HT-1080Z
ISKOLASZÁMÍTÓGÉP

HT-2080Z
SZÁMÍTÓGÉP

basic kézikönyv

HÍRADÁSTECHNIKA SZÖVETKEZET

HIBAJEGYZÉK

50. oldal: 30 PRINT D■

70. oldal: 10 CLEAR 1000 : DIM B%(26)

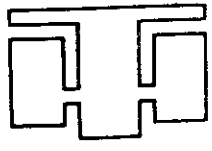
71. oldal: ?ALMA A FA ALATT

HT-1080Z
ISKOLASZÁMÍTÓGÉP

HT-2080Z
SZÁMÍTÓGÉP

ELLENŐRIZVE 1991/1992

basic
kézikönyv



HÍRADÁSTECHNIKA SZÖVETKEZET

TARTALOM

BEVEZETÉS

Az Ön számítógépe négy különböző műveleti módban dolgozhat.

Parancs üzemmód: ebben az állapotban a számítógép a NEW LINE billentyű lenyomása után azonnal reagál a parancsra. Ha a képernyőn a > jel látható, ez mindig azt jelenti, hogy a felhasználó parancs üzemmódban használja a gépet.

Programvégrehajtás üzemmód: ebbe az állapotba a RUN utasítással hozhatjuk a számítógépet és ezáltal a számítógép a tárolóban lévő BASIC programot végrehajtja. Minden numerikus változót nulláz és minden string-változóhoz az üres stringet (nulla string) rendeli. Ezután kezdődik a programvégrehajtás, melyet a RUN utasítással indítottunk el. (Bővebb leírást lásd az 1. fejezetben)

Szövegszerkesztés üzemmód: ez az állapot lehetővé teszi a felhasználónak, hogy megváltoztassa vagy törölje a tárolt programokat, vagy új szöveget írjon be a tárolt programba. A felhasználónak a hibás sorokat nem kell teljesen újraírnia, hanem csak a hibás helyeket kijavítani. (Lásd 2.fejezet)

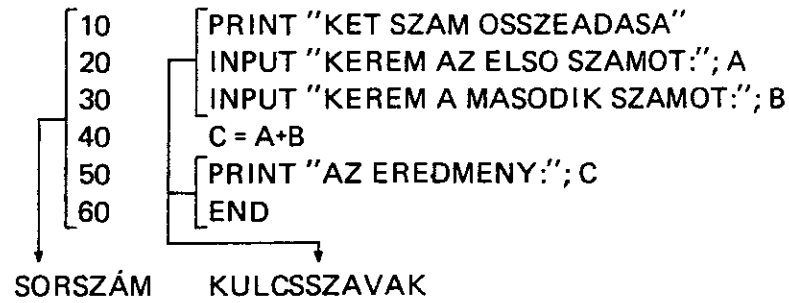
Monitor üzemmód: ez az állapot lehetővé teszi a felhasználónak, hogy gépi kódú file-okat töltsön a memóriába. Ezekhez a file-okhoz más BASIC programok is hozzáférnek, vagy a file BASIC programoktól teljesen függetlenül is végrehajtható. A file-ok adatokat, valamint gépi programokat tartalmazhatnak (lásd SYSTEM utasítás, 1. fejezet).

Mielőtt hozzákezdenénk a programíráshoz, ismerkedjünk meg a programozás alapjaival.

KULCSSZAVAK

A BASIC-ben jónéhány kulcsszó van, melyek a program vázát képezik.
Néhány közülük: PRINT, INPUT, IF, THEN, GOTO, END
A kulcsszavak teljes listáját az „A" függelékben találjuk.

Példa



Ez a program beolvas két számot, összeadja azokat és az eredményt kijelzi.

```
READY
> RUN
KET SZAM OSSZEADASA
KEREM AZ ELSO SZAMOT:?42
KEREM A MASODIK SZAMOT:?45
AZ EREDMENY: 87
```

VÁLTOZÓK

Bizonyára tudja a saját házszámát. Ez olyan jelzés, amely az Ön házát megkülönbözteti a többi háztól az utcájukban. Ha ezt az adatot ismeri, a postás a levelet a helyes postaládába tudja dobni. A változók neveinek ugyanaz a szerepe, mint a házszámoknak, csupán a számok helyett betűk szerepelnek.

Tekintsük a következő — ún. értékadó — utasítást:

```
A = 170
```

Ennél az eseménynél a számítógép az A=170 utasítást hajtja végre: megkeresi azt a helyet, ahol az A változó van és beír oda 170-et.

Nézzük meg a következő programot ugyanebből a szempontból:

```
10      A = 1
20      A = A + 10
30      B = A * 2
40      PRINT "AZ EREDMENYEK:"; A,B
50      END
READY
> RUN
AZ EREDMENYEK: 11        22
```

Eddig csak olyan változókkal foglalkoztunk, amelyek számokat tartalmaznak. Azonban a változók betűket vagy betűsorokat is tartalmazhatnak.

Ezek a változók azonban valamivel másképp néznek ki, mint a számváltozók.

```
10      A$  ="KOVACS PETER"
20      B$  = "PETOFI UTCA 12,"
30      C$  = "1051 BUDAPEST"
40      PRINT A$
50      PRINT B$; C$
60      END
READY
> RUN
KOVACS PETER
PETOFI UTCA 12, 1051 BUDAPEST
```

Ez a program az A$, B$, C$ változókhoz a "KOVACS PETER", "PETOFI UTCA 12,", "1051 BUDAPEST" értékeket rendeli. Ezután ezeknek a változóknak a tartalma megjelenik a képernyőn. Figyeljünk arra, hogy ezeket a karakteres változókat a nevük után a $ jellel jelöljük meg. Ezek az $ jelek jelzik aztán a gépnek, hogy a hozzájuk tartozó változó nemcsak számokat, hanem a számokon kívül betűket is tartalmazhat. Azokat a karaktersorozatokat, amelyeket ezekhez a változókhoz rendeljük, idézőjelek közé kell tenni. Pl: D$="ABCDE12345=?+".

Figyelem: ha egy változótípushoz nem megfelelő típusú értéket rendelünk, akkor a gép ezt nem érti és hibajelzést ad.

Pl: A="HIBAS TIPUSU ADAT" (A itt számváltozó)
B$=100 (B$ string-változó)

Végezetül megjegyezzük, hogy a változóneveknek egyedieknek kell lenniük.

Két háznak egy utcában nem lehet ugyanaz a száma. Az Ön számítógépes rendszere elfogad olyan változókat, amelyek két jelnél hosszabbak. Azonban a változók megkülönböztetésére csak az első két jelet használja.

Pl: az "ER" és az "EREDMENY" változókat azonosnak értelmezi.

A változónévnek betűvel kell kezdődnie (A-tól Z-ig), melyet egy betű vagy egy szám (0-tól 9-ig) követhet.

A következő változók a számítógép számára helyesek, és megkülönböztethetők:

A, AA, AB, AC, AD, BN, BZ, B7, ZZ, Z1, stb.

Figyelem: A BASIC kulcsszavait a változónevek nem tartalmazhatják, pl: a CIF változó nevében benne van az IF.

Az összes kulcsszó felsorolását az A függelékben találjuk.

Változótípusok

A számítógép négyfajta változótípussal dolgozik. Az első három típus numerikus adatok, a negyedik pedig karaktersorozatok tárolására, műveletvégzésre használatosak.

1. % : egész típusú (integer). Intervallum: –32768 ÷ +32767
Pl: A% = –30
BB% = 8000
nem lehet: X % = 3.141

2. ! : egyszeres pontosságú (max. 6 számjegyes)
Pl: A! = 40.33
D4! = 0.1241

3. # : dupla pontosságú (max. 16 számjegyes)
Pl: A# = 3.141592653589
A2# = –4507.896554

4. $: karaktersorozat (maximálisan 255 jel hosszú)
Pl: A $ = "HT – 1080Z SCHOOL-COMPUTER"
M2 $ = "(A*B + 15) / 2.5 EREDMENYE"

Ugyan az A%, A!, A# és A$ változónevek mindegyikében szerepel az "A", a számítógép számára ezek mégis különböző változók, különböző értékekkel.
Típusjelzés nélküli változókat a gép egyszeres pontosságúnak tekint. Ezt az alapértelmezést a DEF xxx utasításokkal (ld. 42–44. old.) lehet megváltoztatni.

Aritmetikai műveleti jelek

Ha egy programnak egy értéket ki kell számolnia, akkor mindig aritmetikai operátorokat használ.

```
 5    R=6
10    A=R*3.1416*2          :REM A KOR KERULETE
20    PRINT "A KERULET:";A
30    B=3.1416 * R[2        :REM A KOR TERULETE
40    PRINT "A TERULET:"; B
50    END
READY
> RUN
A KERULET: 37.6992
A TERULET: 113.098
```

A rendszer a következő aritmetikai szimbólumokat használja: + összeadás, – kivonás, * szorzás, / osztás és [a hatványozás jele.

Pl: $5 \cdot 12^{1/3}$ – a következőképpen írhatjuk le: 5 * 12[(1/3)

A [jelet a ↑ billentyűvel lehet előállítani.

Relációs jelek

Relációs jeleket a program döntések végrehajtásakor alkalmaz. A következő operátorok állnak rendelkezésünkre:

< kisebb	< = kisebb vagy egyenlő
> nagyobb	> = nagyobb vagy egyenlő
<> nem egyenlő	= egyenlő

Példa:

```
120 IF A < B  THEN PRINT  "B NAGYOBB A-NAL"
```

Ha a számítógép ezt az utasítást hajtja végre és a B változó értéke valóban nagyobb, mint az A változóé (azaz az A < B feltétel igaz), akkor a számítógép kinyomtatja a következő mondatot: B NAGYOBB A-NAL. Egyéb esetben a számítógép a következő utasításon folytatja a program végrehajtását.

Logikai műveletek

A számítógép az AND, OR és a NOT logikai műveleteket ismeri.

Példa:

```
10 IF A = 1  AND B = 5 GOTO 50
```

A számítógép az 50-es sorra ugrik, ha A=1, és B=5. Ha nem, akkor a gép a következő programsoron folytatja a program végrehajtását.

```
20 A = (B=2) AND (C > 10)
```

A értéke –1 lesz, ha mindkét állítás (B=2 és C > 10) igaz. Ha a kettő közül bármelyik nem igaz, akkor A értéke nulla lesz.

```
40 A = (D < 2) OR (E < 20)
```

A értéke (–1) lesz, ha D < 2 vagy E < 20. Ha D > 2 és E > 20, akkor A értéke nulla lesz.

```
70 A = NOT (F > 5)
```

Ha F kisebb vagy egyenlő 5-tel, A értéke (−1), különben nulla lesz.
Tehát a (−1) az "igaz", a 0 pedig a "nem igaz" feltételnek felel meg.

String műveletek

Ezekkel a műveletekkel karaktersorozatokat lehet összehasonlítani és összefűzni.

A stringek összehasonlítására használhatjuk az összes, a 8. oldalon felsorolt relációs jelet.

A műveletek sorrendje

A számítógép először a zárójelben lévő műveleteket hajtja végre, ezután következik a többi művelet.

Az egy zárójelben levő műveletek prioritási sorrendje:

1. hatványozás A [B
2. előjelváltás (negáció) −C
3. szorzás, osztás A*B, C/D
4. összeadás, kivonás C+D, E−F
5. relációk A<B, X$=Y$, 15<>16
6. logikai műveletek NOT, AND, OR

Azonos prioritású műveletek esetén a gép balról jobbra hajtja végre a műveleteket.

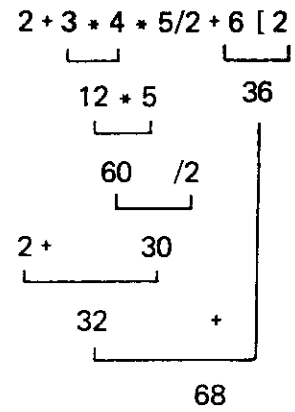
Nézzük pl. a következő képletet:

10 ANS = A+B*C*D/2 + E [2

adjuk a következő értékeket

A=2, B=3, C=4, D=5, E=6

Ezután használjuk a fenti képletet



Tehát az eredmény 68.

1

PARANCSOK

A rendszer összeállítása és a számítógép bekapcsolása után a számítógép parancs üzemmódba kerül. Rendes körülmények között ez arról ismerhető fel, hogy megjelenik a READY és utána egy > jel a következő sorban, a képernyő (monitor v. tv) bal felső sarkában. Mostantól ezt a jelzést "kész" jelzésnek hívjuk.

Most már benyomhatjuk a [NEW LINE] billentyűt és beírhatjuk a következő parancsok egyikét:

1. AUTO	8. EDIT	15. LLIST
2. CLEAR	9. LIST	
3. CLOAD	10. NEW	
4. CLOAD?	11. RUN	
5. CONT	12. SYSTEM	
6. CSAVE	13. TROFF	
7. DELETE	14. TRON	

Ezeket a parancsokat majd még később külön-külön is tárgyaljuk. Jegyezzük meg, hogy a leírás további részében a zárójelben lévő adatok beírhatók, de használatuk nem kötelező. Pl. AUTO (sorszám, inkrementum). A konkrét parancs így néz ki:

AUTO 10,5; vagy AUTO; vagy AUTO 10.

Pl. ha az egész programot ki akarjuk listáztatni, akkor LIST-et kell beírni.

Minden parancsot a [NEW LINE] billentyűvel kell lezárni.
A zárójelben levő adatok opcionálisak, de ha használjuk azokat, akkor a parancs után nem kell zárójelet tenni! Minden parancs — kivéve a CONT — utasításként is használható a programon belül.
A parancsokon túlmenően a > jel után a legtöbb utasítást használhatjuk parancsként, vagyis az utasítás azonnal végrehajtásra kerül, pl.: > PRINT 5*4 [NEW LINE]
20
Ha olyan utasítást próbálunk alkalmazni parancsként, amelynek használata nem megengedett, akkor ID hibajelzést kapunk.
Ha nem kívánjuk alkalmazni az automatikus sorszámozást, írhatunk úgy is programsorokat, hogy a > jel után közvetlenül beírjuk a kívánt sorszámot, majd a programsort. Ha így egy olyan sorszámot adunk meg, amilyent már előzőleg használtunk, az felülíródik. A programsorokat nem kell feltétlenül növekvő sorrendben számozni. Így lehet utólag javítani a programon, pl. az automatikusan számozott 30 és 40 számú programsorok közé beiktathatunk egy 35 számú sort.

1. AUTO (sorszám, inkrementum)

Ez a parancs automatikusan beállítja, és kiírja az aktuális programsor sorszámát. Az opció (zárójeles adatok) lehetővé teszi, hogy megadjuk a kezdő sorszámot és a két sor közötti sorszámkülönbséget. Ha csak az AUTO parancsot írjuk be — ami után természetesen, mint minden parancs után a NEW LINE billentyűt kell beütni —, akkor a számítógép az első számot 10-nek, a sorok közötti különbséget szintén 10-nek veszi (alapértelmezés).
Ha a parancsban csak a kezdő sorszámot adjuk meg, akkor a gép az inkrementumot 10-nek veszi.
A programsor kezdő pozíciója közvetlenül a sorszám után van, ide írhatjuk a sor első utasítását.

Példa:

```
30 PRINT "EZ A 30-AS SOR"
```

Minden egyes programsor beírása után a számítógép automatikusan növeli a sorszámot, az inkrementumban megadott számmal. Az AUTOmatikus sorszámozás állapotot a BREAK billentyű megnyomásával lehet megszüntetni. Ha a számítógép AUTO-üzemmódban egy, már előzőleg begépelt sorszámú sort talál, akkor a sorszám jobb oldalán megjelenik egy csillag, * .

Ha ezt a sort nem akarjuk megváltoztatni, akkor nyomjuk meg a BREAK billentyűt, és ezzel a gép ismét parancs üzemmódba kerül.

Példa

```
READY
> AUTO 1,2
1 LINE 1                NEW LINE
3 LINE 3                   ↓
5 LINE 5
7 LINE 7
9 BREAK
READY
> AUTO 2,2
2 MASODIK SOR           NEW LINE
4 NEGYEDIK SOR             ↓
6 HATODIK SOR
8 BREAK
READY
> AUTO
10 LINE 10              NEW LINE
20 LINE 20                 ↓
30 LINE 30
40 BREAK
```

```
READY
> AUTO 1,1
1 *
2 *
3 *
```

NEW LINE

2. CLEAR (Byteszám)

A CLEAR parancs meghatározott számú byte-ot szabadít fel stringek tárolására, nullázza a változózónát és törli az összes string változót. Ha a CLEAR-hez nem adunk meg byteszámot, akkor a parancs a változókat és a string változókat törli. Ha a byteszámot is megadtuk (pl. CLEAR 100), akkor a fenti műveleteken kívül a megadott darabszámú byte-ot kitörli string változók számára.

A gép bekapcsolása után automatikusan egy CLEAR 50 parancs hajtódik végre.

Ha a számítógép hibajelzést ad "OS" (out of string-space), akkor a CLEAR paranccsal a string változóknak több helyet szabadíthatunk fel.

3. CLOAD (#-kazettás egység száma,"file-név")

Ez a parancs a kazettás egység számával meghatározott magnetofonból tölti be a "file-név" programot. Mielőtt ezt a parancsot adnánk, tekercseljük a szalagot vissza, vizsgáljuk meg, hogy minden vezeték jó helyen van-e, és nyomjuk meg a PLAY billentyűt. Ha minden rendben van, akkor beírhatjuk, hogy pl: CLOAD#–1, "A". A NEW LINE billentyűvel indíthatjuk el ezt a parancsot is, mint az összes többit. Ekkor a magnetofon elkezdi keresni az "A" nevű file-t. Ha megtalálta, akkor egy állandóan világító és egy villogó csillag jelenik meg a képernyő jobb felső sarkában. Ebből látható, hogy a betöltés folyik. Ha a program már teljesen betöltődött, akkor a READY üzenet jelenik meg a képernyőn. Pl: CLOAD#–1, "3" jelentése: az 1. magnetofonról a "3" nevű file-t kell betölteni.

Figyeljünk arra, hogy a gép a file-névnek csak az első karakterét használja a különböző file-ok megkülönböztetéséhez. A "P" és "PÉNZ" ugyanannak a névnek számít.

4. CLOAD? ("file-név")

Ez a parancs összehasonlítja a kazettán lévő file-t a memóriában levő programmal. Rendes körülmények között ez a parancs annak az ellenőrzésére szolgál, hogy a CSAVE parancs után az adatok helyesen lettek-e felvéve a kazettára. Ezzel a paranccsal együtt tanácsos a file-nevet is mindig beírni, mert így a számítógép rögtön a konkrét file-t keresi, és az összehasonlítás nem a szalagon lévő első file-al kezdődik. Ha a képernyőn a BAD üzenet jelenik meg, akkor a kazettát vissza kell tekercselni és a programot a CSAVE parancs segítségével még egyszer fel kell venni.

5. CONT

Ha a programot a [BREAK] billentyűvel megszakítottuk, vagy a program STOP utasítást hajtott végre, akkor a program futtatását a CONT paranccsal a megszakítás helyétől folytathatjuk.

6. CSAVE (#—kazettás egység száma, "file-név")

Ez a parancs a memóriában lévő programot felveszi a kazettára. A kazettás egységet (annak számát) és a file-nevet kötelező megadni. A gép az idézőjeleken kívül minden alfanumerikus karaktert file-névnek értelmez. Hasonlóan, mint a CLOAD-nál, a kazettát be kell állítani, hogy ne töröljük le tévedésből esetleg a szalagon lévő programot. Minden dugaszoló legyen a megfelelő helyen. A PLAY és a RECORD kapcsolót egyidejűleg kell lenyomni.

Példa: a CSAVE#—2, "C" parancs a memóriában levő programot a 2. számú kazettás egységre veszi fel, "C" néven.

7. DELETE sorszám (—sorszám)

Ez a parancs törli a megadott programsort (sorokat).

Példák:

DELETE 5 (Törli az 5-ös sort)
DELETE 7–10 (Törli a 7., a 10. és a köztük levő sorokat)
DELETE –12 (A program elejétől a 12. sorig bezárólag töröl minden sort)
DELETE. (Törli az éppen beírt vagy szerkesztett sort)

A pont más parancsokban is az éppen beírt, kilistázott, szerkesztett vagy hibajelzést kapott – a továbbiakban aktuális sornak nevezendő – sort jelenti.

8. EDIT sorszám

Ennek a parancsnak a hatására a számítógép a parancs üzemmódból a szerkesztés üzemmódba tér át. A szerkesztés üzemmód lehetővé teszi, hogy a tárolóban levő sorokat azok teljes felülírása nélkül megváltoztassuk.

Az EDIT üzemmódnak sok olyan parancsa van, melyet a szerkesztő üzemmódban hajt végre a gép. (Lásd 2. fejezet). Az EDIT parancsot érvényes sorszámnak kell követni.

Példa:

EDIT 20

A gép ekkor átkapcsol a szerkesztő üzemmódba úgy, hogy a 20-as sort módosíthassuk.

9. LIST (sorszám – sorszám)

Ez a parancs utasítja a számítógépet, hogy egy vagy több programsort írjon ki a képernyőre.

Az opció nélkül álló LIST kiírja az egész, a memóriában lévő programot. Ha meg akarja állítani a listázást, nyomja meg egyszerre a SHIFT és a @ billentyűt. A listázás bármelyik billentyű lenyomásakor továbbindul.

Példák:

LIST 3 (A 3-as sort kiírja)
LIST 10–20 (A 10-es, 20-as és a közöttük levő sorokat kiírja)
LIST–50 (Az elsőtől az 50. sorig az összes sort kiírja)

LIST 20– (A 20. sortól a program végéig kiírja a sorokat)
LIST. (Az aktuális sort kiírja)
LIST (Minden, a tárolóban levő sort kiír).

10. NEW

Ez a parancs töröl minden programsort, nullázza az összes változót és minden string változót. A CLEAR-rel definiált string változó területe a tárolóban nem változik.

11. RUN (sorszám)

Ez a parancs elindítja a memóriában levő felhasználói programot. Ha sorszámot nem adunk meg, akkor az első sortól, ellenkező esetben a sorszám által meghatározott sortól indul.

Ha érvénytelen (nem létező) sorszámot adunk meg, akkor hibajelzést kapunk.

A RUN paranccsal egyidejűleg a gép automatikusan a CLEAR parancsot is végrehajtja.

Példák:

RUN 50 (indítsd a programot az 50. sorban)
RUN (indítsd a programot a legalacsonyabb sorszámú sorban)

12. SYSTEM

Ez a parancs a számítógépet a monitor üzemmódba állítja át. Ebben az üzemmódban lehet gépi programokat, adatokat tölteni (a kazettáról) és végrehajtani.

A SYSTEM parancs beírása után megjelenik egy csillag és egy kérdőjel a képernyőn. Ekkor a számítógép azon gépi kódú file (object code) nevének beírását várja, melyet a kazettáról be kell töltenie. A gépi program teljes betöltése után egy további *? jelenik meg a képernyőn. A program elindításához írjunk be egy / jelet, melyet a gépi program indítócíme (decimális írásmódban) követ. Ha csak a / jelet írjuk be, akkor a program a file-ban meghatározott helyről indul.
Ha a már a gépben levő monitor programot kívánjuk egy címről indítani, akkor rögtön a / jelet és az indítási címet kell megadnunk (lásd a RE parancsnál a kibővített BASIC hívásmódját).

13. TROFF

Ez a parancs kikapcsolja a trace-funkciót.

14. TRON

Ezzel a paranccsal a trace- (követés) funkciót kapcsoljuk be. Ez lehetővé teszi, hogy a program futását a hibák megkeresésével kövessük. Új programsor végrehajtása közben a számítógép kiírja ennek a sornak a számát.

Példa:

```
10 PRINT "** PROGRAM 1 **"
20 A=1
30 IF A=3 THEN 70
40 PRINT A
50 A=A+1
60 GOTO 30
70 PRINT "PROGRAM VEGE 1"
80 END
```

Billentyűzzük be:

```
>TRON    [NEW LINE]
>RUN     [NEW LINE]
<10>       * * PROGRAM 1 * *
<20><30><40>  1
<50><60><30><40>  2
<50><60><30><70>  PROGRAM VEGE 1
<80>
```

Ha a program végrehajtását a program vége előtt meg akarjuk szakítani, akkor ezt a [SHIFT] és az @billentyű egyidejű benyomásával érhetjük el, a továbbindítás bármelyik billentyűvel történik. A TRON és a TROFF programban is alkalmazható, utasításként.

Példa:

```
⋮
90 IF A=B THEN 160
100 TRON
110 A=B+C
120 TROFF
⋮
```

Ha ezen programrész végrehajtása során A≠B-vel, akkor a gép a 110. sort hajtja végre. A programon belül használva a TRON és a TROFF parancsokat, a felhasználó pontosan megállapíthatja, hogy a 110. sor végrehajtásra került vagy sem. A képernyőn megjelenik a 110 és 120 sorszám, ha ezek a programsorok végrehajtásra kerülnek. A TRON és TROFF parancsok a program kipróbálása után törölhetők.

15. LLIST

Kilistázza a programot a nyomtatón. Ez a parancs ugyanúgy működik, mint a LIST. Ha a LIST parancsot a számítógép úgy hajtja végre, hogy nincs a rendszerben sornyomtató, akkor a számítógép végtelen ciklusba kerül, amelyből csak a RESET nyomógombbal szabadítható ki.

16. RE (kezdő sorszám, növekmény)

Ez a parancs újraszámozza a BASIC programot. A sorok újrakezdése után új parancsok illeszthetők a programba, továbbá elősegíti a program jobb dokumentálását. Ha a kezdő sorszám vagy a növekmény nincs megadva, akkor azt a gép 10-nek veszi.

Például:

RE, 5 Újraszámozza a programot 10-es kezdő sorszámmal, 5-ös növekménnyel
RE Újraszámozza a programot 10-es kezdő sorszámmal, 10-es növekménnyel

A RE parancs csak a kibővített BASIC-ben alkalmazható.
A kibővített BASIC meghívása:

SYSTEM NEW LINE

Ezután megjelenik a képernyőn:

*?

Erre a következőt kell bebillentyűzni:

/ 12288

Ekkor a képernyőre a következő felirat íródik ki:
NEW KEYBOARD ROUTINE ENABLE
Ezután használhatjuk a RE utasítást.

2

SZÖVEGSZERKESZTÉS

A szövegszerkesztés lehetővé teszi a felhasználónak, hogy a leírt sorokat korrigálja anélkül, hogy a sorokat teljesen újra kellene írnia. A szerkesztés szükségessége különösen a hosszú, bonyolult programsoroknál válik érthetővé. Ebben a fejezetben a rendszer összes szerkesztési funkcióját és ezek parancsait tárgyaljuk. A magyarázatokat sok példával tesszük érthetőbbé. Javasoljuk a felhasználónak, hogy mielőtt a programíráshoz hozzáfogna, próbáljon ki minden EDIT parancsot!

1. EDIT sorszám

Ez a parancs átállítja a számítógépet a parancs üzemmódból a szövegszerkesztés üzemmódba. A felhasználónak meg kell adni azt, hogy melyik sort akarja szerkeszteni. Ha nem adnak meg sorszámot, akkor a gép az FC hibajelzést adja.

Példa:

```
EDIT 100    (A 100. sor szerkeszthető)
EDIT        (Az éppen beírt sor szerkeszthető)
```

2. [NEW LINE] billentyű

Ha a szövegszerkesztés üzemmódban lenyomjuk a [NEW LINE] billentyűt, akkor a gép az addigi változtatásokat eltárolja és visszatér a parancs üzemmódba.

3. n [SPACE] billentyű

Szövegszerkesztés üzemmódban a betűköz (SPACE) billentyű minden lenyomásakor a gép a cursort egy pozícióval jobbra mozgatja és kiírja a megelőző pozícióban levő karaktert. Ha a billentyű leütése előtt az "n" számot megadtuk, akkor a gép a szerkesztett sorban n karaktert ír ki, és a cursort is n karakterrel lépteti jobbra.

Tegyük fel, hogy a következő sorokat írtuk be:

```
>AUTO 100
100 IF A=B THEN 150 : A= A+1 : GOTO 100
```

Ha a 100-as sort szerkeszteni akarjuk, akkor az EDIT 100-at kell beírni.

```
>EDIT 100
```

A számítógép azt válaszolja:

```
100_
```

Ha a space-billentyűt 10-szer ütöttük le, akkor a cursor 10 pozícióval jobbra mozog (cursor: az a jel, amelyet a számítógép minden beírt karakter mögé tesz).

A képernyőn ez áll: 100 IF A=B THE_

Nem kell minden karakterhez egyesével leütni a space-billentyűt. Írjon be 6-ot, utána betűközt; ekkor a sor így módosul:

```
100 IF A=B THEN 150:_
```

Ezek után betűköz előtt írjon be 20-at, ekkor megjelenik az egész sor:

```
100 IF A=B THEN 150 : A=A+1 : GOTO 100_
```

4. n [←] billentyű

A cursort eggyel balra elmozdítja (ellentétben a betűközbillentyűvel). Ha a [←] billentyű beütése előtt beírunk egy számot, akkor a cursor ennek a számnak megfelelően mozdul el balra. Visszalépéskor a törlést csak a képernyőn hajtja végre, a memória tartalma változatlan marad.

Példa:

```
100 IF A=B THEN 150 A=A+1 : GOTO 100_
```

A [←] billentyű 5-szöri leütése után:

```
100 IF A=B THEN 150 : A=A+1 : GOT_
```

Írjunk be most 6-ot, utána a [←] -t, ekkor a kijelzett sor így módosul:

100 IF A=B THEN 150 : A=A_

Még egyszer hangsúlyozzuk; ha ki akarunk lépni a szövegszerkesztés üzemmódból, akkor a [NEW LINE] billentyűt kell lenyomni. A számítógép visszatér parancs üzemmódba és a képernyőn ez jelenik meg:

>_

Ha további változtatásokat akarunk a 100-as sorban végezni, akkor az EDIT 100-al vissza kell térni a szövegszerkesztő üzemmódba.

5. [SHIFT] [↑] billentyű

Ha a [SHIFT] és az [↑] billentyűt egyidejűleg nyomjuk meg, akkor a számítógép kilép az előzőleg aktivizált H, I vagy X inzertálási parancsból. Ezután a számítógép továbbra is szerkesztő üzemmódban marad, a cursor helye nem változik. Az inzertálási parancs végrehajtását úgy is befejezhetjük, ha benyomjuk a [NEW LINE] billentyűt. Ekkor a számítógép visszatér parancsmódba.

6. [H] billentyű

A H parancs segítségével töröljük a sor cursortól jobbra eső részét és ide tetszőleges szöveget illeszthetünk be.

Pl: 100 IF A=B THEN 150 : A=A+1 : GOTO 100

Tegyük fel, hogy az A=A+1 helyére A=A+B-t akarunk írni és a GOTO 100-at törölni akarjuk.

Ekkor először térjünk át szerkesztő üzemmódba az EDIT 100-al. A 19-es szám, majd a szóköz beírása után a cursor a sor 19. pozíciójába kerül:

100 IF A=B THEN 150 : A=A_

Most nyomjuk le a [H] billentyűt, írjuk be a +B-t, majd a [NEW LINE] billentyűt; ezzel a parancs üzemmódba tértünk vissza. Egy másik lehetőség, mellyel a szerkesztő üzemmódban maradhatunk, a [SHIFT] és az [↑] billentyű egyidejű leütése.

Írja be az [L] -t a listázáshoz. Ekkor a számítógép kiírja az egész sort és a cursor a sor elejére mutat:

```
100 IF A=B THEN 150 : A=A+B
100_
```

7. [I] billentyű

Az I parancs jelentése: beszúrás (inzert). Ezzel a paranccsal a cursor által mutatott helyre írhatunk be karaktereket, anélkül, hogy a sor többi része megváltozna.

Példa:

Tegyük fel, hogy a "PRINT A" utasítást akarjuk beszúrni az "A=AH" és a „GOTO 100" közé a 100. sorban.

A 100. sor így nézett ki:

```
100 IF A=B THEN 150 : A=AH : GOTO 100
```

A betűköz-billentyűvel vigyük a cursort a megfelelő pozícióba:

```
100 IF A=B THEN 150 : A=AH :_
```

Most üssük be az [I] billentyűt és írjuk be: "PRINT A". A [SHIFT] és [↑] billentyűkkel oldhatjuk fel a beszúró üzemállapotot. Ekkor az [L] -lel listázhatjuk az egész sort:

```
100 IF A A=B THEN 150 : A=AH : PRINT A : GOTO 100
100_
```

A [NEW LINE] -nal térhetünk vissza parancs üzemmódba.

8. [X] billentyű

Az X parancs jelentése: állj a sor végére. A cursor a sor végére kerül, a számítógép pedig inzert üzemmódba kerül. Ekkor a sor végéhez hozzáírhatunk, vagy a [←] billentyűvel törölhetjük a sor végén levő jeleket.

Példa:

Szerkesztő üzemmódban vagyunk:

```
> EDIT 100
100_
```

Írjuk be az X-et ([NEW LINE] nélkül)!

```
100 IF A=B THEN 150 : A=A+1 : PRINT A : GOTO 100
100_
```

Ezen a helyen most új karaktereket írhatunk be, vagy a már beírtakat törölhetjük. Ezt az üzemmódot a [SHIFT] és [↑] -vel hagyhatjuk el.

9. [L] billentyű

Az L parancs jelentése: listázz ki. Ha a számítógép a szerkesztés üzemmódban van, és nem inzertálási parancsot (H, X vagy I) hajt végre, akkor az L parancs segítségével listázhatjuk ki a sor cursortól jobbra eső részét a képernyőn.

Példa:

```
> EDIT 100
100_
```

Írjuk be az [L] -t.

```
100 IF A=B THEN 150 : A=AH : PRINT A : GOTO 100
100_
```

A második sort szerkeszthetjük úgy, hogy az eredeti szöveget szem előtt tarthatjuk.

10. [A] billentyű

Az A parancs jelentése: kezdj újból! A cursor a sor elejére kerül és minden eddig végzett változtatás törlődik. A sor eredeti állapotában marad.

11. [E] billentyű

Ez a parancs visszaállítja a számítógépet a parancs üzemmódba és minden eddig végzett változtatást bevezet a szerkesztett sorba. Ügyeljünk arra, hogy az E parancsot ne más parancs végrehajtása közben adjuk.

12. [Q] billentyű

Ez a parancs visszaállítja a számítógépet parancs üzemmódba a szerkesztő üzemmódból, de minden elvégzett változtatást töröl. A sorok eredeti állapotukban maradnak.

13. n [D] billentyű

A D parancs jelentése: törölj.
Ha a D előtt beírunk egy számot (n), akkor a cursor jelenlegi helyétől jobbra a karakter törlődik. A törölt karakterek felkiáltó jelek közé kerülnek, annak jelzésére, hogy ezek törölve vannak.

Példa:

```
100 IF A=B THEN 150 : A=A+1 : PRINT A : GOTO 100
```

Lépjünk szerkesztő üzemmódba és mozgassuk a cursort a betűköz-billentyűvel a következő helyre:

```
100 IF A=B THEN 150 : A=A+1_
```

Írjuk be a 11D-t (törölj 11 karaktert):

```
100 IF A=B THEN 150: A=A+1! : PRINT A : GO!
```

Használjuk a teljes sor listázásához az L-t:

```
100 IF A=B THEN 150 : A=A+1! : PRINT A : GO! TO 100
100_
```

Újabb listázás után a következő jelenik meg a képernyőn:

```
100 IF A=B THEN 150 : A=A+1 TO 100
100_
```

A "TO 100" törléséhez használjuk az X-t és a [←]-t. A végeredmény:

```
100 IF A=B THEN 150 : A=A+1
```

14. n [C] billentyű

A C jelentése cserélj!
Lehetőséget ad a felhasználónak a karakter vagy karaktersorozat megváltoztatására.

Ha a C előtt beírunk egy számot (n), akkor a jelenlegi cursor-pozíciótól jobbra n karaktert a C-t követő karakterekre cserélhetünk. Ha nem adunk meg számot, akkor csak egy karakter változik meg.

Példa:

```
100 IF A=B THEN 150 : A=A+1
```

Ha a 150-et 230-ra akarjuk változtatni, akkor először szerkesztő üzemmódba kell lépnünk (EDIT 100), majd a cursort a betűköz-billentyűvel a következő helyre állítanunk:

```
100 IF A=B THEN_
```

Írjuk be most 2C-t (változtassunk 2 karaktert), ezután 23-at (új adatok). Listázzuk ki a sort L-lel:

```
100 IF A=B THEN 230 : A=A+1
100_
```

15. n [S] c

Ez a parancs megkeresi a c karakter n-dik előfordulási helyét az éppen szerkesztett sorban és a cursort erre a helyre viszi. Ha n-et nem adtuk meg, akkor a helyet keresi meg, ahol c először előfordul. Ha nem találja meg a gép c-t, akkor a cursor a sor végére kerül. Mint a többi parancsnál, itt is a pillanatnyi cursor-pozíción kezdi a számítógép a keresést.

Példa:

```
100 IF A=B THEN 230 : A=A+1
```

Térjünk át szerkesztő üzemmódba (EDIT 100)!

```
100_
```

Irjuk be a 2S=-t, hogy közöljük a számítógéppel azt, hogy azt a helyet keresse, ahol az egyenlőségjel másodszor szerepel.

```
100 IF A=B THEN 230 : A_
```

Most egy másik parancs aktivizálható a megtalált adat megváltoztatására.

Például írjuk be a H-t majd utána =A+2-t (új adatok):

```
100 IF A=B THEN 230 : A=A+2
```

16. n [K] c

Ez a parancs a c karakter n-dik előfordulásáig minden karaktert töröl és ide hozza a cursort.

Példa:

```
100 IF A=B THEN 230 : A=A+2
```

Térjünk át szerkesztő üzemmódba (EDIT 100):

```
100_
```

Írjuk be most a következőt: K:. Hatására a számítógép az első: jelig törli a sort.

```
100 ! IF A=B THEN 230 !
```

Ha a kettőspontot is törölni akarjuk, akkor írjunk D-t be:

```
100 ! IF A=B THEN 230 ! ! : !
```

Az eredmény (L-lel listáztatva):

```
100 A=A+2
100_
```

BASIC SZERKESZTÉSI UTASÍTÁSOK ÖSSZEFOGLALÁSA

NEW LINE	tárol minden változtatást (sorzárás)
BETŰKÖZ	cursor eggyel jobbra
←	cursor eggyel balra
SHIFT ↑	befejezi a közbeírási parancsot
H	levágás és beszúrás
I	beszúrás (insert)
X	beírás a sor végén
L	sor kilistázása
A	töröl minden változtatást
E	tárolja a változtatásokat
Q	vissza a parancs üzemmódba a változtatások tárolása nélkül
D	karakter törlés (delete)
C	karaktercsere (change)
S	karakter keresés (search)
K	megadott karakter törlése (kill)

3

BASIC UTASÍTÁSOK

Ebben a fejezetben BASIC programnyelvünk utasításait tárgyaljuk.

Az első részben azokat a bemeneti és kimeneti utasításokat ismertetjük, amelyek segítségével kommunikál a gép a külvilággal.

Részletesebben azokat ismertetjük, amelyek a display-el és a klaviatúrával kapcsolatosak, illetve amelyek lehetővé teszik a kazettás egység használatát.

A fejezet második része a számítógép további utasításait tárgyalja.

A programnyelv alapos megismerése érdekében javasoljuk, hogy az egyes utasításokhoz tartozó példákat gondosan tanulmányozza!

BE- és KIÍRATÁSI UTASÍTÁSOK

1. PRINT tételsorozat

Egy tételt vagy egy tételsorozatot ír ki a képernyőre. Tételnek a következőket tekintjük:

a) szám konstans (számok, mint pl. 0,36872,0.2,–34)
b) számváltozó (olyan nevek, amelyek számokat reprezentálnak, mint A, XX, Z 3)
c) karakterállandók (idézőjelben levő karakterek, mint pl. "HT–1080Z", "123 $ %" stb.)
d) karakterváltozók (olyan nevek, melyek karakterállandót reprezentálnak, mint A$, X$, PR$ stb.)
e) kifejezések (a fenti tételek variációi, mint pl. X+10/Y vagy "COMP" + "UTER" stb.)

Az egyes tételeket vesszővel vagy pontosvesszővel választjuk el egymástól. Ha vesszőt használunk, akkor a cursor automatikusan a következő nyomtatási zónára ugrik, mielőtt a következő tételt kiírja. Ha pontosvesszőt használunk, akkor a tételek között nem marad szabad hely: egymáshoz fűzve jelennek meg. Numerikus tételek között egy hely kimarad.

Példa:

```
10 N = 25+7
20 PRINT "25 + 7 EGYENLO"; N
30 END
```

```
READY
> RUN
25 + 7 EGYENLO 32
```

Példa:

```
10 H$ = "ITT  "
20 C$ = "VAN  "
30 PRINT H$; C$; "A TAVASZ"
40 END
READY
> RUN
ITT VAN A TAVASZ
```

Ha vesszőt használunk a tételek elválasztásához, akkor a számítógép soronként 4, egyenként 16 karaktert tartalmazó zónába ír.

Példa:

```
10 PRINT "PIROS", "KEK", "ZOLD", "SARGA", "LILA"
20 END
READY
> RUN
PIROS           KEK             ZOLD            SARGA
LILA
```

Ha két vagy több vesszőt írunk be, akkor vesszőnként 16 szóköz jelenik meg.

Példa:

```
10 PRINT "PIROS",, "LILA"
20 END
READY
> RUN
PIROS                           LILA
```

Példa:

```
10 PRINT "ELSO SOR"
20 PRINT "MASODIK SOR"
30 END
READY
> RUN
ELSO SOR
MASODIK SOR
10 PRINT "ELSO SOR",
20 PRINT "MASODIK SOR"
30 END
READY
> RUN
ELSO SOR          MASODIK SOR
```

Megjegyzés: A PRINT utasítás végrehajtása után a számítógép automatikusan soremelést hajt végre, kivéve azt az esetet, ha valamelyik elválasztójellel (; ,) ezt meg nem tiltjuk.
A PRINT helyett használhatjuk a ? jelet is.

2. PRINT @ mutató, "tételsorozat"

Ez az utasítás a tételsorozatot a 64 oszlopra és 16 sorra felosztott képernyő megadott helyére írja ki.
Az @ jelnek közvetlenül a PRINT után kell következni. A mutató értéke Ø és 1023 közötti lehet.

A mutató értéke a kívánt sorértékből és oszlopértékből a következőképp számítható ki:
oszlopérték +64 * sorérték.
Példa: 20 PRINT @ 100, "100-AS HELY"

Ha a felhasználó olyan PRINT utasítást ad meg, amely a képernyő utolsó sorára írat ki, akkor ez automatikus soreltolódást eredményez, azaz minden sor eggyel feljebb tolódik. Ennek megakadályozására tegyünk az utasítás végére egy pontosvesszőt.

Példa:

```
10 PRINT @ 999, "UTOLSO SOR";
```

3. PRINT TAB (kifejezés) tételsorozat

Ez az utasítás lehetővé teszi azt, hogy a cursort egy adott sorban tetszőleges helyre vigyük. Ha a kifejezés értéke nagyobb, mint 63, akkor a számítógép az általunk megadott kifejezést elosztja 64-el, és a maradéknak megfelelő helyre viszi a cursort. Egy PRINT utasításban több TAB is használható. A kifejezés értéke azonban 0 és 255 között kell, hogy legyen.

Példa:

```
10 PRINT TAB(10) "POSITION 10" TAB(30) "POSITION 30"
20 END
READY
> RUN
          POSITION 10                    POSITION 30
10 N=4
20 PRINT TAB(N) "POS."; N TAB(N+10) "POS."; N+10
READY
> RUN
      POS.4       POS.14
```

4. PRINT USING formátum; tételsorozat

Ez az utasítás lehetővé teszi adatok előzetesen megadott formátumban való kinyomtatását. Az adatok numerikusak és string értékek lehetnek.

A formátum és a tételsorozat a PRINT USING utasításban változókat vagy állandókat tartalmazhat. Az utasítás a formátum megadásának megfelelő formában írja ki a tételeket.
A formátumot vezérlő, a továbbiakban ismertetett karakterektől eltérő egyéb karakterek is írhatók a formátumba, ezek változatlan alakban fognak megjelenni.

A következő specifikációkat használhatjuk a formátummezőben:

#: ezzel a jellel határozhatjuk meg a tétellista számaiban a számjegyek számát. Ha a formátummező nagyobb, mint a szám számjegyeinek száma, akkor a számtól balra szóközök, a tizedesvesszőtől jobbra pedig 0-k jelennek meg. A tizedesvesszőt (pontot) a # jelek által képezett formátummezőben (ezek között a jelek között) bárhol elhelyezhetjük. Ha a számnak a tizedesvesszőtől jobbra több számjegye van a megadott formátumnál, akkor a gép kerekít. (Megjegyzés: a számítógép az angol-amerikai tizes számrendszer jelölését használja, amelyben a tizedesvesszőnek a pont felel meg.)

Példa:

```
10 INPUT "KIIRATASI FORMATUM"; F$
20 IF F$ = "STOP" END
30 INPUT "EGY SZAM BEOLVASASA"; N
40 PRINT USING F$; N
50 GOTO 10
```

Ez a program egy formátumlistára és egy tételre (ebben az esetben egy számértékre) kérdez rá. Csak akkor áll le, ha "STOP" utasítást adunk. *Próbáljuk ki a programot:*

```
READY
> RUN
KIIRATASI FORMATUM ? ##.##
EGY SZAM BEOLVASASA ?  12.34
12.34
KIIRATASI FORMATUM ? ###.##
EGY SZAM BEOLVASASA ?  12.34
  12.34
KIIRATASI FORMATUM ? ##.##
EGY SZAM BEOLVASASA ?  123.45
% 123.45
KIIRATASI FORMATUM ? STOP
```

A % jelet akkor írja ki a gép, ha a megadott mező kisebb, mint a szám. A tizedesponttól balra levő számot % jel után írja ki a gép. *Indítsuk el ismét a fenti programot:*

```
READY
> RUN
KIIRATASI FORMATUM ? ##.##
EGY SZAM BEOLVASASA ?  12.348
12.35
KIIRATASI FORMATUM ? STOP
```

Mivel a formátumban csak két tizedesjegynek van helye, a gép kerekített.

** Ha két csillagot írunk a formátummező elejére, akkor a gép a tizedesponttól balra levő be nem töltött helyekre csillagokat ír. A két csillag beírásával két pozícióval többet kapunk.

$$ Ha két dollárjelet írunk a formátummező elejére, akkor a gép a legnagyobb helyiértékű számjegy elé egy $ jelet ír.

** $ Ha két csillagot és egy dollárjelet írunk, akkor minden üres helyre csillag, a legnagyobb helyiértékű szám elé pedig $ jel kerül.

Indítsuk ismét az előző programot:

```
READY
> RUN
KIIRATASI FORMATUM ? ** ##.##
EGY SZAM BEOLVASASA ?  12.3
** 12.30
KIIRATASI FORMATUM ? $$ ##.##
EGY SZAM BEOLVASASA ?  12.34
  $ 12.34
KIIRATASI FORMATUM ? ** $ ###.##
EGY SZAM BEOLVASASA ?  12.34
** $ 12.34
KIIRATASI FORMATUM ? STOP
```

Példa az egyéb karakterek alkalmazására:

```
? AZ ERTEK: ###.#
? 12.47
AZ ERTEK: 12.5
```

+ : ha a jelet írjuk a formátummező elejére illetve végére, akkor a gép + jelet ír a pozitív és – jelet a negatív számok elé, illetve azok után

– : ha a – jelet írjuk a formátummező végére, akkor a gép minden pozitív szám után kihagy egy karaktert és – jelet ír minden negatív szám mögé

Példa:

```
READY
> RUN
KIIRATASI FORMATUM ? + ##.##
EGY SZAM BEOLVASASA ?  12.34
+12.34
KIIRATASI FORMATUM ? + ##.##
EGY SZAM BEOLVASASA ?  – 12.34
–12.34
KIIRATASI FORMATUM ? ##.##+
EGY SZAM BEOLVASASA ?  –12.34
12.34 –
KIIRATASI FORMATUM ?##.##–
EGY SZAM BEOLVASASA ?  12.34
12.34
KIIRATASI FORMATUM ?##.###
EGY SZAM BEOLVASASA ?  123456
% 123456.000
KIIRATASI FORMATUM ?     STOP
```

% betűköz(ök) % egy karakternél többet tartalmazó karaktermezőt definiál. Az így kialakított mező hossza: a % jelek között levő üres karakterek száma plusz kettő. Ha a két % között nincs betűköz, a mezőhossz két karakter. Ha csak egy karakternyi mezőhosszt akarunk definiálni, a ! jelet kell használni formátumként.

Példa:

```
10 INPUT "KIIRATASI FORMATUM";F$
20 IF F$ = "STOP" END
30 INPUT "STRING BEOLVASASA"; C$
40 PRINT USING F$; C$
50 GOTO 10
```

Ez a program hasonlóan működik, mint az előző, csak itt nem numerikus, hanem string változókat használtunk.

Indítsuk el a programot:

```
READY
> RUN
KIIRATASI FORMATUM ?!
STRING BEOLVASASA ?  ABCDE
A
KIIRATASI FORMATUM ? %  %
STRING BEOLVASASA ?  ABCDE
ABC
KIIRATASI FORMATUM ? %      %
STRING BEOLVASASA ?  ABCDEF
ABCDE
KIIRATASI FORMATUM ? STOP
```

Az előző két formátum (% betűköz %) segítségével stringek összefűzhetők.

Példa:

```
10 INPUT "HAROM STRING"; A$, B$, C$
20 PRINT "EREDMENY:" ; : PRINT USING "!" ; A$; B$; C$
30 END
```

Futtassuk a programot!

```
READY
> RUN
HAROM STRING ?  ABC, XYZ, IJK
EREDMENY: AXI
HAROM STRING ?  A, PROGRAM, ULA
EREDMENY: APU
```

Ha az előző program 20 sorába a ! helyére % %-ot írunk, akkor a következő eredményt kaphatjuk:

```
READY
>RUN
HAROM STRING ? ABC, XYZ, IJK
EREDMENY: ABXYIJ
```

Ha több felkiáltójelet használunk, akkor a gép a string első betűjét annyi karakterrel később nyomtatja ki, ahány üres karaktert a felkiáltójelek **elé** írtunk.

```
10 INPUT "HAROM STRING"; A$, B$, C$
20 PRINT "EREDMENY:" ; : PRINT USING" ! ! !" ; A$; B$; C$
30 END
READY
> RUN
HAROM STRING ?  XYZ, FGH, ABC
EREDMENY: X F A
HAROM STRING ?  A,PROGRAM, ULA
EREDMENY: A P U
```

5. INPUT ("szöveg") változó sorozat

Az utasítás hatására a számítógép vár, míg a felhasználó bizonyos típusú és számú adatot be nem ír. A beírandó adatok numerikus értékek és alfanumerikus stringek lehetnek, a változók típusától függően. Minden változót (ha többet írunk be) vesszővel kell elválasztani.

Példa: 10 INPUT A$, B$, A, B

Ezen utasítás végrehajtásakor a felhasználó két stringet és két számértéket írhat be. A beírások sorrendjére figyelni kell. Az INPUT utasítás végrehajtásakor a számítógép egy kérdőjelet ír a képernyőre és várja az adato(ka)t.

Stringek beírásánál csak akkor kell azokat idézőjelbe tenni, ha vesszőt vagy kettőspontot tartalmaznak, vagy ha betűközzel kívánjuk azokat kezdeni.

Egy sorba írhatjuk mind a négy értéket, vesszővel elválasztva azokat. Ebben az esetben következőképpen nézhet ki a beírás:

```
BANAN, ALMA, 59,3.14    NEW LINE
```

A számítógép a következő hozzárendeléseket végzi:

```
A$ = "BANAN"
B$ = "ALMA"
A = 59
B = 3.14
```

Egy másik módszer ezen adatok beírására az, hogy minden adatot külön sorban írunk be.

A NEW LINE billentyű minden lenyomása után a számítógép a ?? (két kérdőjel) kiírásával arra emlékeztet, hogy még adatokat vár, mindaddig, amíg az összes adatot meg nem kapta. Ezután a következő utasításra ugrik. A beírt adatoknak (beírásoknak) a változó típusának megfelelőnek kell lenniük. Számváltozóba pl. nem írhatunk be stringet. Ha ilyesmit kísérlünk meg, akkor erre a számítógép a

```
? REDO
?
```

sorokkal válaszol, és új adatot kér abba a változóba, amelynek típusa nem egyezett meg az előzőleg beírtéval.

Példa:

```
10 INPUT A$, A
20 PRINT A$, A
30 END
READY
> RUN
? STRING, 10
STRING                10
READY
> RUN
```

```
? EZ EGY STRING, 13.5
EZ EGY STRING    13.5
READY
> RUN
? ABCD, IJK
? REDO
? ABCDE
?? 25
ABCDE            25
```

Annak érdekében, hogy egyértelmű legyen a kezelő számára, hogy a számítógép éppen melyik adatot várja, mindegyik INPUT utasításhoz olyan szöveget írhatunk, melyet a számítógép a kérdőjel leírása előtt kinyomtat.

A szövegnek közvetlenül az INPUT utasítás mögött kell következnie idézőjelek között, és le kell zárni pontosvesszővel.

Példa:

```
100 INPUT "TETEL NEVE ES MENNYISEGE"; N$, Q
READY
> RUN
TETEL NEVE ES MENNYISEGE?
```

6. DATA tételsorozat

Ez az utasítás lehetővé teszi a felhasználónak, hogy olyan adatokat tároljon a programon belül, amelyeket a READ utasítással érhet el. Az adatokat a számítógép sorban egymás után, balról jobbra haladva veszi elő. RESTORE vagy RUN indítás után az első adat kerül beolvasásra. Minden tétel string és numerikus változó is lehet. Ugyanúgy mint a billentyűzeten történő beíráskor, a betűközzel kezdődő, illetve a kettőspontot vagy vesszőket tartalmazó stringeket idézőjelek közé kell tenni.

A DATA utasításban levő tételek rendezésekor ügyelni kell arra, hogy az egyes változók típusa a DATA és a hozzá tartozó READ utasításban azonos legyen. A DATA utasítás a program tetszőleges helyén lehet.

Példa:

```
10 READ   A$, B$, C,D
20 PRINT  A$, B$, C,D
30 DATA   ALMA, KÖRTE, 4,8
40 END
READY
> RUN
ALMA              KÖRTE              4              8
```

7. READ tételsorozat

Ez az utasítás lehetővé teszi a DATA utasítás adatainak kiolvasását és azok változókhoz történő hozzárendelését. A DATA utasítás értékeit a READ utasítás sorrendben beolvassa. Ha az első DATA utasítás adatait már kiolvastuk, akkor a következő READ utasítás a következő DATA utasítás első adatát olvassa ki. Ha az összes adatot kiolvastuk és READ utasítással további értékeket akarunk elérni, akkor a gép a „nincs több adat" (out of data error) hibajelzést adja (kódja: OD).

Példa:

```
10 READ C$
20 IF C$ = "EOF" GOTO 60
30 READ Q
40 PRINT C$, Q
50 GOTO 10
60 PRINT: PRINT "LISTA VEGE.": END
70 DATA KONYVEK, 4, TOLLAK, 20
80 DATA LABDA, 8, ABLAKOK, 3, EOF
```

```
READY          4
> RUN          20
KONYVEK        8
TOLLAK         3
LABDA
ABLAKOK
LISTA VÉGE
```

8. RESTORE

Ez az utasítás lehetővé teszi azt, hogy a következő READ utasítással az első DATA lista első tételét olvassuk ki akkor is, ha már voltak ezelőtt READ utasítások.

Példa:

```
10 READ A$, A
20 PRINT A$, A
30 RESTORE
40 READ B$, B
50 PRINT A$, A, B$, B
60 DATA "KATI", 13, "PETER", 20, "MARTA" 28
70 END
READY
> RUN
KATI           13
KATI           13              KATI           13
```

Ez a példa azt is mutatja, hogy a RESTORE utasítás az addig elvégzett hozzárendeléseken sem változtat.

9. PRINT # — kazettaszám, tételsorozat

Ez az utasítás felveszi a kazettára a megadott változók vagy konstansok értékét. E program végrehajtása előtt a magnetofont "felvétel" üzemmódba kell állítani. Közelebbi információt erről a Használati útmutatóban találhat. A rendszer két magnetofont tud kezelni, a felhasználó kazettaszámokkal adhatja meg, hogy melyiket akarja használni.

Példa:

```
10 A$ = "FELVETEL INDUL"
20 B = 3.1416
30 C = 50
40 D$ = "DATA"
50 PRINT #—1, A$, B, C, D$, "FILE VEGE"
60 END
```

Ez a program először értékeket ad az A$, B, C és a D$ változóknak, majd ezeket felveszi az 1. sz. magnetofonra A FILE VEGE stringet a gép ugyanúgy rögzíti a szalagon mint a változókat. Ha már felvettük az adatokat, akkor azok visszaolvashatók. Alapjában véve ez ugyanaz, mint amikor zenét veszünk fel magnóra, majd lejátszunk.

Ügyeljünk arra, hogy a visszaolvasáskor az adatok típusa megegyezzen a PRINT utasításban meghatározottakkal. Beolvasáskor a változók nevei természetesen különbözők lehetnek.

FIGYELEM! A tétellistában szereplő változók által képviselt karakterek száma nem lehet több, mint 255. Ha 255 karakternél többet adunk meg, akkor a 255-ön felül levőkről a gép nem vesz tudomást.

Példa:

```
10 PRINT #—1, A$, B$, C$, D$, E$
```

Tegyük fel, hogy az A$, B$, C$ és D$ változókban levő karakterek összege 250, és az E$ 35 karakter.

E$-t ekkor nem rögzíti a gép a szalagra, és ha később megpróbáljuk kiolvasni, akkor a „nincs több adat" hibajelzést kapunk (out of data; kód: OD).

10. INPUT # – kazettaszám, tételsorozat

Ezen utasítás hatására a számítógép kiolvassa a szalagon levő adatokat és beírja azokat a tételsorozat változóiba.

Példa:

```
10 INPUT #–1, A$, B, C, D$
```

Ez az utasítás az 1. sz. magnetofon adatait olvassa ki. Az első értéket az A$-hoz, a másodikat B-hez rendeli és így tovább. A magnetofonon a PLAY billentyűnek benyomva kell lennie. A gép az utasítás végrehajtása során bekapcsolja a magnetofont, mikor kész van, akkor kikapcsolja és a következő utasításra ugrik. Ha az INPUT-tal a szalagon levő stringet numerikus változóba akarjuk beolvastatni, akkor „rossz file" (bad file; kód: FD) hibajelzést kapunk „Nincs több adat" hibajelzést kapunk, ha a szalagon nincs a tétellista összes változója számára adat.
Az INPUT tételsorozatnak azonosnak kell lennie a szalagon lévő adatokat felvitt PRINT utasítás tételsorozatával (ugyanazon sorrendben ugyanannyi és ugyanolyan típusú változó).

PROGRAMUTASÍTÁSOK

11. DEFINT betűkészlet

Az ezen betűkészlet betűivel kezdődő változókat a gép egész típusú változókként kezeli és tárolja.

Típusmegadással (%, $, stb.) felülírható ez a típusmeghatározás. Egész értékű változók (integer) használatával nemcsak tárolókapacitást, hanem gépidőt is megtakaríthatunk, mert az egész típusú változóval a gép gyorsabban számol, mint az egyszeres vagy kétszeres pontosságú változókkal. Ügyeljünk arra, hogy az egész típusú változó értéke csak –32768 és 32767 között lehet.

Példa:

```
10 DEFINT X, Y, Z
```

A 10-es sor végrehajtása után a gép az X-, Y-, Z-vel kezdődő minden változót egészként kezel. Ezért ettől kezdve az X2, X3, YA, YB, ZI, ZJ egész típusú változók. Azonban az X1#, YA#, YB# kétszeres pontosságú változók maradnak, mert a „#"-vel történő típusmeghatározás felülírja a DEFINT-vel történő típusmeghatározást.

Példa:

```
10 DEFINT A–D
```

Ez az utasítás minden A, B, C vagy D-vel kezdődő változót egésznek nyilvánít.

Figyeljünk arra, hogy ugyan a DEFINT a program tetszőleges sorában elhelyezhető, azonban ellenőrzés nélkül megváltoztathatja az explicit típusdeklaráció ($, #, %) nélküli változók jelentését (ugráshiba). Ezért általában legjobb, ha ezt az utasítást a program elejére tesszük.

12. DEFSNG betűkészlet

Az ezen betűkészlet betűivel kezdődő változókat a gép egyszeres pontosságú változókként kezeli. Explicit típusmegadás (#, $, %) felülírja ezt a meghatározást. Az egyszeres pontosságú változókat és konstansokat a gép hét számjeggyel kezeli és hattal jeleníti meg.

Ha másként nem definiáltuk, akkor minden számváltozó egyszeres pontosságú. A DEFSNG utasítás elsősorban arra szolgál, hogy a korábban kétszeres pontosságúnak vagy egésznek definiált változókat újra definiálhassuk.

Példa:

```
10 DEFSNG A–D, Y
```

Minden, A-tól D-ig terjedő betűvel, ill. Y-nal kezdődő változó egyszeres pontosságú lesz ezen utasítás hatására. Azonban A # korábbra is kétszeres pontosságú, Y % pedig továbbra is egész változó marad.

13. DEFDBL betűkészlet

Az ezen betűkészlet betűivel kezdődő változókat a gép kétszeres pontosságú változókként kezeli és tárolja. Explicit típusmegadással (!, $, %-kal) ez a definiálás felülírható. A kétszeres pontosságú változókkal a számolás 17 számjeggyel lehetséges, amelyből a gép 16-ot jelenít meg.

Példa:

```
10 DEFDBL M–P, G
```

Minden, M-től P-ig terjedő betűvel, ill. G-vel kezdődő változó ezentúl kétszeres pontosságú lesz. (Nem befolyásolja az utasítás az M % és a G # típusát.)

14. DEFSTR betűkészlet

Az ezen betűkészlet betűivel kezdődő változókat a gép string változókként kezeli és tárolja.

Explicit típusmegadás (#, !, %) felülírja ezt a definiálást. Ha elég hely áll a string rendelkezésére (lásd CLEAR), akkor a string legnagyobb hossza 255 karakter lehet.

Példa:

```
10 DEFSTR A–D
```

Minden, A-tól D-ig levő betűvel kezdődő változó string lesz, azokon kívül, amelyek típusa explicit típusmegadással (#, !, %) lett meghatározva.

A 10-es sor végrehajtása után a gép elfogadja a következő kifejezést: B3 = "EGY STRING".

15. CLEAR n

Ez az utasítás minden változót nulláz (nulla értéket rendel minden változóhoz).

Ha megadunk egy „n" számot, akkor a számítógép stringek tárolására n byte-ot felszabadít. A rendszer bekapcsolásakor a gép minden alkalommal 50 byte-ot tesz stringek tárolására szabaddá. A CLEAR akkor válik kritikussá, ha a számítógép a programvégrehajtás során „nincs több hely stringek számára" (out of stringspace; kódja: OS) hibába ütközik. Ez a hiba akkor lép fel, ha a program stringjei több helyet igényelnek a szabaddá tettnél.

Példa:

```
10 CLEAR 1000
```

Ez az utasítás szabaddá tesz 1000 byte-ot stringek számára.

16. DIM név (dim 1, dim 2, ... dim n)

Ez az utasítás egy változót, vagy változók egy listáját tömbként deklarálja. A tömb egyes vektoraiban az elemek számát a dim 1, dim 2, stb.-vel adhatjuk meg. Ha dim n t nem adjuk meg, akkor a gép feltételezi, hogy minden vektor 11 elemű.

A vektorok lehetséges számát (a tömb dimenzióját) csak a tárolóhely nagysága korlátozza.

Példa:

```
10 DIM A(5), B(3,4), C(2,3,3)
```

Ez az utasítás A-t 6 elemű (0-tól 5-ig) egydimenziós tömbként (vektorként), B-t 20 elemű (4 x 5) kétdimenziós tömbként, C-t pedig 48 elemű (3 x 4 x 4) háromdimenziós tömbként definiálja.

DIM utasítást a program tetszőleges részén elhelyezhetünk. A dimenzió megadása egész számmal, vagy kifejezéssel történhet.

```
10 INPUT "VEKTOROK SZAMA"; N
20 DIM A (N+2,4)
```

Az A tömb sorainak száma N értéknek megfelelően változhat. Egy tömb újradimenzionálása előtt CLEAR utasítást kell adni — akár argumentummal, akár argumentum nélkül — különben a gép hibát jelez.

Példa:

```
10 X(2) = 13.6
20 PRINT "MASODIK ELEM:" ; X(2)
30 DIM X(15)
40 PRINT X(2)
```

```
50 END
READY
> RUN
MASODIK ELEM: 13.6
? DD ERROR IN 30
```

17. LET változó = kifejezés

Ez az utasítás egy változóhoz egy értéket rendel. Ilyen hozzárendelő utasításban a LET szó nem feltétlenül szükséges a mi rendszerünkben, de szerepel az utasításkészletben, hogy más rendszerekkel kompatibilis legyen.

Példa:

```
10 LET A = 5.67
20 B % = 20
30 S$ = "BETUK"
40 LET D % = D % + 1
50 PRINT A, B %, S$, D %
60 END
READY
> RUN
5.67            20            BETUK         1
```

A fenti hozzárendelésekben az egyenlőség bal oldalán álló változóba a jobb oldalon levő kifejezés értéke íródik be.

18. END

Ez az utasítás a programvégrehajtás befejezésére használatos. Az END utasítás elsősorban a program logikai végét jelzi.

Példa:

```
 5 B = 3 : C = 14
10 A = C+B
20 GOSUB 70
30 D = X+Y
40 PRINT "AZ EREDMENYEK:";
50 PRINT A,D
60 END
70 X=50
80 Y = A*X
90 RETURN
>RUN
AZ EREDMENYEK: 17          900
```

A 60. sorban levő END utasítás miatt a számítógép a 70–90. sorokat csak a 20. sorban levő szubrutinhívás miatt fogja végrehajtani.

19. STOP

Ez az utasítás a program hibáinak megkereséséhez vagy a program egyéb okból történő megszakításához nyújt segítséget. Töréspontot (break point) helyez el a programvégrehajtásban és lehetővé teszi értékek megváltoztatását vagy kiírását. STOP utasítás végrehajtásakor a gép kiírja a BREAK IN LINE sorszám (Megszakítás a . . . sorban) üzenetet.

A CONT paranccsal a program végrehajtása folytatható, attól a ponttól, ahol korábban az meg lett szakítva.

Példa:

```
 5 INPUT B,C
10 A = B+C
20 STOP
30 X = (A+D)/0.74
40 IF X < 0 GOTO 70
```

```
50 PRINT A,B,C
60 PRINT X
70 END
READY
> RUN
?2,4
BREAK IN 20
READY
> PRINT A
  6
READY
> CONT
6                2                4
8.10811
```

A STOP utasítással lehetővé vált az, hogy az A értékét a 30. sor végrehajtása előtt megnézzük.

20. GOTO sorszám

Ennek az utasításnak a hatására a program végrehajtása a megadott soron folytatódik (ugró utasítás). Ha önmagában használjuk, akkor a gép feltétlen ugrást hajt végre. A GOTO elé vizsgálatot írhatunk, ha az ugrást feltételessé akarjuk tenni.

Példa:

```
10 A = 10
20 B = 45
30 C = A+B
40 C = C * 3.4
50 GOTO 100
60 .
70 .
80 .
```

```
 90 .
100 PRINT "A="; A, "B="; B, "C="; C
110 END
READY
> RUN
A=10                B=45                C=187
```

A gép az 50. sor után a 100-as sort hajtja végre.

Példa:

```
10 IF A = 2 GOTO 120
```

Ha a gép végrehajtja a 10-es sort és A egyenlő kettővel, akkor a számítógép a 120 sorra ugrik. Ha A nem egyenlő 2-vel, akkor a gép a soron következő utasítást hajtja végre.

A GOTO utasítás parancs üzemmódban is használható RUN helyett. Ilyenkor a számítógép nem törli a változókat.

21. GOSUB sorszám

Arra a sorra adja át a vezérlést, amelyiken a megadott szubrutin kezdődik. Ha a számítógép a szubrutinban RETURN utasítást talál, akkor visszatér a GOSUB utasítást követő utasításra. A GOTO-hoz hasonlóan itt is megadhatunk feltételt, mint a következő példában.

Példa:

```
10 PRINT "FOPROGRAM"
20 GOSUB 50
30 PRINT "PROGRAM VEGE"
40 END
50 PRINT "SZUBRUTIN"
60 RETURN
READY
> RUN
```

```
FOPROGRAM
SZUBRUTIN
PROGRAM VEGE
```

22. RETURN

Ez az utasítás befejezi a szubrutin végrehajtását és visszaadja a vezérlést a GOSUB után következő utasításra. A gép hibajelzést ad, ha a RETURN utasításhoz nincs előzetes GOSUB.

23. ON n GOTO sorszámlista

Ez az utasítás lehetővé teszi több ugrási cél megadását, mégpedig az n-től függően. Az ON n GOTO teljes formátuma:

ON kifejezés GOTO 1. sorszám, 2. sorszám . . .

A kifejezés értékének 0 és 255 közöttinek kell lenni. ON-GOTO utasítás végrehajtásakor a gép először a kifejezés egész értékét számolja ki (ez megfelel az INT (kifejezés)-nek). Legyen az eredmény n. Ezután a számítógép megkeresi a sorszámlista n-dik elemét és erre a sorra ugrik. Ha n nagyobb, mint a megadott sorszámok száma, akkor a gép az ON-GOTO utasítás után következő utasítást hajtja végre. Ha n negatív, akkor ez hibát jelent. A sorszámlista sorszámokat tetszőleges számban tartalmazhat. Ha n=0, akkor a következő programsoron folytatódik a program végrehajtása.

Példa:

```
 10 INPUT "SZAM"; C
 20 ON C GOTO 100, 120, 130, 150, 130
 30 PRINT "VEGE": END
100 PRINT "100-AS SOR" : GOTO 10
120 PRINT "120-AS SOR" : GOTO 10
130 PRINT "130-AS SOR" : GOTO 10
```

```
150 PRINT "150-ES SOR" : GOTO 10
READY
> RUN
SZAM ? 5
130-AS SOR
SZAM ? 4
150-ES SOR
SZAM ? 1
100-AS SOR
SZAM ? 2
120-AS SOR
SZAM ? 3
130-AS SOR
SZAM ? 0
VEGE
READY
> RUN
SZAM ? 4
150-ES SOR
SZAM ? 6
VEGE
```

Az ON-GOTO utasítás néha elegánsabb módszernek bizonyul, mint az IF-GOTO utasításkombináció:

```
10 IF C=1 GOTO 100
20 IF C=2 GOTO 120
30 IF C=3 GOTO 130
40 IF C=4 GOTO 150
50 IF C=5 GOTO 130
60 IF C < 1 OR C > 5 GOTO 70
70 . . .
```

24. ON n GOSUB sorszámlista

Ugyanúgy működik, mint az ON n GOTO, csak ugrás helyett a gép egy szubrutint hajt végre.

Példa:

```
 10 PRINT "** FUGGVENY RUTINOK **"
 20 PRINT "   A FUGGVENY"
 30 PRINT "   B FUGGVENY"
 40 PRINT "   C FUGGVENY"
 50 INPUT "SORSZAM 1,2 VAGY 3"; N
 60 ON N GOSUB 150, 100, 250
 70 END
100 PRINT "EZ AZ A FUGGVENY": RETURN
150 PRINT "EZ A B FUGGVENY": RETURN
250 PRINT "EZ A C FUGGVENY": RETURN
READY
> RUN
** FUGGVENY RUTINOK **
   A FUGGVENY
   B FUGGVENY
   C FUGGVENY
SORSZAM 1,2 VAGY 3 ? 2
EZ AZ A FUGGVENY
READY
> RUN
** FUGGVENY RUTINOK **
   A FUGGVENY
   B FUGGVENY
   C FUGGVENY
SORSZAM 1,2 VAGY 3? 1
EZ A B FUGGVENY
```

25. FOR név = kifejezés TO kifejezés STEP kifejezés
NEXT név

Ezek az utasítások egy iteratív ciklust hoznak létre. A gép minden, a FOR és a NEXT között álló utasítást a paramétereiben meghatározott alkalommal hajt végre.

Az általános formátum:

```
FOR számláló = kezdeti érték TO végérték STEP lépésköz
*
*
*
utasítások
*
*
*
NEXT számláló
```

A STEP kulcsszó a lépésközzel együtt elhagyható, ekkor a lépésköz értéke 1 lesz.
A FOR utasításnál a kezdőérték, a végérték, és a lépésköz állandó, változó vagy kifejezés lehet. A FOR utasítás első végrehajtásakor a gép ezeket kiszámolja és eltárolja. Ha ezek az értékek a ciklusban megváltoznak, akkor ez magára a ciklus működésére nincs hatással, azonban a számláló értékének megváltoztatása a ciklus működésének megváltozását eredményezheti.

A FOR-NEXT ciklus a következőképpen működik.
A FOR utasítás első végrehajtásakor a gép a számlálót a kezdőértékre állítja be. A programot a gép a következő NEXT utasításig végrehajtja. Itt a gép a számlálót a lépésköz értékével megnöveli. Ha azt nem adtuk meg, a számítógép automatikusan 1-et állít be. Ha a lépésköz negatív, akkor a számláló visszafelé számlál. Ezután a gép a FOR utasítás végértékével összehasonlítja a számlálót. Ha a számláló nagyobb, mint a végérték, akkor a ciklus végrehajtása leáll és a programvégrehajtás a NEXT utasítás után folytatódik. (Ha a növekmény negatív, úgy akkor fejeződik be a végrehajtás, ha a számláló kisebb a végértéknél.)

Ha a számláló még nem lépte túl a végértéket, akkor a számítógép a FOR utasítás utáni első utasításnál folytatja a program végrehajtását.

Példa:

```
10 FOR K=0 TO 1 STEP 0.3
20 PRINT "K ERTEKE:" ; K
30 NEXT K
40 END
READY
> RUN
K ERTEKE: 0
K ERTEKE: .3
K ERTEKE: .6
K ERTEKE: .9
```

A következő alkalommal K = 1.2 és ez nagyobb, mint a végérték (1) Ezért a ciklus itt befejeződik és az 1.2 már nem íródik ki.

Példa:

```
10 FOR N=3 TO 0
20 PRINT "N=" ; N
30 NEXT N
40 END
READY
> RUN
N=3
10 FOR N=3 TO 0 STEP -1
20 PRINT "N=" ; N
30 NEXT N
40 END
READY
> RUN
N=3
N=2
N=1
N=0
```

Mivel nem adtunk meg STEP-et, a gép automatikusan STEP 1-et állít be Ezért az első esetben a gép N-et 1-gyel növeli, és mivel az így számított N=4 nagyobb, mint a 3 végérték; a ciklus befejeződik. A folyamat másképp történik, ha STEP–1-et megadtuk (lásd az előző példát).

Példa:

```
10 FOR A=0 TO 3
20 PRINT "A ERTEKE:" ; A
30 NEXT
40 END
READY
> RUN
A ERTEKE: 0
A ERTEKE: 1
A ERTEKE: 2
A ERTEKE: 3
```

A NEXT A helyett, mint a 30-as sorban, egyszerűen NEXT et is írhatunk. Egymásba ágyazott (skatulyázott) FOR NEXT ciklusok programozásakor azonban célszerű megadni azt, hogy a konkrét NEXT-tel éppen melyik ciklust zártuk le.

A következőkben mutatunk egy példát egymásba ágyazott ciklusokra.

Példa:

```
10 I =1
20 J=2
30 K=3
40 FOR N = I+1 TO J+1
50      PRINT "ELSO HUROK"
60      FOR M=I TO K
70      PRINT  "  CICA"
80      NEXT M
90 NEXT N
```

```
100 END
READY
> RUN
ELSO HUROK
   CICA
   CICA
   CICA
ELSO HUROK
   CICA
   CICA
   CICA
```

A program 80 és 90 sorát összevonhatjuk egy sorrá a következő módon:

80 NEXT M, N

de ez a program áttekinthetőségét csökkenti.

26. ERROR hibakód

Ezt az utasítást ON ERROR GOTO rutin teszteléséhez használhatjuk. ERROR hibakód utasítás végrehajtásakor a számítógép pontosan úgy viselkedik, mintha az a bizonyos hiba fellépett volna; szimulálja a hibát.

Példa:

```
30 ERROR 1
? NF ERROR IN 30
```

Valamennyi ERROR (=hiba) kód jelentését a B függelékben találjuk.

27. ON ERROR GOTO sorszám

Ez az utasítás lehetővé teszi, hogy „hibaelfogó-programot" hozzunk létre (angolul: error trapping routine); ennek segítségével elérhetjük, hogy ha a program végrehajtása során a számítógép hibát talál, akkor ne hibajelzést adjon, és ne állítsa meg a program futását, hanem egy olyan programrészbe ágazzon el, amely ismeri a hiba okát és megszünteti azt.

Legtöbbször egy bizonyos típusú hibát tart a felhasználó szem előtt, mikor az ON ERROR GOTO utasítást végrehajtja. Például, ha a program osztást hajt végre és nincs kizárva a nullával való osztás lehetősége, akkor, ha ez bekövetkezik, egy hibakezelő program léphet életbe.

Példa:

```
  5 B=15:C = 0
 10 ON ERROR GOTO 120
 20 A=B/C
 30 PRINT A,B,C
 40 END
120 PRINT "NULLAVAL OSZTAS!!!"
130 END
READY
> RUN
NULLAVAL OSZTAS !!!
```

Ebben a példában C értéke zérus, ami a 20-as sorban nullával való osztáshoz vezet, ami normális körülmények között hibajelzést és a program leállását eredményezné. A 10-es sor miatt a gép ezt a szituációt azonban nem veszi tudomásul és a 120-as sorra lép, a hibakezelő rutinra. Figyeljünk arra, hogy az ON ERROR GOTO utasítás a hiba lehetséges előfordulása előtt álljon, másképp ugyanis nincs hatása. Ezenkívül a hibakezelő rutint RESUME-vel le kell zárni.

28. RESUME sorszám

Ez az utasítás befejezi a hibakezelő rutint és jelzi, hogy a rendes programvégrehajtásnak hol kell folytatódnia.

RESUME 0, vagy RESUME sorszám nélkül utasítás hatására a számítógép ahhoz az utasításhoz tér vissza, amely a hibához vezetett. Ha megadunk egy sorszámot, akkor a számítógép a megadott sorra tér vissza. RESUME NEXT arra utasítja a számítógépet, hogy a hibát okozó sor utáni sorra térjen vissza.

Példa:

```
 10 ON ERROR GOTO 80
 20 PRINT "OSZTAS"
 30 INPUT "KET SZAM"; A,B
 40 IF A=0 END
 50 C = A/B
 60 PRINT "A HANYADOS:" ; C
 70 GOTO 20
 80 PRINT "EJNYE! AZ OSZTO NULLA"
 90 PRINT "PROBALKOZZ UJRA..."
100 RESUME 20
READY
> RUN
OSZTAS
KET SZAM ? 6,2
A HANYADOS: 3
OSZTAS
KET SZAM ? 0,4
READY
> RUN
OSZTAS
KET SZAM ? 3,0
EJNYE! AZ OSZTO NULLA
PROBALKOZZ UJRA ...
```

29. REM

A REM megjegyzések beírását teszi lehetővé. Ez az utasítás arról tájékoztatja a számítógépet, hogy ebben a sorban csak kommentárok következnek. Ezeket a gép a program futásakor nem veszi figyelembe. A REM lehetővé teszi a felhasználónak, hogy magyarázatokkal áttekinthetőbb programot írjon.
A REM kulcsszó helyett használhatjuk a ' (aposztrof) jelet is.

30. IF kifejezés THEN művelet

Az IF arra utasítja a számítógépet, hogy vizsgáljon meg egy reláció- vagy logikai kifejezést. Ha a kifejezés „igaz", akkor a gép a műveletet végrehajtja, ha „hamis", akkor a műveletet nem veszi figyelembe és folytatja a programvégrehajtást a következő utasítással. Az „igaz" kifejezés numerikus értéke —1, a „hamis"-é nulla.

Példa:

```
10 INPUT "ERTEK (MAX.20) BEVITELE"; A
20 IF A > 20 GOTO 60
30 A= A * 3.1416 * 2
40 PRINT "A KERULET:" ; A
50 END
60 PRINT "AZ ERTEK TUL NAGY!  (MAX. 20)" : GOTO 10
READY
> RUN
ERTEK (MAX. 20) BEVITELE ? 24
AZ ERTEK TUL NAGY! (MAX. 20)
ERTEK (MAX. 20) BEVITELE ? 18
A KERULET: 113.098
```

Ebben a példában minden alkalommal, amikor 20-nál nagyobb számot írtunk be, figyelmeztet a gép és új adatot vár. Ha A értéke kisebb, mint 20 vagy egyenlő 20-al, akkor a számítógép a 20-as sorban levő GOTO 60 utasítást nem veszi figyelembe és befejezi a számolást figyelmeztető jelzés nélkül.

```
120 INPUT A: IF A = 10 AND A > B THEN 160
120 INPUT A: IF A = 10 AND A > B GOTO 160
```

Mindkét utasításnak ugyanaz a hatása.
Az utasításból a THEN elhagyható.

31. THEN utasítás vagy sorszám

A THEN utasítás az IF-THEN típusú utasítások feltételrészének végét jelzi. A THEN használata nem kötelező, kivéve, ha ugrási cél sorszámának megadásához használjuk, mint az IF A = B THEN 100. THEN-t kell IF-THEN-ELSE utasításban is megadni.

32. ELSE utasítás, vagy sorszám

Ez az utasítás csak IF utasítás után állhat. Ha az IF utasításban a kifejezés „hamis", akkor a gép az ELSE után következő sorszámra ugrik ill. az ELSE után álló utasítást hajtja végre.

Pl.: 10 IF A = 1 THEN 60 ELSE 40

Ebben a példában a gép a 60-ra ugrik, ha A=1 és a 40-re, ha nem. IF-THEN-ELSE utasítások egymásba ágyazhatók, de az IF-ek és az ELSE-ek számának meg kell egyezni.

Példa:

```
10 INPUT "ADJ NEKEM HAROM SZAMOT"; X,Y,Z
20 PRINT "A LEGNAGYOBB SZAM:" ;
30 IF X < Y OR X < Z THEN IF Y < Z THEN PRINT Z
   ELSE PRINT Y ELSE PRINT X
40 END
READY
> RUN
ADJ NEKEM HAROM SZAMOT ? 30,75,73
A LEGNAGYOBB SZAM: 75
```

A program három számot vár (mint beírt adatot) és kiírja a legnagyobbat.

33. LPRINT

Az LPRINT adatokat ír ki a nyomtatóra. Ez az utasítás (amely egyben parancs is) ugyanúgy működik, mint a PRINT utasítás. Ha nincs a számítógéphez nyomtató csatlakoztatva, akkor a számítógép ezen program végrehajtásakor végtelen ciklusba kerül, várva az első karakter kinyomtatását, melyből csak a RESET gombbal hozható ki.

Általában elmondható, hogy minden BASIC utasítás – kivéve az INPUT és az INPUT # – használható parancs üzemmódban is. Ekkor az utasítás, illetve az utasítássorozat azonnal végrehajtásra kerül.

Példa:

```
>PRINT (2*4[2+SIN (0.5))     NEW LINE
  32.4794
READY
>A=5:B=10:PRINT A*B  NEW LINE
  50
READY
>
```

Egy BASIC programsor egy vagy több utasítást tartalmazhat. Utasításnak tekintjük az = jelet is (ez az értékadó utasítás). Több utasítást úgy írhatunk egy programsorba, hogy azokat a : jellel választjuk el egymástól.
Egy programsor maximális hossza 255 karakter, tehát a képernyőn több sort is elfoglalhat.

4

TÖMBÖK KEZELÉSE

A tömb egy rendezett adatlista. Az Ön számítógépe szám- és string-tömböket egyaránt tud kezelni. Egy tömbben azonban csak egyfajta adat lehet.

A számítógép alkalmazhatóságának szempontjából a tömbök programozása nagyon fontos, ezért figyelmesen tanulmányozza át az ebben a fejezetben bemutatott példákat!

Tegyük fel, hogy Kovács Ferenc egyetemen tanul. Az egyetem épülete háromemeletes. Minden emeleten négy terem van és minden teremben 45 ülőhely. Ferenc történelem előadáson van és őt is beleszámítva, csak 37 hallgató van a teremben.

NÉVSOR: 1. Ádám Mária
2. Barna János
3. Csengő Gábor
⋮
37. Kovács Ferenc

Ha valakinek a nevét meg akarjuk keresni, elég végigolvasni (pl. fentről lefelé) a névsort. Az a módszer, amellyel a nevet keressük, most nem különösebben fontos.

Fontos azonban az, hogy hogyan találunk meg egy nevet a listán, ha csak a számát ismerjük. A fenti névsorban Ádám Mária az első, Barna János a második, és így tovább. Tehát ezek a számok egy név megtalálásához szisztematikus hozzárendelést biztosítanak.

Ha a számítógépet a névsor felírásához használjuk, akkor minden névhez hozzárendelhetünk egy egységes változót.

```
10 N0$ = "ADAM MARIA"
20 N1$ = "BARNA JANOS"
30 N2$ = "CSENGO GABOR"
 ⋮
100 NS$ = "SZABO TAMAS"
 ⋮
140 NZ$ = "KOVACS FERENC"
```

Ha meggondoljuk, hogy 37 hallgató van a csoportban, akkor látjuk, hogy ez nem túl hatásos, és időrabló módszer. Ha jobban megnézzük a programot, megállapíthatjuk, hogy minden változó

mellé számok kerültek, növekvő sorrendben. Ezt egyszerűen, magától értetődően csináltuk így. Ha ezeket a futó számokat a számítógép hozza létre, akkor tömböt kapunk. Definiáljunk egy 45 elemű tömböt: AR$. Rendeljünk minden elemhez egy nevet.

```
 5 CLEAR 1000
10 DIM AR$ (44)
20 FOR N=0 TO 44
30 INPUT "A HALLGATO NEVE"; AR$ (N)
40 NEXT N
50 END
```

Ez a program beolvas 45 nevet és eltárolja őket az AR$ tömbben. A program végrehajtása után a következő változókat kapjuk.

```
Az AR$ ( 0 ) elem értéke "Ádám Mária"
Az AR$ ( 1 ) elem értéke "Barna János"
Az AR$ ( 3 ) elem értéke "Csengő Gábor"
        :
Az AR$ ( 36 ) elem értéke "Kovács Ferenc"
```

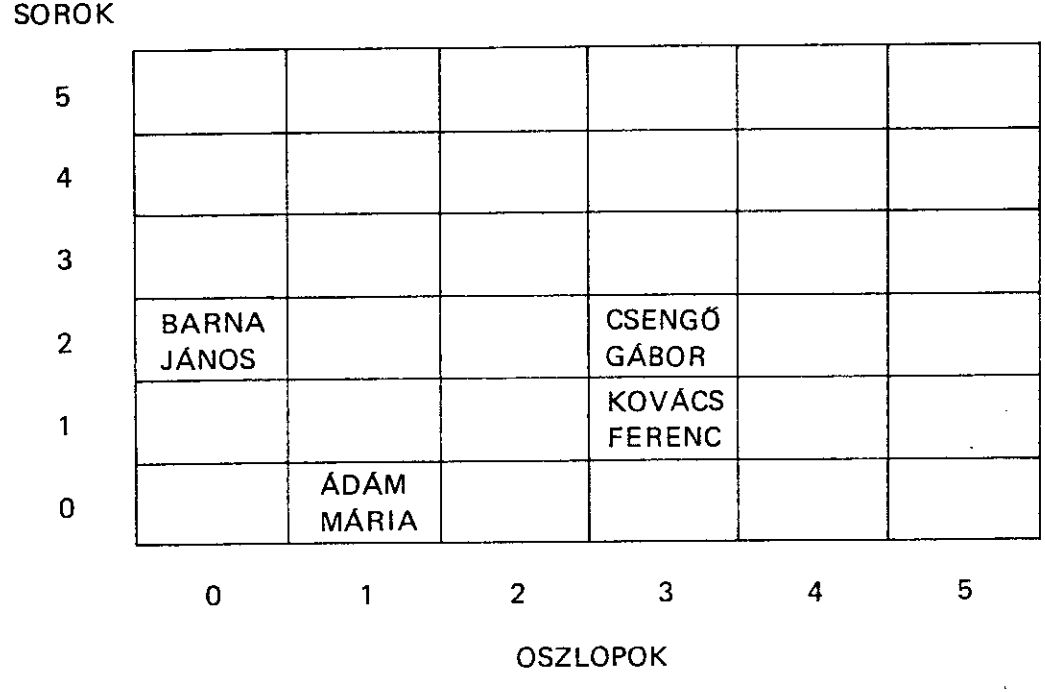
Ha a listát ki akarjuk nyomtatni, akkor a következő programot használhatjuk:

```
 5 CLEAR 1000
10 DIM AR$ (44)
20 FOR N=0 TO 44
30 INPUT "A HALLGATO NEVE"; AR$ (N)
40 NEXT N
50 FOR N=0 TO 44
60 PRINT AR$ (N)
70 NEXT N
80 END
```

A program kiíratórésze a következő 45 utasítást helyettesíti

```
10 PRINT N0$
20 PRINT N1$
30 PRINT N2$
40 PRINT N3$
 :
100 PRINT NS$
 :
140 PRINT NZ$
```

Tegyük fel, hogy az egyik tanár ülésrendet akar készíteni az osztálynak. 6 széksor van és ezek 6 oszlopot alkotnak (minden sorban 6 szék van).



A példában szereplő négy hallgató helye a fenti üléstervben látható. Mivel azonban ők a névsorban nem egymás után következnek, valamilyen más módszert kell a tanárnak kigondolni az ülésrend elkészítéséhez. Például, ha ellenőrizni akarja, hogy Kovács Ferenc jelen van-e, akkor meg kell néznie, hogy üres-e az. 1. sor 3. oszlopa vagy sem. Ellenőrizheti Csengő Gábor jelenlétét is a 2. sor 3. oszlopában. Nos, a számítógép pontosan úgy dolgozik, mint ez a tanár. Ezt az ülésrendet egy kétdimenziós tömbként [SP$ (5,5)] ábrázolhatjuk, ahol az első szám a sorok, a második az oszlopok száma. Ha például Barna Jánost akarjuk felszólítani, akkor őt az SP$ (2,0)-ban kell keresnünk. Nyomtassuk ki az ülésrendünket.

Ezt a következő programmal tehetjük meg:

```
10 CLEAR 1000: DIM SP$ (5,5)
20 FOR R=5 TO 0 STEP -1
25 PRINT TAB (0)
30      FOR C=0 TO 5
40          PRINT SP$ (R,C);
50      NEXT C
60 NEXT R
70 END
```

Ez a program táblázat formájában kinyomtatja az ülésrendet, a csoport utolsó sorával kezdve és az elsővel befejezve.

A program először inicializálja az R=5 és C=0-t, majd kinyomtatja az elemek értékeit.

SP$ (5,0); SP$ (5,1); SP$ (5,2); SP$ (5,3) . . . SP$ (5,5).

Ekkor öt lesz C értéke A számítógép a belső „C" ciklusról visszaugrik a külső „R" ciklusra. Levon R-ből egyet és ezzel R értéke 4 lesz. C ismét 0-ra állítódik és az eljárás újból elkezdődik az R=4-gyel.

SP$ (4,0; SP$ (4,1); SP$ (4,2) . . . SP$ (4,5)

A folyamat addig ismétlődik, amíg R=—1 nem lesz. Ekkor megáll a program. A végső lista így néz ki:

SP$ (5,0); SP$ (5,1); SP$ (5,2); SP$ (5,3); SP$ (5,4); SP$ (5,5)
SP$ (4,0); SP$ (4,1); SP$ (4,2); SP$ (4,3); SP$ (4,4); SP$ (4,5)
SP$ (3,0); SP$ (3,1); SP$ (3,2); SP$ (3,3); SP$ (3,4); SP$ (3,5)
SP$ (2,0); SP$ (2,1); SP$ (2,2); SP$ (2,3); SP$ (2,4); SP$ (2,5)
SP$ (1,0); SP$ (1,1); SP$ (1,2); SP$ (1,3); SP$ (1,4); SP$ (1,5)
SP$ (0,0); SP$ (0,1); SP$ (0,2); SP$ (0,3); SP$ (0,4); SP$ (0,5)

Egy ilyen kétdimenziós tömb segítségével a csoport valamennyi hallgatójának helyét rögzíthetjük. Hogyan lehetne annak a hallgatónak a helyét rögzíteni, aki ugyanott ül, mint Kovács Ferenc, csak egy másik teremben?

Ekkor természetesen a sor és az oszlop mellett a terem számát is meg kell adnunk, ezzel tömbünk új dimenziót kap, ugyanis 12 terem van az épületben. Különböző lehetőségek vannak ennek a problémának a megoldására. A második: a termeket emeletek szerint rendezzük; 1. terem az 1. emeleten, 1. terem a 2. emeleten stb. Ehhez azonban két további dimenzióra van szükségünk, összesen: sor, oszlop, terem, emelet.

Tegyük fel, hogy Ferenc terme a 2. emelet 3.

Az első módszerrel Ferenc helyét az SP$ (N,R,C)-vel adhatjuk meg, ahol N = teremszám, R = sorszám, C = oszlopszám. Barna János például az SP$ (7,2,0)-ban, azaz a 7-es terem 2. sor 0. oszlopában ül.

Ha a második módszert használjuk, akkor SP$ (F,N,R,C)-t kapunk, ahol F emeletszám, N teremszám ezen az emeleten, R sor, C oszlop. Ha Ferencet meg akarnánk szólítani, azt kellene mondanunk: SP$ (2,3,1,3), azaz 2. emelet, 3. terem, 1. sor, 3. oszlop.

A dimenziók száma növekszik, ha a rendszerünkbe újabb hallgatókat akarunk felvenni. További dimenziókat vezethetünk be az épület, az egyetem, a város, az ország stb. megjelöléséhez.

A lehetséges dimenziószámot a számítógépben csak a rendelkezésre álló tárolóhely nagysága korlátozza.

5

STRINGEK ALKALMAZÁSA

Az adatfeldolgozásban a stringeknek nagy jelentőségük van. Az a számítógép, amely nem képes stringeket feldolgozni, nem egyéb, mint egy nagyteljesítményű zsebszámológép. Ezért a számítógépben az aritmetikai funkciók mellett string-műveleti funkciók is a programozó rendelkezésére állnak.

Ebben a fejezetben ezeket, a BASIC programnyelvben használható string-műveleti utasításokat tárgyaljuk.

1. Stringek összehasonlítása

A relációs operátorok (=, >, <, stb.) segítségével stringek egyenlőségét vizsgálhatjuk, ill. összehasonlíthatjuk őket alfabetikus nagyságuk szerint.

Ha egyenlőséget vizsgálunk, akkor minden karakternek, a kezdő és záró szóköznek (blank) meg kell egyezni, különben egyenlőtlenséget kapunk.

Például:

```
100 IF A$ = "IGEN" THEN 250
```

A stringeket a gép karakterről karakterre, balról jobbra haladva hasonlítja össze. Hogy pontosabbak legyünk: a karakterek ASCII kódját hasonlítja össze.

A magasabb kódszámú karakter értéke nagyobb, mint az alacsonyabb kódszámúé, más szavakkal „AC" nagyobb, mint „AB". A betűkre az ABC sorrendje érvényes (A=min., Z=max.).

Ha különböző hosszúságú stringeket hasonlít össze a gép, akkor a hosszabbnak az értéke nagyobb, akkor is, ha ugyanazok a két string karakterei. Így pl. „B" kisebb, mint „B ".

A következő relációs operátorokat használhatjuk stringek összehasonlításához:

<, <=, =>, >, =, <>

2. String műveletek

Csak egyetlen string művelet van, amely a stringek egymáshoz fűzésére szolgál. Az operátor a „+" jel.

Példa:

```
10 S1$ = "A CSILLAG"
20 S2$ = " RAGYOG"
30 S3$ = ","
40 C$ = S1$ + S2$ + S3$ + S2$
50 PRINT C$
60 END
READY
> RUN
A CSILLAG RAGYOG, RAGYOG
```

A további pontokban (3—11.) a string függvényeket ismertetjük. Alkalmazásukkor a függvények általános alakja:

Eredmény = függvény (argumentum)

A függvények mindazon helyre írhatók, ahol egyébként ugyanolyan típusú kifejezések használhatók.

3. ASC (string)

Ez a függvény kiszámolja a megadott string első karakterének decimális ASCII kódját. A stringnek idézőjelben kell lenni. Ha az argumentumba üres stringet írnak, a gép hibajelzést ad.

Példa:

```
100 PRINT "AZ 'O' ASCII KODJA:" ; ASC("O")
105 S$ = "OTTHON": PRINT "A STRING:" ; S$
110 PRINT "AZ ELSO BETU ASCII KODJA:" ; ASC(S$)
120 END
READY
> RUN
AZ 'O' ASCII KODJA: 79
A STRING: OTTHON
```

AZ ELSO BETU ASCII KODJA: 79

Mindkét sor ugyanazt a számot nyomtatja ki. Az ASCII kódok listáját a C függelékben találjuk.

4. CHR$ (kifejezés)

Ez a függvény éppen az ASC függvény ellentettjét hajtja végre, azaz megadja a kifejezés értékéhez tartozó karaktert. Argumentumként 0–255-ig terjedő tartományban lévő szám, ill. ilyen értékű kifejezés szerepelhet. Az argumentumot zárójelbe kell tenni.

```
100 PRINT CHR$ (33): REM IRJ EGY '!' JELET
```

5. LEFT$ (string, n)

Ez a függvény megadja az adott string első n karakterét. A stringnek és az „n"-nek zárójelben kell állni. A string lehet egy kifejezés vagy egy konstans, az n pedig numerikus kifejezés vagy állandó lehet.

Példa:

```
10 A$ = "ABCDEFG"
20 B$ = LEFT$ (A$, 4)
30 PRINT B$
40 END
READY
> RUN
ABCD
```

6. RIGHT$ (string, n)

Ez a függvény megadja a string utolsó n karakterét. Az argumentumoknak zárójelben kell állniuk. A string lehet string-változó vagy konstans, n pedig numerikus változó vagy konstans. Ha a string hossza kisebb, vagy egyenlő n-nel, akkor az utasítás az egész stringet megadja.

Példa:

```
10 A$ = "ABCDEFG"
20 B$ = RIGHT$ (A$, 3)
30 PRINT B$
40 END
READY
> RUN
EFG
```

7. LEN (string)

Ez a függvény megadja a string hosszát. A string változó, konstans vagy string-kifejezés lehet.

Példa:

```
10 A$ = "ABCDEFG"
20 PRINT "A STRING HOSSZA:" ; LEN (A$)
30 END
READY
> RUN
A STRING HOSSZA: 7
```

8. MID$ (string, p, n)

Ez a függvény megadja a string egy részét a p pozíciótól kezdve, a rész-string hossza n karakter. A string lehet konstans, változó, vagy kifejezés, p és n pedig numerikus konstans, változó vagy kifejezés lehet.

Példa:

```
10 A$ = "ABCDEFG"
20 B$ = MID$ (A$, 3,4)
30 PRINT "AZ UJ STRING:" ; B$
40 END
READY
> RUN
AZ UJ STRING: CDEF
```

9. STR$ (kifejezés)

Ez a függvény egy szám-konstanst, vagy kifejezést stringgé alakít át. Az argumentumnak zárójelben kell állnia.

Példa:

```
10 A = 34.56
20 B$ = STR$ (A)
30 B$ = B$ + "%"
40 PRINT "AZ EREDMENY:" ; B$
50 END
READY
> RUN
AZ EREDMENY: 34.56 %
```

10. STRING$ (n, karakter vagy szám)

Ez a függvény egy n hosszúságú, az adott karakterből álló stringet állít elő.

Példa:
```
10 PRINT STRING$ (10, "*")
20 END
READY
> RUN
**********
```

A jel helyett egy szám a (0,255) intervallumból is megadható. Ezt a számot a gép ASCII kódként kezeli és az utasítás az ennek a kódnak megfelelő jelet vagy grafikus kódot állítja elő.

Példa:
```
10 PRINT STRING$ (10, 33)
20 END
READY
> RUN
!!!!!!!!!!
```

11. VAL (string)

Ez a függvény az STR$ ellenkezőjét végzi: numerikus értéket állít elő a string karaktereiből.

Példa:
```
10 A$ = "56"
20 B$ = "23"
30 C = VAL (A$ + "." + B$)
40 PRINT "AZ EREDMENY:" ; C; "," ; C+100
50 END
READY
> RUN
AZ EREDMENY : 56.23, 156.23
```

6

BELSŐ ARITMETIKAI FÜGGVÉNYEK

Ebben a fejezetben a számítógéprendszerben használható aritmetikai függvényeket tárgyaljuk. A legtöbb esetben a függvényérték kiszámítása előtt meg kell adni a függvény argumentumát.

Ez az argumentum konstans, numerikus változó vagy kifejezés lehet. Az általános alak:

Eredmény = függvény (argumentum)

Példa:

```
10 A = RND (3)
20 B = INT (C)/D
30 E = SQR (F * G -4)
```

A következő függvények ismertetése történik a fejezetben:

1. ABS (X)
2. ATN (X)
3. CDBL (X)
4. CINT (X)
5. COS (X)
6. CSNG (X)
7. EXP (X)
8. FIX (X)
9. INT (X)
10. LOG (X)
11. RANDOM
12. RND (X)
13. SGN (X)
14. SIN (X)
15. SQR (X)
16. TAN (X)

1. ABS (X)

Kiszámítja X abszolút értékét.

2. ATN (X)

Kiszámítja X radiánban vett arcus tangensét. Ha az eredményt fokban akarjuk megkapni, akkor ATN (X)-et meg kell szorozni 57,29578-al.

3. CDBL (X)

X-et kétszeres pontossággal ábrázolja. Az eredmény 17 számjegyű, de csak az argumentumban levő számjegyek értékesek.

4. CINT (X)

Kiszámolja az argumentumnál kisebb, ahhoz legközelebb eső egész számot (lefelé kerekít). Az argumentum csak −32768 és 32767 közé eső szám lehet. Az eredmény egész típusú.

Például:

CINT (2.6) eredménye 2, CINT (−2.6) eredménye −3 lesz.

5. COS (X)

Kiszámolja a radiánban megadott X koszinuszát. Ha X fokban van megadva, akkor X helyére a következő kifejezést kell beírni: X * 0.0174533

Tehát a függvény beírása: COS (X * 0.0174533)

6. CSNG (X)

X-et egyszeres pontossággal ábrázolja, kiszámolja a szám 6 számjegyű értékét a kétszeres pontosságú X-ből az ún. 4/5 kerekítéssel.

7. EXP (X)

Kiszámítja az exponenciális függvény értékét az X-helyen, azaz az e^X-t. Ez az LOG X függvény inverz függvénye.

8. FIX (X)

Leválasztja az X argumentum tizedesvessző utáni részét.

FIX (1,5) = 1; FIX (–1,5) = –1

9. INT (X)

X-et egész számként ábrázolja, kiszámítja a legnagyobb X-nél nem nagyobb egész számot (lefelé kerekít). X-nek –32768 és 32767 közé kell esni. Az eredmény egyszeres pontosságú, valós típusú.

Példa:

INT (3.5) = 3; INT (–3.5) = –4

10. LOG (X)

Kiszámítja X természetes alapú logaritmusát, azaz $\log_e(X)$ et. Ha egy másik, b alapú logaritmust akarunk meghatározni, akkor a

$$\log_b(X) = \log_e(X) / \log_e(b)$$ összefüggést kell felhasználnunk,

ahol b az új alap.

11. RANDOM

Ezen utasítás hatására az RND (X) függvény egy véletlenül kiválasztott számtól kezdve állítja elő az álvéletlen számokat. A függvényhez nem kell argumentumot megadni.

12. RND (X)

Ez a függvény egy álvéletlen számot állít elő egy adott intervallumban (ezek a számítógép belsejében keletkeznek; a felhasználó nem tud beavatkozni).

RND (0) egyszeres pontosságú véletlen számot állít elő a (0,1) intervallumból.

RND (X) 1 és X közötti egész számot állít elő, beleértve az 1-et és az X-et.
X 32768-nál kisebb pozitív szám lehet.

13. SGN (X)

Az előjelfüggvény értéke −1 lesz, ha X negatív, nulla, ha X nulla és +1, ha X pozitív.

14. SIN (X)

Kiszámolja a radiánban megadott X szinuszát (X radiánban vett szög). Ha X fokban van megadva, akkor X helyére a következő kifejezést kell beírni: X * 0.0174533.

Tehát a függvény beírása: SIN (X * 0.0174533)

15. SQR (X)

Kiszámolja X négyzetgyökét.

16. TAN (X)

Kiszámolja a radiánban megadott X tangensét.
Ha X fokban van megadva, akkor X helyére a következő kifejezést kell beírni: X * 0.0174533.

Tehát a függvény beírása: TAN (X * 0.0174533).

7
GRAFIKUS FUNKCIÓK

A számítógép négy grafikus funkcióval rendelkezik, melyek rendkívül hatékonyak. Lehetővé teszik a felhasználónak, hogy a BASIC programnyelv segítségével minden rajzos (grafikus) feladatot kirajzoljon a képernyőre.

1. SET (x,y)

Ez a funkció bekapcsolja az (x,y) koordinátákkal megadott grafikus blokkot. A képernyő 128 függőleges, 48 vízszintes vonalból álló hálóra van felosztva. Az x koordináta 0-tól 127-ig, balról jobbra az oszlopokat, az y koordináta tartománya 0-tól 47-ig, fentről lefelé a sorokat jelenti.

Így a (0,0) pont a képernyő bal felső sarkában, a (127,47) pont pedig a képernyő jobb alsó sarkában van. Az (x,y) argumentumok numerikus konstansok, változók, vagy kifejezések lehetnek. A SET (x,y) az argumentum egészrészét használja, ezért nem szükséges, hogy az argumentum egész legyen.

2. RESET (x,y)

Ez a funkció kikapcsolja az (x,y) koordinátákkal megadott grafikus blokkot. Argumentumaira ugyanez érvényes, mint a SET (x,y) funkció argumentumaira.

3. CLS

Ez a funkció törli a képernyőt és kikapcsolja a grafikus blokkokat, ezenkívül a cursort a képernyő bal felső sarkába viszi. Ez a funkció lehetővé teszi a felhasználónak, hogy a képernyőt kiíratáshoz szabaddá tegye, anélkül, hogy annak előző tartalma zavarná a megjelenítést.

4. POINT (x,y)

Ez a funkció megvizsgálja, hogy egy, az (x,y) koordinátákkal megadott grafikus blokk be van-e kapcsolva vagy sem. Ha a blokk be van kapcsolva (azaz itt SET-et hajtott végre a gép), akkor a POINT értéke −1 (logikailag „igaz"), ha a blokk nincs bekapcsolva, akkor értéke 0 (logikai „hamis").

Példa:

```
A = POINT  (3,4)
```

Ha a (3,4) blokkot grafikai ábrázoláskor felhasználtuk, akkor A értéke −1, egyébként 0.

8

SPECIÁLIS FUNKCIÓK

1. INP (kapuszám)

Beolvas egy 8 bites értéket a megadott kapuról. A számítógéprendszer 256 kaput tud címezni, melyeket 0-tól 255-ig számoz.

Példa:

```
10 A = INP (124)
```

Beolvas egy 8 bites értéket a 124-es számú kapuról az A változóba.

Az utasítás periféria-kezeléshez használatos.

2. OUT kapuszám, érték

Ez az utasítás egy 8 bites értéket ad ki a megadott számú kapura. Az utasításhoz két argumentumot kell megadni: a kapuszámot és azt az értéket, amit erre a kapura ki kell adni.

Példa:

```
30 OUT 14,240
```

Ez az utasítás a 240-es értéket adja ki a 14-es kapura. Mindkét argumentumnak 0 és 255 közötti számnak kell lenni (1 byte).

Az utasítás periféria-kezeléshez használatos.

3. PEEK (cím)

Ez a funkció behoz egy 8 bites értéket a megadott című memóriahelyről (a cím decimális lehet). A behozott szám szintén decimális, nagysága 0 és 255 közötti.

Példa:

```
20 B = PEEK (3000)
```

Ezzel a sorral a 3000-es memóriaszám tartalmát beírtuk B-be.

4. POKE cím, érték

Ez az utasítás beír egy 8 bites értéket a – decimális számmal – megadott memóriacímre. Az utasításnak két argumentuma van: cím és érték. Az értéknek 0 és 255 között levőnek kell lenni.

Példa:

```
10 A = 250
20 POKE 19000, A
30 B = PEEK (19000)
40 PRINT "EREDMENY:" ; B
50 END
READY
> RUN
EREDMENY: 250
```

5. MEM

Megadja a memóriában lévő még nem felhasznált és nem védett byte-ok számát.

Példa:

```
200 IF MEM < 180 THEN 700
```

Ha a MEM-et parancsként akarjuk felhasználni, akkor PRINT-tel együtt kell alkalmaznunk. A PRINT MEM megadja a memóriában levő byte-ok számát, amelyekben nincs se program, se változó, se string, stb.

6. ERL

Argumentum nélküli függvény, mely hiba esetén a hibás sor sorszámát adja vissza értékként.

Példa:

```
10 ON ERROR GOTO 30
20 A=7/O
30 PRINT ERL
40 END
READY
>RUN
 11
```

7. ERR

Argumentum nélküli függvény, amely hiba esetén a hibakód számától (lásd a B függelékben) függő értéket ad vissza, amelyet a következőképp számít ki:

ERR = (hibakód–1) *2

Példa:
```
10 ON ERROR GOTO 30
20 A=7/0
30 PRINT ERR/2+1
40 END
READY
>RUN
 11
```

Az utasításkészlet négy további BASIC utasítást tartalmaz.

Ezek: (1) INKEY$, (2) POS, (3) USR, (4) VARPTR

(1) INKEY$

A billentyűzetről beolvas egy karaktert. Ha az INKEY függvény végrehajtásával egyidőben nem ütünk le billentyűt, akkor a gép nullkaraktert állít elő (=" "). Az INKEY funkció által beolvasott adatok nem jelennek meg a képernyőn.

Példa:
```
10 REM * EGY BETU BEIRASA
20 REM * ANELKUL HOGY AZT A KEPERNYON LATHATOVA TENNENK
30 CLS
40 PRINT "EGY BETU BEIRASA"
50 A$ = INKEY$: IF A$ = "O" THEN 60 ELSE 50
60 B$ = INKEY$: IF B$ = "K" THEN 70 ELSE 60
70 PRINT "UDVOZLUNK"
```

Ez a program addig vár, amíg a beütött billentyűk között nem szerepel az O, majd a további betűk között a K. Ekkor írja ki az üdvözlést.

(2) POS (álargumentum)

A POS függvény megadja a cursor aktuális helyét egy 0 és 63 közötti számmal. Az álargumentum tetszőleges lehet.

Példa: 100 PRINT TAB (25) POS (0)

A fenti utasítás hatására a gép 25-öt fog írni a 25. helyre (pontosabban a 25. helyen az előjel szerepel, amely jelen esetben nem íródik ki, mert pozitív, s a szám a 26. helyen kezdődik).

(3) USR (argumentum)

Meghív egy gépi nyelvű szubrutint. A szubrutint vagy POKE-kal állíthatjuk elő, vagy a szalagról olvashatjuk be. Ha felhasználó a gépi nyelvű programozásban még nem rendelkezik kellő gyakorlattal, akkor ezen funkció használata nem tanácsos.

A szubrutin kezdőcímét a függvény hívása előtt POKE-kal az 16526 és 16527-es címre kell írnunk, 16526-ra az alacsonyabb értékű byte-ot.

Ha argumentumot akarunk átadni a szubrutinnak, akkor az alprogram kezdetén egy CALL ØA7FH (2687 decimálisan) utasítást kell végrehajtanunk. Ezután az argumentum a HL regiszterbe íródik.

Ha vissza akarunk térni a BASIC-be, anélkül hogy értéket visszaadnánk, a szubrutint a RET-tel zárhatjuk le. Ha értéket is vissza akarunk adni, akkor azt be kell tölteni a HL regiszterpárba és a szubrutin végén a JP ØA9AH (2714 decimálisan) utasítást kell végrehajtani.

Az értékeket a gép 2 byte-os egész, előjeles számokként kezeli. Az USR funkció 8 stack szintet foglal el a szubrutin részére.

Példa:

```
10 INPUT I %: REM ARGUMENTUM BEIRASA
15 REM * KEZDOCIM BEIRASA *
20 POKE 16526,0 : POKE 16527,120
30 A = USR (I %): REM * AZ ARGUMENTUM VISSZAADODIK*
```

A szubrutint memóriaterület végére ajánlatos írni. Azért, hogy a szubrutin területét megvédjük attól, hogy a BASIC programtárolónak fogja fel és ezzel a tartalmát megzavarja, célszerű a következőt tenni: ha a rendszer a bekapcsolás után kiírja a READY? jelzést, adjuk meg (decimálisan) azt a legmagasabb memóriacímet, melyet a BASIC használhat.

(4) VARPTR (változó név)

Ez a függvény megadja azt a címet, ahol a változó értéke a tárolóban elhelyezkedik..

```
10 K = VARPTR (A)
```

Ezután a különböző változó típusoknak megfelelően K értéke — amelyet itt a továbbiakban ugyancsak K-val jelölünk — a következőket jelenti:

a) 2 byte, egész értékű változók (A %)
K = LSB címe
K+1 = MSB címe

b) Egyszeres pontosságú változók (A)
K = LSB címe
K+1 = következő MSB címe
K+2 = MSB címe
K+3 = kitevő címe

c) Kétszeres pontosságú változók (A#)
K = LSB címe
K+1 = következő MSB címe
⋮
K+6 = MSB címe
K+7 = kitevő címe

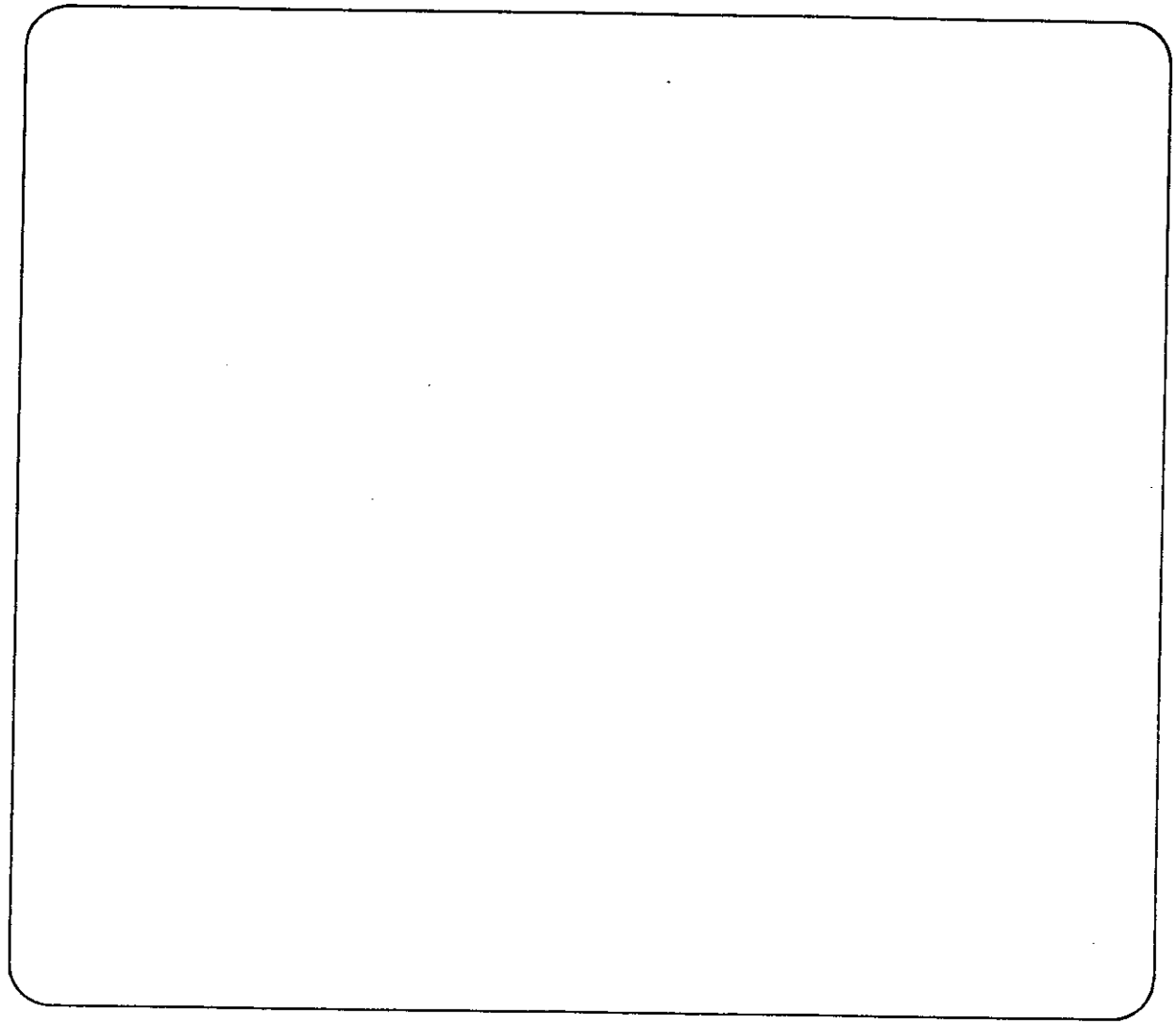
d) string változók (A$)
K = string-hosszúság
K+1 = a string kezdőcímének LSB-je
K+2 = a string kezdőcímének MSB-je

LSB — a legalacsonyabb helyi értékű byte-ot jelöli
Következő MSB — a következő, magasabb helyiértékű byte-ot jelöli
MSB — a legmagasabb helyi értékű byte-ot jelöli

Példa:

Ø3F1H hexadecimális rendszerben ez két byte, Ø3 az MSB és F1 az LSB

FÜGGELÉK



A KULCSSZAVAK

ABS	ELSE	LEN	RESTORE
AND	END	LINE	RESUME
ASC	ERL	LIST	RETURN
ATN	ERR	LOAD	RIGHT$
CDBL	ERROR	MEM	RND
CHR$	EXP	MID$	SET
CINT	FIX	NAME	SGN
CLEAR	FOR	NEW	SIN
CLOSE	FRE	NEXT	SQR
CLS	GET	NOT	STEP
CONT	GOSUB	ON	STOP
COS	GOTO	OUT	STRING$
DATA	IF	PEEK	STR$
DEFDBL	INKEY$	POINT	TAB
DEFFN	INP	POKE	TAN
DEFINT	INPUT	POS	THEN
DEFSNG	INSTR	PRINT	TROFF
DEFUSR	INT	PUT	TRON
DEFSTR	KILL	RANDOM	USING
DELETE	LEFT$	READ	USR
DIM	LET	REM	VAL
EDIT	LSET	RESET	VARPTR

Ezeket a szavakat változónevekben nem használhatjuk.
E kulcsszavak közül nem mindegyik került alkalmazásra a BASIC leírásban és a számítógép programozási rendszerében, azonban további bővítésekben szerepelhet. A nem alkalmazott kulcsszavakat sem lehet használni változónevekben, mert SN hibajelzést kapunk.

B

HIBAKÓDOK

Kód	Rövidítés	Hiba
1	NF	FOR nélküli NEXT
2	SN	Szintaktikai hiba
3	RF	GOSUB nélküli RETURN
4	OD	Nincs több adat
5	FC	Nem megengedett függvényutasítás
6	OV	Túlcsordulás
7	OM	Nincs több tárolóhely
8	UL	Definiálatlan sor
9	BS	A megadottnál nagyobb dimenzió
10	DD	Dimenzió-felülírás
11	/0	Nullával osztás
12	ID	Nem megengedett közvetlen felhasználás
13	TM	Típuskeveredés
14	OS	Nincs több hely string-ek részére
15	LS	A string túl hosszú
16	ST	Túlságosan összetett stringművelet
17	CN	CONT utasítás végrehajthatatlan
18	NR	NO RESUME
19	RW	RESUME ERROR nélkül
20	UE	Nem kiírható hiba
21	MO	Hiányzó operandus
22	FD	Hiányos file

A HIBAÜZENETEK LEÍRÁSA

NF — FOR nélküli NEXT; NEXT utasítást használtunk megfelelő FOR nélkül. Ez a hiba akkor is felléphet, ha egymásba ágyazott ciklusokban NEXT változókat felcseréltünk.

SN — Szintaktikus hiba; valamilyen hibás beírás eredménye, nyitva hagyott zárójel, érvénytelen jel, vagy rosszul írt utasítás.

RG — GOSUB nélküli RETURN: olyan RETURN utasítást talált a gép, amelyhez nincs megfelelő GOSUB.

OD — Nincs több adat: a READ vagy az INPUT # utasításnak nem áll több adat rendelkezésre, mert elfelejtettünk DATA utasítást adni, vagy pedig mert az adatlistáról a gép már az összes adatot beolvasta.

FC — Nem megengedett függvényutasítás; megpróbáltunk egy műveletet olyan paraméterrel végrehajtatni, amellyel az nem lehetséges. Például: negatív dimenziót adni, negatív szám vagy nulla logaritmusát venni, stb.; USR utasítás, a megfelelő POKE indítócím nélkül.

OV — Túlcsordulás; egy kiszámított érték vagy adat (abszolút értékben) túl nagy, így a gép ezt nem tudja tovább kezelni.

OM — Nincs több tárolóhely; az összes tárolóhely már vagy betöltött vagy előre lefoglalt. Az ok a tömbdimenzió túl nagy értéke lehet, vagy egymásba ágyazott ugró utasítások, mint GOTO, GOSUB, FOR-NEXT. Az is lehet azonban, hogy egyszerűen túl hosszú a program.

UL — Definiálatlan sor; megpróbáltunk egy nem létező sorra hivatkozni.

BS — A megadottnál nagyobb dimenzió; megpróbáltunk olyan tömbelemet hívni, amelynek dimenziója nagyobb a DIM utasításban megadottnál.

DD	Dimenzió-felülírás; megpróbáltuk egy olyan tömb dimenzióját megadni, melynek dimenzióját már korábban meghatároztuk (DIM utasítással vagy implicit módon). A helyes programozási gyakorlat az, hogy a DIM utasítást a program elejére tesszük.
/0	Nullával osztás; megpróbáltuk a nullát osztóként felhasználni.
ID	Nem megengedett, közvetlen felhasználás; az INPUT utasítást parancs üzemmódban akartuk végrehajtani.
TM	Típuskeveredés; megpróbáltunk nem string változóba stringet írni.
OS	Nincs több hely stringek részére.
LS	A string túl hosszú; egy string csak 255 karakterből állhat.
ST	Túlságosan összetett stringművelet; a műveletet összetevőire kell bontani.
CN	A CONT utasítást nem tudja a gép végrehajtani; CONT-ot adtunk például END vagy EDIT után.
NR	NO RESUME; a program végét a hibafelismerési üzemben érte el a gép.
RW	ERROR nélküli RESUME; a gép RESUME-t talált, mielőtt ON ERROR GOTO-t hajtott volna végre.
UE	Nem kiírható hiba; megpróbáltunk egy hibát nemlétező ERROR kóddal szimulálni.
MO	Hiányzó operandus; megpróbáltunk egy olyan műveletet végrehajtani, ahol az operandusok egyike hiányzik.
FD	Hiányos file; a külső forrásból (pl. kazettás magnetofonról) történt adatbeolvasás hibás volt, vagy az adatok nem voltak megfelelő sorrendben, stb.

C

VEZÉRLŐ KARAKTEREK

Kód	Művelet
0– 7	nem felhasznált kód
8	visszalépés és karaktertörlés
9	nem felhasznált kód
10—13	kocsi vissza, soremelés
11—12	kocsi vissza lapemelés
14	cursor megjelenítés
15	cursor letiltás
16—22	nem felhasznált kódok
23	32 karakteres módba vált
24	←
25	→
26	↓
27	↑
28	cursor a kezdő pozícióba (0,0), 64 karakteres módba vált
29	cursor a sor elejére
30	sor végéig töröl a cursor pozíciótól
31	a képernyő végéig töröl a cursor pozíciótól

ASCII KARAKTER KÓDOK

Kód	Karakter	Kód	Karakter	Kód	Karakter	Kód	Karakter
32	space	48	0	64	@	80	P
33	!	49	1	65	A	81	Q
34	"	50	2	66	B	82	R
35	#	51	3	67	C	83	S
36	$	52	4	68	D	84	T
37	%	53	5	69	E	85	U
38	&	54	6	70	F	86	V
39	'	55	7	71	G	87	W
40	(	56	8	72	H	88	X
41	)	57	9	73	I	89	Y
42	*	58	:	74	J	90	Z
43	+	59	;	75	K	91	[
44	,	60	<	76	L	92	\
45	-	61	=	77	M	93	]
46	.	62	>	78	N	94	^
47	/	63	?	79	O	95	_

MEMÓRIAKAPACITÁS ÉS A PROGRAMOZÁS KORLÁTAI

INTERVALLUMOK

egész számok	–32768-tól 32767-ig
egyszeres pontosság	–1,701411E ±38-tól 1,701411E ±38-ig
kétszeres pontosság	–1,701411834544556E ±38-tól +1,701411834544556E ±38-ig
string hosszúsága	max. 255 karakter
sorszám	0-tól 65529-ig
programsorok hossza	max. 255 karakter

Memóriaszükséglet

egy programnak	min 5 byte, mégpedig
sorszámnak	2 byte
sormutató (pointer)-nak	2 byte
soremelés (carriage return)-nek	1 byte

Ezenkívül minden utasításnak, operátornak, változónévnek, speciális karakternek és konstans számjegynek (betűnek) 1 byte a memóriaszükséglete.

egész változó	5 byte
egyszeres pontosságú változó	7 byte
kétszeres pontosságú változó	11 byte
(fentiekből 3 a változónévhez)	
string változó	min. 6 byte

(3 a változónévhez, 3 a stack-hez és a pointerhez, 1 minden jelhez)

tömb változó	min. 12 byte

(3 a változónévhez, 2 a nagysághoz, 1 a dimenziószámhoz, és 2,3,4 vagy 8 minden egyes elemhez, a típustól függően.)

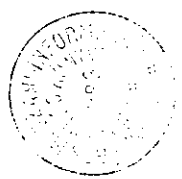
Minden egyes aktív FOR-NEXT ciklus: 16 byte; minden egyes aktív GOSUB-hoz (még nem érte el a RETURN-t): 6 byte; minden zárójelszinthez: 4 byte és 12 byte részeredményenként.

Gyártó:
HÍRADÁSTECHNIKA SZÖVETKEZET

H–1519 Budapest
Pf 268
Telex: 22–6151 htsz h

II/83 nyomat
(Az I/83 nyomat átdolgozott kiadása)

Felelős kiadó:
Somlyay Endre



Készült a MEDIA GM gondozásában

Aranykalász Dunaföldvár

75-