Plenge · Szczepanowsky

SIMON'S BASIC

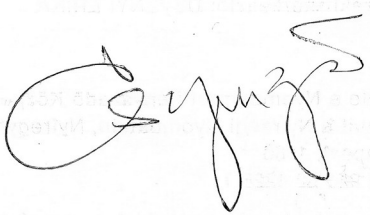
Gyakorlatok

DATA BECKER – NOVOTRADE

Plenge · Szczepanowsky

SIMON'S BASIC

Gyakorlatok



DATA BECKER – NOVOTRADE

A mű eredeti címe: Simon's BASIC (1984)

Fordította: az APEX Szervező-Szolgáltató GMK munkacsoportja (Gyöngyös)

Felelős szerkesztő: TARR KÁLMÁNNÉ

A kiadásért felel: RÉNYI GÁBOR, a NOVOTRADE RT. igazgatója
Kiadványmenedzser: BÉKÉS TAMÁS
Műszaki szerkesztő: DÉVÉNYI ERIKA

Szedte a Nyomdaipari Fényszedő Központ, Budapest
Készült a Nyírségi Nyomdában, Nyíregyházán (13 A/5 ív)
Budapest, 1986
ISBN 963 02 4228 1

FONTOS MEGJEGYZÉS

A könyvben szereplő kapcsolások, eljárások és programok közlése a bejelentett szabadalmak figyelembevétele nélkül kerülnek ismertetésre. Kizárólag amatőr és tanulási célokat szolgálnak, egyéb célokra történő felhasználásuk tiltos!
A szerzők a közölt műszaki adatokat, kapcsolásokat és programokat a legnagyobb gondossággal állították össze. Ennek ellenére előfordulhatnak hibák, amelyekért a DATA BECKER GmbH sem garanciát, sem jogi felelősséget nem vállal.
A könyvvel kapcsolatos bármilyen észrevételt a szerzők köszönettel vesznek.

TARTALOMJEGYZÉK

ELŐSZÓ

A SIMON's BASIC a COMMODORE 64-es nagy népszerűségnek örvendő programbővítése. A programozás legkülönbözőbb területein használható, a száznál is több utasítás a COMMODORE BASIC-ot hatékony programnyelvvé teszi. Ha a SIMON's BASIC által nyújtott programozási lehetőségeket teljes egészében ki akarja használni, részletesen meg kell ismernie az egyes utasításokat. Tapasztalataink szerint ez elég nehéz a programbővítéshez kapott könyv alapján, mivel az nem elég részletes, és a belső összefüggések magyarázatával nem foglalkozik.
Ezért a COMMODORE 64-es felhasználói számos problémával fordultak hozzánk, amely arra ösztönzött bennünket, hogy kiadjunk egy minden igényt kielégítő SIMON's BASIC gyakorlókönyvet. A könyv két szerzője bizonyára nem ismeretlen Önök előtt. Axel Plenge, aki a sikeres „ SUPERGRAPHIK " szerzője, e könyvben – másként nem is lehetne – a SIMON's BASIC grafikai és zenei lehetőségeit ismerteti. A további fejezeteket teljes egészében Norbert Szczepanowski készítette, aki már a "FLOPPY-BUCH"-ban is bebizonyította a COMMODORE-gépek sokoldalúságát.
Ön most e két szerző hónapokig tartó, aprólékos munkájának eredményét tartja a kezében. A könyv ismerteti a SIMON's BASIC utasításait, kiegészítve az alkalmazási lehetőségeket bemutató számtalan példával. Minden egyes utasítással külön foglalkozik, így a könyvet lexikonszerűen használhatja.
Jó szórakozást kíván a könyv tanulmányozásához és a SIMON's BASIC alkalmazásához

Dr. Achim Becker

1. FEJEZET
BEVEZETÉS

A SIMON's BASIC-ről először mint kényelmes BASIC-bővítési lehetőségről hallottunk, majd mikor közelebbről is megismerkedtünk vele, rájöttünk, hogy a C–64-eshez ez az igazi "BŐVÍTÉS".
A Commodore 64-es sok rendkívüli képességéről győzte meg Önt, ezért végül vásárolt is egyet. Hazatérve azonban, mikor a felhasználói kézikönyvet először átolvasta, rá kellett jönnie, hogy a gép által kínált lehetőségek az eredeti BASIC V2-vel (a C–64-es standard BASIC-je) csak nehezen valósíthatók meg. A grafika, a zene és a sprite-ok vezérlése számtalan PEEK és POKE utasítást igényel, amelyek végül áttekinthetetlenné és hosszúvá teszik a programot. Ezenkívül lehetetlen a magas szintű programnyelvektől elvárható szisztematikus és kényelmes programozás megvalósítása. Ciklusok szervezéséhez például mindössze egyetlen utasításpár (FOR...NEXT) áll rendelkezésünkre és nem jobb a helyzet a feltételes operációkra (IF), vagy az ugrási utasításokra (GOTO/GOSUB) vonatkozóan sem.
A program kiíratása egyedül a LIST paranccsal oldható meg, amely nem nyújt túl sok lehetőséget.
Összefoglalva: aki számítógépén nem csak előregyártott programokat kíván futtatni, hanem komoly feladatokat szeretne megoldani, annak mindenképpen szüksége van erre a BASIC bővítésre.
A SIMON's BASIC az univerzális programozóknak készült, és így szinte minden olyan részfunkcióhoz kiegészítő utasításokkal szolgál, amelyek a Commodore 64-est az 1983-as év számítógépévé tették.
Azoknak, akik a programozásnak csak egy részterületével foglalkoznak (pl. zene, grafika stb.) egyéb, speciális bővítésekhez, vagy más programformákhoz kell folyamodniuk. Ettől eltekintve mind a kezdőknek, mind pedig a haladóknak érdemes elsajátítaniuk a SIMON's BASIC-et.
Az olyan programozási segédeszközök, mint a funkcióbillentyűk használata, különböző listázási lehetőségek, vagy a BASIC-sorok átszámozásának lehetősége, mind a kényelmes és gyors programozást segítik. Ahol egyébként hosszadalmas utasításokat kellene billentyűznie, a SIMON's BASIC-kel elég egyetlen gombnyomás. Számos, magas szintű programnyelvből származó szerkezeti utasítás (pl. REPEAT vagy a szimbolikus címzés) könnyíti meg nagy teljesítményű programok készítését.
A Commodore 64 alkalmazási lehetőségeinek egész tárházát nyitják meg Ön előtt a képernyőre vonatkozó utasítások: a formálás, grafika készítés, sprite-ok, a karakterkészlet módosítása, többszínű és nagyfelbontású grafika,... stb. Több grafikai utasítás segíti a bonyolult grafikák képernyőre vitelét és kinyomtatását.
Természetesen nincs program hiba vagy hiányosság nélkül. Ne várjuk ezt el tehát a SIMON's BASIC-től se. Előfordul, hogy valamelyik funkció nem hajtható végre a kívánt módon, vagy alkalmazása a program szétesését eredményezi. Igyekszünk felhívni a figyelmét ezekre a buktatókra és lehetőségeink szerint segíteni fogjuk Önt abban, hogy jól sikerült programokat írjon. Előrebocsátjuk, hogy bizonyos területeken, pl. a grafika tárolása vagy lemezhasználat, még több utasításra volna szükség, mert ezekhez a SIMON's BASIC is csak kevés támogatást nyújt.
Könyvünkben az egyes utasításokat lehetőleg érthetően és főként rendszerezve próbáljuk ismertetni. Az eredeti SIMON's BASIC-kézikönyvet, sok hiányossága és rendszertelen felépítése miatt nem is vettük figyelembe. Az utasításokat teljesen más elven, egymásra építve tárgyaljuk.
Először a programozási segédeszközökkel és hibakezelésekkel foglalkozunk, majd a strukturált programozás utasításain keresztül eljutunk a be- és kivitelvezérlés terjedelmes fejezetéhez, amelyhez a grafika, a sprite és a zene programozása kapcsolódik.

Minden fejezet végén teszt-lap található, amellyel az ismeretek elsajátítása ellenőrizhető. A megoldásokat a könyv végén közöljük.
Reméljük, hogy tanulmányaihoz egy átfogó és információkban gazdag segédeszközt tudunk a kezébe adni ezzel a könyvvel. Külön felhívjuk figyelmét a 9.3 fejezetben szereplő három olyan utasításra, amelyek az eredeti kézikönyvből kimaradtak és egyébként sem ismertek.

Megjegyzések a magyar változathoz:

– Mindazokat a könyvben szereplő programokat, melyeknek listája a következő három sorral kezdődik

```
1 REM * * * * * * *
2 REM * CIM *
3 REM * * * * * * *
```

mágneslemezre rögzítettük, általában a 2. sorban feltüntetett címen. Ahol ettől eltértünk, külön jelezni fogjuk.
– A programok betöltésének általános szabálya, hogy a program címe szóközt (space) nem tartalmazhat. A forma egyébként az általánosan ismert:

```
LOAD"CIM",8
```

parancs.

2. FEJEZET
Programozási segédeszközök

2.1 A funkcióbillentyűk használata

A legtöbb személyi számítógépen megtalálhatók az ún. funkcióbillentyűk, amelyek a gyakran használatos utasítások vitelét leegyszerűsítik. Két változatuk ismert:

a) Az egyénileg használható funkcióbillentyűk, amelyekhez a felhasználó tetszőleges jelsorozatot rendelhet.
b) Az ún. ASCII funkcióbillentyűk, amelyekhez a többi billentyűhöz hasonlóan ASCII érték tartozik. Működtetésük a programban lekérdezhető és a program futása ennek függvényében folytatható.

A C–64-es standard változata csak az utóbbival rendelkezik. A nyolc billentyű ASCII-kódja a következő:

F1 – 133 F3 – 134 F5 – 135 F7 – 136
F2 – 137 F4 – 138 F6 – 139 F8 – 140

Lekérdezésük a programon belül például így történik:

P. 1

```
1 REM **********************
2 REM * FUNKCIOBILLENTYU *
3 REM **********************
100 GET X$:IF X$="" THEN 100
105 IF ASC(X$)<133 OR ASC(X$)>140 THEN 100
110 PRINT" A FUNKCIOBILLENTYU ASCII ERTEKE:";ASC(X$)
120 GOTO 100
```

Ez a program csak valamely funkcióbillentyű lenyomását fogadja el, és kiírja annak ASCII-kódját. A program a RUN/STOP billentyűvel állítható meg.
A funkcióbillentyűk tulajdonképpeni rendeltetése azonban nem ez. A SIMON's BASIC lehetőséget nyújt arra, hogy a COMMODORE billentyű (C=) alkalmazásával további nyolc funkcióbillentyűt hozzunk létre és mindegyikhez egy-egy (összesen tehát 16!) utasítást vagy egyéb jelláncot rendeljünk.
A 16 funkcióbillentyűt a következő táblázatban foglaltuk össze:

Funkcióbillentyű	*Megvalósítás*
F1	F1
F2	SHIFT + F1
F3	F3
F4	SHIFT + F3
F5	F5
F6	SHIFT + F5
F7	F7
F8	SHIFT + F7
F9	C = SHIFT + F1
F10	C = + SHIFT + F1
F11	C = + F3
F12	C = + SHIFT + F3
F13	C = F5
F14	C = + SHIFT + F5
F15	C = + F7
F16	C = + SHIFT + F7

2.1.1 KEY – a funkcióbillentyűk meghatározása

Formátum: KEY sz, "füzér"
Paraméterek: sz. – A funkcióbillentyű száma (1–16)
füzér – (sztring, angol: string) (füzér vagy füzérváltozó max. 15 karakter)
Funkció: Az sz. számú funkcióbillentyűhöz valamilyen utasítást vagy egyéb jelsorozatot rendel.
Példa: KEY 1, "LIST – 100"

Az F1 gomb és a RETURN lenyomása után a programot a 100. sortól kilistázza.
Haz azt szeretnénk, hogy a funkcióbillentyű az utasítást követő RETURN-t is tartalmazza, akkor a jelsorozathoz hozzá kell kapcsolnunk a RETURN ASCII kódját. Az alábbi két példa mutatja a különbséget:

a) KEY 1, "PRINT 16*15"
b) KEY 3, "PRINT 16*16" + CHR$(13)

Az F1 és F3 gombok lenyomásakor tapasztalhatjuk a különbséget.
Természetesen ennek a megoldásnak sincs mindig értelme. Ha például a téves beírás káros következményekkel járhat (pl.: NEW), célszerű az automatikus RETURN-t elkerülni, így módunk lehet a hibás beírás korrigálására. Gyakorlásképpen adjon utasításokat a funkcióbillentyűknek úgy, hogy a záró RETURN-t is tartalmazzák.
A funkcióbillentyűket program keretében is leköthetjük. Ezáltal lehetővé válik a hozzárendelések tárolása és szükség esetén a program behívásával máris rendelkezésünkre állnak a billentyűk által képviselt funkciók.

```
100 KEY 1, "LIST" + CHR$ (13)
110 KEY 3, "GOTO"
120 KEY 5, "PRINT"
130 KEY 6, "OPEN 1,8,15"
```

A KEY-utasítás a következő sajátossággal rendelkezik, ez egy kicsit meg is nehezíti a használatát: az utasításban vagy egyéb karaktersorozatban szereplő idézőjeleket csak ASCII kód formájában írhatjuk be. Például rendeljük hozzá az F1 billentyűhöz a LOAD "$",8 utasítást. Ez csak így lehetséges:

```
KEY 1, "LOAD" + CHR$(34) + "$" + CHR$(34) + ",8"
```

Sajnos ez egy kicsit bonyolult, de hosszabb jelsorozatok esetén mégis érdemes megpróbálkozni vele. Az idézőjeleket tehát a + CHR$(34) + jelkombinációval kell helyettesítenünk.
További jellegzetesség, hogy a funkcióbillentyűk meghatározása közvetlenül a program segítségével is elvégezhető. Ekkor azonban a karakterek között sem vessző, sem kettőspont nem szerepelhet, mivel ezeket az INPUT nem fogadja el.
A következő programmal párbeszédes formában definiálhatók a funkcióbillentyűk:

P. 2

```
1 REM **********
2 REM * K E Y *
3 REM **********
100 INPUT"A FUNKCIOBILLENTYU SZAMA:";N
110 INPUT"FUNKCIO:";Z$
120 PRINT"A RETURN-T IS KITEGYEM (I/N)?"
130 GET X$:IF X$<>"N" AND X$<>"I" THEN 130
140 IF X$="I" THEN Z$=Z$+CHR$(13)
150 IF LEN(Z$)<16 THEN 170
160 PRINT"MAX. 15 KARAKTER!":GOTO 100
170 KEY N,Z$
180 PRINT"VAN MEG LEKOTES (I/N)?"
190 GET Q$:IF Q$<>"N" AND Q$<>"I" THEN 190
200 IF Q$="I"THEN 100
210 END
```

A program behívás után RUN-nal indítható.
Elfogadja az idézőjeleket, de a vesszőt és a kettőspontot nem.
Kívánság szerint hozzáfűzi a RETURN-t. A karaktersorozat hossza a RETURN-nel együtt max. 15 karakter lehet.
Egy ötlet: Ha az F1 billentyűhöz hozzárendeli a KEY1, "?FRE(O)" + CHR$(13) utasítást, „gombnyomásra" bármikor lekérdezheti a szabad tárkapacitást.
FIGYELEM! A rendelkezésünkre álló SIMON's BASIC-lemez nem teszi lehetővé a funkcióbillentyűk definiálásának törlését. Pl.: a KEY 1, " " utasítás nem az F1 billentyű funkciójának törlését, hanem annak határozatlan karakterekkel való feltöltődését eredményezi. Ennek azonban tulajdonképpen nincs is jelentősége, mert egy új KEY utasítással a funkcióbillentyűhöz bármikor új tartalmat rendelhetünk.

2.1.2 DISPLAY – a funkcióbillentyűk tartalma

Formátum: DISPLAY
Paraméterek: –
Funkció: A funkcióbillentyűk tartalmának kiírása.
Megjegyzés: Helyettesíthető a "KEY Ø" utasítással.

A funkcióbillentyűk alkalmazása jelentősen megkönnyíti a programírást, de azok tartalmát, főleg ha egyszerre többet is használunk, könnyen elfelejthetjük.
Hogy ne kelljen a billentyűket egyenként végigpróbálgatnunk, a DISPLAY utasítással kiírathatjuk a képernyőre azok tartalmát.
A SIMON's BASIC betöltése után a funkcióbillentyűk "üresek", ezért a DISPLAY utasítás beírására az alábbi lista jelenik meg a képernyőn:

```
KEY1, " "
KEY2, " "
KEY3, " "
KEY4, " "
KEY5, " "
KEY6, ""
  :
  :
KEY16, " "
```

A kurzort bármelyik sorra állítva a KEY utasítás módosítható (a végén természetesen a RETURN-t is le kell nyomni).
A DISPLAY utasítást minden további nélkül nyomtatóra is alkalmazhatjuk. Gyakorlott programozók most azt mondják, hogy „ez csak természetes", de ne felejtsük el, hogy könyvünk nem kizárólag haladóknak készült.
Tehát az eljárás a következő:

```
OPEN 1,4      <RETURN>
CMD 1         <RETURN>
DISPLAY       <RETURN>
```

A funkcióbillentyűk tartalma most "írásban" is rendelkezésünkre áll.
Végezetül ne felejtsük el lezárni a nyomtatócsatornát a CLOSE 1-gyel.

2.1.3 Ellenőrző kérdések

1. Mint tudjuk, a funkcióbillentyűket a KEY utasítással köthetjük le. Az alábbi utasítások egyike nem helyes:

   a) KEY 1, "LIST"
   b) KEY Ø, "RUN"
   c) KEY B, A$

2. Melyik utasítással rendelné hozzá az F5 billentyűhöz a "LIST" utasítást, automatikus RETURN–nel?

   a) KEY 5, "LIST" + "RETURN"
   b) KEY 5, "LIST" + CHR$(34))
   c) KEY 5, "LIST" + CHR$(13)

3. Rendelje hozzá az F1 billentyűhöz a LOAD "$",8 utasítást, automatikus RETURN-nel. Melyik lehetőséget választaná?

   a) KEY1, "LOAD"$",8" + CHR$(13)
   b) KEY1, "LOAD"$",8" + CHR$(34)
   c) KEY1, "LOAD" + CHR$(34) + "$" + CHR$(34) + ",8" + CHR$(13)
   d) KEY1, "LOAD" + CHR$(13) + "$" + CHR$(13) + ",8" + CHR$(34)

4. A funkcióbillentyűhöz rendelt karaktersorozat legfeljebb hány karakterből állhat?

   a) 255
   b) 15
   c) 16
   d) 256

2.2 A kényelmes programozás

2.2.1 AUTO – automatikus sorszámozás

Formátum: AUTO ssz, lp
Paraméterek: ssz – a sorszámozás kezdőértéke
lp – lépték (1–255)
Funkció: Automatikus sorszámozás tetszőleges kezdőértékkel és léptékkel.
Példa: AUTO 100,10

A RETURN lenyomása után a képernyő bal szélén megjelenik a 100-as sorszám és mögötte a kurzor. Beírható a programsor, majd újabb RETURN után megjelenik a 110-es sorszám. Folytatódhat a program írása.
Mint tudjuk, a programírás során a BASIC-sorokat sorszámmal kell ellátni, amelyek "közlik" az interpreterrel az utasítások végrehajtásának sorrendjét. Léptékként az 1-et is választhatjuk, de akkor a továbbiakban nincs lehetőségünk újabb sorok beszúrására. Érdemes tehát megfelelően nagy (általában 10-es) léptéket választani.
Az AUTO utasítás megkönnyíti számunkra a programírást, hiszen a sorszámok beírásával nem kell foglalkoznunk.
Ha a sorszám megjelenése után közvetlenül, lenyomjuk a RETURN-t (úgy, hogy a sorba nem írunk semmit), az automatikus sorszámozás megszűnik és a hagyományos módon dolgozhatunk tovább. (A begépeléskor elkövetett hibák csak így javíthatók. Utána az AUTO utasítás újra kiadható).
Megjegyzendő, hogy az utasítás a sorszámozás végét (63 999) nem ismeri fel, ezért az alábbi példában látható eset következhet be:

```
AUTO 63500,100          RETURN
63500 PRINT               "
63600 PRINT               "
63700 PRINT               "
63800 PRINT               "
63900 PRINT               "
64000 PRINT               "

?SYNTAX ERROR.
READY
64100
```

A 64000-es sorszám kiadása SYNTAX ERROR-t eredményez, amelyhez a gép a 64100-as sorszámot kapcsolta.
Ha most be szeretnénk fejezni az automatikus sorszámozást, a RETURN lenyomása újabb SYNTAX ERROR-hoz vezet.
A számozás határát persze a legritkább esetben érjük el és ha elérjük is, nincs káros következménye. Elég tehát, ha a fentieket egyszerűen csak tudomásul vesszük.
Ha meglévő programrészt kívánunk az AUTO utasítás használatával folytatni, ügyelnünk kell a következőkre: a kezdőérték és a lépték helyes megválasztásával elkerülhetjük, hogy olyan sorszám keletkezzék, amely esetleg már létezik a programban. Ha ez előfordul, az adott sor felülíródik. Az automatikus sorszámozás befejeződik, ha az utoljára kiadott sorszám a programban már előfordult, ill. a sor törlését eredményezi.
Ha ezt időben észrevesszük, a RUN/STOP és a RESTORE billentyű együttes lenyomásával „mászhatunk" ki a bajból. Mindezt kipróbálhatjuk az alábbi programon:

P. 3

```
1 REM ***********
2 REM * A U T O *
3 REM ***********
100 FOR I=65 TO 90
110 PRINT I;
120 PRINT "ASCII ERTEKHEZ :";
130 PRINT CHR$(I);
140 PRINT " KARAKTER";
150 PRINT " TARTOZIK"
160 NEXT I
```

Tegyük fel, hogy a programot a 200. sortól folytatni szeretnénk. Ehhez az AUTO 200,10-utasítást kellene beadnunk, de véletlenül AUTO 100,10-et írunk. Erre a számítógép kiteszi a 100-as sorszámot. Ha észrevesszük a hibát és a RETURN-t lenyomva ki akarunk lépni az automatikus sorszámozásból, a 100. sor törlődik. Olyenkor csak a RUN/STOP-RESTORE segít, vagy a törölt sort újból be kell írnunk.

2.2.2 RENUMBER – átsorszámozás

Formátum: RENUMBER ssz, lp
Paraméterek: ssz – az átsorszámozás kezdősora
lp – lépték
Funkció: Meglévő program átsorszámozása az adott sortól, tetszőleges léptékkel
Példa: RENUMBER 100,10

A tárban lévő programot 100-as sorral kezdődően 10-es léptékkel átsorszámozza.
Megjegyzés: Az utasítás a GOTO/GOSUB utasítások ugrócímeit nem módosítja!
A RENUMBER segítségünkre van, ha a megírt program túlságosan áttekinthetetlen, vagy helyenként túl sűrű a sorszámozás. Egyetlen, de nem elhanyagolható hiányossága, hogy az ugrócímeket nem változtatja meg az új sorszámozás szerint. Mivel az alap-BASIC-ben történő programozás szinte elképzelhetetlen GOTO és GOSUB utasítások nélkül, azt hihetnénk, hogy emiatt ez az utasítás nem is alkalmazható komolyabb eredménnyel. A későbbiekben az 5.2 fejezetben bemutatjuk a SIMON's BASIC azon lehetőségét, mely szerint ezek az utasítások szimbolikus címekkel is kiadhatók és így a RENUMBER utasítás is maradéktalanul alkalmazható.

Próbáljuk ki az utasítást a következő programon:

P. 4

```
1 REM ********************
2 REM * R E N U M B E R *
3 REM ********************
10 PRINT"10"
11 PRINT"11"
12 PRINT"12"
15 PRINT"15"
20 PRINT"20"
```

Látjuk, hogy a sorszámozás szabálytalan, sőt a 10-11. és 11-12. sor közé már újabb sort nem is iktathatunk. Számozzuk át tehát a programot pl. 10-es léptékkel.

```
RENUMBER 10,10.
```

A LIST utasítással kiírathatjuk az átszámozott, rendezett programunkat.

```
10 REM ********************
20 REM * RENUMBER *
30 REM ********************
40 PRINT "10"
50 PRINT "11"
60 PRINT "12"
70 PRINT "15"
80 PRINT "20"
```

A RENUMBER utasítás további kísérletezésre ösztönöz. Mi történik például, ha a léptéket 0-nak választjuk?
Próbáljuk ki!

Tehát! RENUMBER 10,0 <RETURN>,
majd: LIST

Szokatlan programlista jelenik meg a képernyőn:

```
10 REM ********************
10 REM * RENUMBER *
10 REM ********************
10 PRINT "10"
10 PRINT "11"
10 PRINT "12"
10 PRINT "15"
10 PRINT "20"
```

Próbáljuk ki, hogy működik-e? Meglepődve fogjuk tapasztalni, hogy igen. Sőt, az eljárásból még hasznunk is származott, mert programunk számára létrehoztunk egyfajta védettséget. Ez a program ugyanis többé nem módosítható. Nem iktatható be új sor és az első programsor kivételével a többi nem változtatható.
Győződjünk meg róla! Próbáljuk megváltoztatni a PRINT 15-öt PRINT 17-re. Nem sikerül. Eltűnt viszont az első REM-sor és ehelyett áll most a PRINT 17. Ezentúl, bármely sort módosítjuk, mindig csak ez az egy sor fog változni.
A program egyes sorainak törlése is elég bonyolulttá vált. Ha például a PRINT "20" sort törölni akarjuk, előtte "10" RETURN-nel az összes többi sort is ki kell törölni. A program kiegészítése 10-nél kisebb és 10-nél nagyobb sorszámokkal természetesen továbbra is lehetséges.

Az utasításnak van egy különlegessége:

Ha a sorszámozás átlépi a 63999-es határt, akkor az így kapott sorok nem törölhetők. Kipróbálhatjuk az előbbi programon, ha beadjuk a RENUMBER 63800,100 utasítást. A LIST utasítás az alábbi programot eredményezi.

```
63800 PRINT "10"
63900 PRINT "11"
64000 PRINT "12"
64100 PRINT "15"
64200 PRINT "20"
```

Próbáljuk kitörölni a 64000-es sort. Nem fog sikerülni. A gép mindig SYNTAX ERROR-ral válaszol. Ezenkívül arra sincs módunk, hogy ezekre a sorokra GOTO-val, vagy GOSUB-bal hivatkozzunk. Ha a gép ilyet talál, szintén kiírja a SYNTAX ERROR-t. Ezt a programot csak egy "rendes" RENUMBER-rel hozhatjuk helyre.

2.2.3 MERGE – programok összefűzése

Formátum: MERGE "név", ksz
Paraméterek: név – a betöltendő program neve
ksz – készülékszám, ahonnan a betöltés történik; (1-kazettás egység, 8-lemezegység)
Funkció: A tárban lévő programhoz hozzáfűzi a betöltendő programot
Példa: MERGE"BEV. – RUT." ,8

A lemezegységről betöltendő program neve BEV.-RUT., melyet a tárban lévő programhoz fűz. Természetesen ilyenkor az is megmarad.

Megjegyzés: A két program nem tartalmazhat azonos számú sorokat.

Gyakorlott programozók olyan saját szubrutinokat tudnak kidolgozni, amelyek több programba is beépíthetők. Ezeket a rutinokat valamilyen módon nyilván kell tartani, és jó, ha azok bármikor elérhetők. Elképzelhető, hogy ezeket kinyomtatjuk és ha kell, papírról bebillentyűzzük. Ez azonban nem valami hatékony eljárás. Sokkal célravezetőbb, ha ezeket lemezre rögzítjük, ahonnan szükség esetén bármikor betölthetők és a főprogramhoz fűzhetők. Ez azonban csupán a Commodore operációs rendszerét tekintve, szinte megoldhatatlan. (Kivéve a POKE utasítással, de ez nagyon körülményes.)
A MERGE utasítás ezt a hiányosságot pótolja. Alkalmazásakor a következőkre kell figyelemmel lennünk:
- A két program egyetlen azonos sorszámot sem tartalmazhat.
- A betöltendő program sorszámainak nagyobbnak kell lenniük a tárban lévő program sorszámainál.
- A két programban különböző változóneveket kell alkalmazni, mert az azonosság sok és nagyon nehezen kijavítható hibát eredményezhet a későbbiek során.

A programok MERGE-el történő összekapcsolásakor előforduló problémákat úgy kerülhetjük el a legkönnyebben, ha a programokat azonos elv szerint, áttekinthetően szerkesztjük meg.
A gyakran használt rutinokat célszerű azonos sorszámtartományba írni, amelyet a főprogramok írásához már nem használunk fel. Ugyanez vonatkozik a változónevekre is. Mi történik, ha az előbb említett három szabályt nem tartjuk be? Próbáljuk ki az alábbi programon:

```
10 PRINT"10.SOR – FŐPROGRAM"
20 PRINT"20.SOR – FŐPROGRAM"
30 PRINT"30.SOR – FŐPROGRAM"
```

Próbáljuk hozzáfűzni a lemezen tárolt MERGE-nevű programot:

P. 5

```
1 REM ************
2 REM * M E R G E *
3 REM ************
10 PRINT"10.BEOLVASOTT SOR"
20 PRINT"20.BEOLVASOTT SOR"
30 PRINT"30.BEOLVASOTT SOR"
```

Az összefűzés a MERGE"MERGE",8 paranccsal történik. A READY visszajelzés után LIST-tel megnézhetjük, mit műveltünk. Az eredmény valami ilyesmi:

P. 6

```
10 PRINT"10.SOR-FOPROGRAM"
20 PRINT"20.SOR-FOPROGRAM"
30 PRINT"30.SOR-FOPROGRAM"
1 REM **********
2 REM * M E R G E *
3 REM **********
10 PRINT"10.BEOLVASOTT SOR"
20 PRINT"20.BEOLVASOTT SOR"
30 PRINT"30.BEOLVASOTT SOR"
```

Láthatjuk, hogy némelyik sorból kettő is van, sőt a sorok nincsenek sorbarendezve sem. Vajon hogy működik ez a program? Próbáljuk ki! Indítsuk el RUN-nal és meglátjuk mi fog történni. Legnagyobb meglepetésünkre a BASIC értelmező (interpreter) nem jelez hibát, hanem szó nélkül végrehajtja a programot.
Ez a következőképpen magyarázható:
A RUN beírásakor először meghatározásra kerül az első BASIC-sor címe. Ez a nulláslapban (zero-page), a főtároló első 256 byte-jában található. A számítógép itt tárolja azokat az információkat, amelyekre az operációs rendszernek vagy az interpreternek a rutinoknál szüksége van. Olyasmi ez, mint egy BASIC-program, amely a fontosabb információkat változókban helyezi el. A nulláslap tehát az operációs rendszer és az interpreter „változótartománya".
Az interpreter tehát először az első BASIC-sor címét helyezi el (C64-es esetén mindig $0801, azaz 2049) és rögzíti mint "aktuális" sort a nulláslapon. Minden BASIC-sor ezután a következő sor címével kezdődik, de ezt a LIST utasítás nem jelzi ki. Az összekapcsolás tehát nem a programsorszámok, hanem egy speciális követőcím alapján történik. Az eredeti sorszámokra a továbbiakban csak az ugróutasításoknál (GOTO/GOSUB) van szükség. Ezek megkeresik az adott sort, és "aktuális" sorrá nyilvánítják. Az interpreter a program végét a "0" követőcímről ismeri fel.
Megállapíthatjuk tehát, hogy azonos sorszámokat tartalmazó program esetében csak az ugróutasítások végrehajtásakor merül fel probléma.
Most nézzük meg, hogy pl. egy GOTO 10 utasítás hatására az interpreter melyik 10-es sorra ugrik. Láthatjuk, hogy mindig az elsőre. Ez azért van, mert az interpreter a keresést mindig a program elején kezdi.
Ilyen rendetlen programot természetesen nem használhatunk fel. Tegyük rendbe egy RENUMBER 10,10 utasítással.

P. 7

```
10 PRINT"10.SOR-FOPROGRAM"
20 PRINT"20.SOR-FOPROGRAM"
30 PRINT"30.SOR-FOPROGRAM"
40 REM **********
50 REM * M E R G E *
60 REM **********
70 PRINT"10.BEOLVASOTT SOR"
80 PRINT"20.BEOLVASOTT SOR"
90 PRINT"30.BEOLVASOTT SOR"
```

Megjegyzések: A nulláslapon a program kezdetén kívül a program vége is tárolódik. Ha egy BASIC program végét POKE utasítással "felemeljük", a MERGE alkalmazásakor problémák adódhatnak. Ha ilyesmivel kísérletezünk és a MERGE emiatt hatástalanná válik, a NEW és OLD utasításokkal állítható be újra az eredeti programvég.
Kísérletező kedvű SIMON's BASIC rajongóknak ajánljuk, próbálják ki a MERGE"$",8, majd a RENUMBER 10,10 utasítások beírását. Az eredmény LIST-tel kiíratható.

2.2.4 Ellenőrző kérdések

1. Melyik paranccsal valósítható meg az automatikus sorszámozás 100-tól, 10-es lépték-kel?

   a) AUTO 100,10
   b) AUTO 10,100
   c) AUTO 100 BY 10
   d) AUTO 10 BY 100

2. Az AUTO-parancsban mekkora a lépték maximális értéke?

   a) 255
   b) 256
   c) nincs korlátozva

3. Melyik parancsokra nem hat a RENUMBER-utasítás?

   a) FOR/NEXT-ciklus
   b) GOTO
   c) GOSUB
   d) INPUT

4. Mire kell ügyelni a MERGE-parancs alkalmazásánál?

   a) A kezdő sorszámnak 100-nál nagyobbnak kell lennie.
   b) Azonos sorszámok nem lehetnek.
   c) A betöltendő program sorszámainak nagyobbnak kell lenniük a tárban lévő prog-ram sorszámainál.
   d) A MERGE alkalmazása előtt a programot RENUMBER-rel át kell sorszámozni.

a
a
b, c
b, c

2.3 Listázási segédeszközök

Programozás közben elkerülhetetlen a programlistába történő betekintés. Legkönnyebben a kinyomtatott lista használható, de ennek elkészítése időigényes, nincs is mindenkinek nyomtatója, és kisebb változtatásokhoz felesleges is. Ilyen esetekben legjobb, ha a programlista a képernyőn jelenik meg. A C64-es erre nem sok lehetőséget nyújt. Csak arra van módunk, hogy a LIST-tel elindítsuk, CTRL-lel lassítsuk és RUN/STOP-pal megszakítsuk, majd egy újabb LIST beírásával folytassuk a listázást.
Standard-BASIC-ben a LIST-utasítás a következők szerint használható:

LIST	– a teljes program kilistázása
LIST n	– az n. sor kilistázása (pl. LIST 100)
LIST –n	– listázás az n. sorig (pl. LIST –100)
LIST n–	– listázás az n. sortól (pl. LIST 100–)
LIST n1–n2	– listázás az n1. sortól az n2. sorig (pl. LIST 100–500)

A SIMON's BASIC ennél jóval több lehetőséget nyújt a listázásra.

2.3.1 PAGE – a programlista felosztása

Formátum: PAGE N
Paraméterek: n – az egy részben levő képernyősorok száma mínusz 1 (1–23)
Funkció: A programlistát n+1 soronként részekre osztja
Példa: PAGE 10

A listázás 11 soronként történik.

Megjegyzés: A PAGE utasítást a PAGE∅ hatástalanítja.

A PACE segítségével a BASIC programot részekre oszthatjuk. A részek hossza szabadon választható, de nem haladhatja meg az egy képernyőoldalt. Ne feledjük, hogy nem programsorokat, hanem képernyősorokat kell megadni!

Az utasítás hibája, hogy 23-nál nagyobb számok esetén nem hajtható végre, de hibaüzenetet mégsem kapunk. Az lenne a logikus, ha a 255-nél nagyobb érték beírásakor megjelenő hibaüzenetet (ILLEGAL QUANTITY ERROR) a 23-nál nagyobb, hatástalan értékek is előidéznék.
A PAGE-t a következők szerint alkalmazzuk:

1. LIST <RETURN> (megjelenik az első rész)
2. <RETURN> (megjelenik a következő rész)
3. <RUN/STOP> (a listázás megszakítása)

Megjegyzések: Előfordul, hogy a LIST <RETURN> beadás után a képernyőn megjelenik egy 1-es és eltűnik a kurzor. Ilyenkor újból meg kell nyomni a RETURN-t, hogy a listázás elkezdődjön. RUN/STOP-pal csak addig szakítható meg a listázás, míg egy teljes rész ki nem íródott.

2.3.2 OPTION – SIMON's BASIC utasítások kiemelése

Formátum: OPTION n
Paraméterek: n – bekapcsoláskor 10, kikapcsoláskor 10-től különböző, de 256-nál kisebb szám
Funkció: Programlista készítésekor a SIMON's BASIC utasítások inverz alakban jelennek meg
Példa: OPTION 10 – az inverz ábrázolás bekapcsolása
OPTION 0 – az inverz ábrázolás kikapcsolása

Ha a SIMON's BASIC utasításokat tartalmazó programunkat standard-BASIC-be kívánjuk átírni, rendkívül jól használható ez az utasítás. Gondoljuk csak el, milyen hosszadalmas lenne kiemelések nélkül a SIMON's BASIC-utasítások felkutatása bármely programban. Az utasítás működését kipróbálhatjuk valamelyik SIMON's BASIC-ben írt programunkon, pl. a 2.1.1 pontban található KEY-nevűn. Töltsük be a lemezről, majd először OPTION 10-et beírva LIST-tel írassuk ki a képernyőre.
Amennyiben a nyomtatónk rendelkezik COMMODORE-üzemmóddal, mindezt papíron is ellenőrizhetjük.

2.3.3 DELAY – a listázási sebesség változtatása

Formátum: DELAY n
Paraméterek: n – késleltetési érték (1–255)
Funkció: Az n érték függvényében változtatja a listázás sebességét,
Példa: DELAY 10
A listázás sebessége SHIFT-tel kb. 50 kar/s lesz.

A programok listázását alap állapotban a CTRL billentyűvel lassíthatjuk. Ez az utasítás lehetővé teszi számunkra, hogy a SHIFT lenyomásával a listázást különböző mértékben lassítsuk. Megállítani a C= billentyűvel lehet.
Néhány jellegzetes értékhez a következő listázási sebesség tartozik:

n	kar/s
1	290 (kb. normál)
5	100
10	50 (kb. mint a CTRL-lel)
20	25

A PAGE és DELAY utasítások együttesen is alkalmazhatók, ezáltal egy-egy programrészlet késleltetve jeleníthető meg a képernyőn.
Idővel kialakíthatjuk magunknak a saját PAGE-DELAY értékeinket, amelyeket minden program első sorában helyezhetünk el, például így:

```
1 PAGE 10 : DELAY 5
```

Ezáltal a program beolvasását és indítását követően ez a listázási segédeszköz rendelkezésünkre áll.

2.3.4 FIND – a füzérek (stringek) felkutatása

Formátum: FIND kód, vagy FIND füzér
Paraméterek: kód – minden, ami az idézőjelen kívül áll;
füzér (füzérek) vagyis idézőjelbe tett karakterek;
Funkció: A BASIC programban megkeresi az adott karakterkombinációt.
Példa: FINDPRINT
Minden olyan sort kiír, amelyben a PRINT utasítás szerepel.

Megjegyzés: Szóköz csak akkor állhat a FIND után, ha az is a keresett kombinációhoz tartozik.

Ez az az utasítás, amelyet az eredeti kézikönyv alapján a legkevésbé tudunk megfelelően alkalmazni. Nem is csoda, hiszen összesen 11 sor tárgyalja, holott részletes megismertetéséhez talán 11 oldal is kevés volna.
Mielőtt elmélyednénk megismerésében, fel kell hívni a figyelmet a következő fontos jellemzőre.

Tudjuk, hogy BASIC-ben a szóközöknek nincs hatásuk, tehát a PRINT10 ugyanazt jelenti, mint a PRINT 10. A FIND utasításra mindez nem érvényes, mert a kitett szóköz a "keresõfogalom" részét képezi.
A FINDPRINT 10 kiírja az összes olyan sort, amelyben PRINT 10 szerepel, de azokat nem, amelyekben PRINT10 van.
A paraméterek ismertetésekor már kitértünk arra, hogy ezzel az utasítással BASIC-kódot vagy stringet kereshetünk ki. A különbség érzékeltetésére nézzük a következõ programot:

```
10 PRINT "KÉREM AZ A ÉRTÉKÉT"
20 INPUT A
30 PRINT "BEADTA";A;"-T"
```

Keressük ki azokat a sorokat, amelyekben "A" van. Beírjuk tehát, hogy

```
FINDA          <RETURN>
```

Vajon mely sorok fognak megjelenni? A 20-as és a 30-as. Igaz, hogy a 10. sorban is található egy "A", de az idézõjelen belül van. Ez az utasítás az idézõjelben levõ karakterkombinációkat csak teljes egészében találja meg. Pl. ha a programban szerepel ez a sor:

```
10 PRINT "BUDAPEST MAGYARORSZAG FŐVÁROSA",
```

akkor a 10. sor csak akkor fog a képernyõn megjelenni, ha a FIND"BUDAPEST MAGYARORSZÁG FŐVÁROSA" utasítást adjuk meg, de nem jelenik meg sem a FIND"BUDAPEST" sem a FIND"MAGYARORSZÁG", sem pedig a FIND"FŐVÁROSA" utasításokra.
Röviden tehát: a FIND utasítás nem alkalmas arra, hogy az idézõjelek közt szereplõ szövegnek csak egy részét keressük ki vele.
Lehetõségünk van azonban arra, hogy a fenti szöveget tartalmazó sort egy rövidebb, nevezetesen a

```
FIND"B
```

utasítással keressük ki, bár ilyenkor a képernyõn az összes olyan sor megjelenik, amelyben B-vel kezdõdõ füzér szerepel.
Töltsük be a lemezrõl a FIND címû programot és gyakoroljuk az utasítás használatát!

P. 8

```
1 REM **********
2 REM * F I N D *
3 REM **********
20 PRINT"KEREM AZ A ERTEKET";
30 INPUT A
40 PRINT"KEREM A  B ERTEKET";
50 INPUT B
60 C=A+B
70 PRINT"A KET SZAM OSSZEGE:";
80 PRINT C
```

A következõ táblázat alapján próbáljuk ki a lehetõségeket!

FIND-utasítás	*kijelzett sorok*
FINDREM	1, 2, 3
FINDA	30, 60
FIND A	30,
FINDB	50, 60
FIND B	50,
FIND"KÉREM	20, 40
FINDC	60, 80
FIND C	60,
FINDA+B	60,
FINDPRINT	20, 40, 70, 80
FINDINPUT	30, 50
FIND;	20, 40, 70

A kísérletezést folytathatjuk a FIND PRINT utasítás beírásával. A gép most nem jelez ki semmit. Mivel a sorszám és az első karakter közti szóközt az interpreter automatikusan teszi ki, ebben az esetben azt nem veszi tudomásul.
Természetesen a BASIC kulcsszavak rövidíthetők, valamint a speciális karakterek is kikereshetők. Így például a FINDPRINT egyenértékű a "FIND?" utasítással, a "FIND;" pedig kiírja az összes olyan sort, amelyben pontosvessző szerepel.

2.3.5 Ellenőrző kérdések

1. Az alábbi utasítások közül melyikkel bontható a programlista 10 képernyősorra?

   a) PAGE 10
   b) PAGE 11
   c) PAGE 9

2. Melyik billentyűvel lehet a PAGE utasítással felosztott listát lapozni:

   a) RETURN
   b) COMMODORE (C=)
   c) SHIFT
   d) CTRL

3. Melyik utasítás aktivizálja a SIMON's BASIC utasítások inverz kiemelését?

   a) OPTION
   b) OPTION 1
   c) OPTION 10
   d) OPTION 0

4. Melyik utasítással állítható be a listázási sebesség?

   a) DECAY
   b) DELAY
   c) DELLAY

5. Melyik billentyű késlelteti a listázást a képernyőn?

   a) RETURN
   b) SHIFT
   c) COMMODORE (C=)

6. Adott az alábbi program:

   10 INPUT"KÉREK EGY SZÁMOT";A
   20 PRINT"AZ";A;"SZÁMOT ADTA BE"

   Melyik utasítás nem jelez ki egy sort sem a képernyőn?

   a) FINDA
   b) FIND A
   c) FIND"A

3. FEJEZET
Hibakezelés

Bármilyen jól programoz is valaki, a kész programok átvizsgálására és hibajavításra mindig szükség lesz. Milyen hibakezelési lehetőségeket nyújt a C64-es alapváltozata? Két módozatot különböztetünk meg: külső és belső hibakezelést.
Külső eljárás például a programszakadást követő változó-kijelzés, vagy az interpreter által hibásnak ítélt sor kilistázása.
A belső eljárást olyan utasítások jelentik, amelyek ugyan a programban helyezkednek el, de nem képezik annak szerves részét. Ezeket a hibaellenőrzés végén töröljük, vagy REM-ekkel hatástalanítjuk.
A C64-es összesen egy utasítással rendelkezik a belső hibakezeléshez. Ez a STOP. Funkciója és felhasználási lehetőségei alig ismertek. A programban ahhoz a részhez iktatjuk be, ahol véleményünk szerint célszerű lehet a program megszakításával egy időben a változók ellenőrzése. Amikor a programfutás a STOP-hoz ér, a képernyőn megjelenik a "BREAK IN..." kijelzés. Ilyenkor a változók pillanatnyi értékeire rákérdezhetünk (pl.: PRINT A$), majd CONT-tal folytathatjuk a programot egy újabb STOP-ig, ill. a program végéig.
Figyelmeztetés! A program nem folytatható a CONT-tal, ha a szakadáskor valamelyik programsort megváltoztatjuk. Ilyenkor a változók is törlődnek és az alábbi hibaüzenetet kapjuk:

?CAN'T CONTINUE ERROR

A változókat azonban tetszés szerint módosíthatjuk. Például hozzárendelhetünk egy változóhoz valamilyen ellenőrzőértéket, majd CONT-tal folytathatjuk a programot.
A C64-es más, hibakezelési utasítással nem rendelkezik, ezért minden további eljárást a programozónak kell kidolgoznia. A program ellenőrzése után azonban ezeket a sorokat hatástalanítani kell. Bizonyára csodálkozna az általunk készített program felhasználója, ha a program futása közben egyszer csak megjelenne a „BREAK IN 1220" üzenet a képernyőn. Érdemes tehát feljegyezni a hibakereső sorokat, hogy a végén könnyen megtaláljuk azokat.
Az eredeti STOP és CONT utasításokon túl a SIMON's BASIC néhány újabb utasítással könnyíti a hibakezelést. A következőkben ezeket mutatjuk be.

3.1 Külső hibakezelés

3.1.1 TRACE – a programfutás kijelzése

Formátum: TRACE n
Paraméterek: n – bekapcsoláskor értéke 10, kikapcsoláskor ettől eltérő
Funkció: A képernyő jobb felső sarkában kijelzi a legutóbb feldolgozott sorokat
Példa: TRACE 10
Bekapcsolja a TRACE üzemmódot.

Ez az utasítás a hibakezelés egyik közkedvelt eszköze. Egyes BASIC interpreterek (pl. MICROSOFT-MBASIC), már hardver szinten tartalmazzák ezt a lehetőséget.
A SIMON's BASIC birtokában egy lehetőség a C64-es használóinak is rendelkezésére áll. Működése rendkívül egyszerű. A program indítása előtt a TRACE 10 utasítással aktivizáljuk a TRACE-üzemmódot, amelynek eredményeként a programfutás során a képernyő jobb felső sarkában megjelenő "ablak"-ban látható a legutóbb feldolgozott 6 sor első 6 karaktere. A programfutás így egy kicsit lelassul, mert az interpreternek a főprogramon kívül a TRACE-programot is fel kell dolgoznia.
A lemezről betölthető TRACE című programon tanulmányozhatjuk az utasítást:

P. 9

```
1 REM ************
2 REM * T R A C E *
3 REM ************
5 PRINT CHR$(147);
10 PRINT CHR$(19);"10":GETX$:IFX$="" THEN10
20 PRINT CHR$(19);"20":GETX$:IFX$="" THEN20
30 PRINT CHR$(19);"30":GETX$:IFX$="" THEN30
40 PRINT CHR$(19);"40":GETX$:IFX$="" THEN40
50 PRINT CHR$(19);"50":GETX$:IFX$="" THEN50
60 END
```

A CHR$(147) és a CHR$(19) kódok a CLR/HOME és a HOME speciális karaktereket jelentik.
Mielőtt a programot elindítanánk, aktivizáljuk a TRACE utasítást. Az ablak utolsó sora mutatja a legutoljára feldolgozott 5. sort. A 10. sorban levő PRINT-et a gép már végrehajtotta, de a sor folytatásához le kell nyomnunk egy gombot. Ekkor az ablakban megjelenik a 10. sor is. A további sorok esetében ugyanígy kell eljárni, míg a program végére nem érünk.
Ekkor az ablak tartalma a következő:

```
#5
#10
#20
#30
#40
#50
```

Mivel a program befejeződött, a 60. sor végrehajtását a gép már nem jelzi ki.
Kapcsoljuk ki a TRACE O-val a TRACE üzemmódot, majd ellenőrizzük RUN-nal, hogy a kikapcsolás megtörtént-e.
Néhány dologra ennél az utasításnál is figyelnünk kell. Kapcsoljuk vissza a TRACE üzemmódot, indítsuk el a programot, majd rögtön szakítsuk meg a RUN/STOP-pal. A képernyőn megjelenik a "BREAK IN 10" üzenet.
Ha most az aktuális 5. sorba beírunk egy LIST utasítást, a gép SYNTAX ERROR-ral válaszol. Mi lehet ennek az oka? A válasz egyszerű. Az 5. sor végébe belenyúlik a TRACE-ablak, amelyet az interpreter a LIST utasítás részeként értelmez. Ez okozza a hibát. Kiküszöbölése kézenfekvő: az utasítás beírása előtt a kurzort néhány sorral lejjebb kell vinni.
Ez az utasítás nem teszi lehetővé a program sorról sorra történő végrehajtását ("single-step"), valamint a sorok az ablakban nagyon gyorsan futnak. Emiatt a program ellenőrzése eléggé nehézkes. Megkönnyítésére ajánlatos a programba késleltető cilusokat iktatni, vagy alkalmazni a PAUSE nevű SIMON's BASIC utasítást. A PAUSE 1 például 1 másodperces késleltetést jelent. A végleges programból ezeket törölnünk kell, így a helyüket feltétlenül jegyezzük fel!

3.1.2 RETRACE – az utolsó TRACE-ablak kijelzése

Formátum: RETRACE
Paraméterek: –
Funkció: Ha töröljük a képernyőt, ezzel az utasítással visszahozhatjuk az utolsó TRACE-ablakot

Az előző pontban már láttuk, hogy TRACE-üzemmódban a képernyő első 6 sorába nem írhatunk utasítást, csak akkor, ha előbb töröltük a képernyőt. Ilyenkor természetesen a TRACE-ablak is eltűnik. Ha a későbbiek során mégis kíváncsiak vagyunk a legutolsó TRACE-sorokra, ezzel az utasítással újra megjeleníthetjük.
Hívjuk be most a lemezről a RETRACE-programot és próbáljuk ki a működését:

P. 10

```
1 REM ****************
2 REM * R E T R A C E *
3 REM ****************
5 PRINT CHR$(147);
10 FOR I=1 TO 10
20 PRINT I
30 NEXT
40 PRINT CHR$(147)
```

Kapcsoljuk be a TRACE üzemmódot (TRACE 10), majd futtassuk le a programot. Az utolsó sor törli a képernyőt, eltűnt tehát a TRACE-ablak is. Írjuk be a RETRACE utasítást. Az ablak tartalma a következő lesz:

```
#20
#30
#20
#30
#20
#30
```

Ebből láthatjuk, hogy a tulajdonképpeni ciklus a 20. és a 30. sorban van, az 5., 10. és 40. sor csak egyszer futott le.

3.1.3 DUMP – változók kijelzése

Formátum: DUMP
Paraméterek: –
Funkció: A program összes változóját pillanatnyi értékükkel együtt kijelzi
Megjegyzés: Negatív számok és üres füzérek problémát okozhatnak.

A program ellenőrzése során hasznos lehet az alkalmazott változók pillanatnyi értékeinek figyelemmel kísérése. Ezek kijelzésére a PRINT-utasítás is alkalmas, de hátránya az, hogy a változókra csak egyenként tudunk rákérdezni. A DUMP-utasítás lehetővé teszi, hogy az összes változót, aktuális értékével együtt a képernyőn listaszerűen kiírathassuk. A listázás – hasonlóan a LIST-utasításhoz – a CTRL-billentyűvel késleltethető.
Ezek alapján első ránézésre nagyon kényelmesnek tűnik a DUMP-utasítás, de pozitív benyomásainkat néhány hiányossága mégis elrontja.

1. Nem jelzi ki a tömböket (pl. A(23) vagy A$(2).
2. A negatív számokat pozitívként jelzi ki.
3. A negatív egész-változókhoz (pl. A%) hozzáadja a 2↑16 értéket.
4. A határozatlan üres füzéreket (pl. A$-" ") feltölti 256 határozatlan karakterrel.

Ezek az utasítás súlyos hiányosságai, ezért csak pozitív változók és definiált füzérek esetében használható megbízhatóan.
Írjuk be például a következő sort közvetlen üzemmódban, azaz sorszám nélkül:

```
A = 10 : B = -5:C %= - 57 : D$ = " "
```

Mielőtt a DUMP-ot beírjuk, üljünk le, mert az eredmény eléggé meghökkentő lesz:

```
A=10
B=5
C%=65479
D$=" ???... ???" (256 db kérdőjel!)
```

Csak az A változót kaptuk vissza helyesen. Jobb tehát, ha most mégiscsak a PRINT-et használjuk:

```
PRINT A,B,C%,D$
```

Így hibátlan eredményt kaptunk:

```
 10     -5      -57.
```

Az üres füzéreket ez az utasítás sem jelzi ki.
Ne húzzuk ki azért a DUMP utasítást se a könyvből, hiszen ha szem előtt tartjuk az említett hiányosságokat, korlátozottan bizonyos esetekben hasznunkra lehet. Különben is, egy olyan átfogó programbővítés esetében, mint amilyen a SIMON's BASIC, az efféle hibákat megbocsáthatjuk.

3.1.4 COLD – visszatérés alapállapotba

Formátum: COLD
Paraméterek: –
Funkció: A SIMON's BASIC-et alapállapotba hozza

Ha a C64-et SIMON's BASIC nélkül alapállapotba kívánjuk hozni, a SYS 64738-at kell beírnunk. Ugyanez az utasítás SIMON's BASIC-kel a SIMON's BASIC betöltését követő állapotot eredményezi. Ez az utasítás elég nehezen jegyezhető meg, ill. könnyen elfelejthető, ezért sokkal egyszerűbb a COLD-utasítás alkalmazása, amely hatását tekintve ugyanaz.
Felhasználása előtt ne felejtsük el, hogy a tárban lévő programot törli és az összes változót nullázza. Nevezhetnénk ezt "hidegindítás"-nak, de valójában sem a SYS 64738-cal, sem pedig a COLD utasítással nem hajtunk végre valódi "hidegindítás"-t, hiszen ez alatt azt értjük, hogy a tároló minden címe nullázódik. Nos, itt erről szó sincs, hanem csak annyi történik, hogy a program kezdetét és végét jelző BASIC mutatók visszakerülnek eredeti állapotukba. Mivel nem törlődik az egész tár, inkább csak "melegindítás"-ról beszélhetünk. A Commodore 64-esnél valódi "hidegindítás"-t csak a gép ki- ill. bekapcsolása jelent.

3.1.5 OLD – a törölt program felélesztése

Formátum: OLD
Paraméterek: –
Funkció: Visszahozza (regenerálja) a NEW-val vagy COLD-dal törölt programot

Ha véletlenül beírtunk egy NEW- vagy COLD-parancsot, amelyek az egész BASIC programot törlik, újra felélleszthetjük azt egy OLD utasítással. Próbáljuk ki a következő programon:

```
 1 REM **********
 2 REM * OLD *
 3 REM **********
```

Írjuk be, majd NEW-val töröljük. LIST-tel győződjünk meg arról, hogy a törlés valóban megtörtént-e. Az OLD beírása után ismét listáztassuk. Láthatjuk, hogy a program újra a tárban van. A kísérletet a COLD-dal is megismételhetjük és tapasztalni fogjuk, hogy az OLD ugyanúgy működik, mint az előző esetben.

Figyelmeztetés! Az utasítás csak addig funkcionál, míg a NEW-t vagy COLD-ot követően nem vittünk be új BASIC sorokat, valamint módosítást, vagy törlést nem végeztünk.

3.1.6 Ellenőrző kérdések

1. Melyik utasítás aktivizálja a TRACE üzemmódot?

   a) TRACE
   b) TRACE 0
   c) TRACE 10
   d) RETRACE

2. Melyik utasítás hozza vissza az utolsó TRACE-ablakot?

   a) TRACE
   b) TRACE 0
   c) TRACE 10
   d) RETRACE

3. Mi a DUMP utasítás hatása?

   a) Kijelzi a programban előforduló változókat.
   b) Kijelzi a programban előforduló változók tartalmát.

4. Melyik utasítás jelenti a CBM 64 "hidegindítás"-át?

   a) COLD
   b) OLD
   c) NEW

5. Melyik utasítás hozza vissza a törölt BASIC programot?

   a) COLD
   b) OLD
   c) NEW

3.2 Belső hibakezelés

Belső hibakezelésre azért van szükség, hogy a program futása közben fellépő hibákat korrigálhassuk és ezzel a program összeomlását megakadályozzuk. Erre a SIMON's BASIC kényelmes megoldásokat kínál. Akár SYNTAX ERROR-ról, akár DEVICE NOT PRESENT ERROR-ról van szó, a hiba a programon belül kezelhető és egyedi hibaüzenet formájában kijelezhető.
A következő pontokban ezekről a hatékony módszerekről lesz szó.

3.2.1 ON ERROR – hibaellenőrzés a programon belül

Formátum: ON ERROR:GOTO uc
Paraméterek: uc – A GOTO utasítás ugrócíme
Funkció: A hibaellenőrzés áthárítása a programra
Példa: ON ERROR : GOTO 1000
Ha a programfutás közben valamilyen hiba felmerül, a program az 1000. sorban folytatódik.

Megjegyzés: Az utasítás a hiba számát az ERRN, a hibás sor számát pedig az ERRLN változókban tárolja.

Egy jól szervezett és jól megírt program semmilyen körülmények közt nem omolhat össze. Ez elérhető a felhasználó által beadott értékek rendszeres ellenőrzésével. Ha pl. le akarjuk hívni egy tömb 120. elemét, amelyet csak 100 elemre dimenzionáltunk, ezt a programnak fel kell ismernie és jeleznie kell, mert különben a program a "?BAD SUBSCRIPT ERROR IN..." hibaüzenettel befejeződik.
Nem minden hibát lehet kiküszöbölni a programból. Vegyük például, hogy adatokat akarunk rögzíteni a mágneslemezen, de nem csatlakoztattuk a meghajtóegységet. Az első adatátviteli kísérletkor meg fog jelenni a "?DEVICE NOT PRESENT ERROR IN..." hibaüzenet. Programtechnikailag lehetetlen megállapítani, hogy valamelyik készüléket csatlakoztattuk-e. Ebben és még sok más hibaforrás felderítésében is segítségünkre van a SIMON's BASIC.

A SIMON's BASIC által felismerhető hibákat és azok számait a következő táblázatban foglaltuk össze:

Hiba szám	Hiba	
1	TOO MANY FILES	(Túl sok file)
2	FILE OPEN	(A file nyitva van)
3	FILE NOT OPEN	(A file nincs nyitva)
4	FILE NOT FOUND	(A file nem található)
5	DEVICE NOT PRESENT	(A készülék nincs csatlakoztatva)
10	NEXT WITHOUT FOR	(NEXT, FOR nélkül)
11	SYNTAX	(Szintaktikai hiba)
12	RETURN WITHOUT GOSUB	(RETURN, GOSUB nélkül)
13	OUT OF DATA	(Nincs adat)
14	ILLEGAL QUANTITY	(Nem megengedett mennyiség)
15	OWERFLOW	(Túlcsordulás)
16	OUT OF MEMORY	(A tár betelt)
17	UNDEFINDED STATMENT	(Meghatározatlan állapot)
18	BAD SUBSCRIPT	(Rossz index)

19	RE-DIMENSIONED ARRAY	(Újra dimenzionált tömb)
20	DIVISION BY ZERO	(Osztás nullával)
21	ILLEGAL DIRECT	(Direkt üzemmód tilos)
22	TYPE MISMATCH	(Nem azonos típusú mennyiség)
23	STRING TOO LONG	(Túl hosszú füzér)

A meghajtóegység hibaüzenetei közül azonban egyik sem ismerhető fel az ON ERROR-ral. Ezeket az adott helyen a lemezegység-hibacsatorna kiolvasásával kell analizálni. Erről bővebbet a "AVC 1541-es lemezegység programozása*" című DATA BECKER-NOVOTRADE könyvben találhatunk.

Az ON ERROR utasítás:

Ezt az utasítást a programnak azon a részén helyezzük el, ahol a hibavizsgálatot a programnak át kell adni. Ha tehát el akarja végezni az egész program hibaellenőrzését, akkor az ON ERROR utasítást az első sorban kell elhelyezni. Egy program sem lehet azonban olyan bizonytalan, hogy ez szükséges lenne. Célszerű csak olyan hibák kiszűrésére alkalmazni, amelyek programtechnikailag nem elkerülhetők. A DEVICE NOT PRESENT hibaüzenet kiadását például a programozó nem tudja megakadályozni. Ilyenkor hasznos az ON ERROR utasítás alkalmazása, amely lehetővé teszi, hogy a felhasználó a programfutás megszakadása nélkül kijavíthassa a hibát.
Az ON ERROR alkalmazásakor kiértékelő rutinokat alkalmazunk, ahol a program a hiba felléptekor folytatódik, majd a hiba korrigálása után visszatér a főprogramba.

Pl.: 10 ON ERROR:GOTO 10000
(A kiértékelő rutin a 10000. sorban kezdődik.)

Az ON ERROR utasítás közvetlen üzemmódban nem használható, mivel ezáltal megszűnik a számítógép parancsellenőrző funkciója. Ilyenkor a számítógép a beírás után megakad és csak egy RUN/STOP-pal és az ezt követő NO ERROR utasítással hozható alapállapotba.
Az ON ERROR utasítás párja a NO ERROR, tehát most ezzel fogunk megismerkedni. Részletes tárgyalásukra a 3.2.3. pontban kerül sor.

3.2.2 NO ERROR – ON ERROR kikapcsolása

Formátum: NO ERROR
Paraméterek: –
Funkció: A hibaellenőrzést újra a standard BASIC veszi át

A kézikönyvben leírtaktól eltérően ez az utasítás nem elnyomja a hibaüzeneteket, hanem az ON ERROR-ral átvett hibaellenőrzést visszaadja a standard BASIC-nek. Az ON ERROR-ral végzett hibavizsgálat végén, vagy ha a programot RUN/STOP-pal megszakítottuk, mindig kapcsoljunk vissza a standard hibavizsgálatra a NO ERROR-ral, különben meglepő dolgokat fogunk tapasztalni. Előfordul, hogy ilyenkor a SIMON's BASIC teljesen "megbolondul".
Legcélszerűbb az ON ERROR-t a programon kívül hatástalanítani, nehogy az az értelmetlen helyzet álljon elő, hogy egy utasítás – pl. LIST – beírására a számítógép a programunk hibarutinjára ugrik és a közvetlen üzemmódot (READY) csak a RUN/STOP-pal állíthatjuk vissza.

* Eredeti címe: Das große Floppybuch (1983)

3.2.3 ON ERROR/NO ERROR – példák

Ebben a pontban részletesen ismertetjük, hogy hogyan kell alkalmazni a programban az ON ERROR/NO ERROR utasításokat, az ERRN és ERRLN változókat, milyen fontos szabályokat kell betartanunk, valamint, hogy a helytelen alkalmazásból milyen káros következmények származhatnak.

Igaz, hogy az ON ERROR-t csak ott kell használni, ahol egyéb hibakezelésre nincs mód, a következő példában mégis az egész program hibavizsgálatát ezzel végezzük, hogy minden lehetőségét bemutathassuk.

P. 11

```
1 REM **********************
2 REM * 1.O N  E R R O R *
3 REM **********************
40 ON ERROR:GOTO 10000
50 PRINT" TESZT KEZDETE "
60 PRINT:" START "
70 PRINT CHR$(256)
80 A=10:A$="20"
90 PRINT A+A$
100 X$(11)=" TESZT VEGE "
110 PRINT X$(11)
120 NO ERROR:END
10000 REM ******************
10010 REM * HIBAVIZSGALAT *
10020 REM ******************
10030 PRINT"A HIBA SZAMA:"ERRN "; A HIBAS SOR:"ERRLN
10040 PRINT"UGRAS A" ERRLN+10". SORBA"
10050 GOTO (ERRLN+10)
```

Ahhoz, hogy a teljes program hibavizsgálatát az ON ERROR-ral végezhessük, azt a program elején, esetünkben a 40. sorban kellett elhelyezni. Ennek a sornak a hatására minden hibafelismeréskor a program a 10000. sorba ugrik. A program többi része hibás sorokból, valamint a 10000. sorral kezdődő HIBAVIZSGALAT-alprogramból áll.

Indítsuk el a RUN-nal a programot:

A képernyőn megjelenik a következő kijelzés:

```
TESZT KEZDETE
A HIBA SZÁMA: 11; A HIBÁS SOR:60
UGRÁS A 70. SORBA
A HIBA SZÁMA: 14; A HIBÁS SOR: 70
UGRÁS A 80. SORBA
A HIBA SZÁMA: 22; A HIBÁS SOR: 90
UGRÁS A 100. SORBA:
A HIBA SZÁMA: 18; A HIBÁS SOR 100
UGRÁS A 110. SORBA
A HIBA SZÁMA: 18; A HIBÁS SOR: 110
UGRÁS A 120. SORBA
```

Vizsgáljuk most meg tüzetesen a programunkat.

A hibatáblázat alapján nézzük meg, melyek is a kijelzett hibák:

- A 60. sorban 11-es számú, azaz szintaktikai hiba van, amit az idézőjel előtt szereplő kettőspont okoz.
- A 70. sorban a 14-es számú, azaz nem megengedett mennyiség hiba jelentkezik: (CHR$(256)).
- A 90. sorban 22-es számú, azaz a nem azonos típusú változók okozta hibát találhatunk (A + A$).
- A 100. és 110. sor 18-as, azaz rossz index okozta a hibát: (X$(11)).

Nézzük meg, mivel értük el azt, hogy a program a hibák ellenére tovább futott, azaz a hibarutin feldolgozása után mindig visszatért a következő programsorra.

Látjuk, hogy a hiba száma és a hibás sor száma egy-egy változóba kerül (ERRN és ERRLN). A főprogramba való visszatéréshez az utóbbit használjuk fel. Mivel a sorszámozás szabályosan, 10-esével növekszik, lehetőségünk van arra, hogy a hibás sort követő sor számát kiszámítsuk, a következő egyszerű képlet szerint:

következő végrehajtandó sor = ERRLN + 10.

Mondhatnánk erre azt, hogy mindez szép, de a GOTO utasítás csak egy előre meghatározott sorszámot tud elfogadni, ilyen kiszámított ugrócímet nem. Szerencsére a SIMON's BASIC-ben erre is találunk megoldást, ez pedig a CGOTO-utasítás. Ez ugrócímként változót és matematikai kifejezést is megenged. Érthetővé vált tehát az 10050. sor tartalma, ami a hibás sor számából kiszámítja a következő végrehajtandó sor számát, és CGOTO-val odaküldi a programot ezzel a végrehajtás folytatódhat.

A programunk továbbfejlesztése az lehet, ha megoldjuk a hibák szöveges kiírását. Nem várhatjuk el ugyanis senkitől, hogy a hibatáblázatot fejben tartsa, enélkül viszont mindig vissza kell lapoznia. Ez nem elegáns megoldás, sokkal kedvezőbb, ha a képernyőn rögtön olvashatjuk a hiba jellegét is.
Nézzük a következő példát!

```
P. 12
1 REM ********************
2 REM * 2.0 N  E R R O R *
3 REM ********************
30 GOSUB 20000
40 ON ERROR:GOTO 10000
50 PRINT" TESZT KEZDETE "
60 PRINT:" START "
70 PRINT CHR$(256)
80 A=10:A$="20"
90 PRINT A+A$
100 X$(11)=" TESZT VEGE "
110 PRINT X$(11)
120 NO ERROR:END
10000 REM ********************
10010 REM * HIBAVIZSGALAT *
10020 REM ********************
10030 PRINT FE$(ERRN)" A"ERRLN".SORBAN"
10040 PRINT"UGRAS A" ERRLN+10".SORBA"
10050 CGOTO (ERRLN+10)
20000 REM ********************
20010 REM * HIBATABLAZAT *
20020 REM ********************
20030 DIM FE$(23)
20040 READ I,I$
20050 IF I=0 THEN RETURN
20060 FE$(I)=I$
20070 GOTO 20040
20080 DATA1,"TUL SOK A FILE"
20090 DATA2,"A FILE NYITVA VAN"
20100 DATA3,"A FILE NINCS NYITVA"
20110 DATA4,"A FILE NEM TALALHATO"
20120 DATA5,"A KESZULEK NINCS CSATLAKOZTATVA"
20130 DATA10,"NEXT, FOR NELKUL"
20140 DATA11,"SZINTAKTIKAI HIBA"
20150 DATA12,"RETURN, GOSUB NELKUL"
20160 DATA13,"KEVES AZ ADAT"
20170 DATA14,"ILLEGALIS MENNYISEG"
20180 DATA15,"TULCSORDULAS"
20190 DATA16,"KICSI A TARKAPACITAS"
20200 DATA17,"MEGHATAROZATLAN ALLAPOT"
20210 DATA18,"ROSSZ INDEX"
20220 DATA19,"A TOMB MAR DIMENZIONALT"
20230 DATA20,"OSZTAS 0-VAL"
```

```
20240 DATA21,"DIREKT UZEMMOD TILOS"
20250 DATA22,"A VALTOZO TIP. HIBAS"
20260 DATA23,"A STRING TUL HOSSZU"
20270 DATA0,"VEGE"
```

A 2.ON ERROR program tanulmányozása során meggyőződhetünk arról, hogy a hibaszámokhoz tetszőleges szöveg rendelhető, így részletesebb hibaüzeneteket is készíthetünk. Töltsük be tehát a programot és indítsuk el. Néhány szó a változásokról. A 30. sorban lehívunk egy alprogramot, amely a hibatáblázatot beolvassa egy tömbbe. Ez az FE$(23) tömb, amelynek indexei egyben a hibaszámok is. Tehát a 11-es számú hiba szövege az FE$(11) stringben van (20030 sor). A 20040 sor kiolvassa a DATA-sorokból a hibák számait és a hozzátartozó szövegeket. Ha a végére ért, azaz I=0, a 20050. sor utasítása szerint RETURN-nel visszatér a főprogramba.
A program futási eredménye a képernyőn a következőképpen fog megjelenni:

```
TESZT KEZDETE
SZINTAKTIKAI HIBA A 60. SORBAN
UGRÁS A 70. SORBA
ILLEGÁLIS MENNYISÉG A 80. SORBAN
UGRÁS A 90. SORBA
A VÁLTOZÓ TIP.HIBÁS A 90. SORBAN
UGRÁS A 100. SORBA
ROSSZ INDEX A 100. SORBAN
UGRÁS A 110. SORBA
ROSSZ INDEX A 110. SORBAN
UGRÁS A 120. SORBA
```

A programkészítés során komoly segítséget jelent egy ilyen hibafeldolgozási lehetőség. A megjelenítés a képernyőn gyakran szükségtelen, sőt zavaró is lehet. Sokkal jobb megoldás a hibaüzeneteket nyomtatón megjeleníteni, ill. file-ba "összegyűjteni", így a képernyő szabadon marad. Ehhez a következő módosításokat kell elvégeznünk.

Hibalista készítése a nyomtatón:
- A program elején megnyitjuk a nyomtatócsatornát:
  10 OPEN 1,4
- Módosítjuk a 10030. és 10040. sorokat:
  10030 PRINT#1, FE$ (ERRN)"A"ERRLN". SORBAN"
  10040 PRINT#1, "UGRÁS A"ERRLN+10".SORBA"
- A program végén lezárjuk a nyomtatócsatornát:
  120 NO ERROR:CLOSE 1: END

A hibák összegyűjtése egy file-ban:
- A program elején megnyitjuk a file-t írásra:
  10 OPEN 1,8,2"HIBA-FILE,S,W"
- Az előzőek szerint módosítjuk a 10030. és a 10040. sort
- Lezárjuk a file-t

A program futása után rendelkezésünkre áll egy lista a nyomtatón vagy a lemezen, a programban található hibákról. A file-ban tárolt hibalista a következő programmal íratható ki a képernyőre:

```
10 OPEN 1,8,2 "HIBA-FILE,S,R"
20 GET# 1,X$
30 PRINT X$;
40 IF ST < > 64 THEN 20
50 CLOSE 1
```

Ez a program megnyitja olvasásra a soros hibafile-t, majd byte-onként beolvassa és kiírja a képernyőre a tartalmát. A file végét a státuszváltozó (ST = 64) jelzi.
Megismerkedtünk tehát a professzionális hibaelemzés összes módszerével. Éljünk minél gyakrabban ezzel a SIMON's BASIC nyújtotta kitűnő lehetőséggel.*

3.2.4 OUT

Logikailag most az OUT utasítás bemutatása következne, de a birtokunkban lévő SIMON's BASIC lemezen ez nem működött. Emiatt olvasóink érdekében nem foglalkozunk vele. A kézikönyvben leírtak különben is teljesen ráillenek a NO ERROR-ra. Programban úgy működik, mint az END, közvetlen üzemmódban pedig a gép által utoljára kiadott üzenetet hozza vissza. Ha például a programindítás "?SYNTAX ERROR"-t eredményez, akkor az utána kiadott OUT is ezt jelzi ki. Ez az utasítás tehát használhatatlan!

* (Ebben a fejezetben nem ragaszkodtunk a könyv eredeti szerkezetéhez, amelyet a programok lemezrevitele indokol. – A magyar változat készítői.)

3.2.5 Ellenőrző kérdések

1. A hibaellenőrzést át kell adni a programnak. Ha a hibarutin az 1000. sorban kezdődik, melyik utasítással oldható ez meg?

   a) ON ERROR:GOTO 1000
   b) ON ERROR;GOTO 1000
   c) ON ERROR,GOTO 1000
   d) ON ERROR GOTO 1000

2. Az ON ERROR utasítás melyik változónak adja át a hibaszámot:

   a) ER
   b) EN
   c) ERN
   d) ERRN

3. Melyik változó veszi át a hibás sorszámot?

   a) ERRLINE
   b) ERLN
   c) ERRLN
   d) ERL

4. Melyik utasítás adja vissza a hibaellenőrzést a standard BASIC-nek?

   a) OUT ERROR
   b) NO ERROR
   c) OFF ERROR

5. A következő hibaüzenetek közül melyik az, amelyet ON ERROR-ral nem tudunk kijelezni?

   a) OUT OF MEMORY ERROR
   b) OUT OF DATA ERROR
   c) LOAD ERROR

4. FEJEZET
Programvédelem

4.1 DISAPA – a védeni kívánt sorok megjelölése

Formátum: DISAPA
Paraméterek: –
Funkció: Megjelöli a SECURE 0-val védeni kívánt sorokat
Minden olyan sort, amelyet SECURE 0-val védeni akarunk, DISAPA utasítással kell kezdeni.
Hívjuk be a lemezről a DISAPA-programot:

P. 13

```
1 REM ****************
2 REM *  D I S A P A *
3 REM ****************
10 INPUT"KEREM A PROGRAMKODOT";A$
20 IF A$="A12B13" THEN 40
30 PRINT"A KOD NEM HELYES!":GOTO 10
40 PRINT"A KOD HELYES!"
```

Tegyük fel, hogy a program 20. sorát védeni akarjuk, tehát a sorban az első utasítás a DISAPA lesz.

```
20 DISAPA:IF A$="A12B13" THEN 40
```

Ha most kilistázzuk a programot, láthatjuk, hogy a DISAPA utasítás a 20. sort négy kettősponttal megjelölte.

P. 14

```
1 REM ****************
2 REM *  D I S A P A *
3 REM ****************
10 INPUT"KEREM A PROGRAMKODOT";A$
20 DISAPA::::IF A$="A12B13" THEN 40
30 PRINT"A KOD NEM HELYES!":GOTO 10
40 PRINT"A KOD HELYES!"
```

A teljes program védelmét úgy valósíthatjuk meg, ha minden sor elejére kitesszük a DISAPA utasítást, de ez meglehetősen kényelmetlen megoldás. Rendeltetése nem is ez, hanem, hogy a fontosabb sorok, pl. amelyekben másolás elleni védelem van, a programlistában ne szerepeljenek. A megjelölt sorok a következő utasítással védhetők.

4.2 SECURE – a sorok védelme

Formátum: SECURE 0
Paraméterek: –
Funkció: A DISAPA-utasítással megjelölt sort védi

Ezt az utasítást mindig közvetlen üzemmódban használjuk, a programban DISAPA-val megjelölt sorok védelmére. Az előbb megjelöltük a DISAPA-program 20. sorát:

P. 15

```
1 REM ***************
2 REM *  D I S A P A *
3 REM ***************
10 INPUT"KEREM A PROGRAMKODOT";A$
20 DISAPA:::::IF A$="A12B13" THEN 40
30 PRINT"A KOD NEM HELYES!":GOTO 10
40 PRINT"A KOD HELYES!"
```

Írjuk be most a SECURE 0 utasítást és listáztassuk ki újból a programot. Azt tapasztaljuk, hogy a 20. sor tartalma nem jelenik meg a képernyőn.

P. 16

```
1 REM ***************
2 REM *  D I S A P A *
3 REM ***************
10 INPUT"KEREM A PROGRAMKODOT";A$
20
30 PRINT"A KOD NEM HELYES!":GOTO 10
40 PRINT"A KOD HELYES!"
```

A módszer egyetlen hátránya, ami ebben az esetben végül is előny, hogy a 20. sort ezután már nem hozhatjuk vissza. Igaz, hogy nem is volna értelme olyan védelemnek, amely bármikor feloldható. A védett sorokkal ezután már a törlésen kívül semmit sem csinálhatunk. Ajánlatos tehát a programokból védelem nélküli másolatot készíteni, hogy abba a későbbiek során még beavatkozhassunk.
Jogos a kérdés: hogy létezik az, hogy egy sor eltűnik, de mégis létezik? Nos, ez viszonylag egyszerű trükk. A DISAPA-sor elejére egy nulla-byte kerül, amelyet négy kettőspont követ. A nullabyte hatására a sor a listából kimarad, de a kettőspontok miatt mégis feldolgozásra kerül.
Bemutathatnánk azt a programot, amellyel az így láthatatlanná tett sorok visszahozhatók, de akkor ez az utasítás nem érne semmit. A probléma megoldását átengedjük inkább azoknak, akik szívesen törik ilyesmin a fejüket!

5. FEJEZET
A strukturált programozás

Egy program áttekinthetőségét nagymértékben befolyásolja annak szerkezeti felépítése. Természetesen a standard BASIC-ben is ajánlatos strukturált programozást megvalósítani. Célszerű például az alprogramokat (szubrutinokat) a program végén elhelyezni és REM-ekkel megjelölni, az egymásba skatulyázott programrészeket érdemes a sorok elején elhelyezett kettőspontokkal vagy szünetekkel kiemelni stb.
Az alábbi programrészlete egy FOR...NEXT ciklus kiemelését szemlélteti:

```
10 FOR I=1 TO 10
20 : : : :A=A+1
30 : : : :PRINT I,A
40 NEXT I
```

Egy strukturált IF-lekérdezés szintén áttekinthetőbb:

```
10 IF A$="NYOMTATÁS" THEN GO
20 : PRINT"NYOMTATÁS NINCS."
30 : PRINT"KIJELZÉS A KÉPERNYŐN (I/N)?";X$
40 : GET X$:IF X$ < >"I" AND X$ < >"N" THEN 40
50 : IF X$="I" THEN A$="KIJELZÉS"
60 GOSUB 10000
```

Sokkal könnyebben felismerhetjük, hogy az IF-lekérdezés NEM-ága mit jelent. A standard BASIC-ben a strukturált programozásra kevés lehetőségünk van, de a SIMON's BASIC e területen is számos megoldást kínál.

5.1 Logikai műveletek

5.1.1 IF...THEN...ELSE

Formátum	feltétel THEN IGEN-ág: ELSE:NEM-ág
Paraméterek:	feltétel – valamilyen logikai összehasonlítás
	igen-ág – ha a feltétel eredménye logikai IGAZ, a program futása itt folytatódik
	nem-ág – ha a feltétel eredménye logikai HAMIS, a program futása itt folytatódik
Példa:	IF X$="N"THEN PRINT"VÉGE":END:ELSE:GOTO 1000
	Azaz, ha X$ értéke "N", írja ki, hogy "VÉGE" és fejezze be a programot. Ellenkező esetben ugorjon az 1000. sorba.

Így az IF-feltétel eldöntését az IGEN- és NEM-ággal együtt egyetlen sorba sűríthetjük.
A következő két programrészleten összehasonlíthatjuk, hogy ugyanazt az IF-vizsgálatát hogyan oldhatjuk meg standard BASIC-ben és SIMON's BASIC-ben.
A feladat a következő: olvassuk be egy karaktert a billentyűzetről. Ha ez „D", hívjuk le az 1000. sorban, ha „B", a 2000. sorban kezdődő alprogramot.
Standard BASIC-ben ez így néz ki:

```
10 GET X$:IF X$ < > "D" AND X$ < > "B" THEN 10
20 IF X$="D" THEN GOSUB 1000
30 IF X$="B" THEN GOSUB 2000
40 ...
```

vagy

```
10 GET X$ : IF X$ < > "D" AND X$ < >"B" THEN 10
20 IF X$="D" THEN GOSUB 1000:GOTO 40
30 GOSUB 2000
40 ...
```

Sokkal elegánsabb a következő megoldás, amelyet a SIMON's BASIC tesz lehetővé

```
10 GET X$:IF X$ < > "D" AND X$ < > "B" THEN 10
20 IF X$="D" THEN GOSUB 1000:ELSE:GOSUB 2000
30 ...
```

Amikor csak tehetjük, alkalmazzuk ezt a módszert!

5.1.2 RCOMP

Formátum: RCOMP:IGEN-ág: ELSE:NEM-ág
Paraméterek: (az előző ponthoz hasonlóan)
Funkció: A legutolsó IF...THEN...ELSE feltétel lekérdezése
Példa: RCOMP END:ELSE:GOTO 1000

Ez az utasítás a legutóbbi IF...THEN...ELSE utasítás feltételére hivatkozik. Akkor alkalmazható előnyösen, ha egy programon belül egymás után többször is le kell kérdezni ugyanazt a feltételt. Így valamivel rövidebb formában ugyanazt érhetjük el.
A példában tehát: ha az előzőleg szereplő IF...THEN...ELSE feltételének eredménye most IGAZ, a program befejeződik, ha HAMIS, az 1000. sorban folytatódik.

Nézzük a következő példaprogramot:
Az ARTDAT nevű file-ban szereplő cikkek számát hozzá kell rendelnünk a C változóhoz. Azon termék szállítójának számát (sz), amelynek cikkszáma 999-nél nagyobb, a "CIKK.1000-" a többit a "CIKK.-999", majd ugyanezen feltétel szerint az árakat (PR) a "AR.1000-" és a "AR.-999" elnevezésű file-okban kell gyűjteni.

P. 17

```
100 REM FOPROGRAM
110 REM **********
120 OPEN 1,8,2,"ARTDAT,S,W"
130 INPUT#1,C,SZ,A
140 IF C>999 THEN GOSUB 1000:ELSE:GOSUB 2000
150 RCOMP GOSUB 3000:ELSE:GOSUB 4000
170 IF ST<>64 THEN 130
180 CLOSE 1:END
1000 REM IRAS A "CIKK 1000-" FILE-BA
1005 REM ****************************
1010 .......
1020 .......
1040 RETURN
2000 REM IRAS A "CIKK -999" FILE-BA
2005 REM ****************************
2010 .......
2020 .......
2030 RETURN
3000 REM IRAS AZ "AR 1000-" FILE-BA
3005 REM ****************************
3010 .......
3020 .......
3030 RETURN
4000 REM IRAS AZ "AR -999" FILE-BA
4005 ****************************
4010 ......
4020 ......
4030 RETURN
```

A problémát a következő programmal oldhatjuk meg:
Ha az RCOMP utasítás nem állna rendelkezésünkre, a 140. sorban meg kellene ismételnünk a lekérdezést.
Az RCOMP addig ismételhető, míg újabb IF...THEN...ELSE kapcsolat elő nem fordul.
Mivel az RCOMP az IF...THEN...ELSE-ből csak a "feltétel" paramétert használja fel, az IGEN- és NEM-ág utasításai eltérőek is lehetnek.
A program áttekinthetőségét rontja, ha az IF...THEN...ELSE és a hozzátartozó RCOMP túl messze esik egymástól, ezért ezt az eljárást csak akkor érdemes alkalmazni, ha megoldható, hogy ezek viszonylag közel kerüljenek egymáshoz.

5.1.3 REPEAT...UNTIL – ciklusvezérlés

Formátum: REPEAT: ciklus: UNTIL feltétel
Paraméterek: ciklus – egy ciklus utasításai
feltétel – a ciklus végének feltétele
Funkció: Egy ciklus vezérlése valamilyen feltétellel
Példa: REPEAT:A = A + 1:UNTIL A = 10000
"A" értékét mindaddig növeli 1-gyel, míg el nem éri a 10 000-et.

Az utasításpár alkalmazásával áttekinthetőbb ciklusokat szervezhetünk, mint ahogy az a következő példából is látható:

```
P. 18
1 REM ***************
2 REM * R E P E A T *
3 REM ***************
10 A=10
20 REPEAT
30 : : : : S=S+A
40 : : : : A=A+1
50 UNTIL A>100
60 PRINT S
```

A feladat a 10 és 100 közötti számok összeadása. A végeredmény (5005) megjelenik a képernyőn.
Az 50. sorban a vizsgálatot A > 100-ra végezzük, mivel A értéke mindig csak az összedás után nő meg, tehát az utolsóként elért 100-at is hozzá kell adni.
Példánk nem a legszerencsésebb, mert ez a feladat a hagyományos FOR...NEXT ciklussal sokkal egyszerűbben megoldható:

```
10 FOR A = 10 TO 100
20 : :S = S + A
30 NEXT
40 PRINT S
```

A REPEAT...UNTIL jelentősége nem is itt, hanem a bizonytalan végű ciklusoknál mutatkozik meg. A FOR...NEXT ciklus végét ugyanis csak előre meghatározottan, tehát nem a cikluson belül létrejött változások függvényeként adhatjuk meg. Ezzel szemben a REPEAT...UNTIL esetében a zárófeltétel tetszőleges, a ciklusban előálló feltétel lehet. Tulajdonképpen nem is hasonlítható össze a FOR...NEXT-tel, mert más a rendeltetése. A különbséget egy egyszerű példán szemléltethetjük:
Egy szekvenciális file-t szeretnénk olvasni, amelynek végét az ST státuszváltozó értékéről (64) ismerjük fel. Mivel a file vége bizonytalan, a FOR...NEXT ciklus nem alkalmazható. Standard BASIC-ben így oldhatnánk meg:

```
10 OPEN 1,8,2,"ADATOK,S,R"
20 I = 0
30 : : : :I = I + 1
40 : : : :INPUT#1,A$(I)
50 IF ST< >64 THEN GOTO 30
60 CLOSE 1:END
```

A ciklust itt "kézzel" építettük fel és az IF-lekérdezésével lépünk ki belőle. REPEAT...UNTIL-lal a problémát sokkal elegánsabban is megoldhatjuk:

```
10 OPEN 1,8,2,"ADATOK,S,R"
20 I=0
30 REPEAT
40 : : : : I=I+1
50 : : : : INPUT#1,A$(I)
60 UNTIL ST < > 64
70 CLOSE 1
```

Igaz, hogy eggyel több sort használtunk fel, de kiküszöböltünk egy GOTO utasítást, amellyel jelentősen hozzájárultunk a strukturált programozás megvalósításához

5.1.4 LOOP...EXIT IF...END LOOP – ciklusvezérlés

Formátum: LOOP:Ciklus:EXIT IF feltétel:END LOOP
Paraméterek: ciklus – egy ciklus utasításai
feltétel – a ciklus zárófeltétele
Funkció: Egy ciklus vezérlése valamilyen feltétellel
Példa: LOOP:A=A+1:EXIT IF A=1000:END LOOP

Első ránézésre a ciklus logikája hasonlít a REPEAT...UNTIL utasításra, de azzal szemben néhány előnyös tulajdonsággal rendelkezik. Ezek a következők:
– A ciklus megszakításának feltétele a ciklusutasítások között szerepelhet:

```
10 LOOP
20 A=A+1
30 EXIT IF A> 20
40 B=B+A
50 END LOOP
```

– Tetszőleges helyeken egyszerre több feltételhez is köthetjük a ciklus megszakítását:

```
10 LOOP
20 A=A+1
30 EXIT IF A>20
40 B=B+A
50 EXIT IF B>100
60 END LOOP
```

Térjünk vissza mégegyszer a strukturált programozáshoz. Az utolsó példánk standard BASIC-ben a következőképpen nézne ki:

```
10 A=A+1
20 IF A > 20 THEN 50
30 B=B+A
40 IF B > 100 THEN 50
50 ...
```

Ismét két GOTO-utasítást kellett alkalmaznunk, ami bizony nem a strukturált programozást szolgálja.
Még egy példán bemutatjuk a LOOP alkalmazását. A feladat a következő: Kérdezzük le a billentyűzetről, hogy szükség van-e kinyomtatásra. Ha igen, hívjuk le az 1000. sorral kezdődő alprogramot, ha nem, folytassuk a főprogramot.
Standard BASIC-ben így tudjuk megoldani:

```
10 PRINT"NYOMTATÁS (I/N)?"
20 GET X$:IF X$ < >"I" AND X$ < >"N" THEN 20
30 IF X$ = "I" THEN GOSUB 1000
40 ...
```

Most pedig SIMON's BASIC-ben:

```
10 PRINT"NYOMTATÁS (I/N)?"
20 LOOP
30 GET X$
40 EXIT IF X$="I"
50 EXIT IF X$="N"
60 END LOOP
70 IF X$="I" THEN GOSUB 1000
80 ...
```

Igaz, hogy itt sokkal több sort alkalmaztunk, de sokkal áttekinthetőbbé vált a program. Kiesett egy GOTO, ami szintén előnyére vált a programunknak, hiszen a sorok közti „vad ugrálás" nem célja a strukturált programozásnak.

5.1.5 Ellenőrző kérdések

1. Az alábbi IF...THEN...ELSE-változatok közül melyik a helyes?

   a) IF A=1 THEN 100:ELSE GOTO 200
   b) IF A=1 THEN 100:ELSE:GOTO 200
   c) IF A=1 THEN 100 ELSE GOTO 200
   d) IF A=1 THEN 100 ELSE:GOTO 200

2. Mire vonatkozik az RCOMP utasítás?

   a) Az utolsó IF-lekérdezésre
   b) Az utolsó IF...THEN...ELSE-lekérdezésre
   c) Az első IF-lekérdezésre
   d) Az első IF...THEN...ELSE-lekérdezésre

3. Melyik REPEAT-változat a helyes?

   a) REPEAT:GETX$:UNTIL X$ < > " "
   b) REPEAT GETX$ UNTIL X$ < > " "
   c) REPEAT GET X$:UNTIL X$ < >" "

4. Melyik LOOP-változat helytelen?

   a) LOOP:GET X$:EXIT IF X$="J":EXIT IF X$="N" END LOOP
   b) LOOP:A=A+1:B=B+A:EXIT IF A > 10 OR B > 20:END LOOP
   c) LOOP:EXIT IF I=10:I=I+1:READ A(I):END LOOP

5. Milyen számokat fog az alábbi program kijelezni?
   A=10:LOOP:EXIT IF A > 20:A=A+1:PRINT A:END LOOP

   a) 10 – 20
   b) 11 – 20
   c) 10 – 21
   d) 11 – 21

5.2 Ugrások megvalósítása

5.2.1 PROC – szimbolikus cím

Formátum: PROC címke
Paraméterek: címke – címke azaz más néven egy szimbolikus cím (ang. label)
Funkció: A szimbolikus ugrási cím kiadása
Példa: 100 PROC BEVITELI ALPROGRAM

Ez azt jelenti, hogy a 100. sorral kezdődő alprogramra ezentúl a BEVITELI ALPROGRAM szimbolikus címmel hivatkozhatunk.
Erre a pontra már a RENUMBER tárgyalásakor is utaltunk, amikor is felmerült az a probléma, hogy ez az utasítás a GOTO és a GOSUB ugrócímeit nem módosítja az átsorszámozott programnak megfelelően. Az itt bemutatott utasításhoz nem sorszámot, hanem szimbolikus címet kapcsolhatunk, így az átsorszámozás nem fog gondot okozni. A szimbolikus címzést (létrehozását) a PROC-utasítással végezhetjük, minden gond nélkül. Ha egy sorra szimbolikus címmel akarunk hivatkozni, egy PROC-ot tartalmazó sort kell elhelyezni, például így:

```
100 PROC NÉV BEADÁSA
110 INPUT"KÉREM A NEVET:";X$
```

Ha most a 110. sorra akarunk ugrani, nem a GOTO 110 utasítást, hanem a CALL NÉV BEADÁSA utasítást alkalmazhatjuk. A CALL-t később ismertetjük!

Figyelmeztetés! A PROC-ot tartalmazó sorban más utasítás nem szerepelhet!

5.2.2 END PROC – az alprogram vége

Formátum: END PROC
Paraméterek: –
Funkció: Az alprogram végének meghatározása
Megjegyzés: Rendeltetése a RETURN-éhez hasonló

Az előzőekben ismertettük a PROC-utasítást, amellyel szimbolikus ugrási címeket adhatunk meg. Az END PROC-cal kiegészítve lehetőségünk van teljes alprogramok meghatározására, amelyeket PROC-cal nyitunk és END PROC-cal zárunk.

```
100 PROC KIVITELI ALPROGRAM
110 : : : :  PRINT"VEZETÉKNÉV:        "V$
120 : : : :  PRINT"KERESZTNÉV:        "K$
130 : : : :  PRINT"UTCA               "U$
140 : : : :  PRINT"VÁROS              "V$
150 END PROC
```

Ez az alprogram a KIVITELI ALPROGRAM nevet viseli és az EXEC KIVITELI ALPROGRAM-utasítással hívható be. Az END PROC a RETURN-hoz hasonlóan az alprogram végén visszaküldi az interpretert a főprogramba.

5.2.3 CALL – ugrás a szimbolikus címre

Formátum: CALL címke
Paraméterek: címke – vagy más néven szimbolikus ugrási cím
Funkció: A PROC-cal definiált szimbolikus cím behívása
Példa: CALL BEVITEL
A program a BEVITEL elnevezésű szimbolikus címre ugrik.

Megjegyzés: Működése a GOTO-éhoz hasonló. Így például egy GOTO 100 utasítás átírható CALL 1. CIM-re.
Egy ilyen címkékkel ellátott programrészt mutatunk be a következő listán:

```
100 PROC GET
110 : : : : GET X$:IF X$="" THEN CALL GET
120 : : : : IF X$="A" THEN CALL INPUT A:ELSE:CALL GET
130 PROC INPUT A
140 ....
```

Megint sikerült kiküszöbölnünk néhány GOTO-t. A SIMON's BASIC RENUMBER utasításának hátrányai megint csökkentek.

5.2.4 EXEC – ugrás egy alprogramba

Formátum: EXEC címke
Paraméterek: (mint az előzőekben)
Funkció: Ugrás a PROC...END PROC-cal meghatározott alprogramba
Példa: EXEC BEVITELI ALPROGRAM
Megjegyzés: Működése a GOSUB-éhoz hasonló.
Minden nagyobb program megköveteli a szubrutinok alkalmazását. A strukturált programnyelvek, mint a COBOL vagy a PASCAL, szinte teljes egészében szubrutinokból állnak, melyeket a főprogram vezérel. Erre a SIMON's BASIC is lehetőséget nyújt. A következő példán egy jól szervezett, strukturált programot láthatunk:

```
110 EXEC OPEN-FILE
120 PROC START
130 : : : :EXEC BEVITELI MASZK KIJELZÉSE
140 : : : :EXEC BEVITELI MASZK KITÖLTÉSE
150 : : : :EXEC REKORD ÍRÁSA
160 : : : :PRINT"VAN MÉG ADAT (I/N)?"
170 : : : :EXEC PROC GET I/N
180 : : : : IF AN$="I" THEN CALL START
190 CALL CLOSE-FILE
200 EXEC PROGRAM VÉGE
```

Ez egy adatrögzítő program, amelyhez már csak a megfelelő szubrutinokat kell kidolgoznunk. Lehet, hogy egy kicsit szokatlan ez a programozási stílus, de érdemes elsajátítani. A szabály az, hogy először mindig a főprogramot kell kidolgozni, csak ezután következhetnek az alprogramok. A strukturált programozás megkönnyíti a hibakeresést is.
Ismerkedjünk most meg két hibaüzenettel, amelyekkel a szimbolikus programozás során találkozhatunk.

PROC NOT FOUND
– A CALL-lal vagy EXEC-kel megadott cím nem található.
END PROC WITHOUT EXEC
– A programban EXEC nélküli END PROC-utasítás található.

Az 5.2.1–5.2.4 pontban tárgyalt utasítások alkalmazását a lemezen található "2.PLACE" című programon is tanulmányozhatjuk (8.2 fejezet).

5.2.5 CGOTO – számított ugrási cím

Formátum: CGOTO aritmetikai kifejezés
Paraméterek: Aritmetikai kifejezés – annak a sorszámnak a kiszámítási módja, amelyre a programot küldeni akarjuk
Funkció: A GOTO-utasítás olyan változata, amely számítással meghatározott ugrás címet is elfogad
Példa: CGOTO(N+I*2)

Mikor a programfeldolgozás idáig ér, kiszámítja a változók pillanatnyi értékei alapján az N+1*2 értéket, amely a következő végrehajtandó sor száma lesz.
Az On ERROR-nál már utaltunk erre az utasításra, amikor az ugrási címet a hibás sor száma és a lépték összege adta. A CGOTO nélkül ilyen kiszámított ugrási címet nem alkalmazhatnánk.
A CGOTO-t felhasználhatjuk például a menüből történő választás és az ezt követő programelágaztatás létrehozására.

```
 10 PRINT"MENÜ"
 20 PRINT "----"
 30 PRINT"1-ADATRÖGZITÉS"
 40 PRINT"2-ADATOK MÓDOSITÁSA"
 50 PRINT"3-ADATOK TÖRLÉSE"
 60 PRINT"4-ADATOK KINYOMTATÁSA"
100 PROC GET
110 ::::GET X$:IF X$ < "1" OR X$ > "4" THEN CALL GET
120 Z=VAL(X$)
130 CGOTO Z*10+130
140 CALL ADATRÖGZITÉS
150 CALL ADATOK MÓDOSÍTÁSA
160 CALL ADATOK TÖRLÉSE
170 CALL ADATOK KINYOMTATÁSA
```

A menüből történő választás az 1–4 értékek beírásával, és a CGOTO utasításban kiszámított címre történő elágaztatással történik. Ezen kívül a CGOTO még sok egyéb helyen használható a programozó fantáziája szerint.

5.2.6 Ellenőrző kérdések

1. Milyen előnyökkel rendelkezik a szimbolikus címzés?

   a) Kiküszöbölhetjük a GOTO és GOSUB utasításokat.
   b) Áttekinthetőbb lesz a program.
   c) Kiküszöbölhetjük az alprogramokat.

2. Melyik utasítással helyettesíthetjük a standard GOTO-t?

   a) CALL
   b) EXEC

3. Melyik utasítással helyettesíthetjük a standard GOSUB-ot?

   a) CALL
   b) EXEC

4. Írhatunk-e egy PROC-sorba más utasítást is?

   a) Igen
   b) Nem

5. Melyik utasítással fejezzük be az EXEC-kel lehívott alprogramot?

   a) RETURN
   b) END PROC
   c) PROC

6. Melyik a következő végrehajtandó sor a CGOTO utasítás szerint?

```
10 A=9
20 CGOTO(A+1)*10+130
```

   a) 1310
   b) 210
   c) 230
   d) 220

6. FEJEZET
Változók

Standard BASIC-ben a változók mindig egy meghatározott értékkel rendelkeznek. Ha ezt megváltoztatjuk és az eredeti értéket nem rögzítettük, akkor az elveszik, vissza nem hozható.
A SIMON's BASIC ebben a témakörben is újat tud nyújtani. Ismerkedjünk meg két új fogalommal: globális és lokális érték. Mit is jelentenek ezek? A globális érték a változónak a teljes programfutás során érvényes értéke, amelyet a szokásos módon definiálunk. Ezzel szemben a lokális érték mindig csak egy programszakaszban érvényes. Ha már nincs rá szükség, a változó újra felveszi a globális értéket, ami ezúttal nem veszett el, visszahozható anélkül, hogy azt előzőleg tároltuk volna. Tehát minden változónak két, egy globális és egy lokális értéke lehet.

6.1 LOCAL – a változók lokális értéke

Formátum: LOCAL 1. változó, 2. változó, 3. változó...stb.
Paraméterek: Változók – a programban előforduló, tetszőleges számú és típusú változó (pl. A, A$, A%)
Funkció: A lokális változók meghatározása, kijelölése
Példa: *Példa:* LOCAL A$, X$, B

Az A$, X$ és B változókat lokális változóként jelöli ki, ami azt jelenti, hogy a programban lokális értékeket rendelhetünk hozzájuk.
Hosszabb programokban sokszor annyi változóra van szükségünk, hogy végül könnyen áttekinthetetlenné válik a program. A változók nagy száma a LOCAL utasítással jelentősen csökkenthető, mert lehetőséget ad arra, hogy egy változót a program különböző részeiben alkalmazzunk. Mielőtt a változókat módosítanánk, azaz lokális értéket rendelnénk hozzájuk, lokális változóként kell őket definiálnunk.

P. 19

```
1 REM ************
2 REM * L O C A L *
3 REM ************
100 REM ******************************
110 REM GLOBALIS ERTEK MEGHATAROZASA
120 REM ******************************
130 A=120:B%=850:X$="GLOBALIS"
140 PRINT"GLOBALIS ERTEK:";A;B%;X$
150 REM ******************************
160 REM LOKALIS ERTEK MEGHATAROZASA
170 REM ******************************
180 LOCAL A,B%,X$
190 A=240:B%=1700:X$="LOKALIS"
200 PRINT"LOKALIS ERTEK:";A;B%;X$
```

A 130. sorban hozzárendeljük a változókhoz a globális értéküket. A 180. sorban ezeket a változókat lokálisként definiáljuk, majd a 190. sorban az értéküket megváltoztatjuk. A LOCAL utasítás futás közben tárolja a globális értékeket, így azok visszahívhatók.

6.2 GLOBÁL – a változók globális értéke

Formátum: GLOBAL
Paraméterek: –
Funkció: A változók eredeti értékeinek visszaállítása

Az előző példa globális értékeinek visszaírásához írjuk a következő sorokat a programhoz:

```
170 REM *GLOBÁLIS ÉRTÉK VISSZAÁLLÍTÁSA*
180 GLOBAL
190 PRINT"GLOBÁLIS ÉRTÉK:"A;B%;X$
```

A 180. sorban tehát a változók visszakapják eredeti értéküket. Nem vesztek el azonban a lokális értékek sem, hiszen egy újabb LOCAL-utasítással ismét átadhatók a változónak. Írjuk be például közvetlen üzemmódban a LOCAL A utasítást és PRINT A-val nézzük meg az eredményt. Láthatjuk, hogy az megint 240. Ilyen módon tetszés szerint kapcsolhatunk globálisról lokális értékre és vissza.
A két programot együtt rögzítettük GLOBAL néven. Töltsük be és próbáljuk ki a működését:

P. 20

```
1 REM ***************
2 REM * G L O B A L *
3 REM ***************
100 REM *GLOBALIS ERTEK*
110 A=120:B%=850:X$="GLOBALIS"
120 PRINT"GLOBALIS ERTEK:"A;B%;X$
130 REM *LOKALIS ERTEK*
140 LOCAL A,B%,X$
150 A=240:B%=1700:X$="LOKALIS"
160 PRINT"LOKALIS ERTEK :"A;B%;X$
170 REM *GLOBALIS ERTEK VISSZAALLITASA*
180 GLOBAL
190 PRINT"GLOBALIS ERTEK:"A;B%;X$
200 LOCAL A
210 PRINT A
```

6.3 Ellenőrző kérdések

1. Mi a LOCAL-utasítás feladata?

   a) A globális értékek törlése és a lokális változók meghatározása.
   b) A globális értékek közbenső tárolása és a lokális változók meghatározása.
   c) A globális értékek közbenső tárolása és a lokális értékek hozzárendelése a változókhoz.
   d) A globális értékek törlése és a lokális értékek hozzárendelése a változókhoz.

2. Mi a GLOBAL-utasítás feladata?

   a) A lokális értékek törlése és a globális értékek meghatározása.
   b) A lokális értékek közbenső tárolása és a változók globális értékeinek visszaállítása.
   c) A lokális értékek közbenső tárolása és a globális értékek meghatározása.
   d) A lokális értékek törlése és a globális értékek visszaállítása.

3. Milyen változók vehetnek fel lokális értéket?

   a) Egész típusú változók (pl. A%).
   b) Fűzér típusú változók (pl. A$).
   c) Valós típusú változók (pl. A).

7. FEJEZET
Matematikai eljárások

7.1 Aritmetikai műveletek

7.1.1 MOD – osztás maradéka

Formátum: MOD (X,Y)
Paraméterek: X – osztandó
Y – osztó
Funkció: Megadja az egész számmal (integer) való osztás maradékát
Példa: PRINT MOD (14,3)
A képernyőre kiírja a 14:3 kifejezés maradékát, azaz 2-t.

Ez az utasítás megadja az osztás egész számú maradékát. Mik is azok az integer számok? Minden egész, azaz tizedesjegyeket nem tartalmazó szám.
A CBM64 esetében csak 2-byte-os integer számok léteznek. Ezt egyrészt a tárral történő takarékosság, másrészt a megfelelő számítási pontosság indokolja. Egy 2-byte-os bináris szám maximális értéke 65 535 ($2^{16}-1$) lehet. A Commodore 64-es minden főtárcíme is ilyen 2-byte-os szám.
A BASIC integer változói negatív értékeket is felvehetnek. Mivel az előjelet a bal szélső bit tárolja, a szám ábrázolására csak a fennmaradó 15 bit vehető igénybe. Emiatt az egész típusú változók értéke csak −32768 és +32767 közötti lehet. Ha a hozzárendelt érték ezen a tartományon kívülre esik (pl. A% = 353000), megjelenik az

```
ILLEGAL QUANTITY ERROR
```

hibaüzenet.
A MOD utasítás csak a 0 és 65535 közé eső pozitív egész számokat fogadja el. Minden más esetben az előbbi hibaüzenetet kapjuk.
A MOD programban is, és közvetlen üzemmódban is alkalmazható. Nézzünk példát mindkét esetre:

```
PRINT MOD(144,7)
```

A képernyőn megjelenik az eredmény (4).
Egy kis program:

```
10 INPUT"OSZTANDÓ:     ";X
20 INPUT"OSZTÓ    :    ";Y
30 PRINT"MARADÉK :     ";MOD(X,Y)
```

Az osztó és az osztandó, tehát változóként is megadható.
A Commodore 64-es integer változói az osztás után csak a tizedesvessző előtti értéket tárolják, a maradékot a MOD-dal kell megadnunk. Egy példa:

```
10 A% = 50/12
20 A = 50/12
30 PRINT"A% EREDMÉNY = "A%
40 PRINT"A EREDMÉNY = "A
```

Az A% csak az osztás egész értékét, míg az A a helyes eredményt tartalmazza. Ilyenkor MOD-dal könnyen megkaphatjuk az osztási maradékot.

```
10 A% = 50/12
20 B% = MOD(50,12)
30 PRINT"EREDMÉNY:"A%"MARADÉK:"B%
```

7.1.2 DIV – osztás eredményének egész része

Formátum: DIV (X,Y)
Paraméterek: X – osztandó
Y – osztó
Funkció: Megadja az osztás eredményének egész részét
Példa: PRINT DIV (14,3)
A képernyőn eredményként a 4 jelenik meg.

A MOD és a DIV utasítások kombinálásával két integer szám egész számú hányadosát és a maradékot is meghatározhatjuk.

P. 21

```
1 REM ***********
2 REM * 1.D I V *
3 REM ***********
10 INPUT"OSZTANDO: ";X
20 INPUT"OSZTO   : ";Y
30 A%=X/Y
40 B%=MOD(X,Y)
50 PRINT" EREDMENY: ";A%
60 PRINT" MARADEK : ";B%
70 REM *DIV-VEL*
80 PRINT"■PROBALJA KI A DIV-VEL IS,";
90 PRINT"NYOMJON LE EGY BILLENTYUT!"
100 GET A$:IF A$="" THEN 100
110 INPUT"■OSZTANDO: ";V
120 INPUT"OSZTO   : ";Z
130 D=DIV(X,Y):M=MOD(X,Y)
140 PRINT" EREDMENY: ";D
150 PRINT" MARADEK : ";M
160 PRINT"■■HASONLITSA OSSZE A PROGRAM 10-60 ES 110-150 SORAIT!"
```

Alkalmazásuk tehát nagyon egyszerű és rendkívül jól használhatók. Ezt bizonyítja a következő példaprogramunk:
A 7.2. pontban egy olyan utasítást fogunk megismerni, amellyel a bináris számokat decimálissá alakíthatjuk. A fordított átalakítás azonban csak programmal lehetséges, mint például a következő, amely a MOD és DIV utasítások felhasználásával, a "maradékérték módszer"-rel dolgozik:

P. 22

```
1 REM ***********
2 REM * 2.D I V *
3 REM ***********
100 PROC BEVITEL
110 ::INPUT"KEREK EGY DECIMALIS SZAMOT 0 ES 65535 KOZOTT!";D
120 ::IF D>=0 AND D<=65535 THEN CALL KIVITEL
130 ::PRINT CHR$(145):;CALL BEVITEL
140 PROC KIVITEL
150 ::PRINT"BINARIS ERTEKE: ";
160 ::FOR I=1 TO 16
170 :::R=MOD(D,2):D=DIV(D,2)
180 :::PRINTTAB(16-I);R;CHR$(145)
190 ::NEXT I
```

Indítsuk el a programot és próbáljuk ki a működését. Egy megfelelő számot beírva megjelenik a 16-bites, bináris megfelelője.

7.1.3 FRAC – tört rész kijelzése

Formátum: FRAC (n)
Paraméterek: n – valós típusú szám, vagy kifejezés
Funkció: Valós szám tört részének kijelzése

Példa: PRINT FRAC (10/3)
Kiírja a hányados tizedesvessző utáni értékét, azaz 333333333-at.

A standard BASIC-ben mindez meglehetősen primitív módon oldható meg. A hányadosból le kell vonni annak egész értékét, ami feltételez néhány számátalakítást. SIMON's BASIC-ben mindez egyetlen rövid sorral elintézhető.
Előző példánk decimális számokat binárissá alakított. Nézzünk most egy programot, amely decimálisról hexadecimálisra alakítja át a számokat. Töltsük be a lemezről a FRAC című programot.

P. 23

```
1 REM ***********
2 REM * F R A C *
3 REM ***********
100 DIM A$(15)
110 FOR I=0 TO 15
120 :READ A$(I)
130 NEXT I
140 DATA0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
150 PROC BEVITEL
160 :INPUT"DECIMALIS SZAM 0-65535";D
170 :IF D>=0 AND D<=65535 THEN CALL KIVITEL
180 :PRINTCHR$(145):CALL BEVITEL
190 PROC KIVITEL
200 :PRINT:PRINT"HEXADECIMALIS ERTEKE:"
210 :FOR I=1 TO 4
220 ::R=16*FRAC(D/16):D=INT(D/16)
230 ::PRINTTAB(4-I);A$(R);CHR$(145)
240 :NEXT I
```

Indítsuk el és próbáljuk ki, hogyan működik. 0 és 65535 közti értéket beírva a képernyőn megjelenik annak négyjegyű hexadecimális megfelelője.

7.1.4 EXOR – exkluzív-VAGY kapcsolat

Formátum: EXOR (n, n1)
Paraméterek: n, n1 – összekapcsolandó számok 0 és 65535 között
Funkció: Két szám összekapcsolása exklusiv (kizáró) VAGY-gyal
Példa: PRINT EXOR(255,1)
Átkapcsolja a 255-ös érték 0. bitjét 0-ra.

Valószínűleg nem mindenki tudja, hogy mi is az a BOOLE algebra.
A standard BASIC ennek három kapcsolatát ismeri: OR, AND, NOT. Ezeket a logikai kapcsolatokat használjuk az IF lekérdezésekben, továbbá ezekkel kezelhetők meghatározott tárrekeszek is.
Az utóbbi esetben a logikai kapcsolatok hatása a következő:

OR – a biteket bekapcsolja
AND – a biteket kikapcsolja

A Commodore 64-esben több különböző rendeltetésű regiszter van (hang, grafika stb.), amelyeket ezekkel az utasításokkal programozhatunk. Minden regiszter 1 byte (8 bit) hosszúságú és minden bit be- ill, kikapcsolható a logikai operátorokkal.
Ez a következők szerint történik:
Először is ismernünk kell egy byte felépítését:

	0	0	0	0	0	0	0	0
bit	7	6	5	4	3	2	1	0
értéke	128	64	32	16	8	4	2	1

Egy byte tehát 8 bitből áll, amelyek mindegyike sorszámmal és értékkel rendelkezik. Ha egy bitet be akarunk kapcsolni, akkor az adott rekesz és a bit értéke között VAGY (OR) kapcsolatot kell létrehozni. Pl. az 53248-as tárolórekesz 4. bitjét így kapcsoljuk be:

```
POKE 53248, PEEK (53248) OR 16
```

A logikai kapcsolatot valójában a rekesz tartalma és a 4. bit értéke (16) között hoztuk létre.
Még egy példa: Ugyanezen byte 0, 1, 2 és 3-as bitjét 1-re akarjuk váltani: Ez így történik:

```
POKE 53248, PEEK (53248) OR 15
```

Kicsit nehezebb dolog a bitek kikapcsolása, mert ebben az esetben nem a bit értékét, hanem annak 255-ből kivont értékét használjuk fel.
Így a 0. bit kikapcsolásához az ÉS (AND) jobb oldalán a (255–1) kifejezésnek kell állnia. Az előbb bekapcsolt bitek a következőképpen kapcsolhatók ki:

```
POKE 53248, PEEK(53248) AND (255–16)
POKE 53248, PEEK(53248) AND (255–15)
```

vagy egyetlen utasítással:

```
POKE 53248, PEEK(53248) AND (255–31)
```

A SIMONS's BASIC egy újabb kapcsolat megteremtésével egyszerűsíti a bitek ki- és bekapcsolását. Ez az EXOR, amely a bitek átkapcsolását eredményezi. Vagyis, ha egy bit eddig 0 volt, az EXOR kapcsolat után 1 lesz és fordítva.
Példa: Kapcsoljuk át az 53 250-es tárolórekesz 7. bitjét.

```
POKE 53250, EXOR (PEEK (53250), 128)
```

Az írásmód egy kicsit megváltozott, mivel az EXOR írásmódja (EXOR (n,n1) eltér az OR és az AND írásmódjától (n OR/AND n1). Itt talán a SIMONS's BASIC-nek kellett volna alkalmazkodnia.
Végül nézzük a következő példát:

P. 24

```
1 REM ***********
2 REM * E X O R *
3 REM ***********
100 REM
110 PRINT"■"
120 CENTRE"A FELIRAT NEHANY PILLANAT":PRINT
130 CENTRE"MULVA INVERZ ALAKBAN LESZ":PRINT
140 CENTRE"LATHATO !"
150 PAUSE 5
160 FOR X=0 TO 210
170 :A=PEEK(1024+X)
180 :IF A=32 THEN 200
190 :Z=EXOR(A,%10000000)
200 :POKE1024+X,Z
210 NEXT X
220 END
```

7.1.5 Ellenőrző kérdések

1. Mit eredményez a PRINT MOD (33,7) utasítás?

   a) 4
   b) 5
   c) 0
   d) 28

2. Mit eredményez a PRINT DIV (33,7) utasítás?

   a) 4
   b) 5
   c) 0
   d) 28

3. Melyik a MOD és DIV utasítás paramétereinek legnagyobb megengedett értéke?

   a) 32767
   b) 65535
   c) 1E+38

4. Mit eredményez a FRAC-utasítás?

   a) Integer szám tizedesvessző utáni értékét.
   b) Integer szám tizedesvessző előtti értékét.
   c) Valós szám tizedesvessző utáni értékét.
   d) Valós szám tizedesvessző előtti értékét.

5. Ha át akarjuk kapcsolni az 53300-as rekesz 0. és 1. bitjét, melyik utasítást alkalmazzuk?

   a) POKE 53300, PEEK (53300) EXOR 3
   b) POKE 53300, EXOR (PEEK (53300),3)

7.2 Számok átalakítása

7.2.1 % – bináris-decimális átalakítás

Formátum: % bináris érték
Paraméterek: bináris érték – egy nyolcjegyű bináris szám
Funkció: Bináris érték átalakítása decimálissá
Példa: PRINT % 11111111
Az eredmény: 256.

A %-jellel jelöljük meg a nyolcjegyű bináris számokat, amelyek ezután kiírathatók vagy tárolhatók. A bináris számokat 0 és 1 számjegyekből építjük fel. Ha más számjegyet is felhasználunk, a

```
NOT BINARY CHAR
```

hibaüzenetet kapjuk.
A PRINT % 00001111 írásmódon kívül alkalmazhatjuk a POKE 53280, %00011100 vagy az A=% 00001110 formát is.
Logikai műveleteknél ez az írásmód nagyon előnyös, mert a bitekkel való manipulációkhoz nem kell előzetes átszámításokat végezni. A 49152-es rekesz 0. és 2. bitjének bekapcsolásához pl. az alábbi egyszerűbb utasítást alkalmazhatjuk:

```
POKE 49152, PEEK (49152) OR % 00000101
```

Ez a módszer megkönnyíti a bitekkel és a byte-okkal végzett munkát.
A 7.1.2 pontban egy olyan programot ismertünk meg, amelyik a decimális számokat 16-bites bináris számokká alakítja.

7.2.2 $ – hexadecimális-decimális átalakítás

Formátum: $ hexa-érték
Paraméterek: hexa-érték – négyjegyű hexadecimális szám
Funkció: A hexadecimális számokat decimálisra alakítja
Példa: PRINT $00FF
Az eredmény: 255.

Ha a hexa-érték nem négyjegyű, vagy érvénytelen karaktereket tartalmaz, akkor a

```
NOT HEX CHAR
```

hibaüzenetet kapjuk.

Ez az utasítás az operációs rendszer rutinjainak alkalmazásánál jelent komoly segítséget. Érdekes rutinokat vehetünk át a Commodore 64-es ROM listájából, „A Commodore 64-es belső felépítése" című DATA BECKER-NOVOTRADE könyvből.* Ezeknek a rutinoknak a címei többségükben hexadecimális formában vannak megadva, ezért mielőtt használnánk, decimálissá kell alakítanunk őket. Erre szolgál a $-jel.
Ha pl. az operációs rendszer $FCE2 (hidegindítás) címét akarjuk decimálisra átszámítani, elegendő az alábbi utasítás beírása.

```
PRINT $FCE2
```

* A mű eredeti címe: 64 Intern (1984)

Az eredmény (64738) megjelenik a képernyõn, a SYS 64738 utasítással pedig elindíthatjuk azt a rutint, amely a gépet alapállapotba hozza. A rutin közvetlenül hexadecimális formában is lehívható az alábbi utasítással:

```
SYS $FCE2
```

Ahol decimális értékek beírása szükséges, de csak azok hexadecimális megfelelõjét ismerjük, ott minden esetben alkalmazhatjuk a $-jelölést.

7.2.3 Ellenőrző kérdések

1. Az alábbi utasítások közül melyik eredményez hibaüzenetet?

   a) PRINT % 10101000
   b) POKE 53000, EXOR (PEEK(53000), %00000011)
   c) PRINT $ 0011

2. Milyen szabályok érvényesek a %-jel alkalmazásakor?

   a) A bináris szám max. 8 számjegyből állhat.
   b) A bináris szám max. 16 számjegyből állhat.
   c) A bináris szám csak 0-s és 1-es számjegyekből állhat.
   d) A bináris szám csak az 1-es számjegyből állhat.

3. Melyik utasítás eredményez hibaüzenetet?

   a) PRINT $FF
   b) POKE 1024, $0000
   c) A% = $00FF + $000F

4. Milyen szabályok érvényesek a $-jel alkalmazásakor?

   a) A hexadecimális szám max. 2 számjegyből állhat.
   b) A hexadecimális szám max. 4 számjegyből állhat.
   c) Csak decimális számjegyek (0–9) alkalmazhatók.
   d) Csak hexadecimális számjegyek (0–9 és A–F) alkalmazhatók.

8. FEJEZET
Műveletek füzérekkel

8.1 INSERT – füzérek egymásba építése

Formátum: INSERT ("1. füzér", "2. füzér", poz.)
Paraméterek: 1. füzér – beépítendő (belső) füzér
2. füzér – külső füzér
poz. – az a pozíció, ahonnan az 1. füzért a 2.-ba beépítjük
Funkció: Egy füzér beépítése egy másik füzérbe
Példa:
```
10 X$=INSERT("CBM V2";"EXPANDED BASIC",8)
20 PRINT X$
```

A program eredményeként a képernyőn a következő felirat fog megjelenni:

```
EXPANDED CBM V2 BASIC
```

Az összeépítendő füzérek (stringek) változóként is megadhatók. A példában szereplő sorok ekkor így módosulnak:

```
10 A$="CBM V2"
20 B$="EXPANDED BASIC"
30 X$=INSERT (A$,B$,8)
40 PRINT X$
```

Ha a pozíciót is változóként adjuk meg, így fog kinézni a programunk:

```
10 A$="CBM V2"
20 B$="EXPANDED BASIC"
30 C=8
40 X$=INSERT(A$,B$,C)
50 PRINT X$
```

Az INSERT utasítás alkalmazásakor az alábbi szabályokat kell betartanunk:

1. A belső füzér nem lehet hosszabb, mint a külső.
2. A végeredményként kapott füzér maximális hossza 255 karakter lehet.
3. Ezzel az utasítással a két füzért nem fűzhetjük össze, azaz a pozícióként megadott szám maximálisan a külső füzér karaktereinek száma, mínusz 1 lehet.
4. Közvetlen üzemmódban a füzéreket csak füzérváltozóként adhatjuk meg, jelkombinációként (pl. "xyz") nem. Bár nem sok értelme van ezt az utasítást közvetlen üzemmódban alkalmazni, a SIMONS's BASIC reagálása mégis elgondolkoztató.

Az 1. és 3. pont be nem tartása
INSERT TOO LARGE,

a 2. pont be nem tartása pedig
STRING TOO LARGE
hibaüzenetet eredményez.

A 4. pontban leírtakat próbáljuk ki az előző program füzérváltozóinak felhasználásával. Írjuk be direkt üzemmódban a

PRINT INSERT (A$, B$,8)

parancsot. Az eredmény a helyes szöveg lesz, azaz

EXPANDED CBM V2 BASIC

Mi történik, ha az A$ helyett beírjuk annak tartalmát, vagyis egy állandót?

PRINT INSERT ("CBM V2",B$,8)

Nem történt semmi különös, az eredmény most is a helyes szöveg.

Cseréljük ki most a B$-t is annak tartalmával:

PRINT INSERT ("CBM V2", "EXPANDED BASIC",8)

Az eredmény most egy összevissza keveredett szöveg:

EXPANDED BASIC BASIC

Ez tehát egy kis szépséghibája ennek az utasításnak, de mivel közvetlen üzemmódban úgy sincs sok értelme használni, tekintsünk el tőle. Programban kifogástalanul működik. Gyakorlásképpen hívjuk be a lemezről az INSERT című programot:

P. 25

```
1 REM ***************
2 REM * I N S E R T *
3 REM ***************
100 X$=INSERT("MAGYARORSZAG","BUDAPEST  FOVAROSA",9)
110 PRINT"■■KULSO STRING        : BUDAPEST  FOVAROSA"
120 PRINT"■BEEPITENDO STRING  : MAGYARORSZAG"
130 PRINT"■A PROGRAM EREDMENYE: "
140 PRINT"■■■■■■■";X$
150 REM * PELDA VALTOZOKKAL *
160 PRINT"■■■PROBALJA MEG ON IS AZ OSSZEEPITEST!NYOM-JON LE EGY BILLENTYUT!"
170 GETQ$:IF Q$="" THEN 170
180 INPUT"■KEREM A BEEPITENDO STRINGET: ";A$
190 INPUT"■KEREM A KULSO STRINGET     : ";B$
200 INPUT"■MELYIK POZICIOTOL KERI A BEEPITEST: ";C
210 X$=INSERT(A$,B$,C)
220 PRINT"■■■■■■■";X$
230 END
```

Figyelem! Ha a programot elindítottuk, a feltett három kérdésre feltétlenül válaszolni kell (nem lehet RETURN-nel átugrani), mert különben a gép "kiakad" és csak úgy tudjuk helyrehozni, ha kikapcsoljuk. Ez a program az eredeti könyvben nem szerepel, szemléltetés céljából közöljük – magyar változat készítői.

8.2 INST – füzér felülírása

Formátum: INST ("1.füzér", "2.füzér",poz.)
Paraméterek: 1.füzér – a felülíró füzér,
2.füzér – az eredeti füzér,
poz. – az eredeti füzérben az a pozíció, amelytől kezdve a felülírás történik
Funkció: Egy füzér felülírása egy másikkal
Példa:
```
10 X$ = INST ("130", "MENNYISÉG:500 DB",11)
20 PRINT X$
```

A képernyőn az alábbi szöveg fog megjelenni:

MENNYISÉG: 130 DB,

vagyis az 1. füzér felülírta a 2.-at a 12. karaktertől kezdődően. Láthatjuk, hogy a poz.-paraméter tulajdonképpen nem is azt a pozíciót jelöli, ahonnan az átírás történik, hanem pontosabban annyi karaktert jelent, amennyi az eredeti szövegből megmarad.
Az INST utasítás paraméterei megegyeznek az INSERT utasítás paramétereivel.
Ebben az esetben az 1. füzér hosszabb is lehet a 2.-nál. Káros következményekkel jár, ha a 2. füzér meghatározatlan füzérváltozó. Ekkor a SIMON's BASIC "elszáll" és további füzérfeldolgozásra alkalmatlanná válik.
A poz.-paraméter nem haladhatja meg a 2. füzér hosszát. Ilyenkor ugyan nem kapunk hibaüzenetet, de az így keletkezett füzér felépítése nem fog megfelelni az elképzeléseinknek.
A létrejött füzér természetesen most sem haladhatja meg a 255 karakternyi hosszúságot.
Az INST utasítás érdekes megoldást kínál relatív adatfile-ok szervezésekor a rekordok összeállításához.
Egy rekord általában több mezőből áll, amelyeknek mindig ugyanazon a helyen kell kezdődniük. Egy cím-file rekord felépítése pl. így nézhet ki:

Mező	Kezdőpozíció a rekordban	Változó
Vezetéknév	0	V$
Keresztnév	25	K$
Utca	45	U$
Irányítószám/helység	75	H$

Szerkesszük most meg ebből a négy füzérből azt a füzért, amelynek felépítése a fenti struktúrát követi. A füzér hossza 100 karakter legyen, amelyet először üres karakterekkel töltünk fel. Az alábbi programrészlet azt szemlélteti, hogy hogyan épül fel az új füzérben (R$) a rekord:

```
100 R$=DUP("",100)
110 R$=INST(V$,R$,0)
120 R$=INST(K$,R$,25)
130 R$=INST(U$,R$,45)
140 R$=INST(H$,R$,75)
```

A 100. sorban a DUP-utasítással (8.4. fejezet) előállítottuk a 100 szóközt tartalmazó R$ füzért, amelyet a következő négy sorban rendre felülírunk az adatokat jelölő változókkal. Ezután a rekord beolvasásakor minden mező a megfelelő helyre fog kerülni, a cím-file egységes szerkezetű lesz.
A VC–1541 típusú lemezegység relatív adattárolásáról a "A VC–1541-es lemezegység programozása" című DATA BECKER – NOVOTRADE kiadványból* tudhatunk meg többet. Nagyon jól használható továbbá az INST-utasítás táblázatok kinyomtatásához. Az egyes mezőket elhelyezzük egy füzér adott pozíciójában és ezután a füzért kinyomtatjuk. Az elv az előzőleg ismertetett rekord felépítésével megegyezik.
Ha már végeztünk füzérfeldolgozást standard BASIC-kel, megfelelően fogjuk értékelni az INST által nyújtott segítséget.

* Eredeti címe: Das Große Floppy Buch (1983).

8.3 PLACE – füzér keresése

Formátum: PLACE ("1.füzér",füzérváltozó)
Paraméterek: 1.füzér – a keresett füzér;
füzérváltozó – füzérváltozó, amelyben az 1. füzért keressük
Funkció: Bizonyos karaktersorozat megkeresése egy füzérben
Példa: A$="123*4567"
PRINT PLACE ("*",A$)
Az A$ füzérből kikeresi a "*"-ot.

A PLACE-szel egy rendkívül jól használható utasításhoz jutottunk, amely hatását tekintve a standard BASIC-kel csak nehezen valósítható meg.
INST-tel kombinálva további előnyökhöz juthatunk.
Példa: A Commodore 64-esen, mint az összes többi amerikai szabvány szerint készült számítógépen, nem tizedesvesszőt, hanem tizedespontot használunk. A PLACE segítségével megoldható, hogy a számokat (pl.: pénzösszegeket) a mi gyakorlatunknak megfelelően tizedesvesszős formában írassuk ki.
Nézzük erre a következő példaprogramot:

P. 26

```
1 REM ***************
2 REM * 1.P L A C E *
3 REM ***************
100 REM
110 INPUT"OSSZEG: ";B
120 B$=STR$(B)
130 P=PLACE(".",B$)
140 B$=INST(",",B$,P-1)
150 PRINT"OSSZEG: ";B$
```

A 110. sorban beolvasott "ÖSSZEG"-et a B változóban tároljuk, hogy később számításokat végezhessünk vele. A 120. sorban a numerikus változót füzérváltozóvá alakítjuk, majd a 130. sorban a PLACE-utasítással megkeressük a tizedespont helyét és ennek pozícióértékét a P változóhoz rendeljük.
A 140. sorban az INST utasítást alkalmazva a pontot vesszővel helyettesítjük és a 150. sorban kiíratjuk az eredményt.
A 140. sorban a vessző pozícióját (P – 1)-gyel határoztuk meg, aminek az az oka, hogy az INSERT és INST utasítások a számolást 0-tól kezdik, így a PLACE-utasítás 1. pozíciója ezekben 0-nak felel meg. Ugyanez a probléma standard BASIC-kel is megoldható, de legalább háromszor ennyi sorban egy sokkal áttekinthetetlenebb programmal.
A PLACE-utasítással nem csak egy karakter, hanem tetszőleges karaktersorozat is kikereshető, amint ezt a következő példa is bizonyítja:

```
10 A$="A DATA BECKER KÖNYVEK TANITANAK"
20 PRINT PLACE ("KÖNYVEK",A$)
```

Beírva, majd elindítva a képernyőn megjelenik a 15-ös szám, ami a KÖNYVEK – karaktersorozat kezdőpozícióját jelenti.
Ez az utasítás annyira megnyerte a tetszésünket, hogy kidolgoztunk egy programot, amely tetszőleges információkat keres és tárol. Egy-egy információ max. 2 képernyősor hosszúságú, amelyeket lemezen rögzítünk. A teljes információs-file-t, mint egy táblázatot, betöltjük a memóriába, ahol a címszavak a PLACE-utasítással könnyen és gyorsan kikereshetők.
Hívjuk be tehát a 2.PLACE-című programot és tanulmányozzuk működését, valamint a programlistát:

```
1 REM ***************
2 REM * 2.P L A C E *
3 REM ***************
1000 REM ***************
1010 REM ELOKESZULETEK
1020 REM ***************
1030 POKE53280,2:POKE53281,2
1040 PRINTCHR$(158)
1050 DIMI$(4000)
1060 PROC MENU
1070 :: REM ****
1080 :: REM MENU
1090 :: REM ****
1100 ::EXEC PROGRAM-FEJLEC
1110 ::PRINT" VALASSZA KI A KIVANT FUNKCIOT!"
1120 ::PRINT" ";DUP("-",37):PRINT
1130 ::PRINT TAB(8);" -1- INFO-FILE BEOLVASASA"
1140 ::PRINT TAB(8);" -2- INFO-FILE TAROLASA"
1150 ::PRINT TAB(8);" -3- ADATROGZITES"
1160 ::PRINT TAB(8);" -4- ADATOK TORLESE"
1170 ::PRINT TAB(8);" -5- ADATOK KERESESE"
1180 ::PRINT TAB(8);" -6- PROGRAM VEGE"
1190 ::LOOP
1200 :::GET X$
1210 :::X=VAL(X$)
1220 :::EXIT IF X>0 AND X<7
1230 ::END LOOP
1240 :: IFX=1 THEN EXEC INFO-FILE BEOLVASASA
1250 :: IFX=2 THEN EXEC INFO-FILE TAROLASA
1260 :: IFX=3 THEN EXEC ADATROGZITES
1270 :: IFX=4 THEN EXEC ADATOK TORLESE
1280 :: IFX=5 THEN EXEC ADATOK KERESESE
1290 :: IFX=6 THEN EXEC PROGRAM VEGE
1300 ::CALL MENU
1310 PROC INFO-FILE BEOLVASASA
1320 REM ********************
1330 REM INFO-FILE BEOLVASASA
1340 REM ********************
1350 ::EXEC PROGRAM-FEJLEC
1360 ::INPUT"AZ INFO-FILE NEVE:";DN$
1370 ::OPEN1,8,2,DN$:CLOSE1
1380 ::OPEN15,8,15
1390 ::INPUT#15,F1$,F2$,F3$,F4$
1400 ::IF F1$="00"THEN CLOSE15:CALL OLVASAS
1410 :::PRINT:PRINT"LEMEZHIBA!"
1420 :::PRINT"----------------"
1430 :::PRINTF1$","F2$","F3$","F4$
1440 :::CLOSE15
1450 :::CALL 1.VISSZAUGRAS
1460 ::PROC OLVASAS
1470 :::OPEN1,8,2,DN$
1480 :::PRINT"UJ VAGY HOZZAOLVASANDO (U/H)?"
1490 :::REPEAT
1500 :::GET X$
1510 :::UNTIL X$="U" OR X$="H"
1520 :::IF X$="U" THEN Y=0
1530 :::LOOP
1540 ::::Y=Y+1
1550 ::::INPUT#1,I$(Y)
1560 ::::EXIT IF ST=64
1570 :::END LOOP
1580 :::CLOSE1
1590 :::PRINT"A SZAMITOGEP "Y" INFORMACIOT TARTALMAZ"
```

```
1600 ::PROC 1.VISSZAUGRAS
1610 ::::PRINT:PRINT">>RETURN<<"
1620 ::::REPEAT
1630 ::::::GET X$
1640 ::::UNTIL X$=CHR$(13)
1650 END PROC
1660 PROC INFO-FILE TAROLASA
1670 ::REM ********************
1680 ::REM INFO-FILE TAROLASA
1690 ::REM ********************
1700 ::EXEC PROGRAM-FEJLEC
1710 ::IF Y=0 THEN PRINT"NINCS ADAT!!":CALL2.VISSZAUGRAS
1720 ::INPUT"FILE NEV";DN$
1730 ::PRINTDN$" NEVU FILE TAROLASA !"
1740 ::OPEN 1,8,2,CHR$(64)+":"+DN$+",S,W"
1750 ::FOR I=1 TO Y
1760 ::::PRINT#1,I$(I)
1770 ::NEXT I
1780 ::CLOSE1
1790 ::PROC 2.VISSZAUGRAS
1800 ::::PRINT:PRINT">>RETURN<<"
1810 REPEAT
1820 ::::::GET X$
1830 ::::UNTIL X$=CHR$(13)
1840 END PROC
1850 PROC INFORMACIOK BEADASA
1860 ::REM ***********************
1870 ::REM INFORMACIOK BEADASA
1880 ::REM ***********************
1890 ::EXEC PROGRAM-FEJLEC
1900 ::IF Y>0 THEN CALL BEVITEL
1910 ::::INPUT"FILE NEV: ";DN$
1920 ::PROC BEVITEL
1930 ::EXEC PROGRAM-FEJLEC
1940 ::::Y=Y+1
1950 ::::PRINT"ADJA BE AZ "Y"SZ. INFORMACIOT:"
1960 PRINT"(MAXIMUM 2 SORBAN)"
1970 ::::PRINTDUP("-",39)
1980 ::::INPUTI$(Y)
1990 ::::PRINTDUP("-",40);
2000 ::::PRINT"HELYES (I/N)?"
2010 ::::REPEAT
2020 ::::::GET X$
2030 ::::UNTIL X$="I" OR X$="N"
2040 ::::IF X$="N"THEN Y=Y-1:CALL BEVITEL
2050 ::::PRINT"TOVABBI BEVITELEK(I/N)?"
2060 ::::REPEAT
2070 ::::::GET X$
2080 ::::UNTIL X$="I" OR X$="N"
2090 ::::IF X$="I" THEN CALL BEVITEL
2100 END PROC
2110 PROC INFORMACIOK TORLESE
2120 ::REM ********************
2130 ::REM INFORMACIOK TORLESE
2140 ::REM ********************
2150 ::EXEC PROGRAM-FEJLEC
2160 ::IF Y=0 THEN PRINT"NINCS ADAT!":CALL3.VISSZAUGRAS
2170 ::INPUT"AZ INFORMACIO SZAMA: ";N
2180 ::PRINTDUP("-",39)
2190 ::PRINT"  ";I$(N)
2200 ::PRINTDUP("-",39)
2210 ::PRINT"TOROLNI(I/N)?"
2220 ::REPEAT
2230 ::::GET X$
2240 ::UNTIL X$="I" OR X$="N"
```

```
2250 ::IF X$="N" THEN CALL3.VISSZAUGRAS
2260 ::FOR I=N TO Y
2270 ::::I$(I)=I$(I+1)
2280 ::NEXT I
2290 ::Y=Y-1
2300 ::PRINT"AZ ADAT TOROLVE!"
2310 ::PROC 3.VISSZAUGRAS
2320 ::::PRINT">>RETURN<<"
2330 ::::REPEAT
2340 ::::::GET X$
2350 ::::UNTIL X$=CHR$(13)
2360 END PROC
2370 PROC INFORMACIOK KERESESE
2380 ::REM **********************
2390 ::REM INFORMACIOK KERESESE
2400 ::REM **********************
2410 ::EXEC PROGRAM-FEJLEC
2420 ::IF Y=0 THEN PRINT"NINCS ADAT!!":CALL 4.VISSZAUGRAS
2430 ::PRINT"HANY KULCSSZO (1-5)?"
2440 ::REPEAT
2450 ::::GET X$
2460 ::UNTIL X$>0 OR X$<6
2470 ::S=VAL(X$):VK$="V"
2480 ::EXEC PROGRAM-FEJLEC
2490 ::IF S=1 THEN CALL TOVABB
2500 ::PRINT"ES/VAGY KAPCSOLAT (E/V)?"
2510 ::REPEAT
2520 ::::GET X$
2530 ::UNTIL X$="E"OR X$="V"
2540 ::VK$=X$
2550 ::PROC TOVABB
2560 ::PRINT"KEPERNYO VAGY PRINTER (K/P)?"
2570 ::REPEAT
2580 ::::GET X$
2590 ::UNTIL X$="K" OR X$="P"
2600 ::IF X$="K"THEN OPEN1,3:ELSE:OPEN1,4
2610 ::AU$=X$:PRINT DUP("-",20)
2620 ::FOR I=1TO S
2630 ::::PRINTI",KULCSSZO : ";
2640 ::::INPUT S$(I)
2650 ::NEXT I
2660 ::FOR S1=1 TO Y
2670 ::::U1=0:FL=0
2680 ::::FOR S2=1 TO S
2690 ::::::Z=PLACE(S$(S2),I$(S1))
2700 ::::::IF Z<>0 AND VK$="V" THEN FL=1:EXEC KIVITEL
2710 ::::::IF Z=0 AND VK$"E" THEN CALL NEXT S1
2720 ::::::IF Z<>0 AND VK$"E" THEN U1=U1+1
2730 ::::::IF FL=1 THEN CALL NEXT S1
2740 ::::NEXT S2
2750 ::::IF U1<=S THEN EXEC KIVITEL
2760 ::PROC NEXT S1
2770 ::NEXT S1
2780 ::PRINT"KERESESNEK VEGE!"
2790 ::CALL 4.VISSZAUGRAS
2800 ::PROC KIVITEL
2810 ::::IF AU$="K" THEN EXEC PROGRAM-FEJLEC
2820 ::::PRINT#1,"INFO NR.";S1
2830 ::::PRINT#1,DUP("-",39)
2840 ::::PRINT#1,I$(S1)
2850 ::::PRINT#1,DUP("-",39)
2860 ::::PRINT#1
2870 ::::IF AU$="P" THEN CALL 1.VISSZA
2880 ::::::PRINT">>RETURN<<"
2890 ::::::REPEAT
2900 ::::::::GET X$
```

```
2910 ::::::UNTIL X$=CHR$(13)
2920 ::::PROC 1.VISSZA
2930 ::END PROC
2940 ::PROC 4.VISSZAUGRAS
2950 ::::PRINT">>RETURN<<"
2960 ::::REPEAT
2970 ::::::::GET X$
2990 CLOSE1
3000 END PROC
3010 PROC PROGRAMVEG
3020 ::REM **********
3030 ::REM PROGRAMVEG
3040 ::REM **********
3050 ::EXEC PROGRAM-FEJLEC
3060 ::PRINT"BIZTOS BENNE(I/N)?"
3070 ::REPEAT
3080 ::::GET X$
3090 ::UNTIL X$="I" OR X$="N"
3100 ::IF X$="N" THEN END PROC
3110 ::EXEC PROGRAM-FEJLEC
3120 ::PRINT" A PROGRAM A":PRINT
3130 ::PRINT"       'CALL MENU'UTASITASSAL":PRINT
3140 ::PRINT" UJRAINDITHATO ANELKUL,HOGY":PRINT
3150 ::PRINT" AZ ADATOK ELVESZNENEK!"
3160 ::END
3170 PROC PROGRAM-FEJLEC
3180 ::REM **************
3190 ::REM PROGRAM-FEJLEC
3200 ::REM **************
3210 ::PRINTCHR$(147);
3220 ::PRINTDUP("=",40);
3230 ::PRINTCHR$(18);" UNIVERZALIS INFORMACIO-";
3240 ::PRINT"RENDSZER       ";CHR$(146);
3250 ::PRINTDUP("=",40);
3260 PRINT"INFORMACIOK SZ.:"Y"  FILENEV: "DN$
3270 ::PRINTDUP("=",40)
3280 ::PRINT:PRINT
3290 END PROC
```

Ezzel a programmal olyan információs rendszerhez jutottunk, amelyet a legkülönfélébb célokra hasznosíthatunk, ráadásul az eddig megismert egyéb utasítások alkalmazását is újabb példákkal szemlélteti.
Indítsuk el a programot és nézzük, hogyan működik:
A program hat lehetőséget kínáló menüvel jelentkezik.

–1– INFO-FILE BEOLVASÁSA
Kiválasztása után először megkérdezi az info-file nevét. Itt azt a nevet kell beírni, amelyet előzőleg a tárolásnál a file-nak adtunk. Ezután választanunk kell, hogy új file-t akarunk-e létrehozni, vagy egy meglévőt kívánunk folytatni. Lehetőségünk van ugyanis arra, hogy egy meglévő file-hoz egy új file-t csatoljunk. Az így létrejött file-t új néven rögzíthetjük. Ha a program kijelezte a tárolt információkat, RETURN-nel visszatérhetünk a menühöz.

–2– INFO-FILE TÁROLÁSA
Ha ezt a lehetőséget választjuk, de a tárban nincs adat, a NINCS ADAT!-hibaüzenetet kapjuk.
Amikor a file nevét beadtuk, a program a file-t lemezre viszi.
Figyelmeztetés: A lemezen esetleg azonos néven szereplő file felülíródik!
A menühöz a RETURN-nel térhetünk vissza.

–3– ADATRÖGZÍTÉS

Ezt a funkciót akkor választjuk, ha egy file-ba elöször akarunk adatokat rögzíteni, vagy egy már meglévö file-t bövíteni akarunk. Abban az esetben, ha nincs tárolt információ, a program rákérdez a file nevére, majd ezt az információk számával együtt kijelzi a fejlécben.
Amikor a file-nevet beírtuk, a képernyön megjelenik a beviteli maszk és beadhatjuk a max. 2 képernyösor hosszúságú információ-mondatot. A program a rögzítés elött feltesz egy ellenörzö kérdést a beadás helyességére vonatkozóan. Ha erre "N"-nel válaszolunk, megismételhetjük a beírást. Helyes beírás esetén a program megkérdezi, hogy akarunk-e még adatokat rögzíteni. "I"-re megismétlödik az alprogram, "N"-re visszatérünk a menühöz.

–4– ADATOK TÖRLÉSE

Természetesen lesznek olyan adatok (pl. határidök), amelyek egy idö után feleslegessé válnak és ki kell törölni öket a file-ból. Ez az alprogram a rekordok célirányos törlését végzi. Ehhez be kell írnunk az információ sorszámát. Ha esetleg nem ismerjük, segítségül kell hívnunk a –6–-os alprogramot (PROGRAM VÉGE).
Mielött a rekord törlödne, teljes egészében megjelenik a képernyön és így eldönthetjük, hogy valóban törölni akarjuk-e. Amennyiben nem, úgy "N"-nel visszatérhetünk a menühöz.
Figyelmeztetés! Törléskor átszerveződik a file. A törölt rekordot követö rekordok egy hellyel elörébb kerülnek.

–5– ADATOK KERESÉSE

Ez a rész a program lényege. A legnagyobb file is használhatatlan, ha nem tudjuk belöle rövid idö alatt kikeresni a szükséges információkat. Annyit ér, mint egy telefonszámok szerint rendezett telefonkönyv.
Ez az alprogram biztosítja a file egyszerü kiértékelését és nyomtatón vagy képernyön történö megjelenítését.
A kiválasztást követöen meg kell adnunk, hogy a rekordban hány kulcsszó szerint (1–5) keresünk. Ha egynél többel, akkor el kell döntetünk, hogy azokra ÉS, ill. VAGY kapcsolatban van-e szükségünk. Pl.: ha két címszót keresünk, akkor a rekordnak ÉS kapcsolat esetében mindkettöt, VAGY kapcsolatban pedig legalább az egyiket tartalmaznia kell. Az erre vonatkozó kérdésre "E"-vel, ill. "V"-vel kell válaszolni. Maximálisan 5 kulcsszó szerint kereshetünk.
Ha a logikai müveletre vonatkozó kérdést megválaszoltuk, meg kell választanunk a kivitel módját (képernyö vagy nyomtató).
Végül írjuk be a kulcsszavakat.
A program végét a képernyön megjelenö üzenet jelzi.
A menühöz a RETURN-nel térhetünk vissza.

–6– PROGRAM VÉGE

Egy ilyen példásan megírt programtól joggal elvárhatjuk, hogy ne a RUN/STOP billentyüvel lehessen kilépni belöle. Ez az alprogram befejezi a programot, de elötte válaszolnunk kell arra a kérdésére, hogy valóban nem akarunk-e már tovább dolgozni (BIZTOS ÖN EBBEN? I/N). Ha "N"-nel válaszolunk, visszatérünk a menühöz, míg "I"-re a program befejezödik, de a képernyön megjelenö üzenet szerint az adatok elvesztése nélkül újra indíthatjuk.
Nos, kihasználva a CBM 64-es tárolóját, elkészíthetjük saját adatbankunkat.

8.4 DUP – füzér ismétlése

Formátum: DUP ("füzér", n)
Paraméterek: füzér – valamilyen füzér vagy füzérváltozó
n – szorzótényező
Funkció: Egy füzér meghatározott számú ismétlése
Példa: PRINT DUP ("–",40)
A képernyőn egymás után kiír 40 db "–" jelet.

Ezzel az utasítással leegyszerűsíthetjük pl. egy szövegrész aláhúzását és megmenekülünk egy csomó értelmetlen billentyűzéstől. Ráadásul nem csak egy-egy karakter ismételhető, hanem több karakterből álló füzér is. Például:

```
10 A$ = DUP ("SOS",10)
20 PRINT A$
```

A paramétereket változóként is megadhatjuk:

```
10 A$="SOS"
20 FOR I=1 TO 10
30 PRINT DUP (A$,I)
40 NEXT
```

Az utasítást vezérlőkarakterekre (pl. kurzorvezérlő) is alkalmazhatjuk:

```
10 PRINT "10.SOR"
20 PRINT DUP (CHR$(145),10)
30 PRINT "30.SOR"
```

(A CHR$(145) a KURZOR FEL ASCII-kódja).

Nincs szükség tehát arra, hogy a kurzor többszörös pozicionálásához ciklust szervezzünk.
A következő lista azt szemlélteti, hogy az utasítás nyomtatón is alkalmazható, pl. feliratok aláhúzásánál.

```
10 OPEN 1,4
20 PRINT#1,DUP("–",80)
30 CLOSE 1
```

A DUP tehát megint egy olyan utasítás, amelyre már régóta vártunk.

8.5 Ellenőrző kérdések

1. Mi az eredménye a következő programnak?

```
10 A$="AAA BBB CCC "
20 PRINT INSERT (" XXX",A$,4)
```

a) AAAXXX BBB CCC
b) AAA XXX BBB CCC
c) AAA XXXBBB CCC

2. Milyen szabályok érvényesek az INSERT-utasítás alkalmazásakor?

a) A belső füzér kisebb legyen a külsőnél.
b) A belső füzér nagyobb legyen a külsőnél.
c) A létrejött új füzér hossza max. 255 karakter lehet.

3. Az alábbi INSERT-utasítások közül melyik eredményezheti a KLAGENFURT feliratot?

a) 10 PRINT INSERT("LAGENF","KURT",1)
b) 10 PRINT INSERT("LAGENF","KURT",0)
c) 10 PRINT INSERT("LAGENF","KURT"2)
d) Egyik sem.

4. Mi az eredménye a következő programnak?

```
10 A$="AAABBBCCCDDD"
20 PRINT INST (" ",A$,4)
```

a) AAA BBBCCCDDD
b) AAA BBCCCDDD
c) AAAB BCCCDDD
d) AAA BBCCCDDD

5. Melyik pozíciót jelzi ki az alábbi utasítássorozat?

```
10 A$="AA BB CC"
20 PRINT PLACE (" ",A$)
```

a) 1
b) 2
c) 3

6. A következő utasítások közül, melyiknek *nem* 12 az eredménye?

a) 10 PRINT PLACE(" ","AAA.BBB.CCC DDD.EEE")
b) 10 PRINT PLACE("XY",YX X Y XX YXY XX")
c) 10 PRINT PLACE("100","1000100010100100")

7. A következő utasítások közül melyiknek nem ez az eredménye:

   "----------"?

   a) PRINT DUP ("-",12)
   b) PRINT DUP ("--",4)
   c) PRINT DUP ("----",3)

8. Vezérlőkarakterek ismétlésére alkalmas-e a DUP-utasítás?

   a) igen
   b) nem

9. Hány karakter lehet a DUP-pal előállított jelsorozat hossza?

   a) 256
   b) 1000
   c) 255
   d) 1024

9. FEJEZET
Megjelenítés – (kivitel)vezérlés

A Simon's Basic nagyon sok segítséget nyújt a képernyõn megjelenõ információk formátumának alakításában. Ebben a fejezetben olyan utasításokkal ismerkedünk meg, amelyekkel a program külsõ megjelenítése profi szinten történhet.
Lehetõségünk nyílik a megjelenés formálására, villogtatásra, vagy teljes képernyõrészek kezelésére. Elõször a formáló, majd a villogtató és a gördítés (scroll) utasításokat ismertetjük. A gyorsabb elsajátítás érdekében, lehetõség szerint a tanulással együtt próbáljuk is ki az utasításokat.

9.1 Formált megjelenítés

Formált megjelenítésen azt értjük, hogy a képernyõn megjelenõ szövegek, számítások, ...stb. az általunk meghatározott alakban jelenik meg. Az eredeti CBM BASIC-ben is van néhány utasítás a képernyõ tartalmának áttekinthetõbbé tételére. Ezek a PRINT-tel együtt alkalmazandók (a POS kivételével). Ezeknek az utasításoknak a funkcióját és az abból adódó lehetõségeket az alábbiakban röviden vázoljuk.
Elsõként egy lényeges fogalmat tisztázunk:

Kurzor

Azt a pozíciót jelöli a képernyõn, ahová a következõ karakter fog kerülni. Két megjelenési módja van. Közvetlen üzemmódban jól látható villogó kis négyzet, program üzemmódban vìszont láthatatlan, csak virtuálisan van jelen, így mutatja az aktuális képernyõ pozíciót. Ezért, ha kurzorpozícióról beszélünk, akkor az mindig a következõ kiírandó karakter helyét jelenti.

SPC

Üres helyet hagy ki az aktuális kurzorpozíció és az azt követõ szöveg között, tehát a képernyõi kivitel relatív pozícionálására szolgál.

TAB

Meghatározza a sornak azt az oszlopát, ahonnan a következõ karakter kerül. Itt tehát abszolút pozícionálásról beszélünk. (lásd az AT-utasítást is).

POS

A pillanatnyi kurzorpozíció meghatározására szolgál. Segíti a képernyõn való tájékozódást, és nagyon hatákonyan alkalmazható a SIMON'S BASIC-ben a LIN-utasítással együtt.

,/;

A PRINT utasítást követõen automatikusan végrehajtásra kerül egy kocsi vissza (carriage return) pozícionálás. Mint tudjuk, a PRINT utasítás befejezhetõ vesszõvel vagy pontosvesszõvel.
Ha pontosvesszõvel fejezzük be a PRINT utasítást, akkor szóköz nélkül, folyamatosan folytatódik a kivitel, ha a vesszõt használjuk, akkor automatikus tabulálást végez a gép. Ezek után nézzük a SIMON'S BASIC kiegészítõ utasításait.

9.1.1 CENTRE – szöveg elhelyezése középen

Formátum: CENTRE kf
Paraméterek: kf – karakterfüzér, vagy sztringváltozó, a hossza < 40
Funkció: egy füzért a képernyősor közepén jelenít meg
Példa: CENTRE A$
vagy
CENTRE"SIMON's BASIC"

A CENTRE-utasítással a képernyősor közepére pozícionáljuk a megjelenítést.
Pl. a "Commodore 64" feliratot a képernyő közepére szeretnénk kiíratni.
A rendes BASIC-ben, ehhez előbb meg kell számolnunk a karakterfüzér hosszát, ezt levonjuk a soronkénti 40 karakterszámból és a maradékot osztjuk kettővel, így kapjuk meg a felirat előtti üres helyek számát.
A CENTRE utasítás megkönnyíti a munkánkat, mert automatikusan kihagyja az üres helyek számát. A kiírandó szöveg nem lehet hosszabb, mint 39 karakter, különben a kivitel véletlenszerű lesz. A kurzor a kivitel után közvetlenül a felirat utáni pozíción áll, és ha a következő sorba szeretnénk írni, előbb egy PRINT vagy PRINT CHR$(13) formában egy 'carrigae return'-t kell végrehajtani.
A CENTRE " " – utasítással pozícionálható a kurzor anélkül, hogy írnánk valamit. A későbbi kiírást ettől a helytől folytathatjuk.
Ha a CENTRE utasítást követően közvetlenül beírunk egy másikat, akkor a második kivitel pontosan olyan távolságra kerül az elsőtől, amennyi üres hely volt ez előtt. Ezt ügyesen használhatjuk arra, hogy a megjelenítendő karakterek állandó távolságra legyenek egymástól.
Nézzük a következő példát:

P. 28

```
1 REM ***************
2 REM * C E N T R E *
3 REM ***************
100 REM
110 PRINT CHR$(147):REM KEPERNYOTORLES
120 A$="SIMON'S BASIC"
130 B$="A  CBM 64-HEZ"
140 CENTRE A$:PRINT
150 CENTRE B$:PRINT
160 CENTRE "**************"
170 REM
180 REM ***********
190 REM KIEGESZITES
200 REM ***********
210 PRINT
220 FOR X=15 TO 1 STEP -2
225 PAUSE 1
230 CENTRE DUP("*",X):PRINT
240 NEXT X
```

A példa első részében egy feliratot írtunk ki képernyő közepére, a másodikban pedig egy szép effektust mutattunk be.

9.1.2 USE – a tizedespont pozícionálása

Formátum: USE 1. kf., 2. kf.
Paraméterek: 1. kf. – karakterfüzér vagy füzérváltozó,
formátuma: "###.###" vagy
"### szöveg.### 'szöveg"
2. kf. – karakterfüzér vagy füzérváltozó, numerikus értékkel
Funkció: Numerikus értékek megjelenítése a tizedespont pozícionálásával
Példa: USE "###.## ", "129.345"

Az USE utasítás a tizedestörtek táblázatos kiírásához nyújt segítséget. Anélkül, hogy minden egyes számnál meg kellene állapítani a tizedesjegyek helyét, egymás alá kerülő tizedespontokkal ábrázolhatjuk a különböző számítások eredményét.
A kiírási formátumot az 1. füzérrel adjuk meg. Minden # jel, a kijelzett szám egy decimális helyét jelenti. Ha a tizedespontig nincs számjegy, akkor annyi üres hely lesz a kivitelnél, ahány # jel van a tizedespontig. Ugyanez érvényes a vesszõ (pont) utánra is. Ha viszont a tizedestört hosszabb a kijelölt formátumnál, akkor a fölösleges karakterek elvesznek. Erre különösen figyeljünk, és ha kell, kerekítsünk.
A „Paraméterek" címszó alatt megadott helyeket természetesen szöveggel is megtölthetjük, ha szem elõtt tartjuk a pont és a # jelek szerepét.
A szöveget mint egyszerû karakterfüzért és mint füzérváltozót is beadhatjuk.
A 2. füzér a formatálandó tizedesértéket tartalmazza, és ezt elõzõleg az STR$ ( ) utasítással füzérré kell alakítani.
Ha ugyanabban a sorban szöveget és formálást is alkalmazunk, hiba léphet fel a formálásban, ha a tizedespont utáni utolsó számjegy 0.
Ezt úgy kerülhetjük el, hogy a szöveget PRINT-tel írjuk, és USE-val kapcsoljuk hozzá a formálást.

P. 29

```
1 REM ***********
2 REM * 1.U S E *
3 REM ***********
100 REM
110 PRINT CHR$(147):REM KEPERNYOTORLES
120 A$="###.##"
130 A=12.32
140 B=230.14
150 USE "IRODASZER:  "+A$,STR$(A):PRINT" FT"
160 USE "KONYVEK:    "+A$,STR$(B):PRINT" FT"
170 PRINT"--------------------"
180 USE "OSSZESEN:   "+A$,STR$(A+B):PRINT" FT"
```

A példában a most ismertetett utasítás egy érdekes alkalmazását mutattuk be.
Ha a 130. és a 140. sorban A és B értékét 12.30-ra ill. 230.10-re változtatjuk, a fent említett formálási hibát is tapasztalni fogjuk, melyet elkerülhetünk, ha a programot így módosítjuk:

P. 30

```
1 REM ***********
2 REM * 2.U S E *
3 REM ***********
100 REM
110 PRINT CHR$(147):REM KEPERNYOTORLES
120 A$="###############.##"
130 A=12.30
140 B=230.10
150 PRINT"IRODASZER:  ":USE A$,STR$(A):PRINT" FT"
160 PRINT"KONYVEK:    ":USE A$,STR$(B):PRINT" FT"
170 PRINT"--------------------"
180 PRINT"OSSZESEN:   ":USE A$,STR$(A+B):PRINT" FT"
```

Kis szépséghibája ennek is van, aki gondolja, tovább kísérletezhet! (– a magyar változat készítõi).

9.1.3 AT – füzér kivitele pozícionálással

Formátum: PRINT 1. kf AT(o,s) 2. kf
Paraméterek: 1. kf/2. kf – két kiírandó karakterlánc, vagy füzérváltozó, ahol az 1. kf el is hagyható
o – oszlop
s – sor

Funkció: Füzér pozícionált kivitele
Példa: PRINT AT(10,12)"CBM 64"

A PRINT-tel aktivizált AT-utasítás lehetőséget ad arra, hogy a képernyő tetszés szerinti részébe helyezzük a 2. kf címen megadott füzért. Az o és s paraméterek a kurzor kezdő koordinátáit, vagyis a kiíratás kezdő helyét jelöli. Ez esetben is érvényes, hogy minden további kiíratás a pozícionált füzér utolsó karakterét követően jelenhet meg.
Ha a 2. kf üres füzér, akkor ez egyszerű, kivitel nélküli kurzorpozícionálást jelent. Természetesen a 1. kf-rel a régi helyre is írhatunk szöveget.
Az AT-utasítás egyik különösen érdekes felhasználási területe a kis felbontású grafika (LGR–LOW–RESOLUTION–GRAPHICS). Ehhez "pontként" a Commodore 64-es bármelyik karakterét felhasználhatjuk. Az AT utasítás felhasználási köre az állandó próbálkozások során egyre bővül, tehát rendkívül hasznos és sokoldalúan alkalmazható.
Egy utolsó megjegyzés a formátumhoz: A zárójel az utasítás részét képezi, ezért az AT utasításszó és a "("-jel közé nem szabad szóközt (space) tenni.
A példaprogram:

P. 31

```
1 REM *******
2 REM * A T *
3 REM *******
100 REM
110 REM ********
120 REM 1. PELDA
130 REM ********
140 REM
150 PRINT CHR$(147):REM KEPERNYO TORLES
160 PRINT AT(10,5) "IGY MUKODIK "
170 PRINT AT(13,10)"AZ 'AT'-"
180 PRINT AT(16,15)"FUNKCIO !"
190 PAUSE 5
200 REM
210 REM ********
220 REM 2. PELDA
230 REM ********
240 REM
250 PRINT CHR$(147):REM KEPERNYO TORLES
260 FOR X=0 TO 24
270 PRINT AT(X,X)"SIMON'S BASIC":
280 NEXT X
290 PRINT AT(0,20)" "
300 PAUSE 5
310 REM
320 REM ********
330 REM 3. PELDA
340 REM ********
350 REM
360 PRINT CHR$(147):REM KEPERNYO TORLES
370 FOR X=0 TO 39
380 PRINT AT(X,12*SIN(X/6)+12) "**"
390 NEXT X
```

Az előbbi példákkal bemutattunk néhány alkalmazási lehetőséget az egyszerű és formált szövegkiviteltől a grafikus felhasználásig.
Fel kell hívnunk a figyelmet arra, hogy az AT utasítással csak a képernyői kivitelt vezérelhetjük. Nyomtatón csak a HRDCOPY utasítással együtt működik (lásd: 12.2.2 alfejezetet).
Közvetlen kinyomtatás esetén a CMD vagy PRINT# kiadásakor az AT utasítás hatástalanná válik és a 2. kl a régi nyomtatási pozíciótól kezdve kerül kiírásra.

9.1.4 LIN – a kurzort tartalmazó sor

Formátum: LIN (függvényként)
Paraméterek: –
Funkció: A kurzor sorának meghatározása
Példa: A = LIN

A LIN azon utasítások egyike, melyek nem egyedül szerepelnek, hanem – a SIN vagy SQR utasításhoz hasonlóan – függvények. Ezekkel további számításokat lehet végezni, vagy értéküket egy változóban tárolni (pl.: A=LIN).
A LIN utasítás segítségével könnyen meghatározhatjuk azt a sort, amelyben a kurzor található.
A POS utasítás alkalmazásával az eredeti Commodore-BASIC lehetővé teszi a kurzor soron belüli helyének meghatározását. A kurzor sorát azonban nem lehet megadni. Ez a hiányosság megszűnik, mert a POS és LIN utasításokkal pontosan meghatározhatjuk a kurzor pontos koordinátáit és számításainkban, képernyőterveinkben felhasználhatjuk. Ügyeljünk a két utasítás eltérő szintaktikájára!

P. 32

```
1 REM **********
2 REM * L I N *
3 REM **********
100 REM
110 PRINT CHR$(147):REM KEPERNYO TORLES
120 A=LIN
130 FOR X=1 TO 20
140 PRINT"EZ A" LIN CHR$(20) ". SOR"
150 NEXT X
160 PRINT
170 PRINT"AZ"A"-"LIN" SOROKBA IRTUNK."
```

Ez a példa csak a LIN utasítás működésének bemutatására szolgál. Használatát programírás közben gyakorolhatjuk be leginkább.

9.1.5 PAUSE – a program késleltetése

Formátum: PAUSE kl,s vagy PAUSE s
Paraméterek: kl – karakterlánc vagy füzérváltozó, amely valamilyen megjegyzést tartalmaz
s – a várakozási idő másodpercben
Funkció: A programfutás késleltetése "s" másodperc időtartamig
Példa: PAUSE "KÉSLELTETÉS",3

Amit egyébként csak körülményesen, késleltető ciklusokkal tudnánk megvalósítani, azt most a PAUSE utasítással egyetlen sorban megoldhatjuk.
Az utasítással a programfutást egy bizonyos ideig – mely időt "s"-sel (egysége a secundum) lehet megadni – felfüggeszthetjük. Ezenkívül a várakozás okát is megadhatjuk.
A program alkalmazása során a késleltetést a ⟨RETURN⟩-nel megszakíthatjuk, és a program folytatódik. Ha a program éppen ezt az utasítást hajtja végre, a ⟨RUNSTOP⟩ billentyűvel nem szakítható meg, csak a RUNSTOP/RESTORE vagy a ⟨RETURN⟩ majd ⟨RUNSTOP⟩ billentyűkkel.
Ez az utasítás jól használható a programok hibajavításánál, pl.: közbenső értékek kijelzésére vagy a nagyon gyors folyamatok lassítására.
A PAUSE mintapélda 4 másodperces időközökben 16-szor változtatja a háttérszínt. A szünetet (várakozási időt) a RETURN-nel megszakíthatjuk.
Jó szórakozást a programozáshoz!

P. 33

```
1 REM **************
2 REM * P A U S E *
3 REM **************
100 REM
110 PRINT CHR$(5):REM FEHER KARAKTEREK
120 FOR X=0 TO 15
130 POKE 53248+33,X:REM HATTERSZIN
140 PRINTCHR$(147)"TOVABB <RETURN>-NEL"
150 PRINT
160 PAUSE "SZIN-SZAM"+STR$(X),4
170 NEXT X
```

9.1.6 Ellenőrző kérdések

Most pedig következik a szokásos teszt. (A megfelelő választ, melyből több is lehet, be kell karikázni.)

1. Mire kell ügyelni a CENTRE-utasítás használatánál?

   a) A CENTRE után nem szerepelhet PRINT utasítás.
   b) PRINT-tel kell elválasztani két CENTRE utasítást.
   c) A beírandó füzér legfeljebb 40 karakteres lehet.

2. Szintaktikusan mely utasítások helyesek?

   a) PRINT CENTRE "ABC"
   b) PRINT AT(2,4) "CBM 64"
   c) PRINT AT(2,4) "CBM 64"
   d) A = LIN * 2
   e) PAUSE 5, "NYOMJA LE A RETURN-T"
   f) USE"###FT, ## F", STR$(123,34)

3. Milyen utasítást kell használni, ha a képernyő közepén akarunk két csillagot megjeleníteni?

   a) PRINT "**"
   b) CENTRE "**"
   c) PRINT AT(19,12)"**"
   d) A$="**":PRINT AT(0,12)"":CENTRE A$

A megoldások a könyv végén találhatók!

9.2 A képernyő villogtatása

A SIMON's BASIC kifejlesztője még olyan szép effektusok létrehozására is gondolt, mint a villogó képernyői kijelzés.
Segítségével lehetőség nyílik a fontosabb információk kiemelésére, de játékként is szolgálhat. Felhasználása a "komoly" programoktól egészen a játékig terjed.

9.2.1 FLASH – egy szín villogtatása

Formátum: sz – a villogó szín kódja (Ø–15)
v – a villogás sebessége (Ø–255)
Funkció: Valamelyik képernyőszín villogtatása
Példa: FLASH 4,10

A FLASH utasítás lehetővé teszi minden olyan karakter folyamatos villogtatását, amelyik egy adott színben jelenik meg a képernyőn, "v" villogási sebességgel. Ez azt jelenti, hogy a karakterek felváltva normál, ill. inverz formában jelennek meg a képernyőn. Az sz a villogtatni kívánt karakterek színét jelenti, értéke Ø és 15 között változhat.
Az egyes értékekhez a következő színek tartoznak.

0 = fekete	8 = narancs
1 = fehér	9 = barna
2 = vörös	10 = világosvörös
3 = türkiz	11 = szürke 1
4 = ibolyakék	12 = szürke 2
5 = zöld	13 = világoszöld
6 = kék	14 = világoskék
7 = sárga	15 = szürke 3

A v sebesség egysége kb. 1/60 másodpercnek felel meg. (1 másodperc pontosan v=58-cal egyenlő.)
A v=0 ugyanaz, mint a v=256. A sebesség beállításához az alábbiakat célszerű figyelembe venni.
Ha v értéke nagyon kicsi (v<10) a villogás túl gyors lesz, a gép túltelítődik és az egész képernyő villogni fog. A BASIC program lassul és ezzel párhuzamosan v értéke nő. A képernyő tehát sohasem fog 1/60 másodperces gyakorisággal villogni.
Problémák adódhatnak a tv-készülékkel vagy a monitorral is, mivel a számítógép a filmfelvevőhöz hasonlóan 1/20 másodpercenként állít elő új képet. A gyakorlás érdekében próbáljuk ki a fellépő effektusokat (sávok és megszakítások).
A FLASH utasítás használatánál még egy fontos apróságra oda kell figyelnünk.
Ha a villogtatás bekapcsolása után a kérdéses színnel újabb szöveget írunk a képernyőre, előfordulhat, hogy az részben vagy egészben fordítva villog, vagyis az eredeti szöveg normál állapotánál inverz és fordítva. Ennek az az oka, hogy a számítógép függetlenül a villogási helyzettől, normál üzemmódban (ill. a RVS/ON aktiválása után inverzben) ír és ha az átfordítás pillanatában a régi szöveg éppen nem a normál ábrázolási módban jelenik meg, előáll az említett fordított villogás. Ugyanez az oka annak, hogy a FLASH bekapcsolását követően az inverz feliratok ellentétes fázisban villognak.

9.2.2 OFF – a villogás kikapcsolása

Formátum: OFF
Paraméterek: –
Funkció: A FLASH üzemmód kikapcsolása
Példa: OFF

A FLASH-sel bekapcsolt villogás az OFF utasítással szüntethető meg. Ilyenkor mindig a pillanatnyi villogási helyzet állandósul. Tehát ha a villogtatott karakterek éppen inverz

állásban jelennek meg, az OFF végrehajtása után így is maradnak, és fordítva. Ezért ha az OFF utáni állapotot rögzíteni kívánjuk, akkor az a legjobb, ha töröljük a képernyőt a CHR$(147)-tel és utána újraírjuk.
Alapszabály, hogy minden FLASH után következzen valahol – de legkésőbb a program végén – egy OFF utasítás, különben a képernyő a program befejezése után tovább villog.
Példa a FLASH–OFF utasításpárra:

P. 34

```
1 REM ************
2 REM * F L A S H *
3 REM ************
100 REM
110 POKE53248+33,0:POKE53248+32,0:REM HATTER-/KERETSZIN
120 PRINTCHR$(147)CHR$(158):REM KEPERNYOTORLES +SZIN=SARGA
130 CENTRE "VARJON AMIG A READY MEGJELENIK":PRINT
140 CENTRE "VAGY TARTSA LENYOMVA A RETURN-T"
150 PRINT AT(0,8) "":REM KURZOR POZICIONALAS
160 CENTRE "########################":PRINT
170 CENTRE "########################":PRINT
180 CENTRE "###                  ###":PRINT
190 CENTRE "###                  ###":PRINT
200 CENTRE "###                  ###":PRINT
210 CENTRE "###                  ###":PRINT
220 CENTRE "########################":PRINT
230 CENTRE "########################":PRINT
240 PRINT AT(0,10)CHR$(31):REM KEK
250 CENTRE"                    ":PRINT
260 CENTRE"  COMMODORE 64     ":PRINT
270 CENTRE"  SIMON'S BASIC    ":PRINT
280 CENTRE"                    ":PRINT
290 FOR X=10 TO 1 STEP -1
300 FLASH 6,X:REM VILLOGO KEK SZIN
310 PAUSE 2
320 NEXT X
330 OFF
```

Példaprogramunk a FLASH és OFF, valamint az egyéb kiíró utasítások együttes használatát mutatja be. Miután a képernyő és a keret feketére vált, megjelenik egy sárga keret (160–230. sor) közepén kék felirattal (260–270. sor). A felirat azonnal villogni kezd, a villogás egyre gyorsul, majd a legnagyobb villogási sebességnél megszűnik.
Ugye látványos?

9.2.3 BFLASH – a keret villogtatása

Formátum: BFLASH V,SZ1,SZ2
Paraméterek: V – a villogás sebessége
SZ1/SZ2 – a villogásban egymást váltó két szín
Funkció: A képernyőkeret színének változtatása a két szín között
Példa: BFLASH 30,12,14

A BFLASH utasítás segítségével különböző villogási effektusok állíthatók elő, melyek pl. a számítógépes játékok különleges helyzeteit (találat, győzelem stb.) kiemelik vagy a figyelmet egyszerűen a képernyő tartalmára irányítják.
A BFLASH a keret színét váltogatja meghatározott gyakorisággal anélkül, hogy beavatkoznánk (lásd FLASH). A v paraméter itt is a frekvencia beállítását szolgálja. A paraméter jelentése, számítása megegyezik a FLASH utasításnál leírtakkal. A másodpercenkénti színváltáshoz v-t 60-ra kell beállítani. (Egy megjegyzés: az előbbi szám csak irányérték, a valóságban v=58 esetén vált másodpercenként a képernyő.) Mivel azonban a keret színcseréje egy egyszerű POKE utasítással történik, a színváltás nagyon gyors is lehet ($0<v<10$). Tehát a FLASH-sel ellentétben itt tényleg megvalósulhat az 1/60 másodpercenkénti váltás. A tv-kép azonban csak 1/20 másodpercenként vált, így nagyon érdekes sáveffektusokat idézhetünk elő.

A két további változó a villogásban egymást váltó két színt jelöli. A FLASH utasításnál megismert táblázatot kell használni.
Itt a következőkre kell figyelni:
Ha a két szín erős kontrasztkülönbséggel (pl. fekete-fehér) változik, akkor ez kihat a képernyő belső részére is.
A világos szín elvesz a képernyő színintenzitásából, míg a sötét szín felerősíti azt. Ezáltal a háttér együttvillogását figyelhetjük meg, különösen fényszegény (pl.: fekete) háttérszín esetén.
Ezt az esetet tanulmányozhatjuk a BFLASH 0 utasítás 1 példaprogramján.

9.2.4 BFLASH 0 – a villogás kikapcsolása

Formátum: BFLASH 0
Paraméterek: –
Funkció: A BFLASH üzemmód kikapcsolása
Példa: BFLASH 0

Funkciója megegyezik az OFF utasításéval, de a FLASH utasításra vonatkozóan: kikapcsolja a keret villogását. Elég bizonytalan, hogy a két szín közül melyik állandósul, ezért ajánlatos a keretszínt a POKE 53248+32,sz paranccsal újra megadni (sz=színkód).
A BFLASH a programban bárhol szerepelhet, de legkésőbb a program végén be kell hívni, különben a FLASH utasítás a program befejezése után is érvényben marad.
Példák a BFLASH és a BFLASH 0 utasításokra:

1 Példa:
Helyettesítsük a FLASH-program 300. és 330. sorát a következőkkel:

```
300 BFLASH X,6,7:REM KÉK-SÁRGA VÁLTAKOZÁS
330 BFLASH 0
```

2 Példa:

P. 35

```
1 REM ***************
2 REM * B F L A S H *
3 REM ***************
100 REM
110 POKE 53248+33,2:REM VOROS HATTER
120 BFLASH 1,7,6:REM VALTAKOZO SZINEK=SARGA-KEK
130 PRINTCHR$(5)CHR$(147):REM FEHER KARAKTEREK + KEPERNYO TORLES
140 FOR X=0 TO 39
150 Y=X/2
160 PRINT AT(X,Y)"*":REM EGYIK ATLO
170 PRINT AT(39-X,Y)"*":REM MASIK ATLO
180 PRINT AT(X,0)"*":REM FELSO VIZSZINTES
190 PRINT AT(X,20)"*":REM ALSO  VIZSZINTES
200 PRINT AT(0,Y)"*":REM BAL FUGGOLEGES
210 PRINT AT(39,Y)"*":REM JOBB FUGGOLEGES
220 NEXT X:REM KOVETKEZO PONT
230 FLASH 1,15
240 PRINT AT(3,0)CHR$(150):REM POZICIONALAS ES SZOVEG;VILAGOSVOROS
250 CENTRE "CSODALKOZIK?":PRINT
260 CENTRE "SEMMISEG!"
270 PRINT AT(0,20)"":REM 'READY.'POZICIONALASA
280 PAUSE 6:REM 6 MASODPERC VARAKOZAS
290 BFLASH 0
300 OFF:REM VILLOGAS KIKAPCSOLASA
```

A fenti példán tanulmányozhatjuk a FLASH, BFLASH, OFF BFLASH 0 és néhány egyéb SIMON's BASIC utasítás lehetséges kölcsönhatását. Próbáljuk meg megfejteni az egyes programrészek működését. A különböző utasítások megértéséhez a 140–220. sorokra nincs feltétlenül szükség. A programmal könnyen játszadozhatunk, ha egyes paramétereit változtatjuk. Próbáljunk ki mindent nyugodtan.

9.2.5 Ellenőrző kérdések

Végül nézzük, mit tanultunk:

1. Melyik utasítást használjuk, ha a keretszínt fehér és kék között, másodpercenként kb. egyszer kell változtatni?

   a) BFLASH 0
   b) FLASH 0,1
   c) BFLASH 60,1,0
   d) BFLASH 0,1,1

2. Be akarjuk fejezni a vörös karakterszín villogását. Melyik utasítást kell választanunk?

   a) BFLASH 0,0,0
   b) BFLASH 0
   c) OFF
   d) FLASH 0

3. Mit tegyünk, ha a szövegek csak egy bizonyos részét akarjuk a FLASH utasítással villogtatni?

   a) Először leírjuk a villogtatandó szöveget, bekapcsoljuk a villogást, végül beírjuk a hátralévő nem villogó szöveget.
   b) Pontosan fordított sorrendben járunk el.
   c) A villogtatni kívánt szöveget más színnel írjuk be, mint a stabilt, majd az első szöveg színét villogtatjuk.
   d) Mindkét szöveget ugyanazzal a színnel írjuk be, de a FLASH 12,40 utasítással csak a képernyő azon felső részét villogtatjuk, ahol az első szöveg elhelyezkedik.

9.3 Háttér- és keretszín

Az alábbiakban néhány nem mindennapi felfedezésről számolunk be:
A SIMON's BASIC nemcsak az eredeti kézikönyvben szereplő utasításokkal rendelkezik, hanem néhány olyannal is, melyeket az eredeti kézikönyv meg sem említ (lásd: függelék). Hét ilyen utasítást találtunk, de ezek közül csak három működőképes. A fennmaradó négy bizonyára nem lett kész időre (az ilyen programok előállítása sokszor határidő kérdése). Azt, hogy ezeket az utasításokat a meghirdetett modulváltozatban milyen mértékben vették figyelembe, természetesen még nem tudjuk. Az viszont biztos, hogy további három olyan utasítás áll rendelkezésünkre, amelyek a képernyőn való megjelenítésre vonatkoznak.
Az utasítások leírása előtt néhány dolgot meg kell magyarázni:

Az EXTENDED COLOUR (kiterjesztett szín) üzemmód:

A Commodore 64-es az ismert normál szövegüzemmód (egy háttérszín minden karakterhez és 15 karakterszín) mellett rendelkezik még egy további szövegüzemmóddal is, melynek keretében minden karakterhez külön háttérszínt (összesen négy háttérszín regiszter van, tehát négy választható háttérszín áll rendelkezésre) definiálhat. Ezt az ábrázolásmódot "extended colour" üzemmódnak hívjuk.
Az előbbiek értelmében minden karakterhez négy háttérszínből választhatunk, melyek a megfelelő utasításokkal vagy a mindenkori érték megfelelő regiszterekbe való beírásával (POKE segítségével) rögzíthetők:

0. háttérszín ..... POKE 53248+33,hsz0
1. háttérszín ..... POKE 53248+34,hsz1
2. háttérszín ..... POKE 53248+35,hsz2
3. háttérszín ..... POKE 53248+36,hsz3

Látható, hogy a 0-regiszter azonos a normál háttérszín kijelölésére szolgáló regiszterrel. A "hsz" a mindenkori színkód, amelyet a függelékben találhatunk meg. Azt, hogy egy jelhez milyen háttérszín tartozik, a video-RAM byte-jainak felső két bitje határozza meg. A video-RAM vagy képernyőtároló, a szövegablak karaktereinek kódját tartalmazza.
Mivel a két bit normál körülmények között szintén részt vesz a karakterek kódolásában, az "extended colour"-üzemmódban csak 64 karakter áll rendelkezésünkre. Ilyenkor a grafikus karaktereket, valamint kis-/nagybetűs üzemmódban a nagybetűket és az inverz karaktereket nem tudjuk használni.
Amennyiben ezeket mégis behívjuk, akkor az engedélyezett karakterek valamelyike más háttérszínt kap. Azt, hogy ez melyik karaktert, és mi módon érinti, egyszerűen megtudhatjuk a képernyőkódok táblázatából (CBM 64 felhasználói kézikönyvből).
Itt az alábbi hozzárendelés van érvényben:

MSBs	...képernyőkódok...	Háttérszínek
00	000–063	HSZ0
01	064–127	HSZ1
10	128–191	HSZ2
11	192–255	HSZ3

MSBs alatt a képernyőtároló byte-jainak felső két (6. és 7.) bitjét értjük.
A képernyőkódok mindig decimális értékek. Próbaképpen vessük össze a fenti és a kézikönyvben megadott képernyőkód táblázatot.
Például ha a B betűt a 7-es (sárga) háttérszínnel akarjuk kijeleztetni, akkor az ehhez tartozó háttérszín-regiszterbe (itt az 1-esbe) a 7-es értéket vigyük be, majd a POKE utasítással a képtároló megfelelő regiszterébe a 2-est (B-betű), és – az 1-es regiszter vezérléséhez – a "64" értéket.
Az is egy lehetséges megoldás, hogy a SHIFT/B karaktert, azaz a CHR$(98)-at PRINT utasítással visszük ki a képernyőre, a program így néz ki:

```
10 BCKGNDS 2,3,4,5:REM EXTENDED COLOUR-ÜZEMMÓD BEKAPCSOLÁSA
20 POKE 53248+34,7:REM 1.HÁTTÉRSZÍN=SÁRGA
30 POKE 1024,2+64: REM KARAKTER A BAL FELSŐ SAROKBAN
```

vagy

```
100 BCKGNDS 2,3,4,5:REM EXTENDED COLOUR-ÜZEMMÓD BEKAPCSOLÁSA
110 POKE 53248+34,7:REM 1. HÁTTÉRSZÍN = SÁRGA
130 PRINT CHR$(98) :REM KARAKTER A KURZORPOZÍCIÓBAN
```

A program lefuttatása után tovább kísérletezhetünk, mert az üzemmód érvényben marad. Ezek után térjünk rá a bizonyára türelmetlenül várt utasításokra.
Ha az eddig leírtak nem elég világosak, az első mintaprogramot egészítsük ki a következő utasításokkal:

```
40 POKE 1027,2
50 POKE 1030,128+2
60 POKE 1033,192+2
```

Mielőtt a programot elindítjuk, töröljük a képernyőt. Ezután változtassuk a 20. sorban a 34-es értéket 33-, 35-, 36-ra, majd módosítsuk többször a színkódot is. A látottak alapján érthetőbbé válik a két táblázat közötti összefüggés. (Magyar változat készítői.)

9.3.1 COLOUR – a keret és háttér színe

Formátum: COLOUR ksz, hsz
Paraméterek: ksz – keretszín (0–15)
hsz – háttérszín (0–15)
Funkció: A keret- és a háttérszín meghatározása
Példa: COLOUR 6,7

Az előzőekben már említett három új utasítás közül az első lehetővé teszi, hogy egyszerű módon változtathassuk a keret és a háttér színét. Ennek megfelelően nem kell a POKE utasításokat használni, melyek funkcióját (feltéve ha nem fűzünk hozzá megjegyzést a REM utasítással) könnyen elfelejthetjük.
Az első paraméter: ksz, a külső képkeret színét rögzíti. Értéke a keret színkódját tartalmazza.
A bevitt kódot a megfelelő regiszter automatikusan tárolja. A keretszín kiválasztására a BFLASH-hoz hasonlóan itt is különös figyelmet kell fordítani, mivel a túl világos színek elveszik a fényerőt a képmezőtől, ami a szöveg olvashatatlanságáig fokozódhat. Jól szemlélteti ezt a példa. A COLOUR funkciójának ez a része a következő POKE utasításnak felel meg:

```
POKE 53248+32,hsz
```

A "hsz" paraméter a szöveg háttérszínét jelöli. Ez természetesen csak szövegüzemmódban érvényes. A háttérszín beállítására a grafikus üzemmódban más utasítások vonatkoznak [többszínű üzemmódban (lásd később) a POKE utasításon kívül ez az egyetlen eszköz a háttérszín megválasztására]. A keretszínt ezzel az utasítással a grafikus üzemmódban is tudjuk módosítani. A háttérszínek rögzítésekor lehetőleg olyan színeket válasszunk ki, melyek a szövegszínnel jó kontrasztot adnak, mert ezzel elkerüljük a szürkés színhatás (pl. kék alapon vörös) kialakulását.
Ez az utasítás POKE-kal így írható fel:

```
POKE 53248+33,hsz
```

Összességében tehát a COLOUR az alábbi két utasítással helyettesíthető:

POKE 53248+32,ksz

POKE 53248+33,hsz

ami jól szemlélteti a COLOUR előnyét.

Példa:

P. 36

```
1 REM ***************
2 REM * C O L O U R *
3 REM ***************
100 REM
110 PRINT CHR$(147):REM KEPERNYO TORLES
120 PRINT CHR$(31):REM SZOVEGSZIN=KEK
130 PRINT AT(0,8)"":REM MEGHATAROZASA
131 :                  REM ES A KURZOR POZICIONALASA
140 CENTRE "**********************":PRINT
150 CENTRE "**                    **":PRINT
160 CENTRE "**   KERETSZIN        **":PRINT
170 CENTRE "** VALTOZTATASA       **":PRINT
180 CENTRE "**                    **":PRINT
190 CENTRE "**********************":PRINT
200 REM
210 FOR KSZ=0 TO 15
220 COLOUR KSZ,0:REM HATTERSZIN=FEKETE;KERETSZIN VALTOZIK
230 PAUSE 1
240 NEXT KSZ
250 REM
260 PRINT AT(0,10) "":REM KURZOR POZICIONALASA
270 CENTRE " HATTERSZIN ":PRINT
280 REM
290 FOR HSZ=0 TO 15
300 COLOUR 0,HSZ:REM KERETSZIN=FEKETE;HATTERSZIN VALTOZIK
310 PAUSE 1
320 NEXT HSZ
```

9.3.2 BCKGNDS – az EXTENDED COLOUR bekapcsolása

Formátum: BCKGNDS hsz0, hsz1, hsz2, hsz3

Paraméterek: – hsz0 : 0. háttérszín
– hsz1 : 1. háttérszín
– hsz2 : 2. háttérszín
– hsz3 : 3. háttérszín

Funkció: Az EXTENDED COLOUR üzemmód bekapcsolása és a 4 háttérszín meghatározása

Példa: BCKGNDS 3,4,5,6

Az utasítás tanulmányozása előtt feltétlenül el kell olvasni a 9.3 ponthoz tartozó bevezetést, mivel az ott leírt ismeretekre az alábbi összefüggések megértéséhez mindenképpen szükségünk lesz.

Az utasítás olyan lehetőséget kínál, melyet eddig még nem próbáltunk a számítógépünkkel, nevezetesen az EXTENDED COLOUR üzemmódot. Egyébként a CBM 64 lehetőségeit ismerve, még egy egész halom, ez ideig programokban csak igen ritkán használt funkció található. A kibővített üzemmód fogalmát és tulajdonságait részletesen tárgyaltuk az említett bevezetésben. Ennek elolvasása után megtudhattuk, hogy ilyenkor a képernyő valamennyi karakteréhez egyedi háttérszín rendelhető úgy, hogy az illető háttérszín származási regiszterét megadjuk. Egyidejűleg 64 karakterre csökken a karakterkészlet. A BCKGNDS utasítással szöveges üzemmódban ez az állapot rendelkezésre áll és a négy regiszterhez hozzárendelhetők a kívánt színek.

A négy szín kiválasztására a hsz0, hsz1, hsz2, hsz3 paraméterek szolgálnak, melyekbe tetszés szerinti színkód írható. Ha pl. hsz0 7-tel egyenlő, akkor a 0. háttérszín regiszter

sárga színt kap, ami azt jelenti, hogy a normál körülmények között bevitt karakterek is ezen a háttérrel fognak megjelenni.
A leírtakat két példaprogrammal szemléltetjük:

P. 37

```
1 REM ********************
2 REM * 1.B C K G N D S *
3 REM ********************
100 REM
110 PRINT CHR$(147):REM KEPERNYO TORLES
120 PRINT CHR$(5):REM SZOVEGSZIN=FEHER
130 COLOUR 0,0:REM KERETSZIN=FEKETE
140 BCKGNDS 2,3,6,7:REM EXTENDED COLOUR-UZEMMOD FEKAPCSOLASA
150 REM ES A HATTERSZINEK MEGHATAROZASA
160 FOR X=0 TO 255
170 POKE2*X+1024,CODE:REM KARAKTER BEIRASA
180 POKE2*X+55296,0:REM KARAKTERSZIN A  SZINRAMBA (FEKETE)
190 CODE=CODE+1:REM KOVETKEZO KEPERNYOKOD
200 NEXT X
210 PRINT AT(0,15)"":REM A 'READY.' ES A KURZOR POZICIONALASA
```

Példaprogramunk az összes lehetséges karaktert kijelzi azzal a négy háttérszínnel, amelyeket előzőleg a BCKGNDS utasítással előírtunk. A 170. és 180. sorok részletes ismertetése az FCHR utasítás funkciójának leírásánál található.
A 190. sorban folyamatosan növelt képernyőkódot a 170. sor elhelyezi a képtároló egyik regiszterében. A 180. sor láthatóvá teszi a karaktert, azzal, hogy a színt elhelyezi a szín-RAM megfelelő rekeszében.
Próbáljuk ki az egyes háttérszíneket a BCKGNDS paramétereinek változtatásával.

Figyeljük meg, hogy a P. 38-as program listája kisbetűvel készült.

Ennek oka a következő:
A Commodore 64 alternatív karakterkészletét használtuk, vagyis ami egyébként nagybetűvel íródik ki, az most kisbetűs, továbbá minden SHIFT-elt grafikus karakterből nagybetű lesz. Azért kellett ezt a kiviteli módszert választanunk, mert a háttérszínek lehívásakor ezekre a SHIFT-elt karakterekre szükségünk van. Amennyiben a programlemez nem áll rendelkezésre és a programot be kell billentyűznünk, előtte át kell állítanunk a készüléket nagy-/kisbetűs üzemmódra. Ezt a SHIFT/C=–billentyűk együttes lenyomásával tehetjük. Ha kísérletünk hatástalan, írjuk be a PRINT CHR$(9) parancsot és próbáljuk meg újra az átkapcsolást. Az EXTENDED COLOUR üzemmódban minden másképpen néz ki.
Még valami a magyarázathoz. Minden normál karakter (a nem SHIFT-elt kisbetűk (CHR$(32)-től CHR$(95)-ig) a megszokott módon a 0. háttérszínnel jelenik meg.
Ezzel szemben a SHIFT- vagy a C=–billentyűvel kijelzett karakterek (CHR$(96)-tól CHRS(191)-ig, kivéve a CHR$(125) és CHR$(159) közöttieket) az 1. háttérszínt kapják.
Az alábbi leírás alapján megtudhatjuk, hogy az egyes billentyűk melyik karakternek felelnek meg:

1) Keressük ki a képernyőkódok táblázatából (lásd: kézikönyv) a szükséges karaktert. A karakter kódja azonban nem lehet 63-nál nagyobb, mert nem lehet kijelezni.
2) A leolvasott kódértékhez adjunk 64-et, és ugyanebben a táblázatban nézzük meg, hogy az új kódhoz melyik karakter tartozik.
3) Ezek után a kézikönyv ASCII-táblázatából keressük ki a karakterhez tartozó ASCII-értéket, melyet most a kívánt karakter 1. (vagy 3.) háttérszínen történő megjelenítéséhez meg kell adnunk.

A fenti komplikált műveletsor nélkül is előállíthatjuk a megfelelő háttéren a kívánt *betűt,* ha egyidejűleg a SHIFT-billentyűt is lenyomjuk.
Számok és *írásjelek* esetében viszont kizárólag a bonyolultabb mód alkalmazható (lásd a 2. példa P. 38-as program 470. sorát).
Ha a 2. háttérszínre kívánunk írni, akkor ugyanúgy kell aljárni, mint a 0. szín esetében, azzal a különbséggel, hogy inverz-üzemmódban (az RVS/ON lenyomása után) kell a

P. 38

```
1 rem ********************
2 rem * 2.b c k 9 n d s *
3 rem ********************
150 rem
160 Printchr$(147):rem kePernyo torles
170 Printchr$(5):rem szove9=feher
180 colour 0 0:rem keret=fekete
190 bck9nds 2,5,6,0 :rem extended colour-uzemmod bekaPcsolasa
200 rem es a hatterszinek me9hatarozasa
210 rem normal karakterek kovetkeznek
220 centre "lathatjuk, ho9y az ":Print
230 centre "extended colour-uzemmodban":Print
240 centre "szabalyosan lehet irni!":Print
250 Pause 9
260 rem
270 rem a kovetkezo karakterek shift-eltek(<shift><c>-vel lathatok)
280 Printchr$(147)
290 Print
300 centre "A HATTERSZINEK":Print
310 centre "SZELES":Print
320 centre "SKALAJA":Print
325 centre "ALL RENDELKEZESUNKRE.":Print
330 Pause 9
340 rem
350 Printchr$(147)
360 Print
370 rv$=chr$(18):rem rvs/on a hatterszinre
380 rem a kovetkezo karakterek inverzen,<shift> nelkul jelennek me9
390 centre rv$+"minden karakterhez":Print
400 centre rv$+"4 szinbol valasztott":Print
410 centre rv$+"hatteret rendelhetunk.":Print
420 Pause 9
430 rem
440 Printchr$(147)
450 Print
460 rem a kovetkezo karakterek inverzben es shift-tel jelennek me9
470 ko$=chr$(172):rem vesszokod+64 hf3-raa <c>d karaktert adja
480 centre rv$+"ARRA UGYELJUNK,":Print
490 centre rv$+"HOGY GRAFIKUS karakterek":Print
500 centre rv$+"A KIVITELBEN NEM LEHETNEK !":Print
510 Pause 9
520 rem
530 Printchr$(147)
540 Print
550 centre "termeszetesen me9Probalhatunk"+rv$+" MINDEn":Print
560 Print
570 ro$=chr$(146):rem rvs/off
580 h$="hA"+rv$+"tT"+ro$+"eR"
590 h$=h$+rv$+"sZ"+ro$+"iN"+rv$+"t":rem szoosszetetel
600 Print at(15,4)h$
610 Print
620 centre "e9yidoben "+rv$+"ABRAZOLNI.":Print
630 Pause 15
640 rem
650 Printchr$(147)
660 Print
670 centre "nezzuk me9 me9e9yszer":Print
680 centre "a most kovetkezo Pro9ramlistat":Print
690 centre "ebben az uzemmodban, mi a ma9yarazat?":Print
700 Pause 9
710 rem
720 list
```

karaktereket bevinni. A 3. háttérszín az 1. színhez alkalmazott karakterek inverz bevitelével állítható elő. Az alábbi módon történik pl. a "9"-es számjegy és a "c" betű megjelenítése a négy háttérszínen.

```
0. háttérszín   PRINT "9"
                PRINT "c"
1. háttérszín   PRINT CHR$(185)
                PRINT "C"
2. háttérszín   PRINT CHR$(18)"9"
                PRINT CHR$(18)"c"
3. háttérszín   PRINT CHR$(18) CHR$(185)
                PRINT CHR$(18)"C"
```

Ha ezek után sem teljesen világos még a karakterek ábrázolásának ez a módja, nem kell elkeseredni. Haladjunk tovább és a későbbiekben (pl. a grafikus gyakorlatokat követően vagy a másodszori olvasás alkalmával), a számítógép alaposabb ismeretével mégegyszer visszatérhetünk ehhez a részhez.
Egyet nem szabad elfelejteni, miszerint próba és állandó gyakorlás nélkül nem megy. Minden egyes leírt példán át kell, hogy rágjuk magunkat, mivel az azt követő magyarázatok erre épülnek.

9.3.3 NRM – az EXTENDED COLOUR kikapcsolása

Formátum: NRM
Paraméterek: –
Funkció: Az EXTENDED COLOUR üzemmód kikapcsolása
Példa: NRM

Az EXTENDED COLOUR üzemmód eddig ismertetett utasítás készletéből már csak egyetlen utasítás hiányzik, mégpedig az üzemmód kikapcsolása.
Ezt a feladatot az eddig ismeretlen NRM utasítás végzi, melynek egyetlen funkciója az üzemmód kikapcsolása, az egyeduralkodóvá váló 0. háttárszín meghagyásával. (Ennek módosításához egy COLOUR utasítás szükséges).
A későbbiek során néhányszor még fogunk találkozni az NRM-mel és megismerhetjük további funkcióit, melyekről itt csak röviden szólunk.

1) A grafikus üzemmód kikapcsolása.
2) A MEM üzemmód kikapcsolása (visszaállás a régi karakterkészletre).
3) A nagybetű–/grafikus jelkészlet üzemmód bekapcsolása.

Példa:
Az NRM utasítást írjuk be a P.38-as 2.BCKGNDS című program végére.

9.3.4 Ellenőrző kérdések

1. Melyik utasítás alkalmas arra, hogy a háttérszínt fehérre, a keretszínt pedig sárgára változtassa?

   a) COLOUR 1,7
   b) COLOUR 7,1
   c) BCKGNDS 1,7
   d) POKE 53248+33,1:POKE 53248+32,7

2. Az „A"-karaktert a 2. háttérszínnel kívánjuk megjeleníteni. Hogyan csináljuk?

   a) PRINT CHR$(65)
   b) PRINT CHR$(97)
   c) PRINT CHR$(18) CHR$(65)
   d) PRINT CHR$(18) CHR$(97)

A mostani (egyáltalán nem könnyű) feladatok megoldásai az eddigiekhez hasonlóan a függelékben találhatók.

9.4 Képernyőtartományok vezérlése

Ez a fejezet a nagyobb képernyõtartományokkal végezhetõ mûveletekkel foglalkozik. Az ismertetésre kerülõ utasításokkal gyorsan és hatékonyan kezelhetjük a képernyõt.
Az alábbiak megértéséhez bizonyos alapismeretekkel kell rendelkezni a számítógép képalkotásáról. Elevenítsük fel eddigi ismereteinket a képernyõ-ablak hardver-jét illetõen.
A számítógép az ún. képtárolóban tárolja azokat az információkat, amelyek segítségével minden 1/20 másodpercben új képet állít elõ. A képtároló 1K nagyságú, ami kb. 1000 byte vagyis 1000 karakter. A memóriában a $400 (decimálisan: 1024) és a $7E7 (decimálisan: 2023) címek között helyezkedik el.
A képernyõn megjelenõ minden egyes karakternek külön kódja van, ugyanúgy mint az ASCII kódoknál. A képernyõkód és az ASCII kód sajnos nem egyezik meg.
A képernyõkódoknál szintén megkülönböztetjük a normál és inverz karaktereket, hiszen ha nem így lenne, a gép nem tudná külön megjeleníteni õket.
A képernyõkodok részletesen a felhasználói kézikönyvben találhatók. Az inverz karakterek kódjának kiszámítása a következõ:

```
inverz kód = normál kód + 128
```

Példa:

```
10 PRINT CHR$ (147)CHR$(158): REM KÉPERNYÕ TÖRLÉS,SZIN=SÁRGA
20 FOR X=1 TO 100
30 PRINT " " ;
40 NEXT X
50 FOR X=1024 TO 1124
60 POKE X,1 : REM AZ "A"-BETÜ KÓDJÁNAK BEVITELE A KÉPERNYÕTÁROLÓBA
70 NEXT X
```

Példaprogramunk a képernyõ bal felsõ sarkától kezdve folyamatosan beír 100 „A" betût. A 10.–40. sorokban a karakterek háttérszínét határoztuk meg, majd erre az elõkészített 100 helyre beírtuk az „A"-betûket. Az „A" képernyõkódja 1. Ezzel áthidaltunk egy problémát, amelyre a következõ bekezdésben térünk ki.
A képi megjelenítéshez a karakterek színét is meg kell határozni, hiszen elméletileg minden karakter más és más színben jelenhet meg. A színek tárolására egy újabb tároló szolgál, amelyet szín-RAM-nak nevezünk. Hossza megegyezik a képtároló hosszával és a $D800–$DFFF (decimálisan 55296–56295) címek között helyezkedik el. A szín-RAM is POKE utasításokkal érhetõ el, és minden egyes byte-ja a képtárolóban a hozzá tartozó karakter színét rögzíti.
A szövegmezõhöz tartozó háttér színét viszont a VIDEOCHIP egyetlen regisztere tárolja. A VIDEOCHIP vagy rövidebben VIC a számítógép képelõállításáért felelõs áramköre. Az említett regiszter az 53248+33 memóriacímen van, és pl. a

```
POKE 53248+33,0 : REM HATTER = FEKETE
```

utasítással érhetõ el. A témáról részletesebb információkat a grafikánál közlünk.
Az alábbi jelenségre azonban még fel kell hívi a figyelmet:
Ha egy programban töröljük a képernyõt, a képtárolón kívül a szín-RAM is törlõdik. A szín-RAM minden byte-ja a háttérszín kódját veszi fel. Ezért bevitelkor nem látunk semmit mert a karakterek a háttérszínt veszik fel. A karakterek egyébként léteznek. Erről úgy gyõzõdhetünk meg, hogy a kurzort végigvezetjük a képernyõn, vagy egyszerûen megváltoztatjuk a háttérszínt. Ha a karaktert a kívánt színben akarjuk megjeleníteni, a színkódokat be kell vinni a szín-RAM megfelelõ címére.

Példa:

```
10 PRINT CHR$(147) : REM KÉPERNYŐ TÖRLÉS
20 POKE 1024,2 : REM "B"–BETŰ A KÉPERNYŐTÁROLÓBA
30 STOP: REM ELHAGYHATÓ
40 POKE 55296,0 : REM KARAKTER = FEKETE
```

Először futtassuk a programot a 30. sorral együtt, majd a program befejezésekor vigyük a kurzort a bal felső sarokban, ahol a „B" betűt találjuk. Most töröljük ki a 30. sort és indítsuk újra a programot. Ezután már értéket kap a szín-RAM megfelelő byte-ja, és megjelenik a fekete „B" betű.
Így már megfelelő alapismeretekkel rendelkezünk a következő utasítások megértéséhez.

9.4.1 FCHR – feltöltés azonos karakterekkel

Formátum: FCHR s, o, sz, m, k
Paraméterek: s – a képmező bal felső sarkának sora (0–24), és
o – oszlopa (0–39)
sz – a mező szélessége, és
m – magassága
k – az aktíválandó képernyőkód (0–255)
Funkció: Egy képernyőtartomány ugyanazon karakterekkel való feltöltése
Példa: FCHR 3,5,7,9,1

Az FCHR-utasítás segítségével a képernyő meghatározott tartománya egy k-képernyőkódú karakterrel tölthető meg. Az utasítás a FILL CHARACTERS kifejezés rövidítése. A tartomány csak téglalap alakú lehet. Az utasítás első két paramétere a téglalap helyzetét, míg 3. és 4. paramétere a nagyságát adja meg.
A téglalap helyzetét bal felső sarkának koordinátáival difiniáljuk, ahol s a sort, o pedig az oszlopot jelenti. A m- és sz-paraméterek a téglalap magasságát és szélességét határozzák meg. Értékeik karakter egységekben értendők és a téglalap helyzetét adó s és o koordinátáktól számítjuk. Ebből következik, hogy nem lehetnek nagyobbak, mint a kezdőpont és a képernyő széle közötti különbség. Ha tehát a téglalap bal felső sarka a 10. sorban és 10. oszlopon helyezkedik el, akkor m értéke max. 14 lehet, mivel a képernyő aljáig ennyi sor van még. Az sz értéke viszont legfeljebb 29 lehet, mert a képernyő jobb széle ilyen távol van a kezdőponttól. Ha a megengedettnél nagyobb értékeket adunk meg, akkor a téglalap "kilóg" a képernyőből.
Ha valamelyik paraméternek 0 értéket adunk, akkor a „BAD MODE" hibajelzést kapjuk, amivel a későbbiek során foglalkozunk.
Az FHCR utasítás a következő BASIC sorokkal helyettesíthető:

```
170 FOR Y=S TO M : REM SOR-CIKLUS
180 FOR X=O TO SZ : REM OSZLOP-CIKLUS
190 POKE 1024+40*X,K : REM KARAKTEREK BEÍRÁSA
210 NEXT X,Y : KÖVETKEZŐ OSZLOP/SOR
```

A program S,M,O,SZ,K paraméterei megegyeznek az FHCR utasításnál leírtakkal (sor, magasság, oszlop, szélesség, képernyőkód).
A 190. sor az aktuális pozícióból számítja ki a szükséges címet, a soronkénti lehetséges 40 karaktert feltételezve, majd a kapott eredményhez hozzáadja a tört sor karakterszámát és megkapja a karakter címét. A cím számításához mindig ezt a képletet használjuk, ha a képtárolóval dolgozunk.

```
cim = 1024 + 40 * sor + oszlop
```

ahol: 1024 a képtároló kezdőcíme.
A műveleti sebességek összehasonlítására lássunk egy példát:

P. 39

```
1 REM ************
2 REM * 1.F C H R *
3 REM ************
100 REM
110 REM MINTAPELDA 'FCHR' NELKUL
120 REM
130 PRINT"■"
140 S=1:M=23:O=2:SZ=35:K=1:REM PARAMETEREK MEGADASA
150 FOR Y=S TO  M:REM SOR-CIKLUS
160 FOR X=O TO SZ:REM OSZLOP-CIKLUS
170 POKE 1024+40*Y+X,K:REM KARAKTEREK BEIRASA
180 REM POKE 55296+40*Y+X,4:REM SZIN BEIRASA
190 NEXT X,Y:REM KOVETKEZO SOR/OSZLOP
```

Ha a programunkat lefuttatjuk, láthatjuk, hogy nem sokat csinál.
Ezt a karakter- és színtároló szétválasztása okozza, amelyről a bevezetőben már beszéltünk. Ezért a 180. sorba a REM után tettünk egy kiegészítést. Ez eddig hatástalan volt, így a karakterek nem színeződtek el. Ha most kitöröljük az első REM utasítást és újra indítjuk a programot, már többet láthatunk. Még érdekesebb, ha a színkódot X-szel vagy Y-nal helyettesítjük. A most aktivizált szín-RAM beállító utasítás már az FCOL-utasítás előzetese. A program pedig a két utasítás kombinációja, amelyek egyébként is szoros kapcsolatban állnak egymással. A 180. sorban egy olyan képletet használunk, amellyel kiszámíthatjuk a szín-RAM címeit és ami szükségszerűen hasonlít a képtárolónál megismert képlethez. Az egyetlen különbséget a kezdő cím jelenti, ami a szín-RAM esetében 55296.
Példánk 140. sorában csak egy lehetséges paramétersort állítunk be. Természetesen ezeket meg lehet változtatni és a programot újra futtatni. A karakterek képernyőkódját mint már említettük, a CBM64 felhasználói kézikönyv tartalmazza.
Most nézzünk egy példát az FCHR utasítás alkalmazására.

P. 40

```
100 REM ************
120 REM * 2.F C H R *
140 REM ************
150 REM
160 POKE 53248+33,0:REM HATTER=FEKETE
170 PRINT CHR$(147)
180 CENTRE "■MOST A KEPERNYO URES"
190 PAUSE 3
200 FCHR 3,0,40,22,1:REM (A) KARAKTER IRASA SZIN NELKUL
210 REM SOR=3,OSZLOP=0,SZELESSEG=40,MAGASSAG=22,KARAKTER='A'
220 PRINT AT(0,0)"";:REM KURZOR POZICIONALASA
230 CENTRE "■ A 3-24 SOROK SZIN NELKULI":PRINT
240 CENTRE "KARAKTEREKKEL TELNEK MEG.":PRINT
260 CENTRE "SEMMI NEM LATHATO."
270 PAUSE 6
280 PRINT AT(0,0)"";
290 CENTRE "   MOST MEGVALTOZTATJUK A   ":PRINT
300 CENTRE "   HATTERSZINT ES MINDEN    ":PRINT
320 CENTRE "   LATHATOVA VALIK !    "
330 PAUSE 9
340 POKE 53248+33,14:REM HATTER=VILAGOSKEK
350 PAUSE 6
360 POKE 53248+33,0:REM HATTER ISMET FEKETE
370 PRINT AT(0,0)"";
380 CENTRE "■   HA A KURZORT LEFELE ES  ":PRINT
390 CENTRE "    JOBBRA MOZGATJA    ":PRINT
410 CENTRE "LATNI FOGJA, MIROL VAN SZO!"
```

Reméljük, programunkhoz nem kell külön magyarázat.

9.4.2 FCOL – feltöltés színnel

Formátum: FCOL s,o,sz,m,ksz

Paraméterek: s – a képmező bal felső sarkának sora (0–24), és
o – oszlopa (0–39)
sz – a mező szélessége és
m – magassága
ksz – karakterszín (0–15)

Funkció: Egy képernyőtartomány megtöltése egy színnel

Példa: FCOL 5,4,5,3,8

A fejezet további tanulmányozása előtt föltétlenül el kell olvasni a 9.4 pont bevezetőjét, valamint az FCHR utasításnál leírtakat. Az FCHR- és FCOL-utasítások szorosan kapcsolódnak egymáshoz.

Az FCOL utasítás első négy paramétere megegyezik az FCHR-nál leírtakkal. Meg kell adnunk a kívánt színnel megtöltendő képtartomány helyzetét és nagyságát. Az így kijelölt tartományban a szín-RAM a ksz színt (0–15) veszi fel és a karakterek kizárólag a ksz által megadott színben jelenhetnek meg.

Ha a tartományba új szöveget akarunk írni, akkor a szín-RAM karakterszínét PRINT utasítással meg kell változtatni. A szövegnek tehát új színt kell kapnia.

Ha az eredeti szöveget (karaktereket) is a régi színben szeretnénk átvinni a szín-RAM-ból, akkor a képtárolóhoz közvetlenül a POKE vagy az FCHR utasításokkal kell hozzáférnünk.

A fent leírtakból következik, hogy az FCOL utasítás hatásai kizárólag csak akkor érvényesülnek, ha a tartományban szöveg (karakter) van.

(Hasonló módon, mint az FCHR esetén, ami csak akkor működik, ha a színt a szín-RAM-ban valamilyen módon beállítjuk).

Az FCOL utasítás – az FCHR-hez hasonlóan – helyettesíthető egy kis BASIC rutinnal, mely a következőképpen néz ki:

```
160 FOR Y=S TO H:REM SOR-CIKLUS
170 FOR X=O TO B:REM OSZLOP-CIKLUS
190 POKE 55296 + 40*Y + X,KSZ : REM SZÍN BEÁLLÍTÁSA
200 NEXT X,Y : REM KÖVETKEZŐ SOR/OSZLOP
```

Ne feledkezzünk meg valamilyen szöveg megjelenítéséről, különben nem láthatunk semmit a képernyőn.

Példa:

Az alábbi példaprogram megismerteti velünk az FCOL utasítás működését:

P. 41

```
1 REM **************
2 REM * 1.F C O L *
3 REM **************
150 REM
160 POKE53248+33,4:REM HATTER=LILA
170 PRINT CHR$(147)
180 PRINT CHR$(5):REM A KARAKTER=FEHER
190 CENTRE "MOST A KEPERNYO LILA":PRINT
200 PAUSE 3
210 PRINT AT(0,0)"";
220 CENTRE "AZ 1-24 SOROK ":PRINT
230 CENTRE "MEGTELNEK KEK SZINNEL," :PRINT
240 CENTRE "DE CSAK A LETEZO KARAKTEREK":PRINT
260 CENTRE "FOGNAK MEGVALTOZNI.":PRINT
270 PAUSE 9
280 PRINT CHR$(159):REM KARAKTERSZIN TURKIZ
290 CENTRE "FIGYELEM!":PRINT:PRINT
300 PAUSE 1
320 FCOL 1,0,40,24,6:REM (KEK) SZIN BEIRASA KARAKTER NELKUL
330 REM SOR=0,OSZLOP=0,SZELESSEG=40,MAGASSAG=24
340 PRINT CHR$(5);:REM KARAKTER=FEHER
350 CENTRE "AMIT MOST IRUNK,":PRINT
360 CENTRE "EREDETI SZINBEN JELENIK MEG"
```

Ezután pedig bemutatjuk a két utasítás (FCHR, FCOL) együttes alkalmazását:

P. 42

```
1 REM ****************
2 REM * 2. F C O L *
3 REM ****************
100 REM
110 POKE53248+33,0:REM HATTER=FEKETE
120 PRINT CHR$(147):REM KEPERNYO TORLES
130 FOR X= 0 TO 35 STEP 5
140 Y=Y+2 :REM SORSZAMLALO
150 FCHR Y,X,5,5,Y:REM AZ Y KEPERNYOKODU KARAKTEREK BLOKKJANAK FELTOLTESE
170 FCOL Y,X,5,5,Y/2:REM AZ Y/2 SZINKODU BLOKK FELTOLTESE
190 NEXT X
200 PAUSE 6
210 PRINT CHR$(147)
220 Y=0:REM KEZDO SZIN= FEKETE
230 FCHR 0,0,40,25,128+32
240 FCOL 0,0,40,25,6
250 REM A KEPERNYO MEGTOLTESE INVERZ KEK SZOKOZOKKEL
260 FOR X=4 TO 34 STEP 2
270 FCOL5,X,2,15,Y:REM EGY SAV SZINEZESE
280 Y=Y+1:REM A KOVETKEZO SZIN
290 NEXT X
```

9.4.3 FILL – feltöltés színes karakterekkel

Formátum: FILL s,o,sz,m,k,ksz

Paraméterek: s – a mező felső sarkának sor- (0–2) és
o – oszlop- (0–39) pozíciója
sz – a mező szélessége és
m – magassága
k – az aktiválandó karakterkód (0–255)
ksz – az aktiválandó karakterszín (0–15)

Funkció: A képernyő meghatározott részét azonos színű, egyforma karakterrel tölti fel

Példa: FILL 3,5,7,9,2,11

A továbblépéshez tanulmányozzuk át a 9.4 fejezet eddigi alpontjait. Látjuk, hogy ha a képernyő meghatározott tartományát színes karakterekkel akarjuk megtölteni, két utasításra van szükségünk.
A FILL utasítással azonban ezeket megtakaríthatjuk, sőt kiküszöbölhetjük vele az előző két utasításnál előforduló problémákat is.
Tulajdonképpen a FILL utasítás az FCHR és az FCOL utasításokat helyettesíti ugyanazon paraméterekkel. Az egyes paraméterek jelentését már előzőleg ismertettük.
Tehát a

```
100 FCHR s,o,sz,m,k : FCOL s,o,sz,m,ksz
```

utasításpár helyett egyszerűbben ezt is írhatjuk:

```
100 FILL s,o,sz,m,k,ksz
```

A két sor funkciója egy és ugyanaz, ezért az előbbi két utasításra elmondottak továbbra is érvényesek.

Példák:
A P.42-es, 2. FCOL című példaprogram 230. és 240. sorait helyettesítsük az alábbi sorokkal:

```
285 FILL 0,0,40,25,128+32,6
```

továbbá a 150., valamint a 170. sorokat a következőkkel:

```
200 FILL Y, X, 5,5,Y,Y/2
```

Láthatjuk, hogy a program ugyanúgy működik.
Próbálja ki a következő programot:

P. 43

```
1 REM ***********
2 REM * F I L L *
3 REM ***********
130 REM
140 PRINT CHR$(147)
150 POKE 53248+33,6:REM HATTER=KEK
160 POKE 53248+32,0:REM KERET =FEKETE
170 FOR X=1 TO 100
180 FILL 0,10,20,25,119,7:REM CHR$(183)
190 FILL 0,10,20,25, 69,7:REM CHR$(100)
200 FILL 0,10,20,25, 64,7:REM CHR$(96)
210 FILL 0,10,20,25, 70,7:REM CHR$(102)
220 FILL 0,10,20,25,111,3:REM CHR$(175)
230 NEXT X
```

Ugye látványos?

Magyarázat: A 180–200 sorok minden pozíciójában egymás után jelennek meg a karakterek. Egy-egy karakter egymás után többször, egyre lejjebb villan fel. Ezáltal érjük el ezt az érdekes, lefutó hatást.

9.4.4 MOVE – képernyőtartomány másolása

Formátum: MOVE s,o,sz,m,us,uo

Paraméterek: s – a mező első karakterének sor- (0–24) és
o – oszlop- (0–39) koordinátája
sz – a mező szélessége
m – a mező magassága
us – az új pozíció sora (0–24) és
uo – oszlopa

Funkció: egy képernyőtartomány átmásolása

Példa: MOVE 4,5,2,3,10,11

A MOVE utasítás elméletileg óriási lehetőségeket rejt magában. Használatával a képernyő kijelölt része átmásolható, vagyis elcsúsztatva megduplázható. Tehát egy megtervezett képernyőtartományt egyszerűen átvihetünk a képernyő másik részére.
Az első négy paraméter jelentését már ismertettük az előző utasításokban, ezért ezekre most nem térünk ki. Egyébként a tartomány most is egy tetszőleges téglalap. Az új us és uo paraméterekkel a másolat bal felső sarkának koordinátáit adjuk meg.
Ha a másolat nem fér el a kijelölt helyen, a program BAD MODE hibaüzenettel leáll.
Itt sajnos rábukkantunk egy kellemetlen hibára: a SIMON's BASIC lemez- és kazettaváltozata (V2-ig) egy kicsi, de bosszantó hibát rejt magában.

A fenti utasítás mindaddig korlátlanul használható, amíg us-t vagy uo-t meg nem próbáljuk változóként vagy matematikai kifejezésként megadni. Ekkor kilép a hibátlan működési folyamatból és értelmetlen eredményt kapunk (pl.: csak véletlenszerűen választott részeket, vagy egyáltalán semmit sem másol). Emiatt csak konkrét számokat adhatunk meg. Ez a hiba persze meglehetősen bosszantó, de ilyen összetett programoknál szinte elkerülhetetlen. Mindettől függetlenül ez az utasítás jelentős segítséget nyújt a képszerkesztésben.
Végül a szokásos példa, amely a MOVE utasítás működését mutatja be:

P. 44

```
1 REM ***********
2 REM * M O V E *
3 REM ***********
130 REM
140 PRINT CHR$(147)
145 PRINT "┌──────────────────┐"
150 PRINT "│EZT A KEPERNYO-   │"
160 PRINT "│  TARTOMANYT      │"
170 PRINT "│    FOGJUK        │"
180 PRINT "│  ATMASOLNI.      │"
185 PRINT "└──────────────────┘"
190 PAUSE 4
200 MOVE 1,0,18,6,1,20
210 PAUSE 1
220 MOVE 1,0,18,6,7,14
230 PAUSE 1
240 MOVE 1,0,18,6,13,0
250 PAUSE 1
260 MOVE 1,0,18,6,13,22
270 PAUSE 1
280 MOVE 1,0,18,6,13,14
290 PAUSE 1
300 MOVE 1,0,18,6,17,0
310 PAUSE 1
320 MOVE 1,0,18,6,17,20
330 PAUSE 1
```

9.4.5 INV – képernyőtartomány invertálása

Formátum: INV s,o,sz,m
Paraméterek: s – sor (0–24)
o – oszlop (0–39)
sz – a mező szélessége és
m – magassága
Funkció: egy képernyőtartomány invertálása
Példa: INV 5,6,3,4

Ezzel az utasítással újabb komoly képvezérlő eszközhöz jutottunk, mely lehetővé teszi teljes képtartományok invertálását.
Az utasítás paraméterei az eddigiekből már megszokottak (lásd FCHR, FCOL, FILL és MOVE) és szintén a kezelendő tartományt jelölik ki.
Az INV-utasítás működése egyszerű. A FLASH utasításhoz hasonlóan – ami az azonos színben jelzett karaktereket meghatározott sebességgel villogtatja a normál és inverz állapot között –, inverzre váltja az előzetesen normál üzemmódban (RVS/OFF) megjelenített karaktereket, úgy, mintha a PRINT utasítással aktivizáltuk volna az RVS/ON üzemmódot. Természetesen fordítva is igaz: az inverz karaktereket normál állapotba hozza.
A két utasítás között két lényeges különbség van. Egyrészt az INV-utasítás egyetlen váltást végez, míg a FLASH mindaddig működik, míg ki nem kapcsolják. Másrészt az INV utasítással kiválaszthatunk egy képtartományt, míg a FLASH az egész képernyőre és csak egy színre vonatkozik. Az INV utasításból természetesen FLASH-t is gyárthatunk egy meghatározott képtartományra vonatkozóan.
Végül még egy sajnálatos apróság: a SIMON's BASIC újabb hiányossága folytán előfordulhat, hogy ha 2 db, ugyanarra a képtartományra beadott INV utasítás között módosítjuk a háttérszínét, akkor a tartomány valamelyik részén felbukkan a régi háttérszín.
A hiba megszüntetésének módját sajnos nem ismerjük. Ettől függetlenül természetesen szabadon élhetünk minden felhasználási lehetőséggel.
Néhány ilyet mutatunk be az alábbi példákban:

1. példa:

Egészítsük ki a 9.4.4 alpont MOVE példáját az alábbiakkal:

```
360 INV  1,  0, 11, 4
370 INV  1, 28, 11, 4
380 INV 17,  0, 11, 4
390 INV 17, 28, 11, 4
400 PAUSE 1
410 GOTO 360
```

2. példa:

P. 45

```
1 REM *********
2 REM * I N V *
3 REM *********
130 PRINT CHR$(147)
140 POKE 53248+33,12:REM HATTER=SZURKE
150 FILL  2,10,10,10,127, 7:REM =CHR$(191)
160 FILL 14,10,10,10,102,14:REM =CHR$(166)
170 FILL  2,25,10,10,105,13:REM =CHR$(169)
180 FILL 14,25,10,10, 97, 2:REM =CHR$(161)
190 INV 2,10,10,22:REM ELSO KETTO
200 INV 2,25,10,22:REM MASODIK KETTO
210 FOR X=1 TO 80:NEXT X:REM ROVID KESLELTETES
220 GOTO 190:REM VILLOGAS
```

A P. 45-ös program 210. sora egy késleltető ciklust tartalmaz. Természetesen ezt PAUSE utasítással is helyettesíthettük volna, de ezt inkább a hosszabb ciklusoknál használjuk, mivel nem teszi lehetővé 1 másodpercnél rövidebb várakozási idő kiválasztását. Mindezek ellenére kicserélhetjük, vagy a hosszát módosíthatjuk.
Befejezéshez nyomjuk le a RUN/STOP billentyűt és máris folytathatjuk a munkát.

9.4.6 Ellenőrző kérdések

A fejezet végén ismét itt van néhány feladat, mellyel ellenõrizhetjük tudásunkat.

1. A képernyő 5. és 10. sora, ill. 4. és 9. oszlopa által határolt tartományt át akarjuk másolni a 8. sor és a 11. oszlop által kijelölt kezdőpoziciójú tartományba. Válasszuk ki a megfelelő megoldást!

   a) MOVE 5,10,4,9,8,11
   b) MOVE 5,4,10,9,8,11
   c) MOVE 5,4,6,6,8,11
   d) MOVE 4,5,6,6,11,8

2. Hogyan tölthetnénk meg a képernyő egy tartományát piros "A" karakterekkel?

   a) FCHR 5,5,10,10,1
   b) FILL 5,5,10,10,1,2
   c) FCHR 5,5,10,10,1:FCOL 5,5,10,10,2
   d) FILL 5,5,10,10,65,2

3. Mi történik az INV 0,0,0,0 utasítás hatására?

   a) A gép nem veszi figyelembe az utasítást.
   b) Invertálódik a bal felső sarokban lévő karakter.
   c) "BAD MODE" hibaüzenet jelenik meg.
   d) "ILLEGAL QUANTITY ERROR" hibaüzenet jelenik meg.

A helyes tippelés után folytathatjuk a munkát. (A teszt ezúttal sem volt könnyű!)

9.5 A képernyő gördítése

A következőkben további nyolc nagyon szép és hasznos utasítással ismerkedhetünk meg, melyek a képernyő gördítésével (scrolling) foglalkoznak.
Megígérjük, hogy örömét fogja lelni bennük. Az utasítások által létrehozott hatások ugyanis olyan szépek, hogy akár napokig elgyönyörködhetünk bennük (ez a személyes véleményünk).
Egy kis előzetes az utasítások rendszeréhez:
A nyolc utasítás részben ugyan hasonlít egymásra. Ezért alkalmaztuk a szekunder utasítások elvét. Az elv lényege, hogy összesen négy alaputasítás van (LEFT, RIGHT, UP, DOWN), melyek a gördítés irányát rögzítik. (Jelentésük: balra, jobbra, fel, le). Ezen primer utasításokhoz szekunder utasításként egy betűt kapcsolunk, ami a gördítés típusát határozza meg. A lehetséges szekunder utasítások a következők:

- W = ciklikus képernyő – gördítés
- B – gördítés képátfutás nélkül

Egy utasítás a következőképpen néz ki:

LEFTW 0,0,10,10

A primer (pl.: LEFT) és a szekunder utasítás (ebben az esetben W) közé nem kerülhet szóköz.
A ciklikus gördítésen azt értjük, hogy a megadott tartományból kilépett részek nem vesznek el, hanem az ellentétes oldalon újra megjelennek, tehát a kép állandóan körben forog. Sok forgalomban levő akciójátékból ismeretes ez az effektus, melyek már a Commodore 64-eshez is kaphatóak.
Képátfutás nélküli gördítéskor a kilépett kép nem tűnik fel ismét a másik oldalon, hanem helyette üres sorok jelennek meg, és az eredeti kép néhány átfordulás után eltűnik.
Ha később a grafikában akarjuk ezeket az utasításokat használni, nem fogunk sikerrel járni. Grafikához csak speciális grafikus bővítőprogrammal (pl.: a SUPERGRAPHIK 64) tehető alkalmassá. Egy ilyen program csak a grafikus megjelenítésre koncentrál és ezen kívül számos egyéb lehetőséggel rendelkezik.
Még egy apró technikai megjegyzés: próbáljuk ki a példákat, nem okoznak csalódást.

9.5.1 LEFT – gördítés balra

Formátum: LEFTW s,o,sz,m
Paraméterek: s – sor (0–24)
o – oszlop (0–39)
sz – a tartomány szélessége és
m – magassága
Funkció: a képernyő gördítése balra
Példa: LEFTW 5,7,3,4

Ez az utasítás – mint a neve is mutatja – a kijelölt képtartományt balra gördíti. Első két paramétere a tartomány helyét mutatja a képernyőn, az m és sz pedig a tartomány méreteit. Az s és o abszolút-, az m és sz relatív paraméterek. Ez azt jelenti, hogy m és sz értéke nem lehet nagyobb, mint az s és o által meghatározott kezdőpont és a képernyőhatárok különbsége. Ha pl. a tartomány bal felső sarka a 10. sor 10. oszlopában van, akkor m értéke max. 14 lehet, mert ennyi sor marad a képernyő aljáig. Hasonló elven sz nem lehet több, mint 29. Ezeket a határokat átlépve a téglalap kilóg a képernyőből, míg m = 0 és sz = 0 értékadás esetén "BAD MODE" hibaüzenetet kapunk. Mint látjuk, a paraméterek szintaktikája összhangban van a képtartományt vezérlő utasításokkal, ami természetesen célszerű, hogy így legyen.
(A SIMON's BASIC kézikönyv szerint ilyen esetben hibaüzenetet kapunk, de ez szerencsére téves megállapítás.)

P. 46

```
1 REM ***********
2 REM * L E F T *
3 REM ***********
130 REM
140 PRINT CHR$(147)
150 A$="CBM    64 ---SIMON'S BASIC///":REM FELIRAT DEFINIALASA
160 PRINT AT(7,12) A$:REM FELIRAT POZICIONALASA
170 FOR X=1 TO 216
180 LEFTW 12,7,LEN(A$),1:REM LEN(A$)=AZ A$ HOSSZA
190 FOR Y=X TO 100:NEXT Y:REM GYORSULO  GORDITES
200 NEXT X
210 FOR X=216 TO 1 STEP-1
220 LEFTW 12,7,LEN(A$),1
230 FOR Y=X TO 100:NEXT Y:REM LASSULO GORDITES
240 NEXT X
250 PRINT
260 CENTRE"MEGGYOZODOTT?"
```

Most már tudjuk, mi olyan vonzó ezekben a scrolling utasításokban. A példánkban egy futó feliratot jelentettünk meg, amely kezdetben egyre gyorsabban mozog, majd fokozatosan lassul. A program kulcsa a 190. és a 230. sor késleltető ciklusai, melyek időtartama X aktuális értékétől függ. Ez az aktuális érték a 170–200. és a 210–240. sorban elhelyezett ciklusok hatására először növekszik, majd fokozatosan csökken. A mozgó felirat folyamatossága érdekében a W szekunder utasítást használtuk, ennek hatására a bal oldalon eltűnő szöveg a jobb oldalon ismét megjelenik.

9.5.2 RIGHT – gördítés jobbra

Formátum: RIGHTW s,o,sz,m

Paraméterek: s – a tartomány bal felső sarkának sor (0–24), és
o – oszlop (0–39) koordinátája
sz – a tartomány szélessége és
m – magassága

Funkció: a kijelölt tartomány gördítése jobbra

Példa: RIGHTW 4,5,6,7

A RIGHT a LEFT utasítás ellentéte. Hatására a meghatározott képernyőtartomány szövege jobbra gördül. Elméletileg ez az utasítás ugyanúgy működik, mint a LEFT, de sajnos csak elméletileg, mert itt is találunk egy kisebb (szerencsére azonban nem komoly) szépséghibát.

A hiba folyamatos jobbra görgetésnél figyelhető meg legjobban. Mint a példában is láthatjuk, az utasítás feldolgozása közben a tartomány bal szélén különböző karakterek villannak fel. Az utasítás végrehajtása után ezek a jelek eltűnnek, így valóban csak egy kis szépséghibáról van szó.

Reméljük a későbbi programváltozatok ezt a hibát már nem fogják tartalmazni.

De magyarázkodás helyett inkább nézzük a példaprogramokat:

P. 47

```
1 REM ***************
2 REM * 1.R I G H T *
3 REM ***************
130 REM
140 PRINT CHR$(147)
150 FOR X=0 TO 39
160 Y=12*SIN(X/6)+12
170 PRINT AT(X,Y)"*":REM SZINUSZGORBE RAJZOLASA
180 NEXT X :REM 40-SZER
190 A$="-------------HOGY TETSZIK?-----------------"
200 FOR X=40 TO 1 STEP -1
210 RIGHTB 0,0,40,25:REM GORDITES JOBBRA, ATFUTAS NELKUL
220 PRINT AT(0,9)MID$(A$,X,1):REM FELIRAT BEGORDITESE
```

```
230 PRINT AT(0,11)">"
240 PRINT AT(0,12)"<"
250 PRINT AT(0,13)">":REM KARAKTER BEVITEL
260 NEXT X:REM 40-SZER
```

Példánkban egy tetszőleges szöveg beúsztatását mutatjuk be. A fenti módszerrel bármilyen hosszú szöveg is kiíratható. A szöveg beúsztatását a következő módon értük el: Minden egyes jobbra léptetést követően, a következő kiírandó karaktert a képből éppen kifutott karakter helyére írtuk. A sztringben tárolt szöveget így karakterenként "utántöltjük" a MID$ BASIC utasítás segítségével. Ez az utasítás (lásd CBM64 kézikönyv) az adott füzér tetszőleges részét kiemeli.
Az adott füzér az utasítás szót követő zárójel első eleme. (Sziktaktikája MID$(fk$;a,l)
A kiemelendő rész kezdőpozícióját az "a", míg hosszát az "l" rögzíti.
Részletesebb leírást a felhasználói kézikönyvben találunk.
Hasonlóan látványos példát találunk a DOWN utasításnál is, de egyelőre lássuk nem kevésbé szórakoztató 2. példánkat, a két utasítás (RIGHT/LEFT) kombinációjára:

P. 48

```
1 REM ***************
2 REM * 2.R I G H T *
3 REM ***************
130 REM
140 PRINT CHR$(147)
150 FOR X=0 TO 39
160 Y=12*SIN(X/3)+12:REM AZ ERTEKEK KISZAMITASA
170 PRINT AT(X,Y)"*":REM SZINUSZGORBE RAJZOLASA
180 NEXT X
190 FOR Y=1 TO 5
200 FOR X=1 TO 40
210 LEFTW 0,0,20,25:REM BAL OLDALI RESZ
220 RIGHTW 0,20,20,25:REM JOBB OLDALI RESZ
230 NEXT X:REM GORDITES KIFELE
240 FOR X=1 TO 40
250 RIGHTW 0,0,20,25:REM BAL OLDALI RESZ
260 LEFTW 0,20,20,25:REM JOBB OLDALI RESZ
270 NEXT X:REM GORDITES BEFELE
280 NEXT Y:REM 5-SZOR IDE-ODA
290 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

Egy megjegyzés: a 290. sor egy eddig nem használt utasítást tartalmaz. Leírását a CBM64-es felhasználói kézikönyv tárgyalja. Itt nem is térünk ki rá részletesen, csak megemlítjük!
A WAIT 198,255 kifejezés egy billentyű lenyomására vár, ami a GET utasítással lekérdezhető. Vagyis a

```
WAIT 198,255 : GET A$
```

utasítások az alábbiakat helyettesítik:

```
GET A$ : IF A$ = " " THEN GOTO 10
```

A billentyűzet-pufferben maradt karakter a POKE 198,0 utasítással törölhető ki. (A 198-as címen az operációs rendszer a lenyomott billentyűk mennyiségét, vagyis a pufferben található karakterek számát tárolja, maximum 10-et.)

9.5.3 UP – gördítés felfelé

Formátum: UPW s,o,sz,m
Paraméterek: mint a RIGHT utasítás esetében
Funkció: a kijelölt tartomány gördítése felfelé
Példa: UPW 10, 10, 5,6

Az előbbi két utasításnál leírtak erre az utasításra is teljes egészében vonatkoznak. Az UP a képernyő kijelölt tartományát felfelé görgeti a már jól ismert kétféle módon (9.5 fejezet bevezetése, W és B).
A RIGHT és LEFT utasítással részletesen foglalkoztunk, a további ismertetés csak ismétlés volna. Inkább nézzünk meg egy kis példát:

P. 49

```
1 REM *******
2 REM * U P *
3 REM *******
130 REM
140 PRINT CHR$(147):REM KEPERNYO TORLES
150 PRINT"RUN/STOP -PAL"
160 PRINT"MEGSZAKITHATO!"
170 FOR X=1 TO 11
180 PRINT AT(X+15,X)CHR$(109):REM 1.ATLOS KARAKTEREK
190 NEXT X
200 FOR X=12 TO 23
210 PRINT AT(23-X+15,X)CHR$(110):REM 2.ATLOS KARAKTEREK
220 NEXT X
230 PAUSE 2
240 UPW 0,15,12,24:REM GORDITES FELFELE
250 GOTO 240
```

A kurzorpozíció számítását az AT utasításban még nem tárgyaltuk. Röviden a következőkről van szó: Az első ciklusban (170.–190. sorok) a bal felső sarokból a jobb alsóba futó átlót számítjuk ki pontról pontra. Az AT-utasítás zárójelében szereplő első érték az oszlopokra vonatkozik. Az X értéke ciklusonként 1-gyel növekszik, a 15 hozzáadásával pedig az adott pontot a képernyőn 15-tel jobbra toljuk.
A második érték a sorokra vonatkozik. Itt az X értéke szintén ciklusonként nő. Így elértük, hogy minden sorba csak egy pont jusson.
Ugyanígy történik a másik átló meghatározása is (200.–220. sorok), a különbség csupán annyi, hogy az oszlopok értéke ebben az esetben növekedésével folyamatosan csökken, mivel egyre nagyobb számot kell a 23-ból kivonni. A többi teljesen analóg az előbb leírtakkal. Érdemes gyakorlásképpen néhány hasonló összefüggést kidolgozni.

9.5.4 DOWN – gördítés lefelé

Formátum: DOWN s,o,sz, m
Paraméterek: mint a LEFT utasítás esetében
Funkció: a kijelölt tartomány gördítése lefelé
Példa: DOWN 1,0,10,15

Az utolsó gördítő utasítás.
Ha még nem tudjuk pontosan az egyes paraméterek jelentését, nézzük meg még egyszer a LEFT utasításnál leírtakat. Ha készen vagyunk, folytassuk.
A RIGHT utasításhoz hasonlóan a DOWN utasítás sem problémamentes, de az utasítás végrehajtása után a különböző felesleges jelek itt is eltűnnek. (Lásd az alábbi motocross-példát.)
Az utasítás feldolgozása közben idegen karakterek villannak fel a blokkszélen és a mező felső peremén. A képernyőn lévő képet a különböző jelek véletlenszerűen felvillanó sora zavarja meg. (Pl. – karakter.)
Ezek a hibák nem minden esetben a szoftver hibái. Előfordulhat, hogy hardver-probléma, vagyis a gép képvezérlő egységére (VIDEOCONTROLLER) vezethetők vissza. Mindezek ellenére, az alábbiakban a DOWN utasítás különösen érdekes felhasználási lehetőségét mutatjuk be, egy rövid program keretében.

Természetesen a többi példához hasonlóan ez a program is összevonható néhány sorban. A tömörítés azonban károsan zavarná az alkalmazás bemutatását.
Próbáljuk meg mi a program rövidítését:

P. 50

```
1 REM ***********
2 REM * D O W N *
3 REM ***********
130 REM
140 REM *********
150 REM MOTOCROSS
160 REM *********
170 REM
180 PRINT CHR$(147)
190 POKE 53248+33,13:REM HATTER=VILAGOSZOLD
200 X=10:REM AZ UTPOZICIO KEZDETE
210 RV$=CHR$(18):REM RVS/ON
220 SC$=CHR$(144):REM FEKETE
230 GR$=CHR$(152):REM SZURKE
240 ST$=RV$+SC$+"!"+GR$+"     !   "+SC$+"!"
250 REM UT: FEKETE SZEGELY - SZURKE BURKOLAT(10 SPACE) -FEKETE SZEGELY
260 X=X+RND(1)*2-1  :REM 0,1,-1 A KANYAROK VELETLENSZERU ELOALLITASA
270 IF X<0 THEN X=0
280 IF X>38-LEN(ST$) THEN X=38-LEN(ST$)
290 REM A NEM ENGEDELYEZETT ERTEKEK KIZARASA
300 DOWNB 0,0,40,25:REM AZ UT LECSUSZTATASA
310 PRINT AT(X,0)ST$:REM UT RAJZOLASA
320 GOTO 260
```

A program futását csak a RUN/STOP billentyűvel állíthatjuk meg.

9.5.5 Ellenőrző kérdések

A fejezet végén néhány ellenőrző kérdés az ismeretek elmélyítése érdekében.

1. A 6. és 10. sor, valamint 5. és 13. oszlop által bezárt tartományt egy karakterrel – képfutás nélkül – feljebb kívánjuk hozni. Melyik a helyes formátum?

   a) UPW 6,10,5,13
   b) UPD 6,10,5,13
   c) UP D 6,5,9,5
   d) UPW 6,5,9,5
   e) UPW 6,5,9,5

2. A teljes képernyőt kívánjuk átlósan jobbra lefelé képátfutással eltolni. Melyik a helyes kombináció erre a célra?

   a) RIGHTW 0,0,40,25 : DOWN 0,0,40,25
   b) LEFTW 0,0,10,25 : UPW 0,0,40,25
   c) Nem megoldható
   d) Egyik sem a fenti megoldásokból

10. FEJEZET
BEVITELVEZÉRLÉS

10.1 FETCH – a bevitel ellenőrzése

Formátum: FETCH "ell", h, val
Paraméterek: "ell" – ellenőrző füzér
h – a bevitel maximális hossza
val – tetszőleges típusú változó
Funkció: A bevitel ellenőrzése, a hibás beadás kiszűrése
Példa: 10 FETCH "12345",1,X$

Ez az utasítás az INPUT utasítás minden lehetséges hátrányát kiküszöböli. A program bevitelének ellenőrzése fontos feladat, nehogy az alkalmazó hibája miatt a program ne működjön.
Pl.: egy füzérbe 5 jegyű érték beolvasásához nem célszerű az "INPUT A" formát használni. Ugyanis könnyen előfordulhat, hogy az alkalmazó 5-től eltérő szám – vagy betű – billentyű lenyomásával hibaüzenetet idéz elő.
A FETCH utasítás alkalmazásával 5 jegyű szám bevitele az alábbi módon hajtható végre:

```
FETCH"0123456789",5,A$
```

Ezáltal a számítógép kizárólag a 0 és 9 közötti számjegyeket fogadja el és tárolja az A$ füzérbe. Az utasításba olyan vezérlőkarakterek is beírhatók, melyek a rögzítés szempontjából egy meghatározott tartományt fognak át.

CHR$ (19) – csak nagybetűk (CHR$ (65)-től CHR$ (90)-ig
CHR$(17) – számjegyek és speciális karakterek (CHR$ (32)-től CHR$ (64)-ig)
CHR$ (29) – nagybetűk SHIFT-tel és anélkül

A FETCH-utasítással az alábbi módon történik egy 12 karakteres füzér (A$) bevitele, mely csupán nagybetűket tartalmazhat:

```
FETCH"CHR$(19)",12,A$
```

Bevitelkor a karakterek csak az INST/DEL billentyűvel törölhetők, de a megfelelő számú karakter beírása után a számítógép már csak a RETURN-t fogadja el, az INST/DEL funkcióját blokkolja.
A programozás során gyakran előfordul az igen/nem kérdésvariáció. Az eddigi ismereteink szerint a GET utasítással ez így valósítható meg:

```
10 PRINT"PROGRAM BEFEJEZŐDÖTT (I/N)?"
20 GET X$:IF X$ "I" AND X$ "N" THEN 20
30 IF X$="I" THEN END
40 ...
:
```

A FETCH alkalmazásával elegánsabban tudjuk ugyanazt megoldani:

```
10 PRINT"PROGRAM BEFEJEZŐDÖTT (I/N)?"
20 FETCH "IN",1,X$
30 IF X$ = "I" THEN END
40 ...
:
```

Különbség a két megoldás között még az is, hogy a FETCH utasítással végzett karakterbevitelt a RETURN-nel le kell zárni.

10.2 INKEY – a funkcióbillentyűk lekérdezése

Formátum: INKEY
Paraméterek: –
Funkció: a lenyomott funkcióbilletyűk lekérdezése
Példa: A = INKEY (a lenyomott funkcióbillentyűt az „A" változóban tárolja)

Az INKEY utasítással a lenyomásra váró funkcióbillentyű száma (1–16) adható meg. A Commodore 64-es összesen 8 funkcióbillentyűvel rendelkezik, ami a C= billentyű használatával kibővíthető 16-ra (lásd a 2. fejezetet).
Ezzel az utasítással egy megnyomott funkcióbillentyű számát tudjuk tárolni az előzetesen már definiált változóban. A változó 0 értéket kap, ha egyik billentyűt sem nyomjuk le. Az alábbi kis program egy funkcióbillentyű lenyomására vár, melyet kijelez a képernyőn:

```
10 X=INKEY : IF X=0 THEN 10
20 PRINT " A FUNKCIÓBILLENTYŰ SZÁMA: "X
30 GOTO 10
```

A program csak RUN/STOP-pal hagyható el.
Mivel a Commodore 64-es csak 8 funkcióbillentyűvel rendelkezik, és csak ezeknek van ASCII kódja (133–140), ezért elmaradt az ASCII lekérdezése. A további 8 billentyű kizárólag az INKEY utasítással kérdezhető le. Egy program során működő funkcióbillentyű lekérdezésére többek között menüvezérlésnél van szükség, például:

```
VÁLASSZA KI A KÍVÁNT FUNKCIÓT

F1 – CÍM BEVITELE
F2 – CÍM TÖRLÉSE
F3 – CÍM KIJELZÉSE
F4 – CÍM BEOLVASÁS
F5 – CÍM TÁROLÁSA
```

Az előbbi menü választási lehetőségét az alábbi program valósítja meg:

```
100 A=INKEY: IF A=Ø THEN 1ØØ
110 GOTO (A*1Ø+11Ø)
120 CALL BEVITEL
130 CALL TÖRLÉS
140 CALL KIJELZÉS
150 CALL BEOLVASÁS
160 CALL TÁROLÁS
```

Természetesen lehetőség van döntéshozatalra is a funkcióbillentyűk használatával:

```
KIVITELT A NYOMTATÓRA (F1) VAGY KÉPERNYŐRE (F3)
```

Az ehhez tartozó vezérlő programsorok:

```
100 A=INKEY : IF A=0 THEN 100
110 IF A=1 THEN CALL NYOMTATÓ
```

10.3 RESET – a DATA mutató állítása

Formátum: RESET ssz
Paraméterek: ssz – annak a DATA-sornak a száma, amire a DATA-mutatót állítani kell
Funkció: A DATA-mutató ráállítása a READ-del olvasandó sorra
Példa: RESET 1ØØ (A következő READ a 100. DATA sort kezdi olvasni)

Az előre meghatározott konstans értékek nagyszámú tárolására a DATA–READ utasításpárt használjuk.
A DATA-sorokban lévő értékek READ-del történő olvasását követően a mutató a soron következő DATA-értékre áll, amit a következő READ olvas be stb.
Az "OUT OF DATA ERROR" hibaüzenet akkor jelenik meg, ha a meglévőnél több DATA értéket akarunk olvasni. A mutatónak az első DATA értékre történő visszaállítását a BASIC–RESTORE utasítás végzi.
Az alábbi programrészlet további magyarázattal szolgál:

```
200 DATA PÉTER,GÁBOR,TAMÁS,RÓBERT,GÉZA, *
300 DATA 0,1,2,3,4,5,6,7,8,9,A,B,E,D,E,F, *
400 DATA 102,103,105,108,123,120,142,189, *
500 DATA PÉCS,EGER,ZIRC,SOPRON, *
```

SIMON's BASIC-ben a RESET 130-at kell megadni és a DATA mutató máris a 130. sor első értékére áll. Ha a definiált sor nem létezik, vagy benne nem DATA értékek szerepelnek, a program automatikusan a következő sorra áll.
Így történik a RESET egyedi beállítása:

```
 10 PRINT"KÉREM, VÁLASSZON!"
 20 PRINT
 30 PRINT" 1 – NÉV"
 40 PRINT" 2 – HEXA – ÉRTÉK"
 50 PRINT" 3 – SZÁM"
 60 PRINT" 4 – VÁROS"
 70 FETCH"1234",1,A$
 80 IF A$="1" THEN RESET 200
 90 IF A$="2" THEN RESET 300
100 IF A$="3" THEN RESET 400
110 IF A$="4" THEN RESET 500
120 READ A$
130 IF A$= "*" THEN END
140 PRINT A$
150 GOTO 120
```

Ha az előző DATA sorokat beadjuk, akkor a program a kívánt eredményt fogja adni.

10.4 Ellenőrző kérdések

1. A FETCH "0123456789ABCDEF$",3,A$ utasításokból melyik karakterek nem kerülnek beolvasásra?

   a) $87
   b) $FF
   c) A$A
   d) C$

2. Milyen karaktereket fogad el az FETCH CHR$ (19),3,X$ utasítás?

   a) Csak számjegyeket és különleges karaktereket.
   b) Csak nagybetűket SHIFT-el és anélkül.
   c) Csak nagybetűket.

3. Az INKEY utasítással hány funkcióbillentyű ellenőrizhető?

   a) 8
   b) 16
   c) 15
   d) 7

4. A DATA mutatót melyik utasítás képes egy tetszőleges sorra beállítani?

   a) RESTORE
   b) RESET
   c) CRESTORE

11. FEJEZET
BE- ÉS KIVITELI UTASÍTÁSOK

11.1 Lemezegység parancsok, utasítások

11.1.1 DISK

Formátum: DISK "parancs"
Paraméterek: parancs – bármilyen lemez parancs lehet
Funkció: Egyszerűsített kommunikáció a futtatóval
Példa: DISK "N:TESTDISK,MN"
Ez a parancs a lemez formálását végzi és a következő parancsokat teljes egészében helyettesíti:

```
OPEN 1,8,15, "N:TESTDISK,MN"
CLOSE 1
```

Az eddig alkalmazott parancsokat elhagyhatjuk, mert mindaz, amit azok tudnak, DISK-kel sokkal egyszerűbben végrehajtható. A parancs-csatorna nyitását és zárását is automatikusan elvégzi.
A következő listában a fontosabb disz-parancsokat foglaltuk össze:

NEW – a lemez formálása

Formátum: DISK"N:név,ID"
Paraméterek: név : a lemez neve, amelyet a felhasználó ad (max. 16 karakter)
ID : a lemez kétkarakteres azonosítója
Példa: DISK"N:SIMON's BASIC,A1"

SCRATCH – file törlése a lemezről

Formátum: DISK"S:1.filenév, 2.filenév,..."
Paraméterek: 1.filenév, 2.filenév,... : a törölni kívánt file-ok neveinek felsorolása
Példa: DISK"S:PRÓBA"

RENAME – a file átnevezése

Formátum: DISK"R:új név = régi név"
Paraméterek: új név, régi név: az átnevezni kívánt file új és régi neve
Példa: DISK"R:INFO=INFO.TEST"
Tehát az eddig INFO.TEST néven nyilvántartott file azután INFO néven fog szerepelni.

INITIALISE – a lemez inicializálása

Formátum: DISK"I"
Paraméterek: –

VALIDATE – a lemez újraszervezése

Formátum: DISK "V"
Paraméterek: –

A sokoldalú lemezmeghajló egységről a már említett VC 1541-es lemezegység programozása című DATA BECKER – NOVOTRADE kiadványból szerezhetünk bővebb ismereteket.

11.1.2 DIR

Formátum: DIR"$"
DIR"$: füzér"
DIR"$:?? füzér"
DIR"$:* = ft"
Paraméterek: füzér : file-név vagy a file-név része
? * : "Joker" (lásd a szövegben)
ft : a file típusa (P=PRG? S=SEQ; R=REL; U=USR)
Funkció: Kilistázza a lemez tartalomjegyzékét (direktory) vagy annak egy részét (2., 3., 4. változat) a tárban lévő BASIC-program törlése nélkül
Példa: DIR"$"
Kilistázza a teljes tartalomjegyzéket.

Láthatjuk, hogy milyen sokoldalú ez az utasítás. Négyféle irásmódja van, amelyek egymással keverhetők is. A legelső változat a legegyszerűbb, és mint már tudjuk, a teljes tartalomjegyzék kilistázásához használhatjuk.
Nézzük most a bonyolultabb változatokat. A csillag és a kérdőjel ún. "Joker"-ek. Akik már régebben programoznak, azok előtt nem ismeretlen ez a fogalom.
A csillag segítségével szelektálhatjuk a tartalomjegyzéket úgy, hogy abból csak bizonyos file-neveket listázzunk ki. Például: a

```
DIR "$:A*"
```

csak azokat a file-okat listázza ki, amelyeknek A-val kezdődik a neve. A név többi része a válogatás szempontjából nem lényeges. Ugyanígy működik a

```
DIR "$:TEST*"
```

utasítás is, amely minden olyan file-t és csak azokat listázza ki, amelyeknek a neve TEST-tel kezdődik (Pl.: TEST.1, TEST.UJ, TESTPROGRAM,...stb.).
A kérdőjel szerepe hasonló. Azokat a karaktereket helyettesítjük vele, amelyek a válogatás szempontjából nem lényegesek. (1 kérdőjel egy karaktert helyettesít!) A file-név elején vagy közepén is állhat. Példa:

```
DIR "$ : ???.01"
```

Ez az utasítás azokat a file-okat listázza ki, amelyeknek neve 6 karakterből áll és utolsó három karaktere ".01". A

```
DIR "$ : TEST.??"
```

viszont kilistáz minden olyan file-nevet, amely 7 karakterből áll és TEST.-tal kezdődik.
A kérdőjel és a csillag kombinálhatók egymással, pl.:

```
DIR"$:???.01*"
```

Ez az utasítás minden olyan file-nevet kilistáz, amelynek a 4., 5. és 6. karaktere ".01". Arra is lehetőségünk van, hogy a file-okat típus szerint gyűjtsük ki. A

```
DIR"$:*=P"
```

utasítás például kilistázza a lemezen található összes programfile nevét.
Ha idáig probléma nélkül eljutottunk, próbáljuk kitalálni, hogy a

```
DIR"$:???.18*=P"
```

utasítás melyik file-ok neveiről fog listát készíteni. Az elméletnél sokkal többet ér a személyes tapasztalat. Próbálgassuk hát a DIR utasítás különféle alkalmazásait mindaddig, míg minden érthetővé nem válik.
A DIR programon belül is alkalmazható. Ennek bizonyságául hívjuk be a lemezről az ide vonatkozó programot:

```
P. 51
1 REM *********
2 REM * D I R *
3 REM *********
100 PRINT"TARTALOMJEGYZEK LISTAZASA (I/N)?"
110 REPEAT:GET X$:UNTIL X$="I" OR X$="N"
120 IF X$="N" THEN CALL TOVABB
130 ::PRINT CHR$(147):REM KEPERNYO TORLES
140 ::DIR"$" :REM A TARTALOMJEGYZEK LISTAZASA
150 ::PRINT:PRINT" TOVABB <RETURN>"
160 ::REPEAT
170 :::GET X$
180 ::UNTIL X$=CHR$(13)
190 PROC TOVABB
```

A programot úgy célszerű felépíteni, hogy a DIR utasítás végrehajtása előtt törölje a képernyőt, majd a program "gombnyomásra" folytatódjon.
Érdekes dolog, hogy miért törli a tárban lévő BASIC programot egy normál LOAD"$",8 parancs és miért nem törli a DIR. A különbséget a két parancs eltérő működése magyarázza. Míg ugyanis a LOAD betölti a tartalomjegyzéket a főtárolóba, méghozzá $0801-től kezdődően a BASIC programok helyére (ezáltal a meglévő program természetesen törlődik), addig a DIR, mint egy file-t, byte-onként olvassa és közvetlenül ki is jelzi a képernyőn.

11.1.3 SCRSV

Formátum: SCRSV lfsz, e, mc, "név, S, W"
Paraméterek: lfsz : logikai file szám
e : egységszám (kazetta = 1; floppy = 8)
mc : másodlagos cím
név : a képernyőn lévő file neve
Funkció: Egy kisfelbontású (LOW RESOLUTION) képernyői kép tárolása
Példa: SCRSV 1,8,2, "TEST.SCRSV,S,W"

A kisfelbontású képernyő, tehát az, amelyik a bekapcsolás után aktivizálódik, ezzel az utasítással kazettán vagy lemezen rögzíthető. Ez egyaránt vonatkozik kisfelbontású grafikákra és normál szövegre, amelyek tárolása szekvenciális file formájában történik. Nagyon előnyösen alkalmazható pl. képernyőmaszkok rögzítésére. A maszk alkalmazása az adatrögzítés egyik módszere, hasonlít a nyomtatványok kitöltéséhez. Ahhoz, hogy a beviteli maszkot (nyomtatványt) ne kelljen minden esetben újra felépítenünk, elkészítése után tároljuk, majd szükség esetén újra betölthetjük. Az utasítás paraméterei ugyanazok, mint amiket szekvenciális file megnyitásánál használunk és az alkalmazási előírások is azonosak. Így pl. ugyanazzal a logikai file-számmal nem lehet a lemezen nyitott file, mert az a

```
FILE OPEN ERROR
```

hibaüzenetet fogja eredményezni.
Pédaként hívjuk be a lemezről az SCRSV című programot.

P. 52

```
1 REM **************
2 REM * S C R S V *
3 REM **************
100 REM
110 REM
120 REM
130 PRINT CHR$(147)
140 PRINT"NEV        :.................."
145 PRINT
150 PRINT"KERESZTNEV:.................."
155 PRINT
160 PRINT"UTCA       :.................."
165 PRINT
170 PRINT"HELYSEG    :.................."
175 PRINT
180 PRINT"TELEFON    :.................."
190 SCRSV 1,8,2,"TESZT.SCRSV,S,W"
200 GOTO 200
```

Az utasítás egyaránt alkalmazható programban és közvetlenül üzemmódban, de az utóbbi esetben figyelembe kell vennünk, hogy a képernyő részeként az SCRSV utasítás is rögzítésre kerül.

A kisfelbontású képek tárolása célszerű eszköz a képinformációk rögzítésére. Kár, hogy nagyfelbontású (HIGH-RESOLUTION) megfelelője nincs. Ez a funkció az alábbi előnyökkel rendelkezik és például SUPERGRAPHIK 64-gyel meg is valósítható.

1. Pillanatok alatt betölthetők a lemezről olyan bonyolult grafikák (pl. háromdimenziós grafika), amelyek egyébként csak nehezen, bonyolult számításokkal állítható elő.
2. A rögzített HIGH-RESOLUTION grafikák, amelyeket a legkülönfélébb programokkal készítettek a CMB–64-re, kompatibilisek.

11.1.4 SCRLD

Formátum: SCRLD lfsz, e, mc, "név"

Paraméterek: lfsz – logikai file szám
e – egységszám (kazetta = 1, floppy = 8)
ma – másodlagos cím
név – a képernyő-file neve

Funkció: Az SCRSV-vel rögzített kisfelbontású (LOW RESOLUTION) képernyői kép betöltése

Példa: SCRLD 1,8,2, "TEST.SCRLD"
A TEST.SCRLD néven tárolt képernyői kép betöltése.

Ez az utasítás tehát az előzőleg SCRSV-vel lemezre vagy kazettára rögzített képernyőtartalmat tölti be. A képernyőn megjelennek azok az adatok, emelyeket a rögzítéskor tartalmazott. Lehetőségünk van tehát arra, hogy egy képernyői képet (grafikát vagy szöveget) egy extern programmal előállítsunk, rögzítsük, majd egy másik programmal ezt betöltsük és felhasználjuk. Pl. elkészítjük egy programmal az adatfelviteli maszkot és tároljuk. Ezután megírjuk az adatrögzítés programját, amelyben a maszk ismételt képernyői előállítása egyetlen (betöltő) utasításból áll.

A következő program egy ilyen, előzőleg SCRSV-vel tárolt képernyő betöltésére szolgál.

P. 53

```
1 REM **************
2 REM * S C R L D *
3 REM **************
100 REM
130 SCRLD 1,8,2,"TESZT.SCRSV"
140 GOTO140
```

A program RUN/STOP-pal szakítható meg.

Természetesen ugyanez közvetlen üzemmódban is beadható:

SCRLD 1,8,2, "TEST.SCRSV"

11.1.5 Ellenőrző kérdések

1. Törölni akarjuk a 'TEST.SCRSV" file-t. Melyik parancsot alkalmazzuk?

   a) DISK"S:TEST.SCRSV"
   b) DISK"N:TEST.SCRSV"
   c) DISK"R:TEST.SCRSV"

2. Melyik file-t *nem* jelzi ki a következő parancs: DIR"$:????.TE"?

   a) VARB.TEST
   b) KA01.TE
   c) SCRL.TE

3. Melyik file-t *nem* jelzi ki a következő parancs: DIR"$:??FILE*"?

   a) MNFILES
   b) MAFILE.PROGRAM
   c) FILEMANAGER

4. Egy kisfelbontású képernyő megjelenítő képet akarunk lemezen rögzíteni. Melyik utasítást alkalmazzuk?

   a) SCRSAVE1,8,2,"LOWRES01,S,W"
   b) SCRSV 8,8,3,"LOWRES01",S,W
   c) SCRSV 2,8,2,"LOWRES01"
   d) SCRSV 5,8,3,"LOWRES01,S,W"

11.2 Nyomtatási parancsok, utasítások

11.2.1 COPY

Formátum: COPY
Paraméterek: –
Funkció: Hardcopy készítése egy HI-RES vagy MULTICOLOUR grafikáról

A COPY-parancs automatikusan megnyitja és le is zárja a nyomtató csatornát, tehát nincs szükség az OPEN 1,4 és a CLOSE 4 parancsok beadására.
Hardcopy-t az alábbi nyomtatókon készíthetünk:

- CBM VC–1520
- CBM VC–1525
- SEIKOSHA GP 100 VC
- EPSON RX–80, FX–80 (DATA BECKER interface-szel)

Az új, VC–1526 típusra grafika nyomtatását teljesen más elven szervezték, ezért erre a COPY utasítás nem alkalmazható. (Egy speciálisan erre készített hardcopy rutint a SUPERGRAPHIK 64 című DATA BECKER kiadványban találhatunk).
Ez az utasítás egyaránt alkalmazható programban és közvetlen üzemmódban. Ha programban alkalmazzuk, hozzáfűzhetjük a grafikát előállító programhoz, így egy programon belül megoldható a képernyői kép előállítása és kinyomtatása is.
Ha a képernyői képet előállító programunk lefutott és a számítógép READY-vel bejelentkezett, még mindig megtehetjük, hogy közvetlen üzemmódban beadjuk a COPY-t, annak ellenére, hogy a képernyő ilyenkor már szöveg üzemmódban (LOW RESOLUTION) van. A hardcopy elkészítéséhez tehát nem feltétlenül szükséges látnunk az adott képet a képernyőn.

11.2.2 HRDCPY

Formátum: HRDCPY
Paraméterek: –
Funkció: Hardcopy készítése a LOW RESOLUTION képernyőről

Ezzel az utasítással a normál szöveg-üzemmódú képernyőről, amelyet lemezen is rögzíthetünk, készíthetünk hardcopy-t. Néha szükség lehet arra, hogy az előállított képernyőt egy "gombnyomással" papírra vigyük. Ezt végzi a következő program:

P. 54

```
1 REM ***************
2 REM * H R D C P Y *
3 REM ***************
100 REM
110 PRINT"HARDCOPY (I/N)?"
120 FETCH "IN",1,X$
130 IF X$="I" THEN HRDCPY
140 STOP
```

11.2.3 Ellenőrző kérdések

1. Milyen típusú képernyő hardcopy-ját készíthetjük el a COPY-val?

   a) HIGH-RESOLUTION
   b) MULTI-COLOUR
   c) LOW RESOLUTION

2. Milyen típusú képernyő hardcopy-ját készíthetjük el a HRDCPY-val?

   a) HIGH-RESOLUTION
   b) MULTI-COLOUR
   c) LOW RESOLUTION

3. Meg kell-e nyitnunk a COPY és a HRDCPY beadása előtt a nyomtatócsatornát (OPEN 1,4)?

   a) igen
   b) nem

12. FEJEZET
(Grafika)

Végre elérkeztünk a programozás csúcspontjához, minden potenciális számítógéptulajdonos álmához, a programozás művészetének felső határához: A GRAFIKÁHOZ!
Elsősorban a grafikakészítés képessége teszi a számítógépet igazán számítógéppé. Az, hogy meggyőzően, a lehető legnagyobb valósághűséggel tud információkat közvetíteni és hogy általa a laikusok is élhetnek a modern információfeldolgozás lehetőségeivel. Az akusztikai lehetőségekkel (szintetizátor) kiegészítve programozása szinte művészi színvonalra emelhető.
Nem csoda tehát, hogy a Commodore 64-es tervezői fáradságot nem kímélve azon dolgoztak, hogy nagy népszerűségnek örvendő készülékük mindezekkel a képességekkel felruházzák. A szoftverspecialisták feladata, hogy ezeket a lehetőségeket mindenki számára elérhetővé tegyék. Érthető, hogy a SIMON's BASIC sem nélkülözheti a grafikai utasításokat, bár általános programbővítésről lévén szó, nem vállalkozhat arra, hogy a CBM 64-es ezirányú képességeit teljes egészében kiaknázza. Akit bővebben érdekelnek számítógépének grafikai lehetőségei, annak a speciálisan e témával foglalkozó szakirodalmat ajánljuk, pl. a már nagyon olcsón megvásárolható SUPERGRAFIK 64-et. Ez 183 utasításával és utasításkombinációjával próbálja a téma iránt érdeklődők igényeit kielégíteni.
Térjünk rá most a SIMON's BASIC grafikai utasításainak tárgyalásaira, előbb azonban a könnyebb érthetőség kedvéért ismerkedjünk egy kicsit a CBM 64 grafikus képességeivel, legalábbis ami a SIMON's BASIC-et érinti. Részletesebb ismereteket egyéb irodalmakból szerezhetünk. (pl. A Commodore 64-es belső felépítése stb...).

12.1 Néhány szó a hardverről

Ebben a fejezetben megismerkedünk a számítógépes grafika készítésének alapjaival. A sprite-okat, a karakterkészletet és a szintetizátort a későbbiekben tárgyaljuk.

A színekről:

A Commodore 64-es mind szöveg-, mind pedig grafikus üzemmódban 16 színt tud előállítani. Minden színt egy színkód jelöl, amelyet bináris szám formájában a különböző regiszterek tárolnak. Ha pl. a video interface chip 32. regiszterének tartalma 0, a képernyő keretszíne fekete lesz.

A következő listában a színkódok találhatók:

0 – fekete	8 – narancs
1 – fehér	9 – barna
2 – vörös	10 – rózsaszín
3 – türkiz	11 – szürke 1
4 – ibolya	12 – szürke 2
5 – zöld	13 – világoszöld
6 – kék	14 – világoskék
7 – sárga	15 – szürke

A finom felbontású grafika:

A számítógép kétféle grafikus üzemmódban működik. Ezek a finom felbontású és a többszínű üzemmód. Finom felbontású üzemmódban a képernyő vízszintesen 320, függőlegesen 200 pontból áll, amely a képernyőn egyenletes elosztásban összesen 64 000 pontot jelent.
Természetesen a grafikát ugyanúgy tárolni kell, mint a szöveget (lásd 9. fejezet). Erre szolgál a grafikus tároló, melyben minden ponthoz 1 bit tartozik. A bit olyan információs egység, melynek értéke 0 vagy 1 lehet. Egy karaktert (szót) 8 bittel (azaz 1 byte-tal)

ábrázolhatunk. 1 byte a nyolc bit kikapcsolt (0) és bekapcsolt (1) értékeinek kombinációjával $2^8 = 256$ különböző karakter előállítására képes.
Egy byte tehát a képernyőn 8 pontot jelent, így a finom felbontású grafikus képernyő tárolásához 64000/8 = 8000 byte- 8K (kilobyte) tárkapacitás szükséges. Ezt a SIMON's BASIC helytakarékosan a RAM-ban, az operációs rendszer ROM-ja alatt a $E000 és a $FF40 (vagy decimálisan az 57344 és a 65344) címek között helyezi el. Ha tehát POKE utasítással közvetlenül a képernyőtárolót akarjuk elérni, ebben az intervallumban kell gondolkodnunk.
Mielőtt azonban ilyesmivel kísérleteznénk, nézzük meg, hogyan is építi fel a számítógép a tárolt információkból a grafikus képet:
A képtárolás alapja a grafikatároló 8 byte-ja által képzett 8 × 8 bitből álló mátrix, hasonlóan a szöveg-üzemmód karakterábrázolásához. A képernyőn 1 sorban 320/8 = 40, egy oszlopban pedig 200/8 = 25 ilyen mátrix fér el, amit a következő példán szemléltethetünk.

```
                1.                  2.                ...40.
  bit     0.1.2.3.4.5.6.7.    0.1.2.3.4.5.6.7.          ...
0. byte   ..........................
1. byte   ..........................
2. byte   ..........................
3. byte   ..........................
4. byte   ..........................
5. byte   ..........................
6. byte   ..........................
7. byte   ..........................
    .     ..........................
    .                .
    .                .
200 byte
```

A színmeghatározás szempontjából is fontos ismernünk ezt az elrendezést. A háttér (kikapcsolt pontok) és a bekapcsolt pontok egyaránt 16 féle színt vehetnek fel. (Ezek felsorolását lásd a fejezet elején). Ha mind a 64 000 pont színét egyenként akarnánk meghatározni, akkor ahhoz összesen 64 000*4 = 256 000 bit-re, azaz 32 K byte tárkapacitásra volna szükségünk a RAM-ban. Ez nagyon sok, ezért egyedi színt csak 8 × 8-as mátrix számára határozhatunk meg úgy, hogy hozzárendelhetünk egy háttérszínt (a kikapcsolt pontok, vagy 0 értékű bitek) és egy pontszínt (a bekapcsolt pontok vagy 1 értékű bitek). Mivel ezek különböznek egymástól, végül is minden mátrix számára két színt adhatunk meg. Így már elegendő az 1K, amivel az ún. videoRAM rendelkezik. ($000–$C3E7, vagy 49152–50151). Ebben minden byte felső négy bitje a pont, alsó négy bitje pedig a háttér színét tárolja.
Egy adott pont vezérlése (bekapcsolása vagy lekérdezése) BASIC-nyelven az alábbi programmal valósítható meg:

```
10 REM A PONT KOORDINÁTÁI: X ÉS Y
20 AD = 57344:REM A GRAFIKUS TÁROLÓ BÁZIS-CIME
30 DY = 320*INT (Y/8) + (Y AND 7)
40 DX = B*INT (X/8)
50 MA = 2^(7-(X AND 7))
60 AV = AD + DY + DX
70 REM PÉLDA:EGY PONT ELHELYEZÉSE (BEKAPCSOLÁSA)
80 POKE AV, PEEK (AV) OR MA
```

A SIMON's BASIC-kel ugyanez sokkal egyszerűbben megoldható, ezért ezt a programot nem szükséges elemeznünk:
A következő példában azt mutatjuk be, hogyan állítható elő a POKE-utasítással egy színes csík:

```
10 HIRES 2,1:REM GRAFIKA BEKAPCSOLÁSA
20 FOR X=57344 TO 65344
30 POKE X,%11110000:REM A BYTE FELSŐ NÉGY BITJÉNEK BEKAPCSOLÁSA
40 NEXT X
50 WAIT 198,255:REM VÁRAKOZÁS EGY GOMB LENYOMÁSÁRA
```

Ha lefuttatjuk, láthatjuk, hogy egy ilyen BASIC-program milyen lassú.

Többszínű grafika:

Tapasztaltuk, hogy finomfelbontású üzemmódban a számítógép színfelbontása nagyon kicsi. Ennek ellensúlyozásaként született meg az ún. multicolor üzemmód, amelyben a 8 × 8-as mátrix egyszerre négy színt vehet fel, természetesen a felbontóképesség rovására. Ebben az üzemmódban kétszeres szélességűek a pontok, azaz minden pontot 2 bit tárol. Ennek megfelelően a képernyő vízszintes irányban 160, függőlegesen pedig továbbra is 200 pontból áll.
A grafikatároló minden két bitje, amely 1 pontot jellemez, a blokk lehetséges négy színregisztere közül kijelöl egyet úgy, hogy a bitek értéke megegyezik a regiszter számával. (Szám alatt itt nem a színkódot (0–15), hanem annak a regiszternek a számát értjük, amely a színkódot tartalmazza).
A négy színregiszter a következő tárolórekeszekben található:

színregiszter	bit-kód	tároló
0	00	a VIC 33-as regisztere
1	01	a video-RAM alsó 4 bitje
2	10	a video-RAM felső 4 bitje
3	11	szín-RAM

Ha tehát a grafikatároló megfelelő byte-jának tartalma 10, akkor a pont színe olyan lesz, amilyet a video-RAM-ban hozzátartozó byte felső négy bitje meghatároz.
A VIC (= videocontroller) 33. regiszterének szerepét már ismerjük. Szöveg-üzemmódban a háttér színét rögzíti és a "POKE 53248+33, "f" utasítással vezérelhető. (f-színkód)
A szín-RAM most az a tárolórész, amely a szöveg-üzemmódban a karakterszínt rögzíti.
Elméletileg tehát minden egyes 8 × 8-as mátrix részére 1–3 között variálhatjuk a színeket.
A könnyebb érthetőség kedvéért tanulmányozzuk a következő programot:

```
10 HIRES 2,1:MULTI 12,8,6:REM MULTICOLOR ÜZEMMÓD BEKAPCSOLÁSA
20 FOR X=57344 TO 65344
30 POKE X, % 1100100 :REM 3 KÜLÖNBÖZŐ SZÍNŰ, 1 PONT SZÉLESSÉGŰ CSÍK
40 NEXT X
50 WAIT 198,255:REM VÁRAKOZÁS EGY GOMB LENYOMÁSÁRA
```

Futtassuk le! A képernyőn 1 pont szélességű függőleges csíkok jelennek meg. Az első csík a MULTI-ban meghatározott 3. színben (11) = kék, a 2. csík háttér színű (00), a 3. csík 2. színű (10)-orange, a 4. csík 1. színű (01) = 2. szürke.
Ennyi elég is lesz az alapok ismeretéből. Nem fontos az elmondottakat megjegyezni, de a későbbiek során mindenképpen segíteni fognak a bonyolult grafikák megértésében és előállításában.

12.2 Grafikus üzemmód és a színek meghatározása

Koordináta-rendszer

A SIMON's BASIC a képernyőt finom felbontású-FFB üzemmódban 320 × 200, többszínű-(MULTICOLOR) MC üzemmódban pedig 160 × 200 pontra bontja fel. Ez azt jelenti, hogy az utóbbi esetben a pontok kétszer olyan szélesek, mint az előzőben. Ezt figyelembe kell venni a pontok koordinátáinak meghatározásakor.

A képernyő kiindulási pontja (x=0; y=0) a bal felső sarok, ahonnan az x-koordináták jobbra, az y-koordináták pedig lefelé nőnek. A képernyő utolsó pontja tehát a jobb alsó sarok, melynek koordinátái FFB-ben (319,199), MC-ben (159,199). Nem létező (pl. 400,600) koordináta beírása.

ILLEGAL QUANTITY ERROR

hibaüzenetet eredményez.
Az FFB és a MC színeit színfelbontását már az alapoknál ismertettük.

Grafikát kétféleképpen készíthetünk:
1. Grafikus üzemmódban úgy, hogy a kép közvetlenül megjelenik a képernyőn, elkészültét figyelemmel kísérhetjük.
2. Szöveg-üzemmódban, úgy, hogy a rajz a "háttérben" készül és csak később jelenítjük meg.

A kevés rendelkezésünkre álló utasítással is viszonylag sok funkciót valósíthatunk meg az ún. rajzoló-üzemmód segítségével, amely minden grafikai utasításra (LIN,REC...) érvényes. Ezzel meghatározhatjuk egy pont, vonal, téglalap stb. megjelenési formáját. A kép tehát a rajzoló-üzemmód függvényében különbözőképpen jelenhet meg. A rajzoló-üzemmód kódjait és funkcióikat az alábbiakban ismertetjük:

rajz üzemmód kódja	funkció
FFR-üzemmódban:	
0	a kép törlése
1	a kép rajzolása
2	a kép invertálása
MC üzemmódban:	
0	a kép törlése
1	a kép rajzolása az 1. színregiszter színével
2	a kép rajzolása a 2. színregiszter színével
3	a kép rajzolása a 3. színregiszter színével
4	a kép invertálása

MC-ben az invertálás színcserét jelent a következők szerint:

0. (háttér) és a 3. színregiszter színe között
1. (háttér) és a 2. színregiszter színe között
2. (háttér) és az 1. színregiszter színe között
3. (háttér) és a 0. (háttér) színregiszter színe között

Esetünkben kép alatt értünk minden olyan ábrát (pont, vonal, kör... stb.), amelyet SIMON's BASIC utasítással elő tudunk állítani.
Invertálás a bekapcsolt pontok törlését és az üres pontok bekapcsolását jelenti.
Színregiszternek azt a tároló egységet nevezzük, ami a 8×8-as mátrix négy lehetséges színe közül egyet tartalmaz (lásd a fejezet elején!). A színregiszterben tehát azt a színkódot (0–15) tároljuk, amelyet az ehhez a regiszterhez tartozó pontoknak kell felvenniük. A 0. színregiszter mindig a háttér színét tartalmazza.
A rajzoló-üzemmód a grafikai utasításokat rugalmas és változatokban gazdag utasításkomplexummá bővíti.
Ezek után térjünk rá a tulajdonképpeni grafikai utasítások megismerésére, legelőször is a grafikai üzemmód bekapcsolását eredményező utasításra.

12.2.1 HIRES – FFB üzemmód bekapcsolása

Formátum: HIRES psz,hsz
Paraméterek: psz – a pontok színének kódja (0–15)
hsz – a háttér színének kódja (0–15)
Funkció: Bekapcsolja és törli a finom felbontású (FFB) üzemmódot, meghatározza a pontok és a háttér színét
Példa: HIRES 7,8
A háttér színe narancs, a pontoké sárga lesz.

Ez az utasítás nyitja meg számunkra a grafika világát. Bekapcsolja a grafikus üzemmódot és beállítja a színeket. Ezt az utasítást használjuk a szöveg üzemmódról grafikára történő átkapcsoláshoz. Egyidejűleg törli a grafikustárolót, tehát beírása után a képernyő kiürül. Ezt figyelembe kell vennünk, ha a program során többször vissza kell térnünk a szöveg üzemmódba. Ha azt akarjuk, hogy ilyenkor az előállított grafika ne vesszen el, az átkapcsolást más utasítással kell végrehajtanunk. A grafika megjelenítésekor természetesen a HIRES utasítás is alkalmazható.
A grafikus kép törlésekor a grafikus tároló minden bitje nullázódik és csak a video-RAM-ból származó háttérszín marad meg. Ez és a később használatos pontszín a HIRES-utasítás két paraméterével állítható be:
Az első érték a grafikus képernyő valamennyi bekapcsolt pontjának színét (a színkód felhasználásával) definiálja, ha másképp nem rendelkeztünk. Természetesen ez csak a finom felbontású (FFB) grafikára vonatkozik. Többszínű üzemmódban a színeket újra definiálnunk kell, és ez a második paraméterre is érvényes. Ez állítja elő a háttér, azaz minden kikapcsolt pont színét. Ezt látjuk pl. ha a HIRES utasítást követően a képernyőn egy törölt grafikus kép "jelenik" meg. Az itt előállított háttérszínnek azonban nincs köze a szöveg üzemmódban COLOUR utasítással definiált háttérszínhez (lásd. 9.3. fejezet). Ezek csak többszínű üzemmód esetén egyeznek meg, mégpedig az azonos színregiszter miatt.
Végül még egy megjegyzés: Grafikus képernyő csak programban állítható elő. Ha tehát közvetlenül írjuk be, akár a HIRES-, akár más hasonló utasítást, a számítógép végrehajtja ugyan, de rögtön vissza is tér szöveg üzemmódba. Ugyanez történik a program befejezésekor vagy megszakításakor is. A grafikus üzemmód a programon belül a CSET 0/1- vagy az NRM utasítással kapcsolható ki. (Lásd a 9. és 13. fejezetet).
Egy kis trükkel elérhető, hogy a grafikus kép programon kívül is a képernyőn maradjon. Az alábbi háromsoros program ezt mutatja. Találkozunk két új utasítással is (MEM és BLOCK), amelyeket a későbbiek során fogunk megismerni.

```
10 HIRES 6,7: REM GRAFIKA BEKAPCSOLÁSA
20 MEM:REM KARAKTERKÉSZLET ELŐÁLLÍTÁSA
30 BLOCK 0,0.319,105,0:REM A GRAFIKA FELSŐ RÉSZÉNEK TÖRLÉSE
```

Futtassuk le a programot. A képernyőn megjelenik a C 64-es teljes karakterkészlete először egy pillanatig a HIRES-utasításban meghatározott színekkel, majd átvált fekete alapon piros karakterekre. (10–20. sor). Ez a színösszeállítás a továbbiakban megmarad és módosítani nem tudjuk. A 30. sor hatására a karakterek eltünnek és megmarad egy üres, fekete háttér. Ha most a programot egy rajzzal folytatjuk, az akkor is a képernyőn marad, ha a programot END-del fejezzük be.
Próbáljuk ki. Hívjuk be a TRÜKK nevű programot és futtassuk le:

```
P. 55
1 REM **********
2 REM * T R Ü K K *
3 REM **********
100 REM
```

```
110 HIRES 8,13:REM GRAFIKA BEKAPCSOLASA
120 MEM:REM KARAKTERKESZLET KIJELZESE
130 BLOCK 0,0,319,105,0: REM A KARAKTERKESZLET TORLESE
140 REM
150 REM ***********
160 REM * GRAFIKA *
170 REM ***********
180 REM
190 FOR X=1 TO 319 STEP .5
200 PLOTX,60*SIN(X/25)+100,1
210 NEXT X
220 END
```

A program lefutását a képernyő bal felső sarkában látható színes csíkok megjelenése jelzik. Ezek "eltüntetését" kísérletezőkedvű programozóinkra bízzuk. Mi elég, ha azt próbáljuk ki, mi történik, ha a MEM-utasítást tartalmazó sort kitöröljük.

Ha már teljesen kiismertük ezt a kis programot, behívhatjuk a HIRES-utasítás tulajdonképpeni példaprogramját és folytathatjuk a kísérletezést:

P. 56

```
1 REM **************
2 REM * H I R E S *
3 REM **************
130 REM
140 FOR HF=0 TO 14 :REM HATTERSZIN
150 HIRES 15,HF     :REM GRAFIKA BEKAPCSOLASA,TORLESE ES SZINBEALLITAS
160 FOR X=59264 TO 59583
170 POKE X,%11110000:REM KEPERNYO RESZLET:CSIKOK
180 NEXT X
190 NEXT HF :REM KOVETKEZO HATTERSZIN
200 REM VEGE:VISSZATERES A SZOVEG-UZEMMODBA
```

Mint látjuk, a program két egymásba épített FOR...NEXT ciklusból áll. A külső változtatja a háttérszínt (0–14), a belső pedig egy sor függőleges csíkot rajzol a HIRES-ben megadott 15. színnel. A program végén, mivel más utasítást nem adtunk, visszatérünk szöveg-üzemmódba.

Próbáljuk meg most a háttérszín helyett a pontszínt változtatni!

12.2.2 MULTI – az MC-üzemmód bekapcsolása

Formátum: MULTI 1.sz, 2.sz, 3.sz

Paraméterek: 1. sz. – az 1. színregiszter színe (0–15)
2. sz. – a 2. színregiszter színe (0–15)
3. sz. – a 3. színregiszter színe (0–15)

Funkció: Átkapcsolás FFB üzemmódról MULTICOLOR (többszínű) üzemmódra és a háromféle pontszín meghatározása

Példa: MULTI 4,5,6
Háttérszín a HIRES utasításban megadott, a pontok színe pedig ibolya, zöld és kék lehet

Mint a bevezetőben már olvashattuk, számítógépünk és ezzel együtt a SIMON's BASIC is rendelkezik az ún. többszínű üzemmóddal, ami a nagyfelbontású grafika csekély színfelbontását hivatott ellensúlyozni. Ebben az üzemmódban minden 8 × 8-as (azaz itt a duplaszélességű pontok miatt 4 × 8-as) mátrix négy színt vehet fel.

A MULTI utasítás nem a HIRES-hez hasonlóan működik.

Míg az utóbbi egy része egész sor funkciót lát el (grafikus üzemmód és a FFB-üzemmód bekapcsolása, a háttér- és a pontszín meghatározása), addig a MULTICOLOR feladata mindössze a FFB-ről MC–(többszínű)-re történő átkapcsolás és a megfelelő színek definiálása.

A MULTI utasítást követő három paraméterben a négy színregiszter közül háromnak a tartalmát határozhatjuk meg, amelyek a grafika pontjainak színét fogják tárolni. Hogy melyik ponthoz melyik regiszter (ill. szín) tartozik, azt a pont 2-bites információja rögzíti: 01 = 1. regiszter, 10 = 2. regiszter, 11 = 3. regiszter. A háttérszínt tartalmazó regiszter

paramétere hiányzik. Ezt a POKE 53248+33, 0. szín – utasítással vagy a COLOUR-ral határozhatjuk meg.
Az egyes színregiszterek címével és elhelyezkedésével a bevezetőben már részletesen megismerkedtünk (1. és 2. szín video-RAM-ban, 3. szín a szín-RAM-ban).
Mivel a 3. színt a szín-RAM, azaz a szöveg-üzemmód színtárolója tartalmazza, a MULTICOLOR üzemmód bekapcsolása után a szöveg-ablakban megjelenő karakterek, akár szövegről, akár grafikáról van szó, a 3. regiszterben tárolt színt fogják felvenni. Erről könnyen meg is győződhetünk, ha beadunk néhány karaktert, majd a

MULTI 1,2,3

parancsot. Változtassuk meg most a 3. paramétert és látni fogjuk, hogy az esetleg sok munkával összeállított színes szöveget egyetlen utasítással tönkretehetjük. Természetesen ez fordítva is igaz. Ha tehát grafikakészítés közben a "háttérben" szöveget írunk (ez is lehetséges), akkor ezeken a helyeken megváltozik a 3. szín. Ez persze nem csak hátrány, hanem előny is lehet, ha ügyesen kihasználjuk.
A további vizsgálódást folytassuk a lemezről betöltött MULTI című programmal:

P. 57

```
1 REM **************
2 REM * M U L T I *
3 REM **************
130 REM
140 COLOUR 0,6
150 HIRES 7,6:REM GRAFIKA BEKAPCSOLASA,TORLESE ES HATTERSZIN=KEK
160 AD=60000:REM KIINDULASI PONT BAZISCIM
170 FOR X=0 TO 7
180 POKE X+AD,%00000000:REM 1 CSIK A 0-AS (HATTER)SZINNEL
190 NEXT X:REM KOVETKEZO BYTE
200 FOR X=8 TO 15
210 POKE X+AD,%01010101:REM 1 CSIK AZ 1.SZINNEL
220 NEXT X:REM KOVETKEZO BYTE
230 FOR X=16 TO 23
240 POKE X+AD,%10101010:REM 1 CSIK A 2.SZINNEL
250 NEXT X:REM A KOVETKEZO BYTE
260 FOR X=24 TO 31
270 POKEX+AD,%11111111:REM 1 CSIK A 3.SZINNEL
280 NEXTX:REM A KOVETKEZO BYTE
290 REM
300 PAUSE 5:REM MEG CSAK AZ FFB VAN BEKAPCSOLVA!
310 FOR X=0 TO 15
320 MULTI X,7,5:REM MULTICOLOR BEKAPCSOLASA ES A 3 SZIN KIVALASZTASA:
330 REM 1. SZIN: VALTOZO
340 REM 2. SZIN: SARGA
350 REM 3. SZIN: ZOLD
360 PAUSE 1
370 NEXT X
380 REM
390 REM A KOVETKEZO SZIN VILLOGTATASA:
400 FOR X=0 TO 15
410 MULTI 7,X,5:REM MULTICOLOR BEKAPCSOLASA ES A 3 SZIN KIVALASZTASA:
420 REM 1.SZIN: SARGA
430 REM 2.SZIN: VALTOZO
440 REM 3.SZIN: ZOLD
450 PAUSE 1
460 NEXT X
490 REM
500 REM A KOVETKEZO SZIN VILLOGTATASA:
510 FOR X=1 TO 15
520 MULTI 7,5,X:REM MULTICOLOR BEKAPCSOLASA ES A 3 SZIN KIVALASZTASA:
530 REM 1.SZIN: SARGA
540 REM 2.SZIN: ZOLD
550 REM 3.SZIN: VALTOZO
560 PAUSE 1
570 NEXT X
```

Láthatjuk, hogy a grafika különböző részei a színek szempontjából külön-külön vezérelhetők. A grafikatárolóba a 180., 210., 240., és 270. sorokban írtuk be a megfelelő értékeket, de a PLOT-...stb. utasításokkal rajzolhattunk is volna valamint az adott színnel. 2-2 bit (ill. itt számjegy) értéke határozza meg azt, hogy melyik pont melyik színregiszterhez tartozik, azaz melyiknek a színét veszi fel. Tanulmányozzuk alaposan a programnak ezt a részét, módosítsuk a sorokat és figyeljük meg a következményeit. Csak így érthetjük meg igazán a leírtakat!

12.2.3 NRM – a nagybetű-/grafikus jelkészlet bekapcsolása

Formátum: NRM
Paraméterek: –
Funkció: Átkapcsolás grafikus-üzemmódról a nagybetű-/grafikus jelkészletre

Ezt az utasítást a 9.3.3 pontban már részletesen ismertettük, ezért itt csak megemlítjük. Pillanatnyilag csak a grafikus üzemmódot kikapcsoló funkciója érdekes a számunkra, ezt illusztrálja a lemezről betöltött NRM című program:

P. 58

```
1 REM ***********
2 REM * N R M *
3 REM ***********
130 REM
140 PRINT CHR$(14):REM ATKAPCSOLAS NAGY/KISBETUKRE
150 PRINT CHR$(147):REM SZOVEG-KEPERNYO TORLESE
160 CENTRE "MOST A KIS/NAGYBETUS":PRINT
170 CENTRE "UZEMMOD VAN ERVENYBEN":PRINT
180 PRINT
190 CENTRE "FIGYELJE A KOVETKEZO":PRINT
200 CENTRE"UZENETET!"
210 PAUSE 9
220 PRINT CHR$(147):REM SZOVEG-KEPERNYO TORLESE
230 CENTRE "EZ A SZOVEGOLDAL":REM UZENET A SZOVEGOLDALON
240 REM
250 FOR Y=1 TO 5
260 PAUSE 2
270 HIRES 7,6:REM FFB BEKAPCSOLASA+TORLES (PONTSZIN=SARGA) HATTER=KEK
280 FOR X=59264 TO 59583:REM EGY SORT JELENT A KEPERNYON
290 POKEX,%11110000:REM CSIKOK RAJZOLASA
300 NEXT X
310 PAUSE 2
320 NRM:REM SZOVEG-UZEMMOD VISSZAKAPCSOLASA NAGYBETU-/GRAFIKUS JELKESZLETRE
330 NEXT Y:REM 5-SZOR VALTAS
340 PRINT:PRINT:REM 2 SOREMELES
350 CENTRE "AMINT LATJUK, AZ NRM":PRINT
360 CENTRE"NAGYBETUKRE VALTOTT AT"
370 REM VALTOZTASSA MEG A %11110000 ERTEKET A 290.SORBAN PL.%11000000-RE
380 REM VAGY %11001100-RA!
```

12.2.4 CSET 2 – a grafikus üzemmód visszakapcsolása törlés nélkül

Formátum: CSET 2
Paraméterek: –
Funkció: Bekapcsolja a grafikus kijelzést anélkül, hogy a grafikus tárolót törölné

A HIRES utasításnál már említettük, hogy van olyan SIMON's BASIC utasítás is, amely anélkül kapcsol át szöveg-üzemmódról grafikusra, hogy közben a grafikus tárolót törölné. Nos, ez a CSET 2 utasítás. A CSET utasításnak a későbbiek során (13. fejezet) még két változatával fogunk megismerkedni (CSET 0 és CSET 1).
A CSET 2 újabb eszköz a grafikus műveletekhez. Feladata az, hogy a grafikus üzemmódból NRM-mel kapcsolt szöveg üzemmódot újból grafikusra váltsa, de eközben az előzőleg

kialakított grafikus képet ne törölje, tehát az megjeleníthető legyen anélkül, hogy ismételten elő kelljen állítani.
A program során tehát egy grafika rajzolása közben NRM-mel (vagy CSET 0/1-gyel) szöveg-üzemmódra kapcsolhatunk, kommentárokat és egyebeket fűzhetünk hozzá, majd újra lehívhatjuk a grafikát. A színekkel sem kell törődnünk, mert azok is megmaradnak. Úgy is alkalmazhatjuk, hogy a szöveg kijelzése közben a "háttérben" elkészíthetjük a grafikát és a megfelelő időpontban megjelenítjük.
A számos alkalmazási lehetőség közül a példaprogramunk egyet mutat be:

P. 59

```
1 REM **************
2 REM * C S E T  2 *
3 REM **************
130 REM
140 PRINT CHR$(147):REM SZOVEG-KEPERNYO TORLESE
150 CENTRE "EZ A SZOVEGOLDAL":REM UZENET A SZOVEGOLDALON
160 PAUSE 2
170 HIRES 7,2:REM FFB BEKAPCSOLASA+TORLES(PONTSZIN=SARGA) HATTERSZIN=VOROS
180 FOR X=59264 TO 59583:REM EGY SOR A KEPERNYON
190 POKEX,%11110000:REM CSIK RAJZOLASA
200 NEXT X
210 FOR X=1 TO 5
220 PAUSE 2
230 NRM:REM SZOVEG-UZEMMOD VISSZAKAPCSOLASA NAGYBETU-/GRAFIKUS JELKESZLETRE
240 PAUSE 2
250 CSET 2:REM GRAFIKUS UZEMMOD VISSZAKAPCSOLASA TORLES ES SZINBEALLITAS NELKUL
260 NEXT X
```

Mint látjuk, lehetőségünk van a grafikus és szövegüzemmódban történő párhuzamos munkára. Érdemes ezt a programot is részletesebben tanulmányozni, különös tekintettel a 190. sorra. A bináris érték változtatásával végezzünk néhány kísérletet.

12.2.5 LOW COL – a színek újradefiniálása

Formátum: LOW COL 1.szín, 2.szín, 3.szín
Paraméterek: 1.szín – FFB: pontszín
MC: színmeghatározás az 1.színregiszter számára
2.szín – FFB: háttérszín
MC: színmeghatározás a 2.színregiszter számára
3.szín – FFB: nincs funkciója
MC: színmeghatározás a 3.színregiszter számára
Funkció: A pontok színének újradefiniálása
Példa: LOW COL 3,6,2

Eddigi ismereteink alapján felmerülhet a kérdés, hogy valamennyi pont csak a HIRES-sel vagy a MULTI-val definiált egy ill. három színben jeleníthető-e meg. Nos, nem. Létezik egy olyan SIMON's BASIC utasítás, amellyel a következőkben létrehozandó pontok számára új színeket határozhatunk meg. Ez az utasítás a LOW COL. Az új színkombinációt az utasítás három paraméterével adjuk meg. FFB-ben csak az első kettőnek van jelentősége (pontszín és háttérszín), mivel minden 8 × 8-as mátrix-ra csak egy pontszín és egy háttérszín adható meg. Az első érték jelöli a pont színét, a második pedig a háttér színét (mint ahogy ezt már a 12.1 pontban olvastuk, minden 8 × 8-as mátrix külön háttérszínnel rendelkezhet, mivel ezek az értékek szintén a video-RAM-ból (grafikus-színtároló származnak.) Erre a 2. HICOL példaprogramban még visszatérünk.
Ettől eltekintve, ha nem akarunk SYNTAX ERROR-t előidézni, FFB-ben is mindhárom paramétert be kell adnunk. Az utolsót a számítógép nem veszi figyelembe, így az bármilyen érték lehet.
Többszínű (MC) üzemmódban a 2. és 3. szín a 8 × 8-as (ill. 4 × 8-as) mátrix további két pontszínét jelentik, hiszen ebben az esetben a háttérszínt a COLOUR utasítással egyszer már megadtuk.
Első pillantásra egyszerűnek tűnik ez az utasítás, de majd látni fogjuk, hogy nem is olyan

könnyű bánni vele. A pont vagy grafika rajzolása ugyanis nem azonos a LOW COL előtt és után:
A grafikus üzemmód HIRES-sel történő bekapcsolásakor vagy MC-re történő átkapcsolásakor a képernyő összes pontjainak színe már definiálva és a megfelelő színregiszterek tárolórekeszeiben rögzítve van. Így rajzoláskor nem kell a pontok színét külön megadni, mert miután a grafikus tárolóba egy pontot bevittünk, az automatikusan felveszi a színét a megfelelő színtárolóból.
A LOW COL utasítás alkalmazásával más a helyzet. Ilyenkor a színekre vonatkozó információkon felül a színtárolóba is be kell írni a megfelelő pontszínt, azaz a grafikus tárolóba a pont előállításán (vagy tárolását) kívül még be kell írni FFB esetén a vonatkozó tárrekeszbe, MC esetén pedig a három színregiszterbe a színt is. Mivel a színregiszter egy tárolórekesze mindig egy 8 × 8-as (ill. 4 × 8-as) mátrixot vezérel, annak esetleg már bekapcsolt pontjai felveszik a LOW COL-lal megadott új színt, függetlenül a korábbi előírásoktól. Ha ezután ebbe a mátrix-ba LOW COL nélkül (azaz párhuzamos színdefiniálás nélkül) írunk, akkor a pontok ismét az eredetileg meghatározott színben fognak megjelenni. A LOW COL hatását a HI COL-lal oldhatjuk fel, amit a mintaprogramon majd részletesen is tanulmányozhatunk.
Lényeges probléma az is, hogy két, egymást megközelítő, sőt keresztező alakzat közül melyiknek a színe érvényesüljön. Úgy kell elképzelni, mintha az egyik (a fontosabb) takarná a másikat.
Kétféle megoldás is kínálkozik:

1. Elsőként a háttérbe kerülő alakzatot rajzoljuk meg, utána az előtérbe kerülőt. Így a második rajz készítésekor a megfelelő pontok ennek a színét veszik át.
2. A háttérbe kerülő alakzatot LOW COL utasítás nélkül, míg az előtérbe kerülőt LOW COL-lal ábrázoljuk.

Az első esetben fontos a rajzolási sorrend betartása, a másodikban viszont lényegtelen.
Töltsük be most a mintaprogramot és indítsuk el:

P. 60

```
1 REM ********************
2 REM * 1.L O W  C O L *
3 REM ********************
130 REM
140 COLOUR 0,1:REM KERETSZIN=FEKETE
150 AB=8:REM KET VONAL TAVOLSAGA
160 REM
170 FOR Z=1 TO 2:REM KET ATFUTAS
180 HIRES 7,0:REM FFB BEKAPCSOLASA+TORLES(PONTSZIN=SARGA) HATTER=FEKETE
190 FOR Y=1 TO 15
200 FOR X=1 TO 320
210 PLOT X,Y*AB,1:REM PONTSOR RAJZOLASA (LD.A PLOT UTASITAST)
220 NEXT X:REM 320 PONT (EGY SOR) RAJZOLASA
230 LOW COL Y,6,0:REM 2.SZIN MEGHATAROZASA - AZ UTOLSO PARAMETER FFB-BEN HAT.LAN
240 NEXT Y:REM SZINVALTAS
250 PAUSE 5
260 NRM:REM ATKAPCSOLAS SZOVEG-UZEMMODRA
270 PRINT CHR$(147):REM KEPERNYO TORLESE
280 CENTRE"NEZZUK MEG MI TORTENIK,HA":PRINT
290 CENTRE "A VONALAK TUL KOZEL KERULNEK":PRINT
300 CENTRE "EGYMASHOZ!":PRINT
310 PRINT
320 CENTRE "AZ ALACSONY SZINFELBONTAS":PRINT
330 CENTRE"OKOZZA!"
340 PAUSE 9
350 AB=2:REM VONALTAVOLSAG
360 NEXT Z:REM RAJZOLAS UJBOL, KISEBB VONALKOZZEL
```

A program 210. sorában egy még ismeretlen utasítással, a PLOT-tal találkozunk. Funkciója egy pont megjelenítése vagy törlése a képernyőn az x,y koordinátákkal megadott pozícióban. Részletes ismertetésére a 12.3.1 pontban térünk ki.

A példában két lényeges FOR...NEXT ciklus van. Az egyik az y változó értékeit és ezzel együtt a 15 vonal 15 színét adja meg a LOW COL utasításban. A másik a Z léptetésre, ami a program kétszeri, különböző vonaltávolságokkal történő lefutását eredményezi. Láthatjuk, hogy a második esetben a vonalak olyan közel kerülnek egymáshoz, hogy a később rajzolt vonal színét felveszi néhány előző is, vagyis azok a "háttérbe" kerülnek.

12.2.6 HI COL – a LOW COL feloldása

Formátum: HI COL
Paraméterek: –
Funkció: A LOW COL utasítás feloldása – rajzolás az eredeti színekkel

Mielőtt ezzel az utasítással megismerkednénk, ismételjük át a LOW COL utasításról tanultakat.
Mint azt a 12.2.5 pontban (LOW COL) láttuk, a HIRES-utasítással bekapcsolt grafikus üzemmódban minden előállított pontnak előre meghatározott színe van. A LOW COL beírása előtt tehát az előre beállított színekkel rajzolunk, azaz minden előállított pont a video-RAM-ban (grafikus-színtároló) és MC esetén még a szín-RAM-ban (szöveg-színtároló) tárolt színeket veszi fel.
A LOW COL-nál láttuk, hogy ez a színkezelés okoz néhány problémát, de úgy látszik nincs ésszerűbb megoldás.
Ahhoz, hogy a C 64-es színlehetőségeit minél alaposabban kihasználhassuk, szükségünk van egy olyan utasításra, amely a 12.2.5 pontban ismertetett LOW COL hatását feloldja. Ez a paraméterek nélküli HI COL utasítás, amely értelemszerűen csak egy megelőző LOW COL-lal együtt alkalmazható.
A következő példán keresztül talán sikerül megvilágítani az utasításpár színkezelésben betöltött fontos szerepét:

P. 61

```
1 REM **************
2 REM * H I   C O L *
3 REM **************
100 REM
110 COLOUR 0,1: REM KERETSZIN=FEKETE
120 REM
130 HIRES 6,0: REM FFB BEKAPCSOLASA+TORLES (PONTSZIN=KEK) HATTER=FEKETE.
140 FOR Y=0 TO 11
150 FOR X=0 TO 319
160 PLOT X,Y*16,1: REM PONTSOR RAJZOLASA
170 NEXT X: REM 320 PONT (EGY SOR) RAJZOLASA
180 LOW COL Y+1,0,0:REM PONTSZIN=FEHER (HATTER=FEKETE)3.PAR. FFB-BEN HATASTALAN
190 FOR X=0 TO 319
200 PLOT X,Y*16+8,1:REM PONTSOR RAJZOLASA
210 NEXT X: REM 320 PONT (EGY SOR) RAJZOLASA
220 HI COL : REM VISSZAKAPCSOLAS AZ EREDETI SZINOSSZEALLITASRA
230 NEXT Y: REM SZINVALTAS
240 PAUSE 5
250 NRM:REM  VISSZAKAPCSOLAS SZOVEG-UZEMMODRA
260 PRINT CHR$(147): REM KEPERNYO TORLESE
270 CENTRE"FIGYELJUK MEG,HOGY":PRINT:PRINT
280 CENTRE"A LOW COL ES HI COL"    :PRINT:PRINT
290 CENTRE"UTASITASOK MILYEN HATASSAL"    :PRINT:PRINT
300 CENTRE"VANNAK A SZINEKRE ES A KULONBOZO":PRINT:PRINT
310 CENTRE"SZINU VONALAK EGYMASRA !"
320 PAUSE 10
330 CSET 2:REM GRAFIKUS UZEMMOD BEKAPCSOLASA TORLES NELKUL
340 FOR Y=0 TO 9
350 FOR X=0 TO 199
360 PLOT Y*32,X,1:PLOT Y*32+1,X,1:REM DUPLASZELESSEGU
370 REM PONTOSZLOP RAJZOLASA
380 NEXT X:REM 200 PONT (EGY OSZLOP) RAJZOLASA
```

```
390 LOW COL Y,0,0:REM 2 UJ SZIN - A 3. PARAMETER FFB-BEN HATASTALAN
400 FOR X=0 TO 199
410 PLOT Y*32+16,X,1:PLOT Y*32+17,X,1:REM DUPLASZELESSEGU
420 REM PONTOSZLOP RAJZOLASA
430 NEXT X:REM 200 PONT (EGY OSZLOP) RAJZOLASA
440 HI COL: REM RAJZOLAS AZ EREDETI SZINEKKEL
450 NEXT Y: REM SZINVALTAS
460 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

Észrevehetjük, hogy ez a program szoros összefüggésben áll az elõzõvel, így megértése is egyszerûbb lesz.

Néhány szó a programról:
A képernyõn vízszintes, majd ezeket keresztezõ függõleges vonalak jelennek meg. Az elsõ vízszintes vonalat a 150–170. sor rajzolja a HIRES utasításban elõírt általános pontszínnel, a kékkel. A 180. sorban a pontok számára az Y+1 kifejezéssel új színt határozunk meg (a háttér továbbra is 0, azaz fekete), amely a ciklusnak megfelelõen 12-szer változik. A 220. sorban a HI COL-lal visszakapcsolunk az eredeti színösszeállításra. A 140–230. sorokban tehát az 1–12 kódú színekkel egy-egy vízszintes vonalat (pontsort) rajzolunk, váltakozva egy eredeti (kék) színû vonallal. (Próbáljuk kitalálni, mi történik, ha a HI COL utasítást elhagyjuk!)
A 330. sortól kezdve függõleges vonalakat rajzolunk egymástól 16 pont távolságra egyszer az általános kék színben, egyszer pedig a LOW COL-ban (180. sor) Y-nal megadott, váltakozó színekben. Figyeljük meg, hogy a LOW COL-lal rajzolt vonalak színe milyen hatással van a környezõ pontok színeire, valamint a már meglévõ pontok színe hogy befolyásolja az eredeti (kék) színben megjelenõ pontok színét. (A függõleges vonalakat a jobb láthatóság érdekében dupla szélességben ábrázoltuk.
A következõ példaprogramot tulajdonképpen a LOW COL utasításnál kellett volna ismertetnünk, de az elõforduló HI COL utasítások miatt itt térünk rá:

P. 62

```
1 REM ******************
2 REM * 2.L O W  C O L *
3 REM ******************
100 REM
110 HIRES 5,6:REM GRAFIKA BEKAPCSOLASA+TORLES (HATTER=KEK) PONT=ZOLD
120 LOW COL 1,5,0 :REM UJ SZINEK MEGADASA (HATTER=ZOLD) PONT=FEHER
130 REM
140 REM VIZSZINTES RAJZOLAS AZ UJ SZINEKKEL
150 FOR X=0 TO 319
160 PLOTX,66,1:REM PONTOK RAJZOLASA
170 NEXT X
180 PAUSE 2
190 LOW COL 0,1,0:REM UJ SZIN MEGADASA (HATTER=FEHER) PONT=FEKETE
200 REM
210 REM TORLES VIZSZINTESEN AZ UJ SZINEKKEL
220 FOR X=0 TO 319
230 PLOT X,133,0:REM PONTOK TORLESE
240 NEXT X
250 PAUSE 2
260 HI COL:REM EREDETI SZINOSSZEALLITAS VISSZAKAPCSOLASA
270 REM
280 REM FUGGOLEGES RAJZOLAS AZ EREDETI SZINEKKEL(HATTER=KEK) PONT=ZOLD
290 FOR Y=0 TO 199
300 PLOT 160,Y,1: REM PONTOK RAJZOLASA
310 NEXT Y
320 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

Ebben a példában a LOW COL utasítást az elõzõektõl eltérõen, a háttérszín definiálásának szempontjából vizsgáljuk.
Látjuk, hogy 120. és 190. sorokban a pontszín mellett a háttérszínt is módosítottuk. Ez lehetséges, hiszen a háttérszín – mint ahogy azt a 9.1. fejezetben az alapok ismertetésé-

nél már említettük – a pontszínhez hasonlóan a video-RAM-ból (grafikus színtároló) származik, tehát minden 8 × 8-as mátrix esetében más és más lehet.
A második, amit ezzel a példával szemléltetünk, hogy a színek meghatározása szempontjából közömbös, hogy az adott pontot rajzolni vagy törölni fogjuk, mert mint tudjuk, ezt a PLOT utasításban szereplő rajzoló-üzemmód kódja határozza meg. A törlés tehát azt jelenti, hogy módosul a színregiszterben lévő szín. Emiatt ha az eredeti színt törléssel akarjuk visszaállítani, előtte a LOW COL színeket olyan értékkel kell előírnunk, amelyek a normál HI COL színeknek (amelyeket a HIRES utasítással írtunk elő) felelnek meg.

12.2.7 Ellenőrző kérdések

Mivel ebben a fejezetben egy meglehetősen komplikált témát tárgyaltunk, az ellenőrző kérdések is nehezebbek lesznek. Nem szabad tehát elkeseredni, ha valami nem sikerül. Ilyenkor olvassuk át ismét figyelmesen a vonatkozó részeket.

1. Hogy kell inicializálni a többszínű grafikát, ha a grafikus üzemmódot még nem kapcsoltuk be?

   a) HIRES...
   b) HIRES...: MULTI...
   c) CSET 2:MULTI...
   d) CSET 2

2. Mi az NRM utasítás hatása?

   a) Csak a grafikus üzemmód kikapcsolása.
   b) A grafika kikapcsolása és a nagy-/kisbetűs karakterkészlet bekapcsolása.
   c) A grafika kikapcsolása és a nagybetű-/grafikus jelkészlet bekapcsolása.
   d) Az eredeti színösszeállítás visszakapcsolása.

3. A LOW COL utasítás beírása után el akarunk helyezni egy pontot a képernyőn. Mit kell figyelembe vennünk?

   a) Csak egy színváltás történik.
   b) Egy-egy 8 × 8-as mátrix minden pontjának színe azonos lesz.
   c) A pont háttérszíne is meg fog jelenni.
   d) A pontszín száma eggyel csökken.

12.3 Grafikai utasítások

Az előző részben tárgyalt meglehetősen nehéz téma a tulajdonképpeni grafikus munka, a grafikai utasítások megismerésének előfeltétele volt. Ezután már kifejezetten csak grafikával, pontok, vonalak, körök, mezők és sok egyéb alakzat rajzolásával fogunk foglalkozni. Minden utasítás, amellyel munkánk során találkozni fogunk, segít bennünket abban, hogy saját magunknak szép grafikákat állítsunk elő.
Lássunk hozzá és próbáljuk ki, lépjünk be ebbe az ismeretlen világba. Most lehetőségünk van olyan dolgokat alkotni, amiket ezelőtt még senki nem látott a képernyőn. Itt kezdődik a művészet, a formatervezés. A grafikával játékokat is készíthetünk, amelyhez hatékony segítséget nyújtanak majd a későbbiekben tárgyalt sprite utasítások.
Röviden: rátérünk fejezetünk központi témájára, a rajzoló utasításokra.
Az utasításoknál szereplő esetenként változó és az adott helyen ismertetett paraméterek mellett minden esetben alkalmazhatjuk a már ismert rajzoló-üzemmód kódot is, amit a továbbiakban rü-vel jelölünk. Ezzel határozhatjuk meg, hogy a figurát rajzolni, törölni vagy invertálni akarjuk-e. Az utasítások MULTICOLOR-ban is működnek, így az ábra megjelenítése terén már egész sor lehetőségünk van.
SIMON's BASIC-ben a kép beosztása az x–y koordináták szerint értelemszerűen, a 12.2 pontban leírtaknak megfelelően történik.
Végül egy jótanács: Próbáljuk ki a példák nyújtotta összes lehetőséget. Nem fogjuk megbánni!
Nos, sok sikert és jó szórakozást!

12.3.1 PLOT – pont rajzolása

Formátum: PLOT x,y,rü
Paraméterek: x – Az adott pont X-koordinátája
MC: 0–159/FFB: 0–319
y – A pont Y-koordinátája
MC: 0–199/FFB: 0–199
rü – A rajzoló üzemmód kódja (0–2)
Funkció: A képernyőn elhelyezi (törli vagy invertálja) az X–Y koordinátájú pontot
Példa: PLOT 160,100,1
A képernyő (160,100) koordinátákkal meghatározott, pozíciójában egy pont jelenik meg.

A PLOT a legegyszerűbb rajz utasítás, amellyel belépünk a grafika világába. Mindössze arra alkalmas, hogy a képernyőn kijelezzen (töröljön vagy invertáljon) egy pontot, amelynek helyét az X–Y koordinátákkal adjuk meg. A képzeletbeli koordináta-rendszer (0;0)-pontja a képernyő bal felső sarkában van.
A rü-paraméterrel írjuk elő a pont ábrázolási módját. Értelmezése a következő:

rü	jelentése
0	törlés
1	rajzolás
2	invertálás

(Az invertálás a bekapcsolt pont kikapcsolását és a kikapcsolt pont bekapcsolását jelenti.)
Ez az utasítás a koordináták és az rü-paraméter változtatásával még könnyen gyakorolható, hatásuk kipróbálható.
Végezetül töltsük be a PLOT című programot, amely egyszerre két példát is hoz az utasítás működésének bemutatására.

P. 63

```
1 REM ***********
2 REM * P L O T *
3 REM ***********
100 REM
110 HIRES 1,4:REM GRAFIKA BEKAPCSOLASA (HATTER=IBOLYA) PONT=FEHER
120 FOR X=1 TO 319 STEP .5
130 PLOT X,60*SIN(X/25)+100,1
140 NEXT X
150 PAUSE 5
160 HIRES 1,4 :REM GRAFIKA TORLESE
170 FOR Y=1 TO 0 STEP -1:REM 1.) RAJZ 2.)TORLES
180 FOR X=103 TO 290 STEP .5
190 PLOT 40*SIN(X/30)+160,50*SIN(X/25)+100,Y
200 NEXT X
210 PAUSE 1
220 NEXT Y :REM ISMET TORLES
230 WAIT198,255:REM VARAKOZAS GOMBNYOMASRA
```

A példa első részében egy egyszerű szinuszgörbe ábrázolásán keresztül mutatjuk be a PLOT működését. Érthetőbbé válik, ha néhányszor megváltoztatjuk a paramétereket! A program második részében (160. sortól) egy érdekes alkalmazási lehetőséggel találkozunk, ami egyben az rü-paraméter hatását is szemlélteti. (Értékeit az Y-változó tárolja.) A 230. sorral már több helyen is találkoztunk. Rendeltetése a program megállítása, befejezés nélkül. Ha elhagynánk, a program végén eltűnne a grafika és visszatérne a szöveg-képernyő. Ezzel az utasítással azt érjük el, hogy a grafika mindaddig a képernyőn marad, míg egy gombot le nem nyomunk.

12.3.2 TEST – egy pont állapotának vizsgálata

Formátum: TEST (X,Y)
Paraméterek: X – A pont X-koordinátája
MC: 0–159/FFB: 0–319
Y – A pont Y-koordinátája
MC: 0–199/FFB: 0–199

Funkció: Ellenőrzi, hogy az (X,Y)-koordinátákkal jelzett pont be van-e kapcsolva

Tételezzük fel, hogy a programunk akár a PLOT, akár más utasítással a képernyőn kijelez egy pontot. Valamilyen okból a későbbiek során tudni szeretnénk, hogy a pont éppen bekapcsolt állapotban van-e (metszéspontok, ütközések, behatárolt képrészek). Ennek megállapítására szolgál a TEST utasítás. A pontot meghatározó (x,y)-koordinátákat egyben argumentumként értelmezzük és a TEST utasítást függvényként alkalmazzuk. Amennyiben a kérdéses pont bekapcsolt állapotban van, értéke 1, ellenkező esetben 0. Ezt a függvényértéket a későbbiekben aritmetikai kifejezésekben is felhasználhatjuk, az alábbiak szerint:

```
A= TEST(0,0)
B=B*5 + TEST(0,10)
```

Vagyis A értéke 1, ha a (0;0)-koordinátájú pont bekapcsolt állapotban van és 0, ha nincs. A B értéke 15, ha a (0;10) koordinátájú pont nincs bekapcsolva és 16 az ellenkező esetben.
Az utasítás kiválóan alkalmazható játékprogramokban, ezért a szokásos példaprogramon kívül egy kis játékprogramot is készítettünk, ami kizárólag a már ismert utasításokkal működik és bár rövid, mégis szórakoztató.
Töltsük be először az 1. TEST című programot:

P. 64

```
1 REM **************
2 REM * 1. T E S T *
3 REM **************
100 REM
110 HIRES 6,7:REM GRAFIKA BEKAPCSOLASA+ TORLES
120 T1=TEST(100,100):REM ELLENORZES BEKAPCSOLAS ELOTT
130 PAUSE 3
140 PLOT 100,100,1:REM PONT BEKAPCSOLASA
150 T2=TEST(100,100):REM ELLENORZES BEKAPCSOLAS UTAN
160 PAUSE 3
170 NRM:REM GRAFIKA KIKAPCSOLASA
180 PRINT CHR$(147):REM KEPERNYOTORLES
190 PRINT "BEKAPCSOLAS ELOTTI ALLAPOT:"T1
200 PRINT "BEKAPCSOLAS UTANI ALLAPOT :" T2
```

Ha áttanulmányoztuk, jöhet egy kis szórakozás:

P. 65

```
1 REM **************
2 REM * 2. T E S T *
3 REM **************
100 REM
140 REM ************
150 REM SUPER-KIGYO
160 REM ************
170 REM
180 COLOUR 10,1:REM KERET VILAGOSPIROS
190 HIRES 1,2:REM GRAFIKA BEKAPCSOLASA (HATTER=VOROS) PONT=FEHER
195 NRM:PRINT"□  ■■FEL    = I"
196 PRINT"  ■■LE      = M"
197 PRINT"  ■■BALRA  = A"
198 PRINT"  ■■JOBBRA = D"
200 INPUT"  ■■■SEBESSEG (1-9) :";G
205 CSET2
210 REM ************************
220 X=-G:Y=0:REM STARTIRANY BALRA
230 A=160:B=100:REM STARTKOORDINATAK
240 GET A$:REM BILLENTYU LEHIVASA
250 IF A$="I" THEN X=0:Y=-G:REM FEL
260 IF A$="M" THEN X=0:Y=G:REM LE
270 IF A$="A" THEN X=-G:Y=0 :REM BALRA
280 IF A$="D" THEN X=G:Y=0:REM JOBBRA
290 A=A+X:B=B+Y:REM TOVABBMOZGATAS
300 IF A<0 OR A>319 OR B<0 OR B>199 THEN GOTO 340
310 IF TEST(A,B)=1 THEN GOTO 340
320 PLOTA,B,1 :REM PONT ELHELYEZESE
330 GOTO 240
340 REM
350 REM ********************
360 REM U T K O Z T E T E S
370 REM ********************
380 REM
390 FOR C=1 TO15
400 FOR D=1 TO 15
410 COLOUR D,1:REM KERET VILLOGTATASA
420 NEXT D
430 NEXT C
440 A= INT(RND(1)*319/G)*G
450 B= INT(RND(1)*199/G)*G
460 COLOUR 10,1
470 Z=Z+1:REM SZAMLALO
480 IF Z=20 THEN GOTO 500
490 GOTO 240
```

```
500 REM
510 REM *******
520 REM V E G E
530 REM *******
540 REM
550 NRM:REM GRAFIKA KIKAPCSOLASA
560 PRINT CHR$(147):REM KEPERNYOTORLES
570 PRINT AT(0,10)"":REM KURZORBEALLITAS
580 CENTRE "SEMMI PANIK!"
590 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

A játék a következőképpen működik:
A képernyőn megjelenik egy pont, amely a választott sebességtől függő gyorsasággal egy hosszú kígyóvá növekszik. Mozgását az "I", "M", "A" és "D" billentyűkkel irányíthatjuk. Ha a kígyó nekiütközik a képernyő szélének vagy egy másik kígyónak, vagy mozgásával éppen ellenkező irányban akarjuk mozgatni, "elpusztul". A keret villogni kezd, majd néhány másodperc múlva újabb kígyó jelenik meg. 20 ütközés után a játék befejeződik. A játék persze tovább finomítható, szépíthető. Játszhatják például ketten, a vezérlés történhet joystick-kal és az elért pontszám is kiíratható. A program kis búvárkodással könnyen megérthető és módosítható.
Térjünk vissza azonban a TEST utasításra, hiszen most leginkább erre vagyunk kíváncsiak. A 310. sorban találkozunk vele. Láthatjuk, hogy az (X,Y) koordinátákat az A és B változók tárolják. Ha a függvény értéke 1., azaz a kérdéses pont be van kapcsolva, ütközés történt, a program a 340. sorban folytatódik. Ha 0, újabb pontot helyez el, azaz a kígyó tovább nő.

12.3.3 LINE – egyenes rajzolása

Formátum: LINE x1, y1, x2, y2, rü
Paraméterek: x1 – a vonal kezdőpontjának x-koordinátája
MC: 0–159/FFB: 0–319
y1 – A vonal kezdőpontjának y-koordinátája
MC: 0–199/FFB: 0–199
x2 – A vonal végpontjainak x-koordinátája
MC: 0–159/FFB: 0–319
y2 – A vonal végpontjának y-koordinátája
MC: 0–199/FFB: 0–199
rü – A rajzoló üzemmód kódja (0–2)
Funkció: A megadott pontok összekötése egyenessel
Példa: LINE 0,0,319,199,1
Megrajzolja a képernyő egyik átlóját.

A PLOT utasítással elméletileg bármilyen, általunk kigondolt alakzatot megrajzolhatunk. Gondoljuk el azonban, mennyi fáradságot és időt igényelne egy egyszerű vonal megrajzolása is, ha annak minden pontját egyenként kellene megadnunk. Ráadásul BASIC-ben ezt csak a két pontot összekötő egyenes egyenletének felhasználásával tudnánk kiszámítani, ami nagyon hosszú ideig tartana. Ezen nehézségek kiküszöbölésére született meg a SIMON's BASIC egy sor olyan utasítása, amelyek kész alakzatokat (vonal, ellipszis, négyszög) állítanak elő. Ezek egyike a LINE-utasítás.
A LINE módot ad arra, hogy egy egyenest egyszerűen, mindössze két végpontjának megadásával megrajzoltassunk (töröljünk vagy invertáljunk). Nincs is vele több gondunk, minden mást a SIMON's BASIC old meg. (A későbbiek során majd tapasztalni fogjuk, hogy a SIMON's BASIC "alakzat-készítő" utasításai – előnyeik mellett – bizony elég lassúak, de mint már említettük, léteznek speciális grafikai programok, amelyek ezen a

téren is kielégítően működnek.) Lényegében azonban maga a grafikus program könnyen és gyorsan megírható a rendelkezésre álló utasításokkal.
De térjünk vissza a LINE utasításra. Az első két paraméterrel (x1 és y1) a vonal kezdőpontját határozzuk meg, míg a második két paraméter (x2 és y2) a végpontot jelöli. A sorrend nem lényegtelen, mert ha pl. az előzőekben megrajzolt vonalat törölni akarjuk és a koordinátákat fordított sorrendben adjuk be, előfordulhat, hogy a vonal nem tűnik el teljesen. Ha betöltjük az 1. LINE című programot, megfigyelhetjük ezt a jelenséget.

P. 66

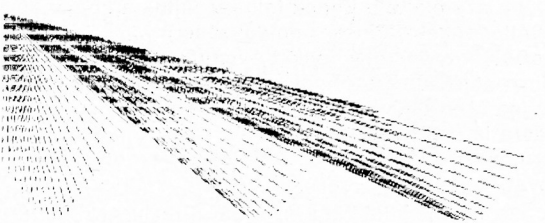
```
1 REM ************
2 REM * 1.L I N E *
3 REM ************
100 REM
110 HIRES 2,3:REM GRAFIKA BEKAPCSOLASA
120 FOR I=1 TO 2
130 REM
140 REM ************
150 REM * RAJZOLAS *
160 REM ************
170 REM
180 FOR X=1 TO 160 STEP 4
190 LINE 0,0,X,40*SIN(X/20)+100,1
200 NEXT X
210 IF I<>1 THEN GOTO 340
220 PAUSE 3
230 REM
240 REM ***************
250 REM * ROSSZ TORLES *
260 REM ***************
270 REM
280 PAUSE 3
290 FOR X=1 TO 160 STEP 4
300 LINE X,40*SIN(X/20)+100,0,0,0:REM A KOORDINATAK SORRENDJE ROSSZ !
310 NEXT X
320 PAUSE 10
330 NEXT I
340 PAUSE 3
350 REM
360 REM ************
370 REM * JO TORLES *
380 REM ************
390 REM
400 FOR X=1 TO 160 STEP 4
410 LINE 0,0,X,40*SIN(X/20)+100,0:REM A KOORDINATAK SORRENDJE HELYES !
420 NEXT X
430 WAIT198,255
```

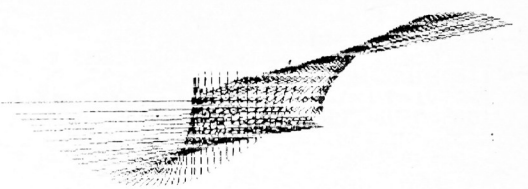
Az eredeti német nyelvű könyvben itt más példa szerepel, de mivel a "rossz" program is jól működött, azaz a vonalat teljesen kitörölte, készítettünk egy olyan programot, ami a leírtakat megfelelően szemlélteti. Ehhez felhasználtuk a könyv következő példaprogramját. (A magyar változat készítőinek megjegyzése.)
A programhoz nincs sok hozzáfűzni valónk. Láthatjuk, hogy ha a törléskor megváltoztatjuk a koordináták sorrendjét, a rajzból egy egész jelentős rész megmarad, míg azonos sorrend esetén a teljes ábra eltűnik.
Mindez az invertálásra is érvényes.
A LINE utasítással nagyon szép hatások érhetők el, amelyekből a következő példán mutatunk be néhányat.

P. 67

```
1 REM ************
2 REM * 2.L I N E *
3 REM ************
130 REM
140 REM ***********
150 REM   1. ABRA
160 REM ***********
170 REM
180 HIRES 6,7
190 FOR X=1 TO 319 STEP 4
200 LINE 0,0,X,40*SIN(X/20)+100,1
210 NEXT X
220 PAUSE 10
230 HIRES 4,1
240 REM
250 REM ***********
260 REM   2. ABRA
270 REM ***********
280 REM
290 FOR X=1 TO 350 STEP 1
300 A=50*SIN(X/10)+160
310 B=60*SIN(X/20)+100
320 C=70*SIN(X/30)+160
330 D=80*SIN(X/40)+100
340 LINE A,B,C,D,1
350 NEXT X
360 PAUSE 10
370 HIRES 0,3
380 REM
390 REM ***********
400 REM   3. ABRA
410 REM ***********
420 FOR X=1 TO 319 STEP 4
430 A=X
440 B=60*SIN(X/60)+100
450 C=40*SIN(X/25)+160
460 D=20*SIN(X/30)+100
470 LINE A,B,C,D,1
480 NEXT X
490 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

Láthatjuk, hogy a koordinátákat bonyolult kifejezésekkel is megadhatjuk. A példában szereplőket nem fontos megértenünk, hiszen úgyis könnyen változtathatók. A legkisebb módosítás is befolyásolja a megjelenített rajzolatot. Idővel – különösen, ha a matematikáját is értjük – már a grafika tervezése is sikerülni fog. (Pl. az első alakzatot úgy készítettük, hogy egy szinuszgörbe minden pontját összekötöttük a (0;0)-ponttal).





12.3.4 REC – téglalap rajzolása

Formátum: REC x,y,b,h,rü
Paraméterek: x – a téglalap bal felső sarkának x-koordinátája
MC: 0–159/FFB: 0–319
y – a téglalap bal felső sarkának y-koordinátája
MC: 0–199/FFB: 0–199
b – a téglalap szélessége
MC: 0–159/FFB: 0–319
h – a téglalap magassága
MC: 0–199/FFB: 0–199
rü – a rajzoló üzemmód kódja (0–2)
Funkció: Egy téglalap rajzolása
Példa: REC 20, 40, 100, 50, 1
Rajzol a képernyőn egy 100 pont széles, 50 pont magas téglalapot, amelynek bal felső sarka a képernyő (20;40)-pontjában van.

Egy további utasítás, amellyel kész alakzatot rajzolhatunk, a REC. Segítségével a képernyő tetszőleges helyén egy tetszőleges nagyságú téglalapot jeleníthetünk meg. Lehetővé válik pl. grafikák készítése a képernyő egy adott területére, vagy szöveg beírása a grafikába, oszlopdiagramok és más hasonló alakzatok készítése.
A téglalap helyzetét bal felső sarkának koordinátáival adjuk meg. Ez a pont egyben meghatározza a téglalap kiterjedését balra és felfelé. Lefelé és jobbra a két második paraméter a meghatározó.

```
A REC 0,0,310,199,1
```

utasítással a teljes képmezőt bekeretezhetjük.
Ügyelni kell arra, hogy csak olyan szélesség- és magasság-értékeket adhatunk be, amelyekkel a képmezőből nem lépünk ki. Helytelen például az alábbi utasítás:

```
REC 100, 150, 50, 80,1
```

Ha kipróbáljuk, láthatjuk, hogy a téglalap jobb oldala csak a képernyő széléig rajzolódik meg, míg a bal oldala felfelé túlnyúlik a kezdőponton. (Az utasítást természetesen csak úgy tudjuk kipróbálni, ha előtte HIRES-sel grafikus üzemmódra kapcsolunk és utána WAIT-tel megakadályozzuk a kép eltűnését.)
Ugyanez fordul elő, ha a szélességet választjuk meg helytelenül.
A következő példaprogramok bemutatnak néhány tippet és trükköt a REC utasítás alkalmazására.
Először azt mutatjuk be, hogy miképpen lehetséges a keretet egy pontnyinál szélesebbre rajzolni. A program 120. sorában a többi paraméterrel együtt határozzuk meg a keretvastagságot (DI), amelynek megrajzolása a 200. sorban kezdődő alprogramban történik. Ezt a rutint más programokba is beépíthetjük.

P. 68

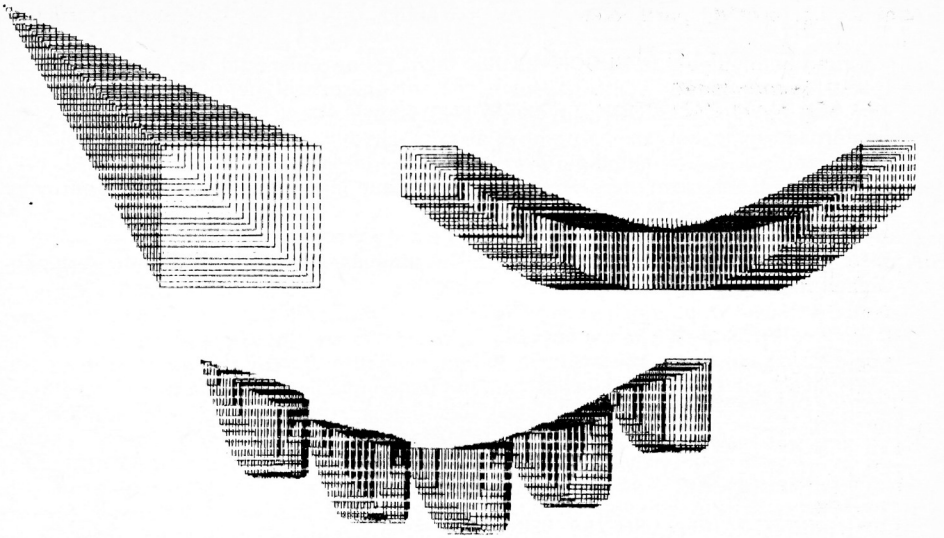
```
1 REM ***********
2 REM * 1.R E C *
3 REM ***********
100 REM * VALTOZO KERETVASTAGSAG *
110 HIRES 1,0
120 DI=9:X=100:Y=50:B=90:H=80:REM BEVITELI PARAMETEREK
130 GOSUB 200  :REM ALPROGRAM HIVASA
140 WAIT 198,255
150 END
160 REM
170 REM *********
180 REM ALPROGRAM
190 REM *********
200 FOR Z=0 TO DI-1
210 REC X+Z,Y+Z,B-2*Z,80-2*Z,1
220 NEXT Z
230 RETURN
```

Próbálgassuk a programot a paraméterek változtatásával.
Végül bemutatunk néhány, a REC utasítást felhasználó grafikát:

P. 69

```
1 REM ***********
2 REM * 2.R E C *
3 REM ***********
100 REM
110 REM **********
120 REM   1.ABRA
130 REM **********
140 HIRES 3,4
150 FOR X=1 TO 99 STEP 2
160 REC X,X,X,X,1
170 NEXT X
180 PAUSE 10
190 HIRES 3,4
200 REM
210 REM **********
220 REM   2.ABRA
230 REM **********
240 REM
250 FOR X=1 TO 290 STEP 3
260 A=X
270 B=60*SIN(X/90)+100
280 C=ABS(B/4)+10
290 D=C
300 REC A,B,C,D,1
310 NEXT X
320 PAUSE 10
330 HIRES 3,4
340 REM
350 REM **********
360 REM   3.ABRA
370 REM **********
380 REM
390 FOR X=1 TO 290 STEP 2
400 A=X
410 B=60*SIN(X/90)+50
420 C=ABS(30*SIN(X/20))
430 D=C*2
440 REC A,B,C,D,1
450 NEXT X
460 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

Ne ijedjünk meg a sok bonyolult számítástól, nem szükséges mindet megérteni. A programok önmagukért beszélnek.



12.3.5 BLOCK – mezõ rajzolása

Formátum: BLOCK x1, y1, x2, y2, rü
Paraméterek: x1 – A mezõ bal felsõ sarkának x-koordinátája
MC: 0–159/FFB: 0–319
y1 – A mezõ bal felsõ sarkának y-koordinátája
MC: 0–199/FFB: 0–199
x2 – A mezõ jobb alsó sarkának x-koordinátája
MC: 0–159/FFB: 0–319
y2 – A mezõ jobb alsó sarkának y-koordinátája
MC: 0–199/FFB: 0–199
rü – A rajzoló üzemmód kódja (0–2)
Funkció: Téglalap alakú, kitöltött mezõ rajzolása
Példa: BLOCK 100,150,160,200,1

Tételezzük fel, hogy egy statisztikai programot szeretnénk készíteni, amelyben oszlopdiagramokat is rajzolni akarunk. Ezt a REC utasítással megoldhatjuk ugyan, ahogy azt az elsõ példaprogramunkban láttuk (addig növelve a keretvastagságot, míg a téglalap két oldala össze nem ér, de képzeljük el, hogy ez mennyi idõt igényel). Ráadásul még egy szubrutint is írnunk kell, ami viszont a tárolóból vesz el helyet. Röviden tehát, szükségünk van egy olyan utasításra, amely néhány paraméter megadásával képes egy kitöltött téglalapot a képernyõre rajzolni. Ezeknek a követelményeknek az ún. BLOCK utasítás felel meg. (A késõbbiekben látni fogjuk, hogy még egy harmadik lehetõségünk is van. Nevezetesen a REC és a PAINT utasítások kombinációja, de ez is meglehetõsen idõigényes és nehézkes.)
A BLOCK utasítás kezelése nagyon egyszerû. Meghatározzuk az adott mezõ nagyságát és elhelyezzük a koordináta-rendszerben, azaz megadjuk bal felsõ és jobb alsó sarkainak x- és y-koordinátáit.

Figyelem! Mivel a REC és a BLOCK utasítás 3. és 4. paraméterének jelentése eltérõ, együttes alkalmazásukkor ez némi nehézséget okozhat. Ne felejtkezzünk el róla!

Mint mindig, az ötödik paraméter az üzemmód kódja, amivel a rajz megjelenési formáját írjuk elő.
Itt kell még bemutatnunk a BLOCK utasítás egyik jellegzetességét. Nevezetesen, ha a rajzolandó mező színét a LOW COL-lal akarjuk meghatározni, nem járunk sikerrel. Ezt az utasítást ugyanis a BLOCK figyelmen kívül hagyja. Csak a HIRES-ben eredetileg meghatározott színekkel rajzol. Amennyiben ragaszkodunk a színváltáshoz, kénytelenek vagyunk mégis a már említett REC szubrutinnál maradni. Ettől eltekintve a BLOCK lehetővé teszi számunkra, hogy gyorsan és elegánsan jeleníthessünk meg a képernyőn különböző színes mezőket.
A következő program néhány alkalmazási lehetőséget mutat be.
Az első részben arra láttunk példát, hogy a REC utasítással egyszer előállított, vastagon keretezett mezőt a BLOCK-kal is megrajzolhatjuk.

P. 70

```
1 REM ************
2 REM * B L O C K *
3 REM ************
100 REM
110 REM ************
120 REM   K E R E T
130 REM ************
140 REM
150 HIRES 7,0 :REM GRAFIKA BEKAPCSOLASA
160 BLOCK 110,60,220,120,1:REM KULSO MEZO RAJZOLASA
170 BLOCK 120,70,210,110,0:REM BELSO MEZO TORLESE
180 PAUSE 10
190 REM
200 REM ***********************
210 REM   I N V E R T A L A S
220 REM ***********************
230 REM
240 HIRES7,0:REM GRAFIKA TORLESE
250 FOR X=1 TO 280 STEP 20
260 BLOCK X,50,X*1.2,100*SIN(X/80)+50,2:REM BLOCK INVERTALASA
270 NEXT X:REM KOVETKEZO BLOCK
280 PAUSE 10
290 REM
300 REM ***********************
310 REM   M U L T I C O L O R
320 REM ***********************
330 REM
340 HIRES7,0
350 COLOUR 2,0:REM (HATTER=FEKETE) KERET=PIROS
360 MULTI 3,4,5:REM MULTICOLOR BEKAPCSOLASA
370 BLOCK 5,20,150,190,1:REM 1.SZIN
380 BLOCK 45,70,150,190,2:REM 2.SZIN
390 BLOCK 85,120,150,190,3:REM 3.SZIN
400 BLOCK 20,50,100,150,4:REM INVERTALAS
410 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

12.3.6 CIRCLE – ellipszis rajzolása

Formátum: CIRCLE mx, my, rx, ry, rü
Paraméterek: mx – Az ellipszis (kör) középpontjának x-koordinátája
MC: 0–159/FFB: 0–319
my – Az ellipszis (kör) középpontjának y-koordinátája
MC: 0–199/FFB: 0–199
rx – Az ellipszis x-irányú sugarának hossza (pontokban)
ry – Az ellipszis y-irányú sugarának hossza (pontokban)
rü – A rajzoló üzemmód kódja

Funkció: Ellipszis (kör) rajzolása
Példa: CIRCLE 150,80,60,30,1

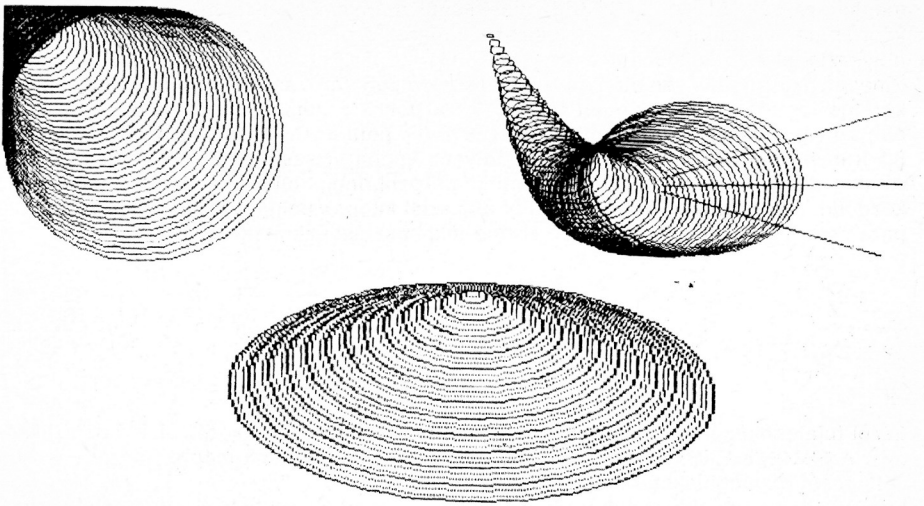
Már eddig is elcsodálkozhattunk azon, hogy a SIMON's BASIC mi mindenre képes. Most is egy olyan utasítással fogunk megismerkedni, ami rengeteg munkától kímélhet meg bennünket, valamint hozzásegít ahhoz, hogy igényes grafikákat készíthessünk.
Ezzel az utasítással a jövőben számtalan kört és ellipszist varázsolhatunk a képernyőre, különösebb fáradság nélkül. Azáltal, hogy néhány paramétert közlünk a számítógéppel, máris meghatároztuk az alakjukat.
Ejtsünk most néhány szót az ebben az összefüggésben szükséges geometriai alapokról. Több módszer is ismert, amellyel egy ellipszist definiálhatunk. SIMON's BASIC-ben a középpontos ábrázolást alkalmazzuk, azaz minden ellipszishez elsősorban megadjuk annak középpontját. Ez felezi az ellipszis két szemközti pontjának távolságát. A középpont és az ellipszis egy pontjának távolsága az ellipszis sugara. Ezek közül kettőnek, a vízszintes és a függőleges sugárnak kitüntetett szerepe van. Ezek határozzák meg ugyanis az ellipszis alakját. Valódi ellipszis esetén ezek különböző hosszúságúak. A kör egy speciális ellipszis, amelynek függőleges és vízszintes sugara azonos hosszúságú és ezáltal az összes pontja azonos távolságra van a középpontjától.
Nos, a CIRCLE utasítás az ellipszis ezen tulajdonságaira épülve működik.
Mindenekelőtt meg kell adnunk az ellipszis középpontjának koordinátáit (mx, my), majd az rx- és ry-sugarakkal meghatározzuk a nagyságát és az alakját. Már csak az üzemmód (rü) előírása van hátra és máris kezdhetjük a rajzolást.
Ha kört akarunk rajzolni, nem mindegy, hogy a számítógép FFB vagy MC üzemmódban van-e. Előző esetben elegendő, ha a két sugarat (rx és ry) azonos nagyságúra választjuk, míg MC-ben szem előtt kell tartani, hogy a felbontóképesség x-irányban a felére csökkent. Ezáltal a kör x-irányú sugara pontokban megadva, az y-irányúnak csak a fele lesz. (rx = 1/2 ry). Köztudott, hogy minden tv-készülék torzít egy kicsit. Ettől vagy eltekintünk, vagy a képernyőn történő közvetlen méréssel megállapítjuk a korrekciós tényezőt. Ebben az esetben nincs szükségünk a SIMON's BASIC kézikönyv vonatkozó adataira.
Ha olyan kört akarunk rajzolni, amely kis mértékben túlnyúlik a képernyőn, akkor az a legegyszerűbb, ha a legutolsó látható pontot egy egyenessel összekötjük a legelső, újból megjelenő ponttal. Nagy kiterjedés esetén kisebb-nagyobb káosz keletkezhet.
A CIRCLE utasítás alkalmazását két példán mutatjuk be. Először egy kis gyakorlóprogram, amely segíti az utasítás megértését. A beadandó paramétereket jól láthatóan megjelöltük a felettük lévő REM-sorokkal. Többszöri változtatásukkal figyeljük meg, mi történik:

P. 71

```
1 REM ******************
2 REM * 1.C I R C L E *
3 REM ******************
100 REM
110 HIRES 7,8: REM GRAFIKA BEKAPCSOLASA
120 REM GYAKORLO PELDA
130 REM MODOSITSUK A PARAMETEREKET !
140 REM
150 REM      MX! MY! RX! RY!ZM
160 REM     -------------------
170 REM        !   !   !   !
180 CIRCLE 160,100, 70, 90,1
190 REM     -------------------
200 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

P. 72

```
1 REM ******************
2 REM * 2.C I R C L E *
3 REM ******************
100 REM
110 REM ************
120 REM    TOLCSER
130 REM ************
140 REM
150 HIRES 3,2:REM GRAFIKA BEKAPCSOLASA
160 COLOUR 2,2:REM KERET ES HATTER PIROS
170 FOR X=1 TO 90 STEP 2
180 CIRCLE X+10,X+10,X,X,1
190 NEXT X
200 PAUSE 10
210 REM
220 REM ****************
230 REM   HAVASI KURT
240 REM ****************
250 REM
260 HIRES 3,2
270 FOR X=1 TO 110 STEP 2
280 CIRCLE 1.3*X+10,50*SIN(X/30-.2)+X+10,X/2+X/8,X/2,1
290 NEXT X
300 LINE 130,105,270,50,1
310 LINE 130,110,280,100,1
320 LINE 130,115,270,150,1
330 PAUSE 10
340 REM
350 REM ***********
360 REM   KAGYLO
370 REM ***********
380 REM
390 HIRES 3,2
400 COLOUR 12,12:REM HATTER ES KERET 2.SZURKE
410 MULTI 6,7,7:REM ATVALTAS MULTICOLOR-RA
420 FOR X=1 TO 78 STEP 2
430 T=T+1:REM SZINREGISZTER VALTAS
440 LOW COL F,F+1,F+2:REM SZINVALTAS
450 CIRCLE 80,0.92*X+10,X,X,T
460 IF T=3 THEN T=0:F=F+3:IF F=16 THEN F=0:REM SZINVALTAS
470 NEXT X
480 PAUSE 10
490 REM
500 REM ****************
510 REM   VILLOGTATAS
520 REM ****************
530 REM
540 FOR X=0 TO 15
```



```
550 COLOUR X,X
560 PAUSE 5
570 NEXT X
580 COLOUR 9,9
590 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

A 2. program lefutása kissé sokáig tart, de legyünk türelemmel. A SIMON's BASIC CIRCLE utasítása nem olyan gyors, mint más grafikus programok (pl. SUPERGRAPHIK), de azért sok örömet szerezhet.

12.3.7 ARC – ellipszisív rajzolása

Formátum: ARC mx, my, ik, iv, t, rx, ry, rü

Paraméterek: mx – A teljes ellipszis középpontjának x-koordinátája
MC: 0–159/FFB: 0–319
my – A teljes ellipszis középpontjának y-koordinátája
MC: 0–199/FFB: 0–199
ik – Az ív kezdőpontjának szöge (0–360°)
iv – Az ív végpontjának szöge (0–360°)
t – Két szomszédos számított pont távolsága szögben kifejezve (0–360°)
rx – Az ellipszis vízszintes sugara
MC: 0–159/FFB: 0–319
ry – Az ellipszis függőleges sugara
MC: 0–199/FFB: 0–199
rü – A rajzoló üzemmód kódja (0–2)

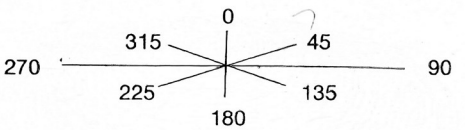
Funkció: Ellipszis- vagy körív rajzolása

Példa: ARC 160,100,20,120,20,50,40,1

A CIRCLE mellett még két olyan utasítással foglalkozunk, amelyek az ellipszissel vannak kapcsolatban. Ezek közül az első az ARC utasítás, amely az ellipszisnek csak egy ívét rajzolja meg. Ahogy létezik a félkör, negyedkör fogalma, ugyanúgy létezik a fél-, negyed-, nyolcad-, ... stb. ellipszis fogalma is. Ugyanakkor ennek az utasításnak még egy jellegzetessége van, de erről majd később lesz szó.

Először is: Mire van szükségünk ahhoz, hogy egy ívet megrajzolhassunk? Mindenekelőtt tudnunk kell azt, hogy melyik az az ellipszis, amelynek egy ívét ki akarjuk rajzolni. Szükségünk van tehát azokra a paraméterekre, amelyek az ellipszist egyértelműen

meghatározzák. Ezeket a CIRCLE utasításnál már ismertettük: mx, my a középpont koordinátái, valamint rx és ry az ellipszis sugarai. A paramétereket a formátum szerint helyezzük el az utasításban.
Emellett szükségünk van még az ív hosszának meghatározására is. Mint tudjuk, minden kör (és természetesen minden ellipszis) 360 fokra osztható. SIMON's BASIC-ben az ellipszis legfelső pontja jelenti a 0 (azaz a 0°) pontot. Ettől kezdve az óramutatóval egyirányba mérjük fel a 360 fokot, amelynek végén visszajutunk a kiindulási pontba. Amikor tehát egy ív hosszát akarjuk meghatározni, meg kell adnunk azt a szöget, ahol kezdődik (ik) és ahol végződik (iv). Fél ellipszist kapunk például az ik=0 és az iv=180 paraméterértékekkel. A következő ábra a fokbeosztást szemlélteti:



Ezzel tulajdonképpen mindent megadtunk, ami az ellipszisív megrajzolásához szükséges. A SIMON's BASIC azonban továbbmegy. Meg kell adnunk még a t paramétert is. Jelentését az alábbiakban magyarázzuk.
Logikus, hogy egy ellipszis megrajzolásához egy csomó pont helyét kell meghatározni. Normál esetben az SIMON's BASIC ezt úgy csinálja, hogy az egymástól megfelelően kis távolságra lévő pontok helyét kiszámítja, majd a szomszédos pontokat egy egyenessel összeköti. Végeredményként egy összefüggő képet kapunk. CIRCLE esetében ezt a távolságot eleve meghatározták és akkorára választották, hogy mind a felbontás, mind pedig a rajzolási sebesség elfogadható legyen. Minél nagyobb ugyanis a pontok közti távolság, annál szögletesebb az ellipszis és minél több pont pozícióját kell kiszámítani, annál lassúbb a rajzolás. Valójában egy sokszöget rajzolunk, amely megfelelően kis ponttávolság esetén ellipszisnek látszik.
Az ARC utasítás megengedi, hogy ezt a távolságot, vagyis a felbontás szögét tetszés szerint megválaszthassuk, amiből az következik, hogy vele sokszöget is rajzolhatunk. Egy normál, CIRCLE utasítással rajzolt ellipszisnél a t paraméter (azaz a szomszédos pontok szögben kifejezett távolsága) 12.
Az ellipszis kiszámított pontjai tehát 0, 12, 24... stb. foknál vannak. A két alábbi utasítás emiatt azonos eredményt ad:

- ARC 160,100,0,360,12,50,70,1
- CIRCLE 160,100, 50,70,1

Ennek megfelelően különböző sokszögeket rajzolhatunk, amelyben a következő táblázat segítségünkre lehet:

a	sokszög
120	háromszög
90	négyszög
72	ötszög
60	hatszög
51	hétszög
45	nyolcszög
40	kilencszög
36	tízszög

A táblázatból kiválasztható a rajzolni kívánt sokszöghöz tartozó t paraméter. Ha teljes egészében meg akarjuk rajzolni, akkor ik=0, iv=360 legyen.
Nos, most már mindent tudunk ahhoz, hogy a gyakorlóprogramot kipróbáljuk:

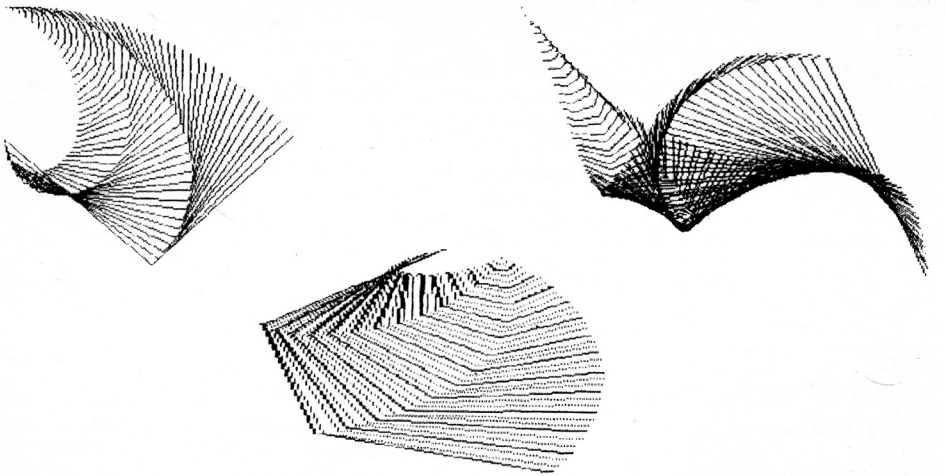
P. 73

```
1 REM ***********
2 REM * 1.A R C *
3 REM ***********
100 REM
110 HIRES7,8:REM GRAFIKA BEKAPCSOLASA
120 REM GYAKORLO PELDA
130 REM VALTOZTASSUK A PARAMETEREKET !
140 REM
150 REM MX! MY! SW! EW!  A! RX! RY!ZM
160 REM ------------------------------
170 REM   !   !   !   !   !   !   !
180 ARC160,100,  0,360, 45, 70, 90, 1
190 REM ------------------------------
200 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

P. 74

```
1 REM ***********
2 REM * 2.A R C *
3 REM ***********
100 REM
110 REM ***********
120 REM   1. ABRA
130 REM ***********
140 REM
150 HIRES 3,9: REM GRAFIKA BEKAPCSOLASA
160 COLOUR 9,9:REM KERET=BARNA
170 FOR X=1 TO 90 STEP 2
180 ARC X+10,X+10,X,2*X+90,X,X,X,1
190 NEXT X
200 PAUSE 10
210 REM
220 REM ***********
230 REM   2. ABRA
240 REM ***********
250 REM
260 HIRES 3,9
270 FOR X=1 TO 160 STEP 2
280 ARC 1.3*X+10,50*SIN(X/30-.2)+X+10,X,190+X,30+X,X/2+X/8,X/2,1
290 NEXT X
300 REM
310 REM ***********
320 REM   3. ABRA
330 REM ***********
340 REM
350 HIRES 3,9
360 COLOUR 12,12:REM HATTER ES A KERET 2-ES SZURKE
370 MULTI 6,7,7:REM MC BEKAPCSOLASA
380 FOR X=1 TO 78 STEP 2
390 T=T+1:REM SZINREGISZTER VALTAS
400 LOW COL F,F+1,F+2:REM SZINVALTAS
410 ARC 80,.92*X+10,90+X,270+X,60,X,X,T
420 IF T=3 THENT=0:F=F+3:IF F=16 THEN F=0:REM SZINKOD VALTOZTATAS
430 NEXT X
440 PAUSE5
450 REM
460 REM ***************
470 REM   VILLOGTATAS
480 REM ***************
490 REM
500 FOR X=0 TO 15
510 COLOUR X,X:REM HATTER- ES KERETSZIN VALTAS
520 PAUSE 4
530 NEXT X
540 COLOUR 9,9
550 WAIT 198,255: REM VARAKOZAS EGY GOMB LENYOMASARA
```

Csodálatos, mi mindenre képes ez a kis utasítás!



12.3.8 ANGL – az ellipszis sugarának megrajzolása

Formátum: ANGL mx, my, sz, rx, ry, rü

Paraméterek: mx – Az ellipszis középpontjának x-koordinátája
MC: 0–159 / FFB: 0–319
my – Az ellipszis középpontjának y-koordinátája
MC: 0–199 / FFB: 0–199
sz – A rajzolandó és az y-irányú sugár által bezárt szög (0–360°)
rx – Az ellipszis x-irányú sugara
MC: 0–159 / FFB: 0–319
ry – Az ellipszis y-irányú sugara
MC: 0–199 / FFB: 0–199
rü – Üzemmód kódja (0–2)

Funkció: Az ellipszis adott szöget bezáró sugarának megrajzolása

Példa: ANGL 160,100,20,60,70,1

Az ANGL a már említett második utasítás az ellipszis témakörében. Nézzük, mire is használhatjuk.

Segítségével megrajzolhatjuk egy tetszőleges ellipszis bármely sugarát anélkül, hogy magát az ellipszist megrajzolnánk.

Ugyanúgy, mint az ARC utasításnál, itt is legelőször azokat a paramétereket kell megadnunk, amelyek az ellipszist egyértelműen meghatározzák. Ezek, a már ismert módon, az ellipszis középpontjának sebességái (mx, my) és az x–y irányú sugarak (rx, ry). De ezt már nem kell részleteznünk.

A továbbiakban meg kell még valahogyan határoznunk, hogy melyik sugarat akarjuk kirajzoltatni. Ezt szintén az ARC utasításban leírtak szerint, a 0 pont és a sugár végpontjának szögértékben kifejezett távolságával adhatjuk meg (sz). Egyszerűen beadjuk tehát a végpont pozícióját, amit azután a számítógép összeköt az ellipszis középpontjával és máris kirajzolódott a keresett sugár. Ez minden, amit tennünk kell.

Nincs még egy ilyen csodálatos hatásokat eredményező utasítás. Rajzolhatjuk, pl. az ellipszist és a neki megfelelő valamilyen sokszöget váltakozva... stb. Az utasítás titkát csak próbálgatás útján fejthetjük meg.

Mint már tapasztaltuk, a gyakorlóprogramon tanulmányozhatjuk az egyes paraméterek hatását. Az ANGL utasítás azonban elsősorban bonyolult alakzatok rajzolására alkalmazható, mint ahogy azt a 2. példaprogramunkon keresztül be is bizonyítjuk. Azt is észre kell

vennünk, hogy egyszerűen nem kerülhetjük el a FOR...NEXT ciklusok alkalmazását. Kísérletezzünk ezen a területen is!

P. 75

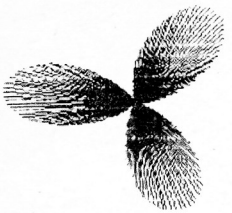
```
1 REM **************
2 REM * 1.A N G L *
3 REM **************
100 REM
110 HIRES 7,8:REM GRAFIKA BEKAPCSOLASA
120 REM GYAKORLO PELDA
130 REM VALTOZTASSUK A PARAMETEREKET !
140 REM
150 REM    MX!  MY!   W!  RX!  RY!  ZM
160 REM   ----------------------------
170 REM      !    !    !    !    !
180 ANGL 160,100, 45, 70, 90, 1
190 REM   ----------------------------
200 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

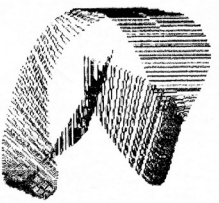
P. 76

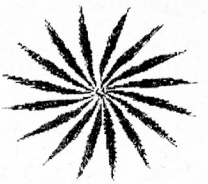
```
1 REM **************
2 REM * 2.A N G L *
3 REM **************
100 REM
110 REM ************
120 REM    1. ABRA
130 REM ************
140 REM
150 HIRES 7,8:REM GRAFIKA BEKAPCSOLASA
160 FOR X=1 TO 140 STEP 2
170 ANGL X,X,X,X,X,1
180 NEXT X
190 REM ***********
200 REM    LOHERE
210 REM ***********
220 REM
230 HIRES 7,8: REM GRAFIKA TORLESE
240 FOR X=1 TO 360 STEP 2
250 A=40*SIN(X/19-.5)+40
260 ANGL 160,100,X,A,A,1
270 NEXT X
280 PAUSE 10
290 REM
300 REM ************
310 REM    2. ABRA
320 REM ************
330 REM
340 HIRES 7,8:REM  GRAFIKA TORLESE
350 FOR X=1 TO 400 STEP 2
360 MX=50*SIN(X/60)+100
370 MY=60*SIN(X/40)+70
380 W=X/2
390 RX=20*SIN(X/20)+30
400 RY=30*SIN(X/30)+30
410 ANGL MX,MY,W,RX,RY,1
420 NEXT X
430 REM
440 REM **************
450 REM    MARGARETA
460 REM **************
470 REM
480 HIRES 7,8:REM GRAFIKA TORLESE
490 FOR Y=0 TO 31
500 T=ABS(T-1)
510 FOR X=1 TO 20
520 ANGL 160,100,X+Y*11,X*3,X*3,T
```

```
530 NEXT X
540 NEXT Y
550 WAIT 198,255
```

Ugye milyen egyszerű ezeknek a szép rajzoknak az elkészítése! Talán nincs is még egy olyan utasítás (kivéve talán a következőt), ami ennyi mindent tud. Győződjünk meg róla!







12.3.9 PAINT – az alakzat "kifestése"

Formátum: PAINT x,y,rü
Paraméterek: x – a hivatkozási pont x-koordinátája
MC: 0–159 / FFB: 0–319
y – a hivatkozási pont y-koordinátája
MC: 0–199 / FFB: 0–199
rü – üzemmód kódja (0–2)
Funkció: Egy tetszőleges alakú, zárt felület "kifestése"
Példa: PAINT 100,120,1

Ebben a fejezetben utolsóként a PAINT utasítással ismerkedünk meg. A SIMON's BASIC kifejlesztője ezzel az utasítással egy nagyon remek dolgot alkotott!
Tételezzük fel, hogy a CIRCLE utasítással megrajzolt ellipszist (vagy egyéb más zárt alakzatot) szeretnénk "kiszínezni". A következőképpen kell eljárnunk: először is meg kell rajzolnunk a képernyőn a szóban forgó alakzat körvonalát. Vigyázzunk arra, hogy ennek teljesen zártnak kell lennie, sehol sem szakítható meg. Ezután határozzunk meg az alakzat belsejében egy pontot (hivatkozási pont), és ennek x–y koordinátáit írjuk be az utasításba. Ha az üzemmódot is előírtuk, a többi már megy magától. A PAINT felismeri a körvonal pontjait és az alakzatot teljesen kitölti ("kiszínezi").
A "színezés" ellentéte a törlés, ami hasonló módon történik. Az utasítás a "nem színezett" pontokat tekinti keretnek és minden pontot töröl, ami ezen belülre esik (a keret pontjait is beleértve).
Természetesen a hivatkozási pontot a körvonalon kívül is meghatározhatjuk. Ebben az esetben az utasítás az egész képernyőt "kiszínezi", kivéve magát az alakzatot.
Az utasítás néha megmagyarázhatatlan dolgokat is "művel", főleg invertálás üzemmódban (rü=2). Ki lehet próbálni, de alkalmazásának nincs értelme.
Időnként előfordulhat különösen hegyes sarkok esetén, hogy az utasítás nem tölti ki az egész alakzatot. Ez azonban nem az utasítás hibája, hanem inkább az az eljárás eredményezi, ahogyan a vonalak, ellipszisek... stb. a képernyőn létrejönnek.
A BLOCK utasítástól eltérően PAINT-nál a LOW COL hatásos marad. Megtehetjük tehát, hogy a BLOCK utasítást szükség esetén a REC-PAINT utasításkombinációval helyettesítjük (lásd: REC, BLOCK).
A leírtakat a következő két példán szemléltetjük:

P. 77

```
1 REM ***************
2 REM * 1.P A I N T *
3 REM ***************
100 REM
110 REM *************
120 REM   ELLIPSZIS
130 REM *************
140 REM
150 HIRES 8,9:REM GRAFIKA BEKAPCSOLASA
160 CIRCLE 160,100,50,80,1:REM ELLIPSZIS RAJZOLASA
170 PAUSE 1
180 PAINT 160,100,1:REM SZINEZES
190 PAUSE 5
200 REC10,10,60,70,1:REM TEGLALAP RAJZOLASA
210 PAUSE 1
220 PAINT 11,11,1
230 PAUSE 10
240 REM
250 REM **********
260 REM   2.ABRA
270 REM **********
280 REM
290 HIRES 8,9:REM GRAFIKA TORLESE
300 REM ***********
310 REM   SOKSZOG
320 REM ***********
330 REM
340 LINE100,100,50,70,1
350 LINE50,70,150,120,1
360 LINE150,120,120,20,1
370 LINE120,20,300,199,1
380 LINE300,199,160,100,1
390 LINE160,100,160,170,1
400 LINE160,170,60,130,1
410 LINE60,130,100,100,1
420 PAUSE 1
430 PAINT 101,101,1:REM MEZO SZINEZESE
440 PAUSE 10
450 PAINT 101,101,0:REM MEZO TORLESE
460 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

P. 78

```
1 REM ***********
2 REM * 2.PAINT *
3 REM ***********
100 REM
110 HIRES 3,2 :REM GRAFIKA BEKAPCSOLASA
120 CIRCLE 160,100,30,40,1 :REM BELSO
130 CIRCLE 160,100,50,60,1 :REM KOZEPSO
140 CIRCLE 160,100,70,80,1 :REM KULSO
150 PAINT  160+29,100+39,1 :REM KOZEPSO SAV KITOLTESE
160 PAINT       0,     0,1 :REM KULSO MEZO KITOLTESE
170 PAUSE 10
180 REM
190 REM *********
200 REM   VIRAG
210 REM *********
220 REM
230 HIRES 3,2
240 CIRCLE 160,100,10,10,1
250 CIRCLE 160,100,30,40,1
260 CIRCLE 160,100,50,20,1
270 PAINT    160,    100,1
```

```
280 PAINT     160,100+38 ,1
290 PAINT     160,100-38 ,1
300 PAINT     160+48, 100,1
310 PAINT     160-48, 100,1
320 WAIT 198,255
```

12.3.10 Ellenőrző kérdések

Végre elérkeztünk a fejezet végére, ellenőrizhetjük, hogy mennyit sikerült elsajátítanunk a tanultakból.

1. Melyik a helyes formátum?

   a) PLOT 1,0,0: (Rajzol egy pontot a (0;0)-pozícióba.)
   b) TEST (100,100): (Megvizsgálja, hogy a (100;100)-koordinátájú pont be van-e kapcsolva.)
   c) IF TEST (100,100) = 0 THEN 90: (Ha a (100;100)-koordinátájú pont nincs bekapcsolva, a program a 90. sorban folytatódik.)
   d) REC 10,20,100,150: (Rajzol egy olyan téglalapot, amelynek két szemközti csúcsa a (10;20) és a (100;150) pontokban van.)
   e) BLOCK 10,20,100,150: (Rajzol egy olyan kitöltött téglalapot, amelynek két szemközti csúcsa a (10;20) és a (100;150) pontokban van.)
   f) ARC 160,100,0,360,12,50,70,1: (Rajzol egy teljes ellipszist.)

2. Szeretnénk a CIRCLE 160,100,40,30,1 utasítással megrajzolt ellipszist "kiszínezni". Melyik utasítást használjuk?

   a) BLOCK 160,100,40,30,1
   b) PAINT 160,100,1
   c) PAINT 0,0,1
   d) BLOCK 160,100,1

3. Egy 10 pont vastagságú keretet akarunk rajzolni. Melyik utasításkombinációt kell választanunk?

   a) FOR X=0 TO 9:REC 100+X, 100+X, 50−2*X, 60−2*X,1: NEXT X
   b) BLOCK 100,100,150,160,1:BLOCK 110,110,140,160,0
   c) REC 100,100,50,601 :REC 110,110,30,40,1:PAINT 105,105,1
   d) REC 100,100,50,60,1

12.4 Egyéb ábrák rajzolása

12.4.1 DRAW – tetszőleges alakzat rajzolása

Formátum: DRAW"füzér",x,y,rü
Paraméterek: füzér – Egy karakterlánc vagy egy füzérváltozó, amely az ábrát definiálja
x – A kezdőpont x-koordinátája
MC: 0–159 / FFB: 0–319
y – A kezdőpont y-koordinátája
MC: 0–199 / FFB: 0–199
rü – Az üzemmód kódja (0–2)
Funkció: Rajzol egy tetszőleges ábrát
Példa: DRAW"5678", 160,100,1
DRAW A$, 160,100,1

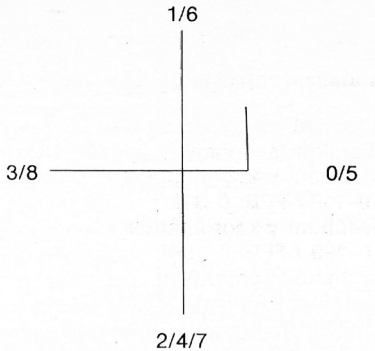
Eddig csak programozott, számítási algoritmussal előre meghatározott alakzatokat rajzoltunk, amelyeket legfeljebb nagyítani vagy torzítani (CIRCLE) tudtunk. Ezekhez a műveletekhez azonban először mindig meg kellett adnunk a kiindulási alakzatot is. Tetszőleges, összetett ábrákat esetleg a PLOT-tal és a LINE-nal rajzolhattunk, de ez nagyon sok időt és tárolókapacitást igénylő eljárás. Ezen túl az elkészült rajz elcsúsztatása vagy esetleges elforgatása BASIC-ben szinte megoldhatatlan. A SIMON's BASIC DRAW utasítása nélkül felesleges is próbálkoznunk vele.
Ezzel az utasítással egyszerűen és minden egyéb módszernél gyorsabban hozhatunk létre tetszőleges alakzatokat, amelyeket azután a képernyőn bárhol elhelyezhetünk, sőt nagyíthatunk és el is forgathatunk.
Az ábrát a "füzér"-paraméter tartalmával definiálhatjuk. Azaz a következőkben ismertetésre kerülő módszerrel létrehozunk egy füzért (ami lehet egyszerűen egy karakterlánc, de lehet füzérváltozó is), amit a DRAW utasításszó után írunk, természetesen idézőjelek közé!

Hogyan is állíthatunk össze egy ilyen ábrát?

Úgy kell elképzelni, mintha a számítógép ceruzával rajzolna a képernyőre. Nekünk csak azt kell megmondanunk, hogy a "ceruza", ill. valójában a kurzor, a négy lehetséges irány (fel, le, jobbra, balra) közül merre mozogjon. Az elmozdulás egységnyi hosszát a későbbiekben tárgyalt ROT utasítással adjuk meg.
Ugyanakkor azt is előírhatjuk, hogy a "ceruza" az adott irányban rajzoljon-e, azaz húzzon-e egy vonalat, vagy csak a "papír" (ill. képernyő) felett rajzolás nélkül mozduljon-e el. Az így létrehozott kétszer négy funkció az alábbi kódokkal adható át a számítógépnek:

kód	mozgásirány	rajzolás
0	jobbra	nincs
1	fel	nincs
2	le	nincs
3	balra	nincs
4	le	nincs
5	jobbra	van
6	fel	van
7	le	van
8	balra	van
9	a rajzolás megszakad	

Tehát:



A 9-es kódhoz nem tartozik rajzolási funkció, hanem a rajzolási művelet megszakítását eredményezi, még akkor is, ha további kódok követik. A kézikönyvben leírtaktól eltérően azonban nem szükséges minden füzért 9-essel zárni.
Az alakzatot tehát az egymás után következő irányutasításokból álló füzérrel definiálhatjuk. A füzér maximális hossza 256 karakter lehet. Mivel egy BASIC sor 80 karakterből áll, az ennél hosszabb füzéreket a következő trükkel, több sorból kell összefűznünk:

```
10 A$="12345678901234567890123456789012345678901
23456789012345678901234567890123456789012345678901234567890"
20 A$=A$ + "12345678901234567890"
```

A füzér mellett még további három paramétert kell megadnunk ahhoz, hogy a szintaktika helyes legyen. Ezek közül az utolsóval (rü) itt már nem foglalkozunk, mindent megtudhattunk róla a 12.2. fejezetből. Az x- és y-paraméterekkel a képernyőn kijelölhetjük a "ceruza" (azaz a grafikus kurzor) indulási helyét. A számítógép ehhez a ponthoz viszonyítva dolgozza fel a definiáló-füzért.
A DRAW utasítás egy szép alkalmazási lehetősége a sprite-ok ilyen alakzatokkal (ún. shape-ok) való kiegészítése, amelyek törléssel és más helyen történő gyors újrarajzolással mozgathatók a képernyőn (lásd a P. 80-as példaprogramot). Ezzel a technikával mindenképpen érdemes megismerkedni!
Az utasítás előnye még többek között, hogy a ROT utasítással (12.4.2 fejezet) nagyítható és elforgatható.
A példaprogramok:

P. 79

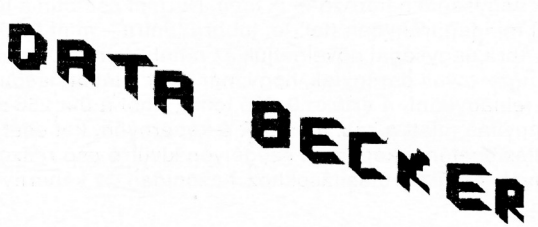
```
1 REM **************
2 REM * 1.D R A W *
3 REM **************
100 REM
110 HIRES 6,7:REM GRAFIKA BEKAPCSOLASA.
120 ROT 0,40:REM ELFORGATAS NINCS, NAGY-SAG=40
130 DRAW "65789",160,100,1:REM NEGYZET  RAJZOLASA
140 REM RAJZOLAS IRANYA - FEL,JOBBRA,LE,BALRA
150 WAIT 198,255:REM VARAKOZAS EGY BILLENTYU LENYOMASARA
```

Fenti példánkat gyakorlóprogramnak szántuk. Változtatgassuk a paramétereket és figyeljük meg az eredményt. A definiáló-füzér végéről a 9-est elhagyhatjuk, mint azt már az előzőekben említettük.

P. 80

```
1 REM ************
2 REM * 2.D R A W *
3 REM ************
100 REM
110 REM ********
120 REM   BETU
130 REM ********
140 REM
150 DATA"666666555757777787888":REM D
160 DATA"66666655557777888800007777":REM A
170 DATA"000666666688000555":REM T
180 DATA"66666655557777888800007777":REM A
190 DATA"":REM SPACE
200 DATA"666666555577778888000077778889":REM B
210 DATA"555553333666555533336665555":REM E
220 DATA"55553333666666555":REM C
230 DATA"666575757333111565656333777":REM K
240 DATA"555553333666555533336665555":REM E
250 DATA"66657575733311166655577788":REM R
260 REM
270 REM ************
280 REM   RAJZOLAS
290 REM ************
300 REM
310 HIRES 2,3
320 ROT 0,4:REM ELFORGATAS NINCS, NAGYSAG=4
330 FOR X=1 TO 11
340 READ A$:REM DATA SOROK KIOLVASASA
350 DRAW A$,X*30-28,50+10*X,1:REM BETUK
360 NEXT X
370 PAUSE 5
380 REM **********************
390 REM   BETUK SZELESITESE
400 REM **********************
410 REM
420 RESTORE :REM ADATMUTATO VISSZAALLITASA
430 FOR X=1 TO 11
440 READ A$:REM DATA SOROK KIOLVASASA
450 FOR Z=0 TO 7:REM SZELESSEG-CIKLUS
460 DRAW A$,X*30-28+Z,50+10*X+Z,1:REM BETUK RAJZOLASA
470 NEXT Z
480 NEXT X
490 WAIT 198,255:REM VARAKOZAS EGY BILLENTYU LENYOMASARA
```

Ez a program a DRAW utasítás felhasználásával egy feliratot készít, amelynek betűit a DATA sorokban lévő füzérekkel definiáltuk. A DATA sorok elemeit a READ utasítással egymás után beolvassuk az A$ stringváltozóba. Minden DATA sor egy betűt definiál, amit a REM kijelzésben feltüntettünk.
Végül a betűk megjelennek a képernyőn és a következő lépésben kialakul a vastagságuk. A vastagságot a sorokban előírt számú, eltoltan megjelenített betűkkel alakítottuk ki. Próbáljuk ki, mi történik, ha itt a STEP értékét megváltoztatjuk!



12.4.2 ROT – elforgatás, nagyítás

Formátum: ROT f,n
Paraméterek: f – az elforgatás szögének kódja (0–7)
n – a nagyítás mértéke (0–255)
Funkció: A DRAW-val megrajzolt alakzat elforgatása és nagyítása
Példa: ROT 0,20

A DRAW egy csomó lehetőséget rejt magában, amellyel megkönnyíti grafikáink készítését. Az egészre azonban a ROT teszi fel a koronát.
A ROT utasítással a kialakított alakzatot a kezdőpontja – mint forgástengely – körül elforgathatjuk, azaz megdönthetjük, vagy akár a feje tetejére is állíthatjuk, ha úgy tetszik. Nyolc meghatározott forgásszöget tudunk az alábbi kódokkal megadni:

f	forgásszög
0	0 fok
1	45 fok
2	90 fok
3	135 fok
4	180 fok
5	225 fok
6	270 fok
7	315 fok

Ha tehát a 0. kódot választjuk, a DRAW-val definiált rajz eredeti helyzetében marad. Ha a kód 1, az alakzat az óramutatóval egyirányban, 45 fokkal elfordul. (Itt nem a matematikai szögméréssel dolgozunk). 7-nél nagyobb érték megadása esetén a BAD MODE hibaüzenetet kapjuk, amellyel már egyéb utasításoknál megismerkedtünk.
A ROT utasítás néhány különlegességére fel kell hívni a figyelmet:
Ha a rajzot a kiindulási pont körül elforgatjuk, előfordulhat, hogy megváltozik más alakzatokhoz vagy a képernyő egészéhez viszonyított mérete.
Páratlan f értékek esetén (1, 3, 5, 7) az ábra mindig nagyítva jelenik meg. Ennek az az oka, hogy az elforgatás során a pontok nem kör-, hanem négyzetpályán mozognak.
A következő néhány soros program ezt szemlélteti:

P. 81

```
1 REM **********
2 REM * 1.R O T *
3 REM **********
100 REM
110 HIRES 2,3:REM GRAFIKA BEKAPCSOLASA
120 FOR R=0 TO 7
130 PAUSE 2
140 ROT R,20:REM FORGATAS
150 DRAW"E",160,100,1:REM VONAL RAJZOLASA
160 NEXT R
170 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

Az f mellett, mint látjuk, még egy paramétert meg kell adnunk: az n-et. Ez az a tényező, amellyel a rajzunk nagyságát határozhatjuk meg. Normál esetben a lépésnagyság (vagy hosszúságegység) minden irányban (fel, le, jobbra, balra – mint DRAW-nál) 1 pont. Ezt, ill. ezzel együtt az ábra nagyságát növelhetjük az n-nel. Ha n = 1, az ábra nagyítás nélkül jelenik meg. Ezek néha olyan arányiak, hogy már csak a láthatóságuk miatt is kénytelenek vagyunk őket felnagyítani. n értéke 0–255 lehet, ahol a 0 a 256-szoros nagyításnak felel meg. Ha a nagyítás miatt a rajz túlnyúlik a képernyőn, két eset lehetséges.
Kismértékű túlnyúlás esetén a képernyő szegélyén kívülre eső részek eltűnnek, egyébként pedig, az egyéb grafikus utasításokhoz hasonlóan, a képernyő egy más részén jelennek meg.

Magától értetődően itt is ügyelnünk kell a megváltozó méretarányokra és a többszínű (MC) üzemmódban előforduló torzításokra.
A továbbiakban példákon tanulmányozhatjuk a ROT és a DRAW utasítások alkalmazását:

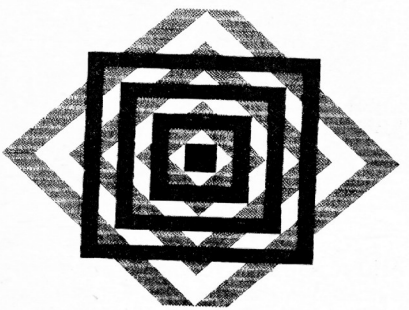
P. 82

```
1 REM ***********
2 REM * 2.R O T *
3 REM ***********
100 REM
110 REM **************
120 REM   ORAMUTATO
130 REM **************
140 REM
150 HIRES 7,6:REM GRAFIKA BEKAPCSOLASA
160 FOR T=1 TO 20:REM KORCIKLUS
170 FOR X=0 TO 7 :REM FORGATAS
180 ROT L,40:REM UTOLSO FORGATASI SZOG BEALLITASA
190 DRAW "6",160,100,0:REM UTOLSO VONAL TORLESE
200 ROT X,40:REM UJ FORGATASI SZOG
210 DRAW "6",160,100,1:REM VONAL RAJZOLASA
220 L=X:REM UTOLSO FORGATASI SZOG TAROLASA
230 NEXT X
240 NEXT T
250 PAUSE 3
260 REM
270 REM **********
280 REM   2. KEP
290 REM **********
300 REM
310 HIRES 7,6:REM GRAFIKA TORLESE
320 FOR X=1 TO 69
330 ROT X/10,X:REM FORGATAS ES NAGYITAS
340 DRAW "365577886",160,100,1
350 NEXT X
360 PAUSE 10
370 REM
380 REM ***********
390 REM   CSILLAG
400 REM ***********
410 REM
420 HIRES 7,6:REM GRAFIKA TORLESE
430 FOR X=1 TO 69
440 ROT X/10,X:REM FORGATAS ES NAGYITAS
450 DRAW "5885766",160,100,1
460 NEXT X
470 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

Az első részben a körmozgás előállítását figyelhetjük meg. Ahhoz, hogy a mozgás lehetőleg folyamatos legyen, először mindig meg kell jeleníteni a figurát a következő pozícióban és csak azután lehet az előző helyről törölni. Néha, elsősorban nagy ábrák esetén ez a módszer a vonalak zűrzavarát okozhatja, ezért ilyenkor ajánlatos fordított sorrendben eljárni. Új koordináták megadásával az elforgatáson kívül különböző irányokba történő elmozdulást is létrehozhatunk. Következő programunkkal ezt mutatjuk be:

P. 83

```
1 REM ***********
2 REM * 3.R O T *
3 REM ***********
100 REM
110 HIRES 2,3:REM GRAFIKA BEKAPCSOLASA
120 ROT 0,20:REM FORGATAS NINCS
130 FOR X=1 TO 319 STEP 4
140 DRAW "6578",A,100,0:REM TORLES A REGI POZICIOROL
150 DRAW "6578",X,100,1:REM RAJZOLAS AZ UJ POZICIOBA
160 A=X:REM A REGI POZICIO TAROLASA
170 NEXT X
180 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```



12.4.3 Ellenőrző kérdések

Ismét egy fejezet végére értünk. Ellenőrizzük, mit sikerült elsajátítanunk belőle.

1. Melyik utasítást kell választanunk, ha egy négyzetet akarunk rajzolni?

   a) REC 160,100,70,70,1
   b) DRAW "0123", 160, 100,1
   c) DRAW "5687",160,100,1
   d) ROT 4,5

2. Ábránkat az óramutatóval ellenkező irányban, 90 fokkal el akarjuk forgatni, valamint ötszörösére akarjuk nagyítani. Melyik utasítást kell választani?

   a) ROT 90,5
   b) ROT 2,5
   c) ROT 6,5
   d) ROT 5,2

3. Melyik kód definíciója helyes?

   a) 0: jobbra, rajzolás nélkül
   b) 8: jobbra, rajzolással
   c) 4: balra, rajzolás nélkül
   d) 6: felfelé, rajzolással
   e) 9: lefelé, rajzolással

12.5 Szöveg a grafikában

12.5.1 CHAR – karakterek rajzolása

Formátum: CHAR x,y,c,rü,n
Paraméterek: x – A karakter bal felső sarkának x-koordinátája
MC: 0–159 / FFB: 0–319
y – A karakter bal felső sarkának y-koordinátája
MC: 0–199 / FFB: 0–199
c – A karakter képernyőkódja (0–255)
rü – Az üzemmód kódja (0–2)
n – A karakter függőleges nagysága (0–255)
Funkció: Karakterek rajzolása a képernyő adott helyén meghatározott nagyságban
Példa: 160,100,3,1,2
Ezentúl nem kell a grafikáinknak névtelenül megjelenniük. Feliratozhatjuk, vagy magyarázattal láthatjuk el azokat.

Ez az utasítás lehetővé teszi számunkra, hogy betűket vagy grafikus jeleket jelenitsünk meg a képernyőn, tetszés szerint normál vagy inverz alakban. A karaktert a c paraméterrel definiáljuk. A 9. fejezetben az FCHR vagy FILL utasításoknál már említettük, hogy a betűket és más karaktereket a C64-es nem az ASCII kóddal tárolja a képernyőtárolóban. Az inverz karakterek ugyanis csak így kódolhatók. Pl. a CBM 64 kézikönyvben megtalálható a képernyőkód táblázat. Az inverz karakterek kódját úgy kapjuk meg, hogy a normál karakterkódhoz hozzáadunk 128-at. A táblázat segítségével értékeket adhatunk a c paraméternek (c=5 esetében, pl. a normál E betű, 5+128 = 133 esetében az inverz E betű jelenik meg).
A jelkészletek közti válogatásra nincs módunk, csak a nagybetű/grafikus jelkészletet használhatjuk.
A továbbiakban az x és y paraméterekkel meg kell adnunk a karakter bal felső sarkának pozícióját.
A paraméterek sorát az "n" zárja. Funkciója, ugyanúgy, mint a ROT utasításban, a karakter nagyságát határozza meg, de csak függőleges irányban. A karakterek vastagságát nem befolyásolja. Az n értéke 0–255 között lehet. Az n=0 és az n=1 azonos hatású. Természetesen nincs értelme 30-nál nagyobbra választani, hiszen akkor már a karakter olvashatatlanná válik, sőt túl is nyúlhat a képernyőn.
Ezzel az utasítással tehát egyszerűen és gyorsan feliratozhatjuk az elkészített grafikát, sőt a grafikus jeleket a kép elemeiként is felhasználhatjuk.
Korábbi gyakorlatunkhoz híven először egy gyakorló példát mutatunk be. A paraméterek változtatásával (amelyeket a REM sorokkal megjelöltünk) tanulmányozhatjuk az utasítás működését.

P. 84

```
1 REM **********
2 REM * 1.C H A R *
3 REM **********
100 REM
110 HIRES 1,0:REM GRAFIKA BEKAPCSOLASA
120 REM GYAKORLO PELDA
130 REM VALTOZTASSUK A PARAMETEREKET !
140 REM
150 REM   X!   Y!   C!ZM! G
160 REM ----------------------
170 REM    !    !    !  !
180 CHAR 160,100,   2, 1,3
190 REM ----------------------
200 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

P. 85

```
1 REM ************
2 REM * 2.C H A R *
3 REM ************
100 REM
110 REM ***************
120 REM    JELKESZLET
130 REM ***************
140 REM
150 HIRES 5,6
160 FOR X=0 TO 255
170 T = X/40
180 CHAR FRAC(T)*40*8,X/2,X,1,2
190 NEXT X
200 PAUSE 20
210 REM
220 REM ************
230 REM    NAGYSAG
240 REM ************
250 REM
260 HIRES 5,6
270 FOR X=1 TO 26
280 CHAR X*8,X,X,1,X
290 NEXT X
300 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

A fenti program első részében a teljes karakterkészlet előállítását, a második részében pedig a nagyítás folyamatát figyelhetjük meg. Jó szórakozást!

12.5.2 TEXT – feliratozás

Formátum: TEXT x,y,str,rü,n,t

Paraméterek: x,y – A szöveg első karaktere bal felső sarkának koordinátái (ld. az előző pontot)
str – A kiírandó szöveg, idézőjelek között. Füzérváltozóként (string) is megadható
rü – Az üzemmód kódja (0–2)
n – A karakterek függőleges irányú nagysága (0–255)
t – Két karakter azonos pontjainak távolsága

Funkció: Felirat megjelenítése

Példa: TEXT 50,100,"TEST",1,3,15

Mint láttuk, a CHAR utasítással a grafikába egyszerre csak egy karaktert írhatunk. Ez azt jelenti, hogy hosszabb szöveg kiírása ezzel az utasítással csak körülményesen oldható meg. Ilyenkor használjuk inkább a TEXT utasítást. Ezzel egyszerre egész karaktersorozatokat írhatunk be a grafikába.

Ez az utasítás működésében kissé eltér a CHAR utasítástól. Először is meg kell határoznunk a felirat pozícióját, ami legelső betűje bal felső sarkának x- és y-koordinátái megadásával történik. Ezután kell megadnunk a felirat szövegét; ez lehet füzérváltozó, amelyhez a megfelelő szöveget előzőleg hozzárendeltük, de lehet konkrét karaktersorozat is. Ez az utasítás lehetővé teszi a különböző karakterkészletek felhasználását. Azt, hogy éppen melyik legyen érvényben, előre meghatározhatjuk, de a füzéren belül is előírhatjuk. A CTRL/A billentyűk lenyomásával az első, azaz nagybetűk/grafikus jelkészletre (a füzérben egy inverz "A"), míg a CTRL/B (füzérben inverz "B") lenyomásával a második, azaz kisbetű/nagybetűs jelkészletre kapcsolhatunk. Összefoglalva:

Billentyű:	Funkció:	Füzérben:
CTRL/A	nagybetű/grafikus jelkészlet	inverz "A"
CTRL/B	kisbetű/nagybetű jelkészlet	inverz "B"

Ezzel a módszerrel a füzéren belül is váltogathatjuk a karakterkészletet anélkül, hogy az előző kijelzéseket befolyásolnánk.
Ha az üzemmódra vonatkozóan semmilyen előírást nem alkalmazunk, automatikusan az első, azaz a CTRL/A lesz érvényben. Az alábbi kis példával mindez kipróbálható:

```
10 HIRES 6,7
20 TEXT 50,100"(CTRL/A)TEXT-(CTRL/B)TEXT",1,1,8
30 WAIT 198,255
```

A CTRL/A és a CTRL/B természetesen a két billentyű egyidejű lenyomását jelenti.
A két további paraméter (rü és n) az előzőekből már ismert. rü a rajz-üzemmódot, n pedig a karakterek függőleges nagyságát határozza meg.
A továbbiakban az utolsó paraméterrel (t) megválaszthatjuk a karakterek közti távolságot. Ezen két egymás mellett lévő karakter bal felső pontjainak képernyőpontokban mért távolságát értjük. Mivel 1–1 karakter egy 8 × 8-as mátrixban helyezkedik el, t > 7 értékek esetén a karakterek átfedik egymást. A feliratot ezzel a paraméterrel tetszőlegesen szét is húzhatjuk és ezáltal különleges hatásokat érhetünk el.
Győződjünk meg a leírtakról. A TEXT utasítás a variációk széles skáláját kínálja, próbáljuk ki.
Ki kell térnünk még az utasítás egy kis hiányosságára: a rendelkezésünkre álló SIMON's BASIC V2 változat nem fogadja el változóként sem a t-, sem az n-paramétert. (Ezt már a 9.4.3 pontban, a MOVE utasításnál is említettük.) Ha mégis így próbáljuk megadni, vagy eltűnik a felirat, vagy további karakterekkel bővül.
Ezek után nézzük a példákat.
Először is a már ismert gyakorlópélda, amelyen tanulmányozhatjuk az egyes paraméterek funkcióját.

P. 86

```
1 REM **************
2 REM * 1.T E X T *
3 REM **************
100 REM
110 HIRES 1,0:REM GRAFIKA BEKAPCSOLASA
120 REM GYAKORLO PELDA
130 REM VALTOZTASSUK A PARAMETEREKET !
140 REM
150 REM   X!  Y!    STR    !ZM! G!  A
160 REM ----------------------------
170 REM    !   !          !  !  !
180 TEXT 50,100, "SZOVEG" , 1, 2, 30
190 REM ----------------------------
200 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

P. 87

```
1 REM **************
2 REM * 2.T E X T *
3 REM **************
100 REM
110 HIRES 9,10:REM GRAFIKA BEKAPCSOLASA
120 FOR X=1 TO 8
130 TEXT 100,25*X-20,"CBM 64",1,3,20
140 NEXT X
150 PAUSE 5
160 REM
170 REM
180 HIRES 9,10:REM GRAFIKA TORLESE
190 REC 4,4,311,191,1
200 LINE 0,0,319,199,1
210 LINE 319,0,0,199,1
220 BLOCK 90,80,220,120,1
230 BLOCK 100,90,210,110,0
240 TEXT 103,97,"SIMON'S BASIC",1,1,8
250 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

12.5.3 Ellenőrző kérdések

A fejezet végén, mint már tudjuk, ismét ellenőrző kérdésekkel győződhetünk meg arról, hogy mit sikerült elsajátítanunk. Nos, munkára fel!

1. Szeretnénk a képernyő 160,100 pozíciójában egy normál nagy A betűt megjelentetni. Melyik utasítást kell választanunk?

   a) CHAR 160,100,128+1,1,1
   b) TEXT 160,100,"A",1,1,1
   c) CHAR 160,100,1,1,1
   d) TEXT 160,100,"A"1,2,3

2. Mi történik a rajzolandó karakterrel, ha a CHAR vagy a TEXT utasításban a t-paraméter értékét növeljük?

   a) A karakter függőleges méretei csökkennek.
   b) A karakter x-irányú méretei nőnek.
   c) A karakter y-irányú méretei nőnek.
   d) A karakter mindkét irányban nő.
   e) A karakter vízszintes irányú méretei csökkennek.
   f) Egyik megoldás sem helyes.

3. Mekkora legyen a t-paraméter értéke, ha azt akarjuk, hogy a TEXT utasításban a karakterek távolsága 1 pontnyi legyen?

   a) t=0
   b) t=1
   c) t=7
   d) t=8
   e) t=9

13. FEJEZET
KARAKTERKÉSZLET LÉTREHOZÁSA

A Commodore 64-es a szokatlanul változatos grafikus utasításokon kívül még más különlegességeket is kínál. Ezek egyike e karakterkészlet szoftver útján történő megváltoztatásának lehetősége, mely bizonyára nagy érdeklődésre tarthat számot. Ez ugyanis szinte az összes számítógépes játék alapja, valamint (a sprite-ok mellett) ez teszi a CBM 64-esen futtatott játékprogramokat gyorssá és ugyanakkor látványossá. Enélkül egy értelmes játékprogram elképzelhetetlen lenne. Míg más számítógépek kínlódnak a nagy grafikus tárolóval, addig a C64-esnek ez meg se kottyan. Ezzel kapcsolatban néhány kérdést azért tisztáznunk kell.

Mit is értünk karakterkészlet alatt?

Első közelítésben a kérdés egyszerűen megválaszolható. Karakterkészletnek azok karakterek (betűk és grafikus jelek) összességét nevezzük, melyeket a különböző billentyűk és billentyűkombinációk (SHIFT,C=) működtetésével a képernyőn megjeleníthetünk.
Ezeknek a karaktereknek az alakját a számítógépnek természetesen ismernie, azaz valahol és valahogyan tárolnia kell. További feltétel még, hogy a karakterek valamilyen jól meghatározott rendszer szerint helyezkedjenek el, mivel a számítógép csak így fogja az "A" billentyű lenyomása után a képernyőn az "A" betűt megjeleníteni. Természetesen ezek az információk minden számítógépben megtalálhatók.
A CBM 64 esetében a karakterkészleteket tartalmazó tár minden programból elérhető, vagyis tartalma olvasható és más tartományokba átvihető. Mindezzel azonban még nem sokra megyünk, ugyanis az említett tár tartalmát nem tudjuk megváltoztatni (az ilyen tár közismert neve: ROM). Más a helyzet az ún. RAM-mal, melyben a programok és a változók találhatók, és ez a számítógép kikapcsolásával kiürül. (RAM = Read Access Memory, írható-olvasható memória).
Az előbbiekből következik, hogy a BASIC operációs rendszer szintén a ROM-ban található.
Van azonban olyan lehetőség is, hogy az egyes karakterek tárolócímét megváltoztassuk. Így olyan utasítást is adhatunk a CBM 64-esnek, hogy a karakterek körvonalát jelentő információkat ezután ne a ROM-ból, hanem az $2000(=8192) címmel kezdődő tartományból (vagyis a RAM-ból) kérdezze le. Ebben a tartományban pedig már elvégezhetjük a kívánt módosításokat. Ha az eredeti karakterkészletet előzetesen a RAM-ba másoltuk, először semmit sem veszünk észre, mivel minden információ változatlan marad. Amennyiben viszont módosítjuk ezen tartomány bizonyos elemeit, megváltozik az ehhez tartozó karakterek alakja is. Ezzel a következő kérdés adódik:
Hogyan tárolja a számítógép az egyes karakterek alakját? Először is – mint tudjuk – számítógépünk összesen 4 karakterkészlettel rendelkezik, melyek mindegyike 128 karaktert tartalmaz. A karakterkészletek (kijelzési üzemmódok) közül legfeljebb kettő lehet érvényben. A 4 karakterkészlet a következő:

A/1 karakterkészlet – normál nagybetűk/grafikus jelek.
A/2 karakterkészlet – inverz nagybetűk/grafikus jelek.
B/1 karakterkészlet – normál nagybetűk/kisbetűk.
B/2 karakterkészlet – inverz nagybetűk/kisbetűk.

Az A és B karakterkészleteket az ismert módon, tehát a C= és SHIFT billentyűk egyidejű lenyomásával közvetlenül cserélhetjük. A programból történő átváltáshoz a 14 és 142 (128+14 = 142) ASCII-értékeket kell használni, tehát

PRINT CHR$(14)-gyel

a B készletre, míg a

PRINT CHR$(142)-vel

az A-ra kapcsolhatunk át. (Lásd még a CSET Ø és CSET 1 utasításokat is). Pl. a

PRINT CHR$(8)

utasítással letilthatjuk a billentyűzetről végzett (programfutás közben is kivitelezhető) átkapcsolási lehetőséget, míg ennek feloldása a

PRINT CHR$(9)-cel

történik (lásd a CBM 64 kezelési kézikönyvet).
A normál (1) és inverz (2) ábrázolás a ⟨RVS ON⟩ és ⟨RVS OFF⟩ billentyűkkel váltható ki. A memóriában minden karakterkészlet külön kell, hogy szerepeljen. Összesen tehát 4 × 128 = 512 karaktert tárolhatunk, de közülük egyszerre csak 256-ot használhatunk.
A képernyőn minden karakternek egy 8 × 8-as pontmátrix felel meg. Ez azt jelenti, hogy a memóriában is 64 pont felel meg egy karakternek. A gyakorlatban ez úgy valósul meg, hogy egy képpontot egy bit képvisel, így egy karakter tárolásához összesen 64 bitre, vagyis 8 byte-ra van szükség. Az egyes byte-ok a karakter különböző sorait ábrázolják. Az "1" értékű bit a bekapcsolt állapotnak felel meg.
A karakter megjelenítése a következőképpen szemléltethető:

	0.	1.	2.	3.	4.	5.	6.	7. bit
0 byte	.	.	.	.	.	.	.	.
1 byte	.	.	.	.	.	.	.	.
2 byte	.	.	.	.	.	.	.	.
3 byte	.	.	.	.	.	.	.	.
4 byte	.	.	.	.	.	.	.	.
5 byte	.	.	.	.	.	.	.	.
6 byte	.	.	.	.	.	.	.	.
7 byte	.	.	.	.	.	.	.	.

A karaktertároló 512 ilyen, egyébként 8 byte-os, kódsorozatból áll, amely összesen 4K (= 4096 byte) ROM-kapacitást foglal le a $D000–$DFFF (53248–57344) címek között. Példaként lássuk, hogy tárolja a számítógép a nagy A betű képét:

	0	1	2	3	4	5	6	7 bit
0 byte	.	.	.	*	*	.	.	.
1 byte	.	.	*	*	*	*	.	.
2 byte	.	*	*	.	.	*	*	.
3 byte	.	*	*	*	*	*	*	.
4 byte	.	*	*	.	.	*	*	.
5 byte	.	*	*	.	.	*	*	.
6 byte	.	*	*	.	.	*	*	.
7 byte	.	.	.	.	.	.	.	.

Ez átszámítva az alábbi 8 byte-nak felel meg:

0 byte: 00011000 = $18 = 024
1 byte: 00111100 = $30 = 060
2 byte: 01100110 = $66 = 102
3 byte: 01111110 = $7E = 126
4 byte: 01100110 = $66 = 102
5 byte: 01100110 = $66 = 102
6 byte: 01100110 = $66 = 102
7 byte: 00000000 = $00 = 000

Ez a 8 érték a karaktertár normál nagy A betű részére fenntartott címén foglal helyet. De hogyan kapjuk meg az egyes karakterek kezdőcímeit?
A számítás alapja az egyes karakterek képernyőkódja. Ez az a kód, amely a karakterek helyét a képtárolóban meghatározza (lásd CBM 64 felhasználói kézikönyv).
Innen már viszonylag egyszerű a dolog. Mivel minden karakter 8 byte-ot foglal el, így a képernyőkód nyolcszorosát ha hozzáadjuk a karakterkészlet tárolójának kezdőcíméhez (normál esetben: D 000), megkapjuk a kívánt címet.
Képlettel:

Keresett cím = kezdőcím + 8* képernyőkód

Számítsuk ki mindezt az A betű esetében:

Az A karakter kezdőcíme: $ D000 + 8*1, = $D008 = 53238

Most már van némi ismeretünk a karakterkészletek előállításáról. A további részleteket a SIMON's BASIC egyes utasításainál találhatjuk meg.

13.1 MEM – a karakterkészlet áthelyezése a RAM-ba

Formátum: MEM
Paraméterek: –
Funkció: A karakterkészlet áthelyezése a $D000 (ROM) címről a $E000 (RAM) címre
Példa: MEM

Mint ahogy azt a bevezetőben láttuk, a karakterkészlet eredetileg a $D000–$DFFF címekkel határolt 4K hosszúságú ROM-tartományban helyezkedik el. Mivel az a célunk, hogy néhány karakter képét megváltoztassuk, ezt a tartományt a RAM-területre át kell helyezni. Ezt a feladatot végzi el a MEM utasítás, amely a karakterkészletet a $E000–$EFFF (57344–61439) RAM-tartományba helyezi át. Mivel azonban a grafikus tároló is ezen a területen ($E000–$EFFF) található, ezért egyszerre karakterkészletet változtatni és grafikát alkalmazni nem lehet. Ha a fentieket figyelmen kívül hagyjuk, érdekes látványnak lehetünk szemtanúi. Ezt szemlélteti az alábbi három sor:

```
10 HIRES 6,7 : REM GRAFIKA BE
20 MEM : REM A KARAKTERKÉSZLET ELTOLÁSA ÉS ÁTMÁSOLÁSA
30 WAIT 198,255 : REM BILLENTYŰ LENYOMÁSÁRA VÁR
```

Tehát először bekapcsoljuk a grafikát, utána pedig karakterkészletet váltunk. Ezáltal a teljes karakterkészlet megjelenik a képernyő felső felében. Mivel a karakterkészlet és a grafika hasonló felépítésű, a karakterek helyesek maradnak, felismerhetők.
A program leállítását követően a gép nem tér vissza a szokásos módon a szöveges kijelzésre, hanem grafikus üzemmódban marad (lásd még a 12. fejezetet). A beírt szöveg csak kis színes négyzetek formájában jelenik meg, mivel a video-RAM (a szín- és képtároló) a MEM-utasítás hatására most a $CC00 (52224) címen található. A SIMON's BASIC azonban grafikus üzemmódban a színtárolót a $C000 (49132) tehát a szokásos címen keresi, ezért nincs lehetőség a szín grafikus utasítással való változtatására. Csak a CSET vagy az NRM utasítással térhetünk vissza ismét szöveges kijelzésre.

Összefoglalva a MEM utasítás funkcióit:

a) Áthelyezi a karakterkészlet-ROM tartalmát a $E000 (57344) RAM címre.
b) Közli a számítógéppel, hogy a karakterkészlet most ezen a címen található.
c) Eltolja a video-RAM-ot a $CC00-ra.
d) Törli a képernyőt.
e) Áthelyezi a sprite tartományt $C00-ra.

Nézzük az erre vonatkozó példákat:

P. 88

```
1 REM *********
2 REM * M E M *
3 REM *********
100 REM
110 MEM:REM KARAKTERKESZLET ATHELYEZESE A RAM-BA
120 REM
130 BA=14*4096:REM BAZISCIM $E000
140 PRINT CHR$(147):REM KEPERNYOTORLES
150 REM
160 REM ************************
170 REM A KARAKTEREK KILISTAZASA
180 REM ************************
190 REM
200 FOR Y=0 TO 24
210 FOR X=0 TO 39 STEP 4
220 FILL Y,X,4,1,C,0
230 C=C+1:REM A KOD NOVELESE
240 NEXT X:NEXT Y
250 PAUSE 2
260 REM
270 C=0: REM KEPERNYOKOD
280 REM
290 REM ************************
300 REM KARAKTEREK VILLOGTATASA
310 REM ************************
320 REM
330 FOR X=0 TO 40
340 T=ABS(T-255):REM VALTOZTATAS 0-255 KOZOTT
350 FOR Y=0 TO 7
360 POKE BA+C*8+Y,T:REM VILLOGTATAS
370 NEXT Y:NEXT X
380 REM
390 REM ************************
400 REM KARAKTEREK VALTOZTATASA
410 REM ************************
420 RE'1
430 REM              %%%%%%%%%%%%
440 POKE BA+C*8+0,%01100000 :
450 POKE BA+C*8+1,%01101100 :
460 POKE BA+C*8+2,%01111110 :
470 POKE BA+C*8+3,%00110110 :
480 POKE BA+C*8+4,%00000110 :
490 POKE BA+C*8+5,%00001111 :
500 POKE BA+C*8+6,%00000110 :
510 POKE BA+C*8+7,%00000000 :
520 REM              %%%%%%%%%%%%
530 REM
540 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
550 PRINT CHR$(147):REM KEPERNYOTORLES
560 LIST
```

Példaprogramunkban először áthelyezzük a karakterkészletet (110. sor), hogy a karaktereket megváltoztathassuk. Ezután megjelenik a képernyőn az A/1 és A/2 karakterkészlet (csaknem) valamennyi karaktere. A megváltoztatandó karakter egy rövid időre felvillan, majd a program kiírja az új karakter 8 byte-ját (440.–510. sorok, ezekben a sorokban adjuk meg az új karakterek byte-jait). A feladat megoldásához már ismertetett címképletet alkalmazzuk (a C változó a karakter képernyőkódját jelenti). Az új karakterek definiálását a 13.3 pontban tanulmányozhatjuk.

13.2 DESIGN 2 – karakter létrehozása

Formátum: DESIGN 2, za
Paraméterek: a módosítandó karakter címe a za = $E000+B*C címképlet alapján
(C: a karakter képernyőkódja)
Funkció: Tetszőleges karakter létrehozása
Példa: DESIGN 2, $E000+8*1

A MEM utasításnál már tapasztaltuk, hogy tetszőleges karaktereket hozhatunk létre. Kényelmesebbé teszi a munkát a SIMON's BASIC másik két összetartozó utasítása, melyek közül most az elsőt mutatjuk be:
A csak a MEM utasítás után bevihető DESIGN utasítás kijelöli a megváltoztatandó karaktert, továbbá a programon belüli pozíciójából meghatározza, hogy melyik karakterkészletet használja fel.
Ez a következőképpen történik:
A za paraméterrel közöljük a módosítandó karaktert, vagyis a karakterminta helyét a karakterkészlet tárolójában. Ehhez a már ismert

cím = kezdőcím + 8*képernyőkód

összefüggést használjuk. Minthogy egy korábbi MEM utasítás hatására a karakterkészlet átkerült a $E000 (57344) címre, a képletben szereplő kezdőcím értéke $E000. A képernyőkód értékét a megváltoztatni kívánt karakter határozza meg, értéke a CBM 64 felhasználói kézikönyvből kereshető ki.
Az egyenlet az A betűre a következő:

za = $E000+8*1 = $E008

Az $E000 helyett alkalmazhatjuk annak decimális megfelelőjét is. A DESIGN utasítás után közvetlenül be kell írni az ún. mintasorba az új karakterminta kódjait. Ellenkező esetben "BAD CHAR FOR A MOB" hibajelzést kapunk. A továbbiakat a 13.3 pontban találjuk meg.

13.3 @ a karakterminta tárolása

Formátum: @... karakterminta...
Paraméterek: karakterminta (lásd alább)
Funkció: a karakterminta tárolása
Példa: lásd alább

A karakterek megváltoztatásának kulcs-utasításához értünk: a MASTER SPACE vagy ADD jelhez. Funkcióját tekintve megfelel a normál BASIC DATA utasításának. Mindig a sor elején áll, és adatok sorozatát, jelen esetben a megváltoztatandó karakterek definícióját jelöli. Működése a következő:
Mint tudjuk, minden karakter egy 8 × 8-as pontmátrixból tevődik össze. Ennek megfelelő a karakterdefiníció is. A karakterhez tartozó 8 sort külön-külön BASIC sorként adjuk be, ahol mindegyik sor egy karaktersor 8 pontjának mintáját rögzíti:(normál nagy A betű)

```
100  . . . BB . . .
110  . . BBBB . .
120  . BB . . BB .
130  . BBBBBB .
140  . BB . . BB .
150  . BB . . BB .
160  . BB . . BB .
170  . . . . . . . .
```

A karakter minden bekapcsolt pontjába egy-egy B betűt, míg az üres helyekre pontot kell írni. Így már a BASIC program listájában megjelenik a betű nagyított képe (a 14. fejezetben látni fogjuk, hogy a sprite-okat is hasonló módon definiáljuk).
A DESIGN utasítást közvetlenül a definíció előtti sorban kell elhelyezni. Ez határozza meg a módosítandó karakter címét:

```
90 DESIGN 2, $E000+C*8
```

ahol a C a képernyőkód.
Legelőször azonban a karakterkészletet kell áthelyeznünk:

```
80 MEM
```

Ezzel a karaktert újradefiniáltuk (ha pl. a C-t 1-re választja, természetesen semmi nem történik, hiszen az A-karakter éppen ilyen alakú).
A mintasorban a pontok helyett szóközöket is használhatunk, viszont a két szélső hely közül legalább az egyiken pontnak kell állnia, különben a "BAD CHAR FOR A MOB" hibaüzenetet kapjuk éppúgy, mintha valamelyik sorban hibás, vagy túl sok karakter szerepelne.
Az utasítás segítségével tehát különleges karaktert, sőt speciális karakterkészletet hozhatunk létre. Minthogy azonban ez meglehetősen idő- és memóriaigényes, ezért jobb, ha ilyen célra külön karakterkészlet-előállító programokat alkalmazunk és a komplett karakterkészletet közvetlenül olvassuk be a memóriába. Másik módszerként alkalmazható az ún. OVERLAY eljárás. Ennek az a lényege, hogy először az új karakterkészletet előállító programot töltjük be és miután ez lefutott, automatikusan hívja a tulajdonképpeni programot. Ehhez persze tudnunk kell, hogy az utóbbi milyen hosszú. Ha rövidebb, mint a karakterkészletet definiáló program, akkor nincs probléma és a

```
200 LOAD" programnév",8
```

sor elvégzi a beolvasást, valamint az automatikus indítást.
Abban az esetben, ha hosszabb, akkor a

```
190 POKE 45,AD-int (AD/256)*256
195 POKE 46,    -int (AD/256)
```

utasításpárral meg kell adnunk, hogy mekkora lesz a főprogram hossza. Az AD cím a BASIC tárolónak azt a címét tartalmazza, ameddig a főprogram terjed. Ez a cím SIMON's BASIC-ben a következő képlettel állapítható meg:

```
AD = 30720-FRE(0)+$801
```

Természetesen ez az összefüggés csak akkor eredményes, ha a program a gép memóriájában van. Most már ezt a konkrét címet kell az első programba beépíteni. Érdekességként nézzük meg a fenti képlet magyarázatát: A SIMON's BASIC betöltésekor a gép memóriájában 30720 szabad, programozható byte marad. A FRE(0) a program betöltése után maradó szabad memóriaterületet adja meg byte-okban. Ehhez az értékhez még hozzá kell adni a BASIC-program kezdőcímét, ami normál esetben $801 (decimálisan 2049), és máris rendelkezésünkre áll a BASIC programunk címe. Tehát a programba a

```
180 AD=100
```

sort kell beírni, ha a programunk 100 byte terjedelmű. Ennyi ismeret röviden elég is lesz az ún. OVERLAY eljárásról, ami persze ennél összetettebb. Az viszont tény, hogy segítségével több BASIC programot is összekapcsolhatunk.
Nem jelent nagy problémát, ha a leírtakból nem értett meg mindent. Ha mélyebben meg szeretne ismerkedni a fenti gondolatmenettel, akkor javaslom, hogy szerezzen be egy olyan könyvet, amely az ilyen és ehhez hasonló profi programozási fogásokat részletesen ismerteti. Elsőként "A BASIC programozás magasiskolája C64-esen" c. DATA BECKER–NOVOTRADE kiadványt ajánlom.
A továbbiakban térjünk rá a mintaprogramunkra:

P. 89

```
1 REM ******************
2 REM * D E S I G N 2 *
3 REM ******************
100 REM
110 MEM:REM A KARAKTERKESZLET ATHELYEZESE
120 PRINT:REM SOREMELES
130 CENTRE"EZUTAN MAR MINDEN SZOVEG":PRINT
140 CENTRE"AZ UJ KARAKTERKESZLETTEL":PRINT
150 CENTRE"IRHATO !":PRINT
160 PRINT:REM SOREMELES
170 CENTRE"A B C D E"
180 PAUSE 5
190 REM
200 REM ***********************
210 REM * KARAKTERDEFINICIO *
220 REM ***********************
230 REM
240 DESIGN 2,$E000+1*8:REM A
250 @ .BBBBBBB
260 @ ..BB..BB
270 @ .BBBBBBB
280 @ .BBBBBBB
290 @ ..BB..BB
300 @ ..BB..BB
310 @ .BBB..BB
320 @ ........
330 DESIGN 2,$E000+2*8:REM B
340 @ .BBBBBBB
350 @ ..BB..BB
360 @ ..BB..BB
370 @ .BBBBBBB
380 @ ..BB..BB
390 @ ..BB..BB
400 @ .BBBBBBB
410 @ ........
420 DESIGN 2,$E000+3*8:REM C
430 @ .BBBBBBB
440 @ ..BB..BB
450 @ ..BB....
460 @ ..BB....
470 @ ..BB....
480 @ ..BB..BB
490 @ .BBBBBBB
500 @ ........
510 DESIGN 2,$E000+4*8:REM D
520 @ .BBBBBBB
530 @ ..BB..BB
540 @ ..BB..BB
550 @ ..BB..BB
560 @ ..BB..BB
570 @ ..BB..BB
580 @ .BBBBBBB
590 @ ........
600 DESIGN 2,$E000+5*8:REM E
610 @ .BBBBBBB
620 @ ..BB..BB
630 @ ..BB....
640 @ ..BBBB.
650 @ ..BB....
660 @ ..BB..BB
670 @ .BBBBBBB
680 @ ........
690 PAUSE 10
```

```
700 PRINT CHR$(147):REM KEPERNYOTORLES
710 CENTRE"A MEM UTASITASSAL MINDENT":PRINT
720 CENTRE"VISSZAALLITHAT !"
730 PAUSE 10
740 MEM
750 PRINT:REM SOREMELES
760 CENTRE"LATJA ?"
770 PRINT:PRINT
780 CENTRE"A B C D E"
790 WAIT 198,255:REM VARAKOZAS EGY BILLENTYU LENYOMASARA
```

13.4 CSET 0/1 – karakterkészlet váltása

Formátum: CSET 0 vagy CSET 1
Paraméterek: –
Funkció: Átkapcsol a második karakterkészletre
Példa: CSET 0

A CSET utasítás CSET 2 változatát már a 12.2.4 pontban megismertük. Itt a grafikát kapcsolta be anélkül, hogy előtte törölte volna a grafikus tárolót. Most a CSET utasítás két újabb változatával ismerkedünk meg, melyek szintén könnyen megérthetők. A CSET 0 teljesen megfelel az NRM utasításnak (lásd részletesen a 9.3.3. fejezetben), azaz néhány egyéb feladat mellett bekapcsolja a nagybetű/grafikus jelkészletet. Ezt a részfeladatot, mint tudjuk, a

```
PRINT CHR$(142)
```

utasítással helyettesíthetjük.
A CSET 1 utasítás segítségével aktivizálható a kisbetűs/nagybetűs karakterkészlet úgy, hogy közben a többi funkciója (a grafika, a EXTENDED COLOUR üzemmód és a MEM utasítás kikapcsolása) is megmarad. A fenti utasítás a

```
PRINT CHR$(14)
```

sorral helyettesíthető.
A jobb megértés érdekében nézzük a következő példát:

P. 90

```
1 REM ***************
2 REM * C S E T 0/1 *
3 REM ***************
100 REM
110 PRINT CHR$(147):REM KEPERNYOTORLES
120 CSET 0:REM NAGYBETU-/GRAFIKUS JELKESZLET
130 CENTRE"JELENLEG A NAGYBETU-/GRAFIKUS"
134 PRINT
135 CENTRE"JELKESZLET UZEMMODBAN VAGYUNK."
140 PRINT:REM SOREMELES
150 PAUSE 5
160 CENTRE"AMIT ROGTON MEGVALTOZTATUNK."
170 PRINT:PRINT
180 PAUSE 5
190 CENTRE"FIGYELEM !"
200 PAUSE 2
210 CSET 1:REM KIS-/NAGYBETUS UZEMMOD
220 WAIT 198,255:REM VARAKOZAS EGY GOMB LENYOMASARA
```

13.5 Ellenőrző kérdések

1. Mely állítások igazak a MEM utasításra?

   a) A MEM utasítás kikapcsolja a grafikát.
   b) A MEM utasítás átmásolja a karakterkészletet a $E000-tól kezdődő tartományba.
   c) A grafika és a saját karakterkészlet egyszerre nem használható.
   d) A MEM utasítás hatását az NRM és a CSET 0 utasítás feloldja.
   e) A MEM utasítás hatását a CSET 2 és a CSET 1 utasítás feloldja.

2. Melyik állítás vonatkozik a DESIGN 2 utasításra?

   a) A DESIGN 2 utasítás kikapcsolja a grafikát.
   b) A DESIGN 2 utasítás a karaktermintát dolgozza fel.
   c) A DESIGN 2 utasítást követő paraméter megadja a karakterdefiníció címét.
   d) A DESIGN 2 utasítás grafikus üzemmódban nem használható.

14. FEJEZET
SPRITE-OK (MOB-OK)

A Commodore 64-es egyik különlegessége, hogy sprite-okat (vagy más néven MOB-okat = Movable Objekt Blocks = mozgó alakzatok) képes előállítani. A sprite-ok olyan önálló, kisméretű, grafikus figurák, melyek egymástól és a képernyő többi részétől függetlenül mozgathatók a szöveges vagy grafikus üzemmódú képernyőn.
Egyidőben 8 sprite lehet a képernyőn. A sprite-ok színe, nagysága, prioritása szabadon változtatható és lehetőség van a képernyővel és egymással való ütközésük érzékelésére. Mindezen funkciók könnyen programozhatók a VIC (Videocontroller 6567) és regiszterei segítségével. A SIMON's BASIC megkönnyíti a sprite-ok kezelését, használatával az összes funkció könnyen és áttekinthetően programozható.
Most tekintsük át a sprite-ok felépítését:
Minden sprite 504 pontból áll, amelyek külön-külön be-, ill. kikapcsolhatók. Az 504 pont egy 24 × 21-es pontmátrixban helyezkedik el, ami azt jelenti, hogy egy sprite 24 pont széles és 21 pont magas. Ezen a tartományon belül a legkülönbözőbb ábrák és figurák alakíthatók ki. A sprite felépítése a következő ábrával szemléltethető:

```
012345678901234567890123
0       ........................
1       ........................
.
.
.       stb.
.
.
20      ........................
21      ........................
```

A felépített sprite-ok pontos tárolásának a kérdésével nem kívánunk foglalkozni, mivel ez a SIMON's BASIC-ben nem játszik szerepet. Itt a sprite-okat nagyon egyszerűen hozhatjuk létre. A vázolt területen belül kedvünk szerint alakíthatjuk ki a figurákat és pontonként 16 színből válogathatunk.
Azonban nem minden sprite épül fel 24 × 21 pontból, csak a nagyfelbontásúak. A többszínű sprite-ok a többszínű grafikához hasonlóan alakíthatók ki. X-irányban feleakkora felbontásúak, tehát soronként 12 duplaszéles pont helyezkedik el. Cserében viszont (a háttérszínnel együtt) 4 szín szerepelhet bennük, míg a finomfelbontású sprite csak 2 színű lehet. A négy szín a VIC különböző regisztereiben tárolódik (lásd CBM 64 felhasználói kézikönyv). Meg kell azonban említeni, hogy csak a 2. szín lehet minden sprite-nál különböző, a többi szín (az 1., a 3. és a háttérszín) valamennyi sprite esetében azonos lesz, mivel ezek azonos regiszterből származnak. A CBM 64 felhasználói kézikönyvében leírtakkal ellentétben MULTICOLOR üzemmódban is használható mind a 16 szín.
A sprite alakján és színén kívül még néhány más jellemzőt is programozhatunk. Például kétszeresére nagyíthatjuk az alakzatot x, y vagy mindkét irányban, ill. megadhatjuk a sprite/háttér prioritást. A sprite-ot helyezhetjük a többi grafika elé, ill. mögé. Az előbbi esetben a sprite a képernyőn található grafikák előtt, az utóbbi esetben ezek mögött mozog. Prioritás állapítható meg a sprite-ok között is, amit a sprite számával adunk meg (a nyolc lehetséges sprite sorszáma: 0–7 között van). A nagyobb sorszámú sprite-ok prioritást élveznek a kisebb sorszámúakkal szemben. Ezzel a módszerrel rendkívül egyszerűen lehet térhatást előidézni.
De ezzel még mindig nincs vége. A Commodore 64 még sok csemegét tartogat a hozzáértők számára. Képes arra, hogy a sprite háttér, ill. sprite ütközéseket érzékelje. Ehhez szintén különböző VIC regisztereket használ, amivel azonban megint nem kell foglalkozni, mert a SIMON's BASIC mindent elintéz. Néhány fontos tudnivalóra az utasítások (DETECT és CHECK) ismertetésénél hívjuk fel a figyelmet. Lényeges még megemlíte-

ni, hogy a háttér sprite, és sprite ütközések között is differenciálhatunk. Az utóbbi esetben még azt is eldönthetjük, hogy a nyolc sprite közül melyek ütközhetnek össze.
Röviden szólni kell még a sprite-ok tárolásáról. A sprite tárolásakor a számítógép a RAM-ot 64 byte-os csoportokra, ún. blokkokra osztja, amelyek mindegyike sorszámmal rendelkezik. Így pl.: a tároló első 64 byte-ja (0–63 közötti címek) a nulladik számú blokk, stb... Ha egy sprite-ot definiálni akarunk (ami – mint látni fogjuk – a DESIGN utasítással realizálható), akkor ezt először egy ilyen blokkban kell elhelyezni. A sprite ábrázolásakor hivatkozni kell a tároló címére. Így megtehetjük azt is, hogy mind a nyolc sprite-hoz ugyanazt az alakot, méretet, színt rendeljük, ha a különböző sprite-számokhoz ugyanazt a tárolóblokkot jelöljük ki. A blokkszámozásnál csak a 0 és 255 közötti értékek állnak rendelkezésére, ami azt jelenti, hogy figurákat csak $255 \times 64 = 16\,384$ byte-ban helyezhetjük el. Azt, hogy ez a 16K melyik memóriacímtől kezdődik, a video-RAM (képernyőtároló) helyzete szabja meg, ami a SIMON's BASIC-ben a következő:

Üzemmód	Tárolócím
normál szöveges üzemmód	0–16383($0000–$3FFF)
új karakterkészlet	49152–65536($C000–$FFFF)
grafikus üzemmód	49152–65536($C000–$FFFF)

Ebből következik, hogy ha a grafikus és szöveges üzemmódban egyszerre dolgozunk, a sprite definíciók a tár eltérő tartományaiban lesznek, bár a blokkszámozás nem változik. Ha a két üzemmódot felváltva használjuk, vegyük figyelembe, hogy a szöveges kijelzésre a tárba bevitt sprite grafikus üzemmódban nem, vagy csak teljesen értelmetlen formában jelenhet meg. Ennek oka az, hogy a szóban forgó blokkért felelős tártartomány a sprite definíciót nem tartalmazza. Fordítva ugyanez érvényes.
Ilyen esetben az a megoldás, hogy ugyanazt a sprite-ot még egyszer átviszi a grafikus blokktartományba, vagy a grafikus üzemmódra való átváltáskor egyszerűen kikapcsolja.
A blokkok kiválasztásánál még egy fontos dolgot kell szem előtt tartani: nem használhatók a már foglalt blokkok (pl.: nulláslap stb.). Most pedig nézzünk meg néhány lehetséges tárolóhelyet a blokkok számára:

Szöveges üzemmód

Blokkszám	Címek	Megjegyzések
13–15	832–1023($0340–$03FF)	Korlátozás nélkül használható
64–255	4096–16383($1000–$3FFF)	hosszú BASIC programoknál vigyázat

Grafikus üzemmód

16–63	50176–53247($C400–$CFFF)	DATA BECKER IEC busszal (S-változat) csak a 47. blokkig

Új karakterkészlet

16–47	50176–52223($C400–$CBFF)	Korlátozottan használható
195–255	61440–65535($F000–$FFFF)	Korlátozás nélkül használható

Vigyázzunk, ha a C-oldalt használjuk (16–63 blokkok grafikus üzemmódban és új karakterkészlet alkalmazásakor), ugyanis itt található a SIMON's BASIC néhány közbenső tárolója (többek között a funkcióbillentyűk tartalma). Elképzelhető, hogy ebben a tartományban néhány olyan puffer is elhelyezkedik, melyek tartalmára, bizonyos funkciók kiesésekor lesz szükség.
Tehát, ha ezzel a tartománnyal akarunk dolgozni, akkor az első indítás előtt tároljuk a programunkat. Ezt egyébként érdemes máskor is megtenni, mivel a SIMON's BASIC hajlamos az összemásolásra.

14.1 A sprite-ok definiálása

14.1.1 DESIGN 0/1

Formátum: DESIGN 0, kc + b*64 vagy DESIGN 1, kc + b*64
Paraméterek: 0 – nagyfelbontású sprite
1 – többszínű sprite
kc – a video-RAM kezdőcíme
szöveges üzemmód: kc = 0
grafika: kc = $C000
új karakterkészlet: kc = $C000
b – a tárolóblokk száma
Funkció: a sprite definíció átvétele a memóriába
Példa: DESIGN 0, $C000X13*64

Bizonyára emlékszünk még a DESIGN 2 utasításra a 13.2. pontból, amivel új karaktert tudunk meghatározni szöveges üzemmódban.
A DESIGN 0 és DESIGN 1 utasítás teljesen analóg módon működik. A DESIGN 0/1 utasítással a megtervezett sprite-ot vihetjük át az ún. mintasorból (ADD sor; lásd 14.1.2) az illető memóriatartományba. Az utasítás mindhárom változatának közvetlenül a mintasor előtti sorban kell állnia, ha a "BAD CHAR FOR A MOB" hibajelzést el akarjuk kerülni. A 0 és 1 paraméterrel a sprite típusát határozhatjuk meg:

– 0 esetén FFB
– 1 esetén többszínű sprite.

A második és harmadik paraméter, a 13.2 ponthoz hasonlóan azt a memóriatartományt jelöli ki, ahová a sprite definíció kerül.
Jelen esetben "kc" a 0. blokk címét (a videocím tartomány báziscímét), b pedig a kijelölt blokk számát jelenti. További részleteket a 14. fejezet bevezetőjében, vagy a 14.1.2 pontban találunk.

14.1.2 @ – a sprite definiálása

Formátum: @ .....mintasor.....
Paraméterek: mintasor – sprite – definíció
pont(.): üres pont
B: pont az 1. színnel
C: pont a 2. színnel
D: pont a 3. színnel
Funkció: egy sprite definiálása
Példa: . . . B . . BBB . . . B . . . B . . . . B . B . B . stb. (21 sor)
A 13. fejezetben már ezzel az utasítással is találkoztunk, ugyanúgy, mint a mintasor jelével, amit az új karakter előállításához használtunk. A @-jel feladata most az, hogy bevezesse a sprite definíciót tartalmazó mintasort. Mint a bevezetőben már említettük, kétféle sprite létezik: a finomfelbontású és a többszínű. Ezeket természetesen különbözőképpen kell definiálni:
– A finomfelbontású sprite (FFB) esetében egy 24 × 21-es pontmátrixból állítjuk össze a sprite-ot. A SIMON's BASIC-ben tehát egy finomfelbontású sprite összesen 21, egyenként 24 pontot tartalmazó sorból építhető fel. A bekapcsolt pontokat B-vel, míg az üreseket (melyek a háttér színét veszik fel) ponttal jelöljük.
– Többszínű sprite esetén az alapszín mellett még két olyan színnel rendelkezünk, amely

valamennyi sprite-ban előfordulhat. Ezeket a színeket a C és D betűkkel szimbolizáljuk. Igaz, így soronként már csak 12 pont áll rendelkezésünkre, mivel a vízszintes felbontás a felére csökkent.

A definiálás módja a példaprogramon tanulmányozható. A betűk alkalmazásánál még egy dologra kell figyelnünk:

A finomfelbontású sprite-ok a B-vel jelölt pontok színét a MOB SET utasítás paraméteréből veszik.

A többszínű sprite-ok esetében a MOB SET utasítás a C-vel jelölt pontokra vonatkozik.

A kétféle sprite-ra az alábbi hozzárendelés szükséges:

– *finomfelbontású sprite-ok*

jel	színkód	Meghatározó színutasítás
.	0	háttérszín
B	1	MOB SET

– *többszínű sprite-ok*

jel	színkód	Meghatározó színutasítás
.	0	háttérszín
B	1	CMOB
C	2	MOB SET
D	3	CMOB

Ha véletlenül eltérünk a helyes formátumtól és más betűket vagy jeleket használunk (bizonyos esetben a szóköz kivételével; ld. 13. fejezet), a sorokat az engedélyezettnél hosszabbra választjuk, vagy a DESIGN utasítás és a mintasorok közé egy újabb BASIC sort írunk, akkor a számítógép a "BAD CHAR FOR A MOB" hibaüzenettel leáll. Igyekezzünk tehát a helyes írásmódot betartani.

Példa:

P. 91

```
1 REM ********************
2 REM * @ / D E S I G N *
3 REM ********************
100 REM
110 DESIGN 1,14*64
120 @ ....BBBBB...
130 @ ..BBB...BBB.
140 @ .BB.CC.CC.BB
150 @ .BB...C...BB
160 @ .BB...C...BB
170 @ ..BBBCCCBBB.
180 @ ....BBBBB...
190 @ .B...DDD....
200 @ .BB..DDD....
210 @ .BBBBDDDBB..
220 @ .....DDD.BB.
230 @ .....DDD..BB
240 @ .....DDD...B
250 @ .....DDD....
260 @ ...CCC.CCC..
270 @ ..CCC...CCC.
280 @ ..CC.....CC.
290 @ .CCC.....CCC
300 @ .CCC.....CCC
310 @ .BBBBBBBBBBB
320 @ ............
330 REM ***********
340 REM KIEGESZITES
350 REM ***********
360 REM
370 COLOUR 9,9
380 MOB SET 1,14,8,0,1
390 CMOB 7,6
400 MMOB 1,0,0,300,205,2,200
```

Példánknak mindössze az volt a célja, hogy megismertesse a kétfajta sprite definiálásának módját.
Ahhoz azonban, hogy a sprite a képernyõn is megjelenjen, még egy sor utasításra van szükségünk. Ezeket az utasításokat a 330. sortól mindenféle magyarázat nélkül egyszerûen hozzákapcsoltuk a programhoz.
Egyelõre fogadjuk el ezeket az utasításokat, használatukra késõbb visszatérünk.

14.1.3 MOB SET – a sprite jellemzõi

Formátum: MOB SET n,b,sz,p,a
Paraméterek: n – a sprite száma (0–7)
b – a sprite-ot tartalmazó blokk sorszáma (0–255)
sz – a sprite színe
finomfelbontású: B betû
többszínû: C betû
p – sprite-háttér prioritás
p = 0 sprite a háttér elõtt
p = 1 sprite a háttér mögött
a – a sprite típusa
a = 0 finomfelbontású
a = 1 többszínû
Funkció: a sprite jellemzõinek meghatározása
Példa: MOB SET 0,13,7,1,0

Mint ahogy a bevezetõben már olvastuk, a sprite bizonyos jellemzõi (szín, nagyság, ...) a definíciótól függetlenül utólag (vagy elõzõleg) is kijelölhetõk vagy megváltoztathatók. Ezt a feladatot részben a MOB SET utasítás végzi, amely mint látjuk, nem szûkölködik paraméterekben. Egy valamit azonban meg kell jegyeznünk: ez az utasítás a sprite-ot nem kapcsolja be, tehát ezzel az utasítással a sprite nem vihetõ a képernyõre.

Most pedig lássuk részletesen az egyes paramétereket.

Elõször is az n paraméterrel a sprite számát kell meghatározni a nyolc lehetséges közül. Mint tudjuk, minden sprite-hoz tartozik egy 0 és 7 közötti sorszám, ami egyben a prioritásokat is meghatározza. Ezt a számot minden sprite utasításban meg kell adni.
A második paraméter (b) kijelöli annak a memóriatartománynak a sorszámát, amelyben a sprite elhelyezkedik. Ezt elõzõleg a DESIGN-utasítással és a @-jellel definiáltuk. A MOB SET utasítás a mintát hozzárendeli a megfelelõ sprite számhoz, így a késõbbiekben elég a sprite számát megadni, és ezzel már meghatároztuk a sprite-ot. Így az is elképzelhetõ, hogy több sprite számot rendelünk egy minta-blokkhoz. Ezzel elérhetjük, hogy több sprite-nak ugyanaz legyen az alakja.
Az sz paraméterrel a sprite színét határozhatjuk meg az ismert 0–15 kódokkal.
A p paraméter a sprite háttér prioritást állítja be. Ebbõl következõen csak két értéke lehet: 0 vagy 1. Ha értéke 0, a sprite a képernyõn található ábrák elõtt helyezkedik el, míg ha értéke 1, mögöttük.
Az utolsó paraméter (a) szabja meg a sprite típusát.
a = 0 finomfelbontású,
a = 1 többszínû sprite-ot használunk.
Az utolsó két paraméter esetében az egynél nagyobb érték után a számítógép automatikusan 0-át állít be. Töltsük be a mintaprogramot és mielõtt elindítjuk, listázzuk ki. Erre azért van szükség, hogy a sprite megjelenésekor a képernyõ ne legyen üres. (Természetesen egyéb szöveg is megfelel.)

P. 92

```
1 REM ****************
2 REM * M O B  S E T *
3 REM ****************
100 REM
110 COLOUR 0,0:REM A KERET ES A HATTER FEKETE
120 DESIGN 0,13*64:REM MOB A 13.BLOKKBAN
130 @ ...........BB...........
140 @ ..........BBBB..........
150 @ ..........BBBB..........
160 @ ...........BB...........
170 @ .....BB....BB....BB.....
180 @ ......BB...BB...BB......
190 @ .......BB..BB..BB.......
200 @ ........BB.BB.BB........
210 @ ..BB.....BBBBBB......BB.
220 @ .BBBBBBBBBBBBBBBBBBBBBB
230 @ .BBBBBBBBBBBBBBBBBBBBBB
240 @ ..BB.....BBBBBB......BB.
250 @ ........BB.BB.BB........
260 @ .......BB..BB..BB.......
270 @ ......BB...BB...BB......
280 @ .....BB....BB....BB.....
290 @ ...........BB...........
300 @ ..........BBBB..........
310 @ ..........BBBB..........
320 @ ...........BB...........
330 @ ........................
340 REM
350 MMOB 0,100,100,100,100,3,0:REM A MOB MEGJELENIK A KEPERNYON
360 FOR X=0 TO 15
370 MOB SET 0,13,X,0,0:REM JELLEMZOK MEGHATAROZASA ES SZINVALTAS
380 PAUSE 2
390 NEXT X
400 COLOUR 8,8
410 REM
420 FOR X=1 TO 6
430 MOB SET 0,13,7,1,0:REM MOB A KARAKTEREK MOGOTT
440 PAUSE 2
450 MOB SET 0,13,7,0,0:REM MOB A KARAKTEREK ELOTT
460 PAUSE 2
470 NEXT
```

Ebben a programban is találkozunk az MMOB-utasítással, amely a sprite tényleges megjelenítését szolgálja. A későbbiekben részletesen is fogjuk ismertetni.

14.1.4 CMOB – MULTICOLOR sprite színei

Formátum: CMOB SZ1,SZ3

Paraméterek: SZ1 – a többszínű sprite 1. színe (0–15) (B betű)
SZ3 – a többszínű sprite 3. színe (0–15) (D betű)

Funkció: a többszínű sprite mellékszíneinek kiválasztása

Példa: CMOB 2,7

A CMOB utasítás a MOB SET kiegészítő utasítása, ami kizárólag a többszínű sprite-okra vonatkozik. A többszínű sprite-ok három színnel dolgoznak, ellentétben a finomfelbontású sprite-okkal. A két mellékszín megadását szolgálja ez az utasítás. Az SZ1 paraméterrel a sprite mintasorában B betűvel jelölt pontok, az SZ3 paraméterrel pedig a D betűvel jelölt pontok színezhetők ki. A sprite fennmaradó (2.) színét, azaz a C-vel jelölt pontokat a MOB SET utasítás definiálja. Fontos még tudnunk, hogy ezek a paraméterek, színösszeállítások egyszerre érvényesek mind a nyolc sprite-ra.

Egyedül a C betűvel jelölt pontok kaphatnak sprite-onként eltérő színeket.

Az utasítást az előző példákban már alkalmaztuk.

Ismétlésként nézzük át még egyszer ezeket a programokat.

14.1.5 Ellenőrző kérdések

Lássunk végül néhány tesztfeladatot, melyek remélhetőleg nem okoznak nehézséget.

1. Melyik utasítás írásmódja helyes?

   a) DESIGN 0
   b) MOB SET 1,15,3,0,0
   c) DESIGN 0,832
   d) CMOB 2,3

2. Mit kell figyelembe venni a DESIGN utasításnál?

   a) A DESIGN csak a sprite ábrázolásához használható.
   b) A 13-as blokk más memóriatartományhoz tartozik a grafikus és szöveges üzemmódban.
   c) Az első paraméter a sprite nagyságát határozza meg.
   d) Az utasítás és az első mintasor között nem szerepelhet más BASIC-sor.
   e) Nem használható többszínű sprite-ok esetén.

3. Mit határoz meg a MOB SET utasítás első két paramétere?

   a) Rögzítik a sprite típusát (finomfelbontású vagy többszínű) és meghatározzák a prioritást.
   b) A megnevezett blokkból származó definíciót az utasítás hozzárendeli a megadott számhoz.
   c) Az első paraméter a képernyőn ábrázolja a sprite-ot.
   d) A második paraméter megadja a sprite alakját.

14.2 A sprite-ok vezérlése

14.2.1 MMOB – megjelenítés és mozgatás

Formátum: MMOB n,x1,y1,x2,y2,m,v
Paraméterek: n – a mozgatni kívánt sprite száma (0–7)
x1 – a kezdőpont x-koordinátája (0–511)
y1 – a kezdőpont y-koordinátája (0–255)
x2 – a végpont x-koordinátája (0–511)
y2 – a végpont y-koordinátája (0–255)
m – a sprite mérete (0–3)
v – a sprite sebessége a két végpont között (0–255)
Funkció: egy sprite megjelenítése és mozgatása a képernyőn
Példa: MMOB 1,100,50,200,160,0,10

Eddigi ismereteink alapján már sokat tudunk a sprite-okról, de bosszantó volt, hogy nem láthattuk a képernyőn munkánk eredményét. Nos, a MMOB ezt a hiányosságot küszöböli ki. Segítségével nemcsak kijelezhetjük, hanem keresztbe-hosszába mozgathatjuk, pörgethetjük... stb. a figuráinkat. Alapfunkciója a sprite megjelenítése a képernyőn.
Kijelölhetjük azt a pontot, ahol a sprite-nak először fel kell tűnnie (x1, y1 pontpár) és azt a pontot is, amelyik felé haladnia kell (x2, y2 pontpár) közvetlenül a megjelenés után.
A koordinátákhoz néhány megjegyzést kell fűzni.
Ezek a koordináták nem azonosak az eddigi, a grafikusképernyőnél megszokott koordinátákkal. A MMOB utasításhoz megadott koordináták a sprite bal alsó pontjára vonatkoznak. Ha tehát a sprite-unkat a képernyő bal felső sarkába akarjuk elhelyezni, akkor az x=20, y=30 koordinátákat kell megadnunk. Ez a koordináta-transzformáció az egész képernyőre érvényes.
Azt is megtehetjük, hogy képernyőn kívüli koordinátaértékeket adunk meg a sprite számára. Az ütközésekre ebben az esetben is figyelni kell. Ha a sprite-ot nem akarjuk mozgatni, akkor egyszerűen adjunk azonos értékeket a két pontpárnak. Ekkor a kezdő- és végpont egybeesik, és a sprite mozdulatlan marad. A sprite a kezdőponttól (x1, y1) a végpontig (x2, y2) halad a képernyőn lévő alakzatokon, szövegen keresztül. Mozgási sebességét az utolsó paraméter (v) határozza meg: 0 és 255 között. Minél kisebb a v értéke, annál gyorsabb lesz a sprite. A v=255 esetén a sprite csigaként vánszorog, míg v=0 értéknél vadászrepülőként süvít át a képernyőn.
De álljunk csak meg egy pillanatra. Valamit elfelejtettünk:
Az m paraméterrel módunk van a sprite méretének meghatározására is. Mint tudjuk, minden sprite nagyítható mindkét irányban.
Az m nagyítási lehetőségei a következők:

m	Nagyítás	Nagyítási tényező	Pontmátrix
0	nincs	1×1	24×21
1	x-irányú	2×1	48×21
2	y-irányú	1×2	24×42
3	x,y-irányú	2×2	48×42

Pontmátrix alatt itt a képernyőn megjelenő mátrixot értjük. (A többszínű sprite-ok is ilyen méretűek), tehát ezt a téglalapot, amelyet egy sprite maximálisan képes lefedni.
Az előbb leírtak jobb megértése érdekében lássuk a következő példát:

P. 93

```
1 REM ***********
2 REM * M M O B *
3 REM ***********
100 REM
110 COLOUR 0,0:REM A KERET ES A HATTER FEKETE
120 DESIGN 0,13*64:REM MOB A 13.BLOKKBAN
130 @ ........................
140 @ ........................
150 @ ........................
160 @ ........................
170 @ ........................
180 @ ........................
190 @ ........................
200 @ ...BB.....................
210 @ ...BBB....................
220 @ ...BBBB...................
230 @ ...BBBBB.......BBBB......
240 @ ...BBBBBB.....BBB.BB.....
250 @ ...BBBBBBBBBBBBBBBBB...
260 @ .BBBBB.BB.BB.BB.BB.BBBBB
270 @ ...BBBBBBBBBBBBBBBBB...
280 @ ........................
290 @ ........................
300 @ ........................
310 @ ........................
320 @ ........................
330 @ ........................
340 REM
350 MOB SET 0,13,7,0,0:REM 0.MOB=SARGA
360 MOB SET 1,13,2,0,0:REM 1.MOB=PIROS
370 REM
380 REM ********
390 REM MOZGATAS
400 REM ********
410 REM
420 MMOB 0,0,0,200,200,1,50:REM NAGYITASX IRANYBAN
430 MMOB 1,0,100,220,220,0,70:REM NORMAL
440 PAUSE 5
450 COLOUR 9,9:REM A KERET ES A HATTER BARNA
```

Példaprogramunkban két azonos alakú, de különböző színű, nagyságú és sebességű sprite mozog a képernyőn. Látjuk, hogy a 0-ás és 1-es sprite-hoz a 350., illetve a 360. sorban rendeltük hozzá ugyanazt a blokkot.

14.2.2 MOB OFF – a sprite kikapcsolása

Formátum: MOB OFF n
Paraméterek: n – a kikapcsolandó sprite száma (0–7)
Funkció: a sprite kikapcsolása
Példa: MOB OFF 1

Miután megismerkedtünk azzal, hogy miképpen lehet egy sprite-ot a képernyőn megjeleníteni, meg kell tanulnunk természetesen a sprite kikapcsolásának módját is. Ezt a feladatot a MOB OFF utasítás látja el, melyhez egyetlen paraméterként a kikapcsolni kívánt sprite számát kell megadni. A MOB OFF utasítás törli a sprite és a blokk közötti összes hozzárendelést úgy, hogy a blokk újra felhasználható lesz. Ez azt jelenti, hogy ha a sprite-ot újra be akarjuk kapcsolni, vagyis a képernyőn megjeleníteni, a megfelelő sprite-számot a MOB SET utasítással hozzá kell rendelni valamelyik blokkhoz.
Többszínű sprite-ok esetén a két kiegészítő szín érvényben marad, a sprite újra a MMOB utasítással jeleníthető meg. Ezzel az utasítással válik lehetővé a találatot kapott űrhajók

eltüntetése a képernyőről, vagy olyan mozgó alakzat ábrázolása, amely tulajdonságait folyamatosan változtatja. (Lásd a példaprogramot.)
A MOB OFF utasítás természetesen helyettesíthető olyan módon, hogy a szóban forgó sprite-ot MMOB-utasítással kivisszük a képernyőről és ezáltal eltüntetjük. E módszer előnye, hogy a sprite összes paramétere érvényben marad, azaz bármikor és bárhol megjeleníthető anélkül, hogy ehhez a MOB SET utasítást ki kellene adni.
Ha a sprite-ot csak módosítani akarjuk, akkor nem feltétlenül szükséges a MOB SET előtt a MOB OFF utasítást is kiadni.

P. 94

```
1 REM ****************
2 REM * M O B  O F F *
3 REM ****************
100 REM
110 COLOUR 0,0:REM KERET ES A HATTER FEKETE
120 DESIGN 0,13*64:REM MOB A 13.BLOKKBAN
130 @ ........................
140 @ ........................
150 @ .........B..B...........
160 @ .........B..B...........
170 @ .........B..B...........
180 @ .........B..B...........
190 @ .........BBBB...........
200 @ ........BBBBBB..........
210 @ ........BB....BB........
220 @ ...B...BB...BB.BB...B...
230 @ ...B..BB..B.BB..BB..B...
240 @ ...BBBBBBBBBBBBBBBBB...
250 @ ..BBBBB.B.B.B.B.B.BBBB..
260 @ .BBBBB.B.B.B.B.B.BBBBBB.
270 @ BBBBBBBBBBBBBBBBBBBBBBBB
280 @ BB.....BBBBBBBBBB.....BB
290 @ BB.....BB......BB.....BB
300 @ ......BB........BB......
310 @ ....BBBB........BBBB....
320 @ ....BBBB........BBBB....
330 @ ........................
340 REM
350 DESIGN 0,14*64:REM MOB A 14.BLOKKBAN
360 @ ........................
370 @ ........................
380 @ ......BB........BB......
390 @ ........BB......BB......
400 @ .........BB....BB.......
410 @ ..........BB..BB........
420 @ ...........BBBB.........
430 @ .........BBBBBB.........
440 @ ........BB....BB........
450 @ .......BB...BB.BB.......
460 @ ......BB..B.BB..BB......
470 @ ...BBBBBBBBBBBBBBBBB...
480 @ ..BBBB.B.B.B.B.B.BBBBB..
490 @ .BBBBBB.B.B.B.B.BBBBBBB.
500 @ BBBBBB3BBBBBBBBBBBBBBBB
510 @ .......BBBBBBBBBB.......
520 @ .......BB..BB..BB........
530 @ ......BB........BB......
540 @ ....BBBB........BBBB....
550 @ ....BBBB........BBBB....
560 @ ........................
```

```
570 REM
580 REM ***********
590 REM VILLOGTATAS
600 REM ***********
610 REM
620 FOR X=1 TO 10
630 REM ----------
640 MOB SET 0,13,6,0,0
650 MMOB 0,200,100,200,100,3,0:REM AZ EGYIK MOB BEKAPCSOLASA
660 FOR Y=1 TO 70:NEXT Y
670 MOB OFF 0:REM A MOB KIKAPCSOLASA
680 REM
690 MOB SET 0,14,6,0,0
700 MMOB 0,200,100,200,100,3,0:REM A MASIK MOB BEKAPCSOLASA
710 PAUSE 1
720 MOB OFF 0:REM A MOB KIKAPCSOLASA
730 REM
740 NEXT X
750 COLOUR 9,9
```

Példánkban két hasonló sprite definíciót váltogatva villogást, ill. mozgást szimuláltunk. A 670. és 720. sor mindig kikapcsolja az előző sprite-ot, így a fentiek szerint el is hagyható.
Ezzel a módszerrel egyhelyben mozgó figurákat is létrehozhatunk. Ha ezeket a figurákat több sprite-ból állítjuk össze, akkor rendkívül látványos és mozgalmas dolgokat alkothatunk.
Mielőtt önálló programok írásához fognánk, tanulmányozzuk még a következő utasítást is. Ezután már semmi akadálya, hogy csodákat műveljünk a képernyőn.

14.2.3 RLOCMOB – a sprite mozgatása

Formátum: RLOCMOB n,x2,y2,m,v
Paraméterek: n – a mozgatandó sprite száma (0–7)
x2 – a célpont x koordinátája
y2 – a célpont y koordinátája
m – a sprite mérete (0–3)
v – a sprite sebessége (0–255)
Funkció: sprite ábrázolása vagy áthelyezése a képernyőn
Példa: RLOCMOB 1,200,160,0,10

Szintén nagyon hasznos utasítás, amely alkalmas arra, hogy az MMOB utasítással megjelenített sprite-ot egy másik pontba áthelyezzük. Itt nincs szükség a kezdőpont megadására, egyébként ugyanúgy működik, mint a MMOB (ld. 14.2.1 pontot).
A RLOCMOB utasítással egy sprite-ot be is lehet kapcsolni. Paraméterei azonosak az előzőekben leírtakkal. Ha nincs rá feltétlenül szükség, azt javasoljuk, hogy ne használja ezt az utasítást a sprite lekapcsolására, mert ez ilyenkor a bizonytalan kiindulási pont miatt átrepül a képernyőn, hogy az előírt célpontba jusson. Ezzel az utasítással most már nagyon egyszerűen megvalósíthatjuk akár több sprite egyidejűleg vezérelt mozgatását. A SIMON's BASIC kézikönyv erre nagyon szép példát mutat. Ott a példához minden magyarázat megtalálható, úgyhogy ezzel nem is húzzuk az időt, hanem belevágunk a közepébe, vagy ahogy a latin mondja: "in medias res".

```
1 REM ******************
2 REM * R L O C M O B *
3 REM ******************
100 REM
110 COLOUR 0,0:REM A KERET ES A HATTER FEKETE
120 DESIGN 0,13*64:REM MOB A 13.BLOKKBAN
130 @ ..............BBB.........
140 @ ..BBB.......BB.BB........
150 @ ...BB..B...BBBBBBBB......
160 @ .BBBBBB....BBBBBBB..BB..
170 @ ...BB..B....BB..BBBBBB..
180 @ ..BBB.........BBB........
190 @ ..............BBB.........
200 @ ..............BBB.........
210 @ ..............BBBBB.......
220 @ ..............BBB.BB......
230 @ ..............BBBB..BB.....
240 @ ...........BBBBB...BB.....
250 @ ...........B.BBB.........
260 @ .........B..BBBB.........
270 @ ........B..BB.BBB........
280 @ .......B...BB..BBB.......
290 @ ......B.....BB....BB.....
300 @ ......BB...BB....BB......
310 @ .BBBBBBB...BB....BBBB...
320 @ B.BBBB.....BB...........
330 @ ..B..B.....BBBB.........
340 REM
350 DESIGN 0,14*64:REM MOB A 14.BLOKKBAN
360 @ ..............BBB....B.B.
370 @ ............BB.BB....B..
380 @ ..BBB.B....BBBBBBBB.....
390 @ .BBBBB.....BBBBBBB..BB..
400 @ ..BBB.B.....BBBBBBBBBB..
410 @ ..............BBB.........
420 @ ..............BBB.........
430 @ ..............BBB.........
440 @ ..............BBB.........
450 @ ..............BBBB........
460 @ ..............BBBBB.......
470 @ ..............BBB.BB......
480 @ ............BBBB..........
490 @ ...........B.BBBB........
500 @ .........B..BB.BB........
510 @ .........B...BB.BB.......
520 @ .......B..BBB..BB........
530 @ ......BB.BB....BB........
540 @ .BBBBBBB.BBB...BB........
550 @ B.BBBB.....BB..BB.......
560 @ ..B..B.........BBBB.....
570 PRINT CHR$(30)CHR$(147):REM ZOLD+KEPERNYOTORLES
580 PRINT AT(0,9)DUP(CHR$(18)+" ",40):REM PAZSIT RAJZOLASA
590 REM
600 REM *****
610 REM FUTAS
620 REM *****
630 REM
640 MMOB 0,0,100,0,100,1,0:REM A MOB POZICIONALASA
650 G=10:REM SEBESSEG MEGHATAROZASA
660 F=1 :REM KEZDOSZIN
670 FOR X=1 TO 300 STEP G
680 F=F+0.3:REM SZINVALTAS
690 IF F=16 THEN F=1
```

```
700 REM
710 MOB SET 0,13,F,0,0
720 RLOCMOB 0,X,101,1,0:REM AZ ELSO MOB MOZGATASA
730 FOR Y=1 TO 20:NEXT Y
740 MOB OFF 0:REM A MOB KIKAPCSOLASA
750 REM
760 MOB SET 0,14,F,0,0
770 RLOCMOB 0,X+G/2,101,1,0:REM A MASIK MOB MOZGATASA
780 FOR Y=1 TO 20:NEXT Y
790 MOB OFF 0:REM A MOB KIKAPCSOLASA
800 REM
810 NEXT X
820 COLOUR 9,9
```

Példaprogramunk azt demonstrálta, hogyan dolgozzunk egyszerre több sprite-definícióval. Mint már az előző fejezetben említettük a 740. és 790. sorok itt is elhagyhatók, csak a jobb érthetőség kedvéért használjuk. A 650. sorban a figura sebességét változtathatjuk. Próbáljunk ki minden lehetséges esetet. Mozgassunk az emberke mellett egy méhrajt, vagy helyezzünk el az út mentén fákat.
Vezérelhetjük a figurát a billentyűzetről, vagy esetleg joystick-kal is!

14.2.4 DETECT – ütközés vizsgálat előkészítése

Formátum: DETECT ü
Paraméterek: ü – az ütközés módja
ü = 0 sprite – sprite
ü = 1 sprite – háttér
Funkció: az ütközés lekérdezés előkészítése
Példa: DETECT 1
Ez az utasítás az ütközés lekérdezést készíti elő. Ez így első hallásra elég bonyolultan hangzik, és valóban nem is olyan egyszerű. Mint tudjuk a VIC chip két regisztere az ütközéseket regisztrálja. Az egyik a sprite–sprite, a másik a sprite–háttér ütközésekért felelős. Ütközés esetén a regiszterekbe egy meghatározott érték kerül; ami azonban mindaddig a regiszterben marad, amíg onnan el nem távolítjuk. Tehát ha a regisztert egy ütközést követően nem töröljük, úgy tűnik, mintha a korábbi eredmény még mindig aktuális volna, vagyis minden pillanatban ütközés lépne fel.
Emiatt vagyunk kénytelenek,használni a DETECT utasítást, ami törli a megfelelő regisztert, előkészítve így egy újabb lekérdezést. Következésképpen ezt az utasítást minden lekérdezés után, ill. előtt be kell adni. Ütközésnél nem elég az utasítást egyszer alkalmazni, hanem a program folyamán (még akkor is, ha a program befejeződött; ld. példa), a következő lekérdezésig többször, de legalább kétszer le kell hívni azért, hogy a számítógép nehogy véletlenül egy nem létező ütközést regisztráljon. Ez a jelenség hardver jellemzőkre vezethető vissza. A könyvünkben terjedelmi okok miatt ezzel nem foglalkozunk.
Látjuk, hogy a DETECT nem alkalmazható olyan egyszerűen, mint ahogy első látásra gondoltuk, de némi gyakorlással ez is elsajátítható.
Az utasítás egyetlen paramétere azt mondja meg, hogy a sprite–sprite vagy a sprite–háttér regisztert kell előkészíteni az ütközést (ld. bevezetés).
További részleteket a következő utasításból tudhatunk meg.

14.2.5 CHECK – az ütközés lekérdezése

Formátum: CHECK(n1, n2), vagy CHECK(n)
Paraméterek: n1 – azon első sprite száma, melynek ütközését egy másik sprite-tal meg kell vizsgálni (0–7)

n2 – azon második sprite száma, melynek ütközését egy másik sprite-tal meg kell vizsgálni (0–7)
n – azon sprite száma, melynek ütközését a háttéralakzattal meg kell vizsgálni (0–7)

Funkció: ütközés lekérdezése
Példa: CHECK(0,7), vagy CHECK(1)

A SIMON's BASIC utolsó sprite utasításához értünk. A CHECK függvény jellegű, mint a SIN, SQR, LIN stb. Hasonlóképpen kell alkalmazni, mint a TEST utasítást, amely azt vizsgálja, hogy valamely grafika meghatározott helyén egy pont be van-e kapcsolva (ld. ott).
Feladata annak lekérdezése, hogy az adott pillanatban végbement-e sprite–sprite vagy sprite–háttér ütközés. Bevezetése az előző pont DETECT utasításával történik. Ha végrehajtásának időpontjában nem történt ütközés, értéke 1 lesz. Ez a paraméter változóként kezelhető, vele számításokat végezhetünk.
Például:

```
A = 2*CHECK(0)+1
```

Ellenkező esetben (vagyis ütközésnél) értéke 0. A megelőző DETECT utasítás írja elő, hogy a sprite–sprite vagy a sprite–háttér ütközést figyeli-e. A sprite–sprite ütközés figyelése a következő szintaktikával lehetséges:

```
CHECK(n1, n2)
```

Ezzel a két paraméterrel adhatjuk meg, hogy melyik két sprite közötti ütközést vizsgáljuk. Más esetleges ütközéseket az utasítás nem vesz figyelembe, és értéke csak akkor lesz 0, ha a megjelölt két objektum ütközött. Ha csak arra vagyunk kíváncsiak, hogy a kiválasztott sprite összeütközött-e egy tetszőleges másikkal, akkor a második paramétert tegyük egyenlővé az elsővel (n1 = n2). Ennek hatására minden olyan ütközést regisztrálni fog, amelyben ez a sprite részt vesz.
Hasonló a helyzet a második esetben is, ha a vizsgálatot a DETECT 1-gyel előkészítette. Itt a sziktaktika a következőképpen néz ki:

```
CHECK(n)
```

ahol n azt a sprite-ot jelöli, amelynek a háttérrel való ütközését éppen vizsgáljuk. (A kézikönyvben található információk ezzel kapcsolatban természetesen rosszak.)
A szintaktikával kapcsolatban még fontos megjegyezni, hogy az utasításszó és az első zárójel szorosan összetartoznak (mint az AT-utasításnál), ezért nem választhatók el szóközzel.
Tehát:

```
CHECK(2,2)
```
és nem → CHECK (2,2)

Végül szórakozásképpen lássunk egy kis példát:

```
1 REM **************
2 REM * C H E C K *
3 REM **************
100 REM
110 COLOUR 0,0:REM A KERET ES A HATTER FEKETE
120 DESIGN 1,13*64:REM MOB A 13.BLOKKBAN
130 @ ..............
140 @ ..............
150 @ ..............
160 @ ..............
170 @ ..............
180 @ .C............
190 @ CCC...........
200 @ CCCBB.........
210 @ CCBDBBDDDB...
220 @ CCCBBDBBDBB
230 @ CCBD.... ....
240 @ CCC...........
250 @ .C............
260 @ ..............
270 @ ..............
280 @ ..............
290 @ ..............
300 @ ..............
310 @ ..............
320 @ ..............
330 @ ..............
340 REM
350 DESIGN 0,14*64:REM MOB A 14.BLOKKBAN
360 @ ........................
370 @ ........................
380 @ ........................
390 @ ........................
400 @ ........................
410 @ ........BBB.............
420 @ ...........B............
430 @ ............B...........
440 @ .............B..........
450 @ ............BBBBB.......
460 @ .......BB...BBBBBB.....
470 @ ......BBB...BBB..BB.....
480 @ ....BBBBB..BBB..BBB.....
490 @ ..BBBBBBBBBBB..BBBB....
500 @ ..BBBBBBBBBBBBBBBBB...
510 @ ..BBBBBBBBBBBBBBBBB....
520 @ ...BB..BBBBBBBB..BB.....
530 @ BBBB.BB.BBBBBB.BB.BB....
540 @ ...BB.........BB........
550 @ ........................
560 @ ........................
570 REM
580 MMOB 0,0,100,0,100,0,0:REM AZ ELSO MOB POZICIONALASA
590 MMOB 1,300,100,300,100,1,0:REM A MASIK MOB POZICIONALASA
600 DETECT 0:REM UTKOZESI REGISZTER TORLESE
610 REM
620 REM *********
630 REM MOZGATAS
640 REM *********
650 REM
660 FOR X=1 TO 300 STEP 4
670 CMOB 2,7:REM SZINEK A MULTICOLOR MOB-HOZ
680 REM
690 MOB SET 0,14,9,0,0
```

```
700 RLOCMOB 0,X,100,0,0:REM ELSO MOB MOZGATASA
710 REM
720 CMOB 7,2:REM SZINEK A MULTICOLOR MOB-HOZ
730 MOB SET 1,13,11,0,1
740 RLOCMOB 1,320-X,40*SIN(X/30+1.3)+100,1,0:REM MASODIK MOB MOZGATASA
750 REM
760 DETECT 0:REM AZ UTKOZESI REGISZTER LEKERDEZESENEK ELOKESZITESE
770 IF CHECK(0,1)=0 THEN GOTO 790
780 NEXT X
790 FOR Z=1 TO 15
800 COLOUR Z,Z-1
810 FOR S=1 TO 30:NEXT S:REM KESLELTETES
820 NEXT Z
830 COLOUR 13,14
```

14.2.6 Ellenőrző kérdések

Ebben a bizonyára nagyon érdekes fejezetben a SIMON's BASIC sprite-okat kezelő utasításairól volt szó.
A következő néhány kérdés alapján eldönthető, mennyire sikerült az olvasottakat elsajátítani.

1. Mire kell ügyelni a DETECT és a CHECK utasítások alkalmazásánál?

   a) Ütközés után a DETECT-utasítást egyszer kell beadni az ütközési regiszter törléséhez.
   b) A DETECT paraméter meghatározza a CHECK szintaktikáját is.
   c) A CHECK(1,1) utasítás nem engedélyezett.
   d) A CHECK(9) utasítás nem engedélyezett.
   e) Ütközés esetén a CHECK értéke 1 lesz.

2. Melyik szintaktika helyes?

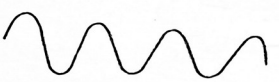
   a) MMOB 1,30,40,50,60,2,70
   b) MOBB OFF
   c) RLOCMOB 3,70,3
   d) CHECK (2,3)

15. FEJEZET
A ZENE

Elérkeztünk ahhoz a pillanathoz, hogy hamarosan belépjünk a hangok különös világába. Engedjük át magunkat a hangkeltés végtelen harmóniájának és tapasztalni fogjuk, hogy ezidáig néma barátunk, a Commodore 64-es ezen a téren is micsoda nagyszerű dolgokra képes.
Először is tudnunk kell azt, hogy számítógépünk egy jól felszerelt, háromszólamú szintetizátor. A rádióból már ismert szintetizátor-zene hasonló módon jön létre. Némi ügyességgel és fantáziával különféle zajokat, zörejeket és nagyon szép dallamokat csalogathatunk elő a segítségével.
Természetesen a profi szintetizátor programokkal a SIMON's BASIC nem versenyezhet, mivel azok a számítógép képességeit teljes mértékben ezirányban használják ki, de néhány funkciója mégis figyelmet érdemel.

15.1 A hardver előfeltételek

A hangkeltő utasítások helyes alkalmazása érdekében nagyvonalakban meg kell ismerkedni a különböző hangok szerkezetével.
Ugyan léteznek bonyolult és összetett zajok, valamint hangminőségek, de legelőször azzal kell tisztában lenni, hogy milyen szerkezeti elemei vannak egy hangnak.
Tekintsük először a hangmagasságot. Milyen tényező határozza meg, hogy milyen magas egy hang? A fizikai tanulmányainkból talán még emlékszünk, hogy a hang olyan gyors levegőmozgás, melyet a fülünkkel érzékelhetünk. A szabályos szinuszgörbe az elképzelhető legegyszerűbb hullám:

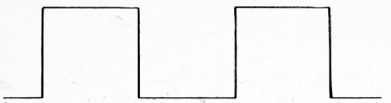


Nos, a hangmagasság kialakulását az eredményezi, hogy a hullámhegyek és -völgyek milyen "sűrűek", vagyis hogy gyorsabb vagy lassúbb ütemben jutnak-e el hozzánk. A másodpercenként a fülünkhöz érkező hullámok számát frekvenciának nevezzük, melyet hangmagasságként érzékelünk. A másik igen fontos jellemző: a hang erőssége. Az erősség a hullámhegyek és -völgyek legmagasabb és legmélyebb pontjai különbségének, vagyis az amplitúdónak a függvénye. A hangminőséget harmadikként meghatározó tényező a hullámforma, melyet a hanghullám alakja határoz meg. Ebben az esetben is számtalan variáció létezik. A Commodore 64-es ezekből négyet ismer:

1) Háromszöghullám

2) Fűrészfoghullám

3) Négyszöghullám

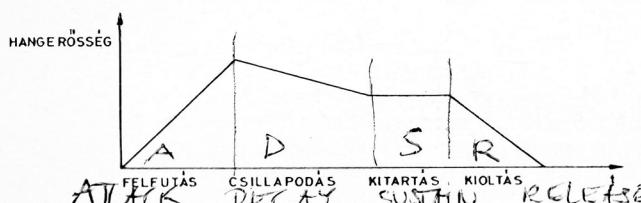


4) Zaj

Ezek a hullámformák az ÉS logikai relációval összekapcsolhatók.
A hangok további lényeges tulajdonsága, hogy felharmonikussal rendelkeznek. A szinuszhang kivételével nincs olyan hang, mely pontosan követné az előbbiekben vázolt hullámformákat. Ugyanis az egyes hullámokon belül is olyan kisebb-nagyobb rezgések lépnek fel, amelyek olyanná teszik, mintha egy reszketõ kéz rajzolta volna. Ezeket a hullámon belüli frekvencia-eltéréseket is halljuk, sõt ezek határozzák meg a hang karakterét, ami lehetővé teszi, hogy megkülönböztessük egymástól pl. a hegedũ és a fuvola hangját.
A Commodore által kínált hullámformák különbözõ felharmonikusokkal rendelkeznek, ezáltal különbség tehetõ közöttük. Lehetõségünk van továbbá arra is, hogy ezeket a felhangokat vagy levágjuk, vagy elhagyjuk. Ezeket a mũveleteket szũrõkkel valósíthatjuk meg. Pl. a rádió hangszínszabályozója egy ilyen típusú szũrõ. Ezekkel szũrhetjük ki a felesleges magas hangokat, és ezáltal a zene teltebb hangzásúvá válik.
Ezeket a lehetõségeket a számítógépen is megtaláljuk.
Megadhatjuk a hang frekvenciáját (hangmagasság), hullámformáját, valamint a kiszũrendõ frekvenciákat.
A hangkeltésnek még hiányzik egy összetevője: a hang idõbeni lefutása.
Nincs olyan hang, ami megszólal és azonnal el is hallgat. Vannak lassan erõsödõ, majd lassan lecsengõ hangok és vannak olyanok, mint pl. a lövések, amelyek hihetetlen nagy hangerővel szólalnak meg, majd kissé visszhangzanak. Számítógépen lehetõség van az ilyen hangok programozására. A hangok lefutásában négy fázist különböztetünk meg:

1) Felfutás
2) Csillapodás
3) Kitartás
4) Kioltás

Grafikusan ábrázolva:



A kifejezések angol megfelelõi: ATTACK, DECAY, SUSTAIN, RELEASE.
A hang keletkezésének folyamata a következõ: Elõször is megszólal a hang. Megadhatjuk a teljes hangerõ nagyságát és elérési idejét (ATTACK), valamint azt az idõt, ami a SUSTAIN fázis eléréséig szükséges. A hang ilyen hangerővel szól egészen a kikapcsolá-

sig. Itt kezdődik a lecsengési (RELEASE) fázis, melynek időtartama szintén programozható. A hang lefutásának jellemzőit (a hang burkológörbéjét) természetesen csak a zenemű elején kell megadni. Módosításáig a görbe állandó marad.
Összefoglalva: A számítógépen három, egymástól függetlenül és egyidőben működtethető szólam programozható, melynek mindegyikére külön-külön előírhatjuk a frekvenciát, a hullámformát és a burkológörbét. A hangerő és a szűrők azonban mindhárom szólamra érvényesek. A CBM64-es szintetizátorának a fentieken kívül egyéb funkciói is vannak, de ezeket nem tárgyaljuk, mert a SIMON's BASIC nem tudja kezelni őket. Részletesebb információkat az alkalmazói kézikönyvből és a Commodore 64-es zenei "képességeivel" foglalkozó szakkönyvekből szerezhetünk.
A következő fejezetekben a hangképzés egyes utasításaival ismerkedünk meg.

15.2 VOL – a hangerő beállítása

Formátum: VOL h
Paraméterek: h – hangerő (0–15)
Funkció: a megszólaló hang erősségének beállítása
Példa: VOL 15
A SIMON's BASIC öt hangutasítása közül az első a VOL– (a VOLume szóból), amellyel megadhatjuk a három hanggenerátor hangerejét. Ezt célszerű általában a zenemű elején megtenni, de a hang kibocsátása közben megváltoztathatjuk (ld. példa). Ezáltal speciális effektusokat is előidézhetünk. Sajnos a hardver nem teszi lehetővé, hogy a szólamok hangerejét külön-külön is változtassuk. Ha a VOL paraméterét 0-ra választjuk, mind a három generátor néma marad. A paraméter 15-ös értékénél kapjuk a maximális hangerőt.
Töltsük be a P. 97-es programot és próbáljuk ki:

P. 97

```
1 REM ***********
2 REM * 1.V O L *
3 REM ***********
100 REM
110 REM ****
120 REM HANG
130 REM ****
140 REM
150 VOL 15
160 ENVELOPE 1,1,8,10,10
170 WAVE 1,00100000
180 MUSIC 150,"□1C2■"
190 PLAY 2
200 REM
210 REM ********
220 REM EROSITES
230 REM ********
240 REM
250 FOR X=0 TO 15
260 VOL X:REM  HANGERO
270 FOR W=1 TO 80:NEXT W
280 NEXT X :REM A HANG EROSITESE
290 PAUSE 2
295 VOL 0:REM A HANG KIKAPCSOLASA
300 REM
310 REM *********
320 REM VISSZHANG
330 REM *********
340 REM
350 FOR X=15 TO 0 STEP -1
360 FOR Y=0 TO X
```

```
370 VOL Y:REM HANGERO
380 FOR W=X TO 35:NEXT W:REM VARAKOZAS
390 NEXT Y:REM A HANG EROSITESE
400 NEXT X:REM ELHALKULAS
```

A 160.–190. sorok tartalma még nem teljesen érthető. Egyelőre elég annyit tudni róluk, hogy ez az utasítássor egy hosszú hangot állít elő. Az egyes utasításokra a későbbiekben visszatérünk.
Egy dolgot azonban ismerni kell a program beviteléhez.
A 180-as sorban néhány érthetetlen karakter áll az idézőjelben. Ezek a program átvitelét segítik. A MUSIC utasításban szereplő nagybetűk vezérlő karakterek, melyek a képernyőn is megjelennek inverz formában, ha a SHIFT és C= billentyűk lenyomása után karakterkészletet váltunk:
Ezek a karakterek a következőket jelentik:

S: (SHIFT) , (CLR/HOME)
I: (f2)
J: (f4)
K: (f6)
L: (f8)
E: (f1)
F: (f3)
G: (f5)
H: (f7)

Ez a táblázat a könyv valamennyi zenei programjára érvényes. Ha elfelejtjük egy betű jelentését, ezen az oldalon mindig megtaláljuk.
A MUSIC utasítás minden kisbetűje egy egyszerű karaktert jelent. A program 180-as sora tehát így is írható:

```
180 MUSIC 150,CHR$(147)+"1c2"CHR$(137)
```

15.3 WAVE – a hullámforma előírása

Formátum: WAVE sz, rrsdtrsg
Paraméterek: sz – annak a szólamnak a sorszáma, melyre a hullámformát megadjuk
rrsdtrsg – 0-ból és 1-ből álló 8 jegyű számsorozat, amely bizonyos funkciók be- (1), ill. kikapcsolását (0) jelentik
Funkció: a lejátszandó hang hullámformájának beállítása
Példa: WAVE 2, 10000101
A WAVE utasítás egy szólam hullámformájának beállítására és egyéb műveletek elvégzésére alkalmas.
Lássuk az egyes paraméterek jelentését:
Az elsővel a szólam kiválasztását végezzük.
A második ennél lényegesen bonyolultabb. Itt nyolc különböző funkciót kapcsolhatunk be és ki. Tehát nem kell mást tennünk, mint az alábbi ismertetett funkciók közül a használni kívántakra 1-est, míg a többire 0-t írni. Így végeredményben nyolc 1-es, ill. 0 adódik, melyet a fenti formátumban a rrsdtrsg paraméter jelöl. A számjegyek között szóköz nem szerepelhet!
Az utasítással közvetlenül a SID chip (hangprocesszor) néhány kitüntetett regiszterét programozzuk (a három szólamhoz rendre a 4., 11. és 18. regisztert). A bővebb információk a kézikönyvben találhatók.
A második paraméter nyolc értéke ezen regiszterek egyikének nyolc bitjét ábrázolja.
Az egyes bitek jelentése az alábbi:

Bit	Funkció
7	Zajok
6	Négyszöghullám
5	Fűrészfoghullám
4	Háromszöghullám
3	Teszt bit
2	Moduláció
1	Szinkronizáció
0	Gate bit

Ezek után pedig következzen a részletes magyarázat. (Vö. a CBM64-es kézikönyvével.)

0. bit – Gate bit

A hang megszólaltatására és megszüntetésére alkalmas. Ha értéke 1, akkor a többi bittel előállított hang az előre meghatározott burkológörbéje szerint szólal meg és addig marad SUSTAIN fázisban, amíg a bit újra 0 nem lesz. Amint ez bekövetkezik, a hang a RELEASE-nek megfelelően lecseng. Normál körülmények között a SIMON's BASIC ezt a bitet automatikusan vezérli. A WAVE utasításban azért kell mégis szerepelnie, mert csak így tudjuk a hang kitartása közben valamelyik paraméterét megváltoztatni. Egyébként a hang elhallgatna. Az utasítás a hang kikapcsolására is alkalmazható.

1. bit – Szinkronizáció

Egyidejűleg több oszcillátor megszólaltatása esetén, a hang jellege szerint, különböző ingadozásokat hallunk. Ezek úgy jönnek létre, hogy például az egyik szólam hullámvölgye a másik szólam hullámhegyével találkozik. Ekkor a hangok kioltják egymást, és egyik hang sem hallható. Előfordulhat az is, hogy két hullámhegy találkozik és a két hang felerősödik. Ez a két szélső helyzet folyamatosan változik, és ennek eredménye a hang ingadozása.
Ez a jelenség néha nagyon zavaró lehet. Ezt a problémát oldja meg a szinkronizáló bit. Bekapcsolásakor a két oszcillátor által gerjesztett hullámok azonos fázisban találkoznak. Az 1. bit a három szólamban eltérő feladatokat lát el.

Szólam	Szinkronizáció
1.	az 1. és 3. szólam között
2.	a 2. és 1. szólam között
3.	a 3. és 1. szólam között

2. bit – Gyűrűs moduláció

A SIMON's BASIC kézikönyvben minden információ megtalálható.

3. bit – Teszt bit

Ha a zajgenerátorral párhuzamosan egy másik hullámformát is választunk, előfordulhat, hogy ezzel blokkoljuk a zajgenerátort. Feloldása ezzel a bittel történik.

4. bit – Háromszöghullám

Ezzel a bittel a háromszöghullámot választhatjuk ki. Ez hasonlít a tiszta szinuszhullámhoz, ami köztudottan nem tartalmaz felharmonikusokat. Hangzása a fuvolához hasonló.

5. bit – Fűrészfoghullám
Sok, nem egyenes és egyenes felhanggal rendelkezik. Nagyon keményen cseng. Hangzása hasonló a hegedűhöz.

6. bit – Négyszöghullám
Ennél a hullámformánál lehetőség nyílik arra, hogy a völgyek és hegyek időtartamának viszonyát az ún. impulzusviszonyt változtassuk. Bár a SIMON's BASIC nem teszi lehetővé ennek változtatását, a következő POKE utasítások segítségével azonban a módosítás viszonylag egyszerűen elvégezhető:

POKE 54272−5+7*ST,LB:POKE 54272−4+7*ST,HB

ahol ST a szólamszám, HB és LB a beírandó érték alsó és felső byte-ja.

7. bit – Zajok
Bekapcsolja a megfelelő szólam zajgenerátorát. Nagyon sokoldalúan használható (pl.: tengermorajlás, lövések, sziszegés stb.).
Végül még egy megjegyzés a 4–7 bitekhez:
A gyakorlatban egyszerre több hullámformát is kiválaszthatunk, de a 3. bitnél elmondottakon kívül ekkor az eredmény nem az egyes összetevők eredője, hanem azok logikai ÉS kapcsolata lesz.

P. 98

```
1 REM ************
2 REM * 2. V O L *
3 REM ************
100 REM
110 REM ****
120 REM HANG
130 REM ****
140 REM
150 VOL 15
160 ENVELOPE 1,10,10,10,10
170 MUSIC 250,"▯1C2▩"
180 PLAY 2
190 REM
200 REM ************
210 REM HULLAMFORMA
220 REM ************
230 REM FURESZFOG
240 WAVE 1,00100001
250 PAUSE2
260 REM ZAJ
270 WAVE 1,10000001
280 PAUSE2
290 REM HAROMZOG/FURESZFOG
300 WAVE 1,00110001
310 PAUSE2
320 REM HAROMSZOG
330 WAVE 1,00010001
340 PAUSE2
350 REM HAROMSZOG/NEGYSZOG
360 WAVE 1,01010001
370 PAUSE2
380 REM NEGYSZOG/FURESZFOG
390 WAVE 1,01100001
400 PAUSE2
410 REM NEGYSZOG/HAROMSZOG/FURESZFOG
420 WAVE 1,01110001
430 PAUSE2
```

A 150–180-as sorok megértéséhez lapozzunk a VOL utasításnál ismertetett program magyarázatához.

15.4 ENVELOPE – a burkológörbe meghatározása

Formátum: ENVELOPE sz,a,d,s,r
Paraméterek: sz – az a szólam, amelyre meg akarjuk határozni a burkológörbét (1–3)
a – ATTACK – a felfutási fázis időtartama (0–15)
d – DECAY – a csillapodási fázis időtartama (0–15)
s – SUSTAIN – a kitartási fázis hangereje (0–15)
r – RELEASE – a lecsengési fázis időtartama (0–15)
Funkció: a burkológörbe meghatározása egy adott szólamra
Példa: ENVELOPE 3,9,6,10,15

Olyan utasításhoz értünk, amellyel alkotó módon határozhatjuk meg a hang burkológörbéjét.
A bevezetőből már ismerjük a burkológörbe fogalmát.
Ez a hang teljes lefutását jelenti a megszólalástól az elhallgatásig, az idő függvényében.
A burkológörbét szólamonként külön-külön meg lehet határozni. Az egyes hangfázisokhoz tartozó négy paraméterrel gyakorlatilag minden hangot meg lehet szólaltatni, csak tudni kell vele bánni.
A hangok keletkezésének elvéből kiindulva két módszert különböztetünk meg:

1) A paramétereket addig változtatjuk, amíg egy eddig ismeretlen, szép hanghoz nem jutunk. Nem rossz módszer, érdemes gyakran alkalmazni. Használata során megismerhető az egyes paraméterek konkrét hatása. Sikeresen azonban csak egyedi hangeffektusok előállítására alkalmazható.
2) Ha egy előre meghatározott hangot akarunk beprogramozni (pl. a fuvola hangját), akkor a kísérletezés helyett sokkal célravezetőbb, ha előre megtervezzük a hang lefutását és ennek segítségével választjuk ki a szükséges paramétereket. A két eljárás közül ez sokkal céltudatosabb megoldás. A fuvolahang esetén ez hosszú felfutási fázist (az ATTACK nagy lesz), a maximális hangerő észre nem vehető csökkenését (SUSTAIN és DECAY nagy), majd nagyon gyors kioltást jelent (RELEASE kicsi).

Példa:
Most csak egy rövid példaprogramot kínálunk. A 200-as sorban változtathatjuk a paramétereket, hogy megismerjük működésüket. A 280-as sorban a hullámformát is módosíthatjuk. Próbáljuk ki a zajt is, mert itt a burkológörbe nagyon jól látható.

P. 99

```
1 REM ********************
2 REM * E N V E L O P E *
3 REM ********************
100 REM
110 REM ************
120 REM BURKOLOGORBE
130 REM ************
140 REM
150 REM GYAKORLO-PELDA
160 REM VALTOZTASSUK A PARAMETEREKET !
170 REM        ST!   A!   D!   S!   R!
180 REM ----------------------------
190 REM        !    !    !    !    !
200 ENVELOPE   1,  10,  10,   5,   9
210 REM ----------------------------
```

```
220 REM
230 REM ****
240 REM HANG
250 REM ****
260 REM
265 VOL 15
270 REM FURESZFOG
280 WAVE 1,00100001
290 MUSIC 50,"■1C2■"
300 PLAY 2
310 PAUSE1:REM KIKAPCSOLAS
320 WAVE 1,00100000
```

A 290-es és 300-as sorokkal, de különösen a 290-es sorral kapcsolatban olvassuk el a VOL programhoz fűzött megjegyzéseket a 15.2 pontban. Itt ismét megnézhetjük a füzérek írásmódját.

15.5 MUSIC – a dallam előállítása

Formátum: MUSIC te,fü
Paraméterek: te – a zenemű tempója (0–255)
fü – a hangsorozatot rögzítő füzérváltozó vagy karakterlánc
Funkció: dallam definiálása
Példa: MUSIC 8,A$, vagy MUSIC 10,"...hangok..."

A MUSIC utasítással végre leírhatjuk a játszani kívánt dallamot. Ez a számítógép tárjába kerül, és a PLAY paranccsal játszhatjuk le. A te paraméterrel előírható, hogy a hangjegyek milyen sebességgel kövessék egymást, tehát ezzel a hangok ütemét határozzuk meg. (A zenében ehhez különböző tempók vehetők igénybe, pl.: allegró, andante, presto stb.) Az alapütem mindig 1/60 másodperc. Ennél alacsonyabb értékek nincsenek. A te = 0 esetén az egész hangjegy (1/1) 2/60 másodperc, míg a fél 1/60 másodperc hosszúságú és ugyanilyen hosszúak lesznek az 1/4, 1/8 stb. hangjegyek is. A te = 1 esetén az egész hang 5/60, azaz 1/12 másodpercig fog szólni.
Negyedhang csak pontatlanul szólaltatható meg, mert az 5 maradék nélkül nem osztható 4-gyel.
Az egész hangok időtartamára általánosan az alábbi képlet érvényes (másodpercben).

Z = te/12 vagy Z = 5*te/60

A képlet csak közelítő értékeket ad. (A te = 0 érték teljesen kilóg a sorból.)
A többi hangjegy értéke az eredmény tört része vagy többszöröse. Ha pontosan akar negyed, nyolcad stb. hangjegyeket lejátszani, a te értékét válassza 4-gyel, 8-cal stb. osztható számra.
Most pedig lássuk hogyan kell egy hangjegyet definiálni. Legjobb lesz, ha ebből a célból a kisbetűs-/nagybetűs karakterkészletre váltunk át a SHIFT és a C= billentyűk egyidejű lenyomásával. Amennyiben ez hatástalan lenne (ez a SIMON's BASIC-ben néha előfordul), adjuk be az alábbi parancsot:

PRINT CHR$ (9)

(ld. karakterkészlet fejezet). A kisbetűs karakterkészletben az egyes karakterek jobban megkülönböztethetők, továbbá megfelelnek a példaprogramban használt karaktereknek. A füzér legelején a szólamot kell megadni. Ez úgy történik, hogy idézőjel után lenyomjuk a SHIFT és a CLR/HOME gombokat. Ez az A karakterkészlet esetén egy inverz szívet, a B karakterkészlet esetén pedig egy inverz S-betűt jelent a képernyőn. Ezt követi a szólam száma (1–3). Ha új szólamra akarunk áttérni, akkor is ezt az utasítást alkalmazzuk. Ez akkor előnyös, ha pl. a 3. szólamra más hullámformát vagy burkológörbét adunk meg és zene közben akarjuk módosítani a hangszínt.
Többszólamú zenére azonban nincs lehetőség (leszámítva azt az esetet, amikor az előző szólam utolsó hangja áthallatszik a következő szólam hangjaiba).
Következő paraméterként a lejátszandó hangjegyeket (és nem hangjegy értékeket, mint a kézikönyv írja) kell megadni két értékkel.
Először szerepeljen a hangjegy neve, utána az oktáv száma. A hangok elnevezése a szokásos amerikai írásmódot követi:

C D E F G A H

(az utolsó betű H, és nem B, ahogy a kézikönyvben áll). Ha a hangot félhanggal magasabban akarjuk megszólaltatni, akkor a hangjegyet a SHIFT billentyűvel kell beírni, ami megfelel a hangjegy előtti #-nak. Ezután adjuk meg az oktávot (0–7), amelyben a hang megszólal. (Pl. a normál A-hang, 440 Hz frekvenciával a 4. oktávban van. Megszólaltatása "A 4" bevitelével lehetséges.)
A legmagasabb bevihető hang a 7. oktáv magas A-ja: "A7".
A hangra vonatkozó információkat a hangjegy időtartamával zárjuk, melyeket a funkcióbillentyűkkel hívunk le:

Billentyű	Tempó	Képernyő
f1	1/16	inverz nagy E
f3	1/8	inverz nagy F
f5	1/4	inverz nagy G
f7	1/2	inverz nagy H
f2	1/1	inverz nagy I
f4	2/1	inverz nagy J
f6	4/1	inverz nagy K
f8	8/1	inverz nagy L

Ezzel befejeződött egy hang definiálása. A többi hangra ugyanúgy kell eljárni. Az információk között nem lehet szóköz.

Ha egy 3/2 (1 1/2) értékű hangot akarunk megszólaltatni, akkor ezt több funkcióbillentyű nyomásával érhetjük el. A hangjegyértékek ilyenkor összegződnek. Esetünkben ez az f7 f7 f7 vagy az f2 f7 billentyűket jelenti. Normál körülmények között az egyes hangok kötve (legato-ban) szólalnak meg. Ha a következő hangot külön akarjuk indítani, akkor a SHIFT/CLR/HOME és a g billentyűket kell működtetni. Ugyanaz a helyzet, ha szólamváltáskor el akarjuk kerülni az előző szólam továbbhangzását. Ez kikapcsolja a hangot, és ha mögé egy vagy több hangjegyértéket teszünk a megadott hosszúsághoz, akkor még szünetet is előállíthatunk.
Az ismertetett módon nagyon könnyedén megszólaltathatunk egy egyszerű dalocskát, míg hosszabb darabokhoz (ld. példa) több füzért kell összekapcsolni.
Könnyen beiktathatunk ismétléseket is.
Az utasítás alkalmazását a PLAY utasításnál ismertetjük.

15.6 PLAY – a dallam lejátszása

Formátum: PLAY m
Paraméterek: m – játékmód
m = 1 várakozás a játék végére
m = 2 játék és számlálás
Funkció: a MUSIC utasítással rögzített dallam lejátszása
Példa: PLAY 1

Az m paramétertől függően a PLAY utasításnak két változata van.
Ha m = 1, akkor a program a zene lejátszásának végéig vár. Tehát a PLAY 1 utasítás után a többi utasítás csak a zene lejátszása után kerül végrehajtásra. Ilyenkor a program csak a RUN/STOP billentyűvel szakítható meg. Ezzel szemben, ha m = 2, a zene megkezdése után a program azonnal rátér a következő utasítás végrehajtására úgy, hogy a zene szól. Jól érzékelhető ez az alábbi programban. Ha a program még a zene befejezése előtt lefutott, akkor a zene is félbeszakad, és az utoljára megszólaló hang szól mindaddig, amíg a VOL 0, vagy a WAVE sz, 00000000 utasítással ki nem kapcsoljuk. A kézikönyvben ismertetett PLAY 0 utasítás teljesen hatástalan.
A PLAY 2 utasítás akkor is jól használható, ha zene közben akarunk különböző zenei paramétereket változtatni (ld. a VOL és WAVE példákat). Sajnos ha a zene a program lefutásánál rövidebb, semmit sem tehetünk. Ezért a program időtartamát próbálgatással be kell állítani.

P. 100

```
1 rem **********************
2 rem * m u s i c/P l a y *
3 rem **********************
100 rem
110 rem
120 rem ********************
130 rem a naP nyu9odni ter
140 rem ********************
150 rem
160 vol 15:rem han9ero
170 rem
180 rem han9je99ek
190 rem
200 a$="S1c3■f3■■93■A3■a3■a3■a3■■93■■f3■■93■a3■■A3■d4■c4■c4■■A3■■a3■"
210 a$=a$+"c4■c4■■A3■■a3■■A3■c4■a3■■■■"
220 b$="c4■a3■■c4■f4■d4■d4■■c4■■A3■■A3■■93■■A3■d4■c4■"
230 b$=b$+"c4■■A3■■a3■c4■c4■■A3■■a3■■A3■c4■a3■■■■■"
240 rem
250 rem
260 rem haromszo9
270 wave 1,00010001
280 envelope 1,4,5,5,9
290 rem ********************
300 music 10,a$+b$+b$
310 play 2
320 rem ********************
330 print chr$(147) :rem a kepernyo torlese
340 print "a naP nyu9odni ter "
350 for w=1 to 2600:next w
360 print"leszall a csendes ej"
370 rem
380 for w=1 to 2600 :next w
390 print "mar bucsut mondunk "
400 for w=1 to 2600 :next w
```

```
410 for x=1 to 2
420 Print
430 Print"a tolgy a berc alatt"
440 for w=1 to 2600 :next w
450 Print"oly busan bolo9at "
460 for w=1 to 2600 :next w
470 Print"mert el kell valnunk"
480 for w=1 to 2600 :next w
490 next x
500 vol 0
```

Bizonyára nem okozott gondot a füzérekben található karakterek értelmezése. Ha mégis, akkor lapozzuk fel ismét a VOL utasítást.
A legfontosabb tudnivalókat ismételjük át itt is. (Fontos, hogy a kisbetűs/nagybetűs karakterkészletet használjuk.) A kisbetűk a normál hangjegyeket jelölik, bevitelük a megfelelő billentyűvel a szokott módon történhet. A listában szereplő nagybetűk vezérlő karakterek, melyek a nyolc funkcióbillentyűvel (E–L) és a SHIFT/CLR/HOME/S billentyűkkel érhető el. Ezek a betűk a képernyőn inverz alakban láthatók.
A nagybetűk megemelt hangokat jelentenek, és a megfelelő betű és a SHIFT együttes lenyomásával állíthatók elő. A képernyőn ezek normál nagybetűként jelennek meg. A számok a szokott módon a szólamra, illetve az oktávra vonatkoznak.

15.7 Ellenőrző kérdések

1. Mi történik a VOL 2 utasítás hatására?

   a) Valamennyi szólam hangerejét kettes fokozatra állítja.
   b) Meghatározza 2 szólam hangerejét.
   c) A szintaktika nem jó.
   d) A VOL 2 a harmadik legerősebb fokozat.

2. Mit kell a WAVE utasításnál figyelembe venni?

   a) A WAVE az összes szólamra beállítja a hullámformát.
   b) A zaj nem kombinálható más hullámformával.
   c) A szintaktikája: WAVE z, % 00010001.

3. Hogyan használjuk a MUSIC-utasítást?

   a) Ilyen utasítás nincs.
   b) Elõször a ritmust, majd a hangokat adjuk meg.
   c) A b) válasz fordítottja.
   d) Egy hangot a név, oktáv és az érték megadásával definiálunk.
   e) Ha a hangjegyet a SHIFT billentyűvel ütjük be, leszállított hangot kapunk.

16. FEJEZET
VEZÉRLŐ PERIFÉRIÁK

A Commodore 64-esnek megvan az a lehetősége is, hogy vezérlő egységeket, mint perifériális építőelemeket csatlakoztassunk hozzá. A számítógép oldalán két kontrol-port csatlakozási lehetőség található, melyeken keresztül joystick, paddle, fényceruza, ill. egyéb saját építésű vezérlőegység csatlakoztatható. Az egyes kivezetések műszaki jellemzőit a CBM 64 kézikönyvében találjuk meg. A joystick csatlakoztatásához természetesen nem kell belemélyedni a technikai részletekbe. Ebben a fejezetben azon perifériális egységekről szólunk, melyeket a SIMON's BASIC támogat.

16.1 Fényceruza

Fényceruza alatt egy olyan perifériális egységet értünk, amelynek segítségével közvetlenül a képernyőre vihetünk, vagy csak meghatározhatunk egy pontot. Használatával arra is lehetőségünk van, hogy egyszerűen a képernyőhöz érintve átadhassunk a számítógépnek egy képpozíciót. De hogyan történik mindez?
Az 1. kontrol-port-ra csatlakoztatott fényceruzával rámutatunk a képernyő valamelyik pontjára (lehet a képmezőn kívülre is). A számítógép azonnal képes a pont koordinátáit meghatározni. Így írhatunk pl. olyan programot, amellyel megállapíthatjuk, hogy egy adott helyen létezik-e a keresett tárgy (betű vagy grafika). De ennél létezik egy sokkal érdekesebb felhasználási terület is, mégpedig a rajzolás. Szabadkézzel rajzolhatunk a fényceruza segítségével. Felhasználható a fényceruza, kényelmes kurzorvezérlő eszközként és még számtalan módon. A fényceruza más koordináta-rendszerben működik, mint amit a grafikánál megismertünk.
Az alábbiakban megadott számértékek csak irányértékek. Értékük a különböző tv-készülékeknél más és más lehet. Mint már utaltunk rá, a koordináta-rendszer nem a képmező, hanem a képernyő szélénél kezdődik. Az x értéke 30-nál, ami a képernyő peremének felel meg, míg a szövegmező széle 40-nél található. A megfelelő értékek a jobb oldalon: 200, ill. 210.
Az y-tengely irányában a helyzet kicsit bonyolultabb. Az x koordinátához hasonlóan y is 30-nál kezdődik, a képmező 40-nél indul és 240-nél fejeződik be. A képernyő alsó fele két részre oszlik. Az y=255-ig a koordináták még szabályosan változnak, de nagyjából a képmező közepétől y=0-tól újra indul a számozás és y=25-ig tart.
Ahhoz tehát, hogy a fényceruza koordinátáit az általunk megszokott beosztásra átszámítsuk, az alábbi képletet kell használnunk:

```
x = (xp-40)*2
y = yp-40
```

ahol: x, y a grafika, míg az xp, yp a fényceruza koordinátái. A képletekből láthatjuk, hogy a képernyő felbontása a grafikáénak a fele. Ez azonban nem jelent sokat, mert a fényceruza felépítésénél fogva x-irányban úgyis szór.

16.1.1 PENX – a fényceruza vízszintes irányú helyzete

Formátum: PENX
Paraméterek: –
Funkció: a fényceruza x koordinátájának meghatározása
Példa: x=PENX

A fényceruza képernyőn elfoglalt aktuális pozíciójának meghatározásához egy egész sor PEEK és POKE utasítást kell bevinni a képvezérlő különböző regisztereibe. A SIMON's BASIC segítségével mindez leegyszerűsödik.

A PENX utasítás használatával egyszerűen és gyorsan megtudhatjuk a fényceruza x-koordinátáját. Függvényként kell alkalmazni úgy, mint a TEST, LIN vagy az SQR utasítást. Értékét tárolhatjuk változóban és különböző számításokat végezhetünk vele. Pl.:

```
x = 5 * PENX + 4
```

Az utasítás mindig a fényceruza legutolsó pozícióját adja meg. A koordináta-rendszer kezdőpontját most nem a képmező bal felső sarka, hanem a képernyő bal széle jelenti. De még ezzel a bővítéssel is kisebb a felbontás, mint a nagyfelbontású grafikánál.

16.1.2 PENY – a fényceruza függőleges irányú helyzete

Formátum: PENY
Paraméterek: –
Funkció: a fényceruza y-koordinátájának meghatározása
Példa: y = PENY

A PENY utasítás teljesen analóg módon működik, mint a PENX az x-koordináta meghatározásánál.
A mérés szintén a képernyő szélétől történik, nevezetesen az y = 30 értéktől. Függőlegesen összesen 250 pont van. Ha ezt a SIMON's BASIC grafikájára át akarjuk számítani, a bevezetőben ismertetett képletet használjuk. Töltsük be a példaprogramot és indítsuk el:

P. 101

```
1 REM ********************
2 REM * P E N X/P E N Y *
3 REM ********************
100 REM
110 HIRES 6,7
120 PROC PLOT
130 X=(PENX-40)*2:REM X KOORDINATA KISZAMITASA
140 Y=PENY-40     :REM Y KOORDINATA KISZAMITASA
150 IF X<0 OR X>319 OR Y>199 THEN CALL PLOT
160 LINE XA,YA,X,Y,1:REM VONAL RAJZOLASA A REGI KOORDINATABOL AZ UJBA
170 XA=X
180 YA=Y   :REM ERTEK MEGADASA
190 GET A$:REM BILLENTYURE VAR
200 IF A$="=" THEN HIRES 6,7 :REM "="-NEL TORLES
210 CALL PLOT
```

Ennek a programnak a segítségével fényceruzával rajzolhatunk a képernyőre.
Próbáljuk ki!
Vigyük be az aláírásunkat vagy valamilyen rajzot.

16.2 Paddle és joystick

Ebben a fejezetben azokról a vezérlő perifériákról lesz szó, amelyeket főleg a játékprogramok futtatásánál használunk.

Paddle:

A CBM 64 összesen 4 ún. A/D átalakítót (analóg/digitális átalakító) tartalmaz. Ezek olyan elektronikus kapcsolások, melyek lehetővé teszik analóg jelek mérését (pl. hőmérséklet) és ezek digitális tárolását a számítógépben. Ezek a digitális értékek egy 256-os skála részei. Az A/D átalakítókhoz a kontrol-port-on keresztül paddle is csatlakoztatható. Ez tulajdonképpen egy forgó potenciométer, amin forgatással 0 és 255 közötti értékeket állíthatunk be. A kiválasztott értékek megfelelő programmal feldolgozhatók. Szolgálhatják pl. a hangutasítások paramétereinek vezérlését és ezáltal szép, különleges effektusok előállítását.

Joystick

A joystick a legelterjedtebb perifériális egység, mivel a játékprogramok egyik szükséges eszköze. Ez egy valódi botkormány, amit minden irányban elmozdíthatunk. Ezen felül rendelkezik egy tűzgombbal, ami nyomogatható. Lekérdezhetjük a joystick helyzetét, valamint a tűzgomb állapotát.
A botkormánnyal különböző figurák mozgathatók a képernyőn, és lövések adhatók le. Szinte minden, a Commodore 64-esre írt játék megkívánja a használatát, de természetesen nagyon jól vezérelhető vele a kurzor is.

16.2.1 POT – a paddle állásának leolvasása

Formátum: POT (p)
Paraméterek: p: a paddle sorszáma
Funkció: a paddle állásának leolvasása
Példa: x = POT (1)

A paddle aktuális értéke határozható meg ezzel az utasítással. A függvényekhez hasonlóan a POT utasítás is beépíthető a számításokba. Kezelésére ugyanazok érvényesek, mint a PEN-re. Nézzünk két mintaprogramot:

P. 102

```
1 REM ***********
2 REM * 1.P O T *
3 REM ***********
100 REM
110 PRINT CHR$(147)  :REM A KEPERNYO TORLESE
120 P=POT(0)         :REM A PADDLE ERTEKENEK KIOLVASASA
130 PRINT P
140 GOTO 120
```

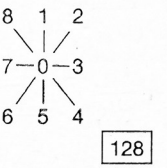
P. 103

```
1 REM ***********
2 REM * 2.P O T *
3 REM ***********
100 REM
110 PRINT CHR$(147)  :REM KEPERNYO TORLESE
120 PO=POT(0)/6.4    :REM A PADDLE 0 ERTEKENEK KIOLVASASA
130 P1=POT(1)/10.3   :REM A PADDLE 1 ERTEKENEK KIOLVASASA
140 PRINT AT(PO,P1) "*":REM RAJZOLAS A  KIJELOLT HELYRE
150 GOTO 120
```

16.2.2 JOY – a joystick állásának leolvasása

Formátum: JOY
Paraméterek: –
Funkció: a joystick állásának leolvasása
Példa: J = JOY

A JOY utasítással a joystick aktuális helyzetét lehet lekérdezni. Sajnos azonban csak a 2. kontrol-port-ra csatlakoztatott joystickra érvényes.
Az 1 kontrol-portot valószínűleg azért nem vonták be az utasítás hatáskörébe, mert az ehhez kapcsolt botkormány bizonyos állásaiban betűket és más jeleket küld a számítógépnek.
Az utasítás minden irányra más értéket vesz fel, az alábbi elrendezésnek megfelelően:



Példa:
Ha a joysticket előretoljuk, az eredmény 1 lesz, balra előre 8. Ha a tűzgombot is lenyomjuk, a pillanatnyi értékhez a számítógép hozzáad 128-at.

P. 104

```
1 REM  *********
2 REM  * J O Y *
3 REM  *********
100 REM
110 IF JOY=1 OR JOY=129 THEN PRINT "                    FELFELE"
120 IF JOY=3 OR JOY=131 THEN PRINT "                    JOBBRA"
130 IF JOY=5 OR JOY=133 THEN PRINT "                    LEFELE"
140 IF JOY=7 OR JOY=135 THEN PRINT "                    BALRA"
150 IF JOY>=128         THEN PRINT "TUZGOMB"
160 GOTO 110
```

17. FEJEZET
FÜGGELÉK

17.1 Az "Ellenőrző kérdések" megoldásai

Az alábbiakban közöljük a tesztfeladatok helyes megoldásait:

FEJEZET	MEGOLDÁS				
2.1.3	1/b	2/c.	3/c.	4/c.	
2.2.4	1/a.	2/a.	3/b, c.	4/b, c.	
2.3.5	1/c.	2/a.	3/c.	4/b.	5/b. 6/b.
3.1.6	1/c.	2/d.	3/a, b.	4/a.	5/b.
3.2.5	1/a.	2/d.	3/c.	4/b.	5/c.
5.1.5	1/b.	2/b.	3/b.	4/a.	5/d.
5.2.6	1/a, b.	2/a.	3/b.	4/b.	5/b. 6/c.
6.3	1/b.	2/b.	3/a, b, c.		
7.1.5	1/b.	2/a.	3/b.	4/c.	5/b.
7.2.3	1/c.	2/a, c.	3/a.	4/b, d.	
8.5	1/c.	2/a, c.	3/d.	5/c.	6/c.
	7/b.	8/a.	9/c.		
9.1.6	1/b.	2/b, d, f.	3/c, d.		
9.2.5	1/c.	2/c.	3/c.		
9.3.4	1/b, d.	2/c.			
9.4.6	1/c.	2/b, c.	3/c.		
9.5.5	1/d.	2/a, d.			
10.4	1/d.	2/c.	3/b.	4/b.	
11.1.5	1/a.	2/a.	3/c.	4/d.	
11.2.3	1/a, b.	2/c.	3/b.		
12.2.7	1/b.	2/c.	3/b, c.		
12.3.10	1/c, e, f.	2/b.	3/a, b, c.		
12.4.3	1/a, c.	2/c.	3/a, d.		
12.5.3	1/b, c.	2/b.	3/b.		
13.5	1/b, c, d.	2/b.			
14.1.5	1/b, c	2/d	3/b, d		
14.2.6	1/b, d.	2/a.			
15.7	1/a.	2/b.	3/b, d, e.		

17.2 Színkód táblázat

Grafikus utasításoknál szükséges a színek színkód alakjában történő bevitele. A 16 szín mindegyikéhez tartozik egy ilyen kód. Ezeket a kódokat tartalmazza az alábbi táblázat, amely ezen kívül megadja a működtető vezérlőbillentyűket is.
A K a CTRL, míg a C a Commodore (C=) gombot jelöli.

Kód	Szín	Billentyű
0	fekete	K1
1	fehér	K2
2	vörös	K3
3	türkiz	K4
4	ibolya	K5
5	zöld	K6
6	kék	K7
7	sárga	K8
8	narancs	C1
9	barna	C2
10	világos vörös	C3
11	1. szürke	C4
12	2. szürke	C5
13	világoszöld	C6
14	világoskék	C7
15	3. szürke	C8

17.3 Hibaüzenetek

A SIMON's BASIC teljes hibaüzenetlistáját és leírását az alábbiakban ismertetjük. Sajnos ezek a kézikönyvből teljesen hiányoznak.

PROC NOT FOUND

Ez az üzenet akkor jelenik meg, ha a programban nem definiált szimbolikus ugrási címre hivatkozunk. Körülbelül az UNDEF'D STATEMENT ERROR hibajelzésnek felel meg, ami a BASIC-ben a nem létező sorszámra történő ugráskor kerül kiírásra.

INSERT TOO LARGE

Az INST és INSERT utasításoknál (ld. a 8.1./8.2. fejezetet) léphet fel, ha nemlétező karakterpozíciót adtunk meg, vagy a belső füzér hosszabb a külsőnél.

STRING TOO LARGE

üzenetet kapunk, ha az INST és INSERT utasításokkal 255 karakternél hosszabb füzért állítunk elő. Megegyezik a normál BASIC"STRING TOO LONG ERROR" hibaüzenetével.

NOT BINARY CHAR

Ezt a hibaüzenetet akkor kapjuk, ha a % jellel kezdett bináris szám a 0 és 1 karaktereken kívül mást is tartalmaz, ill. túl hosszú. Ugyanílyen üzenetet kapunk, ha a WAVE utasításnál kitesszük a % jelet, mert itt nincs rá szükség.

NOT HEX CHAR

Analóg az előbbi hibajelzéssel, csak ez a $ jellel kezdett hexadecimális számokra vonatkozik.

END PROC WITHOUT EXEC

Azonos a normál BASIC "RETURN WITHOUT GOSUB ERROR" hibajelzésével. Akkor lép fel, ha END PROC utasítást alkalmazunk, megelőző EXEC utasítás nélkül.

END LOOP WITHOUT LOOP

Analóg a "NEXT WITHOUT FOR ERROR" normál BASIC hibaüzenettel. Akkor lép fel, ha END LOOP utasítást alkalmazunk, megelőző LOOP utasítás nélkül.

UNTIL WITHOUT REPEAT

Megegyezik az előző hibaüzenettel, de a REPEAT ciklusra vonatkozik.

STACK TOO LARGE

Hasonló az "OUT OF MEMORY ERROR" hibaüzenethez. Túl sok ciklus vagy szubrutin alkalmazása idézi elő.

BAD CHAR FOR A MOB

Ez a hibaüzenet a sprite- vagy karakterdefinícióban használt túl sok (vagy túl kevés) karakter szerepeltetését jelzi. Ezt a hibaüzenetet kapjuk, ha a definícióhoz sok vagy kevés sort használunk fel, ill. a DESIGN utasítás és a definíció közé valamilyen BASIC sort iktatunk.

BAD MODE

Hasonló az "ILLEGAL QUANTITY ERROR" hibajelzéshez. Akkor jelenik meg, ha a SIMON's BASIC engedélyezett tartományait átlépjük.

17.4 A SIMON's BASIC utasításkészlete ABC sorrendben

17.5 A SIMON's BASIC jellemzői

- BASIC-tár igénye: 8 kbyte ($7FFF–$9FFF).
- Részben a BASIC–ROM alatti területet foglalja el.
- Részben a BASIC–ROM-ra, részben a BASIC–RAM-ra kapcsol.

- A RUN/STOP védelem POKE 788,52-vel, a RUN/STOP–RESTORE védelem POKE 792/193-mal nem lehetséges, mivel az interrupt-szektor "elhajlik".

- DATA BECKER IEC-busz változat S 3.0-val futtatható (kivéve a jelkészlet változtatást – 13. fejezet).

- DISKOMAT-tal (SUPERTWIN és DISK MONITOR) futtatható, ha a "SUPERTWIN IEC-busz" üzemmódot választjuk ki.

- 80 karakteres "MAXI64" kártyával nem futtatható.

- A PROFIMAT csomagban lévő PROFIMON 64-gyel működőképes.

- VC 1526-os nyomtatón a HIRES grafika nem biztosít hardcopyt.

– JEGYZET –

Megnyílt a NOVOTRADE Rt. COMMODORE User's klubja!

SZOLGÁLTATÁSAINK:

- klubtagoknak szabad gépidőt és szakirodalmat biztosítunk.
- közületeknek és magánszemélyek részére C 16-os, C 64-es programozói, valamint speciális tanfolyamokat szervezünk.

RÉSZLETES FELVILÁGOSÍTÁS:

2C

Számítástechnikai szaküzlet

1136 Budapest, Balzac u. 35.
Telefon: 402–954
Gábriel László klubvezetőnél