &7E &18
&30 &0C &3C &18
&00 &00 &18
asc: 8 asc: 9 asc: 10 asc: 11
&08 &09 &0A &0B
00001000 00001001 00001010 00001011
&18 &00 &3C &3C
&5A &03 &66 &66
&3C &33 &FF &C3
&99 &63 &DB &DB
&DB &FE &DB &DB
&7E &60 &FF &C3
&3C &30 &66 &66
&18 &00 &3C &3C
asc: 12 asc: 13 asc: 14 asc: 15
&0C &0D &0E &0F
00001100 00001101 00001110 00001111
&FF &3C &3C &3C
&C3 &7E &66 &66
&C3 &DB &C3 &C3
&FF &DB &DF &FB
&C3 &DF &DB &DB
&C3 &C3 &DB &DB
&C3 &66 &7E &7E
&FF &3C &3C &3C
asc: 16 asc: 17 asc: 18 asc: 19
&10 &11 &12 &13
00010000 00010001 00010010 00010011

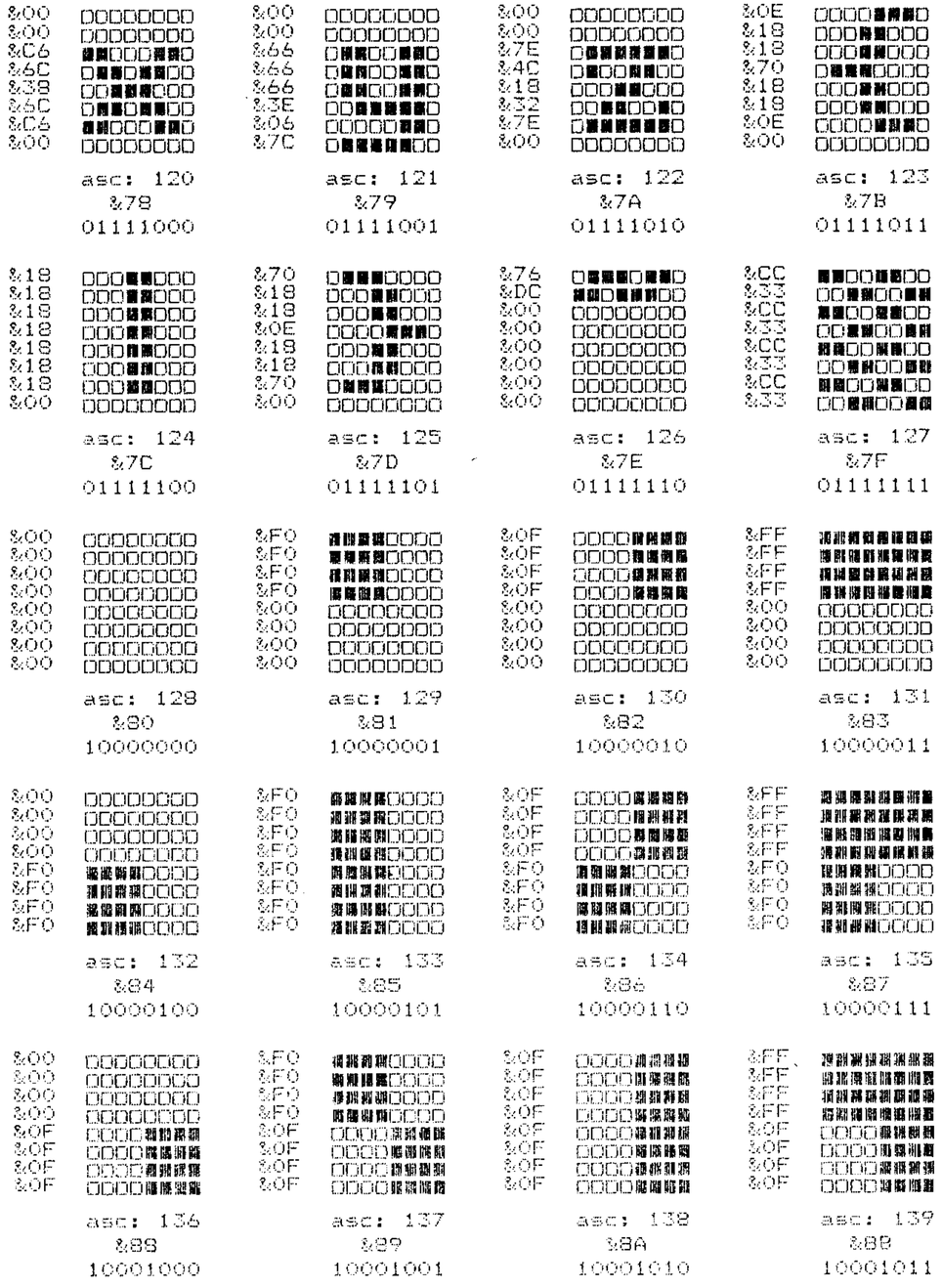

&3C &00 &7E &03
&7E &01 &66 &03
&DB &33 &66 &03
&DB &1E &66 &FF
&FB &CE &66 &03
&C3 &7B &66 &03
&66 &31 &66 &03
&3C &00 &E7 &00
asc: 20 asc: 21 asc: 22 asc: 23
&14 &15 &16 &17
00010100 00010101 00010110 00010111
&FF &18 &3C &3C
&66 &18 &66 &66
&3C &3C &66 &C3
&18 &3C &30 &FF
&18 &3C &18 &C3
&3C &3C &00 &C3
&66 &18 &00 &66
&FF &18 &00 &3C
asc: 24 asc: 25 asc: 26 asc: 27
&18 &19 &1A &1B
00011000 00011001 00011010 00011011
&FF &FF &FF &FF
&DB &C3 &C3 &DB
&DB &C3 &C3 &DB
&DB &FB &DF &DB
&FB &DB &DB &DF
&C3 &DB &DB &C3
&C3 &DB &DB &C3
&FF &FF &FF &FF
asc: 28 asc: 29 asc: 30 asc: 31
&1C &1D &1E &1F
00011100 00011101 00011110 00011111
&00 &18 &6C &6C
&00 &18 &6C &6C
&00 &18 &6C &FE
&00 &18 &00 &6C
&00 &18 &00 &FE
&00 &00 &00 &6C
&00 &18 &00 &6C
&00 &00 &00 &00
asc: 32 asc: 33 asc: 34 asc: 35
&20 &21 &22 &23
00100000 00100001 00100010 00100011
&18 &00 &38 &18
&3E &C6 &6C &18
&58 &CC &38 &30
&3C &18 &76 &00
&1A &30 &DC &00
&7C &66 &CC &00
&18 &C6 &76 &00
&00 &00 &00 &00
asc: 36 asc: 37 asc: 38 asc: 39
&24 &25 &26 &27
00100100 00100101 00100110 00100111

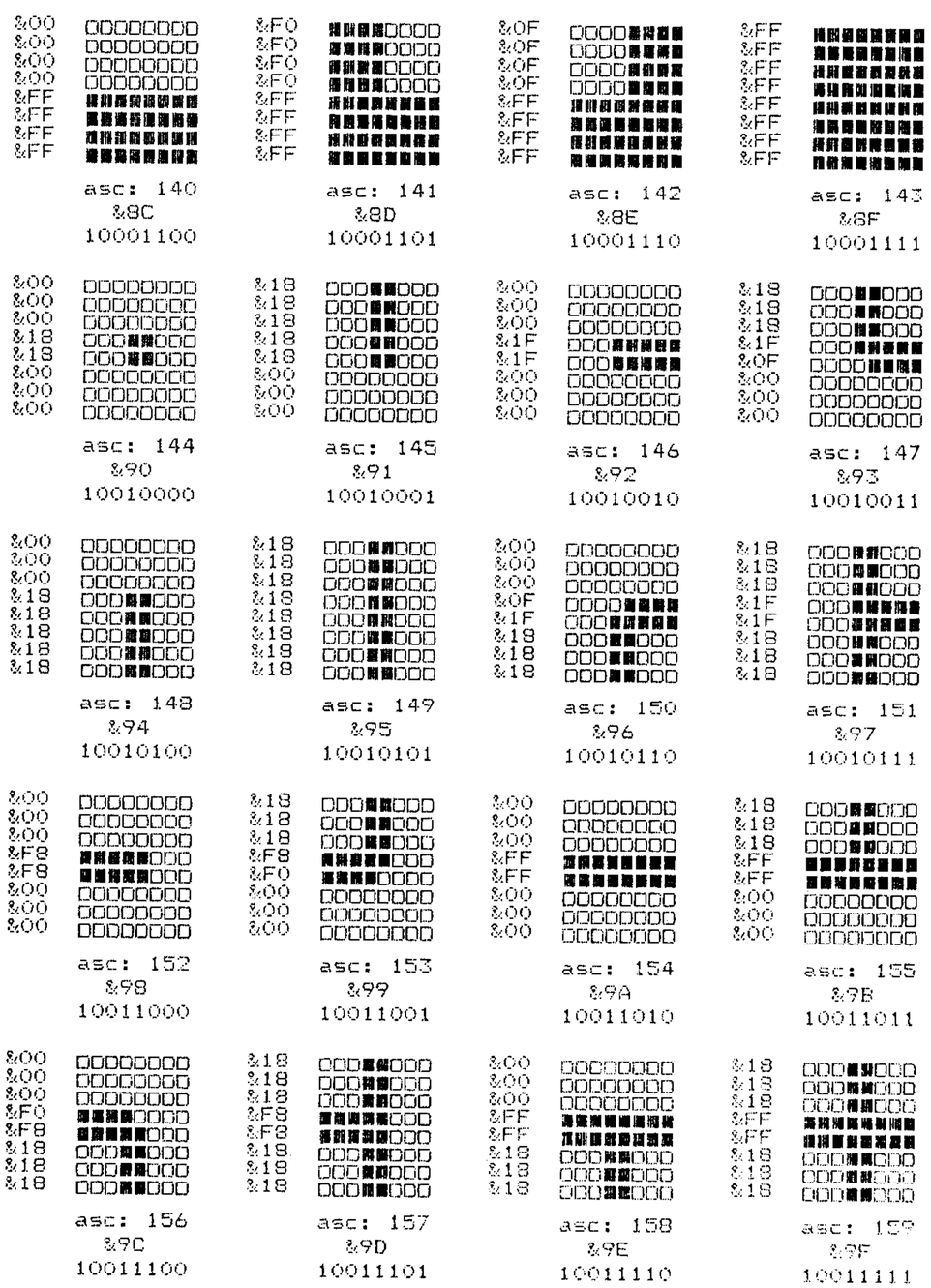

asc: 40
&28
00101000
asc: 41
&29
00101001
asc: 42
&2A
00101010
asc: 43
&2B
00101011
asc: 44
&2C
00101100
asc: 45
&2D
00101101
asc: 46
&2E
00101110
asc: 47
&2F
00101111
asc: 48
&30
00110000
asc: 49
&31
00110001
asc: 50
&32
00110010
asc: 51
&33
00110011
asc: 52
&34
00110100
asc: 53
&35
00110101
asc: 54
&36
00110110
asc: 55
&37
00110111
asc: 56
&38
00111000
asc: 57
&39
00111001
asc: 58
&3A
00111010
asc: 59
&3B
00111011

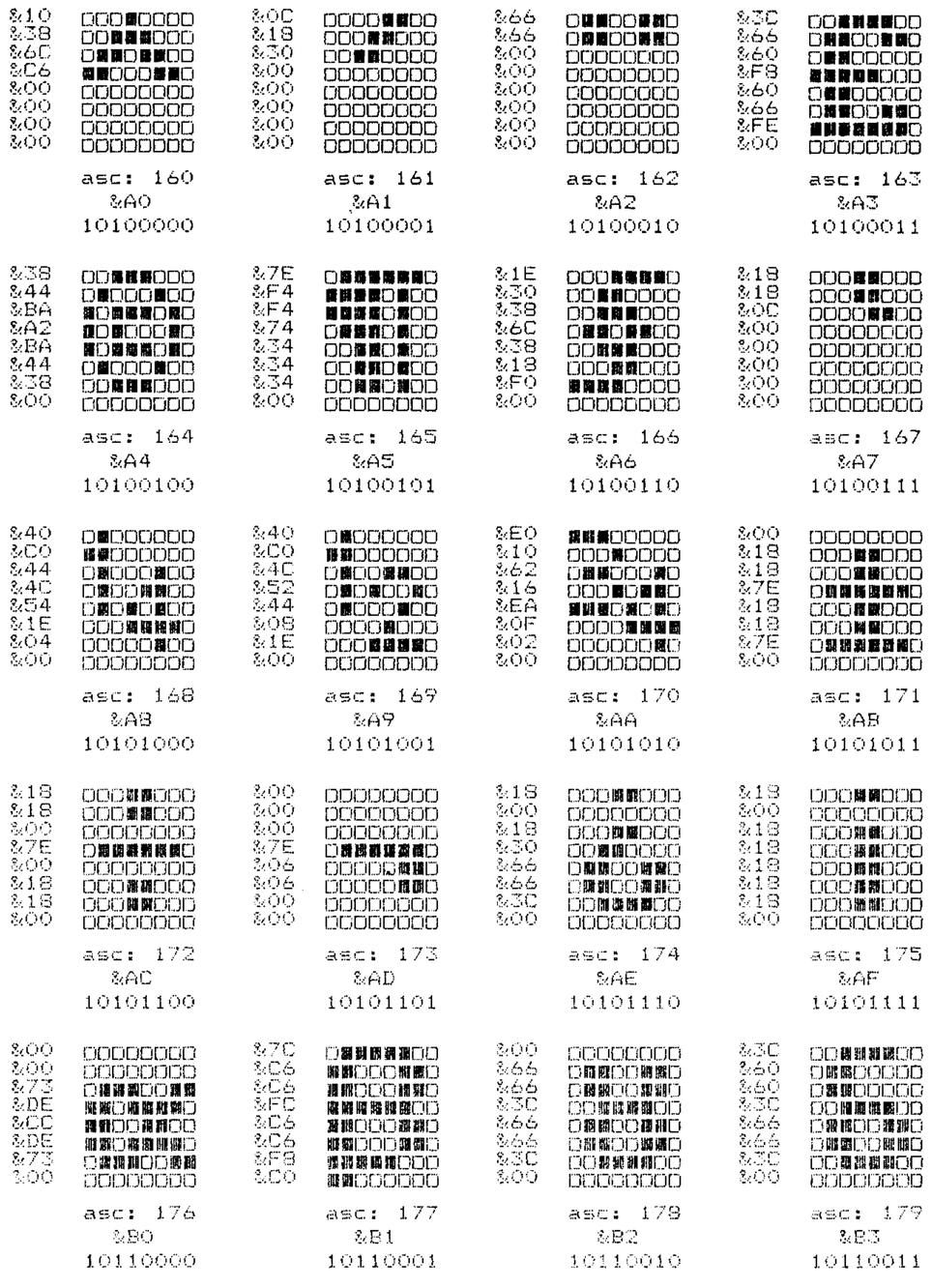

asc: 60
&3C
00111100
asc: 61
&3D
00111101
asc: 62
&3E
00111110
asc: 63
&3F
00111111
asc: 64
&40
01000000
asc: 65
&41
01000001
asc: 66
&42
01000010
asc: 67
&43
01000011
asc: 68
&44
01000100
asc: 69
&45
01000101
asc: 70
&46
01000110
asc: 71
&47
01000111
asc: 72
&48
01001000
asc: 73
&49
01001001
asc: 74
&4A
01001010
asc: 75
&4B
01001011
asc: 76
&4C
01001100
asc: 77
&4D
01001101
asc: 78
&4E
01001110
asc: 79
&4F
01001111

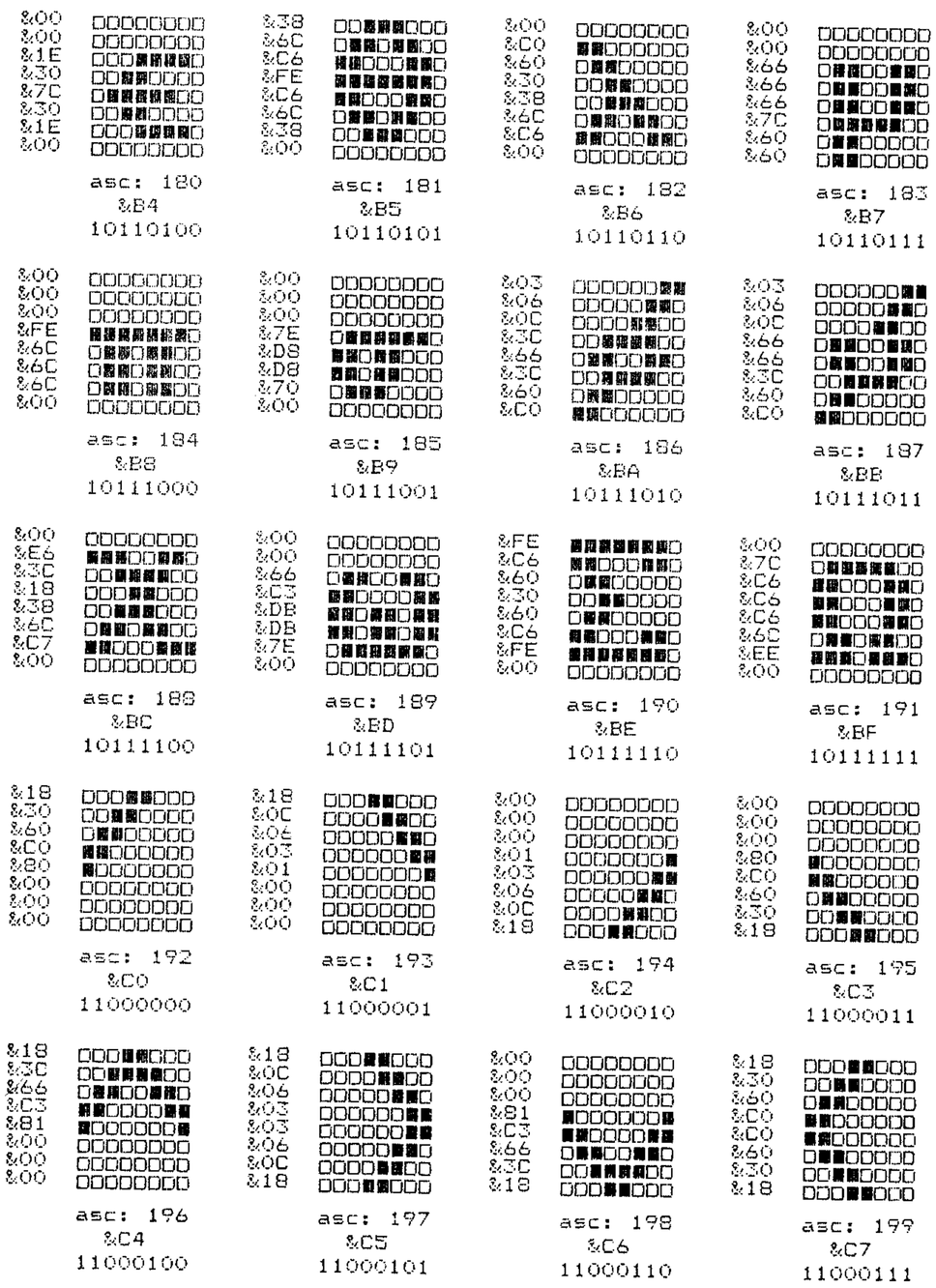

asc: 80 &50 01010000
asc: 81 &51 01010001
asc: 82 &52 01010010
asc: 83 &53 01010011
asc: 84 &54 01010100
asc: 85 &55 01010101
asc: 86 &56 01010110
asc: 87 &57 01010111
asc: 88 &58 01011000
asc: 89 &59 01011001
asc: 90 &5A 01011010
asc: 91 &5B 01011011
asc: 92 &5C 01011100
asc: 93 &5D 01011101
asc: 94 &5E 01011110
asc: 95 &5F 01011111
asc: 96 &60 01100000
asc: 97 &61 01100001
asc: 98 &62 01100010
asc: 99 &63 01100011

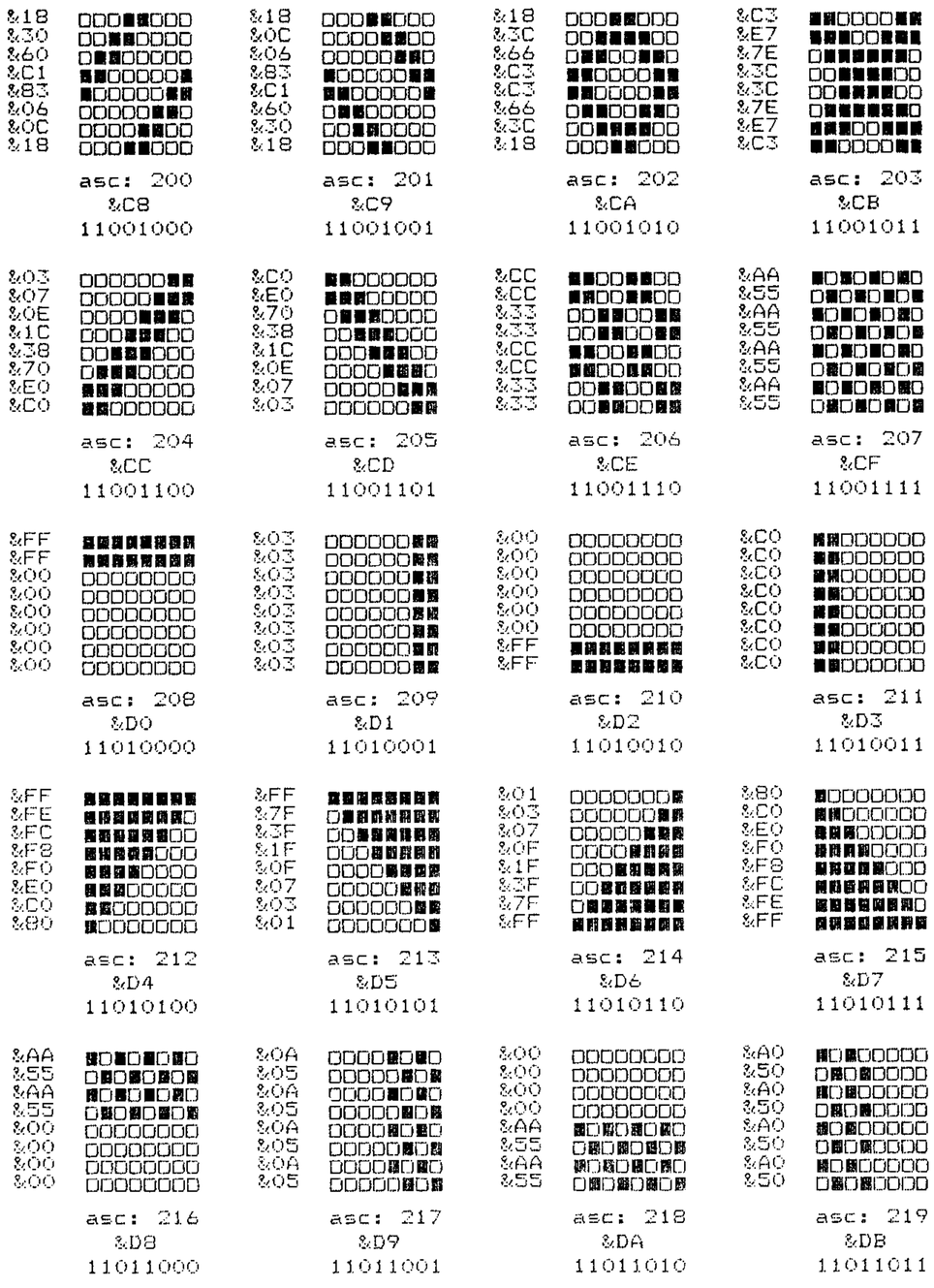

asc: 100 &64 01100100
asc: 101 &65 01100101
asc: 102 &66 01100110
asc: 103 &67 01100111
asc: 104 &68 01101000
asc: 105 &69 01101001
asc: 106 &6A 01101010
asc: 107 &6B 01101011
asc: 108 &6C 01101100
asc: 109 &6D 01101101
asc: 110 &6E 01101110
asc: 111 &6F 01101111
asc: 112 &70 01110000
asc: 113 &71 01110001
asc: 114 &72 01110010
asc: 115 &73 01110011
asc: 116 &74 01110100
asc: 117 &75 01110101
asc: 118 &76 01110110
asc: 119 &77 01110111

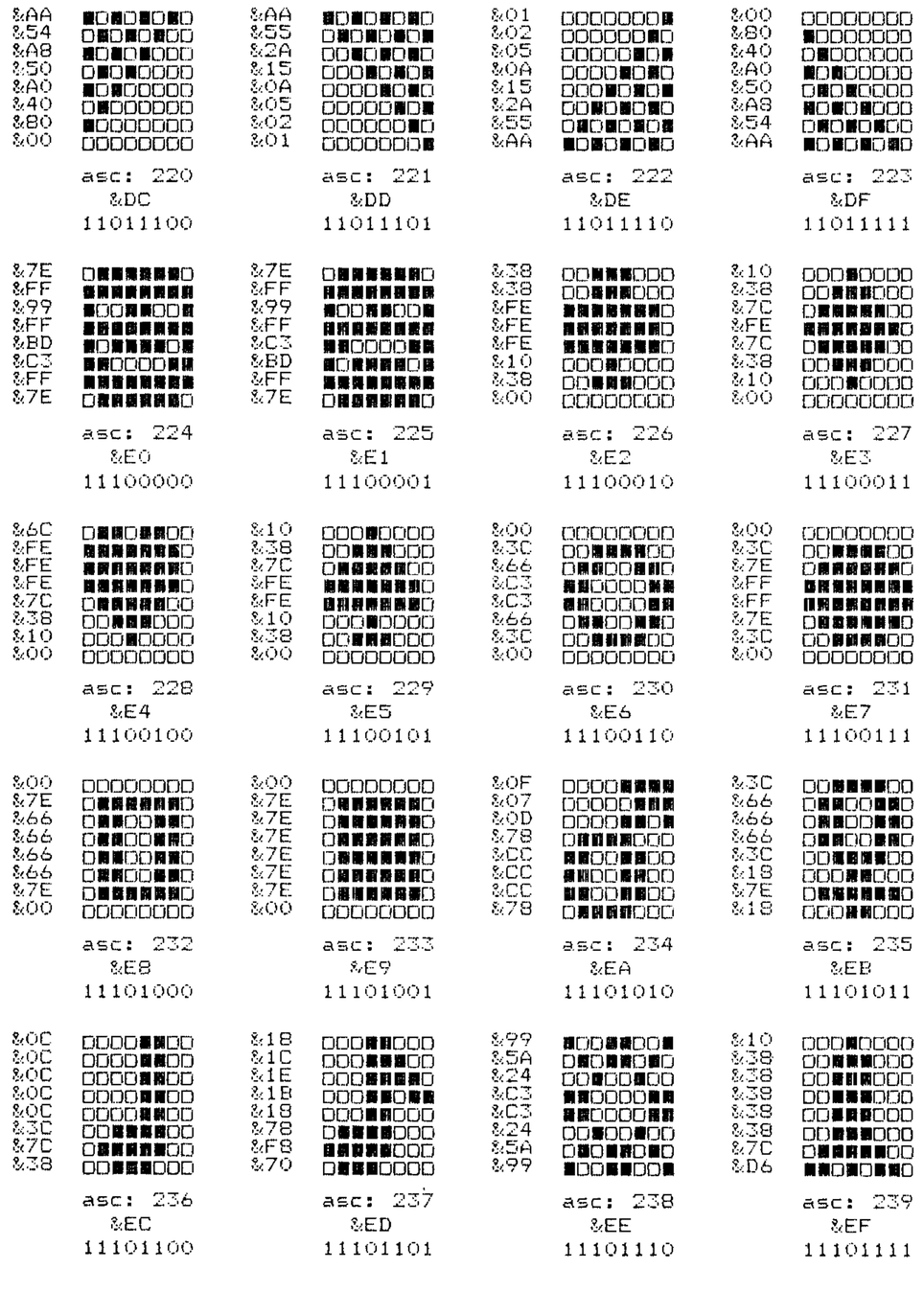

&00 &00 &C6 &6C &38 &6C &C6 &00
asc: 120 &78 01111000
&00 &00 &66 &66 &66 &3E &06 &7C
asc: 121 &79 01111001
&00 &00 &7E &4C &18 &32 &7E &00
asc: 122 &7A 01111010
&0E &18 &18 &70 &18 &19 &0E &00
asc: 123 &7B 01111011
&18 &18 &18 &18 &18 &18 &18 &00
asc: 124 &7C 01111100
&70 &18 &18 &0E &18 &18 &70 &00
asc: 125 &7D 01111101
&76 &DC &00 &00 &00 &00 &00 &00
asc: 126 &7E 01111110
&CC &CC &CC &CC &CC &CC &CC &CC
asc: 127 &7F 01111111
&00 &00 &00 &00 &00 &00 &00 &00
asc: 128 &80 10000000
&F0 &F0 &F0 &F0 &00 &00 &00 &00
asc: 129 &81 10000001
&0F &0F &0F &0F &00 &00 &00 &00
asc: 130 &82 10000010
&FF &FF &FF &FF &00 &00 &00 &00
asc: 131 &83 10000011
&00 &00 &00 &00 &F0 &F0 &F0 &F0
asc: 132 &84 10000100
&F0 &F0 &F0 &F0 &F0 &F0 &F0 &F0
asc: 133 &85 10000101
&0F &0F &0F &0F &F0 &F0 &F0 &F0
asc: 134 &86 10000110
&FF &FF &FF &FF &F0 &F0 &F0 &F0
asc: 135 &87 10000111
&00 &00 &00 &00 &0F &0F &0F &0F
asc: 136 &88 10001000
&F0 &F0 &F0 &F0 &0F &0F &0F &0F
asc: 137 &89 10001001
&0F &0F &0F &0F &0F &0F &0F &0F
asc: 138 &8A 10001010
&FF &FF &FF &FF &0F &0F &0F &0F
asc: 139 &8B 10001011

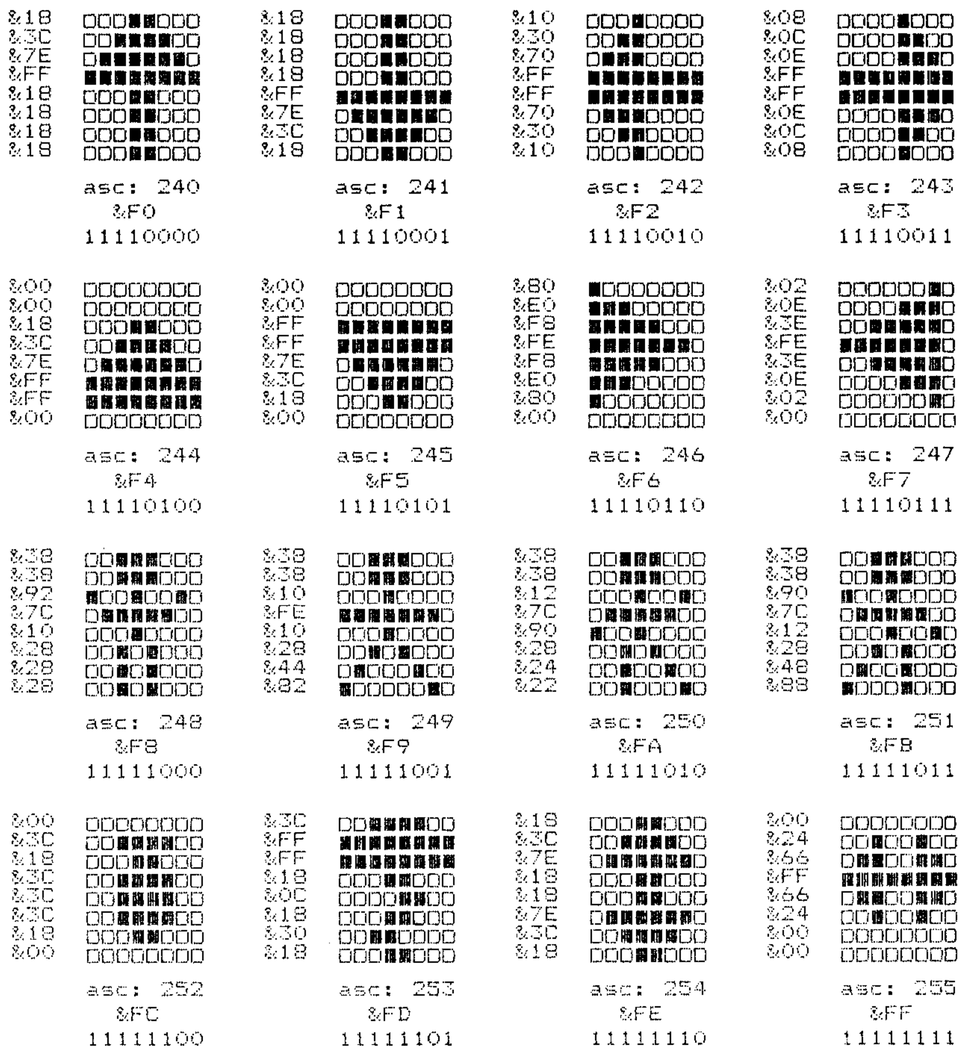

&00 &00 &00 &00 &FF &FF &FF &FF
asc: 140 &8C 10001100
&F0 &F0 &F0 &F0 &FF &FF &FF &FF
asc: 141 &8D 10001101
&0F &0F &0F &0F &FF &FF &FF &FF
asc: 142 &8E 10001110
&FF &FF &FF &FF &FF &FF &FF &FF
asc: 143 &8F 10001111
&00 &00 &00 &18 &18 &18 &18 &00
asc: 144 &90 10010000
&18 &18 &19 &18 &18 &00 &00 &00
asc: 145 &91 10010001
&00 &00 &00 &1F &1F &00 &00 &00
asc: 146 &92 10010010
&18 &18 &18 &1F &0F &00 &00 &00
asc: 147 &93 10010011
&00 &00 &00 &18 &18 &18 &18 &18
asc: 148 &94 10010100
&18 &18 &18 &19 &18 &19 &18 &18
asc: 149 &95 10010101
&00 &00 &00 &0F &1F &18 &18 &18
asc: 150 &96 10010110
&18 &18 &18 &1F &1F &18 &18 &18
asc: 151 &97 10010111
&00 &00 &00 &F8 &F8 &00 &00 &00
asc: 152 &98 10011000
&18 &18 &18 &F8 &F0 &00 &00 &00
asc: 153 &99 10011001
&00 &00 &00 &FF &FF &00 &00 &00
asc: 154 &9A 10011010
&18 &18 &18 &FF &FF &00 &00 &00
asc: 155 &9B 10011011
&00 &00 &00 &F0 &F8 &18 &18 &18
asc: 156 &9C 10011100
&18 &18 &18 &F8 &F8 &18 &18 &18
asc: 157 &9D 10011101
&00 &00 &00 &FF &FF &18 &18 &18
asc: 158 &9E 10011110
&18 &18 &18 &FF &FF &18 &18 &18
asc: 159 &9F 10011111


&10 &38 &6C &C6 &00 &00 &00 &00
asc: 160 &A0 10100000
&0C &18 &30 &00 &00 &00 &00 &00
asc: 161 &A1 10100001
&66 &66 &00 &00 &00 &00 &00 &00
asc: 162 &A2 10100010
&3C &66 &60 &F8 &60 &66 &FE &00
asc: 163 &A3 10100011
&38 &44 &BA &A2 &BA &44 &38 &00
asc: 164 &A4 10100100
&7E &F4 &F4 &74 &34 &34 &34 &00
asc: 165 &A5 10100101
&1E &30 &38 &6C &38 &18 &F0 &00
asc: 166 &A6 10100110
&18 &18 &0C &00 &00 &00 &00 &00
asc: 167 &A7 10100111
&40 &C0 &44 &4C &54 &1E &04 &00
asc: 168 &A8 10101000
&40 &C0 &4C &52 &44 &08 &1E &00
asc: 169 &A9 10101001
&E0 &10 &62 &16 &EA &0F &02 &00
asc: 170 &AA 10101010
&00 &18 &18 &7E &18 &18 &18 &00
asc: 171 &AB 10101011
&18 &18 &00 &7E &00 &18 &18 &00
asc: 172 &AC 10101100
&00 &00 &00 &7E &06 &06 &00 &00
asc: 173 &AD 10101101
&18 &00 &18 &30 &66 &66 &3C &00
asc: 174 &AE 10101110
&18 &18 &00 &18 &18 &18 &18 &00
asc: 175 &AF 10101111
&00 &00 &73 &DE &CC &DE &73 &00
asc: 176 &B0 10110000
&7C &C6 &C6 &FC &C6 &C6 &F8 &C0
asc: 177 &B1 10110001
&00 &66 &66 &3C &66 &66 &3C &00
asc: 178 &B2 10110010
&3C &60 &60 &3C &66 &66 &3C &00
asc: 179 &B3 10110011


&00 &00 &1E &30 &7C &30 &1E &00
asc: 180 &B4 10110100
&38 &6C &C6 &FE &C6 &6C &38 &00
asc: 181 &B5 10110101
&00 &C0 &60 &30 &38 &6C &C6 &00
asc: 182 &B6 10110110
&00 &00 &66 &66 &66 &7C &60 &60
asc: 183 &B7 10110111
&00 &00 &00 &FE &6C &6C &6C &00
asc: 184 &B8 10111000
&00 &00 &00 &7E &D8 &D8 &70 &00
asc: 185 &B9 10111001
&03 &06 &0C &3C &66 &3C &60 &C0
asc: 186 &BA 10111010
&03 &06 &66 &66 &3C &60 &C0
asc: 187 &BB 10111011
&00 &E6 &3C &18 &38 &6C &C7 &00
asc: 188 &BC 10111100
&00 &00 &66 &C3 &DB &DB &7E &00
asc: 189 &BD 10111101
&FE &C6 &60 &30 &60 &C6 &FE &00
asc: 190 &BE 10111110
&00 &7C &C6 &C6 &C6 &6C &EE &00
asc: 191 &BF 10111111
&18 &30 &60 &C0 &80 &00 &00 &00
asc: 192 &C0 11000000
&18 &0C &06 &03 &01 &00 &00 &00
asc: 193 &C1 11000001
&00 &00 &00 &01 &03 &06 &0C &18
asc: 194 &C2 11000010
&00 &00 &00 &80 &C0 &60 &30 &18
asc: 195 &C3 11000011
&18 &3C &66 &C3 &81 &00 &00 &00
asc: 196 &C4 11000100
&18 &0C &06 &03 &03 &06 &0C &18
asc: 197 &C5 11000101
&00 &00 &00 &81 &C3 &66 &3C &18
asc: 198 &C6 11000110
&18 &30 &60 &C0 &C0 &60 &30 &18
asc: 199 &C7 11000111


&18 &18 &18 &C3
&30 &0C &3C &E7
&60 &06 &66 &7E
&C1 &83 &C3 &3C
&83 &C1 &C3 &3C
&06 &60 &66 &7E
&0C &30 &3C &E7
&18 &18 &18 &C3
asc: 200 &C8 11001000
asc: 201 &C9 11001001
asc: 202 &CA 11001010
asc: 203 &CB 11001011
&03 &C0 &CC &AA
&07 &E0 &CC &55
&0E &70 &33 &AA
&1C &38 &33 &55
&38 &1C &CC &AA
&70 &0E &CC &55
&E0 &07 &33 &AA
&C0 &03 &33 &55
asc: 204 &CC 11001100
asc: 205 &CD 11001101
asc: 206 &CE 11001110
asc: 207 &CF 11001111
&FF &03 &00 &C0
&FF &03 &00 &C0
&00 &03 &00 &C0
&00 &03 &00 &C0
&00 &03 &00 &C0
&00 &03 &00 &C0
&00 &03 &FF &C0
&00 &03 &FF &C0
asc: 208 &D0 11010000
asc: 209 &D1 11010001
asc: 210 &D2 11010010
asc: 211 &D3 11010011
&FF &FF &01 &80
&FE &7F &03 &C0
&FC &3F &07 &E0
&F8 &1F &0F &F0
&F0 &0F &1F &F8
&E0 &07 &3F &FC
&C0 &03 &7F &FE
&80 &01 &FF &FF
asc: 212 &D4 11010100
asc: 213 &D5 11010101
asc: 214 &D6 11010110
asc: 215 &D7 11010111
&AA &0A &00 &A0
&55 &05 &00 &50
&AA &0A &00 &A0
&55 &05 &00 &50
&AA &0A &00 &A0
&55 &05 &00 &50
&AA &0A &00 &A0
&55 &05 &00 &50
asc: 216 &D8 11011000
asc: 217 &D9 11011001
asc: 218 &DA 11011010
asc: 219 &DB 11011011


&AA &AA &01 &00
&54 &55 &02 &80
&A8 &2A &05 &40
&50 &15 &0A &A0
&A0 &0A &15 &50
&40 &05 &2A &A8
&80 &02 &55 &54
&00 &01 &AA &AA
asc: 220 &DC 11011100
asc: 221 &DD 11011101
asc: 222 &DE 11011110
asc: 223 &DF 11011111
&7E &7E &38 &10
&FF &FF &38 &38
&99 &99 &FE &7C
&FF &FF &FE &FE
&BD &C3 &FE &7C
&C3 &BD &10 &38
&FF &FF &38 &10
&7E &7E &00 &00
asc: 224 &E0 11100000
asc: 225 &E1 11100001
asc: 226 &E2 11100010
asc: 227 &E3 11100011
&6C &10 &00 &00
&FE &38 &3C &3C
&FE &7C &66 &7E
&FE &FE &C3 &FF
&7C &FE &C3 &FF
&38 &10 &66 &7E
&10 &38 &3C &3C
&00 &00 &00 &00
asc: 228 &E4 11100100
asc: 229 &E5 11100101
asc: 230 &E6 11100110
asc: 231 &E7 11100111
&00 &00 &0F &3C
&7E &7E &07 &66
&66 &7E &0D &66
&66 &7E &78 &66
&66 &7E &CC &3C
&66 &7E &CC &18
&7E &7E &CC &7E
&00 &00 &78 &18
asc: 232 &E8 11101000
asc: 233 &E9 11101001
asc: 234 &EA 11101010
asc: 235 &EB 11101011
&0C &18 &99 &10
&0C &1C &5A &38
&0C &1E &24 &38
&0C &1B &C3 &38
&0C &18 &C3 &38
&3C &78 &24 &38
&7C &F8 &5A &7C
&38 &70 &99 &D6
asc: 236 &EC 11101100
asc: 237 &ED 11101101
asc: 238 &EE 11101110
asc: 239 &EF 11101111



Függelék 9. Kevésbé ismert szavak magyarázata

adatfile Olyan állomány valamely külső tárban, amely adatokat tartalmaz: számokat és szövegeket, de nem programot.

ASCII-kód Mikroszámítógépeken gyakorlatilag elfogadott kódrendszer, amely minden ábrázolható és vezérlő karakterhez egy kódszámot rendel, pl. a nagy A-betű ASCII kódszáma 65.

argumentum Függvény kiszámításához szükséges adat, BASIC-ben zárójelbe kell tenni: x=sin(y).

bináris A kettes számrendszert használó. A kettes számrendszer csak két számjegyet ismer: 0 és 1. Egy bináris számjeggyel így két állapotot lehet kifejezni, két számjeggyel már négyet, nyolccal 256-ot, tizenhattal pedig 65536-ot. A CPC a bináris számok jelölésére '&X' vagy '&x' előjelet használ.

bit Az operatív tár legkisebb (közvetlenül nem címezhető) egysége: állapota 0 (ki van törölve) vagy 1 (be van állítva). Egyes bitek állapotát megváltoztatni a logikai műveletekkel lehet, l. a 2.4 szakaszban.

blokk A kazettára egyszerre kiírt adatok mértékegysége: 2 Kbyte ◀▶ adatblokk : összetartozó adatok, nevezik még rekordnak is.

byte Az operatív tár legkisebb címezhető egysége. Nyolc bitet tartalmaz, így 256 állapot egyikét lehet vele jelölni, számokat 0-255-ig, számokat -127-től +127-ig, 256 darab különböző karakter egyikét, ugyanennyi parancsszót, stb.
Azt, hogy egy byte-ot hogyan kell értelmezni, az interpreter (és végső soron a CPU) dönti el.

CPU (Central Processing Unit) Központi egység, mikroprocesszor, a CPC esetében Z80-as típusú. Egy mikroprocesszor önmagában semmit sem ér: a megfelelő szoftver az, ami a számítógépet számítógéppé teszi.

editálás Szerkesztés, a program szövegének módosítása.

fejléc Kazettás és lemezes programfile-ok esetében a program előtti információ, amely tartalmazza a program nevét, típusát, hosszát, betöltési címét stb. Lemezen az adatfile-oknak nincs fejlécük.

felhasználói program Valamely hasznos célra készült program: táblázatszámító, szövegszerkesztő, adat-bázis és nyilvántartó stb., amelyet programozási ismeretek nélkül is lehet használni.

formattálás Lemezek használatbavétel előtti beavatása, lásd a 2.9.1.1 részben. ◆▶ Számok tetszés szerinti alakban történő kiíratása.

gépi kód A gép központi egysége (a CPU) számára közvetlenül értelmezhető program. A gépi kód rendkívül primitív lépésekből áll: ezt a számot tedd ide, azt add hozzá stb. Olyan magas szintű utasításai, mint a BASIC 'CLS' vagy 'PRINT' nincsenek.

hexadecimális A tizenhatos számrendszert használó. A hexadecimális számrendszer a 10-től 15-ig terjedő számok jelölésére betűket használ fel: A - F. Amíg a tízes számrendszer 9 után lép egyet balra: 10, a hexadecimális csak &F (=15) után: &10 (=16). A CPC-n a hexadecimális számokat '&' vagy '&H' jel vezeti be.

interfész Olyan csatlakoztató egység, amellyel a számítógép külső eszközökkel tarthat fenn kapcsolatot, pl. nyomtatóval, lemezmeghajtóval, mérőegységekkel stb.

interpreter Értelmező, olyan rendkívül bonyolult gépi kódú program, amely egy magas szintű nyelv, pl. a BASIC utasításait egyenként értelmezi és végrehajtja.

karakter Betű, jel vagy számjegy, amely a képernyő egy kockájában fér el. Mivel egy karakter kódolása 1 byte-on (= 8 biten) történik, összesen 256 karakter kezelhető egyszerre.

kifejezés Több tagból álló, felhasználás előtt kiszámítandó adat az egyenlőséggjel jobb oldalán, pl. x = **1+2/3*4-(3↑4)**

kiterjesztés A nyolckarakteres file-név után ponttal bevezetett három karakter, amellyel a file típusára utalhatunk, pl. CICAMICA.BAS (a BAS kiterjesztés BASIC programra utal). A kiterjesztés a file nevének részét alkotja és megkülönböztető szerepe van.

kontroll kód Vezérlő kód, amelynek PRINT-tel történő kiíratása egy cselekvést eredményez, pl. a képernyőt törli, hangjelzést ad stb.

LOGO Magas szintű programozási nyelv, amely "teknőcgrafikájáról" ismert. A barátságos grafika mellett a LOGO-nak több előnye is van a BASIC-kel szemben: strukturált programozásra nevel, lista-kezelő utasításai szubrutinok tucatjait helyettesíthetik stb.

LSB (least significant byte = kevésbé fontos byte) Két byte-on tárolt számok első byte-ja. A két byte-on tárolt szám értékének kiszámításához az LSB-hez hozzá kell adni az MSB értékének 256-szorosát (szám = LSB + MSB*256), l. MSB-nél.

mátrix Kétdimenziós tömb. Karakterek mátrixa az a tömb, amely a karakterek képét határozza meg soronként.

meghajtó Lemezmeghajtó, floppy lemezre adatokat kiíró-beolvasó eszköz. A lemezek írás-olvasás közben percenként 300 fordulatot tesznek.

monitor A televízióhoz hasonló elven dolgozó, de annál jóval élesebb és stabilabb képet adó megjelenítő eszköz. ◆▶ Olyan program, amellyel a számítógép tárát lehet kiolvasni, azt közvetlenül módosítani, stb. Gépi kódú programok belövéséhez hasznos segédeszköz.

MSB (most significant byte = fontosabb byte) Két byte-on tárolt szám második byte-ja. Mivel egy byte-tal csak nulla és 255 közötti számokat lehet ábrázolni, nagyobb értékek (egész számok, memóriacímek stb.) ábrázolása két byte-on történik: LSB + 256*MSB. Például, 1 : LSB=1, MSB=0; 255 : LSB=255, MSB=0; 256 : LSB=0, MSB=1; 257 : LSB=1, MSB=1; két byte-on ábrázolható maximális érték: 65536 : LSB=255, MSB=255. Egész típusú változók esetében csak a **pozitív** számok ábrázolása történik az ismertetett módon.

opcionális Nem kötelező, tetszés szerinti.

operációs rendszer Azon rutinok összessége, amelyek egy számítógép működéséhez, a felhasználóval való kapcsolathoz szükségesek, pl. egy karakter beolvasása a billentyűzetről, kiíratása a képernyőre stb.

operatív tár A számítógépben levő írható-olvasható tár (RAM), amellyel azonnal kapcsolatba lehet lépni, eltérően a különböző típusú háttértáraktól, amelyekre általában várni kell (floppy, magnószalag, stb.). Az operatív tárat **memóriának** is nevezik.

operandus Egy művelet adata, amelyen a művelet végrehajtódik, pl. x = 4 + 5 -ben a 4 és az 5 operandus, a '+' operátor.

operátor Műveletet végrehajtó jel vagy szó, pl. +, -, *, OR, /, AND stb.

paraméter Parancs vagy utasítás végrehajtásához szükséges adat. A paramétert BASIC-ben nem kell zárójelbe tenni, több paramétert vesszővel választunk el: 'PLOT 320,200' pontot tesz arra a pozícióra, amelynek koordinátái x=320, y=200.

parancs Olyan utasítás, amelyet az interpreter azonnal végrehajt. Programsorokban a parancsot utasításnak is nevezik.

pixel A képernyőn ábrázolható legkisebb pont. Minél több pixel fér el egy sorban vagy oszlopban, annál nagyobb a gép

grafikus felbontása.

puffer Adatok ideiglenes tárolására szolgáló tár. A CPC pl. a billentyűzetről beolvasott karakterek kódjait egy 20 byte hosszúságú pufferban tárolja. A külső tárral folytatott műveletek szintén puffert használnak.

RAM Írható-olvasható tár, a programok és adatok tárolására szolgáló operatív tár (l. ott). A gép kikapcsolása törli tartalmát.

relokálható Olyan gépi kódú program, amely nem tartalmaz önmagára mutató abszolút címeket, az operatív tár bármely részén működőképes.

rendszer-rutin Az operációs rendszer elemi rutinja, pl. egy pont megjelenítése, egy karakter kiírása stb.

ROM Csak olvasható tár. Tartalma sohasem törlődik, így a gép intelligenciáját (az operációs rendszerét, az interpretert stb.) helyezik el benne.

szintaxis Helyesírási szabály, amelyet az interpreter előír a parancsok beadásához. A nem megfelelően beadott parancsot nem érti meg: 'Syntax error'.

szóköz A hosszú szóközbillentyűvel a képernyőre vihető üres négyzet, amellyel pl. szavakat választhatunk el. ASCII kódja 32. Szóköz = " ". Semmi = "". A semminek nincs kódszáma!

szövegszerkesztő Olyan (al)program, amely megkönnyíti irományok (így pl. programszövegek) létrehozását és módosítását. Általában a teljes képernyőn szabad mozgást biztosító szövegszerkesztőt részesítik előnyben.

token Egy BASIC kulcsszónak megfelelő rövidebb (egy vagy két byte-os) kód. A tokenizálás lerövidíti a program hosszát és meggyorsítja a végrehajtást.

valós szám Olyan szám, amellyel (többé-kevésbé) bármilyen érték ábrázolható. A CPC korlátozásait l. a 1.2.3 pontban.

vektor Egydimenziós tömb. ◆▶ (Rendszer)rutin ugrócíme, amely általában RAM-ban van és így akár át is írható.

Függelék 10. A CPC BASIC kulcsszavainak áttekintése

Az alábbi oldalak a CPC BASIC összes utasítását, parancsát, függvényét és operátorát tartalmazzák ábécé-sorrendben. A jelekkel írt operátorok (+-/* stb.) nem kerültek be az összesítésbe. Minden kulcsszót a tokenje követi, valamint az utasítások és a parancsok esetében a BASIC interpreter megfelelő rutinjának belépési címét is megadjuk: az első cím a CPC 464, a második cím a CPC 664 és 6128-as gépekre érvényes.

A kulcsszavakat a használatukkal kapcsolatos szintaxis követi: a [szögletes zárójelekben] álló paraméterek opcionálisak. A kulcsszavak leírása tájékoztató jellegű, részletesen csak azokat ismertetjük, amelyek - helyhiány miatt - nem kerülhettek be a könyv megfelelő fejezetébe. Az AMSDOS és a CPC 6128 '|' kezdetű RSX parancsainak leírása a 2.9.5 pontban és a 3.5 szakaszban található.

ABS (numerikus kif.)
token: &00

Függvény: a numerikus kifejezés abszolút értékét számolja ki: negatív számból is pozitív értéket eredményez.

AFTER token: &80 belépési pont: &C971/&CA25
Önállóan nem használható.

AFTER időtartam [,időmérő] **GOSUB** sorszám

Utasítás: az opcionálisan megadott 'időmérőn' (0-3) letelt 'időtartam' után egy BASIC szubrutint hív meg. Az 'időtartam'-ot 0.02 másodperces egységekben kell megadni.

AND token: &FA

Logikai operátor: két kifejezés logikai 'és'-ét számítja ki az egész típusúra konvertált operandusok bitenkénti vizsgálatával: az eredő bit 1 ha mindkét bit 1 volt, egyébként 0.

ASC (szöveges kif.)
token: &01

Függvény: a szöveges kifejezés első karakterének ASCII kódját adja eredményül.

ATN (numerikus kif.)
token: &02

Függvény: a numerikus kifejezés arkusz tangensét számítja ki. Az eredményt fokban vagy radiánban adja az előzetesen végrehajtott DEG vagy RAD utasításoknak megfelelően.

AUTO [induló sorszám][,növekmény]
token: &81 belépési pont: &CODF/&COEA

Parancs: BASIC programíráshoz automatikusan állít elő sorszámokat. Amennyiben az 'induló sorszám'-ot nem adjuk meg, a sorszámok 10-től indulnak. Ha a 'növekmény'-t nem határozzuk meg, a sorok tizesével követik egymást.
Kiszállni az [ESC] billentyűvel lehet.
- ♦ CPC 464: a már meglevő programsort '*' jelzi
- ♦ CPC 664,6128: a már meglevő programsort kilistázza és 'EDIT' üzemmódba áll.

BIN$ (numerikus kif. [, hossz])
token: &71

Függvény: a 'numerikus kifejezés' bináris formáját tükröző, '0'-ákból és '1'-ekből álló karakterláncot állít elő, amely hossza nem lesz rövidebb az opcinálisan megadott 'hossz'-nál.

BORDER szín [,szín]
token: &82 belépési pont: &C221/&C24B

Utasítás: a képernyő keretének színét állítja be. A 'szín' intervalluma: 0 - 26. Amennyiben két színt adunk meg, azok váltakoznak a SPEED INK utasításban megadott időközönként.

BREAK önállóan nem használható, l. ON BREAK CONT, ON BREAK GOSUB, ON BREAK STOP.

CALL cím [, paraméterlista]
token: &83 belépési pont: &F1BA/&F261

Utasítás: gépi kódú szubrutint hív a BASIC-ből a megadott 'cím'-en. A gépi kódú program belépésnél az A-regiszterben megkapja a paraméterek számát, az IX-regiszter a paramétertáblázatban elhelyezett utolsó paraméterre mutat.

CAT token: &84 belépési pont: &D246/&D299

Utasítás: kilistázza a lemezen vagy kazettán levő file-ok nevét. Lemez esetén megadja a még szabad lemezkapacitást is, kazetta esetén a file-ok típusáról ad információt.

CHAIN file-név [, sorszám]
token: &85 belépési pont: &EA3C/&EB02

Utasítás: lemezről vagy kazettáról programot tölt az operatív tárba. A program a megadott 'sorszám'-tól indul, annak hiányában pedig az első sortól.

CHAIN MERGE file-név [, sorszám] [,DELETE sortól - sorig]

Utasítás: lemezről vagy kazettáról programot olvaszt egybe az operatív tárban levő programmal, előzőleg kitörli annak 'sortól' 'sorig' tartó részét. Az így összeolvasztott program a 'sorszám'-tól vagy annak hiányában az első sortól folytatódik.

CHR$ (numerikus kif.)
token: &03

Függvény: a 'numerikus kifejezés'-nek megfelelő sorszámú karaktert adja eredményül.

CINT (numerikus kif.)
token: &04

Függvény: a 'numerikus kifejezés'-t egész típusúra konvertálja.

CLEAR token: &86 belépési pont: &C132/&C12F

Utasítás: minden változót nulláz, a megnyitott file-okat elhagyja, a felhasználói függvényeket kitörli. A szögfüggvények számítási módját radiánra állítja át.

CLEAR INPUT ♦ csak CPC 664 és 6128

Utasítás: a billentyűzet-puffert kiüríti.

CLG [tinta]
token: &87 belépési pont: &C4B5/&C509

Utasítás: az ORIGIN utasítással definiált grafikus ablakot a 'tinta' színére, annak hiányában a grafikus papír színére törli.

CLOSEIN token: &88 belépési pont: &D298/&D2F0

Utasítás: az OPENIN utasítással az olvasásra megnyitott file-t lezárja.

CLOSEOUT token: &89 belépési pont: &D2A1/&D2F8

Utasítás: az OPENOUT utasítással az írásra megnyitott file-t lezárja.

CLS [# csatornaszám]

token: &8A belépési pont: &C25A/&C283

Utasítás: a 'csatornaszám'-mal megadott ablakot papírszínűre törli.

CONT token: &8B belépési pont: &CBC0/&CC96

Parancs: az [ESC] billentyűvel vagy a STOP utasítással megállított programot továbbindítja, ha a programszöveg időközben nem lett módosítva.

COPYCHR$ (# csatornaszám) ♦ csak CPC 664 és 6128

token: &7E

Függvény: a 'csatornaszám'-mal megadott ablak aktuális kurzorpozícióján található karaktert adja eredményül. Ha a karakter nem azonosítható, az eredmény nulla string: "".

Példa: locate #0,1,1: print #0,"A": locate #0,1,1:
a$=copychr$(#0) : print a$

COS (numerikus kif.)

token: &05

Függvény: kiszámítja a 'numerikus kifejezés' koszinuszát. Az eredményt szögfokban vagy radiánban adja, a DEG és RAD utasításoknak megfelelően.

CREAL (numerikus kif.)

token: &06

Függvény: a 'numerikus kifejezést' valós típusúvá konvertálja.

CURSOR 0,0 vagy 1,1 ♦ csak CPC 664 és 6128

token: &E1 belépési pont: &C363

Utasítás: **CURSOR 1,1** a kurzort INKEY$ után is megjeleníti, **CURSOR 0,0** az INPUT utáni kurzort eltünteti. PRINT utasítás előtt a kurzort ajánlatos eltüntetni.

DATA konstans-lista

token: &8C belépési pont: &E8EF/&E9A8

Utasítás: a programban felhasznált állandók elhelyezésére szolgál. Az adatokat a READ utasítással lehet egymás után kiolvasni. A RESTORE utasítással egy bizonyos adatcsoport jelölhető ki.

DEC$ (numerikus kif., formátum) ♦ csak CPC 664 és 6128

token: &72

Függvény: a 'numerikus kifejezés' decimális megfelelőjét adja eredményül egy szöveges változónak a 'formátum'-ban meghatározott alakban. A 'formátum'-ban felhasznált jelölés megfelel a PRINT USING formátumának.

DEF token: &8D belépési pont: &D117/&D174

Önállóan nem használható.

DEF FN függvény-név [(formális paraméterek)] = kifejezés

Utasítás: felhasználói függvények definiálására szolgál. A 'függvény-név' alakilag megegyezik a BASIC változók nevével, a definiált függvény aritmetikai és szöveges típusú lehet.

DEFINT betűlista, betűtől-betűig

token: &8E belépési pont: &D618/&D657

Utasítás: a 'betűlista' vagy/és a 'betűtől-betűig' tartomány betűivel kezdődő nevű változóit egész típusúnak deklarálja.

DEFREAL betűlista, betűtől-betűig
token: &8F belépési pont: &D61C/&D65B

Utasítás: a 'betűlista' és/vagy a 'betűtől-betűig' tartományba eső betűkkel kezdődő nevű változókat valós típusúnak deklarálja.

DEFSTR betűlista, betűtől-betűig
token: &90 belépési pont: &D614/&D653

Utasítás: a 'betűlista' és/vagy a 'betűtől-betűig' tartományba eső betűkkel kezdődő nevű változókat szöveges típusúnak deklarálja.

DEG token: &91 belépési pont: &D4E7/&D52C
Utasítás: végrehajtása után az összes szögfüggvény (SIN, COS, TAN, ATN) eredményét szögfokban kapjuk meg.

DELETE sorszámtól-sorszámig
token: &92 belépési pont: &E728/&E7F3

Parancs: a 'sorszámtól-sorszámig' tartományba eső sorszámú sorokat kitörli a BASIC programból. A tartomány állhat egy sorszámból is. A tartomány kezdetét vagy végét el lehet hagyni: ez esetben a tartomány a program elejétől az adott sorig, ill. az adott sortól a program végéig terjed.

DERR token: &49 ♦ csak CPC 664 és 6128

Függvény: a file-kezelés kapcsán utoljára előfordult hiba kódját adja eredményül, így az ERR függvényhez hasonlóan lehetővé teszi a kényelmes hibakezelést.

DERR értéke	A hibakód jelentése
0 vagy 22	[ESC] lenyomva
142	a csatorna nincs megnyitva
143	elérte a file fizikai végét
144	rossz parancs vagy file-név
145	a file már létezik
146	a file még nem létezik

147	a tartalomjegyzék betelt
148	a lemezen nincs több hely
149	megnyitott file mellett lemezcsere
150	a file csak olvasható
154	elérte a file végét (ASCII &1A-t)

DI token: &DB belépési pont: &C8E1/&C99A

Utasítás: a BASIC időzítőket kezelő megszakító rutinokat érvényen kívül helyezi. Az időzítők tovább számlálnak.

DIM indexes változók listája
token: &93 belépési pont: &D67D/&D6B9

Utasítás: az 'indexes változók listájában' megadott nevű, típusú és méretű tömböknek foglal helyet a memóriában.

DRAW x-koordináta, y-koordináta [, tinta][, mód]
token: &94 belépési pont: &C4C6/&C53C

Utasítás: a pillatnyi grafikus kurzorpozícióból kiindulva vonalat húz az 'x,y-koordinátákkal' meghatározott pontig. Ha a 'tintát' nem adjuk meg, a vonal az utoljára használt tintával lesz meghúzva.
♦ csak CPC 664 és 6128: az opcionális 'mód' az új vonal és az alatta levő színek egymásrahatását, az így kialakuló új színt határozza meg. A 'mód'-ok leírását l. a 2.3.7 pontban.

DRAWR x-távolság, y-távolság [, tinta][, mód]
token: &95 belépési pont: &C4CB/&C541

Utasítás: vonalat húz az aktuális grafikus kurzor pozíciótól 'x,y-távolságra' eső pontig. Ha a 'tintát' nem adjuk meg, a vonal színe az utoljára használt grafikus toll színével lesz azonos.
♦ csak CPC 664 és 6128: az opcionális 'mód' az új vonal és az alatta levő színek egymásrahatását, az így kialakuló új színt határozza meg. A 'mód'-ok leírását l. a 2.3.7 pontban.

EDIT sorszám
token: &96 belépési pont: &C052/&C046

Parancs: a kívánt 'sorszámú' BASIC programsort szerkesztéshez kilistázza, és szerkesztő üzemmódba áll át.

EI token: &DC belépési pont: &C8E7/&C9A0

Utasítás: a DI utasítással érvényen kívül helyezett BASIC időzítőket újra engedélyezi. EI automatikusan végrehajtódik akkor is, ha egy megszakítást-használó szubrutinban kiadott DI utasítás után a BASIC RETURN-hoz ér.

ELSE token: &97 belépési pont: &E8F3/&E9B2

Döntéshozó elágazások 'máskülönben' ágát vezeti be.
L. 'IF'-nél.

END token: &98 belépési pont: &CB65/&CC34

Utasítás: a program végrehajtását fejezi be és a BASIC-et parancsmódba téríti vissza.

ENT borítékszám [, borítékrészlet][, borítékrészlet] ...
token: &99 belépési pont: &D385/&D3D7

Utasítás: a 'borítékszámmal' (1-15) meghatározott hangfrekvenciagörbét állítja be a max. 5 'borítérészlet' által megadott alakúra. Minden 'borítékrészlet' további 3 paramétert tartalmaz. Részletes ismertetésüket l. a 2.6.3 pontban.

ENV borítékszám [, borítékrészlet][, borítékrészlet] ...
token: &9A belépési pont: &D34E/&D3A1

Utasítás: a 'borítékszámmal' (1-15) meghatározott hangerőgörbét állítja be a maximum 5 'borítékrészlet' által megadott alakúra. Minden 'borítékrészlet' további 3 paramétert tartalmaz. Részletes ismertetésüket l. a 2.6.3 pontban.

EOF token: &40

Függvény: az olvasásra megnyitott file végét ellenőrzi. Értéke -1 (igaz), ha a file-végét elértük, különben 0.

ERASE tömbváltozók listája
token: &9B belépési pont: &D9C0/&D9F4

Utasítás: a 'tömváltozók listájában' megadott tömböket törli.

ERL token: &E3

Függvény: BASIC programok hibakezelése során az utoljára megtalált hibás programsor sorszámát adja értékül.

ERR token: &41

Függvény: BASIC programok hibakezelése során az utoljára megtalált hiba kódját adja értékül.

ERROR numerikus kif.
token: &9C belépési pont: &CA8F/&CB54

Utasítás: BASIC programok hibakezeléséhez a 'numerikus kifejezés' által meghatározott sorszámú hibát generálja. A BASIC csak az 1-32 intervallumba eső sorszámú hibákat értelmezi (l. Függelék 6.), a 33-255 sorszámú hibakódok a felhasználó rendelkezésére állnak.

EVERY token: &9D belépési pont: &C979/&CA2D
Önmagában nem használható.

EVERY időegység [, időmérő] **GOSUB** sorszám

Utasítás: Minden 'időegység' letelte után meghívja a kívánt 'sorszámú' BASIC szubrutint. Minden 'időmérő'-höz egy-egy szubritin rendelhető.

EXP numerikus kif.
token: &07

Függvény: kiszámítja az exponenciális függvény értékét a 'numerikus kifejezés' által megadott helyen

FILL [tinta] ♦ csak CPC 664 és 6128
token: &DD belépési pont: &C515

Utasítás: a grafikus kurzor aktuális pozíciójából kiindulva a 'tinta' színére festi a grafikus toll vagy a 'tinta' színével határolt területet. Ha a grafikus kurzor eleve a toll vagy a 'tinta' színével megegyező színű pont fölött van, az utasítás nem hajtódik végre.

FIX numerikus kif.
token: &08

Függvény: levágja a 'numerikus kifejezés' tizedes ponttól jobbra eső részét és a bal oldalát adja eredményül.

FN token: &E4
Önmagában nem használható, l. DEF FN.

FOR ciklusváltozó = kezdőérték **TO** végérték [STEP lépésköz]
token: &9E belépési pont: &C529/&C5D7

Utasítás: a NEXT utasítással lezárt programrész ismételt végrehajtására szolgál. A végrehajtások számát a 'végérték' és a 'kezdőérték' különbsége osztva 'lépésköz'-zel adja meg. Ha a 'lépésköz' hiányzik, a BASIC 1-et tételez fel.

FRAME token: &E0 belépési pont: &BD19
♦ csak CPC 664 és 6128

Utasítás: szinkronizálja a képernyőn megjelenő grafikát a monitor katódsugarának a visszafutásával. Villogásmentes ábrázolást tesz lehetővé.

FRE (numerikus kif.) vagy (szöveges kif.)
token: &09

Függvény: tetszőleges 'numerikus kifejezés' paraméterrel a még szabad tárkapacitást adja értékül. Ugyanezt teszi 'szöveges kifejezés' paraméterrel, előzőleg azonban ún. szemétgyűjtést is végrehajt: a szöveges változókat összefüggő területen tömöríti.

GOSUB sorszám
token: &9F belépési pont: &C6ED/&C78F

Utasítás: a kívánt 'sorszámon' kezdődő BASIC szubrutinra ugrik, az első RETURN a GOSUB utáni utasításhoz téríti vissza a program futását.

GOTO sorszám
token: &A0 belépési pont: &C6E8/&C789

Utasítás: a program futását a kívánt 'sorszámon' folytatja.

GRAPHICS token: &DE belépési pont: &C5D9
Önmagában nem használható.

GRAPHICS PAPER tinta ♦ csak CPC 664 és 6128

Utasítás: a grafikus papír színét a 'tintával' kiválasztott színre állítja be. A grafikus papír színe csak CLG, TAG és MASK utasítások hatására vált át a 'tinta' színére.

GRAPHICS PEN tinta [, háttérmódus] ♦ csak CPC 664 és 6128

Utasítás: a grafikus toll színét a 'tintával' kiválasztott színre állítja be. A 'háttérmódus' értéke 0 és 1 lehet:
0 : normális háttér
1 : átlátszó háttér (szaggatott vonalaknál és TAG utasítás után a hátteret a grafikus papír színe írja felül).

HEX$ (numerikus kif. [, mezőszélesség])
token: &73

Függvény: értékül egy karakterláncot ad, amely a 'numerikus kifejezés' hexadecimális ábrázolását tartalmazza. A karakterlánc minimális hosszát a 'mezőszélesség' határozza meg.

HIMEM token: &42

Függvény: a BASIC által felhasznált legmagasabb byte címét adja vissza. Ez a cím a MEMORY utasítással csökkenthető.

IF logikai kif. **THEN** igaz ág [ELSE hamis ág]
token: &A1 belépési pont: &C6C7/&C76A

Utasítás: feltételes vezérlésátadást hoz létre. A 'logikai kifejezés' kiértékelése után, ha az igaz (nem 0), az 'igaz ág' utasításait hajtja végre, különben az ELSE-et követő 'hamis ág' utasításaira kerül a vezérlés. ELSE-ág hiányában a következő programsort tekinti 'hamis ág'-nak.

INK tinta, szín1 [, szín2]
token: &A2 belépési pont: &C22A/&C254

Utasítás: a 'tinta' sorszámú tintához a 'szín1' sorszámú

palettaszínt rendeli hozzá. Ha 'szín2'-t is megadunk, a két szín a SPEED INK utasításban meghatározott időközönként váltja egymást.

INKEY (numerikus kif.)
token: &0A

Függvény: a 'numerikus kifejezés' által meghatározott sorszámú billentyű aktuális állapotát adja eredményül. A billentyűsorszámok megtalálhatók a Függelék 3.-ban. A kapott érték értelmezését l. a 2.7.1 pontban.

INKEY$ token: &43

Függvény: a billentyűzet-pufferben található legelső karaktert adja eredményül. Ha a puffer üres, a függvény eredménye egy üres string: "".

INP (portszám)
token: &0B

Függvény: a 'portszám'-mal meghatározott portról beolvasott értéket adja eredményül. A CPC 16 (!) bites portcímzéssel dolgozik, így a 'portszám' &0000 - &FFFF között lehet.

INPUT [# csatornaszám][;]["szöveg" prompt] változólista
token: &A3 belépési pont: &DB2B/&DB48

Utasítás: a 'változólistá'-ban felsorolt változóknak értéket ad. Az értékadáshoz szükséges adatokat a billentyűzetről várja, ha a 'csatornaszám' ablakot határoz meg: 0 - 7. A 9-es 'csatornaszám' esetében az adatokat egy megnyitott file-ból olvassa be. Az első opcionális ';' (pontosvessző) az utasítás végrehajtása utáni soremelést akadályozza meg. A felhasználót informáló "szöveg" a 'csatornaszámmal' meghatározott ablakban jelenik meg. Ha a 'prompt' ';' (pontosvessző), az utasítás végrehajtásakor egy kérdőjel jelenik meg a képernyőn, ',' (vessző) 'prompt' letiltja a kérdőjelet.

INSTR ([kezdő pozíció,] szöveges kif. , keresett szöveg)
token: &74

Függvény: az opcionális 'kezdő pozíciótól' kezdve (annak hiányában az első karaktertől kiindulva) megkeresi a 'szöveges kifejezésben' a 'keresett szöveg' első előfordulását és annak pozícióját adja értékül. Nullát ad vissza, ha a 'keresett szöveget' nem találta meg.

INT (numerikus kif.)
token: &0C

Függvény: a 'numerikus kifejezés'-nél nem nagyobb egész

számot adja eredményül.

JOY (numerikus kif.)
token: &0D

Függvény: a 'numerikus kifejezés'-sel meghatározott (értéke 0 vagy 1 lehet) botkormány állapotát bitenként tükröző számot ad eredményül.

KEY funkciókódszám, szöveges kif.
token: &A4 belépési pont: &D439/&D489

Utasítás: a 'funkciókódszám'-mal meghatározott billentyűhöz a 'szöveges kifejezés'-t rendeli hozzá. A programozható funkcióbillentyűk szövegekkel való feltöltésére szolgál.

KEY DEF billentyű, ismétlés [,normál [,shift [,kontrol]]]

Utasítás: a billentyűzet átprogramozását teszi lehetővé. A 'billentyű' sorszámú billentyű által önmagában ('normál') [SHIFT]-tel ('shift') és a [CTRL] ('kontrol') billentyűvel együtt szolgáltatott ASCII karaktert határozza meg. Az 'ismétlés' értéke 1, ha a billentyű automatikus ismétlését be kívánjuk kapcsolni, és nulla, ha erre nincs szükség.

LEFT$ (szöveges kif., hossz)
token: &75

Függvény: a 'szöveges kifejezés' első 'hossz' számú karakteréből új karakterláncot állít elő.

LEN (szöveges kif.)
token: &0E

Függvény: a 'szöveges kifejezés' karakterekben számított hosszát adja eredményül.

LET változó = kifejezés
token: &A5 belépési pont: &D654/&D691

Utasítás: a 'változó'-hoz rendeli a 'kifejezés'-t. A CPC BASIC-jében a használatára nincs szükség.

LINE token: &A6 belépési pont: &DAF8/&DB18
Önmagában nem használható.

LINE INPUT [# csatornaszám][;]["szöveg" prompt] szöv. változó

Utasítás: működését és paramétereit l. az INPUT utasításnál. A LINE INPUT-tal csak egyetlen szöveges típusú változó olvasható be, amely azonban szabadon tartalmazhat idézőjeleket és vesszőket. A beadást az [ENTER] billentyűvel zárhatjuk, ha az a billentyűzetről történik, külső tároló esetében egy CHR$(13) terminálja a beolvasást.

LIST [kezdő sor - záró sor][,# csatornaszám]
token: &A7 belépési pont: &E0F7/&E12D

Parancs: A 'csatornaszám'-nak megfelelő eszközre kilistázza a BASIC program 'kezdő sor'-tól 'záró sor'-ig terjedő részét. Alapállapotban a listázás a 0-ás ablakba (vagyis a teljes képernyőre) történik. 8-as 'csatornaszám' a nyomtatóra, a 9-es 'csatornaszám' a külső tárolóra irányítja a listát. A 'kezdő sor' és/vagy a 'záró sor' elhagyható, ez esetben a program első sorától kezdődik, ill. az utolsó soráig tart a listázás.

LOAD file-név [, töltőcím]
token: &A8 belépési pont: &E9F6/&EABA

Utasítás: a 'file-név'-vel meghatározott programot a külső tárolóról az operatív memóriába tölti. Gépi kódú programok esetén megadható a 'töltőcím', így a kívánt címre tölti be a programot, függetlenül attól, hogy milyen címtől lett kimentve.

LOCATE [# csatornaszám,] x-koordináta, y-koordináta
token: &A9 belépési pont: &C2D2/&C302

Utasítás: a szöveges kurzort a 'csatornaszám'-mal meghatá-

rozott ablak x,y koordinátájú pozíciójára helyezi. A koordináták a bal felső sarokból indulnak (1,1).

LOG (numerikus kif.)
token: &0F

Függvény: a 'numerikus kifejezés' természetes logaritmusát számítja ki.

LOG10 (numerikus kif.)
token: &10

Függvény: a 'numerikus kifejezés' tízes alapú logaritmusát számítja ki.

LOWER$ (szöveges kif.)
token: &11

Függvény: a 'szöveges kifejezés' összes nagybetűjét (A-Z) kisbetűkre cseréli (a-z), minden más karaktert változatlanul hagy.

MASK [numerikus kif.] ♦ csak CPC 664 és 6128
token: &DF belépési pont: &C5C3

Utasítás: a 'numerikus kifejezés' (0-255) bitmintájának megfelelő szaggatott vonal rajzolásásra készül fel. Minden bit egy-egy grafikus pontot határoz meg: a beállított bit tollszínű pontot eredményez, a törölt bit pedig grafikus papír színű pontot hagy.

MAX (numerikus változólista)
token: &76

Függvény: a 'numerikus változólista' elemei közül a legnagyobb értékűt adja eredményül.

MEMORY (numerikus kif.)
token: &AA belépési pont: &F4EF/&F570

Utasítás: a 'numerikus kifejezés'-nek megfelelően beállítja a BASIC interpreter által használható legmagasabb memóriacímet.

MERGE file-név
token: &AB belépési pont: &EAA6/&EB59

Parancs: a tárban levő programhoz másolja a 'file-név'-vel meghatározott programot, azonos sorszámok esetén annak sorai felülírják a tárban levő program sorait.

MID$ (szöveges kif., kezdő pozíció [, hossz])
token: &AC

Függvény: a 'szöveges kifejezés'-ből a 'kezdő pozíció'-val meghatározott karaktertől kezdve új karakterláncot állít elő 'hossz' karakternyi hosszan. Ha a 'hossz'-at nem adjuk meg, az új karakterlánc a 'kezdő pozíciótól' a 'szöveges kifejezés' végéig tart.

MID$ (szöveges változó, kezdő pozíció [, hossz]) = új szöveg
token: &AC belépési pont: &F993/&FA07

Utasítás: a 'szöveges változó'-ban a 'kezdő pozíció'-tól kezdve elhelyezi az 'új szöveg'-et, felülírva annak régi tartalmát. Ha a 'hossz'-at is megadjuk, az 'új szöveg'-ből csak 'hossz' számú karakter kerül be a 'szöveges változó'-ba.

MIN (numerikus változólista)
token: &77

Függvény: a 'numerikus változólista' elemeiből a legkisebb értékűt adja eredményül.

MOD token: &FB

Kétváltozós aritmetikai operátor: az előtte álló argumentumot elosztja az utána állóval és a maradék egész részét adja vissza: 22.2 MOD 5 = 2.

MODE numerikus kif.

token: &AD belépési pont: &C24F/&C278

Utasítás: a képernyő üzemmódját állítja be a 'numerikus kifejezés' értéke szerint (0 - 2). A képernyőt törli és nulláz minden szöveges és grafikus ablakot.

MOVE x-koordináta, y-koordináta [, tinta][, mód]

token: &AE belépési pont: &C505/&C532

Utasítás: a grafikus kurzort a koordinátákkal meghatározott abszolút pozícióra állítja. A koordináták a bal alsó sarokból indulnak (0,0). A 'tinta' és 'mód' paraméterek csak csak a CPC 664 és 6128-on használhatók, értelmezésüket l. a DRAW utasításnál.

MOVER x-távolság, y-távolság [, tinta][, mód]

token: &AF belépési pont: &C50A/&C537

Utasítás: a grafikus kurzort helyezi át. Az új pozíciót a a régi pozícióból számítja ki az 'x-távolság' és az 'y-távolság' hozzáadásával. Mindkét relatív távolság negatív is lehet. A 'tinta' és a 'mód' paraméterek csak a CPC 664 és 6128-on használhatók, értelmezésüket l. a DRAW utasításnál.

NEXT [ciklusváltozó(k listája)]

token: &B0 belépési pont: &C5FB/&C6A5

Utasítás: a FOR utasítással megnyitott ciklus végét jelzi. A FOR-ra utaló 'ciklusváltozó' elhagyható, vagy egyszerre több hurok is lezárható a 'ciklusváltozók listájával'.

NEW token: &B1 belépési pont: &C12B/&C128

Utasítás: törli a memóriában levő programot és a változókat, nullákal tölti fel a BASIC programtárat. A képernyőt nem törli, minden képernyő-karakterisztika (MODE, PEN, INK, WINDOW stb.) érintetlenül marad.

NOT argumentum

token: &FE

Logikai függvény: az egész típusú 'argumentum' minden bitjét invertálja. A gyakorlatban ez azt jelenti, hogy a szám előjelét kifordítja és 1-et levon belőle. Használata feltételes vezérlő utasításokban és kapcsolókban gyakori:
Ha 'igaz' = -1, akkor NOT 'igaz' = 0, ugyanígy
ha 'hamis' = 0 , akkor NOT 'hamis'=-1.

ON token: &B2 belépési pont: &C7E3/&C885
Önmagában nem használható.

ON numerikus kif. **GOSUB** sorszámlista

Utasítás: kiválasztja a 'sorszámlistának' a 'numerikus kifejezés' által meghatározott elemét és arra a sorra ugrik, hogy szubrutint hajtson végre. Ha a sorszámlista nem tartalmaz elég elemet, vagy a 'numerikus kifejezés' értéke nulla, a következő soron folytatódik a program. Ugyancsak ott folytatódik a szubrutinból való visszatérés után is.

ON numerikus kif. **GOTO** sorszámlista

Utasítás: hatása azonos az ON GOSUB utasításával a szubrutinból való visszatérés kivételével.

ON BREAK token: &B3 belépési pont: &C8CB/&C979
Önmagában nem használható.

ON BREAK CONT ♦ csak CPC 664 és 6128
Utasítás: hatálytalanítja az [ESC] billentyűt. Az elindított BASIC programot többé nem lehet megállítani!

ON BREAK GOSUB sorszám

Utasítás: az [ESC] billentyű kétszeri lenyomására a program a 'sorszám'-mal meghatározott sorral kezdődő szubrutinra ugrik, ahelyett, hogy megállna.

ON BREAK STOP

Utasítás: hatálytalaanítja az ON BREAK CONT valamint az ON BREAK GOSUB utasítást. Az [ESC] billentyűvel a BASIC programot újra meg lehet állítani.

ON ERROR GOTO sorszám

token: &B4 belépési pont: &CBF8/&CCCD

Utasítás: a BASIC interpreter hiba észlelésekor nem állítja le hibaüzenettel a programot, hanem a 'sorszám' által meghatározott sorra ugrik. A hibakezelés utasításai: ERR, ERL, DERR, ERROR, RESUME. A BASIC interpreternek a hibakezelés az ON ERROR GOTO 0 (nulla) utasítással adható vissza.

ON SQ (hangcsatorna) GOSUB sorszám

token: &B5 belépési pont: &C940/&C9F8

Utasítás: a 'sorszám'-mal kezdődő szubrutinra ugrik, ha a 'hangcsatorna' az utasítás végrehajtásának pillanatában szabad (nem ad ki hangot). A 'hangcsatorna' értéke 1, 2 és 4 lehet az A, B és C csatornának megfelelően.

OPENIN file-név

token: &B6 belépési pont: &D25F/&D2B7

Utasítás: a 'file-név'-vel megadott ASCII file-t lemezről vagy kazettáról történő olvasáshoz megnyitja.

OPENOUT file-név

token: &B7 belépési pont: &D256/&D2AB

Utasítás: a 'file-név'-vel megadott ASCII file-t lemezen vagy kazettán létrehozza és/vagy íráshoz megnyitja.

OR token: &FC

Logikai operátor: az előtte és az utána álló egész típusú argumentumok logikai 'vagy'-át számítja ki bitenként. Az eredő bit 0, ha mindkét bit nulla, egyébként 1. Feltételes vezérlésátadásnál az 'igaz ág' hajtódik végre, ha az 'OR'-ral összekapcsolt feltételek közül akár csak egy is igaz.

ORIGIN x,y [, balszél, jobbszél, fent, lent]

token: &B8 belépési pont: &C48C/&C4E1

Utasítás: a grafikus koordinátarendszer kiinduló pontját az 'x,y' abszolút koordinátájú pontra állítja be. A négy további paraméterrel grafikus ablakot lehet meghatározni, az ablak széleit az eredeti abszolút koordinátákkal kell megadni.

OUT portszám, numerikus kif.

token: &B9 belépési pont: &F177/&F228

Utasítás: a 'portszám'-mal meghatározott portra kiadja a 'numerikus kifejezés' (0-255) értékét. Az I/O portok címzése a CPC-n 16-bites, vö. INP.

PAPER [# csatornaszám,] tinta

token: &BA belépési pont: &C20A/&C227

A 'csatornaszám'-mal meghatározott ablak papírjának színét 'tinta' színűre állítja. A 'tinta' színét az INK utasítás választja ki a színek palettájáról. Az egész ablakot a CLS utasítás festi át a papír színére.

PEEK (numerikus kif.)

token: &12

Függvény: a 'numerikus kifejezés'-sel (0 - 65535) meghatározott RAM-cím tartalmát adja eredményül.

PEN [# csatornaszám,] tinta [, háttérmódus]

token: &BB belépési pont: &C212/&C227

Utasítás: a 'csatornaszám'-mal megadott ablakban minden további kiíratás a 'tinta' sorszámú tintával fog történni.
♦ csak CPC 664 és 6128: a 'háttérmódus' lehet normális, felülíró (1) és átlátszó (0).

PI token: &44

Konstans: értéke 3.14159265, a kör kerülete és átmérője közötti arány közelitő értéke.

PLOT x-koordináta, y-koordináta [, tinta][, mód]
token &BC belépési pont: &C4D0/&C546

Utasítás: a képernyő x,y koordinátákkal meghatározott pontját a 'tinta' színére állitja, 'tinta' hiányában a pont az utoljára felhasznált grafikus toll színét kapja.
♦ csak CPC 664 és 6128: a 'mód' értelmezését l. a DRAW utasításnál.

PLOTR x-távolság, y-távolság [, tinta][, mód]
token: &BD belépési pont: &&C4D5/&C54B

Utasítás: a grafikus kurzor pozíciójától vízszintesen 'x-távolság'-ra, függőlegesen 'y-távolság'-ra levő pontot a 'tinta' színére állitja. A 'tinta' hiányában a pontot az utoljára felhasznált grafikus toll színére állitja be.
♦ csak CPC 664 és 6128: a 'mód' értelmezését lásd a DRAW utasításnál.

POKE numerikus kif., egész típusú kif.
token: &BE belépési pont: &F15F/&F214

Utasítás: a 'numerikus kifejezés' által meghatározott RAM-címre beírja az 'egész típusú kifejezés' (0 - 255) értékét.

POS (# csatornaszám)
token: &78

Függvény: a szöveges kurzor aktuális x-pozicióját adja értékül. Az x-pozíciót a 'csatornaszám'-mal kötelezően meg-

határozott ablak bal szélétől számitja ki.
POS (#8) a nyomtatófej pozíciójáról tájékoztat. POS (#9) a megnyitott file-ba küldött karakterek számát adja meg.

PRINT [# csatornaszám,][adatok listája][elválasztójel]
token: &BF belépési pont: &F1FD/&F2A9

Utasítás: a 'csatornaszám'-mal megadott ablakba vagy eszközre kiírja az 'adatok listája' elemeit. Az adatokat vagy ',' (vessző) választja el egymástól: minden adat új zónába kerül, l. ZONE), vagy ';' (pontosvessző): az adatokat szorosan egymás mellé írja ki. Amennyiben egy adat nem fér el teljesen a zónájában vagy az aktuális sorban, számára új zonát vagy sort kezd. A listát záró 'elválasztójel' szintén ',' és ';' lehet, a szöveges kurzor helyét határozza meg a fent leírt módon. L. még SPC, TAB, USING, ZONE.

RAD token: &C1 belépési pont: &D4EB/&D530

Utasítás: végrehajtása után a szögfüggvényeket radiánban számolja ki. A gép bekapcsoláskor, valamint NEW, CLEAR, RUN stb. parancsok kiadása után is egy RAD utasítást hajt végre.

RANDOMIZE [numerikus kif.]
token: &C2 belépési pont: &D559/&D59C

Utasítás: újraindítja a gép véletlenszám-generátorát a 'numerikus kefejezés'-nek megfelelően. Ha megismételhetelen véletlenszám sorozatra van szükségünk, 'numerikus kifejezés'-ként adjuk meg a TIME belső változót: RANDOMIZE TIME.

READ változók listája
token: &C3 belépési pont: &DCEB/&DCDF

Utasítás: a 'változók listája' elemeinek adja értékül azon DATA sor adatait, amelyre a belső adatmutató éppen rámutat. A belső adatmutató a RESTORE utasítással bármely DATA sorra ráállítható.

RELEASE hangcsatorna
token: &C4 belépési pont: &D31E/&D373

Utasítás: engedélyezi a azokat a 'hangcsatornákat', amelyeket a SOUND utasítással 'HOLD' állapotba állitottunk. A 'hangcsatorna' értéke 1-7 lehet.

REM megjegyzés a sor végéig
token: &C5 belépési pont: &E8F3/&E9AC

Utasítás: tetszőleges megjegyzésnek biztosit helyet az adott sor végéig. A :REM utasítás helyettesithető a felső vessző (') karakterrel, tokenje: &C0.

REMAIN (időmérő)
token: &13

Függvény: eredményül az 'időmérő' sorszámú belső időzitőn megmaradó időt adja, és nullázza valamint letiltja az időzitőt. Az AFTER, EVERY, EI, DE utasításokkal kapcsolatban használatos.

RENUM [új sorszám][,[régi sorszám][, növekmény]]
token: &C6 belépési pont: E7DF/&E8A3

Parancs: átszámozza az operativ tárba lévő Basic programot vagy annak egy részét. Az 'új sorszám' az átsorszámozandó rész első sorának új sorszáma lesz, a 'régi sorszám' az átsorszámozandó rész első sorának pillanatnyi sorszáma. A 'növekmény' az új sorszámok közötti különbséget határozza meg. A paraméterek elhagyása esetén az egész programot átsorszámozza tizes növekménnyel. A GOTO, GOSUB, RESUME stb. után álló sorszámokat minden esetben figyelembe veszi.

RESTORE [sorszám]
token: &C7 belépési pont: &DCD9/&DCCD

Utasítás: a belső adatmutatót a 'sorszám'-mal megadott DATA sorra, annak hiányában az első DATA sorra állitja.

RESUME [sorszám] vagy NEXT
token: &C8 belépési pont: &CC03/&CCD8

Utasítás: a hibakezelő rutinból a 'sorszám'-mal megadott sorral folytatja a program végrehajtását. Ha nem adunk meg 'sorszám'-ot, a program annak a sornak az elejére ugrik, amelyikben a hiba előfordult. A 'sorszám' helyett megadott NEXT a hiba után következő sorra adja a vezérlést.

RETURN token: &C9 belépési pont: &C70F/&C7B3

Utasítás: BASIC szubrutin végét jelzi. A vezérlést az őt hívó GOSUB-ot követő utasításnak adja át.

RIGHT$ (szöveges kif., hossz)
token: &79

Függvény: új karakterláncot állit elő a 'szöveges kifejezés' utolsó 'hossz' karakteréből.

RND [(numerikus kif.)]
token: &45

Függvény: a soron következő véletlenszámot adja eredményül, ha a 'numerikus kifejezés' nincs megadva, vagy értéke pozitiv. Nulla érték esetén az előző véletlenszámot adja meg. Ha a 'numerikus kifejezés' értéke negativ, újra kezdi a véletlenszám-sorozatot és annak első számával tér vissza.

ROUND (numerikus kif. [, formátum])
token: &7A

Függvény: a 'numerikus kifejezés' szabályosan kerekitett értékét számitja ki. A pozitiv 'formátum' a tizedes pont utáni számjegyek maximális számát határozza meg. Negativ 'formátum'-nál egész számot eredményez, amely a 'formátum'-mal meghatározott számú nullával végződik: kerekités tizesekre, százasokra stb.

RUN [sorszám] vagy file-név
token: &CA belépési pont: &E9BD/&EA7D

Parancs vagy utasítás: nullázza az összes változót és elindítja a tárban levő programot az első sortól vagy a 'sorszám'-mal megadott sortól. 'File-név' megadásánál külső tárból (lemez vagy kazetta) betölti és elindítja az adott nevű programot.

SAVE file-név [, file-típus][, bináris paraméterek]
token: &CB belépési pont: &EC09/&ECE1

Parancs és utasítás: a külső tárba (lemezre vagy kazettára) menti az operatív tárban levő BASIC vagy gépi kódú programot. A file-típusok és esetleges paraméterek leírása a 2.10.1 pontban található.

SGN (numerikus kif.)
token: &14

Függvény: a 'numerikus kifejezés' előjelét állapítja meg. Eredménye -1 ha a szám negatív, +1 ha a szám pozitív, és 0, ha a szám értéke nulla.

SIN (numerikus kif.)
token: &15

Függvény: a 'numerikus kifejezés' szinuszát számítja ki. Az eredményt radiánban vagy fokban adja meg, vö. RAD, DEG.

SOUND hangcsatorna, hangperiódus [, időtartam [, hangerő [, hangerőboríték [, hangszínboríték [, zajperiódus]]]]]
token: &CC belépési pont: &D2C0/&D316

Utasítás: hangeffektusok és (zenei) hanghatások előállítására szolgál. Korlátozott használatra elegendő az első két (esetleg négy) paramétert megadni.

SPACE$ (numerikus kif.)
token: &16

Függvény: a 'numerikus kifejezés' által meghatározott számú (0 - 255) szóközből álló karakterláncot eredményezi.

SPC (numerikus kif.)
token: &E5

Függvény: a PRINT utasításban a kiírandó adatok közötti elválasztójelként használva a 'numerikus kifejezés' által meghatározott számú (0 - 255) szóközt eredményez.

SPEED token: &CD belépési pont: &D494/&D4DE
Önmagában nem használható.

SPEED INK időtartam1, időtartam2

Utasítás: az INK utasításban megadott színek váltakozásának időtartamát állítja be 0.02 másodperces időegységekben.

SPEED KEY várakozás, ismétlési periódus

Utasítás: az automatikus ismétlésre állított billentyűk ismétlési sebességét határozza meg 0.02 másodperces időegységekben. A 'várakozás' időtartama az első ismétlésig tartó időt állítja be, az 'ismétlési periódus' pedig a további ismétlések közötti intervallumot szabályozza.

SPEED WRITE 0 vagy 1

Utasítás: a kazettára történő írás sebességét határozza meg. '0' paraméter az írás sebességét 1000 baudra állítja be (alapállapot). '1' paraméter esetében a sebesség 2000 baud lesz. Ezt a sebességet csak megbízható magnetofon és szalag használatakor érdemes választani. A beolvasás sebességét a CPC automatikusan állítja be.

SQ (hangcsatorna)
token: &17

Függvény: a 'hangcsatorna' pillanatnyi állapotáról tájékoztat. A 'hangcsatorna' értéke 1, 2 és 4 lehet. A függvény által visszaadott egész szám (0 - 255) minden bitje egyedi információt hordoz.

SQR (numerikus kif.)
token: &18

Függvény: a 'numerikus kifejezés' négyzetgyökét számitja ki.

STEP numerikus kif.
token: &E6

A FOR-NEXT ciklus lépésközét állitja be. L. a FOR utasitásnál.

STOP token: &CE belépési pont: &CB5A/&CC29

Utasítás: megszakitja a BASIC program végrehajtását és parancsmódba tér vissza. A program CONT paranccsal a STOP utáni utasitással kezdve folytatható.

STR$ (numerikus kif.)
token: &19

Függvény: a 'numerikus kifejezést' karakterlánccá alakitja át.

STRING$ (hossz, karakter)
token: &7B

Függvény: a 'hossz'-szal meghatározott hosszúságú (0 - 255) karakterláncot állit elő az ASCII kóddal vagy karakterként megadott 'karakter'-ből.

SWAP token: &E7
Önmagában nem használható. L. WINDOW SWAP.

SYMBOL karaktersorszám, bitmintalista
token: &CF belépési pont: &F69D/&F784

Utasitás: a 'karaktersorszám'-mal meghatározott ASCII kódszámú karakter áttervezését teszi lehetővé. A 'bitmintalista' 8 elemet tartalmaz, értékük: 0 - 255. Minden elem a karakter egy-egy sorának mintáját adja.

SYMBOL AFTER karaktersorszám

Utasitás: a 'karaktersorszám'-mal megadott (0 - 255) sorszámú karaktertől kezdve a 255-ös karakterrel bezárólag az összes karakter bitmintáját RAM-ba másolja, igy lehetőség nyilik azok áttervezésére.

TAB (numerikus kif.)
token: &EA

Függvény: a PRINT utasitásban a kiirandó adatok közötti elválasztójelként használva a következő adat kiirását a 'numerikus kifejezés'-sel megadott oszlopban kezdi.

TAG [# csatornaszám]
token: &DO belépési pont: &C319/&C346

Utasitás: a 'csatornaszám'-mal megadott ablakban a PRINT utasitással kiiratásra kerülő szövegek a továbbiakban az aktuális grafikus kurzorpozición jelennek meg a grafikus toll színében.

TAGOFF [# csatornaszám]
token: &D1 belépési pont: &C320/&C34D

Utasitás: a 'csatornaszám'-mal megadott ablakban hatástalanitja a TAG utasitást. Minden további kiiratás a szöveges kurzor aktuális pozicióján jelenik meg.

TAN (numerikus kif.)
token: &1A

Függvény: kiszámítja a 'numerikus kifejezés' tangensét. Az eredményt radiánban vagy fokban adja, vö. RAD, DEG.

TEST (x-koordináta, y-koordináta)
token: &7C

Függvény: a koordináták által meghatározott abszolút pozícióra helyezi a grafikus kurzort és eredményül adja a kurzor alatti pont tintájának sorszámát.

TESTR (x-távolság, y-távolság)
token: &7D

Függvény: a grafikus kurzort aktuális pozíciójától 'x' és 'y' távolságra helyezi, és eredményül adja a kurzor alatti pont tintájának sorszámát.

TIME token: &46

Függvény: a gép bekapcsolása óta eltelt időt adja vissza 1/300 másodperces egységekben.

TO token: &EC

FOR-NEXT ciklusban a ciklusváltozó végértékét jelzi. L. a FOR utasítás leírásánál.

TROFF token: &D2 belépési pont: &DDE6/&DECA

Utasítás: a BASIC program nyomonkövetését kikapcsolja.

TRON token: &D3 belépési pont: &DDE2/&DEC6

Utasítás: a BASIC program nyomonkövetését teszi lehetővé, minden végrehajtásra kerülő sor sorszámát szögletes zárójelekben jeleníti meg.

UNT (egész típusú kif.)
token: &1B

Függvény: az 'egész típusú kifejezés' előjel nélküli értékének kettes komplemensét adja eredményül.

UPPER$ (szöveges kif.)
token: &1C

Függvény: olyan karakterláncot eredményez, amelyben a 'szöveges kifejezés' összes kisbetűje (a - z) helyett nagybetű (A - Z) áll.

USING szöveges konstans vagy változó
token: &FB

A PRINT utasítással kiíratandó adatok formatált megjelenítését segíti elő a 'szöveges konstans'-sal vagy 'szöveges változó'-val megadott formátum szerint. A formátum részletes ismertetése a 2.1.3 pontban található.

VAL (szöveges kif.)
token: &1D

Függvény: a 'szöveges kifejezés' által képviselt szám numerikus értékét adja meg.

VPOS (# csatornaszám)
token: &7F

Függvény: a kötelezően feltüntetett számú ablak szöveges kurzorának függőleges pozícióját adja meg az ablak felső keretéhez viszonyítva.

WAIT port-szám, maszk [, inverzió]
token: &D4 belépési pont: &F17D/&F22E

Utasítás: a program a megadott porton beolvas egy értéket és a következő műveleteket végzi el vele: ['inverzió' megadása esetén a beolvasott értéket XOR-olja az 'inverzió-

val majd] a kapott értéket AND-eli a 'maszk'-kal.
A beolvasást addig ismétli, míg a kapott eredmény nem nulla. Az utasítás használata végtelen hurokhoz vezethet!

WEND token: &D5 belépési pont: &C776/&C81D

Utasítás: olyan programrész végét jelzi, amelyet az előző WHILE utasításban megadott feltétel meglétéig ismételten végre kell hajtani.

WHILE logikai kif.
token: &D6 belépési pont: &C747/&C7EA

Utasítás: olyan programrész kezdetét jelöli, amelyet ismételten végre kell hajtani, amíg a 'logikai kifejezés'-ben meghatározott feltétel igaz.

WIDTH numerikus kif.
token: &D7 belépési pont: &C3E3/&C42D

Utasítás: a géphez kapcsolt nyomtatóra a 'numerikus kifejezés'-sel meghatározott számú karakter kiküldése után egy kocsivissza-soremelés kombinációt küld ki. Ezt megakadályozhatjuk a WIDTH 255 utasítással, amely felfüggeszti a nyomtatás szélességének ellenőrzését.

WINDOW [# csatornaszám,] balszél, jobbszél, fent, lent
token: &D8 belépési pont: &C2E1/&C311

Utasítás: a képernyőn ablakot definiál, amelyre a 'csatornaszám'-mal (0 - 7) lehet hivatkozni a PRINT, PEN, PAPER, LOCATE stb. utasításokban. Az ablak vízszintes paramétereit az aktuális MODE határozza meg.

WINDOW SWAP csatornaszám, csatornaszám

Utasítás: a két 'csatornaszám'-mal meghatározott ablak paramétereit (hely, papírszín, tollszín stb.) kicseréli.

WRITE [# csatornaszám,] adatlista
token: &D9 belépési pont: &F1F6/&F2A6

Utasítás: a PRINT utasításhoz hasonlóan a 'csatornaszám'-mal megadott ablakba vagy eszközre kiírja az 'adatlista' elemeit. A PRINT-től eltérően a kiírt elemeket vesszővel választja el egymástól, a szöveges adatokat pedig idézőjelek közé zárja.

XOR token: &FD

Logikai operátor: 'kizáró vagy' műveletet végez az előtte és utána álló kifejezésekkel. A műveletet bitenként végzi az egész típusúvá konvertált operandusokon. Az eredő bit '1' ha a bitek eltérőek voltak, egyébként '0'.

XPOS token: &47

Függvény: a grafikus kurzor aktuális vízszintes (x) koordinátáját adja eredményül.

YPOS token: &48

Függvény: a grafikus kurzor aktuális függőleges (y) koordinátáját adja eredményül.

ZONE numerikus kif.
token: &DA belépési pont: &F47B/&F50D

Utasítás: a PRINT utasítás által használt kiírási zónák szélességét állítja be a 'numerikus kifejezés'-nek megfelelően. Alapállásban a gép 13 karakter széles zónákkal dolgozik.

Kiadja a Műszaki Könyvkiadó
Felelős kiadó: Szűcs Péter igazgató
Felelős szerkesztő: Votisky Zsuzsa

VTV 87/003
Felelős vezető: Erős László

Műszaki vezető: Kőrizs Károly
A borítót és a kötést tervezte: Kováts Tibor
A könyv ábrái számítógéppel készültek
A könyv formátuma: A5
Ívterjedelme: 24 (A5)
Papír minősége: 80 g ofszet
Azonossági szám: 61 491
MŰ: 3982–i–8789
Kézirat lezárva: 1986. december
Készült az MSZ 5601 és 5602 szerint

198 Ft

Ahány személyi számítógép, annyi BASIC nyelvjárás – ferdíthetnénk a közmondást. A sokféle BASIC-ből is kiemelkedik a Schneider/Amstrad CPC gépcsalád nyelvi gazdagsága és igényessége. Ez is lehetett annak az oka, hogy e gép nyerte el 1985-ben az ÉV GÉPE megtisztelő elnevezést a maga kategóriájában.
A hazai érdeklődők és géptulajdonosok, egyszóval az Olvasók e könyvből nemcsak a kitűnő gép tulajdonságaival, hanem egy szokatlanul közvetlen hangú és mégsem csak kezdőknek szóló BASIC leírással ismerkedhetnek meg.

BUDAPEST
MŰSZAKI KÖNYVKIADÓ