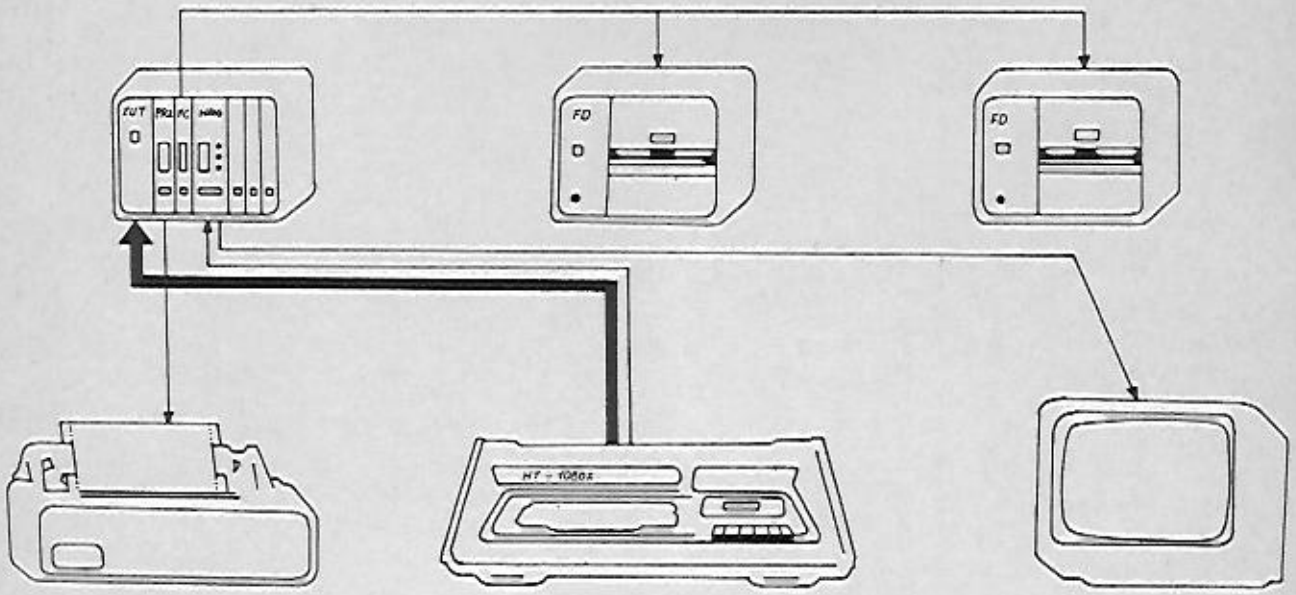
FŐVÁROSI PEDAGÓGIAI INTÉZET
TUDOMÁNYSZERVEZÉSI
ÉS INFORMATIKAI INTÉZET SZÁMÍTÁSTECHNIKAI SOROZATA

A HT–1080Z iskolaszámítógép programozása

INTERPRESS
BUDAPEST 1985

HIRADÁSTECHNIKA
SZÖVETKEZET
1519. Budapest, Pf. 268



HT – 1080Z/64 School Computer

Az alapgép már 64 kbyte memóriakapacitású ! BASIC és FORTH programozási nyelv!

Nyomtató illesztő

Minden u. n. Centronics tipusú nyomtatóhoz alkalmazható (pl. Epson RX - 80, Seikosha GP80 GP100, TERTA TMT 12X, MOM K6311, stb.)

HT – 1080Z/PR1 kábelre szerelt változat
HT – 1080Z/PR2 bővitő egységbe tolható kártya

Bővitő egység HT –1080Z/EU7

Különféle illesztő és bővitő kártyák / max. 7db. / fogadására alkalmas doboz. Busz illesztő és tápegy - séget tartalmaz.

Floppy illesztő HT – 1080Z/F1

Illesztő egység hajlékony mágneslemezes tárolóhoz / 4 db. MOM 1800/900 tipusú drive meghajtásra alkal - mas. / Bővitő egységbe tolható kártya.

Nagybontású grafika HT – 1080Z/HRG

512 x 512 felbontású grafikus megjelenitést tesz lehetővé. Minden képpont külön címezhető. Kép - pontonként négyféle fénnyességi állitható be. - Nagyitási lehetőség ! BASIC – ből programozható.

FŐVÁROSI PEDAGÓGIAI INTÉZET
TUDOMÁNYSZERVEZÉSI
ÉS INFORMATIKAI INTÉZET SZÁMÍTÁSTECHNIKAI SOROZATA
III.

Szerkesztők: APPEL GYÖRGY–MIHÁLYFI JÁNOS

A HT–1080Z iskolaszámítógép programozása

(utánnyomás)

INTERPRESS
BUDAPEST, 1985.

A KÉZIKÖNYV SZERZŐI AZ EÖTVÖS LÓRÁND TUDOMÁNYEGYETEM MATEMATIKAI INTÉZET SZÁMÍTÁSTECHNIKAI TANSZÉKÉNEK ALKOTÓ MUNKAKÖZÖSSÉGE

Bán Péter
Harmathy Zoltán
Helfenbein Henrik
Horváth László
Sándor Antal
Szlávi Péter
Veszprémi Anna
Zsakó László

A munkaközösség vezetője:
Kőhegyi János

Lektorok:
Hubert Tibor
Lukács József

Szerkesztők:
dr. Appel György
Mihályfi János

ISBN 963 7222 01 4
ISSN –0236–9230

ELŐSZÓ

Az ország valamennyi középiskolájában és néhány felsőoktatási intézményben megtalálható a HT–1080Z iskolaszámítógép. Sokan fordulnak hozzánk, hogy a gépkönyv mellett hasznos lenne egy kézikönyv megjelentetése, amelyben a BASIC-nyelv egy konkrét gépi megvalósítása mellett a gépi szintű programozás elemeit is ismertetjük.

A kézikönyv célja, hogy segítsük a tanárokat, diákokat, szülőket abban, hogy minél jobban elsajátítsák a személyi számítógép programozását.

Célunk volt továbbá, hogy a tanártovábbképzések mellett e könyvvel az iskolai számítástechnikai szakkörök munkáját is segítsük. Ezért a könyv anyaga a javasolt iskolai számítástechnika-szakkör tematikájához is igazodik (Köznevelés, 1983 június), feladatkiválasztása pedig a középiskolai tananyaghoz. A könyv szerkezete három fő szempont köré csoportosítható:

- Az első a „programozási alapok" elsajátítása, amit az 1–2. fejezetben valósítunk meg;
- A második fontos feladat a BASIC-nyelv „alapszókincsének", vagyis utasításainak és parancsainak számbavétele, ezt a célt szolgálják a 3–12. fejezetek;
- a továbbiakban a speciálisabb programokhoz szükséges sajátos HT–1080Z BASIC szolgáltatásokat tárgyaljuk. Ennek keretében (13–17. fejezetek) ismerkedünk meg az oktatási szempontból oly jelentős demonstratív lehetőségekkel, mint a képernyő grafikus használata, továbbá a hanggenerátor „programozása".

Célszerűnek tartottuk, hogy adjunk olyan *átfogó lexikális ismereteket is,* amelyek általában a kézikönyvekben is megtalálhatók.

Így ebben a könyvben (függelékek):

- az egyes kulcsszavak (utasítások, parancsok) könnyen visszakereshetők (*A* függelék),
- hiba esetén részletes diagnosztikai magyarázatokkal segítjük az olvasót a hiba okának kiderítésében (*B* függelék),
- a szövegkezeléshez, illetve a grafikus megjelenítéshez szükséges információkat (ASCII kódtáblázat, képernyőfelosztás) a *C*, illetve a *D* függelék tartalmazza,
- a számítástechnikában kevésbé járatos olvasó megtalálja az egyes szakmai fogalmak magyarázatát (*E* függelék),
- hasznos tudnivalókat közlünk ezen számítógép „lelkivilágáról" (adatábrázolás – *F* függelék, memóriafelosztás – *H* függelék),
- végül segítséget adunk programjaik átírásához azoknak, akik az ABC80-at ismerik (*I* függelék).

A könnyebb megértés kedvéért a „kritikus" helyeket *bőven elláttuk példákkal, a tudás ellenőrizhető a kitűzött feladatok megoldásával.* Ehhez csak egy tanáccsal járulunk elő̈ljáróban: a feladat megoldását kezdjük mindig a feladat pontos megértésével, majd próbálkozzunk a megoldással, és csak ezután nézzük meg a közölt megoldást. Sose keseredjünk el azon, hogy nem egyezik pontosan a két megoldás. Tudnunk kell, hogy a programozási feladatok mindig többféleképpen megoldhatók, így a közölt megoldás is csak egy a sok közül. Mindenesetre nagyon lényeges az, hogy – ha tehetjük – géppel „ellenőriztessük" saját megoldásunkat, hiszen kizárólag a gép „mondhatja rá az ámen".

Ahol lehetséges, bővebb ismereteket próbálunk adni a BASIC nyelvről és ennek alkalmazási lehetőségeiről. Így kiemeljük majd, hogy melyek azok az alapvető tulajdonságai a nyelvnek, amelyekkel minden BASIC implementációnak rendelkeznie kell és melyek a HT–1080Z BASIC specialitásai. A szövegben ●-jelekkel emeljük ki azokat a megjegyzéseket, amelyekben a BASIC nyelv más implementációira utalunk.

Előbb-utóbb e gép – vagy egy változata – az általános iskolákban is elterjed. Nem lesz kárbaveszett munka, ha valaki megismerkedik a HT–1080Z BASIC nyelvvel, hiszen a BASIC „nyelvjárásai" igen nagy mértékben hasonlóak.

Nyomdatechnikai okokból "#" helyett a "■" jelet használjuk. A jobb olvashatóság kedvéért programunkban használjuk a magyar nyelv ékezetes betűit, bár ezeket nem lehet megtalálni a HT–1080Z gép mindegyik változatánál.

Az elektronika, ezzel együtt a számítástechnika rohamosan fejlődik. Újabb és újabb személyi számítógépek jelennek meg. A kézikönyv tartalmazza az alapvető programozási és BASIC ismereteket. Ezen ismeretek és a gépekhez adott kezelési útmutatók segítségével már könnyen elsajátítható a különböző típusú személyi számítógépek kezelése, alkalmazása.

Ezúton szeretnénk köszönetet mondani a Szolnok megyei Pedagógiai Intézet munkatársainak a könyv elkészítésében nyújtott segítségükért.

A szerzők

Budapest, 1984. november

A HT–1080Z-t először mint egyszerű feladatmegoldó (számításokat végző) eszközt ismerjük meg. Ez lehetőséget nyújt ezen számítógép kalkulátor-szerű használatára. Nézzünk néhány egyszerű feladatot (minden különösebb magyarázat nélkül):

Feladat: Adott egy kör, amelynek sugara 5 cm.
Számítsuk ki a kör kerületét és területét!

Megoldás: Egy 5 cm sugarú kör kerületét a

$$K = 2 \cdot \pi \cdot 5$$

kifejezéssel tudjuk kiszámítani, a megfelelő parancs, amellyel a számítógépet felszólíthatjuk a feladat megoldására:

```
PRINT 2*3.14159*5
```

(Írhatjuk kis és nagy betűkkel is; a beírás végét a NEWLINE billentyű lenyomása jelzi.)
A fenti kifejezéshez képest csupán az változott, hogy a szorzást a *-jellel jelöltük, a π közelítő értéke pedig 3.14159, továbbá a kifejezés elé egy angol szót tettünk (PRINT = nyomtasd ki). Ez felszólítja a számítógépet az eredmény – televízió képernyőre való – kiírására, amely ezt a tevékenységet azonnal végrehajtja.
A terület kiszámítása hasonló módon történik:
a

$$T = 5^2 \cdot \pi$$

kifejezés alapján a

```
PRINT 5*5*3.14159
```

parancsot adjuk.

Feladat: Adott egy háromszög két oldalának hossza (5 cm, 7 cm) valamint az általuk közbezárt szög (50 fok). Számítsuk ki a háromszög területét!

Megoldás: Matematikai ismereteink szerint a területet a

$$T = \frac{a \cdot b \cdot \sin \gamma}{2}$$

kifejezéssel számolhatjuk ki. A feladat megoldásában két dolog okozhat gondot. Az első az, hogy az írógépen nem találhatunk ilyen hosszú törtvonalat (helyette a /-jelet használjuk, pontos szabályai a 3. FEJEZET-ben találhatók); a másik pedig a szinusz függvény használata. A szöget fokban vagy radiánban kell (?), lehet (?) megadni? Az utóbbira a válaszunk az, hogy radiánban, ezért a fenti adatot (50 fok) át kell számítani a

$$\frac{\pi}{180} \cdot 50$$

kifejezéssel radiánba. A feladatot a következő paranccsal oldhatjuk meg:

```
PRINT (5*7*SIN (3.14159/180*50))/2
```

A parancsból látható az osztás megoldása. A szinusz függvény nevének (SIN) leírása után zárójelbe kell tenni azt az értéket, amelynek szinuszát kell venni!
Mivel a kifejezés elég bonyolult, a feladatot megoldhatjuk több lépésben is. Ehhez szükség lehet egy közbenső eredmény tárolására. Erre a célra a zsebszámológépek ún. m e m ó r i á -t használnak. Ezt itt is megtehetjük például a következő parancsokkal:

```
F = 3.14159/180*50
S = SIN(F)
PRINT 5*7*S/2
```

Részletes magyarázat helyett – az megtalálható a 3. FEJEZET-ben – elég annyit megjegyezni, hogy ilyen célra tetszőleges betűvel jelölt „memóriarekeszeket" (ún. v á l t o z ó -kat) használtunk és egy érték változóban való elhelyezésére úgy adtunk parancsot, hogy felírtunk egy egyenlőséget, melyben a változó neve k ö t e l e z ő e n az egyenlőségjel bal oldalán szerepelt.

Ezzel fejezzük be a HT–1080Z kalkulátor-szerű használatát, az ún. k ö z v e t l e n ü z e m m ó d -dal való ismerkedést és térjünk át a p r o g r a m -ok készítésére!

A programozásba való bevezetést kezdjük néhány kérdéssel és a kérdésekre adott válaszokkal:

Kérdés: Mi a program?

Válasz: A program utasítások sorozata, amelyeket a számítógépnek meghatározott sorrendben, egymás után kell elvégeznie a feladat megoldása érdekében. A program végrehajtására az összes utasítás leírása után egy külön parancs szolgál (RUN = fuss).

Megjegyzés:
Vannak programozási nyelvek, amelyek lehetőséget nyújtanak több utasítás egyidejű, párhuzamos elvégzésére, de ezekkel most nem foglalkozunk. A továbbiakban csak a BASIC nyelvről lesz szó.

Kérdés: Mi a parancs?

Válasz: A parancs a számítógépet valamilyen tevékenység elvégzésére szólítja fel, amelyet azonnal végre kell hajtania.

Kérdés: Mi az utasítás?

Válasz: Az utasítás a számítógépet valamilyen tevékenység elvégzésére szólítja fel, amelyet akkor kell végrehajtania, ha az utasítás-sorozatban az adott utasításhoz ért.

Kérdés: Milyen tevékenységek elvégzésére szólíthatnak fel az utasítások?

Válasz: Néhény példát már láttunk az előző fejezetben ismertetett parancsok kapcsán, a többivel pedig a továbbiakban fogunk megismerkedni.

Kérdés: A két feladatban leírt parancsokat a számítógép azonnal végrehajtotta. Ha ezek utasításként is használhatók, akkor hogyan tudjuk megkülönböztetni őket a parancsoktól? (Ez ugyanis kell ahhoz, hogy ne azonnal hajtsa végre őket.) Egyáltalán hogyan tudjuk megmondani, hogy milyen sorrendben kell az utasításokat végrehajtani?

Válasz: Minden utasítás elé egy sorszámot kell írni. Az utasítássor végét a NEWLINE billentyű lenyomásával jelezzük. Ez a sorszám egyben az utasítások sorrendjét is meghatározza. Így a sorrend nem feltétlenül azonos a beírás sorrendjével és a végrehajtás sorrendjével sem.

Kérdés: Ezek szerint egy utasítássorba pontosan egy utasítást lehet írni?

Válasz: Egy utasítássorba (ami nem biztos, hogy a képernyőn is egy sor) több utasítást is leírhatunk (sorszáma az utasítássornak van), ilyenkor azonban ezeket egymástól kettősponttal kell elválasztani!

Kérdés: A sorszámok milyen számok lehetnek?

Válasz: A sorszámok tetszőleges pozitív számok 1-től 65529-ig.

Kérdés: Ezek szerint a program első sora az 1-es, a második a 2-es, . . . ?

Válasz: Lehet, de nem fontos. A BASIC program sorszám szerint növekvő sorrendbe rendezett sorok sorozata, két sorszám között azonban tetszőleges számú felhasználatlan sorszám lehet. Célszerű például tízesével sorszámozni, ezzel lehetőséget teremtünk, hogy később (!) két utasítás közé újabbat, újabbakat tegyünk.

Kérdés: Mi történik, ha a számítógép például két 10-es sorszámú sort kap?

Válasz: A második 10-es sor beírásakor az első 10-es sor megszűnik, helyét az újonnan beírt sor foglalja el.

Kérdés: Hogyan lehet egy sort kitörölni úgy, hogy ne legyen helyette új sor?

Válasz: A sor sorszámának leírásával (üres sor) az illető sor törlődik a programból.

Kérdés: Mit tesz a számítógép egy hibásan beírt utasítással?

Válasz: Ez a számítógép minden utasítást megjegyez, és majd végre is próbálja hajtani, s végrehajtás közben jelez hibát, pl. formai hibákat, 0-val osztást stb.; de az is lehet, hogy sikerül végrehajtania, ilyenkor azonban valószínűleg hibás eredményt fog adni (formailag helyes – logikailag hibás utasítás). Az ilyen típusú hibák megkeresésához a 17. FEJEZET-ben található a segédeszközök leírása.
●●● Egyes implementációk (pl. ABC80) a beírt sort azonnal ellenőrzik és formai hibás sort nem fogadnak el. ●●●

E néhány fogalom megismerése után nézzünk egy példát, egyelőre az utasítások részletes ismertetése nélkül!

Feladat: Készítsünk olyan programot, amely egy tetszőleges valós együtthatós, legfeljebb másodfokú egyenlet megoldását adja meg a valós számok körében!

Megoldás: A másodfokú egyenlet legyen

$$a \cdot x^2 + b \cdot x + c = 0$$

alakú! A programnak tetszőleges 'a,b,c' valós számokra el kell tudni készíteni a megoldást, így a program készítésekor nem ismerhetjük ezek pontos értékét. A pontos értékeket csak a program használata során tudjuk megadni, szükség van olyan utasításra, amely végrehajtás közben megkérdezi egyes változók konkrét értékét, ez a beolvasó (INPUT = bemenet) utasítás. Ez előtt a programnak természetesen valamilyen módon közölnie kell a használójával (nem biztos, hogy azonos a program írójával!), hogy mit vár tőle. (Ha szöveget akarunk kiírni, azt a PRINT utasításban idézőjelek közé tesszük.) A program kezdetén a képernyőt letöröljük (CLS utasítás). Így a program első néhány sora:

```
10 CLS
20 PRINT "MÁSODFOKÚ EGYENLET MEGOLDÁSA"
30 PRINT
40 PRINT "AZ EGYENLET A*X[2+B*X+C=0 ALAKÚ"
50 PRINT
60 INPUT "AZ EGYÜTTHATÓK (ABC-SORRENDBEN)";A,B,C
```

Az 60-as sorszámú sorban kiírandó szöveg szerepel a felsorolt változók előtt, pontos hatásával azonban csak később ismerkedünk meg. A PRINT utasítás, ha mögé nem írunk semmit, egy üres sor kiírását eredményezi. A 40-es sorban a [-jel a hatványozást jelenti (a HT–1080Z a képernyőre ezt a jelet írja).
Most már használhatnánk a másodfokú egyenlet megoldóképletét, de előtte még egy problémába ütközünk: A,B,C tetszőleges valós szám lehet, így 0 is! Lehetséges, hogy a 3 együttható nem valódi másodfokú egyenletet határoz meg, így több lehetőségünk van:

A=0 esetén elsőfokú egyenlettel van dolgunk,
kivéve ha
B=0, mert így már csak nulladfokú,
és ekkor ha
C=0, akkor azonosságról van
szó, egyébként
ellentmondásról.

Ezért a programban szükség van olyan utasításra, amelynek segítségével esetszétválasztást végezhetünk, ez lesz a feltételes utasítás (IF [ha] . . . THEN [akkor] . . . ELSE [különben] . . . – szerkezet). Az IF alapszó után egy feltételt, a másik két alapszó után egy-egy sorszámot írhatunk, a feltétel teljesülése esetén a THEN utáni, nem teljesülése esetén az ELSE utáni sorszámú soron folytatódik a program végrehajtása. Folytathatjuk a program készítését:

```
70 IF A<>0 THEN 200
80 IF B<>0 THEN 150
90 IF C=0 THEN PRINT "AZONOSSÁG" ELSE PRINT "ELLENTMONDÁS"
```

Mint a fenti sorban látható, az IF utasítás ágaira nem csak sorszámot írhatunk, hanem utasításokat is. Ha az ELSE-ág elmarad, akkor is a következő sorszámú soron folytatjuk a programot, amennyiben a feltétel nem teljesül. Ezek értelmezése a korábbiak alapján kézenfekvő, egyetlen probléma van: mi történik, ha az IF valamelyik ágán levő utasítást végrehajtottuk (a feltételtől függően)? Ilyenkor a programot a következő sorszámú soron folytatjuk. A fejezet elején azt mondtuk, hogy a számítógép az utasításokat meghatározott sorrendben, egymás után hajtja végre. Mit kell ebben a pillanatban tennie? A válasz az lehetne, hogy semmit, készen vagyunk a megoldással (a nulladfokú egyenlet megoldásával), ezt azonban valamilyen módon tudatni kellene a számítógéppel is, a programba kell valamilyen megállító utasítás:

```
100 STOP
```

Ezután a 150-es soron meg kell írni a valódi elsőfokú egyenlet megoldását az

$$x = \frac{-c}{b}$$

képlet alapján.

```
150 REM ELSŐFOKÚ EGYENLET
160 X=-C/B
170 PRINT "AZ ELSŐFOKÚ EGYENLET MEGOLDÁSA:";X
180 GOTO 100
```

Két új utasítást találunk ebben a programrészben: az első a REM (REMARK = megjegyzés) utasítás, amely a végrehajtás szempontjából hatástalan, csupán a programszöveg olvasójának ad magyarázatot a program egyes részeinek funkciójáról. A másik újdonság a GOTO (= menj) utasítás, amely közvetlen vezérlésátadást jelent. Megmondhatjuk benne, hogy milyen sorszámú soron kell folytatni a program végrehajtását. Ezután foglalkozhatunk a valódi másodfokú egyenlettel. A megoldások:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

Itt 3 eset lehetséges, a négyzetgyökjel alatt szereplő kifejezés (diszkrimináns) értékétől függően:

- ha negatív akkor nincs valós megoldás,
- ha nulla, akkor egy kétszeres gyök van,
- ha pozitív, akkor két különböző valós megoldás van.

Így a programunk:

```
200 REM VALÓDI MÁSODFOKÚ EGYENLET
210 D=B*B-4*A*C
220 IF D<0 THEN 300
230 IF D>0 THEN 250
240 REM ITT D=0!
243 PRINT "A KÉTSZERES GYÖK:";-B/(2*A)
246 GOTO 100
```

Figyeljük meg a 243-as sorban a kifejezés felírását! A –B/(2*A) kifejezés nem egyenlő a –B/2*A kifejezéssel!!! (ld. 3. FEJEZET!)
A PRINT utasításban a ";" elválasztó jel, hatásával a 6. FEJEZET-ben ismerkedünk meg. Mivel a számítógépnek négyzetgyök jelet nem tudunk beírni, ezért a négyzetgyök számítást másképpen kell jelölni. A BASIC-ben lehetőség van egy SQR nevű függvény használatára, amely egy szám négyzetgyökét adja.

```
250 X1=(-B+SQR(D))/2/A
260 X2=(-B-D[(1/2))/(2*A)
270 PRINT "X1=";X1
280 PRINT "X2=";X2
290 GOTO 100
300 PRINT "NINCS VALÓS MEGOLDÁS"
310 GOTO 100
```

A 250, 260-as sorokban láthatjuk, hogy a négyzetgyökvonást és az osztást kétféleképpen is elvégezhetjük a számítógéppel!
Ezzel a feladat megoldásával végeztünk, nézzük meg a teljes megoldást!

```
 10 CLS
 20 PRINT "MÁSODFOKÚ EGYENLET MEGOLDÁSA"
 30 PRINT
 40 PRINT "AZ EGYENLET A*X[2+B*X*C=0 ALAKÚ"
 50 PRINT
 60 INPUT "AZ EGYÜTTHATÓK (ABC-SORRENDBEN)";A,B,C
 70 IF A<>0 THEN 200
 80 IF B<>0 THEN 150
 90 IF C=0 THEN PRINT "AZONOSSÁG" ELSE PRINT "ELLENTMONDÁS"
100 STOP
150 REM ELSŐFOKÚ EGYENLET
160 X=-C/B
170 PRINT "AZ ELSŐFOKÚ EGYENLET MEGOLDÁSA:";X
180 GOTO 100
200 REM VALÓDI MÁSODFOKÚ EGYENLET
210 D=B*B-4*A*C
220 IF D<0 THEN 300
230 IF D>0 THEN 250
240 REM ITT D=0!
243 PRINT "A KÉTSZERES GYÖK:";-B/(2*A)
246 GOTO 100
250 X1=(-B+SQR(D))/2/A
260 X2=(-B-D[(1/2))/(2*A)
270 PRINT "X1=";X1
280 PRINT "X2=";X2
290 GOTO 100
300 PRINT "NINCS VALÓS MEGOLDÁS"
310 GOTO 100
```

Feladat: Módosítsuk úgy az előző feladat megoldását, hogy ne csak egy, hanem 100 egyenletet tudjon megoldani! A program számolja, hogy hány egyenletet oldott már meg, a 100. után fejezze be működését!

Megoldás: A legegyszerűbb megoldás az, ha az egész programot (úgy, ahogy van) százszor hajtjuk végre. Ennek érdekében a programot foglaljuk keretbe, vezessünk be egy számlálót, amely azt mutatja, hogy már mennyi egyenletet oldottunk meg! Ezt kezdetben nyilván 0-ra kell állítanunk. Minden egyes egyenlet megoldása után a STOP utasítás helyett számoljunk egyet, kezdjük elölről a programot, és ha a századik is megvolt, csak akkor fejezzük be! Az új sorok:

```
  5 I=0
100 I=I+1
110 IF I<100 THEN 50
120 STOP
```

Ezzel a program módosításával végeztünk.

Megjegyzés:

A HT–1080Z sajnos nem ismeri az ékezetes betűket, így a program kiíró utasításai furcsa, magyartalan szöveget eredményeznek.

A BASIC alapvetően két adattípust ismer:

- a szám típusú (numerikus) illetve,
- a szöveg típusú (string) adatokat.

A szám típusú adat fogalma nem igényel különösebbb magyarázatot, olyan adatok ezek, amelyekkel aritmetikai műveleteket (pl.: összeadást, kivonást, szorzást stb.) lehet végezni. (Számon általában – valamilyen véges alakban tárolt – valós számot értünk.)
A szöveg típusú adatok nem szorosan vett „matematikai" információkat tartalmaznak, hanem – mint nevük is utal rá – szöveget, például neveket, szavakat, sőt a BASIC karakterkészletébe tartozó tetszőleges jelből összeállítható jelsorozatot. A jelekhez (karakterekhez) hozzárendeltek egy 1 b y t e -ban tárolható számot, amelyet a karakter kódjának szokás nevezni. E kód segítségével tárolja a gép a szövegeket. (Ld. a C. FÜGGELÉK-et!)
A számokat leírhatjuk a szokásos fixpontos alakban pl.: 987.75, vagy l e b e g ő p o n t o s a n 9.8775E+2. Ez utóbbi a matematikában szokásos féllogaritmikus ábrázoláson alapszik. A fenti példának a $9.8775 \cdot 10^2$ felel meg. (*) Tehát

a valós szám fixpontos alakja:	előjel
	egészrész
	törtrész (tizedespont történték)
lebegőpontos alak:	előjel
	egészrész
	törtrész
	„E"
	exponens előjele
	exponens értéke

(*): Fel kell hívnunk a figyelmet arra, hogy a számítástechnikában a magyar szabályoktól eltérően tizedespontot használnak!

Az előjel elhagyható akár az egészrész, akár az exponens elejéről, ha az pozitív. Az egészrész vagy törtrész egyike elhagyható.
A bonyolultabb lebegőpontos alak alkalmazásának értelme abban keresendő, hogy a számítógépek csak bizonyos véges hosszúságú számokkal képesek számolni pl.: 6-jegyre, így fixpontosan (abszolút értékben) a 999999-nél nagyobb, ill. 0.000001-nél kisebb számmal nem lehetne dolgozni, ez pedig a számolós feladatok többségénél megengedhetetlen.
Megjegyzésként közöljük, hogy akár fixpontosan, akár lebegőpontosan írjuk le a számot a szám az alábbi ún. normalizált lebegőpontos alakban ábrázolódik a számítógépben.

LEÍRVA	SZÁMÍTÓGÉPBEN ÁBRÁZOLVA
–1234	–.123400E+4
6.02E23	+.602000E+24
.000012	+.120000E–4
12E–6	+.120000E–4
0	+.000000E+0

Megjegyzés:

A belső ábrázolás nem decimálisan (azaz tízes alapú számrendszerben) történik (ahogy feltüntettük), hanem binárisan (azaz kettes számrendszerben).

A belső számábrázolásból adódóan ennél a gépnél a legnagyobb abszolút értékű valós szám: 1.70141E+38, a legkisebb (ami nem 0) 5.87748E–39.
Megengedett, hogy az ábrázolási pontosságnál több számjegyű számot is leírjunk.(*) Ekkor a HT–1080Z BASIC a következőként intézkedik: ha csak egy „többlet" jegye van a számnak, akkor ez alapján 6-jegyűre kerekíti. Ha további számjegyeket is tartalmaz a szám, akkor ún. d u p l a p o n t o s számábrázolásra tér át automatikusan.

Példák: az 1.234567 szám helyett az 1.23457 számmal,
az 1234567890 helyett a duplapontos
1234567890.000000 számmal fog számolni.

(*): Érdemes észrevenni, hogy általános törekvés a számítógépes nyelveknél a „normálisnál" nagyobb pontosságú számítás elősegítése, amelyre sokszor eltérő lehetőségeket biztosítanak (●●● ld. az ABC80 BASIC ASCII aritmetika ●●●).

Nézzük, mit kell pontosan tudnunk a HT–1080Z BASIC ezen ábrázolási specialitásáról!
Maximum 16 számjegyet tartalmazhat. Ábrázolása nagyban hasonlít az egyszeres pontosságú valósokéhoz, csak a mantissza tárolása nem három, hanem hét byte-on történik.
Bizonyos célszerűségi indokok miatt a számábrázolás közvetlenül is (még ha a szám nagyságrendje és pontossága nem kívánná meg!) kijelölhető. Ezt a kijelölést tudja a számítógéppel a szám mögé írt !-jel, illetve ■-jel. Ha a számot a !-jel zár le, akkor egyszeres pontosságot írtunk elő, a ■ a dupla pontosság jele.
A lebegőpontosan leírt számoknál e megkülönböztetést az exponens elé írt E – egyszeres pontosság esetén –, illetve D – dupla pontos esetben – jelöli ki.

Példák: 987, 987!, 9.87E2 – egyszeres pontosságúak;
654■, .3333333333333333,
.5■,0.12345D–19 – dupla pontosak.

Néhány BASIC implementáció (így a HT–1080Z BASIC is) lehetővé tesz kifejezetten e g é s z számokkal való számolást is, ekkor az alapvető eltérés a fentitől ennek belső ábrázolásában van (hiányzik az exponens és a törtrész). Az új típus korlátja a kisebb ábrázolási tartomány. A HT–1080Z BASIC esetében ez: –32768 és +32767 között van, szemben a lebegőpontos már megismert tartományával. Az egész számokat is kettes számrendszerben tároljuk, 2 byte-on helyezzük el, s ebből adódik a fenti korlátozás. Az ábrázolásból származó előny a valósokénál kisebb helyfoglalás és gyorsabb műveletvégzés. (Az adatok belső ábrázolását az F. FÜGGELÉK foglalja össze!)
Az egész számokat természetesen a leíráskor is meg kell tudni különböztetni a fixpontos törtrész nélküli számoktól. Erre a HT–1080Z BASIC szabálya szerint az egész szám mögé írt %-jel szolgál.
Az alábbi fixpontos és egész típusú számokat – bár értékük megegyezik – mégis meg kell különböztetnünk:

0	0%
1	1%
–32478	–32478%

A három számtípusról mondottakat gondoljuk végig újra az alábbi táblázat segítségével!

	Megkülönböztető kiegészítés		Érték-tartomány	Legkisebb abszolút érték	Értékes jegyek száma
	fix-pontos	lebegő-pontos			
Dupla pontos	"■"	D	±DMAX	DMIN	16
Egyszeres valós	"!"	E	±EMAX	EMIN	6
Egész	"%"		–32768— +32767	—	—

DMAX = 1.701411834604692D +38 , EMAX = 1.70141E +38
DMIN = 5.877480161902224D –39 , EMIN = 5.87748E –39

Megjegyzések:

A számolás során a valósak fenti „elvi" maximumát kihasználni nem lehet a túlcsordulás veszélye nélkül!

A típusjelző karakter után szóközt ne írjunk, mert hibát okozhat!

Sokszor jól használható az egész típusú számok egy másik értelmezése, amely szerint tekinthetjük az egész típusú adatot logikai értékként is. A –1%-hez (a szám minden bitje 1-es) az igaz, valamint a 0%-hoz (minden bitje 0) a hamis logikai értéket rendeli a BASIC.

Ahogy megkülönböztettük az egész típusú számokat a valósaktól, ugyanúgy világosan el kell különülnie a szöveg típusú (karakteres) adatoknak is. (Ezeket az adatokat idegen szóval 'string'-eknek is szokták hívni.)

Ezért a szöveges-adatot idézőjelpárok (". . . .") közé kell zárni. ●●● Némely BASIC implementáció – ilyen az ABC80 BASIC is – lehetővé teszi azt, hogy az idézőjelpárt aposztrofpárral ('. . . .') helyettesítsük. ●●●

A szöveg tartalmazhat tetszőleges a kezdő "-jel kivételével bármely a C. FÜGGELÉK-ben felsorolt karaktert.

Példák: "a HT–1080Z BASIC szöveges típusú adata"
"ez is 'HT–1080Z BASIC' által megengedett szöveg!"

Ezeket az adatokat szokás összefoglalóan konstans-oknak nevezni, utalva arra, hogy ezek a memória egy területén tárolódva a program futása során nem változnak, ellentétben az úgynevezett változók-kal, amelyekre épp a változtathatóságuk miatt van szükség.

A változók

Lássuk tehát, mit kell tudnunk a változókról!
A matematikában szokásos változók mintájára, a BASIC is használ szimbolikus neveket adatok azonosítására.
Ezeket a neveket szokás azonosítók-nak hívni. A BASIC szabályai szerint

az azonosító: egy betű, vagy
egy betű és egy számjegy lehet.

Használhatók az angol ABC betűi A-tól Z-ig.

FONTOS megjegyzés:
A HT–1080Z BASIC e szabályt némiképpen kiterjeszti, és lehetővé teszi akármilyen hosszúságú ún. alfanumerikus karakterekből (=betűk és számok) álló azonosítók használatát. Három megkötést kell említenünk:
1. az első karaktere csak betű lehet,
2. csak az első kettő jelet használja fel azonosításra!
3. az azonosítók nem tartalmazhatják a BASIC egyetlen utasításának ún. kulcsszavát. (ld. az A. FÜGGELÉKben)

Példák:
A , B9 , Z7 – szabványos változó nevek;
AA , ALFA , ALMUSKA – a HT–1080Z BASIC által megengedett azonosítók, az ALFA és ALMUSKA ugyanazt a változót azonosítja!
9K , FOTO , REUMA , IFJU hibásak, ugyanis az első nem betűvel kezdődik, a továbbiak pedig tartalmaznak BASIC kulcsszót (TO, RE, IF).

Mivel a változók adatok tárolására "születtek" és egy változóban csak egyféle típusú adat tárolható, így a változók típusok szerinti felbontása megegyezik a fentiekben tárgyalt adatokéval. Valóban beszélhetünk szám típusú (ezen belül egyszeres, és dupla pontos valós, illetve egész típusú), valamint szöveg típusú változókról.
Megkülönböztetésük leírásban:

```
egész típus = azonosító %-jellel,
valós típus = azonosító !-jellel
vagy          azonosító (megkülönböztető jel nélkül)
dupla pontos valós típus =
              azonosító ■-jellel
szöveg típus = azonosító $-(dollár) jellel lezárva,
```

Példák:

valós változók:	A, B1, X9!, XENAKIS
dupla pontosak:	AA■, OSSZEG■
egész típusúak:	A%, B1%, X9%, XERXES%
szöveg típusúak:	ASZOVEG$, B1$, X9$, XE$

Természetesen ugyanolyan azonosítójú, de más típus-megjelölésű változók m á s - m á s változókat jelölnek.

A típus deklaráció

A HT–1080Z BASIC ezen kiegészítő jelek "megspórolására" bevezet ún. típus deklarációs utasításokat, amelyekkel a bizonyos betűcsoportokhoz típusokat rendel. A megadott betűvel kezdődő változók automatikusan a deklarált típusba fognak tartozni. A típus deklarációs utasítások:

```
DEFINT   betűtartomány(ok) = egész típusúak
DEFSNG   betűtartomány(ok) = egyszeres pontosságú valósak
DEFDBL   betűtartomány(ok) = dupla pontosságú valósak
DEFSTR   betűtartomány(ok) = szöveg típusúak
```

A betűtartomány megadása:

egy betű = ekkor az adott betűvel kezdődő változók a meghatározott típusba fognak tartozni,

betű-intervallum = ekkor minden a zárt betű-intervallumba eső betűvel kezdődő változó a specifikált típusba fog tartozni.

Példák:
```
DEFINT  I–N,S
DEFSTR  C
DEFDBL  D,X–Z
```

Megjegyzések:

1. a deklarációs utasítások használata esetén nem kell kitenni a típus jelző karaktert. Tehát, ha érvényes a fenti példában szereplő típusdeklaráció, akkor az IRINKO, JANKO, K, LAMPA, MICI, N99 változók egészek, a CHAR, CICA karakter típusú, a DUPLA, XILOPHON dupla pontos változó.
2. ha a deklarációban szereplő betűvel az adott típustól eltérő típusú változót akarunk használni, akkor az azonosító után tett típus kijelölő jellel felülbírálhatjuk a típus deklarációt.

Aritmetikai műveletek és kifejezések

Mivel a konstans és a változó a program futásának minden pillanatában jól meghatározott értékkel rendelkezik (az értéke típustól függően szám vagy szöveg), van értelme rajtuk m ű v e l e t-eket (o p e r á c i ó-kat) végezni.
A s z á m típusúak között a matematikában megszokott aritmetikai műveletek végezhetők: összeadás (jele: +), kivonás (–), szorzás (*), osztás (/), hatványozás (jele: ↑ , kijelzése: [– ●●● néhány implementációban: ** ●●●).

Az ezekkel képzett a r i t m e t i k a i k i f e j e z é s e k kiértékelését egyértelművé teszi az alábbi két szabály:

– először a nagyobb p r i o r i t á s -ú (elsőbbségű) művelet hajtódik végre,
– az azonos prioritásúak között a b a l r ó l elsőként előforduló értékelődik ki először. (Balról-jobbra szabály)

A prioritások sorrendje a legmagasabbal kezdve:
– hatványozás
– szorzás, osztás
– összeadás, kivonás

Megjegyzések:

A hatványozást a kitevő é r t é k e szerint megkülönböztetve egész esetén ismételt szorzással, törtrésszel is rendelkező kitevő esetén a $e^{B \cdot Ln A}$ formulával számol (az A^B helyett).

(●●● Egyes implementációkban (így például az ABC80 BASIC-ben is) a fentitől eltérő a műveletek értelmezése:
A hatványozáskor valós esetben mindig a fenti formulával dolgozik, így ügyelni kell az alap értékére: csak pozitív alap esetén működik helyesen!
Ha az osztás mindkét operandusa egész típusú (tehát az osztó is, az osztandó is), akkor az eredmény is egész típusú lesz. Ez egyszerűen azt jelenti, hogy a hányados törtrészét figyelmen kívül hagyja a számítógép a további számolás során. Így pl.: 5%/2% 2%-t eredményez, illetve –5%/2% –2%-t. ●●●)

Például:

MATEMATIKAI KIFEJEZÉS	BASIC MEGFELELŐJE
$\frac{1}{2} \cdot A \cdot T^2$	0.5*A*T[2
$\frac{4}{3} \cdot R^3 \cdot 3.14$	4/3*R[3*3.14

A fenti kiértékelési sorrendtől el lehet térni zárójelezéssel, ekkor ugyanis a zárójelek közé írt kifejezés önállóan kiértékelődik a környezetében levő műveletek prioritásától függetlenül. A zárójelezés természetesen nagyobb mélységig egymásba skatulyázható. Ilyenkor a "legbelső" zárójelek közé tett kifejezést értékeli ki a gép leghamarabb. (Csak így válik lehetővé pl. a több tagú nevezőkkel számolás.)

$$\frac{R \cdot S}{R + S} \qquad R*S/(R+S)$$

$$\frac{A + B}{(B - C)^2} \qquad (A+B)/(B-C)[2$$

$$A + \frac{B}{C} + D \qquad A+B/C+D$$

$$\frac{1}{\frac{1}{A} + \frac{1}{B}} \qquad 1/(1/A+1/B)$$

Ügyelnünk kell arra, hogy több műveleti jel nem kerülhet egymás mellé még akkor sem, ha az egyik a számhoz tartozó előjel. (Ugyanis az előjel maga is műveleti jel, egy ún. unáris, vagyis egyváltozós operációé).
Ilyen problémák elkerülésére is felhasználhatjuk a zárójelezést.

Függvények

Ahogy a matematikában megszoktuk, célszerű bizonyos gyakran használt f ü g g v é n y -ek alkalmazását lehetővé tenni a programozási gyakorlatban is.
A változókhoz hasonlóan a függvényeket is nevükkel lehet azonosítani. A nevük – mint látni fogjuk – többnyire a matematikából származik. A név mellett meg kell adnunk azt az értéket (argumentumot) is, amely felhasználásával a függvény értékét meg kell határoznunk.

A HT–1080Z BASIC a következő standard (beépített) függvényekkel rendelkezik:

– trigonometrikus függvények közül:

SZINUSZ	(azonosítója: SIN)	
KOSZINUSZ	(COS)	
TANGENS	(TAN)	

– arkusz függvények közül:

ARKUSZ TANGENS	(ATN)	

– gyök-függvények közül:

NÉGYZETGYÖK	(SQR)	
	(SQuare Root = négyzetgyök)	

– hatvány-függvények közül:

e–ALAPÚ	(EXP)	e=2.71828...

– logaritmus függvények közül:

e–ALAPÚ	(LOG)	
●●● egyes implementációkban van 10-es alapú is ●●●		

– egyéb alapvető függvények:

ABSZOLÚT ÉRTÉK	(ABS)	
EGÉSZRÉSZ	(INT)	legnagyobb egész, amely nem nagyobb az adott számnál
EGÉSZ ÉRTÉK	(FIX)	törtrésztől megfosztott szám
ELŐJEL	(SGN)	ha a szám <0, akkor –1, ha =0, akkor 0 ha >0, akkor +1

– speciális függvények:

VÉLETLENSZÁM	(RND)	ld. 10. FEJEZET

– konverziós függvények:

KONVERZIÓ EGÉSZ – TÍPUSBA	(CINT)	
KONVERZIÓ VALÓS – TÍPUSBA	(CSNG)	
KONVERZIÓ DUPLA – PONTOSBA	(CDBL)	értékes jegyeinek száma megegyezik az argumentumbeliével

A függvények leírására a következő általános formai szabály vonatkozik:

függvény-név (függvény argumentum[ok])

Megjegyzés:
a fenti függvényleírási szabályt a BASIC szabványban rögzítették így. A HT–1080Z BASIC-ben csak egy argumentumos függvények léteznek.

Példák:
(a PI változónak előzőleg a 3.14159 értéket adtunk)

$\frac{1}{\sqrt{2 \cdot \pi}} e^{-\frac{x^2}{2}}$ 1/SQR(2*PI)*EXP(–X[2/2)

$B \cdot \sqrt{R^2 - \left(\frac{X}{A}\right)^2}$ B*SQR(R[2–(X/A)[2)

Szöveg művelet és szövegkifejezés

A HT–1080Z BASIC a szövegek között is definiál egy műveletet az ún. k o n k a t e n á c i ó (összeláncolás, másként mondva egymásután írás) műveletét. Jele a "+". Nyilván nem keverhető össze az aritmetikai összeadással hiszen ennek operandusai szöveg típusúak. Ez a művelet egy olyan szöveget eredményez, amely az operandusként megadott szövegek (legyen ez konstans formájában leírva, vagy változóban) egymás után írásával áll elő. (Itt az operandusok sorrendje lényeges!)

Példák:

"BABA"+"OLAJ" = "BABAOLAJ"
"OLAJ"+"BABA" = "OLAJBABA"
ha az A$ a "KISS" vezetéknevet, a B$ a "BÉLA"
keresztnevet tartalmazza, akkor
A$+" "+B$ = "KISS BÉLA"

A szövegen értelmezett, illetve szöveget eredményező függvények is vannak, ezekről és a szövegmanipulációról a 8. FEJEZET-ben lesz szó.

Az értékadó utasítás

Elérkeztünk a BASIC programokban leggyakrabban előforduló utasításhoz, az ún. értékadó utasításhoz. Ehhez még egy általános fogalom megismerésére van szükségünk, nevezetesen a kifejezés fogalmára gondolunk. A kifejezés: konstansok, változók és függvények műveleti jelekkel összekapcsolt sorozata, amelyben zárójelekkel módosíthatjuk a normál (a megismert két szabály szerinti) kiértékelési sorrendet. Az egyszerű fogalmazás érdekében a kifejezés fogalmába beletartozónak vesszük az egyedül álló konstanst is, változót is, függvényt is. Így például szám – (aritmetikai) kifejezések:

1.05, A, A+1.05, A+(COS(PIPI/2*Z+0.77)–R%)[2%

szöveg (string) kifejezések:

"W. Disney: 101 kiskutya" , B$, "ez a COLA"+B$

Az értékadó utasításról alakja:

	sorszám	LET változó = kifejezés
vagy		
	sorszám	változó = kifejezés

(Mint látható, a LET alapszó el is maradhat!)
A változó értékül kapja a kifejezés értékét:
először kiértékelődik az "=" jobb oldalán álló kifejezés, majd ezt az értéket a bal oldalon levő változó felveszi (korábbi értéke elvész).
A "LET" alapszó utal arra, hogy itt nem a szorosan vett matematikai egyenlőségről van szó, hanem egy tevékenységről (tudniillik, hogy "LEGYEN EGYENLŐ").
Így már érthető a matematikailag értelmetlen X=X+1 formájához hasonló BASIC értékadás: LET X=X+1.
Természetes megkötés, hogy a kifejezés és a változó típusa megegyezzen.

Tehát

> aritmetikai kifejezés értékét csak szám típusú változó kaphatja, ill.
> szöveg típusú kifejezés eredményét csak szöveg típusú változó veheti föl!

A HT–1080Z BASIC-re (ahol többféle szám típus is megtalálható) is vonatkozik ez a megkötés azzal a megjegyzéssel, hogy a kifejezésben szerepelhetnek együtt egész és valós (egyszeres valamint dupla pontosságú) konstansok vagy változók ugyanazon művelet operandusaiként. Ilyenkor kiértékelés közben a kifejezésben szereplő legnagyobb pontosságú típusúvá konvertálva használja a BASIC az ettől eltérő típusú tagokat, tényezőket (v e g y e s a r i t m e t i k a).
Ha az aritmetikai értékadásban a baloldalon levő változó típusa illetve a kifejezés típusa nem egyezik meg, akkor automatikus konverzió következik be.
Ez a konverzió egész típusú változó esetén a kifejezés egész részét eredményezi. (Tulajdonképpen az INT függvény alkalmazását jelenti a kifejezés értékére.)

Szöveg típusú értékadásnál egyetlen hibalehetőség adódhat: a szövegkifejezés hossza nagyobb, mint 255.

(●●● Némely BASIC implementációban ennél rugalmasabb a szöveg típusú változók hosszdefiniálási lehetősége.
Így például az ABC80 BASIC-e nem ró formális korlátozást az alkalmazott szöveg típusú változók hosszára. Dinamikusan, az aktuális szabad tárterület határozza meg a változó maximális hosszát. ●●●)

Példák:

```
N$="W. DISNEY"
L$="101 kiskutya"
NL$=N$+" : "+L$
```

A feltételes utasítás megismeréséhez szükség van a reláció és a feltétel fogalmára.

Reláció

A reláció két mennyiség összehasonlítását jelenti. Az összehasonlítás eredménye a logikai igaz vagy a logikai hamis érték. A relációk egyik általános alakja:

szám típusú kifejezés1	relációjel	szám típusú kifejezés2

A következő relációjelek használhatók:

< : kisebb-e
A reláció értéke igaz, ha a kifejezés1 kisebb mint a kifejezés2. A reláció értéke hamis, ha a kifejezés1 nagyobb vagy egyenlő a kifejezés2-nél.

Példa:
2 < 6.3 A reláció értéke igaz.

> : nagyobb-e
A reláció értéke igaz, ha a kifejezés1 nagyobb mint a kifejezés2, egyébként hamis.

Példa: Legyen A értéke 3, B értéke 8.
2*A > B A reláció értéke hamis.

= : egyenlő-e
A reláció értéke igaz, ha a kifejezés1 egyenlő a kifejezés2-vel, egyébként hamis.
(Nem azonos az értékadásban használt =-jellel!!!)

Példa: Legyen J értéke 2.
J = 0 A reláció értéke hamis.

<= : kisebb- vagy egyenlő-e
A reláció értéke igaz, ha a kifejezés1 kisebb vagy egyenlő a kifejezés2-nél, egyébként hamis.

Példa: Legyen B értéke –5.
B[2 <= 10 A reláció értéke hamis.

>= : nagyobb- vagy egyenlő-e
A reláció értéke igaz, ha a kifejezés1 nagyobb vagy egyenlő a kifejezés2-nél, egyébként hamis.

Példa: Legyen X% értéke 5, Y% értéke –2.
X%–3%>= –Y% A reláció értéke igaz.

<> : különbözőek-e
A reláció értéke igaz, ha a kifejezés1 nem egyenlő a kifejezés2-vel. A reláció értéke hamis, ha kifejezés1 egyenlő kifejezés2-vel.

Példa: Legyen P értéke 6.
P/2 <> 6 A reláció értéke igaz.

A fenti összehasonlításokat nemcsak számok, hanem szövegek között is megtehetjük:

szöveg tip. kifejezés1	relációjel	szöveg tip. kifejezés2

A szövegek összehasonlítása a karakterekhez rendelt kód alapján történik (ld. C. FÜGGELÉK). Az a karakter 'kisebb' a másiknál, amelyik kódja kisebb. Ez a sorrend a használható nagybetűknél megegyezik az abc-sorrenddel.
Több karakteres szöveg esetén az első eltérő karakterpár határozza meg a sorrendjüket. Amennyiben egy szöveg egy másik szöveggel kezdődik, a hosszabbat tekintjük nagyobbnak (Példák utolsó sora).

Példák: Az A$ értéke legyen "MA".
"BARNA" <= "MEDVE" A reláció értéke igaz.
"AL"+A$="ALMA" A reláció értéke igaz.
"BARNABA'S" > "BARNA" A reláció értéke igaz.

Feltétel

Egy feltétel értéke vagy igaz (teljesül), vagy hamis (nem teljesül). Így a legegyszerűbb feltétel egy reláció. Újabb (összetettebb) feltételeket úgy kaphatunk, ha logikai műveleti jeleket (NOT, AND, OR) is használunk:

NOT A feltétel alakja: NOT feltétel1
A feltétel igaz, ha feltétel1 hamis és a feltétel hamis, ha a feltétel1 igaz.

Példa: Legyen A értéke 1.
NOT A>0 A feltétel értéke hamis.

AND A feltétel alakja: feltétel1 AND feltétel2
A feltétel igaz, ha feltétel1 és feltétel2 is igaz, egyébként a feltétel hamis.

Példa: X értéke legyen 0.6.
0<=X AND X<1 A feltétel értéke igaz.

OR A feltétel alakja: feltétel1 OR feltétel2
A feltétel hamis, ha feltétel1 és feltétel2 is hamis.
Minden más esetben a feltétel értéke igaz lesz.

Példa: K értéke legyen 1.
K>=2 OR K=1 A feltétel értéke igaz.

A logikai műveleti jelekkel nemcsak relációkat, hanem tetszőleges (összetett) feltételeket is összekapcsolhatunk. Itt említjük meg, hogy a gép az igaz feltétel értékét –1-ként, a hamisat 0-ként ábrázolja. A feltételek felírásánál zárójelezés is használható, amely a kiértékelés sorrendjét írja elő.
Egy feltétel kiértékelése a relációk logikai értékének meghatározásával kezdődik. A logikai műveletek elvégzésének a sorrendje – amennyiben a zárójelezés mást nem ír elő – a következő:

NOT először
AND másodszor
OR harmadszor.

Azonos prioritású műveletek kiértékelése – az aritmetikai műveletekhez hasonlóan – balról jobbra történik.

Példák: Legyen A értéke 1, B értéke 2, C értéke 3.
(A=1 OR B>0) AND C<=0
A kiértékelés a zárójelben lévő résszel kezdődik, így a feltétel értéke hamis (mert C>0).

A=1 OR B>0 AND C<=0
Nincs zárójelezés, így a nagyobb prioritású AND műveletet kell először kiértékelni. A feltétel értéke igaz.

C>B>A
Két azonos prioritású művelet közül először a bal oldali kerül kiértékelésre (C>B): a reláció értéke igaz. Az igaz érték gépi ábrázolása –1, így a –1>A reláció kiértékelése után a feltétel értéke hamis! A 'B értéke A és C közé esik' feltétel helyesen: C>B AND B>A.

A logikai műveleteket feltételek összeállításán kívül bitkezelésre is használhatjuk (ld. 15. FEJEZET).

Feltételes utasítás – elágazás

A feltételes utasítás lehetővé teszi, hogy az utasítások szekvenciális végrehajtásától eltérjünk:

sorszám	IF feltétel	THEN sorszám1	ELSE sorszám2

Ha igaz a feltétel értéke, akkor a program végrehajtása a sorszám1, ha hamis, akkor a sorszám2 sorszámú soron folytatódik.

Példa: Legyen R értéke 5, S értéke 4.
60 IF S>0 AND R=S THEN 100 ELSE 20
A feltétel értéke hamis, így a program végrehajtása a 20-as soron folytatódik.

Az utasítás "ELSE-ág" nélkül is használható:

sorszám	IF feltétel	THEN sorszám1

Hamis feltétel esetén a program végrehajtása az IF utasítást követő soron folytatódik.

Példa: V% abszolút értékének meghatározása.

```
20 IF V%>=0%THEN 40
30 V%=-V%
40 REM Az abszolút értéket meghatároztuk
50 PRINT V%
```

Az IF utasítás végrehajtásakor, ha V% negatív, akkor a 30-as sorban megváltoztatjuk az előjelét.

Az IF utasításban szereplő sorszám1 és sorszám2 helyett egy vagy több BASIC utasítást is írhatunk:

sorszám	IF feltétel	THEN utasítás(ok)1	ELSE utasítás(ok)2

Ha a feltétel igaz, akkor utasítás(ok)1, egyébként utasítás(ok)2 kerül végrehajtásra. Amennyiben a végrehajtott utasítás(ok) nem 'mondanak' mást / GOTO,STOP,END és ON . . . (ld. még 9., 17. FEJEZET /, a vezérlés az IF-et követő sorra adódik.

Példa: Az A és B maximumának meghatározása.

```
. . . .
63 IF A>=B THEN M=A ELSE M=B
70 REM M=(A és B maximuma)
```

Az IF utasítás sorszámos alakjához hasonlóan az ELSE ág itt is elhagyható:

sorszám	IF feltétel	THEN utasítás(ok)1

Példa: V% abszolút értékének meghatározása.

```
. . . .
20 IF V%<0 THEN V%=-V%
30 PRINT V%
```

A THEN illetve az ELSE ágra több utasítást kettősponttal elválasztva kell írni.

Példa: A és B sorbaállítása.

```
. . . .
50 IF A>B THEN M=A : A=B : B=M
52 REM A kisebb vagy egyenlő mint B
. . . .
```

Megengedett a két alak vegyes használata is:

sorszám	IF feltétel	THEN sorszám1	ELSE utasítás(ok)2
sorszám	IF feltétel	THEN utasítás(ok)1	ELSE sorszám2

Ha a feltétel igaz, akkor a THEN ág, egyébként az ELSE ág kerül végrehajtásra.

Példa:

```
520 IF I=J THEN 500 ELSE X=I+J : Y=I-J
```

Fontos megjegyezni, hogy itt a kettőspont az IF utasítás ELSE ágán lévő utasításokat választja el egymástól. Tehát az Y=I-J utasítás az ELSE ág része. (I=5, J=3 esetén X értéke 8, Y értéke 2 lesz.)

Az IF utasításban szerepelhet egy újabb IF utasítás is:

Példa:

```
210 IF K<>0 THEN IF K>0 THEN PRINT "POZITÍV" ELSE PRINT "NEGATÍV"
```

Ilyen esetben az ELSE ág – definíció szerint – a belső IF utasításhoz (K > 0 feltétel) tartozik. Így

ha K értéke pozitív akkor POZITÍV,
ha K értéke negatív akkor NEGATÍV,
ha K értéke nulla akkor semmi sem kerül megjelenítésre.

Megjegyzés:

A feltételes utasításból a THEN szó, sőt a teljes THEN ág bizonyos esetekben elhagyható. Ez azonban a program hordozhatóságát és olvashatóságát nagy mértékben csökkenti, ezért alkalmazását nem javasoljuk.

Feltétlen vezérlésátadás

Az utasítás feltétel nélküli vezérlésátadást eredményez:

```
sorszám      GOTO sorszám1
```

A programvégrehajtás a sorszám1 soron folytatódik. Amennyiben a GOTO utasításban egy nem létező sorra hivatkozunk, az csak akkor derül ki, ha a program végrehajtása a 'hibás' GOTO utasításhoz érkezik.

Példa: Írassuk ki K%-tól V%-ig az egész számokat!

```
. . . . . .
100 I%=K%
105 IF I%>V%THEN 140
110 PRINT I%
120 I%=I%+1
130 GOTO 105
140 PRINT "Kiíratás vége"
. . . . . .
```

Több irányú elágazás

Egy szám típusú kifejezés értékétől függően a program különböző soraira adódik át a vezérlés:

sorszám	ON	szám típusú kifejezés GOTO sorszám1,...,sorszámN

Amennyiben a szám típusú kifejezés értéke

1	akkor	sorszám1,
2	akkor	sorszám2,
...		...
N	akkor	sorszámN

soron folytatódik a program végrehajtása. A szám típusú kifejezés nemcsak egész, hanem valós típusú is lehet. Ebben az esetben a kifejezés értékének az egészrésze alapján történik a sorszám kiválasztása.

Példa: Legyen I% értéke 1, J% értéke 2.
30 ON I%+J% GOTO 150,200,150
A kifejezés értéke 3, így a harmadik sorszám jelöli ki a folytatás helyét. A 150-es sorra adódik a vezérlés.

Az utasítás futási hibát okoz, ha a kifejezés értéke negatív, vagy ha 255-nél nagyobb. Amennyiben a kifejezés értéke nulla vagy nagyobb mint ahány sorszámot megadtunk (N), akkor a végrehajtás a következő soron folytatódik.

Példa: Legyen A értéke 4.75.
90 ON A+1 GOTO 50,100,310,230
91 PRINT "NEM UGRIK"
A valós típusú kifejezés értéke 5.75, az egészrésze 5. Mivel csak négy sorszámot adtunk meg a végrehajtás a 91-es soron folytatódik.

Megállító utasítások

A program futása befejeződik a következő két utasítás valamelyikének hatására:

```
sorszám STOP
sorszám END
```

Az utasítások hatása majdnem azonos:
- a program végrehajtása befejeződik, a gép parancsra vár:
  READY
  >
- a program változóinak az értéke változatlanul megmarad (lekérdezhető);
- a program futtatása a CONT paranccsal folytatható (ld. még 11. FEJEZET).

A két utasítás annyiban tér el egymástól, hogy a STOP utasítás egy üzenetet is ír ki a képernyőre:

```
Break in sorszám
READY
>
```

ahol sorszám a végrehajtott STOP sorát jelzi.

Megjegyzés:

●●● A legtöbb gépen a két utasításnak más-más szerepe van, így a hatásuk jobban eltér. Például az END utasítás után a CONT parancs nem adható ki és a változók értéke 'törlődik'. ●●●

Magyarázó szövegek a programban

Az elkészült program érthetőségét, "olvashatóságát" nagyon megkönnyíthetik a programban elhelyezett magyarázó szövegek:

```
sorszám      REM szöveg
```

Az utasítás "hatástalan", azaz nem változtatja meg a program változóinak értékét és a végrehajtás a következő soron folytatódik. A REM után tetszőleges szöveg állhat, így a kettőspont nem jelenti egy új utasítás kezdetét.

Ciklus

Egy vagy több utasítás ismételt végrehajtását c i k l u s -nak nevezzük. Az ismételten végrehajtott utasítássorozat a c i k l u s m a g .
Ciklust a BASIC F O R és N E X T utasításpárjával szervezhetünk: a ciklusmag kezdetét a FOR, a végét a NEXT utasítás jelzi. Az ismétlések számát és "mikéntjét" a FOR utasításban fogalmazzuk meg:

```
sorszám1   FOR ciklusváltozó = kezdőérték TO végérték
  . . .
  . . .    utasítások (ciklusmag)
  . . .
sorszám2   NEXT ciklusváltozó
```

ahol:
- kezdőérték: szám típusú kifejezés
- végérték: szám típusú kifejezés
- ciklusváltozó: a két utasításban azonos, szám típusú változó, a duplapontos típus kivételével.

Példa:

```
50 FOR  I=1 TO 3
. . . . .  (utasítások)
90 NEXT  I
```

A ciklus végrehajtása a következő módon történik:

a ciklusváltozó felveszi értékül	a kezdőértéket	és a ciklusmag végrehajtódik
– – –„– – –	a kezdőérték+1 –t	– – –„– – –
– – –„– – –	a kezdőérték+2 –t	– – –„– – –
. . . . .	. . . . .	. . . . .

mindaddig, amíg a ciklusváltozó túl nem lépi a végértéket.

Így, ha a ciklusváltozó értéke nagyobb lesz a végértéknél, a ciklus b e f e j e z ő d i k , a program végrehajtása a NEXT-et követő utasításon folytatódik.

Példa: Írjuk át a GOTO utasításnál (előző fejezet) levő példát FOR és NEXT utasítások segítségével!

```
. . . . . .
100 FOR I%=K%TO V%
110 PRINT I%
130 NEXT I%
140 PRINT "Kiíratás vége"
. . . . . .
```

Próbáljuk ki a két programrészletet a

```
10 K%=6%:V%=3%
```

sorral kiegészítve. Vegyük észre, hogy a két megoldás ebben az esetben nem teljesen azonos!

Mint láttuk, a ciklusváltozó értéke egyesével növekszik. A FOR utasításban azonban megadhatunk tetszőleges l é p é s k ö z -t is:

sorszám FOR ciklusváltozó = kezdőérték TO végérték STEP lépésköz

ahol:
– lépésköz : szám típusú kifejezés.

A ciklusváltozó (a kezdőértékből kiindulva) a lépésköz hozzáadásával változik. Pozitív lépésköz mellett a ciklus befejeződik, ha a ciklusváltozó nagyobb mint a végérték. Negatív lépésköz esetén a ciklus akkor ér véget, ha a ciklusváltozó kisebb mint a végérték.

Feladat:

Egy 'm' tömegű testet 'h' magasságból leejtünk. A test helyzeti energiája folyamatosan átalakul mozgási energiává. Ezeket az értékeket foglaljuk táblázatba, a magasság függvényében!

Megoldás:

Egy test helyzeti energiája 's' magasságban:

$$E_h(s) = m * g * s \qquad (g=9.81)$$

A csúcspontban ('h') a mozgási energia nulla, így a test mozgási energiája 's' magasságban:

$$E_m(s) = E_h(h) - E_h(s) = m*g*h - m*g*s$$

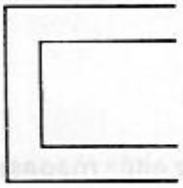
A program bemenő adatai legyenek: a test tömege, az ejtés magassága és a táblázat "lépésköze".

```
 10 PRINT "Szabadon eső test helyzeti"
 15 PRINT "és mozgási energiája"
 20 INPUT "A test tömege";M
 30 INPUT "Az ejtés magassága";H
 40 INPUT "A táblázat lépésköze (poz. szám)";L
 60 REM Az input adatokat beolvastuk
 70 W=9.81*M : REM Az m*g szorzat
 80 E=W*H : REM Az energia a csúcspontban
 90 REM A táblázat fejléce
100 PRINT "Magasság", "helyzeti energia", "mozgási energia"
120 REM A táblázat elkészítése
130 FOR S=H TO 0 STEP -L
140 PRINT S,W*S,,E-W*S
150 NEXT S
160 STOP
```

A PRINT utasítás és az utasításban írható kifejezések, elválasztó jelek részletesebb leírása a 6. FEJEZET-ben található.

Egy programban több ciklus is használható. Megengedett a ciklusok skatulyázása is, az egymást tartalmazó ciklusok ciklusváltozója legyen különböző! Ügyeljünk a FOR és NEXT utasítások helyes párosítására!

Példa: Írassuk ki az 1 és 2 számokból összeállítható összes számpárt!



```
110 FOR I%=1%TO 2%
120 FOR J%=1%TO 2%
130 PRINT I%;",";J%
140 NEXT J%
150 NEXT I%
```

A kiirt számpárok:
1 , 1
1 , 2
2 , 1
2 , 2

Megjegyzések:

- A ciklus lefutása után a ciklusváltozó értéke az lesz, amivel a ciklusmag már nem hajtódott végre.
- A ciklusmagban ne változtassuk meg a ciklusváltozót!
- A ciklusmagon belül a kezdőérték, végérték és lépésköz megváltoztatása nincs hatással a ciklus lefutására.
- Egy ciklusba csak a FOR – ciklusnyitó – utasításon keresztül lépjünk be!
- A NEXT utasításban szereplő ciklusváltozót nem kötelező kiírni. Ez azonban a program hordozhatóságát és olvashatóságát nagy mértékben csökkenti, ezért alkalmazását nem javasoljuk.
- Az egymás után álló két vagy több NEXT utasítás összevonható, például:

```
90 NEXT J
95 NEXT I
```

helyett

```
90 NEXT J,I
```

is írható. Ez azonban a program hordozhatóságát és olvashatóságát nagy mértékben csökkenti, ezért alkalmazását nem javasoljuk.

– A HT–1080Z ciklusa úgy működik, hogy a ciklusváltozót (nem a FOR-nál, hanem) a NEXT utasításhoz érve vizsgálja, így minden ciklus magja legalább egyszer (!) lefut. Például a

```
15 FOR A=2 TO 1
25 PRINT "A=";A
35 NEXT A
```

programot futtatva

```
A= 2
```

íródik ki.

– Amennyiben a ciklus lépésközének nullát adunk meg, a ciklus magja vég nélkül ismétlődni fog.

– Az alábbi furcsaság a számábrázolással magyarázható:

```
10 FOR X=1 TO 1.3 STEP .1
20 PRINT X;
30 NEXT X
```

A kiírt eredmény:

```
1 1.1 1.2
```

A ciklus magja – legnagyobb meglepetésünkre – X=1.3-ra nem hajtódott végre. Ennek az az oka, hogy a kiírt szám az ábrázolt szám kerekítésével keletkezik. Így lehetséges az, hogy az 1-hez 3-szor adva (körülbelül!) 0.1-et, X tartalma (a gépben) nagyobb lesz mint 1.3 (ld. még F. FÜGGELÉK).

Indexes változók

A HT–1080Z BASIC "ismeri" a matematikában használatos v e k t o r és m á t r i x (tömb, táblázat) fogalmát, de műveleteket csak az egyes elemek között tud elvégezni. Így például a mátrixok összeszorzását a programban elemek közti műveletekre kell visszavezetnünk.
Ez a BASIC lehetővé teszi kettőnél több indexű tömbök használatát is. A tömbök elemeit hívjuk i n d e x e s v á l t o z ó k-nak.
Az indexes változók alakja vektor elem esetén:

adott típusú azonosító (index)

mátrix elem esetén:

adott típusú azonosító (sor index, oszlop index)

általában:

adott típusú azonosító (index1, index2 . . . , indexN)

ahol mindegyik index szám típusú kifejezés.

A BASIC nyelvben nemcsak számokból, hanem szövegekből álló tömböket is használhatunk.

Példák:

V(2)	A valós típusú "V" vektor 2. eleme.
A0%(I)	Az egész típusú "A0%" vektor I. eleme.
S$(K%+1%)	A szöveg típusú "S$" vektor (K%+1%). eleme.
M■(P,Q)	A dupla pontosságú "M■" mátrix P. sorában és Q. oszlopában álló eleme.
T$(2%,L)	A szöveg típusú "T$" mátrix 2%. sorában és L. oszlopában álló eleme.

Egy szöveg típusú tömb minden egyes elemében egy-egy tetszőleges szöveget helyezhetünk el.

Példák:

```
110  S$(1)="Arany Ja'nos"
120  T$(2,1)="TOLDI"
```

A tömbök indexelése nullától kezdve, a pozitív egész számokkal történik. Alapértelmezés szerint a legnagyobb használható index a 10. Így tehát a vektorok 11 eleműek (0,1 , . . . , 10-es elem), a mátrixok pedig 11 sorból és 11 oszlopból állnak.

Megjegyzés:

Az alapértelmezéstől eltérően is lehet definiálni a tömbök méretét (ld. Tömbdeklarációk).

Példa:

Az 'a' és 'b' 11 elemű vektorok skaláris szorzata definíció szerint:

$$s = \sum_{i=0}^{10} a_i * b_i$$

Határozzuk meg az "A" és "B" 11 elemű vektorok skaláris szorzatát!

Megoldás:

```
. . .   . . .
200  S=0
210  FOR I%=0%TO 10%
220  S=S+A(I%)*B(I%)
230  NEXT I%
240  REM S tartalmazza a skaláris szorzatot
. . .   . . .
```

Az indexben lévő szám típusú kifejezés nemcsak egész, hanem valós típusú is lehet. Ebben az esetben a kifejezés értékének az egészrésze jelöli ki a tömb elemét. Amennyiben az index értéke a megengedett számoktól különbözik (például: nem 0, 1 , . . , 10 , hanem –1 /, a program futási hibával megáll.

Megjegyzések:

A 3. FEJEZET-ben megismert változókat szokás s k a l á r i s változóknak nevezni (megkülönböztetve őket az indexes változóktól).
A BASIC nyelv jellegzetessége, hogy egy programon belül használhatunk azonos nevű skaláris változót és tömböt. T i l o s azonban azonos nevű, de eltérő indexszámú változót használni!
A FOR utasítás ciklusváltozója nem lehet indexes változó!

Példa:

Számítsuk ki a "C%" mátrix (11 sorból és 11 oszlopból áll) elemeinek összegét! Az eredmény a "C%" skaláris változóba kerüljön!

Megoldás:

```
. . ...
40  C%=0%
50  FOR S%=0%TO 10%
60  FOR T%=0%TO 10%
70  C%=C%+C%(S%,T%)
80  NEXT T%: NEXT S%
. . ...
```

Tömbdeklarációk

A tömbök méretét d e k l a r á c i ó s utasítással adhatjuk meg.
A tömbök indexelése – mint láttuk – alapértelmezés szerint 0-tól 10-ig lehetséges. A felső határt azonban az igényünknek – és a tároló méretének – megfelelően egy D I M utasítással módosíthatjuk:

sorszám DIM adott típusú azonosító (index1, . . . indexN)

ahol index1 , . . . , indexN mindegyike szám típusú kifejezés.

Példák:

```
10  DIM Q(3%)
```

A "Q" valós típusú vektor elemei:
Q(0), Q(1), Q(2), Q(3).

```
20  DIM C%(100,1)
```

A "C%" egész típusú mátrix 101 sorból (0-tól 100-ig) és két oszlopból (0 és 1) áll, azaz a mátrixnak 202 eleme van.

Egy DIM utasításban több tömb méretét is megadhatjuk. Ezeket vesszővel kell elválasztani egymástól.

Példa:

```
15  DIM Q(3%),C%(100,1)
```

Egy tömb h e l y f o g l a l á s a , azaz méretének meghatározása futás közben – dinamikusan történik:

- a végrehajtott DIM utasítás rögzíti a tömb méretét, ezt a program futása során csak egy C L E A R utasítás után lehet megváltoztatni (ld. még 11. FEJEZET);
- ha a tömbre történő első hivatkozás előtt DIM-ben nem adtunk meg méretet, akkor a helyfoglalás az alapértelmezés szerint történik.

Példa:

A program egy futása során n e m lehet a tömbök méretét (CLEAR nélkül) újradefiniálni. (Ld. 11. FEJEZET)

```
10  A(0)=6.3
20  DIM A(20)
. .  . . .
```

A 10-es sorban az "A" vektor 0. eleme értéket kap, azaz hivatkozunk rá. Korábban DIM utasításban nem deklaráltuk, így automatikusan DIM A(10) lép életbe. A 20-as sorban a program futása megszakad, mivel kísérlet történik a vektor újradefiniálására.

Példa:

Változó hosszúságú vektor elemei átlagának kiszámításához a következő kezdeti lépéseket tehetjük meg:

```
10  REM Olvassuk be a vektor méretét
20  INPUT "Elemszám"; N%
30  IF N%<1%THEN PRINT "Hibás érték": GOTO 20
40  REM N% elemszámú "A" vektor deklarálása
50  DIM A(N%-1)
60  REM *** Feldolgozás ***
. .  . . .
```

Ebben a fejezetben az író-olvasó (output-input) utasításokkal foglalkozunk. Ezen utasítások segítségével adatokat jeleníthetünk meg a képernyőn, illetve olvashatunk be a billentyűzetről.

Kiíró utasítás

sorszám	PRINT	felsorolás

A PRINT utasítás hatására a felsorolásban megadott értékek megjelennek a képernyőn, az aktuális karakterpozíciótól kezdve (ahol a c u r s o r éppen áll). A PRINT szó helyettesíthető – a HT–1080Z BASIC-ben a kérdőjel ("?") karakterrel. Jegyezzük meg, hogy a PRINT utasítás kérdőjellel való helyettesítése után, a program listázáskor már a PRINT szót fogjuk találni a kérdőjel helyén! Az üres PRINT utasítás – a PRINT szó után nem írunk semmit – a cursor-t a következő sor elejére állítja. A felsorolás elemei közé elválasztó jelet teszünk.
A felsorolás elemei lehetnek:

- akár szám, akár szöveg típusú kifejezés
- TAB függvény
- @mutató, felsorolás
- USING formátum; felsorolás.

A kifejezés a legegyszerűbb esetben egy konstans. Ekkor a PRINT utasítás hatására az illető konstans jelenik meg a képernyőn. Ügyelni kell arra, hogy a valós típusú konstansok ábrázolási pontossága miatt, a nagyon sok számjegyű konstansok lebegőpontos alakban, bizonyos számú jegyre csonkítva jelennek meg (lásd részletesen a 3. FEJEZET-ben).

Példák:

Utasítás	Hatása
10 PRINT–61.3	–61.3
20 PRINT 3.234;–4.51	3.234 –4.51
30 PRINT 4123456.12	4123456.12
40 PRINT –1.237865345666	–1.237865345666
50 PRINT "HT"	HT
60 PRINT "Pest"	Pest

Megjegyzés:
Nemcsak a szám elé (ott az előjel miatt), hanem utána is szóköz íródik ki a képernyőre! Ld. a 20-as sorban!

A kifejezés másik speciális esetben egyetlen változó. Ilyenkor a PRINT utasítás hatására a változó értéke jelenik meg a képernyőn.

Példák:

Tegyük fel, hogy A(4) értéke 12.3, I% értéke –83, D■ értéke 1.2345678901 és M$ értéke "SZOMBAT".

Utasítás	Hatása
10 PRINT A(4)	12.3
20 PRINT I%	–83
30 PRINT D	1.2345678901
40 PRINT M$	SZOMBAT

Általánosabb esetben a megjelenítést megelőzi a kifejezés kiértékelése (a 3. FEJEZET-ben leírt módon).

Példák:

Legyen A értéke 5, K%(2) értéke –20 és S$ értéke "DUNA".

Utasítás	Hatása
10 PRINT 2*A–SGN(A)	9
20 PRINT K%(2)/2	–10
30 PRINT A/3■	1.666666666666667
40 PRINT S$+"ÚJVÁROS"	DUNAÚJVÁROS

A PRINT felsorolásbeli elemei közé vesszőt vagy pontosvesszőt tegyünk. A pontosvessző közvetlen 'egymásután írást' jelent, a vessző hatására pedig a kiírást a következő t a b u l á t o r pozíción folytatja (0,16,32,48). A negyedik tabulátor pozíció utáni kiírást a következő sor 0. pozícióján kezdi. Ezek a tabulátor pozíciók csak akkor érvényesek, ha soronként 64 karakter a kijelzési formátum! (Ld. D. FÜGGELÉK)
Amikor egy PRINT utasítást pontosvessző karakterrel zárunk le (fejezünk be), a következő PRINT utasítás ott kezdi a kiírást, ahol az előző abbahagyta, vagy a következő INPUT onnan olvas be. Ha vessző karakterrel fejezzük be, akkor a következő tabulátor pozíción, egyébként a sor elejétől kezdi a megjelenítést, illetve az beolvasást.
A TAB(P) függvény alkalmazásával kijelölhetjük, hogy az adott soron belül hányadik oszloptól (pozíciótól) kezdjen írni.

Példa: 10 PRINT TAB(32);A

Az A változó tartalma az aktuális sor 32. oszlopától kerül a képernyőre. Az aktuális oszloptól a 32. oszlopig szóközöket ír.

Példa: 210 PRINT TAB(12);"1";TAB(9);"2"
Az "1" szám a sor 12. oszlopában jelenik meg, a "2" szám azonban nem a 9.-ben, hanem az "1" mögött a 13.-ban, ugyanis előbb kell megadni a kisebb pozíciójút. Ilyen helyzetben a második TAB már hatástalan.

A PRINT@ mutató, felsorolás hatására a képernyőn a mutató által megadott helytől írja ki a felsorolás elemeit. A @-jelnek közvetlenül a PRINT szó után kell következnie. A mutató értéke 0 és 1023 között lehet. (A képernyő 16 soros és 64 oszlopos. Lásd D. FÜGGELÉK.)
A sor és oszlop sorszám a következő lehet:
0 <= sor <= 15 és 0 <= oszlop <= 63

Példa: 10 PRINT@3*64+25,"NYOLC"

A PRINT utasítás az idézőjelek között megadott szöveget a képernyő 3. sorának 25. oszlopától kezdve írja ki.

Feladat:
Töröljük a képernyőt, majd írassuk ki a sorok számait a képernyőre átlósan mindkét irányban! A kiírást kezdjük a képernyő alján!

A képernyő törlése

sorszám CLS

utasítás szolgál.

Az utasítás végrehajtásakor a képernyő tartalma törlődik, és a cursor a képernyő 0. sorának 0. oszlopába áll.

Megoldás:

```
10 CLS : PRINT@20,"S O R O K";
20 FOR I=15 TO 0 STEP -1
30 PRINT@I*64+I*3,I;
40 PRINT@I*64+3*(15-I),I;
50 NEXT I
60 GOTO 60
```

A program utolsó utasítása végtelen ciklust eredményez! Ezt azért csináltuk, hogy a program végrehajtása után az ábránk a képernyőn ne mozduljon el. Programunkat a BREAK billentyű lenyomásával fejezhetjük be!

A PRINT szó után írt 'USING formátum;felsorolás' lehetővé teszi a felsorolásnak a megadott formátumban való kiírását. A formátum csak szöveg típusú kifejezés lehet.

Megjegyzés:

A PRINT utasítás erre a formájára kezdő programírónak nincs szüksége, így ez a rész első olvasásra kihagyható.

A formátum elemei a következő jelek (karakterek) lehetnek:

Szám típusú kifejezések formátumai:

JEL	HATÁSA
■	A szám típusú kifejezés számjegyeinek a számát határozza meg. Ha a számoknak több számjegye van mint a megadott formátumnak (a tizedesponttól jobbra levő számjegyekről van szó), akkor kerekítve jelenik meg az eredmény, ha kevesebb, akkor a számtól balra szóközök, a tizedesponttól jobbra pedig nullák jelennek meg. A tizedespontot a ■-jelek között bárhová tehetjük. Ha a tizedespont és az első ■-jel közé tetszőleges helyre vesszőt teszünk, akkor kiíráskor a tizedespont előtti minden harmadik számjegy elé egy vessző kell. Ha a megadott formátum kisebb mint a szám, akkor egy % jellel kezdődik a kiírás.

Példák:

Utasítás	Hatása
10 PRINT USING"■■.■■";23.455	23.46
20 PRINT USING"■■■.■";23.4	23.4
30 PRINT USING"■■■.■■";–23	–23.00
40 PRINT USING"■,■■■.■";1234.6	1,234.6
50 PRINT USING"■.■";12.5	%12.5

JEL	HATÁSA
**	Ha a formátum elejére két csillagot teszünk, akkor a tizedesponttól balra a számjegyekkel be nem töltött helyekre csillagokat ír. Ezzel a számjegyek számát növeltük.
$$	Ha a formátum elejére írjuk, akkor a legnagyobb helyiértékű számjegy elé egy dollárjel ($) kerül a kiíráskor.
**$	Ha a formátum elejére írjuk, akkor az előző két tulajdonságot tudjuk vele elérni. Azaz minden üres helyre csillag és a legnagyobb helyiértékű elé egy $ jel kerül.

Példák:

Legyen F$ értéke "**■ ■.■", N értéke 46.5, K$ értéke "$$■ ■.■", és L$ értéke "**$■ ■.■".

Utasítás	Hatása
10 PRINT USING F$;N	**46.5
20 PRINT USING K$;N	$46.5
30 PRINT USING L$;N	**$46.5

–	Ha kötőjelet írunk a formátum végére, akkor kiíráskor negatív szám esetén – jelet ír a szám után, pozitív számnál pedig egy szóközt tesz oda.
+	Ha plusz jelet teszünk a formátumunk elejére vagy végére, akkor a szám elé vagy után plusz jel, illetve mínusz jel kerül, a szám előjelétől függően.
[[[[	Ha az eddig említett formátumok után négy felfelé nyilat írunk (felfelé nyíl megadásakor a képernyőn szögletes nyitott zárójel jelenik meg), a megadott formátumú szám lebegőpontos alakban jelenik meg.

Példák:

Utasítás	Hatása
10 PRINT USING "+■.■";4.7	+4.7
20 PRINT USING "■■.■+";3.2	3.2+
30 PRINT USING "■.■–";–1.16	1.2–
40 PRINT USING "■■.■■ [[[[";2■/3	6.67D–01
50 PRINT USING "■■.■■ [[[[";5/3	1.67E+00

Szöveg típusú kifejezések formátumai:

JEL	HATÁSA
!	A felkiáltójel segítségével a szöveg típusú kifejezések első karakterét tudjuk kiírni. Ha több felkiáltójelet adunk meg a formátumunkban, akkor szöveg típusú kifejezések első karakterét vonhatjuk össze velük ('egymásután írást' jelent). A szövegek száma megegyezik a felkiáltójelek számával. Ha a felkiáltójelek között szóköz van, akkor az összevont karakterek között is ugyanannyi szóköz lesz.
% %	Százalékjel, szóköz, százalékjel. A felsorolásban megadott szöveg típusú kifejezésből balról a szóközök száma plusz kettő darab karaktert leválaszt, és csak ezt írja ki a képernyőre.

Példák:

Utasítás	Hatása
10 PRINT USING "!";"ALMA"	A
20 PRINT USING "!!!";"FIÚ";"TOK";"CICA"	FTC
30 PRINT USING "%%";"HOLLÓ"	HO

Megjegyzés:

Ha a formátumban olyan jel (karakter) szerepel, amit nem ismertettünk, vagy nem a megfelelő helyen használtuk, akkor ezek egyszerűen megjelennek a képernyőn!
Ha egy formátumhoz több számot adunk meg, akkor a számok a formátum szerint jelennek meg egymásután!

Példa:

Utasítás	Hatása
10 PRINT USING"A(■)=";5	A(5)=
20 PRINT USING"■■.■";32.4,45.3	32.445.3

Olvasó utasítások

sorszám	INPUT felsorolás

Az utasítás hatására egy kérdőjel jelenik meg a képernyőn, (ahol éppen a cursor áll) és a program adatok beadását várja a billentyűzeten keresztül.

A felsorolásban szereplő azonosítók:
- (skaláris) változó,
- tömbelem (indexes változó).

A kérdőjel elé az adatokhoz tartozó, magyarázó mondatot is írhatunk. Ez a mondat (megjegyzés) a kérdőjel előtt fog megjelenni a képernyőn.

sorszám	INPUT "szöveg";felsorolás

Az INPUT szó után tetszőleges számú azonosítót írhatunk vesszővel elválasztva. A beadott adatokat vesszővel lehet elválasztani egymástól. Ha kevesebb adatot adtunk meg, mint amennyi változót felsoroltunk, akkor a gép két kérdőjelet ír és várja a többi adat megadását.

Példa:
```
10 INPUT "A KÖR SUGARA";R
20 INPUT A(5),B%(3,4),K,Z
30 INPUT N
```

Ha több adatot írunk, mint amennyi változót az INPUT-nál felsoroltunk, akkor a képernyőn a következő üzenet jelenik meg:

```
?EXTRA IGNORED
```

majd a kiírás után a gép a felesleget figyelmen kívül hagyja, és folytatja az utasítások végrehajtását.

A bevitelt a NEWLINE billentyű zárja le. A beírt adatoknak és az azonosítók típusának meg kell egyeznie. Ha pl. szöveget adunk meg szám típusú változónak, akkor az alábbi két sor fog megjelenni a képernyőn:

```
? REDO
?
```

A hibajelzés után az INPUT utasítás után felsorolt összes változó adatait újra kéri.

Az INPUT utasítás a szöveg típusú változóba történő beolvasásnál figyelmen kívül hagyja a bevezető szóközöket és a vessző, vagy pontosvessző karakternél befejezi az adatbevitelt. Tehát, ha szóközöket és egyéb írásjeleket is be akarunk olvasni pl. egy változóba, akkor a kérdőjel után az adatokat idézőjelek közé kell tennünk.

Példa:

```
10 INPUTA$
20 PRINT A$
30 STOP
```

A program futása a képernyőn:

```
READY
>RUN
?"ADATOK : JO,JOBB"
ADATOK : JO,JOBB
BREAK AT LINE 30
READY
>
```

A következő függvény segítségével egy karaktert tudunk beolvasni a billentyűzetről.

INKEY$

A leütött billentyű képe a képernyőn nem jelenik meg. Ha nem ütünk le semmilyen karaktert, akkor az üres szöveget adja – "".

Megjegyzés:

Ez nem a szóközt jelenti!

Feladat:

Várjunk addig, amíg leütnek egy tetszőleges billentyűt!

Megoldás:

```
10 PRINT "HA FOLYTATHATJUK, ÜSSÖN LE EGY BILLENTYŰT!";
20 IF INKEY$="" THEN 20
30 REM A PROGRAM FOLYTATÁSA
   . . .
```

Ezt az utasítást jól használhatjuk pl. táblázatok készítésénél a képernyő teleírása után egy billentyű leütésére töröljük a képernyőt, majd folytatjuk a táblázat listázását, illetve nagy folyamatok leállítására, megváltoztatására.

Példa:

```
. . .
110  REM ELDÖNTENDŐ KÉRDÉS A KÉPERNYŐ 10 SORÁBAN, CSAK I ÉS N!
120  PRINT@ 10*64,"FOLYTASSUK(I/N)?; : REM I-RE IGEN, N-RE NEM
130  A$=INKEY$ : IF A$="" THEN 130
140  IF A$="I" OR A$="i" THEN 160
150  IF A$="N" OR A$="n" THEN STOP ELSE 130
160  REM A PROGRAM FOLYTATÁSA!
       . . .
```

Példa:

```
990  REM PÉLDA NAGY FOLYAMAT LEÁLLÍTÁSÁRA!!
1000 A$=INKEY$
1010 IF A$="V" OR A$="v" THEN 2000 : REM FOLYAMAT VÉGE!!
1020 REM FOLYAMAT VÉGREHAJTÁSA
       .
       .
       .
1990 GOTO 1000
2000 REM FOLYAMAT VÉGE
2010 STOP
```

A következő függvénnyel a cursor aktuális pozícióját tudjuk megkapni egy soron belül.

POS (argumentum)

Az argumentum értéke 0 és 63 között lehet. Az argumentum jelenléte formális, nem befolyásolja az eredményt.

Példa:

```
100  PRINT TAB(32); : B=POS(0)
```

Az utasítás végrehajtása után B értéke 32 lesz.

Feladat:

Írassuk ki táblázatosan a sin(x) és a cos(x) függvény értékét! Az intervallum kezdő- és végpontját, valamint a lépésközt adjuk meg! A kiírásnál ügyeljünk arra, hogy ha a képernyőt teleírtuk, akkor várjunk a táblázat listázásával egy billentyű leütéséig!

Megoldás:

```
 10 CLS : PRINT TAB(10);"FÜGGVÉNY TABELLÁZÁS"
 20 PRINT : INPUT "AZ INTERVALLUM KEZDŐ- ÉS VÉGPONTJA";A,B
 30 INPUT "A LÉPÉSKÖZ";C
 40 J=1
 50 FOR X=A TO B STEP C
 60 IF J<>1 THEN 130
 70 PRINT "FOLYTATHATJUK(I)?";
 80 A$=INKEY$ : IF A="" THEN 80
 90 IF A$<>"I" THEN 80
100 REM FEJLÉC ÍRÁS A TÁBLÁZATHOZ
110 CLS : J=13 : REM J = A KIÍRT SOROK SZÁMA
120 PRINT "X";TAB(5);"SIN(X)";TAB(25);"COS(X)"
130 PRINT X,SIN(X),COS(X)
140 J=J-1
150 NEXT X
160 STOP : REM PROGRAM VÉGE!
```

Az utasításcsoport alapvető feladata, hogy bizonyos változókhoz megadott értékeket rendeljen hozzá.
Az előzőekből ismert, hogy ez megoldható értékadó utasításokkal is (ld. 3. FEJEZET). A mostani fejezetben ismertetett utasítások így tulajdonképpen nem nélkülözhetetlenek, de bizonyos feladatoknál áttekinthetőbbé, rövidebbé teszik a programot. (Ld. Példák.)

Az adattároló utasítás (DATA)

Alakja:

```
sorszám DATA konstans1, konstans2, . . . , konstansN
```

Az utasításban konstans1, konstans2, . . . szám típusú vagy szöveg konstans lehet (ld. 3. FEJEZET). A szöveg konstansokat – eltérően az értékadó utasítástól és általában a szöveg kifejezésektől – nem kötelező idézőjelek (") közé tenni.
A szöveg konstanst idézőjelek (") közé kell tenni, ha:

- a szöveg konstans vesszőt tartalmaz – ugyanis a vessző a konstans végét jelezné,
- a szöveg konstans kettőspontot tartalmaz,
- a szöveg konstans szóközt tartalmaz – ellenkező esetben a szóközöket a konstans elejéről elhagyja.

Példa: 10 DATA 1.7,N+1

Jelentése: a DATA listán van egy 1.7-es számkonstans és egy N+1 szöveges konstans. A DATA utasításban nem írhatók le kifejezések!

A programban előforduló DATA utasításokat úgy képzelhetjük el, mintha a konstansaik egymás után lennének felfűzve egy láncra.

A tárolt konstansokat olvasó utasítás (READ)

Az utasítás alakja:

```
sorszám READ változó1, változó2, . . . , változóN
```

Az utasítás fenti alakjában változó1, változó2, . . . , változóN tetszőleges típusú, indexes és index nélküli változó lehet.
Az utasítás hatására a változók felveszik egy DATA utasítás soronkövetkező konstansainak értékeit.

A READ és DATA utasítások elemeinek megfeleltetése: a program futásának elindításakor egy mutató rááll a program első DATA utasításának első konstansára. Ezt a konstanst kapja értékéül a READ utasítás következő változója, majd a mutató a következő konstansra áll rá.

Példa:

```
10 READ A,B$,C(1),D%
20 DATA 1,"KONSTANS",3.14,5
```

Eredménye:

A=1, B$="KONSTANS", C(1)=3.14, D%=5

Emlékeztetünk rá, hogy a DATA utasítások konstans-listái egy láncra vannak felfűzve, a DATA konstansok mutatója egy DATA utasítás konstansainak elfogyása után a következő DATA utasítás első konstansára fog mutatni. (Ha van következő DATA utasítás.) A READ és a DATA utasítások megfelelő elemeinek típusaikban is illeniük kell egymáshoz az alábbi táblázatnak megfelelően:

READ-ben változó típusa	DATA elem típusa		
	valós	egész	dupla pontos
Valós	+	+	Konverzió és kerekítés
Egész	INT függv.	+	INT függv.
Dupla pontos	Konverzió ld. példa	HIBA	+ ld. 1. példa

Ha a READ utasításban szöveg típusú változó következik, ennek a DATA listán tetszőleges típusú konstans felelhet meg.

Példa:
```
10 DATA 1.1234E10
20 READ A
30 PRINT A
```

Eredménye: 11234000896
A valós–> duplapontos konverzió tehát pontatlan.

Amennyiben nincs ! vagy E betű a konstansban, akkor a valós számot dupla pontosnak tekinti, ha az értékes számjegyek száma 6-nál több.

```
10 DATA 123456789,1.23456789,123456789E0,1.23456789E0
20 READ A■,B■,C■,D■
30 PRINT A■,B■,C■,D■
```

Eredménye:

123456789 1.23456789 123456792 1.234567880630493

```
10 DATA 1 23,  1.23,123E0  ,ASD
20 READ A$,B$,C$,D$
30 PRINT A$;B$;C$;D$
```

Eredménye: 1 231.23123E0 ASD
Látható, hogy az idézőjel nélkül leírt szöveges konstansok elejéről a szóközöket elhagyja.

A DATA listák mutatóját mozgató utasítás (RESTORE)

Az utasítás alakja:

sorszám RESTORE

A RESTORE utasítás hatására a program legelső DATA utasítására áll a mutató. ●●● Egyes implementációkban (pl. ABC80) a RESTORE utasítás paramétere lehet egy sorszám, s ilyenkor a mutató a paraméterben megadott sorszám utáni első DATA utasításra áll. ●●●

READ utasításokkal csak annyi változónak adhatunk értéket ahány konstans a DATA utasításokban van. Azt azonban figyelembe kell venni, hogy egy READ utasításra többször is ráadódhat a vezérlés, egy DATA utasításra pedig RESTORE utasítással többször visszatérhetünk.

Nézzünk néhány példát a fenti utasítások használatára!

Feladat:

döntsük el egy – a billentyűzetről beírt – karakterről, hogy magánhangzó-e!

Megoldás:

```
10  DATA a,e,i,o,u
20  N%=5%
30  A$=INKEY$:IF A$="" THEN 30
40  RESTORE
50  FOR I%=1%TO N%
60  READ B$:IF B$=A$ THEN PRINT "Magánhangzó":GOTO 30
70  NEXT I%
80  PRINT "Nem magánhangzó":GOTO 30
```

Megjegyzés:

mivel a HT–1080Z gépen nincsenek ékezetes betűk, a megoldás nem teljes értékű.

Feladat: pénzösszeg címletezése.

Megoldás:

```
10  DATA 1000,500,100,50,20,10,5,2
20  N%=8%
30  INPUT "Kérem a címletezendő pénzösszeget";P%
40  FOR I%=1%TO N%
50  READ C%:K%=0%
60  IF P%>=C%THEN K%=K%+1%:P%=P%-C%:GOTO 60
70  IF K%<>0%THEN PRINT K%;" db",C%;" Ft-os címlet"
80  NEXT I%
90  IF P%<>0%THEN PRINT P%;" db",1;" Ft-os címlet"
100 STOP
```

Megjegyzés:

A feladat megoldását több helyen módosíthatnánk.
Pl. az 1 Ft-ost is felsorolhatnánk a 10-es sor DATA listájában, s így feleslegessé válna a 90-es sorszámú utasítás. Az adott címletből kifizetendő bankjegyek számát is meghatározhatnánk másképpen:

```
60 K%=P%/C%: P%=P%-K%*C%
```

Az eddigi fejezetekben találkoztunk már ún. szöveg típusú változókkal, vektorokkal, mátrixokkal. Tudjuk, hogy egy szöveg típusú változóban vagy az indexes változó egy elemében egy tetszőleges, a BASIC karakterkészletébe tartozó karakterekből álló szöveget helyezhetünk el. Ezekkel a karaktersorozatokkal műveletet is lehet végezni, "össze lehet őket adni" (ld. 3. FEJEZET). Ezt azonban még nem nevezhetjük szövegkezelésnek! Mit is szeretnénk szövegekkel csinálni? Egy-egy szövegrészletet megvizsgálni, kiemelni, szövegrészletekből új szöveget összeállítani stb. Hogy ezeket a műveleteket kényelmesen elvégezhessük, a BASIC nyelv tartalmaz ún. szövegkezelő függvény-eket. Ezek az egyes BASIC implementációkban nevükben ill. paramétereikben eltérhetnek, itt most a HT–1080Z BASIC nyelvben használható függvényekkel fogunk megismerkedni.
Egy-egy szöveg típusú változóban ill. szöveg típusú vektor egy-egy elemében elhelyezett karaktersorozat nem lehet hosszabb 255 karakternél. Természetesen a változó lehet "üres" is, ilyenkor 0 db karaktert tartalmaz. A számítógép memóriájának egy elkülönített helyén helyezi el az éppen értékkel rendelkező szöveg típusú változók értékeit, azaz a karaktersorozatokat. Ez a terület – ha másképpen nem intézkedünk – mindössze 50 byte hosszú. Ez nagyon hamar kevésnek bizonyulhat, ilyenkor a gép OS hibajelzést küld. Ha szövegek kezeléséhez nem elég (vagy túl sok) az 50 byte a

sorszám CLEAR n

utasítással (lehet parancsként is használni) mondhatjuk meg, hogy hány byte-ot kívánunk szövegkezelésre fenntartani. "n" értékének meghatározásakor figyelembe kell venni az egyidőben használni kívánt szöveg típusú változók hosszát, sőt a szövegkezelő függvények kiszámításakor illetve a konkatenációk elvégzésekor keletkező részeredmények hosszát is, ugyanis ezek a szövegkezeléshez lefoglalt memóriaterületen lesznek – ideiglenesen – elhelyezve. Ne felejtsük el a CLEAR parancs ill. utasítás egyéb hatásait: változók értékének, típusának, tömbök méretének törlését! Ha szükség van rá, célszerű a program első utasításaként elhelyezni. Természetesen nem könnyű dolog a lefoglalandó terület nagyságának becslése, hosszabb programoknál, ahol takarékoskodni kell a memóriával, fontos lehet minden byte. A szükséges méret meghatározásánál segít a

FRE (szöveg típusú változó vagy konstans)

függvény, mely eredményül megadja, hogy mennyi szabad terület, hány byte áll rendelkezésre szövegek kezeléséhez.

Példa:

```
30 PRINT FRE("A")
```

A függvény értékének meghatározásakor az interpreternek végig kell vizsgálni az egész, szövegkezelésre kijelölt területet, hogy mely részei szabadok, ami esetenként hosszabb időbe telhet. A függvény argumentuma lehet egy szöveg konstans, szöveg típusú változó. Szerepe nincs, úgynevezett "álváltozó", használata formai szabályok miatt kötelező: minden függvénynek legalább egy argumentuma kell, hogy legyen.

Térjünk rá a szövegkezelő függvényekre:
Eredményük szempontjából két alapvető csoportra oszthatjuk:

- a függvény eredményül egy új szöveget ad,
- a függvény eredménye szám jellegű.

Azoknak a függvényeknek a neve után, amelyek eredményül egy szöveget adnak, a "$" – szöveg jelet kell írni.
A függvények argumentumai szöveg kifejezések (K$,K1$,K2$-ral jelöljük) vagy aritmetikai kifejezések (tetszőlegesen egész vagy valós típusúak K,K1,K2-vel jelöljük) lehetnek. Szöveg kifejezés lehet egy szöveg konstans, szöveg típusú változó, vagy szöveg függvények és a "+" művelettel előállított karaktersorozat.

Szöveg hosszának megállapítása:

LEN(K$)

A függvény megadja K$ hosszát.

Példa:

ebben és az ezután következő példákban tegyük fel, hogy az A$ változó a "CSIPKEBOKOR" szöveget tartalmazza.

```
25 H=LEN(A$)
```

után H értéke 11.

Szöveg egy részének kiválasztása:
LEFT$(K$,K)
RIGHT$(K$,K)
MID$(K$,K1,K2)
MID$(K$,K1)

LEFT$(K$,K)
– K$ szövegből kiemeli az első K db karakterből álló rész-szöveget.

Példa:
10 B$=LEFT$(A$,6)
után B$ értéke a "CSIPKE" szöveg lesz.

RIGHT$ (K$,K)
– K$ szövegből kiemeli az utolsó K db karakterből álló rész-szöveget.

példa:
30 C$=RIGHT$(A$,5%)
után C$ értéke a "BOKOR" szöveg lesz.
Ha K értéke nagyobb, mint a szöveg hossza, eredményül az egész szöveget kapjuk.

MID$(K$,K1,K2)
– K$ szövegből a K1-edik karaktertől kezdve K2 db karakter hosszúságú szöveget emel ki. A karakterek sorszámozása 1-től kezdődik.

Példa:
80 D$=MID$(A$,2,3)
után D$ értéke a "SIP" szöveg lesz.

Ha a K1-edik karaktertől kezdődő rész-szöveg hossza kisebb, mint K2, eredményül a teljes maradék részt adja.

MID$(K$,K1)
– K$ szövegből kiemeli a K1-edik karaktertől a szöveg végéig tartó rész-szöveget.

Példa:
80 PRINT MID$(A$,7)
hatására a gép a "BOKOR" szöveget nyomtatja ki.

Tudjuk, hogy a gép a számokat kettes számrendszerben ábrázolja, de hogyan ábrázolja például az "A" betűt? A dolog igen egyszerű, minden betűnek és ábrázolni kívánt jelnek megfeleltetünk egy számot és amikor mondjuk "A" betűt szeretnénk a géppel "megjegyeztetni", akkor helyette a neki megfelelő számot jegyzi meg. Ezt a számot a betű vagy jel b e l s ő á b r á z o l á s i k ó d -jának nevezzük. Több kódrendszer is létezik, a HT–1080Z gép az A S C I I kódrendszert használja. A karakterek kódját részletesen a C. FÜGGELÉK tartalmazza. Ez alapján történik a szöveg típusú adatok, változók összehasonlítása is, mint azt a 4. FEJEZET-ben olvashatjuk.

Karakterek belső kódjának kezelése:
ASC(K$)
CHR$(K1)

ASC(K$)

– A K$ szöveg első karakterének ASCII kódját adja meg.

Példa: 100 V=ASC("A")
után V=65, azaz az A betű kódja,
vagy
100 V=ASC(C$) és C$ értéke a "(ABC)" szöveg, akkor az utasítás végrehajtása után V=40, azaz a "(" jel kódja.

CHR$(K1)

– A CHR$ függvény argumentuma egy szám, pontosabban egy karakter ASCII kódja. Az eredmény ennek a kódnak megfelelő karakter (szöveg) lesz.

Példa: 50 A$=CHR$(66)
után A$ a "B" betűt tartalmazza.

Megjegyzés:
A 14. FEJEZET-ben a grafikus üzemmód tárgyalásánál fogunk találkozni az ún. grafikus jelekkel: bizonyos kódokhoz nem a mindennapi életben megszokott jeleket rendeltek, hanem világító és nem világító pontokból összeállított jelet. Ezek megjelenítéséhez is a CHR$ függvény szükséges.

Példa: PRINT CHR$(153)
hatására ▚ jel jelenik meg.
Más kódokhoz pedig egy funkciót rendeltek, például a 28-as kódú karakter kiírása a "HOME" funkció.

PRINT CHR$(28); végrehajtásakor a cursor a képernyő bal felső sarkába áll.

Karaktersorozatok generálása:
STRING$(K1,K2)
STRING$(K1,K$)

STRING$(K1,K2) vagy STRING$(K1,K$)
- Egy olyan szöveget készít, mely a K2 kódú karaktert, vagy K$ karaktert ill. K$ szöveg első karakterét tartalmazza K1-szer.

Példa:

35 PRINT STRING$(20,42)
kiír 20 db"*" karaktert a képernyőre.
Vele egyenértékű:
35 PRINT STRING$(20,"*")

K1 0 és 255 közötti tetszőleges értékű kifejezés, változó vagy konstans lehet.

A gépben a numerikus értékek (egész számok, valós számok) egy – az aritmetikai műveletek elvégzése szempontjából könnyen kezelhető – bináris formában tárolódnak, ugyanis amit mi megszoktunk és használunk, a számok tízes számrendszerben felírt alakja a gép által nehezen kezelhető lenne. Amikor egy numerikus értéket kiíratunk a képernyőre, a gép, hogy számunkra könnyen érthető legyen, belső alakról decimális alakra hozza, "konvertálja", és csak ezután írja ki. Ez a decimális alak azonban már csak egy "karaktersorozat", szöveg a gép számára, ezzel aritmetikai műveletet nem tudna végezni. Lehetőség van arra, hogy előállítassuk a géppel egy szám tízes számrendszerbeli alakját szövegként: STR$; ill. a fordítottjára is: egy szöveget, amely egy tízes számrendszerben felírt számot tartalmaz átkonvertáltathatunk belső formára: VAL.

Konverziók:
STR$(K)
VAL(K$)

STR$(K)

– szöveggé konvertál egy numerikus értéket (változót, konstanst vagy aritmetikai kifejezést), amely lehet egész, egyszeres – vagy dupla pontosságú valós típusú.

Példa:

```
10  A=12.34 : B=2.8E+10
20  PRINT STR$(A)
30  C$=STR$(B) : PRINT C$
40  PRINT STR$(A);"+"RIGHT$(C$,LEN(C$)–1)
50    . . .
RUN
 12.34
 2.8E+10
 12.34+2.8E+10
      . . .
```

Ha a változó értékét PRINT-tel kiíratjuk a szám után egy szóközt is ír a gép, míg ha előbb szöveggé alakítjuk, és úgy írjuk ki, ez elmarad. A szám előtti szóköz – a pozitív előjel helyett – mind a két esetben megjelenik.

VAL(K$)

– numerikus típusúvá alakítja a szövegként megadott számértéket, mely lehet egész, egyszeres vagy dupla pontosságú valós szám.

Példa:

```
10  A$="63" : B$="10"
20  A=VAL(A$+"."+B$)
```

után A értéke: 63.10 lesz.

vagy

```
20  A=VAL(A$+"E"+B$)
```

után A értéke: 63E10, pontosabban 6.3E+11 lesz.

Ha a szöveg nemcsak számot, hanem utána egy betűkből álló szöveget is tartalmaz, a gép csak a számként értelmezhető részen végzi el a konverziót.

Példa:

```
10  A$="100 FORINT"
20  A=VAL(A$)
```

után A értéke: 100 lesz.

Ezek a konverziós függvények hasznos segítséget nyújthatnak szép (helyiértékenként egymás alá igazított táblázatok készítésénél): STR$ függvény, vagy a beolvasás szervezésénél: VAL függvény.

Feladat:

adott három szöveg, jelöljük őket A,B,C-vel! Cseréljük ki az A szövegben a B szöveg minden előfordulását a C szövegre.

(Például:

A szöveg: CSÁMPÁS LÁMPÁS
B szöveg: MPA
C szöveg: BO
Eredmény: CSÁBOS LÁBOS)

Megoldás:

```
 10 REM SZÖVEGEK BEOLVASÁSA
 20 CLS: PRINT "SZÖVEGEK ÁTGYÚRÁSA" : CLEAR 1000
 30 PRINT "KÉREM AZ ALAKÍTANDÓ SZÖVEGET"
 40 INPUT A$
 50 INPUT "MIT CSERÉLJEK";B$
 60 INPUT "MIRE";C$
 70 REM SZÖVEGEK ÁTALAKÍTÁSA
 80 P1=1 : D$=""
 90 IF P1>LEN(A$) THEN 210
100 IF P1>LEN(A$)-LEN(B$)+1 THEN 190
110 FOR P2=P1 TO LEN(A$)-LEN(B$)+1
120 IF B$=MID$(A$,P2,LEN(B$)) THEN 150
130 NEXT P2
140 GOTO 190
150 REM SZÖVEGRÉSZLETEK CSERÉJE
160 D$=D$+MID$(A$,P1,P2-P1)+C$
180 GOTO 90
190 REM MARADÉK ÁTMÁSOLÁSA
200 D$=D$+MID$(A$,P1)
210 PRINT "A KICSERÉLT SZÖVEG:"
220 PRINT D$
230 STOP
```

Figyeljük meg a következőket:

- Az eredmény szöveget D$ változóban állítjuk elő, D$ változót a D$="" értékadással állítjuk "üresre" a 80-as utasításban.
- Két segédváltozót használunk, P1 mutatja, hogy meddig dolgoztuk fel az A szöveget; P2 azt, hogy hol találjuk meg a B szöveget.
- Az algoritmus: megkeressük a B első előfordulását, majd D$-hoz hozzáfűzzük az A szöveg változatlan részét P1-től P2–1-ig (P2–P1 db. betűt) és az új C szöveget, majd P1 változót megnövelve a B hosszával a keresést folytatjuk. Az eljárás befejeződik, ha már nem találtunk B szöveget az A-ban, (ilyenkor a maradék részt még át kell másolni), vagy ha P1 változó már nagyobb, mint az A hossza – ez az az eset, ha az A szöveg pont a B szöveggel végződik.

Feladat:

egy adott szövegben számoljuk meg, hogy az ABC betűi hányszor fordulnak elő!

(Például:

A vizsgált szöveg legyen "ALMA A FA ALATT",
ebben az A betűk száma: 6,
az F betűk száma: 1, stb.)

Vegyük észre, hogy a betűk kódja folyamatosan növekvő, első, legkisebb kódú betű az "A" = 65, utolsó a "Z" = 90 (ld. C. FÜGGELÉK). Ezt kihasználva a számlálást egy vektorban végezzük, legyen a neve B%. A vektor 0. elemében számláljuk az A betűk számát, az 1. elemében a B betűk számát stb., tehát minden betű kódjához a vektornak valahányadik elemét rendeljük:

A kódja: 65 –> B%(0)
B kódja: 66 –> B%(1)
C kódja: 67 –> B%(2)
.
.
.
Z kódja: 90 –> B%(25)

A megfelelő elem indexét tehát úgy kapjuk meg, hogy a betű kódjából levonunk 65-öt. A vektor 26-os indexű elemében pedig számoljuk a szövegben előforduló egyéb jelek számát.

Megoldás:

```
 10  DIM B%(26) : CLEAR 1000
 20  CLS : PRINT "BETŰK GYAKORISÁGÁT SZÁMLÁLÓ PROGRAM"
 30  PRINT "KÉREM A VIZSGÁLANDÓ SZÖVEGET"
 40  INPUT S$
 50  REM SZÖVEG KARAKTERENKÉNTI VIZSGÁLATA
 60  FOR I%=1%TO LEN(S$)
 70  J%=ASC(MID$(S$,I%,1%))
 80  REM INDEX MEGÁLLAPÍTÁSA
 90  IF J%<65%OR J%>90%THEN J%=26%ELSE J%=J%-65%
100  B%(J%)=B%(J%)+1%
110  NEXT I
120  REM KIÍRATÁS
130  PRINT STRING$(64%,"*") : REM CSUPA CSILLAG SOR
140  PRINT "AZ EGYES BETŰK GYAKORISÁGA:"
150  FOR I%=0%TO 25%
160  IF B%(I%)>0%THEN PRINT CHR$(I%+65%);" BETŰ:";B%(I%),
170  NEXT I%
```

```
180  PRINT : PRINT "EGYÉB JELEK:";B%(26%)
190  END
```

Figyeljük meg a következőket:
– A kiírás formája: a vizsgálandó szöveget egy csupa csillag sorral elválasztjuk az értékeléstől. (Megjegyezzük, hogy a 64 db csillag karakter után egy üres sor is keletkezik a képernyőn, mivel a képernyő épp 64 karakter széles, így a PRINT utasítás végén a a soremelés már a következő sor elején történik, és ez egy üres sort eredményez.) Ezt követően kiírjuk a betűk gyakoriságát, 4 oszlopra tagolva (ezért van a 170-es PRINT utasítás végén ","). Csak azokat a betűket írjuk ki, amelyek a szövegben előfordulnak. Magát a betűt a CHR$ függvény segítségével a betű kódját felhasználva írjuk ki, így egy egyszerű ciklussal megoldható az egész ABC kiírása. Az egyéb írásjeleket külön sorba, magyarázó szöveggel írjuk ki. (Itt nem lenne jó a CHR$ függvény!)
– Ahol lehetett, egész típusú változókat és konstansokat használtunk, így gyorsabban fut, és kisebb helyet foglal a program. (Ld. 3. FEJEZET)

Példánk esetén a képernyő a következő képpen néz ki:

```
BETŰK GYAKORISÁGÁT SZÁMLÁLÓ PROGRAM
KÉREM A VIZSGÁLANDÓ SZÖVEGET
ALMA A FA ALATT
**********************************************************************

AZ EGYES BETŰK GYAKORISÁGA:
A BETŰ: 6          F BETŰ: 1          L BETŰ: 2          M BETŰ: 1
T BETŰ: 2
EGYÉB JELEK: 3
READY
>
```

Tudjuk, hogy a HT–1080Z a kisbetűk használatát is lehetővé teszi. Az index képzésének kis korrigálásával a kisbetűvel írt szövegek vizsgálatára is felkészíthetjük a programot:

```
.
.
.
90  REM NAGYBETŰ VIZSGÁLATA
92  IF J%>=65%AND J%<=90%THEN J%=J%-65%: GOTO 100
94  REM KISBETŰ VIZSGÁLATA
96  IF J%>=97%AND J%<=122%THEN J%=J%-97%: GOTO 100
98  REM EGYÉB JEL
99  J%=26%
.
.
.
```

Gyakran előfordul, hogy a programunk több helyén is ugyanazt a néhány utasítást kell végrehajtani. Nagyon egyszerű példa: tegyük fel, hogy több táblázatot szeretnénk készíteni, mindegyik táblázat készítésénél meg kell oldani a képernyő kezelését: képernyő törlés, fejléc kiírása, a táblázat képernyő nagyságú lapokra tagolása stb. Készíthetjük a programot úgy, hogy minden szükséges helyre ugyanazt a kis programrészletet leírjuk, de ez bizony kényelmetlen, főleg ha nem is 2–3 utasításról van szó. Ügyesebb megoldásra is van lehetőségünk: a többször használandó programrészt kiemeljük: csak egyszer, a program egy elkülönített helyén írjuk le. Azokon a pontokon, ahol szükség van rá, valahogy jelezzük, hogy itt most azt a kis különálló programrészletet kell végrehajtani. Ezzel el is érkeztünk új anyagunkhoz, a szubrutinhoz. A szubrutin szó jelentése: "alprogram", ez a neve annak az utasítás-sorozatnak, melyet egyszer, a program egy különálló részén adunk meg, majd több helyen is használunk. Hogyan is megy a használat? Egy újfajta vezérlésátadással jelezzük, hogy az alprogramot szeretnénk végrehajtani, ezt nevezzük s z u b r u t i n h í v á s nak:

```
sorszám GOSUB sorszám1
```

Ennek hatására a vezérlés átadódik a sorszám1-en kezdődő alprogramnak (szubrutinnak), és végrehajtódik a kívánt programrészlet. Igen ám, de jogosan felmerül a kérdés, ezt egy egyszerű feltétlen vezérlésátadással (GOTO utasítással) is megtehetnénk, mi akkor a különbség? A szubrutin végrehajtása után vissza tudunk térni a hívás utáni első utasításra, ami akkor lényeges, ha a szubrutint a program több helyéről is szeretnénk aktivizálni, meghívni, majd annál a pontnál folytatni, ahonnan a szubrutint hívtuk. Ezt már nem tudnánk GOTO utasítással megoldani, hisz a szubrutin végéről GOTO-val csak egy konkrét sorszámra adhatnánk a vezérlést, és nem a mindenkori hívás utáni első utasításra! Hogyan jelezhetjük az alprogram végét, vagyis a szubrutinból való v i s s z a t é r é s -t? Erre külön utasítás szolgál:

```
sorszám RETURN
```

Példa:

```
                .
                .
100  GOSUB 500
110  REM IDE FOG VISSZATÉRNI ELŐSZÖR
                .
                .
                .
300  GOSUB 500
310  REM IDE FOG VISSZATÉRNI MÁSODSZOR
                .
                .
                .
500 REM ITT KEZDŐDIK A SZUBRUTIN
                .
                . a szubrutin utasításai
                .
580  RETURN
```

Amikor a program a 100-as sorszámú utasítást hajtja végre, a vezérlés átadódik az 500-as utasításra, s ott folytatódik a végrehajtás mindaddig, amíg a RETURN utasításhoz nem ér, amelynek hatására a vezérlés visszatér a hívás utáni első utasításra, példánkban a 110-es utasításra. Természetesen amikor a 300-as utasítás hívja meg ugyanazt a szubrutint, az a 310-es utasításra tér vissza. Ha a szubrutin tartalmaz elágazó utasításokat, a szubrutin végét jelző RETURN utasítást több helyre is elhelyezhetjük. Éppúgy, ahogy a ciklusok egymásba skatulyázhatók, szubrutin belsejéből hívhatunk újabb szubrutin(oka)t, azokban ismét lehet szubrutinhívás stb. Ilyenkor a RETURN mindig az utolsó szubrutinhívás mögé tér vissza. Megjegyezzük, hogy a RETURN utasítás végrehajtása egy megelőző GOSUB nélkül értelmetlen, hibajelzést okoz, ezért a fenti programnak valahol a 310-es és 500-as utasítás között STOP v. END utasítást kell tartalmaznia (esetleg vezérlésátadással GOTO) ki kell "kerülnie" a szubrutint. Éppen emiatt célszerű a szubrutinokat a program végére tenni, elkülönítve így a főprogramtól. A program áttekinthetőségét segítik a REM utasítások. Szubrutinokat tartalmazó program esetén mind a hívás helyén érdemes elhelyezni egy magyarázó szöveget, hogy milyen szubrutint hívunk meg, mind pedig a szubrutin első utasítása legyen egy REM utasítás, ami a szubrutin feladatát írja le.

Példa:

```
      .
      .
100   GOSUB 1000 : REM SKALÁRIS SZORZAT SZÁMOLÁSA
      .
      .
      .
1000  REM A ÉS B VEKTOR SKALÁRIS SZORZATÁNAK KISZÁMOLÁSA
1010  REM AZ EREDMÉNY A C VÁLTOZÓBA KERÜL.
1020  C=0
1030  FOR I=1 TO N
1040  C=C+A(I)*B(I)
1050  NEXT I
1060  RETURN
```

Most, hogy megismerkedtünk a szubrutinnal és az általa nyújtott lehetőségekkel, gyűjtsük össze, hogy milyen esetekben érdemes használni. Első és legkézenfekvőbb alkalmazása, amiről a fejezet elején már szó volt: ha ugyanazt a programrészletet többször is tartalmazza a programunk különböző helyeken. Ilyenkor egyszer, egy szubrutinban megírjuk, és ahol szükséges, szubrutinhívással aktivizáljuk. Másik példa a használatára: hosszú, bonyolult (néhány 100 soros) programok esetén a program áttekinthetősége és jól tagoltsága érdekében célszerű az elkülöníthető, önállóbb részleteket (pl. adatok beolvasása, eredmények kiírása, algoritmus egyes részletei stb.) szubrutinnak megírni, így a program "fő" vezérlési struktúrája könnyen áttekinthetővé válik, szinte csak szubrutinhívásokat tartalmaz.
Annak eldöntésekor, hogy a programban helyezzünk-e el szubrutint (és ha igen, mi legyen az), érdemes mind a két szempontot figyelembe venni.
Éppúgy, ahogy a GOTO utasításnak volt ún. számított GOTO (ON GOTO) alakú változata, a szubrutinhívásnak is van számított szubrutinhívás (ON GOSUB) alakja:

sorszám ON kifejezés GOSUB sorszám1, sorszám2, . . . , sorszámN

Végrehajtása úgy történik, hogy a vezérlés a kifejezés értékének egész része szerint a megfelelő, sorszám1, sorszám2-n stb. kezdődő szubrutinra adódik, majd szubrutinból való visszatérés után az ON GOSUB utasítás utáni első utasítás hajtódik végre. Ha a kifejezés értéke <1 vagy $>N$, akkor a következő soron folytatódik a végrehajtás. (Pontosabban ld. még az ON GOTO utasításnál, 4. FEJEZET!)

Példa:

K=1 esetén az első szubrutint hajtjuk végre, K=2 esetén a másodikat stb.

```
.
.
100  ON K GOSUB 500,600,700
110  REM IDE FOG VISSZATÉRNI, A MEGHÍVOTT SZUBRUTINBÓL
.
.
500  REM ELSŐ SZUBRUTIN
.
.
550  RETURN
600  REM MÁSODIK SZUBRUTIN
.
.
680  RETURN
700  REM HARMADIK SZUBRUTIN
.
.
850  RETURN
.
.
```

Feladat:

Írjunk programot, amely egy háromszög oldalainak ismeretében kívánságra kiszámítja a területét vagy kerületét!

Megoldás:

A megoldásnál felhasználjuk az új ismereteinket, jóllehet ezek nélkül is megoldható a feladat. Két szubrutint készítünk, az egyik a kerületet, a másik a területet határozza meg. A felhasználó egy "menü" alapján választhat, a választ felhasználva a megfelelő szubrutint számított szubrutinhívással aktivizáljuk.

```
 10  CLS
 20  PRINT "KÉREM A HÁROMSZÖG OLDALAINAK HOSSZÁT!"
 30  INPUT A,B,C
 40  IF A+B>C AND A+C>B AND B+C>A THEN 60
 50  PRINT "EBBŐL NEM LESZ HÁROMSZÖG!" : GOTO 20
 60  PRINT "MIT MONDJAK MEG?"
 70  PRINT TAB(10);"1. KERÜLET"
 80  PRINT TAB(10);"2. TERÜLET"
 90  INPUT "VÁLASSZON (1 VAGY 2)"; V%
100  IF V%<1%OR V%>2%THEN 90
110  ON V%GOSUB 200,300
```

```
120 INPUT "VAN MÉG KÉRDÉSED (I/N)"; V$
130 IF V$="" THEN 120
140 IF LEFT$(V$,1)="I" THEN 20
150 IF LEFT$(V$,1)<>"N" THEN 120
160 PRINT "VISZONTLÁTÁSRA"
170 STOP
200 REM KERÜLET SZÁMÍTÁSA
210 PRINT "A HÁROMSZÖG KERÜLETE:"; A+B+C
220 RETURN
300 REM TERÜLET SZÁMÍTÁSA
310 S=(A+B+C)/2
320 T=SQR(S*(S-A)*(S-B)*(S-C))
330 PRINT "A HÁROMSZÖG TERÜLETE:"; T
340 RETURN
```

Figyeljük meg a következőket:

– a lehetséges ellenőrzéseket az INPUT utasítás után rögtön elvégezzük.

– a program a szubrutinok alkalmazása miatt könnyen kibővíthető, hogy a háromszög egyéb adatait is ki tudja számolni: be kell venni a "menü"-be az új lehetőséget, megírni a tevékenységet elvégző szubrutint, és az ON GOSUB utasítást kiegészíteni az új szubrutin sorszámával.

– az IGEN/NEM kérdésre adott választ akkor tekintjük "IGEN"-nek, ha az I-vel kezdődik, tehát elfogadja az "IGEN" minden rész-szavát, de például az "ISKOLA" választ is. Hasonlóan kezeljük a "NEM" választ.

Feladat: A BASIC implementációk többsége tartalmazza az INSTR szövegkezelő függvényt. Általában három paramétere van: INSTR (A$,B$,I). A függvény funkciója, hogy eldöntse A$ szöveg tartalmazza-e B$ szöveget az I-edik pozíciótól kezdve. Ha igen, akkor a függvény értéke azt a karaktersorszámot adja meg, ahol A$ szövegben B$ szöveg kezdődik, ha nem, akkor az értéke 0. A HT–1080Z BASIC nyelve nem tartalmazza az INSTR függvényt, írjunk helyette egy szubrutint. Főbb változói a fentiek szerint legyenek A$, B$, I az eredményt tegyük I változóba. Szükségünk lesz még egy segédváltozóra, amit ciklusszervezésre használunk. Jótanácsként mondhatjuk, hogy a szubrutinban használt segédváltozóknak érdemes minél szokatlanabb, ritkán használt nevet adni, így elkerülhetjük, hogy a szubrutin elrontja valamely fontos, a főprogramban is használt változó értékét. Szubrutinunkban ciklusváltozónak ezért egy I9 nevű változót használunk.

Megoldás:

```
1000  REM INSTR SZUBRUTIN
1010  REM PARAMÉTEREI A$,B$,I
1020  IF LEN (A$)-LEN(B$)+1 < I THEN I=0 : RETURN
1030  FOR I9=I TO LEN(A$)-LEN(B$)+1
1040  IF B$=MID$(A$,I9,LEN(B$)) THEN I=I9 : RETURN
1050  NEXT I9
1060  I=0 : RETURN
```

Ezt a szubrutint jól használhattuk volna az előző fejezet első feladatának megoldásához, de alkalmazható a következő feladat megoldásánál is.

Feladat: Írjunk programot, amely kódtáblázat alapján szövegek kódolását, dekódolását végzi el!

Megoldás:

A kódtáblázat egy szöveg lesz: rendre minden betűhöz megadjuk, hogy milyen jel (betű vagy egyéb karakter) kódolja. A kódtáblázat első jele a szóköz kódja, a következő sorrendben az A,B,C, . . . betűk kódjai. Az egyszerűség kedvéért az írásjelek kódolásától most eltekintünk. A kódtáblát A$ nevű változóba olvassuk be DATA utasításból. Ezek után a kódolás úgy történhet, hogy a kódolandó betű ASCII kódjából kiszámítjuk a betű kódtáblázatbeli sorszámát és MID$ függvénnyel kiemeljük, a dekódolás pedig: INSTR szubrutinunk segítségével megkeressük, hogy hányadik a dekódolandó jel a kódtáblázatban, és ebből állítjuk elő először a dekódolt betű ASCII kódját, majd magát a betűt. Ha a kódolás vagy a dekódolás sikertelen, egyaránt a "*" jelet adja a program az értelmetlen jel helyett. Írjuk meg először egy jel kódolását, majd dekódolását végző szubrutint, végül ezek felhasználásával magát a programot!

Kódolást végző szubrutin: B$ tartalmazza a kódolandó karaktert, C$-ban adja meg az eredményt.

```
1100  REM KÓDOLÁS
1110  IF B$="  " THEN C$=LEFT$(A$,1) : RETURN
1120  C=ASC(B$)-ASC("A")+2
1130  IF C < 2 OR C > LEN(A$) THEN C$="*" : RETURN
1140  C$=MID$(A$,C,1)
1150  RETURN
```

Dekódolást végző szubrutin: B$ tartalmazza a dekódolandó karaktert, C$-ban adja meg az eredményt.

```
1200  REM DEKÓDOLÁS
1210  I=1
1220  GOSUB 1000 : REM INSTR SZUBRUTIN
1230  IF I=0 THEN C$="*" : RETURN
1240  IF I=1 THEN C$=" " : RETURN
1250  C$=CHR$(ASC("A")-2+I) : RETURN
```

Kódtáblát beolvasó szubrutin:

```
1300  REM KÓDTÁBLÁZAT BEOLVASÁSA
1310  DATA "QWERTYUJOPASDFGHJKLZXCVBNM"
1320  READ A$
1330  RETURN
```

Ezek felhasználásával a kódoló – dekódoló program:

```
 10  CLEAR 1000
 20  CLS
 30  PRINT "KÓDOLÁS (1) / DEKÓDOLÁS (2)"
 40  INPUT "MELYIKET VÁLASZTJA";V
 50  V=INT (V)
 60  IF V < 1 OR V > 2 THEN 20
 70  PRINT: PRINT "ADJA MEG A SZÖVEGET:"
 80  INPUT S$
 90  E$=""
100  GOSUB 1300: REM KÓDTÁBLÁZAT BEOLVASÁSA
110  FOR J=1 TO LEN(S$)
120  B$=MID$(S$,J,1)
130  ON V GOSUB 1100,1200 : REM KÓDOLÁS/DEKÓDOLÁS
140  E$=E$+C$
150  NEXT J
160  PRINT : PRINT "AZ EREDMÉNY:" : PRINT E$
170  STOP
```

A természeti jelenségek nagy részének lefolyását statisztikus törvények szabályozzák. Gondoljunk például a gázok viselkedésére, a radioaktív bomlásra, vagy a biológiában a fajok fejlődésére. A számítógép nagy előnye, hogy ilyen véletlen jelenségeket nagyon sokszor le lehet játszani, s az így kapott eredményeket sok célra fel lehet használni.
Ehhez nyújt segítséget a számítógépek véletlenszám generátora, amely a [0,1) intervallumban képes egyenletes eloszlású véletlenszerű számokat adni.
Véletlenszámok előállításának legtermészetesebb módja az lenne, ha vennénk egy fizikai folyamatot (elektronikus zaj, radioaktív jelenségek) és egy ebből vett értéket alakítanánk át a számítógép számára valamilyen számmá. A gyakorlatban ez a módszer túlságosan drága lenne, ezért a számítógépek nem ezt használják. Számítógépekkel mindig ál-véletlenszámok-at állítunk elő, amelyek nagyon hasonlítanak a valódi véletlenhez. Véletlenszám generátoraink az előző véletlenszám értékéből állítják elő a következőt. (Véletlenszámok készítésének néhány módszere megtalálható a G. FÜGGELÉK-ben.)
Mivel véletlenszámokat egy nem véletlen algoritmus segítségével készítünk, így ugyanazokat a véletlenszámokat kapjuk, ha ugyanabból a kezdőszámból indultunk ki. Ahhoz, hogy ezt elkerüljük, az első véletlenszám választása előtt tennünk kell valamit, amivel meghatározzuk, hogy milyen számból induljunk ki, és ennek valami véletlen jelenségen kell alapulnia (pl.: a gép belső órájának aktuális állása).

Erre szolgál a BASIC nyelv RANDOM utasítása, alakja:

```
sorszám RANDOM
```

Megjegyzés:

alkalmazására a HT–1080Z BASIC-ben nincs szükség. A véletlenszám-generátor az utasítás használata nélkül sem ugyanabból a kezdőszámból indul.

Ezután már használhatjuk a véletlenszámokat generáló függvényt (RND) , amelynek – első – alakja:

```
RND(0)
```

Megjegyzés:

●●● Egyes BASIC implementációkban az RND függvénynek olyan argumentuma lehet, amely tetszőleges valós szám (P). Ilyenkor az RND függvény a [0,P) intervallumba eső véletlenszámot ad. ●●●

A véletlenszám-generátortól azt várjuk, hogy ne legyenek kitüntetett, gyakrabban előforduló számok. Ezt átfogalmazva: annak a valószínűsége, hogy egy véletlenszám a [0,1) intervallum egy adott részintervallumába esik, csak annak a hosszától függjön (egyenletes eloszlású véletlenszám). Használjuk fel ezt a tulajdonságot, készítsük el 10 darab pénzfeldobás eredményét:

```
10 FOR I=1 TO 10
20 IF RND(0) < .5 THEN PRINT "FEJ" ELSE PRINT "ÍRÁS"
30 NEXT I
40 STOP
```

A véletlenszám-generátor csak a [0,1) intervallumban tud valós számokat készíteni, de nekünk sokszor más véletlenszámok kellenek. A [0,A) intervallumban véletlenszámokat így tudunk előállítani:

```
10 X=A*RND(0)
```

Az [A,B) intervallumban véletlenszámokat így állíthatunk elő:

```
10 X=(B-A)*RND(0)+A
```

A matematikában jártas olvasóink számára:
Ha nem egyenletes eloszlású véletlenszámokra van szükségünk, akkor is adhatunk megoldást, de ehhez természetesen komolyabb matematikai ismeretekre van szükség. Például az L paraméterű exponenciális eloszlás esetén a következő:

```
10 X=-LOG(RND(0)+1E-38)/L
```

Megjegyzés:

Az 1E-38 konstansra a LOG(0) elkerülése miatt van szükség (a 0 ugyanis nincs a LOG függvény értelmezési tartományában).

Más a helyzet, ha egész számokra van szükségünk, ekkor használhatjuk az RND függvény másik alakját:

```
RND(N)
```

Ekkor 1 és a megadott N természetes szám (N > 1) között kaphatunk véletlenszerűen választott egész számot. Például ha lottószámokat akarunk készíteni (1 és 90 közötti egész számokat), nézzük, hogyan lehet ezt tenni:

```
. . . . .
20  DIM A(5)
30  FOR I=1 TO 5
40  A(I)=RND(90)
. . . . .
. . . . .
80  NEXT I
. . . . .
```

A programunk lényegében kész, még arra kell ügyelni, hogy egy számot ne válasszunk ki kétszer:

```
. . . . .
45  IF I=1 THEN 80
50  FOR J=1 TO I-1
60  IF A(I)=A(J) THEN 40
70  NEXT J
. . . . .
```

Feladat: Készítsünk egy olyan szubrutint, amely a radioaktiv bomlást szimulálja.

Megoldás:

A radioaktív bomlás során egy adott típusú anyag (A) fog átalakulni egy másik típusúvá (B). Időegységenként vizsgáljuk a kétféle anyag mennyiségét. A bomlás sebességét ezen időegység alatti átalakulás valószínűségével adjuk meg (P).

A megoldásban az atomoknak egy A$ nevű, N% elemű vektor elemeit feleltetjük meg. A$(I) A-betű, vagy B-betű lehet. A szubrutin lejátsza az egy időegység alatti változásokat. Az S1% nevű, egész típusú változóban számolja az első (A) anyag megmaradt atomjainak számát, S2%-ban a másodikét.

```
1000  REM RADIOAKTÍV BOMLÁST SZIMULÁLÓ SZUBRUTIN
1010  S1%=0 : S2%=0
1020  FOR I=1 TO N%
1030  IF A$(I)="B" THEN S2%=S2%+1 : GOTO 1050
1040  IF RND(0) < P THEN A$(I)="B" : S2%=S2%+1
                ELSE S1%=S1%+1
1050  NEXT I
1060  RETURN
```

A parancsok általában a BASIC programmal csinálnak valamit – listázzák, átszámozzák, futtatják – szemben az utasításokkal, amelyek a felhasználó által megadott adatokkal végeznek műveleteket. Ebből a tartalmi különbségből általában következik az a formai különbség, hogy a parancsokat nem lehet a programban használni. Ez a HT–1080Z esetében nem igaz. Itt a parancsok általában nemcsak közvetlen üzemmódban, de programban is használhatók. Ez alól csak a CONT parancs kivétel. Ha egy parancs programban hajtódik végre, a parancs végrehajtása után a program futása megszakad. Ez alól értelemszerűen kivétel a RUN, továbbá a TRON és a TROFF. (Programokban csak a CLEAR, TRON, TROFF parancs használata értelmes.)

A HT–1080Z gép az alábbi parancsokat ismeri:

AUTO automatikusan sorszámot ír ki a beírandó sorok elejére.
Lehetséges alakjai:

```
AUTO  sorszám1, sorszám2
AUTO  sorszám1
AUTO, sorszám2      Ne használjuk, mert 0-s sorszámot ad!
AUTO
```

A fenti alakokban sorszám1 jelenti a kezdő sorszámot, sorszám2 a növekményt. Az elmaradó sorszám helyére 10-et helyettesít a gép. Az automatikus programsorszámozást a BREAK billentyűvel lehet megszakítani.
Ha a programban már van ilyen sorszámú sor, akkor a kiírt sorszám mögé még egy * karaktert is kiír a gép.
Sorszám1 helyett "." használható az aktuális sor jelzésére.

Példa:	beírt parancs	kiírt sorszámok
	AUTO	10,20,30, . . .
	AUTO 25,12	25,37,49, . . .

CLEAR Ha argumentum nélkül használjuk, akkor 0 értéket tölt az összes numerikus változóba, a karakteres változók hosszát pedig 0-ra állítja. Ha numerikus argumentummal használjuk (pl. CLEAR 100), akkor a változók törlésén kívül az argumentumban megadott számú byte-ot tart fenn karakterláncok tárolására (a fenti példában 100 byte-ot). A gép bekapcsoláskor automatikusan végrehajt egy CLEAR 50 parancsot.
A parancs argumentum nélküli formája nem változtatja meg a szövegek tárolására fenntartott terület méretét.

Megjegyzés:
a parancs valójában a programon kívül szinte mindent töröl. Így például törlődik a szimbólumtábla, érvényét veszti az összes DIM utasítás – ez teszi lehetővé az újra deklarálást – és típusdeklaráló utasítások – DEFDBL, DEFSTR, DEFSNG, DEFINT is. Törlődik a ciklusok visszatérési címeit tároló verem stb.
Gondoljuk meg, hogy a CLEAR parancsot hová írjuk a programban!

CLOAD BASIC programot tölt be kazettáról. A gépben esetleg bentlévő BASIC program törlődik. A parancs lehetséges alakjai:

```
CLOAD
CLOAD "karakter"
CLOAD ■–1, "karakter" vagy CLOAD ■-2, "karakter"
```

A fenti példákban ha megadjuk a "karakter"-t, akkor egy ilyen nevű programot keres a gép. Ha közben más nevűeket talál a szalagon, azok nevét a jobb felső sarokba kiírja. Ha "karakter"-t nem adjuk meg, akkor a kazettán soronkövetkező programot tölti be a gép.
A CLOAD parancsnak ■–1formában a kazettás egység azonosítóját adtuk meg. Ha elhagyjuk, akkor az 1-es sorszámú kazettás egységről tölt be a gép programot.
A CLOAD parancs kiadása előtt a szalagot tekercseljük a megfelelő program elé, és a magnót állítsuk lejátszásra (F1 gomb felengedve).
A megfelelő hangerőt ki kell kísérletezni (ha a gépen nincs automatikus hangerőbeállítás)!
A CLOAD parancs maga indítja a magnót.

CLOAD? A parancs segítségével ellenőrizhetjük, hogy jól sikerült-e a BASIC program mentése. A parancs alakjai:

```
CLOAD?
CLOAD ■–1,? vagy CLOAD ■–2,?
CLOAD? "karakter"
CLOAD ■–1,? "karakter" vagy CLOAD ■–2,? "karakter"
```

A paraméterek jelentése megegyezik a CLOAD-nál leírtakkal.
A parancs byte-onként összehasonlítja a kazettán levő programot a gépben tárolttal. Ha eltérést talál, akkor BAD hibajelzést ad. Ilyenkor a mentést meg kell ismételni. (Ld. CSAVE)

CONT Ha egy program futását megszakítottuk – BREAK billentyűvel vagy STOP utasítással, akkor a CONT paranccsal a megszakított futás folytatható. A megszakítás alkalmas arra, hogy bizonyos változók értékeit megvizsgáljuk, esetleg megváltoztassuk.
Nem hajtható végre a CONT parancs, ha a program megállítása után a programot megváltoztattuk; pl. sorok módosítása, sorok törlése új sorok beszúrása.
A CONT parancs programban nem használható!

CSAVE Ezzel a paranccsal BASIC programot menthetünk ki a kazettára. A parancs lehetséges alakjai:

```
CSAVE "karakter"
CSAVE ■–1, "karakter" vagy CSAVE ■–2, "karakter"
```

A parancsnak kötelező megadni a kimentendő program nevét. Ez 1 darab tetszőleges karakter lehet, kivéve az idézőjelet (").

DELETE A paranccsal sorok törölhetők a BASIC programból. A parancs alakjai:

```
DELETE sorszám1–sorszám2
DELETE –sorszám2
DELETE sorszám
```

Törölhetők tehát egyes sorok, illetve megadott intervallumok. A megadott sorszám-intervallum felső határa létező sor legyen! A "." karakter akármelyik sorszám helyett használható, és az aktuális sort jelöli.

Megjegyzés:
egy sort nem érdemes a DELETE paranccsal törölni, könnyebb csak a sorszámot leírni és a NEW LINE billentyűt lenyomni – a sor így is törlődik.

EDIT az EDIT parancs leírását ld. külön a 12. fejezetben.

LIST A paranccsal a BASIC programot a képernyőre listázhatjuk. A parancs alakjai:

```
LIST
LIST sorszám1
LIST sorszám1–
LIST sorszám1–sorszám2
LIST –sorszám2
```

NEW Törli a programot a memóriából, 0-t ír a numerikus változókba, 0-t ír a karakteres változók hosszába. Nem változtatja meg a karakteres változók tárolására fenntartott memória méretét

Megjegyzés:
a NEW parancs hatása hasonló a CLEAR parancséhoz, de a gépben levő BASIC program is törlődik.

RE Lehetővé teszi a program újrasorszámozását (REnumber). Lehetséges formái:

```
RE
RE sorszám1
RE sorszám1, sorszám2
```

A fenti formáknál a sorszám1 lesz az átszámozott program 1. sorszáma sorszám2 a sorszámozás növekménye. Ha valamelyik elmarad, 10-es értékkel pótolja a gép.
A RE parancs meggondolatlan használata nehezen kideríthető hibákhoz vezethet. (Ld. példa)
A RE parancs nem használható a gép bekapcsolása után közvetlenül. Előbb el kell indítani SYSTEM módban a 12288 címen található programot.

FONTOS:
- A RE parancs nem számozza át a RESUME utasítás (ld. 17. FEJEZET) és a RUN után álló sorszámot,
- szintén nem sorszámozza át a RE parancs azokat a sorszámokat, amelyeknél a sorszámra utaló kulcsszó hibás vagy hiányzik. (Ld. példák)

Példa:

RE	10-től 10-esével átszámozza a programot
RE 100	100-tól 10-esével számoz
RE 20,32	20-tól 32-es növekménnyel számoz.

Hibát okoz a RE használata a következő esetben:

```
10  A=1
20  GOTO 100:REM NINCS ILYEN SORSZÁM
RE  100
```

Az eredmény:

```
100  A=1
110  GOTO 100:REM NINCS ILYEN SORSZÁM
```

Látható, hogy a program 2. sorának értelme teljesen megváltozott.

Megjegyzés:

természetesen általában a REnumber parancs helyesen számozza át a vezérlésátadó utasításokban található sorszámokat is. A fenti hiba akkor fordul elő, ha a hivatkozott sorszám hiányzik.

Példa: nem sorszámozza át a RE parancs a sorszámokat a következő utasításokban:

```
10  GODUB 100:REM A KULCSSZÓ HIBÁS
20  ON I GUSOB 10,20,30:REM A KULCSSZÓ HIBÁS
30  IF A=7 TEHN 10:REM A KULCSSZÓ HIBÁS
40  IF X>Y TEHN 20 ESLE 40:REM A KULCSSZAVAK HIBÁSAK
50  IF Q<6 100:REM A KULCSSZÓ HIÁNYZIK
```

A következő példában viszont a sorszám át lesz számozva:

```
60  OF I GOSUB 10,20,30:REM BÁR AZ ON KULCSSZÓ HIBÁS
```

RUN

```
RUN
RUN sorszám
```

A parancs után egy sorszám is írható, akkor a program a megadott sorszámú sor végrehajtásával indul. A RUN parancs egyben egy CLEAR parancsot is végrehajt. Ha azt akarjuk, hogy a CLEAR ne hajtódjon végre, GOTO-val indítsuk a programot.

SYSTEM monitor üzemmódba vált át a gép. Részletesen ld. a 15. FEJEZET-ben (A gépi szintű programozás eszközei).

TROFF kikapcsolja a nyomkövetést. Részletesen ld. a 17. FEJEZET-ben.

TRON bekapcsolja a nyomkövetést. Részletesen ld. a 17. FEJEZET-ben.

Programsorok javítására, módosítására a HT–1080Z gépnek külön parancsa van az EDIT.
Az EDIT paranccsal csak sorszámmal rendelkező sorok javíthatók.
A javítandó sornak a sorszáma nem módosítható.
Bizonyos futási hibák után a képernyőn automatikusan megjelenik a hibás sor sorszáma, jelezve hogy a sor azonnal javítható. A parancs megengedett formája:

```
EDIT sorszám
```

Az aktuális sor sorszáma "."-al helyettesíthető.
A parancs kiadása után különböző ún. alárendelt parancsokkal, ill. speciális billentyűkkel javíthatunk.
Nagyon vigyázzunk arra, hogy a beszúrási módokban (X, I, H után) a szóköz közönséges karakterként viselkedik, az alárendelt parancsok nevei pedig közönséges karaktereknek minősülnek!

NEWLINE ha ezt a billentyűt szerkesztési módban leütjük, a javított sort tárolja a program, és kilép a szerkesztési módból.

nSZÓKÖZ ha a SZÓKÖZ billentyűt egyszer leütjük, a cursor eggyel jobbra mozdul és láthatóvá válik a következő karakter. Ha a SZÓKÖZ billentyű leütése előtt beírunk egy számot –n– akkor a cursor a megadott számú pozícióval mozdul el jobbra, és n darab karakter válik láthatóvá.

n<– (balranyíl) hatására a cursor balra mozog. Ha n-t nem adjuk meg a cursor eggyel mozdul balra, egyébként a megadott számú pozícióval. Ha nem beszúrási módban használjuk, akkor a karakter csak a képernyőről tűnik el, beszúrási módban az eltűnő karakterek a sorból is törlődnek.

SHIFT/↑ (fölnyíl) hatására kilépünk a beszúrási módokból (ld. X, I, H parancs). Bár kilépünk a beszúrási módból, de változatlanul szerkesztési módban maradunk.

L kilistázza a sornak a cursor-tól jobbra eső részét. Célszerű a szerkesztést mindig L alárendelt paranccsal kezdeni, hogy lássuk mit is javítunk.

X a képernyőre íródik a sornak a cursor-tól jobbra eső része, s beszúrási módba kerülünk. Ezzel az alárendelt paranccsal lehet kényelmesen beszúrni a sor végére, vagy onnan törölni.

I beszúrási módba kerülünk. A cursor helyétől kezdve a beírt karakterek be lesznek szúrva. Beszúrási módból csak a NEWLINE vagy a SHIFT/↑ speciális billentyűkkel lehet kilépni.

A előlről lehet kezdeni a sor szerkesztését. Ha tehát szerkesztés közben hibáztunk, az A alárendelt paranccsal visszatérhetünk az eredeti sorhoz, és újra kezdhetjük a javításokat anélkül hogy a szerkesztési módból kilépnénk. Az A parancs kiadása előtt használjuk a SHIFT/↑ karaktert, hogy kilépjünk a beszúrási módból. (Beszúrási módban ugyanis az A nem minősül parancsnak.)

E a javított sor tárolódik, kilépünk a szerkesztési módból. (NEWLINE használata célszerűbb.)

Q kilépünk a szerkesztési módból, a szerkesztett sor marad az eredeti. (A teljes szerkesztés hatástalan.)

H törli a sornak a cursor-tól jobbra eső részét, majd beszúrási módba kerülünk.

nD a cursor-tól jobbra megadott számú karaktert töröl. Ha nem írunk n-t akkor egy karaktert töröl, egyébként n darabot. A törölt karakterek !-k közé zárva jelennek meg.

nC a cursor-tól jobbra karaktereket cserél ki. Ha csak C-t írunk akkor a cursor-tól jobbra eső karaktert kicseréli az első leütött karakterre. Ha megadjuk n-t, akkor egyszerre több karakter cserélhető.

nSc a cursor-tól jobbra megkeresi a c karakter n. előfordulását és a cursor-t ráállítja. Ha n-t nem adjuk meg, akkor az 1. előfordulást keresi meg. Ha a karaktert nem találja, a cursor a sor végére áll.

nKc a c karakter n. előfordulásáig töröl mindent, és a cursor-t a megmaradó karakterre állítja.

Megjegyzés:
Kezdő programozóknak elégséges az aláhúzott funkciók ismerete.

Példák: a szerkesztési parancsokat ezen a BASIC utasításon mutatjuk be:

100 FOR I = 1 TO 10 STEP .5: PRINT I,I[2,I[3 : NEXT

Most írjuk le EDIT 100 (és persze üssük le a NEWLINE-t)
Ezt látjuk a képernyőn:

100_

ha most néhányszor leütjük a SZÓKÖZ billentyűt, egyre újabb karakterek válnak láthatóvá.
Ha a <– (balranyíl) billentyűt használjuk, a cursor balra mozog, karakterek tűnnek el, de a sorból nem törlődnek.
Ha leütjük az L billentyűt, a sor nem látható része is kiíródik a képernyőre, a cursor új sor elejére áll és ismét kiírja a sorszámot.
Írjuk most be a sor végére a ciklusváltozót. Használjuk ehhez az X parancsot. A képernyőn ezt látjuk:

100 FOR I = 1 TO 10 STEP .5 : PRINT I,I[2,I[3 : NEXT_

Írjuk be a ciklusváltozót és üssük le a NEWLINE billentyűt. (Ha az X parancsban használnánk a balranyíl billentyűt, karaktereket törölne.)
Cseréljük ki a .5 lépésközt 1.2-re!
Írjuk le: EDIT 100
Ezt látjuk:

100_
Nyomjuk le néhányszor a SZÓKÖZ billentyűt, míg ezt nem látjuk:

100 FOR I = 1 TO 10 STEP_
Írjuk le: I1SHIFT/↑
Ezt látjuk:

100 FOR I = 1 TO 10 STEP 1_
Írjuk le: SZÓKÖZC2
Ezt látjuk:

100 FOR I = 1 TO 10 STEP 1.2_
Írjuk le: NEW LINE
Ezt látjuk:

100 FOR I = 1 TO 10 STEP 1.2: PRINT I,I[2,I[3 : NEXT I

A kazettát nem csak programok tárolására használhatjuk. Egy program az adatait beolvashatja kazettáról, illetve az eredményeit kiírhatja kazettára. Erre akkor van szükség, ha:

- nagy tömegű adatot kell többször feldolgozni egy programmal,
- nagy tömegű adatot több programban is használ,
- az adatok gyűjtése és feldolgozása több lépésben, esetleg felváltva történik,
- szükség van az adatok módosítására, javítására stb.

Az adatkezelés a kazettára kiíró- és kazettáról beolvasó utasításokkal történik. Ezekhez hasonló utasításokkal a 6. FEJEZET-ben már találkoztunk, most kisebb módosítással használjuk őket.

Kazettára kiíró utasítás

sorszám	PRINT ■-kazettaszám, felsorolás

A kazettaszám lehet:

- 1 (ez a számítógép beépített magnója);
- 2 (kimenetre csatlakoztatható magnó);

Az utasítás hatására a kazettaszámmal kijelölt magnetofon motorja elindul (a magnónak célszerűen felvétel állapotban kell lennie) és a felsorolásban megadott elemeket kiírja a magnóra – a képernyőre való kiíráshoz hasonlóan, karakteresen. Egy PRINT utasítással maximum 247 db karaktert írhatunk ki kazettára.
A felvétel után a magnetofon motorja megáll. Az adatrekord egy rekord bevezető részből (257 byte), valamint a program által kiírt valódi adatokból áll. A felsorolás bármilyen lista lehet, amely a korábban definiált PRINT utasításban megengedett.

Megjegyzések:

- Kivéve a '@mutató,felsorolás' lehetőséget!
  Az idézőjelek között felírt egyéb írásjelek közül a kettőspontot nem tudjuk visszaolvasni! Lásd részletesen a kazetta olvasó utasításnál!
  A vessző a kazettán adat elválasztó szerepet tölt be, ezért kell kiírnunk!

Példa:

```
10 PRINT ■–1,"ADATOK";",";1.2;",";3.14;",";"KÁROLY";",";
"ADATOK VÉGE"
20 PRINT ■–1,N;",";K(5)
30 PRINT ■–1,D$
```

Feladat: Írjuk ki N ember adatait kazettára! Az I. ember adatai a következők legyenek: N$(I)-ben legyen a neve, S(I)-ben a születési éve és M(I)-ben pedig a testmagassága!

Megoldás: A kiírás elkezdése előtt várjunk addig, amíg a magnót felvételre nem állították! Az adatok elé írjunk egy címet, amivel azonosítani tudjuk beolvasáskor! Írjuk ki az emberek számát is a cím után! A kiírás végét jelezzük a képernyőn!

```
 60 PRINT "ÁLLÍTSA A MAGNÓT FELVÉTELRE!"
 70 PRINT "HA KÉSZ, ÜSSÖN LE EGY BILLENTYŰT !";
 80 IF INKEY$="" THEN 80
 90 PRINT "A KIÍRÁS ELKEZDŐDÖTT!"
100 PRINT #-1,"FELMÉRÉS ADATAI. 1983" : REM A CÍM
110 PRINT #-1,N : REM EMBEREK SZÁMA
120 FOR I=1 TO N
130 REM KIÍRÁS KÖV.
140 PRINT #-1,N$(I);",";S(I);",";M(I)
150 NEXT I
160 REM VÉGE
170 PRINT "***VÉGE A KIÍRÁSNAK !***"
180 STOP
```

Megjegyzés:

Az emberek adatait a tömbökbe a 60-as utasítás előtt már elhelyeztük.

- Célszerű egy PRINT utasítással minél több adatot kivinni a kazettára! Mint már említettük egy adatrekord írása az adatokon kívül bevezető rész kiírását is jelenti. Emiatt egy rekord kazettára kiírása (egy PRINT utasítás) hosszú lesz.
- Több adat egy PRINT utasítással történő kiírásakor, a kiírt adatok közé írjunk karakteresen vesszőt (lásd 140-es sor). Ugyanis beolvasáskor – a billentyűzetről való beolvasáshoz hasonlóan – a kiírt vesszők karakter választja el egymástól az adatokat (lásd 6. FEJEZET olvasó utasítás).

Kazettáról olvasó utasítás

sorszám	INPUT ■–kazettaszám, felsorolás

Az utasítás végrehajtásakor a magnó motorja elindul (a magnónak lejátszás állapotban, és az F1 gombnak felengedve kell lennie!). A felsorolásban megadott változók a kazettán tárolt adatok beolvasásával kapják meg értéküket. A soron következő változóhoz tartozó adatok végét vagy egy vessző karakter jelzi – ugyanúgy mint a billentyűzetről való beolvasásnál –, vagy az adatok vége. Miután a változók megkapták értéküket, a magnó motorja leáll és az utasítás végrehajtása befejeződik.
Ha a kiírt adatok között kettőspont karakter előfordul, vagy több adat esetén, a következő hibaüzenet jelenik meg a képernyőn:

?EXTRA IGNORED

A kettőspont utáni karakterek, illetve a többi adatok nem hozzáférhetőek. A hibaüzenet kiírása után a program folytatja az utasítások végrehajtását.

Példa:

```
10 INPUT ■–1,S(5),B%,J■,X$
20 INPUT ■–1,K
30 INPUT ■–2,D$,Y(3,2,1)
```

Megjegyzés:

A 30-as sor INPUT utasítása a külső magnóról olvas be.

- Egy INPUT utasítással max. 247 karakter olvasható be.
- Ha kevesebb adatot olvasunk be, mint amennyit egy rekordba kiírtunk, akkor a következő INPUT már a következő rekordot olvassa.

Feladat: Olvassuk be a korábbi feladatunkban kiírt adatokat, az ott ismertetett tömbökbe! A beolvasás megkezdése előtt várjunk, amig a magnót lejátszásra állítják! Írjuk ki a képernyőre az olvasás kezdetét és végét!

Megoldás:

```
.
.
.
170 PRINT "ÁLLÍTSA A MAGNÓT LEJÁTSZÁSRA!"
180 PRINT "HA KÉSZ, ÜSSÖN LE EGY BILLENTYŰT !";
190 IF INKEY$="" THEN 190
195 PRINT TAB(20);"OLVASÁS KEZDETE"
200 INPUT ■-1,A$
210 IF A$<>"FELMÉRÉS ADATAI. 1983" THEN 200
220 INPUT ■-1,N : REM EMBEREK SZÁMA
230 DIM N$(N),S(N),M(N)
240 PRINT "AZ EMBEREK SZÁMA:";N
250 FOR I=1 TO N
260 INPUT ■-1,N$(I),S(I),M(I)
270 NEXT I
280 REM BEOLVASÁS VÉGE
290 PRINT "VÉGE A BEOLVASÁSNAK!!"
300 STOP
```

A HT–1080Z számítógép karakterkészlet-ében (ld. C. FÜGGELÉK) találhatók olyan karakter kódok, amelyeknek grafikus képek – szimbólumok – felelnek meg. Ez lehetővé teszi, hogy különböző grafikus ábrákat, alakzatokat tudjunk rajzolni a képernyőre. Mint tudjuk, a képernyő 16 soros és 64 oszlopos (ld. D. FÜGGELÉK). Egy karakter pozíció hat pontból (3 sor és 2 oszlop) álló grafikus szimbólumot tartalmaz. Különböző általunk összeállított ábrákat és egyéb jeleket tetszés szerint jeleníthetünk meg a képernyőn. Ezen úgynevezett grafikus karakterek kódjai és képei a C. FÜGGELÉK-ben láthatók.
A grafikus szimbólumokkal történő rajzoláson kívül rendelkezésünkre állnak utasítások is, amelyekkel a képernyő bármely pontja kigyújtható, illetve törölhető.
Kigyújtható – az adott pontot megjelenítjük, azaz világítani fog. Törölhető – az adott pontot kioltjuk, azaz nem fog világítani. A képernyő 48 grafikus sorból és 128 grafikus oszlopból áll (ld. D. FÜGGELÉK). A grafikus utasítások segítségével, az így felosztott képernyő tetszőleges pontját kigyújthatjuk, illetve törölhetjük.
A grafikus utasítások sor és oszlop határai a képernyőn a következők:

$0 <= \text{sor} <= 47$ és $0 <= \text{oszlop} <= 127$

Most pedig nézzük az utasításokat! A képernyő egy pontjának a kigyújtása:

sorszám	SET (oszlop, sor)

Példa: 10 SET (10,21)
A képernyő 10. oszlopában és 21. sorában lévő pontot kigyújtja, megvilágítja.

A képernyő egy pontjának a törlése:

sorszám	RESET (oszlop, sor)

Példa: 110 RESET (120,43)
A képernyő 120. oszlopában és 43. sorában lévő pontot törli.

Arról pedig, hogy az adott pont világít-e, vagy nem, a következő – logikai értékű – függvény segítségével győződhetünk meg:

POINT (oszlop, sor)

Példa:

```
420 IF POINT (30,29) THEN PRINT "A PONT VILÁGÍT" ELSE PRINT "A PONT
NEM VILÁGÍT"
```

Ha a sor vagy oszlop koordináták a képernyő tartományán kívülre mutatnak, akkor a BASIC interpreter hibát jelez.

Feladat: Gyujtsunk ki és oltsunk ki fénypontokat a képernyőn véletlenszerűen! Figyeljük azt is, hogy a véletlenszerűen kiválasztott pont világít-e! Ha igen akkor oltsuk ki, ellenkező esetben pedig gyujtsuk ki a pontot! A megoldásnál használjuk a véletlenszámokat generáló függvényt (RND – bővebben a 10. FEJEZET-ben)! A rajzolásnál ügyeljünk arra, hogy a pont a képernyőre kerüljön. Ezért a pont koordinátáit a következőképpen állítjuk elő:

```
20 O=RND(128)-1   :  REM  0 <= OSZLOP <= 127

30 S=RND(48)-1    :  REM  0 <= SOR <= 47
```

Az így kiválasztott pont vizsgálatára a POINT függvényt használjuk!

Megoldás:

```
10 CLS
20 O=RND(128)-1
30 S=RND(48)-1
40 IF POINT (O,S) THEN RESET (O,S) ELSE SET (O,S)
50 GOTO 20
```

A programot csak a BREAK billentyű leütésével fejezhetjük be!

Feladat: Írjunk egy olyan programrészletet – szubrutint –, amely egy koordinátával adott pontot mozgat adott irányban, állandó sebességgel. Adjuk meg a pont mozgatás határát is! A szubrutinnak a kezdőértékeket a következő változókkal adjuk át:

I – a mozgás iránya: 1 ha a képernyőn lefelé halad,
2 ha felfelé halad,
3 ha balra halad és
4 ha jobbra halad a pont;

V – a mozgás sebessége: 1 és 10 közé eső szám,
10 – a legnagyobb és
1 – a legkisebb sebességű.

S – a pont sor koordinátája;
P – a pont oszlop koordinátája;
Z(I) – adott irányban a pont mozgatás határa.
Fontos, hogy S,P,Z(I) értékei a képernyő intervallumába essenek!

Megoldás:

```
. . .
500 REM SZUBRUTIN PONT MOZGATÁSÁRA
510 S1=S : P1=P
520 SET(P,S) : REM PONT RAJZOLÁS
525 REM SEBESSÉGTŐL FÜGGŐ VÁRAKOZÁS
530 FOR J=1 TO 500 STEP V : NEXT J
540 ON I GOTO 570,600,630,660
550 RESET(P1,S1) : REM AZ ELŐZŐ HELYZET TÖRLÉSE
560 GOTO 510
570 REM LEFELE
580 S=S+1
590 IF S< = Z(I) THEN 550 ELSE RETURN
600 REM FELFELE
610 S=S–1
620 IF S > = Z(I) THEN 550 ELSE RETURN
630 REM BALRA
640 P=P–1
650 IF P > = Z(I) THEN 550 ELSE RETURN
660 REM JOBBRA
670 P=P+1
680 IF P < = Z(I) THEN 550 ELSE RETURN
. . .
```

Az 530-as sorban található az ún. lassító ciklus. Ezzel két pont kigyújtása között eltelt időt lehet, a V értékétől függően meghatározni (a pont mozgatás sebessége).

Feladat: Mozgassunk egy adott pontot a képernyőn véletlenszerűen! A megoldásnál használjuk fel az előző feladat megoldásának egy részét is!
I – a mozgás iránya : szintén 4 irányban mozoghasson ;
V – a mozgás sebessége : ugyanaz mint az előző feladatnál ;
S,P – a pont kezdeti koordinátái :
N – a véletlenszerű mozgás (bolyongás) lépéseinek száma. /50/

Megoldás:

```
 10 REM BOLYONGÁS
 20 CLS : REM KÉP. TÖRLÉS
 30 S=RND(48)-1 : P=RND(128)-1
 40 V=RND(10) : N=50
 50 FOR K=1 TO N
 60 I=RND(4)
 70 REM PONT MOZGATÁS
 80 S1= S : P1=P
 90 SET(P,S)
100 FOR J=1 TO 500 STEP V : NEXT J
110 ON I GOTO 120,140,160,180
120 REM LE
130 IF S+1 > 47 THEN 210 ELSE S=S+1 : GOTO 200
140 REM FEL
150 IF S-1 < 0 THEN 210 ELSE S=S-1 : GOTO 200
160 REM BAL
170 IF P-1 < 0 THEN 210 ELSE P=P-1 : GOTO 200
180 REM JOBB
190 IF P+1 > 127 THEN 210 ELSE P=P+1
200 RESET(P1,S1)
210 NEXT K
220 END
```

Az előző feladat szubrutin része most a programban található, a 70-es sortól a 200-as sorig. Itt azonban a pont mozgatás határai a képernyő szélei, és ha e határok valamelyikéhez eljutottunk, akkor a pont mindaddig ott áll, amíg más irányt nem választunk (véletlenszerűen) neki.
Ezután folytatja a bolyongást, míg el nem éri az előírt lépésszámot (N).

Ha a képernyőre gyorsabban akarunk rajzolni, akkor PRINT utasítással elhelyezhetünk grafikus karaktereket.

Feladat: Írjuk ki a képernyőre a grafikus szimbólumok kódjait és a képeit! A kiírás előtt töröljük a képernyőt!

Megoldás:

```
10  CLS : PRINT CHR$(23); : REM A KIJELZÉSI FORM.=32 KAR.
20  FOR I=129 TO 191 : REM A GRAF. KARAKTEREK KÓDJAI
30  PRINT I;CHR$(I);"  ";
40  NEXTI
```

Megjegyzés:

A képernyő kijelzési formátumának leírását lásd a D. FÜGGELÉK-ben.

Feladat: Téglalap rajzolás grafikus szimbólumokkal! A megoldáshoz szükség van az előző feladat megoldására, vagy a C. FÜGGELÉK táblázatára!

Megoldás:

```
 10  CLS
 20  FOR I=0 TO 30
 30  PRINT@I,CHR$(176) : REM FELSŐ VÍZSZINTES
 40  NEXT I
 50  FOR I=1 TO 10
 60  PRINT@64*I,CHR$(149) : REM BAL OLDALI FÜGG.
 70  PRINT@64*I+30,CHR$(170) : REM JOBB OLDALI FÜGG.
 80  NEXT I
 90  FOR I=0 TO 30
100  PRINT@64*11+I,CHR$(131) : REM ALSÓ VÍZSZINTES
110  NEXT I : STOP
```

– Külön ciklusba írtuk a két vízszintes sor rajzolását. Próbálják ki a programot más megoldással is.

Gépi szintű író, olvasó utasítások

A HT–1080Z típusú számítógépek BASIC nyelvében lehetőség van a benne levő Z80 mikroprocesszor elemi input–output utasításainak használatára. Ezek lehetővé teszik, hogy a számítógépet ne csak számolásra, hanem mérési, vezérlési funkciók ellátására is felhasználjuk. Ha a számítógépünkhöz bármilyen berendezést, mérő-, beavatkozó műszert szeretnénk kapcsolni, akkor ezekkel valamilyen kommunikációra van szükség. Egy hőmérő, egy elektromotor kezelésére egészen más típusú utasítások kellenének, mint a BASIC nyelv hagyományos utasításai, de ilyenek elkészítésére a sokféle lehetőség miatt nincs mód. Mivel ez a kapcsolat mindenképpen valamilyen információátadásra épül, ezért a gépi szintű utasítások használatával mi magunk építhetjük fel ezt a kommunikációt.
Ezekkel az utasításokkal 1 byte (8 bit) átvitelére van lehetőségünk és mindig meg kell mondani, hogy melyik berendezésről van szó (0–255 közötti szám).
A bemeneti utasítás alakja:

INP (berendezés cím)

Példa: 10 X%=INP(N%)

amely az N% bemenetről beolvas 1 byte-ot és az X% egész változóba teszi. (Az INP egy függvény!)

A kimeneti utasítás alakja:

sorszám OUT berendezés cím, egész kifejezés

Példa: 10 OUT N%,X%

amely az N% kimenetre kiírja az X% egész típusú változót. A változó 0 és 255 közötti szám lehet. A kimeneten ez az érték mindaddig megmarad, amíg más értéket nem írunk oda (vagy a RESET billentyűvel nem töröljük).

A berendezés számára az egyes biteknek külön értelme lehet, ugyanazt a berendezést lehet használni egyszerre bemenetként és kimenetként is (pl. kazettás magnó: 1-es magnó 255-ös berendezés, a 2-es magnó pedig a 254-es berendezés).

A HT–1080Z számítógépet használhatjuk külső eszközök vezérlésére is, ekkor az OUT utasítást alkalmazzuk.

Feladat: Közlekedési lámpa vezérlése.
A közlekedési lámpát használjuk 2. című berendezésnek, s egy byte kiírásával vezéreljük! Ha a 0. bitje = 1, akkor kapcsoljuk be a PIROS lámpát, ha az 1. bitje = 1, akkor a SÁRGÁT és ha a 2. bit = 1, akkor a ZÖLDET!
Amelyik bit 0, azt a lámpát oltsuk le!
A PIROS és a ZÖLD lámpa 5 másodpercig fog égni, a PIROS-SÁRGA és a SÁRGA pedig 1 másodpercig.

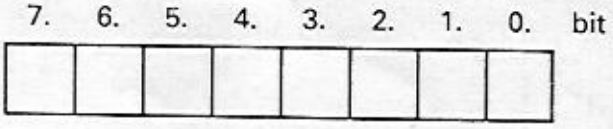
Megoldás:

```
100 OUT 2,1 : REM PIROS LÁMPA
110 FOR X=1 TO 2000 : NEXT X
120 OUT 2,3 : REM PIROS-SÁRGA
130 FOR X=1 TO 400 : NEXT X
140 OUT 2,4 : REM ZÖLD LÁMPA
150 FOR X=1 TO 2000 : NEXT X
160 OUT 2,2 : REM SÁRGA LÁMPA
170 FOR X=1 TO 400 : NEXT X
180 GOTO 100
```

Bitkezelés

Mivel minden bitnek más jelentése lehet, ezért szükségünk van az egyes bitek kezelésére is. Erre a korábban említett logikai műveletek alkalmasak (AND, OR, NOT), amelyek egész számok esetében bitenkénti műveletet eredményeznek (valós típusú változók esetén a műveletet ezek egészre konvertált értékeivel végzi el):

1 byte felosztása:



kifejezés AND kifejezés	– az eredmények azon bitjei lesznek 1-esek, amelyek mindkét kifejezésben azok voltak.
kifejezés OR kifejezés	– az eredménynek azon bitjei lesznek 1-esek, amelyek legalább az egyik kifejezésben azok voltak.
NOT kifejezés	– az eredménynek azon bitjei lesznek 1-esek, amelyek a kifejezésben 0-k voltak.

Példák:

– A 0. bit kiolvasása V%=(X%AND 1%)
– A 3. bit kiolvasása V%=(X%AND 8%)

Megjegyzés:

V%=8% vagy V%=0% lesz.

– A 2.–4. bitek kiolvasása: V%=INT((X%AND 28%)/4%)

Megjegyzés:

28 = 16 + 8 + 4. A néggyel osztás miatt V% értéke a [0,7] zárt intervallumban lesz.

– A 2. bit 1-re állítása X%=(X%OR 4%)
– A 2. bit 0-ra állítása X%=(X%AND 251%)

Megjegyzés:

251 = 255 – 4, ahol 255 a csupa 1-est tartalmazó byte.

Az OUT utasítás használatára látunk még példát a 16. FEJEZET-ben, amelyben a hang és zene generálással foglalkozunk.

Fizikai tárcímzés

Bizonyos esetekben szükség lehet a számítógép memóriájának közvetlen címzésére (általában a gép speciális lehetőségeinek kihasználásakor, pl. H. FÜGGELÉK). Ehhez a BASIC nyelv két alapvető lehetőséget biztosít. Az első: egy byte elhelyezése a memóriában. Alakja:

sorszám	POKE cím, kifejezés

Az utasítás hatása: a 'cím' címre elhelyezi a kifejezés értékét, ami (az OUT utasításhoz hasonlóan) csak 0 és 255 közti szám lehet.

Példa:

Egy egész típusú változó (A%) tartalmának elhelyezése adott címtől (C%) kezdve.

```
10 POKE C%,A%-INT(A%/256)*256: REM ALACSONYABB HELYIÉRTÉKŰ BYTE
20 POKE C%+1,INT(A%/256): REM A MAGASABB HELYIÉRTÉKŰ BYTE
```

Megjegyzés:

Egész szám két byte-on ábrázolható. Részletesen lásd F. FÜGGELÉK. A cím felső határa: 65535!!

Egy byte kiolvasása a memória adott részéből a következő függvénnyel történhet:

PEEK (cím)

A PEEK függvény segítségével a memória egy byte-jában tárolt adatot kaphatjuk meg.

Példa:

Az előző példában a C% címre elhelyezett egész szám kiolvasása A% változóba.

```
10 A%=PEEK(C%)+PEEK(C%+1)*256
```

A változó memóriabeli helyének lekérdezése:

```
VARPTR (változó)
```

A függvény kiszámítja az argumentumaként megadott változó memóriabeli címét. Ettől a címtől kezdve helyezkedik el a változó értéke a memóriában. A különböző típusú változók más-más hosszal tárolódnak (ld. F. FÜGGELÉK).

A memóriában levő szabad terület (byte-ok) lekérdezése:

```
MEM
```

A függvény megadja a memóriában fel nem használt területet byte-okban.
A fenti utasítások használatára szükség lehet néhány speciális lehetőség kihasználásához. A HT–1080Z esetében ilyen a képernyő mögötti memória közvetlen kezelése (így olyan funkciókat is használhatunk, amelyet egyébként a BASIC értelmező nem engedélyez, a cursor helyének lekérdezése stb.).

Feladat:

Írjuk a képernyőre a számítógép karaktereit a kódok ismeretében (ld. C. FÜGGELÉK)!
A karaktereket a képernyő mögötti memóriába írjuk ki a képernyő első sorától kezdve!
Ügyeljünk arra, hogy a kiírt karakterek a program befejezésekor ne törlődjenek!

Megoldás:

```
10 CLS : REM KÉPERNYŐ TÖRLÉSE
20 C%=15360 : REM KÉPERNYŐ MÖGÖTTI MEM. KEZDETE
30 FOR I=0 TO 255
40 POKE C%,I
50 C%=C%+1
60 NEXT I
70 PRINT@4*64,
80 STOP
```

Megjegyzés:

A 70-es sor PRINT utasítása a cursort a képernyő 4. sorába állítja. Erre azért van szükség, mert a ciklus végrehajtása után a képernyő 0.–3. sorában lesznek a memóriába közvetlenül beírt karakterek.

Gépi nyelvű szubrutinok

Ha egy programot a BASICnyelvben csak bonyolultan, vagy egyáltalán nem lehetne megírni, akkor szükség lehet gépi nyelvű szubrutin írására.
A szubrutin elkészítésének módjára itt nem térünk ki, mivel ennek bemutatása nem feladata e könyvnek. Itt csupán megemlítjük, hogy egy BASIC program hogyan képes aktivizálni egy ilyen szubrutint, amelyet HT–1080Z esetén Z80-as gépi kódban kell megírni.
A legtöbb BASIC implementációban lehetőség van egy adott fizikai címen kezdődő gépi kódú szubrutin végrehajtására, a HT–1080Z BASIC-ben:

```
USR (változó)
```

Mint látható, egy függvény szolgál a szubrutinhívásra. A szubrutin kezdőcímét a függvény hívása előtt POKE utasítással a 16526-os és a 16527-es címre kell írnunk. A kezdőcím alacsonyabb helyiértékű részét a 16526-os címre kell beírni. Ha a függvény argumentumát át akarjuk adni a szubrutinnak, akkor a gépi nyelvű szubrutin elején egy CALL 2687 Z80 utasítást kell végrehajtanunk. Ekkor a változó értéke a HL regiszterpárba kerül.

A szubrutinból kétféleképpen térhetünk vissza a BASIC-be:
- a RET Z80 utasítással, ekkor érték visszaadás nélkül,
- a JP 2714 Z80 utasítással, ekkor a HL regiszter tartalmát kapja értékül a változó.

Megjegyzés:

●●● Egyes implementációkban a fenti típusú függvény paramétere lehet a gépi szubrutin címe is (pl. ABC80). ●●●

Feladat:

Adjunk át egy pozitív egész számot a gépi nyelvű szubrutinnak! A szubrutin kezdőcíme 32000 legyen! A szubrutin eredményül a szám felének egész részét adja!

Megoldás:

```
10  INPUT "A SZÁM";A%
20  DATA 11,32000 : REM SZUBRUTIN HOSSZ,KEZDŐCÍM
30  READ N%,C%
40  REM A SZUBRUTIN KEZDŐCÍME
50  POKE 16526,C%-INT(C%/256)*256
60  POKE 16527,INT(C%/256)
70  REM A GEPI KÓDÚ PROGRAM!
80  DATA 205,127,10,175,203,28,203,29,195,154,10
90  FOR I%=1 TO N%: REM PR. BEÍRÁSA POKE-KAL!
100 READ Z% : POKE C%,Z% : C%=C%+1
110 NEXT I%
120 REM GÉPI KÓDÚ PR. HÍVÁSA
130 PRINT "EREDMÉNY:";USR(A%)
140 STOP
```

Megjegyzés:

A gépi kódú program utasításai decimális számként a DATA utasításban vannak megadva (80-as sor).
A gépi kódú programhoz szabad területet tudunk biztosítani a READY? után beírt számmal!

Példa:

Szeretnénk a fent megírt gépi kódú programunknak szabad területet biztosítani. A gép bekapcsolásakor a READY? után a 32000 beírásával tudjuk elérni, hogy a BASIC programunk csak eddig érjen.

Speciális funkciók (SYSTEM)

A HT–1080Z személyi számítógép rendelkezik úgynevezett speciális funkciókkal, többek között a gépi kódú monitorral. Mielőtt ezeket a funkciókat ismertetnénk, ismerkedjünk meg a funkciót kiváltó paranccsal:

```
SYSTEM
```

A parancsot a NEWLINE billentyűvel zárjuk le. A SYSTEM parancs a számítógépet monitor üzemmódba állítja. A parancs hatása a következő:

Egy üres sor kiírása után a sor elején megjelenik egy csillag és egy kérdőjel (*?), és az alábbi lehetőségek közül választhatunk:

1. gépi kódú program betöltése kazettáról;
2. gépi kódú program végrehajtása;
3. gépi kódú monitor aktivizálása;
4. BASIC bővítés aktivizálása.

Nézzük sorban a felsorolt lehetőségeket.

1. Gépi kódú program betöltése kazettáról

A *? után a gépi kódú program nevét kell megadnunk, amit a kazettáról szeretnénk betölteni. A név maximum hat karakterből állhat. Természetesen a betöltéshez a magnót lejátszásra kell állítani. A betöltés a CLOAD parancshoz hasonlóan megy végbe. Hibajelzés esetén (a képernyő jobb felső sarkában megjelenő két csillag közül a bal oldali helyére egy C betű kerül) meg kell ismételni a betöltést. Sikeres betöltés után ismét a *? jelenik meg a képernyőn.

2. Gépi kódú program végrehajtása

A gépi kódú program végrehajtása kétféleképpen lehetséges:

- a monitor aktivizálása után a G paranccsal;
- megadjuk a gépi kódú program indítási címét decimálisan egy "per" jel (/) után;

Példa: *? /32005

- egy / jelet írunk, majd a NEWLINE billentyűvel lezárjuk. Ekkor a betöltött program egy meghatározott helyéről (ez az indítási cím) kezdődik a végrehajtás.

Megjegyzés:

Ezt a lehetőséget csak közvetlenül a betöltés után használhatjuk!

3. Gépi kódú monitor aktivizálása

A / jel után írt 12710 decimális számmal tudjuk aktivizálni a gépi kódú monitort. Ez törli a képernyőt, majd kiírja a regiszterek tartalmát – ALAPÁLLAPOT –. A gépi kódú monitor lehetővé teszi Z80 gépi kódú programok bevitelét, megjelenítését, módosítását stb.

A gépi kódú monitor parancsai:

- B: visszatérés a BASIC-be;
- Dnnnn: memória tartalom megjelenítése a képernyőn hexadecimális formában. Az nnnn egy hexadecimális címet jelent. Letörli a képernyőt, ettől a címtől kezdve kiírja 16 byte tartalmát (memória rekesz). A megjelenítést vezérelhetjük lefelé mutató nyillal, ekkor az nnnn címtől kezdve a rákövetkező magasabb című memória tartalmakat írja ki a képernyőre. Felfelé mutató nyíl esetében pedig a kisebb című memória tartalmakat listázza. Más billentyű megnyomására abbahagyja a megjelenítést és visszatér az alapállapotba.
- Mnnnn: A memória tartalom módosítása a megadott címtől (nnnn). A parancs törli a képernyőt, majd az első sorban megjelenik a módosítandó memória cím és mellette a tartalma hexadecimális formában. Ezután tudjuk a tartalmat módosítani egy hexadecimális számpárral. A módosítás után megjelenik a következő cím és mellette a tartalma. A módosítás végét az X billentyű leütésével jelezzük.

– R: Regiszterek módosítása. Törlődik a képernyő, majd megjelenik az IY regiszterpár tartalma hexadecimális formában. A megjelenés után egy négyjegyű hexadecimális számmal megváltoztathatjuk a regiszterpár tartalmát. Az első regiszterpár után sorban megjelennek a következő regiszterpárok. Az X billentyű lenyomásával az adott regiszterpár módosítását átugorhatjuk. A módosítás befejezése az utolsó regiszterpár – PC – módosítása után történik.

– Gnnnn: A parancs hatására az adott címtől (nnnn) kezdődik a program végrehajtása. A parancsban megadhatunk egy tttt töréspontot is (Gnnnn, tttt), ekkor a töréspont elérésekor visszatér a monitorba. A töréspont elhelyezésekor a tttt címre generál egy CALL 3347H Z80 utasítást. Ez valamilyen hiba előfordulásakor ott marad tttt címen. (Visszaállítás M paranccsal történhet, RESET után!)

Megjegyzés:

A parancsok után nem kell a NEWLINE billentyűt használni!

4. BASIC bővítés aktivizálása

A *? után perjellel megadott számmal aktivizáljuk a bővítést. A következő lehetőségek közül válogathatunk:

Cím	Tevékenység
12288	Újrasorszámozás (RE), villogó cursor, kisbetű használata, ismétlési funkció
12299	Újraszámozás, kisbetű használata, ismétlési funkció
12294	Kisbetűk használata

– Villogó cursor letiltása, illetve engedélyezése: a SHIFT és BREAK billentyű egyidejű lenyomásával.

A HT–1080Z beépített hanggenerátora lehetővé teszi jó minőségű zenei hanghatások létrehozását. Ezek a hangok elsősorban figyelemfelkeltésre, a grafikai hatások fokozására alkalmasak.
A hang előállításában résztvevő egységek:

- hanggenerátor,
- zajgenerátor,
- keverő,
- amplitúdó-vezérlő,
- burkológörbe-generátor,
- digitális-analóg átalakítók.

A hanggenerátor három csatornás. A csatornákat jelöljük A, B, C-vel! A három csatorna tiszta hangokat állít elő. Mindhárom csatorna külön programozható.
A zajgenerátor mint neve is mutatja zajokat, zörejeket állít elő. A keverő a hanggenerátor csatornáinak és a zajgenerátornak a kimenő jeleit összegzi.
Az amplitúdó-vezérlő szolgáltatja a kimenő jel burkológörbéjét. Ez lehet állandó vagy változó amplitúdójú.
A burkológörbe-generátor állítja elő a változó burkológörbe mintákat.
A digitális-analóg átalakítók szolgáltatják a hanggenerátor kimenő jelét.

A hanggenerátort 14 darab regiszter segítségével programozzuk.
A továbbiakban a regiszterekre egységesen mint a hanggenerátor regisztereire hivatkozunk, függetlenül attól, hogy melyik szerkezeti egységben vannak. A hanggenerátor regiszterei 8 bitesek. Ebből következik, hogy értékük 0 és 255 között lehet, de nem minden regiszternél van értelmezve a teljes intervallum.

A hanggenerátor egyes regisztereinek funkciója:

- R0 A csatorna hangmagasság finom szabályozása,
- R1 A csatorna hangmagasság durva szabályozása,
- R2 B csatorna hangmagasság finom szabályozása,
- R3 B csatorna hangmagasság durva szabályozása,
- R4 C csatorna hangmagasság finom szabályozása,
- R5 C csatorna hangmagasság durva szabályozása,
- R6 a zajgenerátor frekvenciájának beállítása,
- R7 a keverő vezérlése és a csatornák engedélyezése,
- R8 A csatorna hangerő szabályozása,
- R9 B csatorna hangerő szabályozása,
- R10 C csatorna hangerő szabályozása,
- R11 burkológörbe-generátor periódusidő finom szabályozása,
- R12 burkológörbe-generátor periódusidő durva szabályozása,
- R13 burkológörbe hullámforma kiválasztása,
- R14 és R15 ebben az egységben vannak, de funkciójuk más.

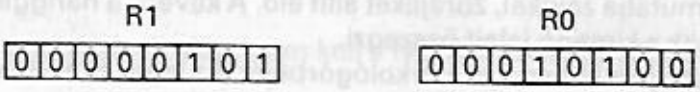
Az egyes regiszterekbe

OUT 31%, regiszter-sorszám: OUT 30%, tartalom

utasítás párokkal írhatunk.

R0–R5 A hangmagasság beállítása:
Egy-egy csatorna hangmagasságát 2–2 regiszterrel szabályozhatjuk. Egy-egy teljes 8 bites regiszterrel (R0,R2,R4) és egy-egy 8 bites regiszter (R1,R3,R5) 4 bitjével. Az összesen 0...4095, azaz 4096 különböző hangmagasság beállítását teszi lehetővé. Rajzon szemléltetve ez az A csatornán így néz ki:

Az x-el jelölt bitek értéke a hangmagasság beállítása szempontjából közömbös. Ha például 1300-as hangmagasságot állítunk be az A csatornán, a regiszterek így lesznek feltöltve:

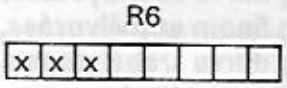


Képlettel kifejezve, ha egy H Hertz-es magasságú hangot akarunk kiadni valamelyik csatornán, akkor a megfelelő regiszterekbe

P=FIX(109166/H)

értéket kell betölteni. Ha ez az érték 256-nál kisebb, akkor a durva szabályozás regiszterébe 0-t, a finomszabályozás regiszterébe P-t kell írni. Egyébként a finomszabályozás regiszterébe (P AND 255%)-t a durvaszabályozásába (CINT(P/256) AND 15%)-ot.
A regiszterek elnevezése – finom, ill. durva szabályozás – onnan ered, hogy a durva szabályozás értékét kicsit megváltoztatva a hangmagasság nagyot változik, ezzel ugyanis – mint az ábrákból látható – a hangmagasságot megadó számérték magasabb helyiértékeit változtatjuk. A regiszterek nagyobb értékeihez mélyebb hangok tartoznak.

R6 A zajgenerátor frekvenciájának beállítása:
a regiszterbe 0 és 31 közötti értékek írhatók.



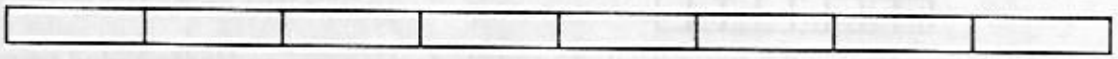
Az x-el jelölt bitek értéke közömbös.

R7 Keverő vezérlés és bemeneti/kimeneti csatorna engedélyezés:
az egyes hang csatornák egyéb paramétereit hiába állítjuk be, míg a csatorna működését nem "engedélyezzük". Erre való az R7 regiszter.

R7

Digitális be/kimeneti vezérlés
=0 : bemenet, =1 : kimenet
II. csat I. csat

Hanggenerátor engedélyezés
(=0 engedélyezve)
C csat B csat A csat

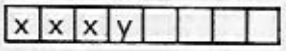


C csat B csat A csat
Zajgenerátor engedélyezés
(=0 engedélyezve)

A hanggenerátor adhat hangot akkor is, ha egyik csatorna sincs engedélyezve, de nem állandó hangerősségű hangot állítunk be – R8,R9 vagy R10-ben az y-al jelölt bit 1-re van állítva – és periódusidő valamint hullámforma be lett állítva.

R8–R10 Hangerőszabályozás: ezek a regiszterek szabályozzák az egyes csatornák hangerejét. Például az A csatornánál:

R8



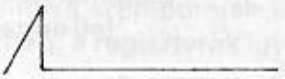
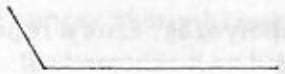
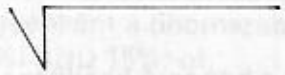
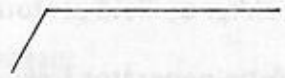
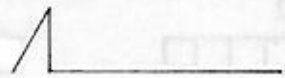
Az x-el jelölt bitek értéke a hangerő beállítása szempontjából közömbös. Ha az y-nal jelölt bit értéke 1 akkor a hangerőt szabályozó bitek értéke közömbös. Ebben az esetben ugyanis a kimenő jel amplitudóját a burkológörbe generátor szabja meg.
A hangerő értéke 0 és 15 között lehet. A nagyobb érték nagyobb hangerőnek felel meg. Ha a hangerőt 0-ra állítjuk be, akkor az illető csatorna nem ad ki tiszta hangot.

R11–R12 A burkológörbe-generátor frekvenciájának szabályozása: ha az R8, R9, R10 regiszterek valamelyikében az y-nal jelölt bit 1, akkor van értelme a burkológörbe frekvenciáját szabályozni. A szabályozás az R11,R12 regiszterekkel történik.

R12 R11

R12 a frekvencia durva szabályozója, R11 a frekvencia finom szabályozója.

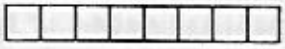
R13 A burkológörbe hullámformájának kiválasztása:
ha az R8, R9, R10 regiszterek valamelyikében az y-nal jelölt bit 1, akkor van értelme a burkológörbe alakját megadni. Egyébként a hang állandó amplitúdójú lesz.

R13	Hullámforma
x x x x	
0 0 x x	
0 1 x x	
1 0 0 0	
1 0 0 1	
1 0 1 0	
1 0 1 1	
1 1 0 0	
1 1 0 1	
1 1 1 0	
1 1 1 1	

Az x-el jelölt bitek értéke a burkológörbe hullámformája szempontjából közömbösek.

Most néhány példával segítséget szeretnénk nyújtani a hanggenerátor programozásához.

Az előzőekben gyakran jelöltük a hanggenerátor egy-egy regiszterét így:



Most az egyes helyiértékekhez – az egyes bitekhez – beírjuk, hogy milyen számértéket képviselnek.

128	64	32	16	8	4	2	1

Ennek segítségével így állíthatjuk be a hanggenerátor egy-egy regiszterét: – engedélyezzük az A hangcsatorna működését, tiltsuk a többi hangcsatorna és a zajgenerátor működését! Az R7 regiszternél közölt ábra szerint ehhez a regiszter összes bitjét 1-re kell állítani, kivéve a legutolsót. A regiszterbe tehát a 2+4+8+16+32+64+128=254 értéket kell beírni,

- az A csatornát állítsuk be úgy, hogy ne állandó amplitúdójú hangot adjon. Az ábra szerint ehhez az y-nal jelölt bitet kell 1-re állítani. A lap elején mutatott számértékek segítségével megállapítható, hogy az y-nal jelölt bit értéke 16. Az R8 regiszterbe tehát 16-ot kell írni,
- a burkológörbe alakjául válasszuk az R13 regiszternél bemutatott jelformák közül a 4.-et. Ehhez a jobb szélső és a jobbról számított 4. bitet kell 1-re állítani. Leolvasható, hogy az R13 regiszterbe 1+8=9 értéket kell beírni.

A következő BASIC program figyelemfelhívó hangot ad.

```
10 OUT 31,0:  OUT 30,120: REM A CSATORNA HANGMAGASSÁG BEÁLLÍTÁSA
20 OUT 31,7:  OUT 30,62:  REM CSAK A HANGGENERÁTOR A CSATORNA ENGED.
30 OUT 31,8:  OUT 30,16:  REM A CSATORNÁN NEM ÁLLANDÓ HANGERŐ BEÁLLÍTÁSA
40 OUT 31,12: OUT 30,16:  REM BURKOLÓGÖRBE FREKVENCIÁJÁNAK BEÁLLÍTÁSA
50 OUT 31,13: OUT 30,9:   REM BURKOLÓGÖRBE HULLÁMFORMA MEGVÁLASZTÁSA
```

A fenti program egy hangimpulzust ad. Ha az 50-es sorban R13 regiszterbe 9 helyett 8-at írunk, akkor periódikusan ismétlődő hangot hallunk.

Példa: Az alábbi példaprogrammal zenei hangok állíthatók elő.

```
 10  CLS: CLEAR 1000
 20  OUT 31%,7%:OUT 30%,254%:OUT 31%,8%:OUT 30%,15%
 30  D%=8%
 40  DATA C,208,D,185,E,165,F,155,G,139,A,124,H,110,c,104,"  ",0
 50  S$=STRING$(64%,CHR$(191%)):PRINT@64%*8%,S$;:
     PRINT@64%*12%,S$;
 60  FOR I%=0%TO 63%STEP 7%
 70  FOR J%=9%TO 11%
 80  PRINT@64%*J%+I%,CHR$(191%);
 90  NEXT J%
100  NEXT I%
110  RESTORE
120  FOR I%=0%TO D%
130  READ T$,T%
140  PRINT@64%*10%+7%*I%+4%,T$;
150  NEXT I%
160  W$="  ":T$=W$:I%=D%
170  PRINT@64%*10%+7%*I%+4%,W$;
180  IF W$="  " THEN W$=T$ ELSE W$="  "
190  H$=INKEY$:IF H$="" THEN 170 ELSE PRINT@64%*10%+7%*I%+4%,T$;
200  RESTORE
210  FOR I%=OTO D%
220  READ T$,T%:IF H$=T$ THEN OUT 31,0:OUT 30,T%:GOTO 170
230  NEXT I%
240  PRINT@64%*15%,CHR$(30%);"ISMERETLEN HANG ÜSSÖN LE EGY
     BILLENTYŰT!";:OUT 31%,0%:OUT 30%,0%:IF INKEY$=""
     THEN 240 ELSE PRINT@64%*15%,CHR$(30%);:GOTO 190
```

A 20-as sor engedélyezi az "A" hangcsatorna működését, és a hangerejét 15-re állítja be.
A 40-es sor a hangokhoz tartozó betűket és a megfelelő hangmagasságokat adja meg.
A 60...100-as sorszámú sorok kirajzolják a "billentyűket".
A 120–150-es sorszámú sorok beolvassák a hangokat jelölő betűket, és a "billentyűkre" ráírják.
A 170-es sorszámú sor írja ki a leütött hangot jelölő betűt.
A 180-as sorszámú sor váltakozva szóközt ill. a megfelelő betűt teszi a kiírandó hang nevébe. Ez okozza, hogy a leütött hangot jelölő betű villog.
A 190-es sor olvassa be a hangot jelölő betűt. Mindaddig, míg nem adnak meg újabb hangot, a vezérlés a villogtatásra adódik vissza. Ha leütnek egy betűt, akkor az eddig kiadott hang betűjét a megfelelő helyre kiírja, nehogy a villogtatás miatt a szóköz maradjon ott.
A 210...230-as sorok megkeresik a leütött jelet az ismert hangokat jelölő betűk között. Ha megtalálja, ki is adja a hangot.
A 240-es sor hibajelzést ad, ha a leütött billentyű nem jelöl ismert hangot.

Hibakezelés:

Amikor a program futása során olyan esemény következik be, amely veszélyezteti vagy megakadályozza a program további végrehajtását, (gondoljunk szintaktikus hibára, vagy tömb "túlindexelésre": olyan elemre való hivatkozás, amilyen a deklaráció szerint már nincs, vagy aritmetikai túl-, ill. alulcsordulásra: a kifejezés eredménye túl nagy, túl kicsi, a gép már nem tudja ábrázolni – megjegyzendő, hogy gépünk csak a túl nagy számnál jelez hibát, a túl kicsi számból 0 lesz –, vagy kazettahibára: például olvasás közben szalaggyűrődés miatt egy nem értelmezhető jelet talál stb.), a gép védekezik: hibajelzést ad, és "önkényesen" megállítja a programot. Ezt a BASIC interpreterbe beépített hibakezelő eljárás hajtja végre. Hiba esetén erre a rutinra adódik a vezérlés, ez kiírja az észlelt hiba kétbetűs kódját, helyrehozza az "elrontott" dolgokat, és megállítja a programot.
Lehetőségünk van, hogy saját programunkkal átvegyük ennek a hibarutinnak a szerepét, azaz ha az interpreter hibát észlel nem saját hibakezelő eljárását aktivizálja, hanem kezünkbe adja a döntést: intézkedjünk mi kedvünk szerint. Így, miután gondoskodtunk a hiba kijavításáról, programunk nyugodtan futhat tovább. Jól használható lehetőséget jelent ez "biztonságos" – a felhasználó tudatlanságából, véletlen elírásaiból fakadó hibákat kivédő – program készítésénél.
Azt, hogy az előforduló hibáról programunkban saját hibakezelő eljárással próbálunk intézkedni, egy kapcsoló beállításával jelezhetjük, erre szolgál az ONERRORGOTO utasítás:

```
sorszám ONNERRORGOTO sorszám1
```

Példa:

```
140  ONERRORGOTO  200
```

Amikor a program ezt az utasítást végrehajtja, a saját hibakezelés bekapcsolódik, s hiba esetén a vezérlés a sorszám1-gyel kezdődő programrészre adódik, ahol elvégezhetjük a hiba feldolgozását. Ehhez két függvény is segítséget nyújt. Az ERR

függvény értékéből megtudhatjuk a hiba okát, kiszámíthatjuk a bekövetkezett hiba kódszámát. Minden kétbetűs hibaüzenethez tartozik egy hiba kódszám, melyet a B. FÜGGELÉK-ben találhatunk meg.

Bekövetkezett hiba kódja: ERR/2+1

Példa: 210 IF ERR/2+1 < > 14 THEN 100

Annak a sornak a sorszámát pedig, amelyben a hiba bekövetkezett az ERL függvény adja meg:

Annak a sornak a sorszáma, ahol a hiba bekövetkezett: ERL

Példa: 300 IF ERL =10 THEN 350

A hiba kijavítása után a megfelelő helyen folytathatjuk a programot a

sorszám RESUME helymegadás

utasítással. ONERRORGOTO végrehajtása után csak RESUME utasítással lehet a vezérlést a programnak visszaadni és fordítva is igaz, RESUME utasítás csak végrehajtott ONERRORGOTO után adható ki. A RESUME utasításnak többféle alakja is van:

– "helymegadás" elmaradhat: ekkor annál az utasításnál folytatódik a végrehajtás, ahol a hiba bekövetkezett, ismét végrehajtásra kerül a hibát okozó utasítás.

Példa: 100 RESUME

– ha a "helymegadás" a NEXT szócska: a végrehajtás a hibát okozó utasítást követő utasítással folytatódik.

Példa: 100 RESUME NEXT

– a "helymegadás" lehet egy sorszám, ekkor a megadott sorszámú sornál folytatódik a végrehajtás.

Példa: 100 RESUME 15

Ha a kritikus részen túljutott a program, minden esetben tanácsos kikapcsolni a saját hibakezelést, ha ezt nem tesszük meg, a program más részén – egészen más okból – bekövetkezett hiba esetén is (nem biztos, hogy mindenre felkészíthetjük a programot) az általunk más hiba kezelésére írt programrészlet hajtódik végre, értelmetlenül. Sőt, az ONERRORGOTO érvényben marad a futás után is! A saját hibakezelő eljárás érvénytelenítését, a hibakezelés visszaadását a BASIC interpreternek az ONERRORGOTO 0 utasítással tehetjük meg:

```
sorszám ONERRORGOTO 0
```

Egy programban több saját hibarutint is alkalmazhatunk, s ONERRERGOTO-val szabályozhatjuk, hogy mindig az aktuális legyen érvényben.

Figyelem! A program átsorszámozása (RE parancs) nem változtatja meg az ERL-lel relációban lévő ill. a RESUME utasításban szereplő utasítás sorszámot, ezek változatlanok maradnak, kézzel kell hozzáigazítani az új sorszámokhoz!

A hibakezeléshez készített programrészletünk teszteléséhez nyújt segítséget az ERROR utasítás:

```
sorszám ERROR hibakód
```

Az utasítás végrehajtásakor a számítógép ugyanúgy viselkedik, mintha az adott kódú hiba bekövetkezett volna.

Példa: 30 ERROR 14
végrehajtásakor a képernyőn a "?OS ERROR IN 30" üzenet jelenik meg, feltéve, hogy nincs érvényben ONERRORGOTO utasítás.

Példa: legyen a feladatunk két egész szám beolvasása és a hányadosuk kiszámítása, kiírása. Használjuk ehhez az ONERRORGOTO utasítást! Milyen hibákat figyeljünk? Beolvasásnál a túlcsordulást, osztásnál a nullával való osztást. A hibakezelő programrész a hiba bekövetkezésének sorszáma alapján dönti el, hogy a beolvasás, vagy az osztás elvégzésekor történt-e a hiba. Ha sikeresen futott a program, hatástalanítjuk az érvényben lévő ONERRORGOTO-t.

```
100  ONERRORGOTO 5000
110  INPUT "OSZTANDÓ=";N1%
120  INPUT "OSZTÓ=";N2%
130  L%=N1%/N2%
140  ONERRORGOTO 0
150  PRINT "HÁNYADOS=";L%

. . .

5000 REM HIBÁK KEZELÉSE
5010 IF ERL =130 THEN 5060
5020 REM TÚLCSORDULÁS
5030 PRINT "KISEBB SZÁMOT ADJON MEG!"
5040 RESUME
5060 REM NULLÁVAL VALÓ OSZTÁS
5070 PRINT  "NULLÁVAL NEM LEHET OSZTANI!":
     PRINT  "ADJA MEG ÚJRA AZ ADATOKAT!"
5080 RESUME 110

. . .
```

Példa: tegyük fel, hogy egy hosszú vektort ciklikusan ismétlődő számsorozattal kell feltöltenünk. Megadjuk a számsorozat egy ciklusának számait, előre nem tudjuk, hogy hány számból fog állni, sőt az egyes alkalmazásokkor változhat mind a ciklus hossza, mind a sorozat, így könnyen változtatható programot kell írnunk. A megoldáshoz a számokat egy DATA utasításban helyezzük el, s ezt a DATA-t többször beolvasva töltsük fel a vektort! Hogy ne kelljen előre leszámolni a számsorozat tagjainak a számát, a feladatot ONERRORGOTO-val, az OD hiba figyelésével oldjuk meg. Ha nem ez a hiba következett be, kiíratjuk a sorszámot, a hiba kódszámát és megállunk, előtte hatástalanítva az ONERRORGOTO utasítást!

```
. . .
100  REM A CIKLUS SZÁMAI
110  DATA 5,8,65, . . . , 3
120  ONERRORGOTO 600
130  FOR I=1 TO N
140  READ A(I)
150  NEXT I
160  ONERRORGOTO 0
. . .
600  IF ERR/2+1 <  > 4 THEN 620
610  RESTORE : RESUME
620  PRINT "VÁRATLAN HIBA A ";ERL;". SORBAN!"
630  PRINT "A HIBA KÓDJA:";ERR/2+1
640  RESUME 650
650  ONERRORGOTO 0 : STOP
. . .
```

Hibakeresés

Eddigiekben áttekintettük, hogyan védekezhet egy jó program az esetleg hibásan megadott adatok következtében előforduló futási hibák ellen. Most tekintsük át, milyen segédeszközök állnak rendelkezésünkre, ha a megírt programunk nem úgy működik, ahogy szeretnénk, és tanácstalanok vagyunk: nem tudjuk hogy mit csinál, merre "jár" futás közben a program, nem tudjuk, hogy hol követtük el a hibát, azaz miért rossz a program. A HT–1080Z-nek több lehetősége is van, amelyet jól használhatunk ebben az esetben. Az egyik a BREAK billentyű és a CONT parancs használata. A BREAK gomb lenyomásával tetszőleges pillanatban felfüggeszthetjük a program futását. Amikor leütjük, a képernyő "megmerevedik", a program végrehajtása felfüggesztődik, megjelenik egy üzenet, hogy melyik sornál szakítottuk meg a futást. Ilyenkor tetszőleges változó tartalmát kiíratathatjuk a PRINT parancs segítségével, sőt értékadással meg is változtathatjuk azt! Ha tájékozódtunk a kritikus változók értékéről és szeretnénk, hogy a program tovább fusson a CONT parancsot kell begépelni, ennek hatására programunk ott folytatja működését, ahol a BREAK leütésekor abbahagyta. Ha megtaláljuk a hibás részletet, lehetőségünk van, hogy csak programunk kritikus részét futtassuk a "RUN sorszám" parancs segítségével. Ilyenkor a program a megadott sorszámtól kezd el futni; természetesen a programban gondoskodni kell, hogy a fontos változók megfelelő értéket kapjanak. Hasonlóan használható a "GOTO sorszám" parancs. Előnye, ugyanakkor veszélye is, hogy a változók értékei nem nullázódnak, mint a RUN parancs hatására, így a fontos változókat értékadásokkal beállíthatjuk, de számítani kell rá, hogy más változók esetleg már értékekkel rendelkeznek.
Végső esetben hasznos lehet a HT–1080Z nyomkövetési lehetősége, a TRACE üzemmód, amely a következő segítséget nyújtja: a futó program minden végrehajtott utasításának sorszáma kikerül a képernyőre, " < " " > " jelek közé írva, így leolvashatjuk, hogy melyik programokat hajtotta végre a BASIC értelmező. Szépséghibája, hogy a program is a képernyőt használja és a nyomlista is ugyanabban az időben kerül a képernyőre, s így meglehetősen áttekinthetetlenné, kuszává válhat a kép, ugyanis a TRACE a végrehajtott utasítások sorszámait

mindig oda kezdi el írni, ahol éppen a fénypont (cursor) van, így elrontva esetleg a grafikus képet, a program által szervezett kommunikációt. Ha közben a programból a képernyőt töröljük: a nyomlista is törlődik, vagy a fénypontot mozgatjuk: a nyomlista sorai egymásra kerülhetnek stb. Így ezt a lehetőséget csak korlátozott mértékben, végső eszközként érdemes használni.
A nyomkövetés bekapcsolása, a TRACE üzemmód beállítása egy TRON utasítás végrehajtásával, vagy a TRON parancs kiadásával érhető el:

```
        sorszám      TRON
```

A TRACE üzemmód mindaddig érvényben marad, amíg ki nem kapcsoljuk a gépet, vagy egy új program nem kerül a gépbe: kazettáról betöltve, vagy a NEW paranccsal az előzőt kitörölve, vagy meg nem szüntetjük a TROFF paranccsal:

```
        sorszám      TROFF
```

Ezzel lehetőségünk van arra, hogy ne az egész programot nyomkövessük, hanem csak a kritikus programrészt: a figyelni kívánt programrész elé TRON, a legvégére pedig egy TROFF utasítást elhelyezve.

Nézzünk egy példát a TRACE használatára! Tekintsük az alábbi kis programot:
– a program "n" pozitív szám összegét számolja ki. Elsőként beolvassuk n értékét, definiálunk egy n +1 hosszúságú (az indexek 0-tól n-ig mehetnek) vektort, majd rendre beolvassuk a számokat a vektor 1,2, . . . n indexű elemeibe.

```
 10  INPUT "N=";N
 20  DIM A(N)
 30  C=0
 40  FOR I=1 TO N
 50  PRINT I;". SZÁM="; : INPUT A(I)
 60  IF A(I) <= 0 THEN 50
 70  C=C+A(I)
 80  NEXT I
 90  PRINT "SZUMMA=";C
100  STOP
```

Ha lefuttatjuk, a képernyőn a következő párbeszéd jelenik meg:

```
RUN
N=? 3
1 . SZÁM=? 2
2 . SZÁM=? -4
2 . SZÁM=? 3
3 . SZÁM=? 1
SZUMMA=6
BREAK IN 100
READY
>
```

Ugyanez nyomkövetéssel:

1-es sorszámú sorként írjuk be a nyomkövetést bekapcsoló TRON utasítást:

```
1 TRON
```

Most futtassuk újra a programot!

```
RUN
<10>N=? 3
<20><30><40><50> 1 . SZÁM=? 2
<60><70><80><50> 2 . SZÁM=? -4
<60><50>2 . SZÁM=? 3
<60><70><80><50> 3 . SZÁM=? 1
<60><70><80><90>SZUMMA=6
<100>
BREAK IN 100
READY
>
```

IRODALOMJEGYZÉK

1., Lőcs–Sarkadi Nagy–Szlankó: A BASIC programozási nyelv.
Műszaki Könyvkiadó, 1976.

2., Némethy Katalin: Elemi ismeretek a BASIC nyelvről.
KFKI, 1980.

3., TEASYS – Kisszámítógépes oktató rendszer.
KFKI, 1977–81.

4., HT–1080Z BASIC kézikönyv.
Híradástechnikai Szövetkezet, 1983.

5., ABC80 BASIC példatár, I–VII.
ELTE TTK Numerikus és Gépi Matematikai Tanszék, 1982–84.

6., ABC–BASIC eljárások.
ELTE TTK Numerikus és Gépi Matematikai Tanszék, 1983.

7., A programozás ABC-je = az ABC' programozása.
ELTE TTK Numerikus és Gépi Matematikai Tanszék, 1984.

8., Dusza Árpád: A BASIC programozási nyelv. (Jegyzet)
OPI kiadvány, 1981.

9., Némethy Katalin–Veszprémi Anna–Zsakó László: Az ABC80 BASIC elemei.
FPI kiadvány, 1981.

10., Dr. Fercsik János: Az ABC80 BASIC nyelve. Kézirat.
Dunaújvárosi Műszaki Főiskola, 1983.

11., D. Alcock: Ismerd meg a BASIC nyelvet!
Műszaki Könyvkiadó, 1983.

12., Dr. Kocsis András: BASIC programozási kézikönyv.
SZÁMALK, 1983.

13., Kőhegyi János (szerk.): Ismerd meg a BASIC nyelvjárásait.
Műszaki Könyvkiadó, 1984.

14., Dr. Appel György: Módszertani segédanyag a számítástechnikai tanártovábbképzési tanfolyamokhoz 1983.
A Fővárosi Pedagógiai Intézet Számítástechnikai sorozata I.

15., ELTE szerzői munkaközössége: Kézikönyv a személyi számítógépes tanárképzési tanfolyamhoz (Szerk.: dr. Appel György)
A Fővárosi Pedagógiai Intézet Számítástechnikai sorozata II. 1983.

A HT–1080Z BASIC utasításai
F betűvel jelöljük a függvényeket.

Név		Leírás	Fejezet
ABS	F	Abszolút érték függvény	3
ASC	F	Egy karakter ASCII kódja	8
ATN	F	Arkusz tangens függvény	3
AUTO		Automatikus sorszámkiírás	11
CDBL	F	Konverzió dupla pontosságú valós típusra	3
CHR$	F	ASCII kód (szám) karakteres formára hozása	8
CINT	F	Konverzió egész típusra	3
CLEAR		A definiált változók nullázása, vagy helyfoglalás szöveg kezeléshez	8,11
CLOAD		A program betöltése kazettáról	11
CLOAD?		A kazettára kiírt program ellenőrzése	11
CLS		Képernyőtörlés	6
CONT		Megszakított programfutás folytatása	11
COS	F	Koszinusz függvény	3
CSAVE		A program kiírása kazettára	11
CSNG	F	Konverzió egyszeres pontosságú valós típusra	3
DATA		Adatokat tárol a programban (ld. még READ utasítás)	7
DEFDBL		Típus deklaráció (dupla pontosságú valós típus)	3
DEFINT		Típus deklaráció (egész típus)	3
DEFSNG		Típus deklaráció (egyszeres pontosságú valós típus)	3
DEFSTR		Típus deklaráció (szöveges típus)	3
DELETE		Sorok törlése	11
DIM		Változók helyfoglalásának (tömbméret) meghatározása	5
EDIT		Egy utasítás javítása	11,12
END		A program futásának befejezése	4

Név		Leírás	Fejezet
ERR	F	Az utoljára bekövetkezett hiba kódja	17
ERL	F	Az utoljára bekövetkezett hiba sorának sorszáma	17
ERROR		Hiba előidézés	17
EXP	F	Exponenciális függvény	3
FIX	F	Valós szám csonkítása egészre	3
FOR.TO.STEP		Cikluskezdő utasítás	5
FRE	F	Szabad memória szövegek részére	8
GOSUB		Szubrutinhívás	9
GOTO		Feltétlen vezérlésátadás	4
IF.THEN.ELSE		Elágazás	4
INKEY$	F	Egy karakter beolvasása	6
INP	F	Gépi szintű input függvény	15
INPUT		Beolvasó utasítás	6,13
INT	F	Egészrész függvény	3
LEFT$	F	Szöveg baloldalának leválasztása	8
LEN	F	Szöveg hossza	8
LET		Értékadó utasítás	3
LIST		A program kiírása képernyőre	11
LOG	F	Természetes alapú logaritmus	3
MID$	F	Szöveg részének kiválasztása	8
MEM	F	Szabad memória mérete	15
NEW		A program törlése a memóriából	11
NEXT		Ciklusvég utasítás	5
ON . . GOSUB		Számított szubrutinhívás	9
ON . . GOTO		Számított vezérlésátadás	4
ONERRORGOTO		Hibafigyelés	17
OUT		Gépi szintű output utasítás	15,16
PEEK	F	Olvasás fizikai memóriacímről	15
POINT	F	Meghatározza, hogy a képernyő adott pontja világít-e	14
POS	F	A cursor soron belüli pozíciója	6
POKE		Írás fizikai memóriacímre	15
PRINT		Kiíró utasítás	6,13
PRINT USING		Kiírás adott formátumban	6,13
PRINT@		Kiírás pozícionálással	6
RANDOM		Véletlenszám generátor kezdőérték beállítás	10
READ		Olvasás DATA utasításból	7
REM		Megjegyzés elhelyezése a programban	4
RE		A program átsorszámozása	11
RESET		A képernyő egy pontjának kioltása	14
RESTORE		Az első DATA utasításra 'áll'	7
RESUME		Visszatérés hibarutinból	17
RESUME NEXT		Visszatérés hibarutinból	17
RETURN		Visszatérés szubrutinból	9

Név		Leírás	Fejezet
RIGHT$	F	Szöveg jobboldalának leválasztása	8
RND	F	Véletlenszám előállítása	10
RUN		Program elindítása	11
SET		Egy képernyőpont kivilágítása	14
SGN	F	Előjel függvény	3
SIN	F	Színusz függvény	3
SQR	F	Négyzetgyök függvénye	3
STOP		A program futásának befejezése	4
STR$	F	Konverzió szám típusból szöveg típusba	8
STRING$	F	Azonos betűkből álló szöveg előállítása	8
SYSTEM		Monitor üzemmódba váltás	11,15
TAB	F	Kiírási pozíció meghatározása (ld. PRINT utasítás)	6
TAN	F	Tangens függvény	3
TRON		Nyomkövetés bekapcsolása	11,17
TROFF		Nyomkövetés megszüntetése	11,17
USR	F	Gépi nyelvű szubrutin hívása	15
VAL	F	Karakteres változó átalakítása numerikussá	8
VARPTR	F	Változó kezdőcíme	15

A HT–1080Z BASIC alapkiépítésének hibajelzései

Kód	Rövidítés	Jelentés
1	NF	NEXT utasítást akar végrehajtani a neki megfelelő FOR utasítás nélkül.
2	SN	Szintaktikus hiba.
3	RG	RETURN utasítást akar végrehajtani megelőző GOSUB utasítás nélkül.
4	OD	READ utasítást akar végrehajtani, miután a megfelelő DATA utasításból már beolvasta az összes adatot, vagy INPUT ■, utasítást és és a kazettán nem található több adat.
5	FC	Hibás argumentumú utasítás (pl. negatív dimenzió, 0 vagy negatív szám logaritmusa stb.).
6	OV	Túlcsordulás: a művelet eredménye nem ábrázolható.
7	OM	Nincs több tárolóhely (a tömbök túl nagyok, vagy a program túlságosan hosszú).
8	UL	Nem létező sorszámú sorra hivatkozás.
9	BS	Indextúllépés: a tömb olyan indexű elemére hivatkozunk, amely kívül esik a DIM utasításban megadott tartományon.
10	DD	A program már megadott dimenziójú tömb méretét akarta DIM utasítással újradefiniálni.
11	/0	Nullával osztás.
12	ID	Az utasítást parancs üzemmódban nem lehet használni.
13	TM	Típuskeveredés: az utasításban hibás típusú változót, függvényt használtunk.
14	OS	Nincs több hely szövegek tárolására (ld. még CLEAR utasítás!).
15	LS	Szöveg típusú változóba 255-nél több karaktert próbált elhelyezni.
16	ST	Ilyen bonyolult szöveg kifejezést nem tud kiértékelni.
17	CN	Nem végrehajtható a CONT utasítás (a program END utasítással állt meg, vagy javítottunk a program valamelyik sorában).
18	NR	Hibakezelő szubrutin végén nem a RESUME utasítást hajtottuk végre.
19	RW	RESUME utasítást akart végrehajtani megelőző hiba előfordulás nélkül.
20	UE	ERROR utasítás nem létező hibakóddal.
21	MO	Hiányzó operandus a kifejezésben.
22	FD	Hiányos file kazettán (beolvasási hiba).

A HT–1080Z BASIC karakterkészlete

Az ASCII (American Standard Code for Information Interchange) kódon alapul, de attól kissé eltér (pl. grafikus karakterek esetén).

Kód	Karakter		Kód	Karakter	
00	blank		32		szóköz
01	CTRL/A		33	!	felkiáltójel
02	CTRL/B		34	"	idézőjel
03	CTRL/C		35	■	hashmark
04	CTRL/D		36	$	dollárjel
05	CTRL/E		37	%	százalékjel
06	CTRL/F		38	&	et
07	CTRL/G		39	'	aposztrof
08	CTRL/H	1 karakterrel visszalép és töröl	40	(	nyitózárójel
09	CTRL/I		41	)	csukózárójel
10	CTRL/J	linefeed (soremelés)	42	*	csillag
11	CTRL/K	új sort kezd	43	+	plusz
12	CTRL/L	új sort kezd	44	,	vessző
13	CTRL/M	RETURN (új sor)	45	–	mínusz
14	CTRL/N		46	.	pont
15	CTRL/O		47	/	per-jel (törtvonal)
16	CTRL/P		48	0	
17	CTRL/Q		49	1	
18	CTRL/R		50	2	
19	CTRL/S		51	3	
20	CTRL/T		52	4	
21	CTRL/U		53	5	
22	CTRL/V		54	6	
23	CTRL/W	ritkított kiírás	55	7	
24	CTRL/X	visszalép 1 karakterrel	56	8	
25	CTRL/Y	előrelép 1 karakterrel	57	9	
26	CTRL/Z	1 sorral lejjebb lép	58	:	kettőspont
27	CTRL/[	1 sorral feljebb lép	59	;	pontosvessző
28	CTRL/\	64. kar./sor és HOME	60	<	kisebbjel
29	CTRL/]	curs. a sor elejére áll	61	=	egyenlőségjel
30	CTRL/↑	a cursortól jobbra eső sorrészt törli	62	>	nagyobbjel
31	CTRL/_	a cursor mögötti képernyőrészt törli	63	?	kérdőjel

Kód	Karakter		Kód	Karakter	
64	@		96	'	(SHIFT+@)
65	A		97	a	
66	B		98	b	
67	C		99	c	
68	D		100	d	
69	E		101	e	
70	F		102	f	
71	G		103	g	
72	H		104	h	
73	I		105	i	
74	J		106	j	
75	K		107	k	
76	L		108	l	
77	M		109	m	
78	N		110	n	
79	O		111	o	
80	P		112	p	
81	Q		113	q	
82	R		114	r	
83	S		115	s	
84	T		116	t	
85	U		117	u	
86	V		118	v	
87	W		119	w	
88	X		120	x	
89	Y		121	y	
90	Z		122	z	
91	[	szögletes nyitózárójel	123	{	kapcsos nyitózárójel
92	\		124	¦	
93	]	szögletes csukózárójel	125	}	kapcsos csukózárójel
94	^	kalap	126	~	
95	_	aláhúzás	127	▩	

Kód	Karakter	Kód	Karakter
128		160	
129		161	
130		162	
131		163	
132		164	
133		165	
134		166	
135		167	
136		168	
137		169	
138		170	
139		171	
140		172	
141		173	
142		174	
143		175	
144		176	
145		177	
146		178	
147		179	
148		180	
149		181	
150		182	
151		183	
152		184	
153		185	
154		186	
155		187	
156		188	
157		189	
158		190	
159		191	

Grafikus karakterek képeinek előállítása:

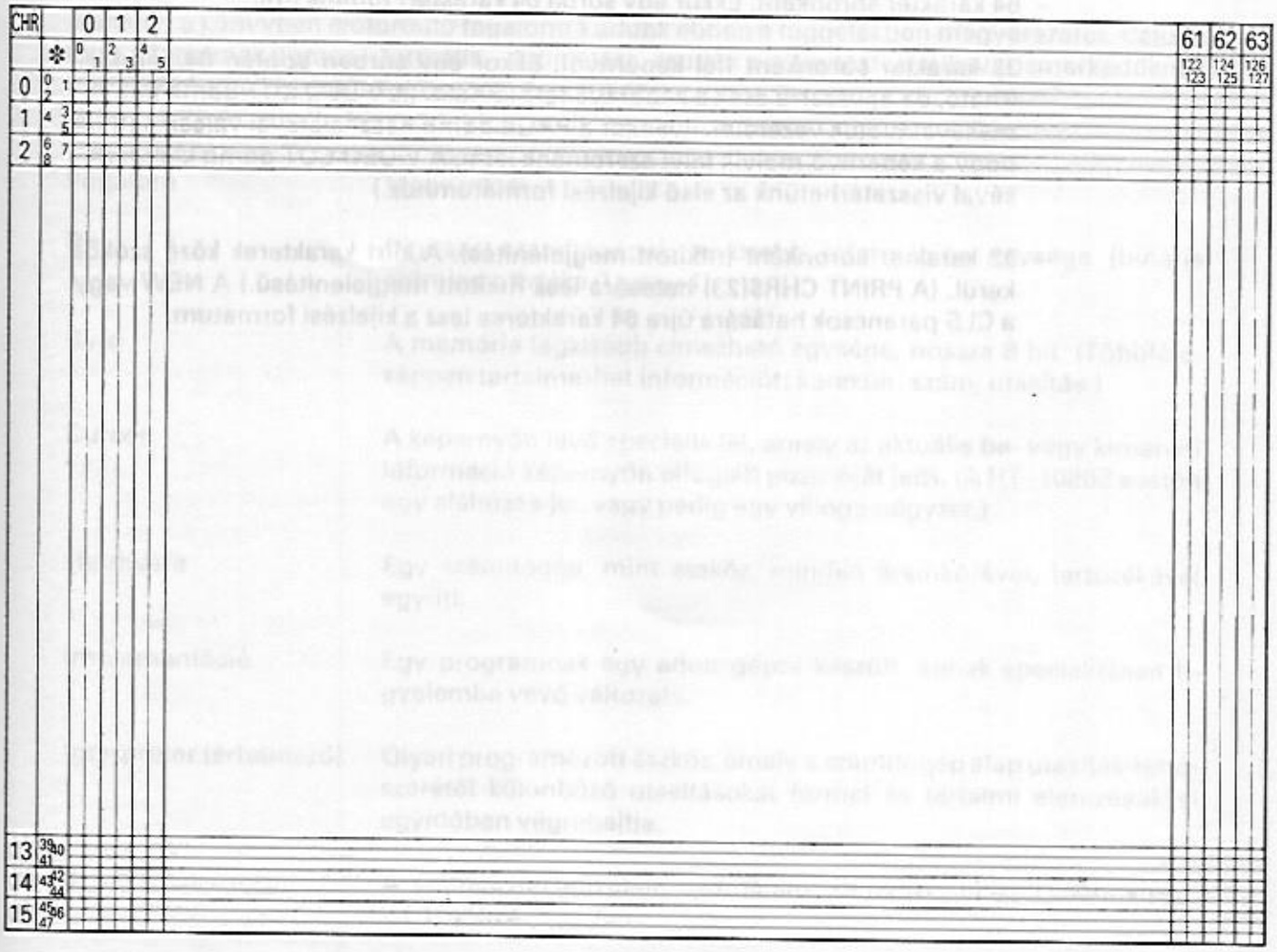
PRINT CHR$(128+A)
A képzése:

1	2
4	8
16	32

Például a "teli" karakter képzése:
128+1+2+4+8+16+32=191
PRINT CHR$(191)

192–255: nincs látható képük, de ezek kód–192 db szóköz kiírását eredményezik PRINT CHR$(kód) végrehajtásakor.

A képernyő felosztása



* grafikus felosztás

A HT–1080Z számítógépen a következő kijelzési formátumok közül választhatunk:

- 64 karakter soronként. Ekkor egy sorba 64 karaktert tudunk írni.
- 32 karakter soronként (fél képernyő). Ekkor egy sorban szintén 64 karakter írható, de egyszerre csak a képernyő fele látható. (A VIDEO CUT gomb benyomásával tudjuk vezérelni, ilyenkor a PAGE gomb használatával választhatjuk, hogy a képernyő melyik felét szeretnénk látni. A VIDEO CUT gomb kiengedésével visszatérhetünk az első kijelzési formátumhoz.)
- 32 karakter soronként (ritkított megjelenítés). A kiírt karakterek közé szóköz kerül. (A PRINT CHR$(23) hatására lesz ritkított megjelenítésű.) A NEW vagy a CLS parancsok hatására újra 64 karakteres lesz a kijelzési formátum.

Számítástechnikai fogalmak magyarázata

Néhány, a könyvben előforduló fogalomra adunk ebben a függelékben magyarázatot. Célunk nem a fogalmak pontos – formális – definiálása, csupán a számítástechnikával ismerkedőknek szeretnénk egy kis segítséget nyújtani.

Fogalom	Magyarázat
Bit	Digitális számítógépek legkisebb információs egysége (bináris számjegy, értéke 0, vagy 1).
Byte	A memória legkisebb címezhető egysége, hossza 8 bit. (Többféleképpen tartalmazhat információt: karakter, szám, utasítás.)
Cursor	A képernyőn levő speciális jel, amely az aktuális be- vagy kimeneti információ képernyőn elfoglalt pozícióját jelzi. (A HT–1080Z esetén egy aláhúzás-jel, vagy pedig egy villogó négyzet.)
Hardware	Egy számítógép, mint eszköz, minden áramkörével, tartozékával együtt.
Implementáció	Egy programnak egy adott gépre készült, annak specialitásait figyelembe vevő változata.
Interpreter (értelmező)	Olyan programozott eszköz, amely a számítógép alap utasítás-rendszerétől különböző utasításokat formai és tartalmi elemzésükkel egyidőben végrehajtja.
K	A számítástechnikában használatos dimenzió nélküli mennyiség, 2↑10=1024.
Karakter	Kódrendszerek által definiált jelhalmazok eleme. (A HT–1080Z az ASCII kódrendszert használja.)

Fogalom	Magyarázat
Lebegőpontos	Számábrázolási forma valós számokra, amely két funkcionális részből áll: mantissza és karakterisztika. A mantissza egy rögzített nagyságrendű valós szám (a HT–1080Z esetén ez a szám (m) : $0.5 \leq m < 1$); a karakterisztika ennek szorzófaktora, az alapszám (a HT–1080Z esetén 2) egész kitevőjű hatványa; a valós számot ez a szorzat reprezentálja. Előnye: ezután a valós számok nagyságrendtől függetlenül képezhetők adott számú byte-ra valamilyen pontossággal.
Mikroprocesszor	A mikroszámítógép központi utasítás értelmező – és műveletvégző áramköre egy nagy integráltságú áramköri tokban realizálva.
Mikroszámítógép	Nagy integráltságú félvezető elemekből összeállított kis méretű és energiafogyasztású, olcsó, megbízható, igénytelen számítógép kategória.
Periféria	Azon berendezések gyűjtőneve, melyekkel az ember-gép dialógus megvalósul, illetve amelyek nagy tömegű adat tárolását teszik lehetővé. Ilyenek pl.: billentyűzet, képernyő, nyomtató, mágneslemeztároló, X–Y rajzoló stb.
RAM	A számítógépek munkatárolója (Random Access Memory). Ez szolgál a felhasználói programok, azok változóinak tárolására. Nevét onnan kapta, hogy az információ elérési ideje nem függ annak memóriabeli helyétől.
ROM	Csak olvasható információtároló memória (Read Only Memory). Kisebb teljesítményű számítógépek esetén ROM memóriában van pl. a BASIC interpreter.
Software	A számítógép működtetését és felhasználását biztosító – esetleg különböző szintű – programtermékek összessége.

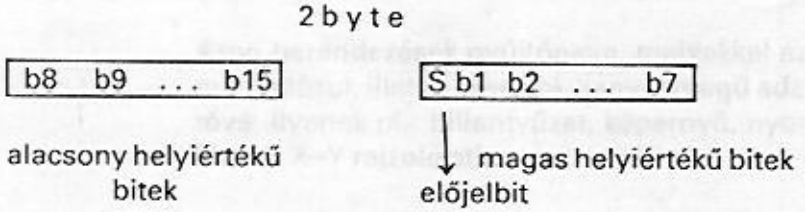
Fogalom	Magyarázat
Személyi számítógép	Olyan mikroszámítógép, amely egyszerűsége, ára és egyéb paraméterei alapján sok ember számára elérhető és birtokolható.
Szintaxis	Azon szabályok összessége, amelyek definiálják egy programozási nyelven írt program formai helyességét. (Programnyelvi helyesírási szabályzat.)

Adatábrázolás a HT–1080Z-n

I. Szám típusúak

1. Egész adatok

Ábrázolásuk 2 byte-on, kettes komplemens kódban történik. Az első byte tartalmazza az alacsonyabb helyiértékű biteket, a második a magasabb helyiértékeket (ennek első bitje az előjelbit: negatív=1, nem negatív=0).



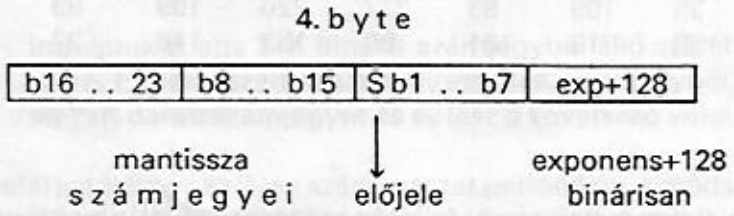
Példák az ábrázolásra:

a decimális szám	ábrázolása	
	Magas helyiértékű byte	Alacsony helyiértékű byte
	S1234567	01234567. bitje
10 =	00000000	00001010
127 =	00000000	01111111
256 =	00000001	00000000
32767 =	01111111	11111111
–1 =	11111111	11111111
–10 =	11111111	11110110
–127 =	11111111	10000001
–256 =	11111110	00000000
–32767 =	10000000	00000001
–32768 =	10000000	00000000
0 =	00000000	00000000

2. Valós adatok

A. Egyszeres pontosságú adatok

Ábrázolásuk 4 byte-on történik m * 2 exp alakban, 6 decimális jegy pontossággal, aritmetikai kerekítéssel. A bináris ábrázolású mantissza – amely az első 3 byte-ot foglalja el – abszolút értékét tárolja a gép normalizáltan úgy, hogy az első bitet nem ábrázolja. Az exponens additív kódú bináris ábrázolású (ha exp=0, az 128-ként van ábrázolva). A valós nullát a karakterisztika 0 volta jelzi, s a mantissza bitek értéke érdektelen.



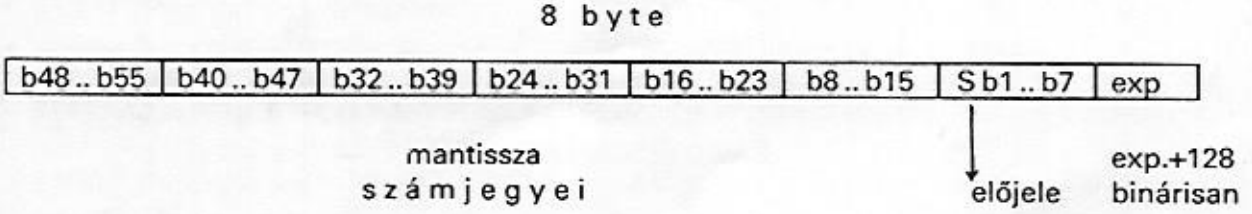
Példák az ábrázolásra:

a decimális szám	ábrázolása 1.	2.	3.	4. byte
0 =	–	–	–	0
1 =	0	0	0	129
10 =	0	0	32	132
127 =	0	0	127	135
999999 =	240	35	116	148
123456789 =	163	121	107	155
–32768 =	0	0	128	144
–0.987654321 =	234	214	252	128
–0.5 =	0	0	128	128
–0.0625 =	0	0	128	125

– jelentése = közömbös

B. Dupla pontosságú adatok

Ábrázolásuk elve azonos az egyszeres pontosságú adatokéval, csak a mantissza tárolása céljából itt 7 byte van fenntartva, ami 16 decimális jegy pontosságot garantál.

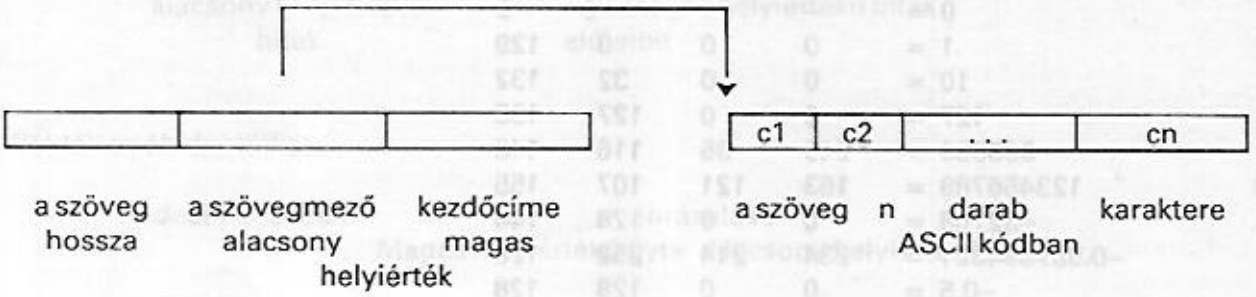


Példák az ábrázolásra:

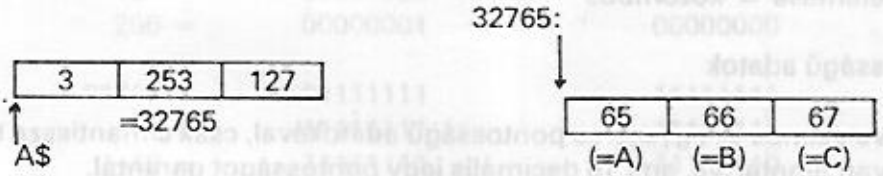
a decimális szám	ábrázolása							
	1.	2.	3.	4.	5.	6.	7.	8. byte
0 =	–	–	–	–	–	–	–	0
1 =	0	0	0	0	0	0	0	129
123456789 =	0	0	0	160	162	121	107	155
–0.987654321 =	197	181	114	224	233	214	252	128
123.456D–20 =	166	225	204	173	91	48	54	69
1D–38 =	35	199	83	237	220	199	89	2
1D+38 =	139	13	181	80	153	118	22	255

II. Szöveg típusú adatok

A szövegek ábrázolása két objektummal történik: fejjel és szövegmezővel. Egy szöveges adat feje mindig 3 byte-ból áll, a szövegmező pedig annyi byte-ot foglal, ahány jelből a szöveg áll. Tehát n db. karakter tárolásához n+3 byte helyet foglal a HT–1080Z BASIC.



Példa: 10 A$ = "ABC" esetén A ábrázolása



Véletlenszámok előállításának néhány módszere

Az egyik első véletlenszám előállítási módszert Neumann János javasolta, amelynek lényege a következő volt:

> Induljunk ki egy 2*K bináris számjegyből álló számból. Ez lesz az első véletlenszám. Ennek a számnak a négyzete 4*K számjegyből fog állni, vegyük ki a középső 2*K darab számjegyet és ez lesz a következő véletlenszám.

Könnyen belátható, hogy az ilyen számsorozat periodikus, a módszer hibája, hogy a periódushossz erősen függ a kezdőértéktől és gyakran igen rövid. A másik gond az, hogy nem teljesül a "véletlenség" követelménye sem, a kisebb számok gyakrabban fordulnak elő az így előállított sorozatban. (Ez volt a 'négyzetközép módszer'.)
Véletlenszám generátoraink egy másik módszeren alapulnak, a 'multiplikatív kongruenciamódszer'-en (ld. a számelmélettel foglalkozó irodalmat). Lényege a következő:

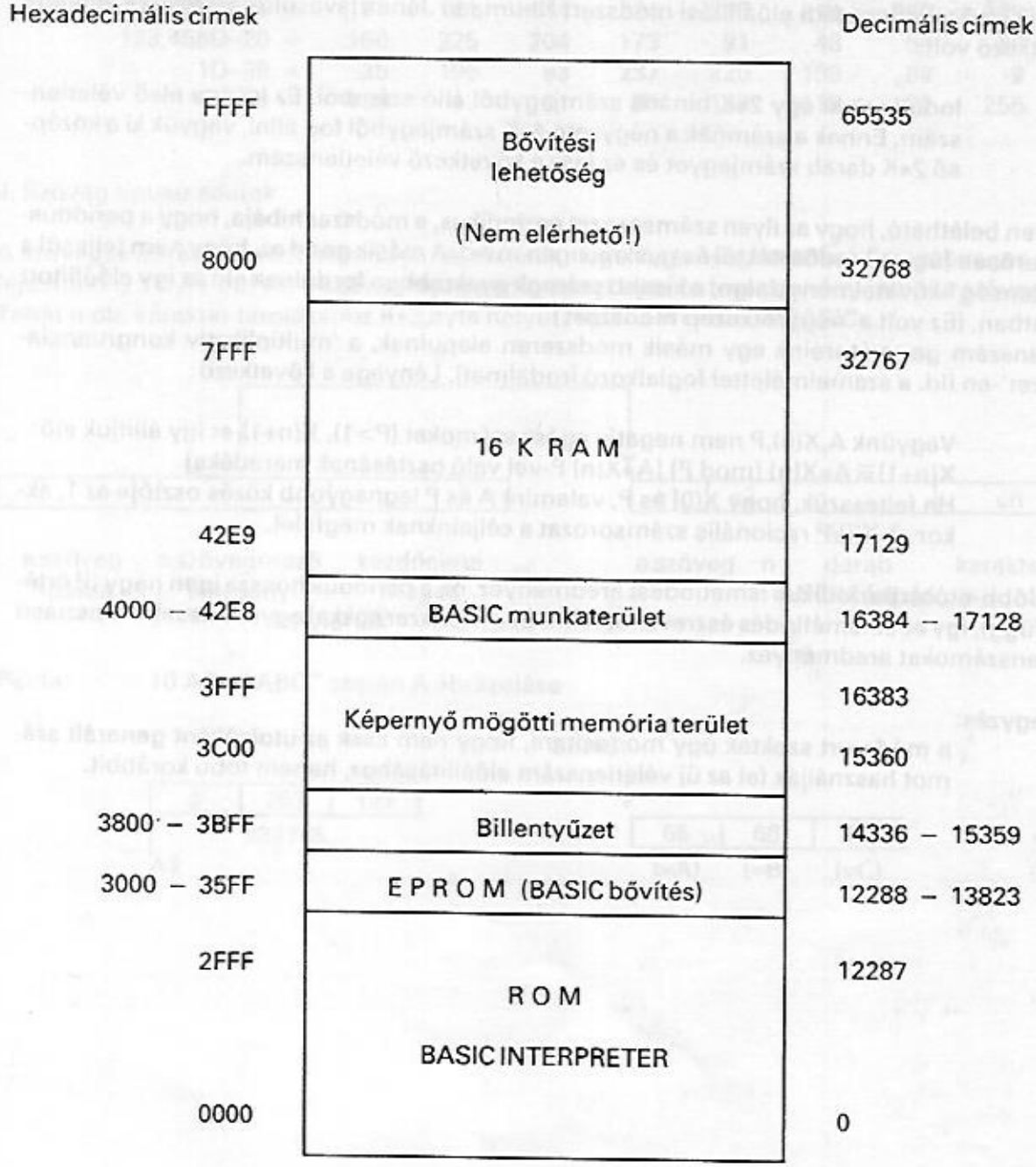
> Vegyünk A,X(0),P nem negatív egész számokat (P>1), X(n+1)-et így állítjuk elő:
> $X(n+1) \equiv A*X(n) \pmod P$ [A*X(n) P-vel való osztásának maradéka]
> Ha feltesszük, hogy X(0) és P, valamint A és P legnagyobb közös osztója az 1, akkor az X(i)/P racionális számsorozat a céljainknak megfelel.

Ez is előbb-utóbb periodikus ismétlődést eredményez, de a periódus hossza igen nagy (P értékétől függ), így ez az ismétlődés észrevehetetlen. Ez a módszer sokkal egyenletesebb eloszlású véletlenszámokat eredményez.

Megjegyzés:

> a módszert szokták úgy módosítani, hogy nem csak az utolsóként generált számot használják fel az új véletlenszám előállításához, hanem több korábbit.

A memória felosztása



ABC80 programok átírása HT–1080Z-re

- Ékezetes betűk nincsenek,
- a HT gépnek van ún. duplapontos adattípusa,
- különbözik az egyszeres pontosságú adatok ábrázolási módja; a HT gép bináris számábrázolást használ,
- a szövegek hossza a HT gépen legföljebb 255 karakter lehet,
- a HT gépen a típusjelző karakter (%,■,$,!) után nem szabad szóközt írni (nem mindig okoz hibát, de érdemes betartani),
- a HT gépen a változók azonosítója teljes hosszában megjelenik, de az első 2 karaktert használja csak a BASIC – ennek tehát egyedinek kell lennie.
  További megkötés, hogy a változó azonosító nem tartalmazhat kulcsszót,
- a HT gépen lehetőség van rá, hogy megadott betűkkel kezdődő változó azonosítók automatikusan egy adott adattípusba tartozzanak. Ilyenkor a típusjelző karaktert nem kell használni,
- a HT gépen tetszőleges dimenziójú tömbök használhatók, lehetséges indexes változók indexhatárainak dinamikus változtatása,
- a hatványozás jele a képernyőn [karakter, a billentyűn egy felnyil (↑),
- hatványozásnál a HT gép ki tud értékelni olyan kifejezést amelynél a kitevő valós és az alap negatív – ha a kitevő AKTUÁLIS ÉRTÉKE egész. Pl. A=1 : PRINT (–1)[A kiszámítható,
- egészek osztásakor a HT gép valós eredményt ad (PRINT 3%/4%),
- a HT gépen nincs nagypontosságú ASCII vagy decimális aritmetika. Helyette dupla pontosságú aritmetika van,
- a HT gépnek nincs LOG10, SPACE$, INSTR függvénye,
- a RIGHT$ függvény értelmezése más;
  RIGHT$(A$,3) az A$ utolsó 3 karakterét adja meg,
- az RND függvénynek argumentuma van;
  RND(0) 0 és 1 közé eső valós véletlenszámokat generál,
  RND(I) egy 1 és I közé eső egész véletlenszámot ad,
- a NUM ¤ függvény neve a HT gépen STR$,
- nincs PI függvény;
- a HT gépen csak AND, OR, NOT logikai művelet van,
- a HT gépnél többsoros megjegyzések is írhatók egy sorszámhoz,
- a REM kulcsszó '-nak rövidíthető,
- szöveg konstans határolójele csak az "-jel lehet,
- szöveg konstans belsejében nem használhatunk " karaktert,
- a HT gépen NINCS NULLASZOR lefutó ciklus,
- nincs duplapontos ciklus,
- a számábrázolás különbözősége miatt egy ciklus eltérő számban futhat le ABC80 ill. HT gépen,
- a HT gépen írható változó azonosító nélkül is NEXT utasítás, és egy NEXT utasításhoz több változó azonosító is írható,
- a HT gép ismeri a PRINT utasítás ún. PRINT USING alakját,

- a HT gépen nincs CUR függvény, más módon lehet a képernyő megadott pontjára írni,
- a PRINT a kiírt számok mögé is mindig tesz egy szóközt,
- a PRINT utasítást ?-el lehet rövidíteni,
- magnóra a PRINT ■–1, nyomtatási lista utasítással lehet írni,
- az INPUT utasításba beírható egy kiírandó szöveg,
- hibás adat beírását nem lehet figyelni, a gép maga ismételteti meg a hibás adatot,
- mód van arra, hogy INPUT utasításnál ne írjunk be adatot, ilyenkor a változó korábbi értéke megmarad,
- GET utasítás helyett az INKEY$ függvényt kell használni,
- a RESTORE utasításban sorszám nem használható,
- nincs ON RESTORE utasítás,
- a szövegek számára kezdetben 50 byte van fönntartva. Ha ezt változtatni akarjuk, használni kell a CLEAR n parancsot. Erre n byte lesz fönntartva a szövegek számára,
- szöveg típusú változó hossza nem deklarálható,
- DEFFN utasítás a HT gépen nincs,
- RANDOMIZE helyett RANDOM utasítás van, de használata szükségtelen, a véletlenszám generátor úgyis mindig más értékkel indul,
- a HT gépen adatfile-ok lényegében nincsenek, kazettára különálló rekordokat ír a gép. Minden egyes PRINT létrehoz egy 257 karakteres bevezető rekordot és az adatrekordot.
  Egy PRINT utasítás nem eredményezhet 247 karakternél többet.
  Egy INPUT utasítás nem olvashat 247 karakternél többet,
- a HT gépen nincs óra,
- a hanggenerálás lényegesen eltér az ABC80-étól. Programozása bonyolultabb, de az előállított hang sokkal jobb,
- a HT gép grafikája lényegesen eltér az ABC80-tól. A képernyő grafikusan 128x48-as méretű. (48 sor, 128 oszlop) A koordinátarendszer origója, a tengelyek irányítása megegyezik az ABC80-éval. A sorokat nem kell külön grafikus állapotba hozni. A koordináták megadásának sorrendje az ABC80-hoz képest fordított,
- CLS törli a képernyőt (a PRINT CHR$(12) nem törli),
- hibakezelő programrészt RESUME utasítással kell lezárni,
  RESUME sorszám a megadott sorszámra adja a vezérlést;
  RESUME vagy RESUME 0 a hibás utasításra adja a vezérlést;
  RESUME NEXT a hibását követő utasításra adja a vezérlést,
- szinte minden hiba lekezelhető (de INPUT hiba nem),
- ERROR n utasítással az n kódú hiba kiváltható, próbálható,
- az ERL függvény megadja a hibás sor számát,
- ERR/2+1 kifejezés megadja a hibakódot,
- a hibakódok teljesen eltérnek az ABC80 hibakódjaitól,
- a nyomkövetés TRON paranccsal kapcsolható be és TROFF-fal kapcsolható ki.

TARTALOMJEGYZÉK

A kiadvány a Fővárosi Pedagógiai Intézet, a Tudományszervezési és Informatikai Intézet,
valamint az INTERPRESS Kiadó és Nyomda Vállalat közös kiadásában,
az INTERPRESS Kiadó és Nyomda Vállalat, Budapest, gondozásában készült.

A kiadásért felel dr. Pollák Miklós,
az INTERPRESS Kiadó és Nyomda Vállalat, Budapest, igazgatója

Megjelent 18,5 (A/5) ív terjedelemben, helvetica betűtípusból
Eng. száma: 48 889
Készült a Szolnoki Nyomda Vállalatnál
85.73 SzNyV – Felelős vezető: Gombkötő Béla igazgató

Szeret ?
játszva töprengeni, szórakozni ?
Szereti ?
a rejtvényeket ?
Szeretné ?
próbára tenni számítástechnikai ismereteit ?

KERESSE a

Azonos játékok,
Azonos feladatok...
... különböző
személyi számítógépekre!

c. könyvet!

amely versenyfeladatokat tartalmaz
az M08X, a HT1080Z, a Commodore 64
és a Sinclair Spectrum számítógépekre!

ÁRA: 66 Ft