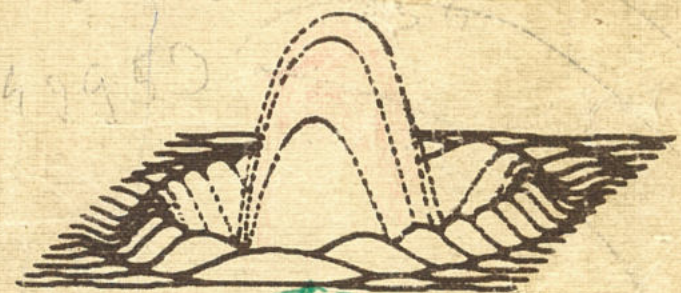
AIRCOMP-16
BASIC
Programozási
Kézikönyv



1984

65 = 7
66 = 13
67 = C
68 = D
69 = E
70 = F
71 = 6

PERSONAL

PERSONAL AGROELEKTRONIKAI
GAZDASÁGI TÁRSASÁG
2040. BUDAÖRS, MOLNÁR P. U. 1.
TELEFON: 260–612
TELEX: 22–5962

10,0
90,10
70,20
60,30

Írta:
BODOR TIBOR
és
GERŐ PÉTER

Lektorálta:
RÓNA PÉTER

BUDAPEST, 1984

ELŐSZÓ

Ez a könyv azoknak a „boldog keveseknek", de immár egyre többeknek szól, akiknek AIRCOMP-16 személyi számítógép van a birtokukban, vagy ilyenhez hozzáférhetnek. Vagyis könyvünk a tulajdonosoknak és a felhasználóknak készült.

A személyi számítógépek általában, így az ARCOMP-16 is, alapvetően kétféle módon használhatók: amatőr és profi célokra.

Az amatőr felhasználás jellemzője, hogy a gépet játékra, szórakozásra, tanulásra, hobbyra, egyszóval egyéni célok megvalósítására használjuk, nem pedig megélhetésünk biztosítására. Az amatőr felhasználás nem jelentheti szükségképpen azt, hogy felesleges, szakszerűtlen, alacsony színvonalú.

A profi felhasználást egyértelműen az jellemzi, hogy ilyenkor a gépet részben vagy teljesen létfenntartásunk biztosítására használjuk. Természetesen a profi felhasználás sem jelenti szükségképpen a szakszerűséget, a magas színvonalat, habár a körülmények általában ezt többé-kevésbé kikényszerítik.

Előfordulhat, hogy valaki a személyi számítógépét mind amatőr, mind profi célokra használja, de soha nem állhat elő, hogy egyikre sem. Ha valaki már kipróbált egy AIRCOMP-16 számítógépet, többé már nem mond le róla önként. Vagy így, vagy úgy, de használni fogja.

A profi felhasználók többsége a gépet nem önállóan, hanem második gépként, fejlesztésre, tesztelésre, tanulásra, általában egy nagyobb gép kiegészítésére használja.

Az ARCOMP-16 mindezen feladatok ellátására kiválóan alkalmas.

Ezért könyvünk egyrészt kézikönyv, amely pontosan leírja a BASIC lehetőségeit. Ilyen minőségében mind az amatőröknek, mind a profiknak szól.

Másrészt egy kicsit tankönyv is, amelyből az amatőr felhasználók megtanulhatják nemcsak a gép és a BASIC használatát, hanem a korszerű programozás elemeit és alapelveit is.

Harmadrészt pedig némi kitekintést is nyújt a profiknak a további, bővebb lehetőségekre.

A könyv számos példával illusztrálja a gép sokoldalúságát. Így bemutatunk oktató, zenélő, grafikus, adatfeldolgozó, szalagkezelő, matematikai számításokat végző, kötegelt feldolgozó, nagy gépre való szimulált programot.

Minthogy a könyv mindenkinek szól, anyagát úgy állítottuk össze, hogy az AIROMP-16 gépet pusztán a könyv segítségével az is használhassa, akinek semmilyen számítástechnikai alapismerete nincs, és az is tudjon BASIC prog-

ramokat írni, aki mindeddig a BASIC-ről, és egyáltalán a programozásról még csak nem is hallott.

Mindazonáltal több helyütt célzunk olyan lehetőségekre, illetve tárgyalunk olyan szempontokat is, amelyeknek véleményünk szerint az amatőrök szempontjából nincs nagy jelentősége. Ilyenkor azokra a profi felhasználókra gondultunk, akik más gépeken is dolgoznak, és az ő kedvükért tértünk ki olyan programozástechnikai sajátosságokra, amelyeknek a jelentősége az AIRCOMP-16 gépen sem elhanyagolható, de csak nagyobb gépeken domborodik ki igazán.

Jelen könyvünkben az AIRCOMP és a BASIC lehetőségeit nem egyforma részletességgel tárgyaljuk. Vannak, amelyeket csak éppen megemlítünk, másoknál viszont hosszabban elidőzünk. Az utóbbiak azok, amelyeket minden felhasználó szempontjából egyaránt alapvetőnek és fontosnak ítéltünk.

A könyvben bemutatott példákat kereskedelmi forgalomban lévő, tehát nem egyedi gyártású, AIRCOMP-16 gépeken próbáltuk ki. Nincs a könyvben olyan mintaprogram, amely nem futott legalább egyszer.

Aktív AIRCOMP-16 felhasználói tapasztalatainkat úgy összegezhetjük, hogy a gép mind amatőr, mind profi célokra, az utóbbi esetén elsősorban programfejlesztésre, tesztelésre kiválóan alkalmas. Igen kellemes tulajdonsága, amely személyi számítógép voltából fakad, hogy mindig kéznél van, a nap 24 órájában rendelkezésre áll, az átfordulási idő néhány perc vagy másodperc, és az üzemköltsége gyakorlatilag néhány fillér.

Meggyőződésünk, hogy az AIRCOMP-16 minden tulajdonosnak nemcsak attraktív szobadísze, hanem kedvenc munkaeszköze is lesz.

Ehhez kíván segítséget nyújtani ez a könyv.

BEVEZETÉS A KÖNYV HASZNÁLATÁBA

Mindenekelőtt erősen javasoljuk, hogy mindazok, akik először használnak személyi számítógépet, vagy még soha nem programoztak BASIC-ben, sőt még azok is, akiknek van ilyen tapasztalatuk, de először ülnek le az AIRCOMP-16 gép mellé, elejétől végéig olvassák el ezt a könyvet, próbálgassák ki a bemutatott lehetőségeket, mielőtt a gépet üzemszerűen használni kezdik.

Szinte minden oldalán a könyvnek található új és fontos információ, ugyanis igyekeztünk a gépre és a BASIC-re vonatkozó ismereteket olyan sorrendben közölni, amilyenben azokra a gépet nem ismerő amatőr felhasználónak várhatóan szüksége lesz. Ettől az elvtől csak szükség esetén tértünk el.

A könyv információtartalma három kategóriába sorolható.

Az elsőben az AIRCOMP-16 kezeléséhez szükséges ismereteket adtuk meg, vagyis mindazt, amit a gép üzemeltetéséhez tudni kell.

A gépen használható BASIC nyelv leírása képezi a második kategóriát. Ebben találhatók meg az utasítások kódolásának szintaktikus szabályai – ezeket minden programozónak figyelembe kell vennie. Ezen túlmenően, mindenütt, ahol csak szükségesnek tartottuk, az utasítások használatát példákkal illusztráltuk, működési módjukat pedig magyarázatokkal tettük világosabbá. Ezekre természetesen elsősorban az amatőr felhasználóknak van szüksége, de a nyelvet jól ismerők sem hagyhatják teljesen figyelmen kívül, mert egyrészt az egyes BASIC gépi megvalósítások között igen jelentős eltérések is lehetnek, másrészt a magyarázataink programtervezési kérdésekre is kiterjednek.

Külön kategóriát képeznek azok a részek, ahol az AIRCOMP-16 és a rajta használható BASIC nyelv hatékonyságát, sokoldalúságát, esetleges felhasználási területeit mutatjuk be, vagy speciális lehetőségeire térünk ki. Még ennek a kategóriának is csak elenyésző része az, ami csak a profi felhasználóknak szól. Például az ide tartozó mintaprogramok mindegyikét meg lehet érteni csupán a könyv korábbi fejezeteinek ismeretében.

A gyors információkeresést a könyvben a szokásosnál részletesebb tartalomjegyzék támogatja.

Ezen kívül a külön táblázatba kigyűjtött BASIC kulcsszavak mindegyikéhez megadjuk annak az oldalnak a számát, amelyen a szóbanforgó kulcsszót tartalmazó utasítást részletesen tárgyaltuk.

AZ AIRCOMP–16 SZEMÉLYI SZÁMÍTÓGÉP TECHNIKAI ADATAI

Mikroprocesszor: Z80A
Operatív memória: 16115 bájt RAM
Szoftver: 8 Kbájt ROM BASIC interpreter
Perifériák:

Monitor: OIRT normál TV készülék
Háttértároló: kazettás magnetofon
Billentyűzet: 53 standard kiosztású multifunkciós fólia tasztatúra
Hálózat: 220 V +10% –15% 50 Hz 20 VA
Érintésvédelem: kettős szigetelés (II. oszt.)

Fejlesztő:

HOMELAB
HARD-SOFT
Elektronikai GMK
1156 Budapest, Nádastó park 13.

Gyártó és szervíz:

PERSONAL Agroelektronikai Gazdasági Társaság
2040 Budaörs, Molnár Pál utca 1.

AZ AIRCOMP–16 ÜZEMBE HELYEZÉSE

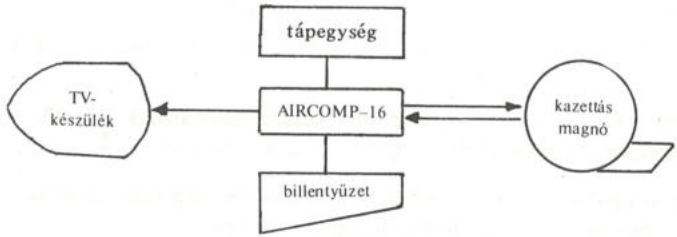
A TV ÖSSZEHANGOLÁSA A GÉPPEL

MEGJEGYZÉSEK A GÉP ÜZEMBEHELYEZÉSÉHEZ

KARBANTARTÁSI UTASITÁS

BIZTONSÁGI RENDSZABÁLYOK

Személyi számítógépünkhöz TV és kazettás magnó csatlakoztatható, amelyekből az alábbi konfiguráció építhető ki:

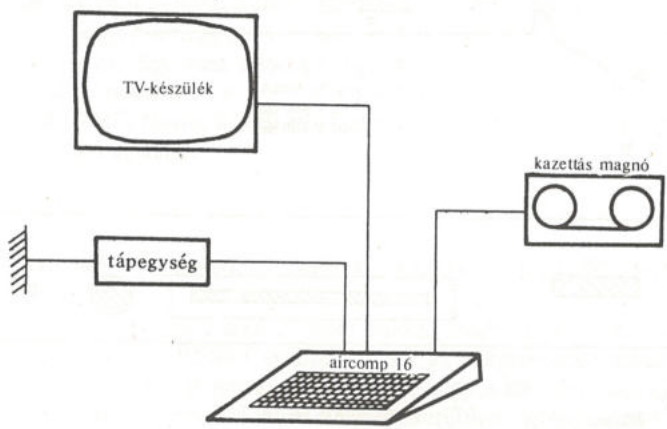


A tápegység biztosítja a hálózatból a gép üzemeléséhez szükséges energiát.

A billentyűzet parancsok, utasítások és adatok bevitelére használható. Minthogy egybe van építve magával a számítógéppel, valójában nem tekinthető külön egységnek.

A TV készülék parancsok, utasítások és adatok, valamint programok és azok eredményeinek a megjelenítésére szolgál.

A kazettás magnó programok vagy adatok tárolását teszi lehetővé. Segítségével programjainkat vagy adatainkat kazettára vihetjük fel, illetve onnan tetszés szerint beolvashatjuk.



A TV készüléket a géphez mellékelt 75 Ohm impedanciájú szabványos árnyékolt (koaxiális) kábellel kapcsolhatjuk össze a számítógéppel. Ennek egyik végét a TV antennacsatlakozójába, másik végét pedig az AIRCOMP-16 TV jelzésű csatlakozójába dugjuk. Lehetőség van az antennacsatlakozás helyett közvetlenül a TV videobemenetére csatlakozni. Ehhez megfelelően (szakemberrel!) átalakított TV szükséges.

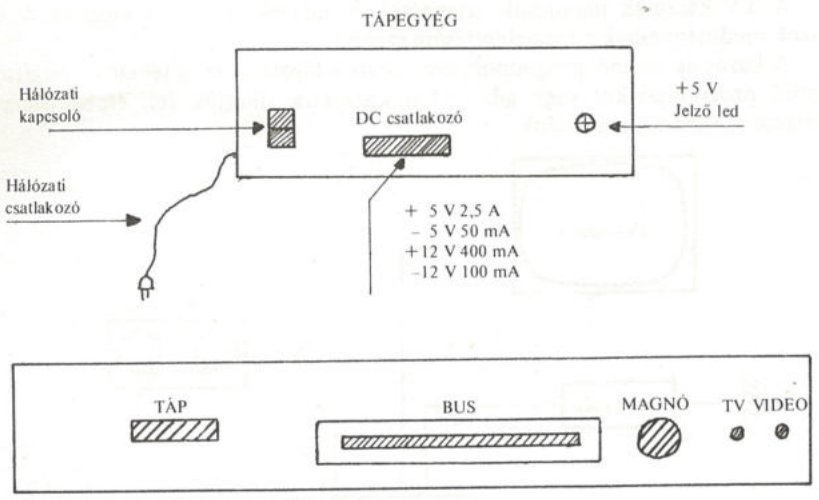
Az AIRCOMP-16 és a kazettásmegnó összekötéséhez szükséges átjátszókábel szintén a gép tartozéka. Ezt a magnón az átjátszó csatlakozóba, a gépen pedig a magnó jelzésű csatlakozóba kell bedugni.

Végül a készülékhez adott kábellel kössük össze a számítógépet a tápegységgel.

Ezzel összeállt a rendszerünk. Már csak a hálózatba kell a TV-t, a magnót és a tápegységet bekapcsolni, hogy működőképes is legyen.

Számítógépünk most már bekapcsolható, de még nem üzemképes, mert a TV-t is, és a magnót is össze kell hangolni a géppel.

A magnó és az AIRCOMP-16 összehangolását a magnóhasználattal foglalkozó fejezetben részletesen ismertetjük, így itt csak a TV behangolását írjuk le.



AIRCOMP-16

A TV ÖSSZEHANGOLÁSA A GÉPPEL

(1) Kapcsoljuk be a TV-t.

(2) Állítsuk a csatornaváltót a 6-os vagy 21-es csatornára, mert ezek valamelyikén fog „bejönni" az AIRCOMP-16.

(3) Kapcsoljuk be a gépet a tápegységen lévő hálózati kapcsolóval. Ha a tápegység bekapcsolt állapotban van, a rajta lévő jelzőlámpa világít. A rendszer bejelentkezését egy csipogó hang jelzi.

Ekkor a képernyőn az alábbi feliratoknak kell megjelenniük:

– AIRCOMP –
16115 BYTES FREE COMPUTER
HOMELAB BASIC REV.1.2
OK

Az OK üzenet alatt, a képernyő szélén egy kis négyszög alakú jel, a kurzor villog.

(4) Ha a kép nem akar bejönni, akkor a finomhangolóval addig szabályozzuk a TV készüléket, amíg a kép meg nem jelenik. Ha ezt a finomhangolóval sehogyan sem sikerül elérnünk, próbálkozzunk a durva hangolóval is.

(5) Ha a kép minősége nem megfelelő, például világos, sötét, grízes, szétesik, fut, torz, stb., a TV készülék kezelőszerveivel próbáljuk meg beállítani.

A kép kontrasztját és fényerősségét úgy szabályozzuk be, hogy szemünknek kellemes legyen, a betűk jól olvashatóan, tisztán látszódjanak. A TV hangerejét állítsuk nullára, különben monoton búgó hangot ad.

(6) Ha a képet sehogyan sem sikerül beállítanunk, ellenőrizzük a csatlakozásokat. Egyrészt nézzük meg, hogy a dugóknál az érintkezés megfelelő-e, másrészt ha a TV-nken több antennabemenet is van (például UHF, VHF, távoli, közeli stb.), akkor sorra próbáljuk ki, hogy melyiken a legjobb az átvitel.

MEGJEGYZÉSEK A GÉP ÜZEMBEHELYEZÉSÉHEZ

Felhívjuk a figyelmet arra az apró lukra, amely a gép oldalán található. Ebben van elrejtve a RESET gomb, aminek az üzembehelyezéskor nincs szerepe, így itt csak annyit jegyzünk meg róla, hogy a kis lukon bedugott gyufaszál (nem foszforos!) végével lehet a legegyszerűbben megnyomni.

A géphez elvileg bármilyen kereskedelmi forgalomban kapható TV készülék csatlakoztatható, ha az alkalmas az OIRT 1–24 csatornák vételére, és szabványos antennacsatlakozóval van felszerelve.

A kazettás magnót is szintén tetszésünk szerint választhatjuk meg, feltéve hogy képes kb. 300 mV erősségű jelek felvételére, és a felvételt legalább 500 mV erősségű kimenő jellel tudja visszaadni; valamint szabványos 5 pólusú átjátszócsatlakozóval rendelkezik.

Ha a TV antennacsatlakozója vagy a magnó átjátszócsatlakozója nem szabványos, vagy át kell alakíttatnunk, vagy megfelelő adaptert kell használnunk.

Tapasztalataink szerint a hordozható, 24–28 cm-es kisképernyővel szerelt TV készülékek, és az egyszerű hordozható magnók a legalkalmasabbak. Az ennél kisebb képernyő nehezen olvasható. Ami a kazettákat illeti, elvileg bármilyen megfelel, de a gyakorlatban az olcsó, az igényes zenei felvételekhez nem túl jó minőségű kazettatípusok váltak be. Elvileg orsós magnetofon is használható lenne, de a kazetta sokkal kényelmesebben kezelhető. Lehetőleg számlálóberendezéssel rendelkező magnót használjunk, hogy a rajta lévő felvételeket könnyebben visszakereshessük.

Akkor is hálózatról üzemeltessük a TV-t vagy a magnót, ha egyébként elemmel vagy akkumulátorról is működtethetők lennének. A hálózat biztosabb, és a géphez amúgyis szükséges.

Noha a gépre a „családi" TV-t is ráköthetjük, mégis ajánljuk, hogy vegyünk egy külön készüléket erre a célra. A TV-t ugyanis egészen másképpen kell hangolni a géphez, mint a TV műsorokhoz, így az átkapcsolás és visszakapcsolás nem oldható meg egyszerű csatornaváltással, hanem a készülék esetenként igen nehézkes oda- és visszahangolásával jár. Ugyanígy érdemes külön magnót is venni a géphez, mert annál is lehetnek áthangolási problémák, noha egyszerűbben megoldhatók, mint a TV-nél.

Mellesleg ha a TV készüléket nem használjuk műsorok vételére, akkor átalakítható úgy, hogy ne az antennabemeneten keresztül csatlakozzon a géphez. Figyelem! Ezt az átalakítást a szervízzel végeztessük el! Érdemes megtenni, ha a gépet rendszeresen és sokat használjuk, mert így stabilabb és jobb minőségű képet kapunk. A műsoros kazetták lejátszására nem használt magnókon is végeztethetünk (szakemberrel!) olyan átalakításokat, amelyektől az adatátvitel biztonságosabb, zavarmentesebb lesz.

KARBANTARTÁSI UTASÍTÁS

A gép a tisztántartáson kívül semmilyen karbantartást nem igényel.

A gépet száraz, vagy enyhén nedves puha ronggyal tisztogassuk. Ha szükséges, csak semleges mosószert használjunk!

Óvjuk a gépet az ütődésektől, és a szélsőséges hőmérsékletektől, valamint a nedvességtől. Ne tegyük ki tűző nap hatásának.

A gép rázkódásra nem érzékeny.

A klaviatúra nyomásra érzékeny, ne könyököljünk, ne tenyereljünk rá. Fokozottan óvjuk az ütődéstől. Kemény, éles tárgyak kárt okozhatnak benne.

A TV-re és a magnóra vonatkozóan tartsuk be a gyártó karbantartási utasításait.

BIZTONSÁGI RENDSZABÁLYOK

A gép kettős szigetelésű, a II. érintésvédelmi osztályba tartozik. Védőföldelt hálózatba nem köthető. Ne használjuk 240 V-nál magasabb és 190 V-nál alacsonyabb feszültségű hálózatról.

A gépet ne szedjük szét, a belsejét ne piszkáljuk. Ha ennek a kísértésnek semmiképpen nem tudunk ellenállni, legalább húzzuk ki előbb a konnektorból.

A feszültség alatt álló gépen csak a kezelőszervekhez nyúljunk, és azokat is rendeltetésszerűen használjuk.

A TV-re és a magnóra vonatkozóan a gyártó által előírt biztonsági rendszabályok a mérvadók. De ezeknél is vegyük figyelembe a gépről fentebb elmondottakat.

Fokozott veszélyforrást jelent, hogy a gép üzemszerű használatához legalább három konnektor szükséges. Ezért célszerű többlukú csatlakozó aljzattal szerelt hosszabbítót használni. Ilyenek készen kaphatók. Vigyázzunk, hogy ez úgy legyen elhelyezve, hogy gyerekek ne férhessenek hozzá. Ne használjunk úgynevezett T dugót, valamint toldott vezetékeket.

Bekapcsolásnál először mindig a hálózati dugót csatlakoztassuk a hálózathoz és csak utána kapcsoljuk be a tápegység hálózati kapcsolóját.

Kikapcsolásnál pont fordítva, először kapcsoljuk ki a hálózati kapcsolóval a gépet és csak utána húzzuk ki a hálózati csatlakozót a konnektorból. A számítógép tápegységhez történő csatlakoztatását mindig csak feszültségmentes (kikapcsolt) állapotban szabad elvégezni!

A készülék bármi nemű meghibásodása esetén forduljon a gyártóhoz ill. a gyártó által megadott szervízhez!

AZ AIRCOMP–16 BASIC HASZNÁLATA

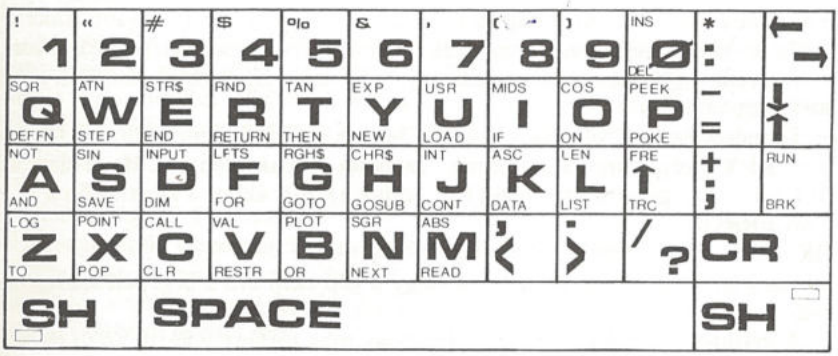
EGYSZERŰ TEVÉKENYSÉGEK

FELTÉTELES TEVÉKENYSÉGEK

ISMÉTLŐDŐ TEVÉKENYSÉGEK

Programozni nem lehet könyvből megtanulni. Az Olvasónak azt javasoljuk, hogy üljön le az előbbi fejezetek szerint üzembehelyezett géphez és minden újabb lehetőséget, amivel a programozói kézikönyv alapján megismerkedik, nyomban próbáljon is ki. Eltérés esetére az Olvasó figyelmébe ajánljuk a svéd hadsereg harcászati szabályzatának vonatkozó passzusát: amennyiben a terep nem olyan, amilyennek a térkép szerint lennie kellene, akkor a terep szerint kell eljárni.

A gép billentyűzete a következő:



A gépelni tudó felhasználó figyelmét különösen felhívjuk arra, hogy a ϕ (nulla) és az O betű, az 1-es számjegy és az L betű ezen a billentyűzeten nem ugyanaz; nagyon vigyázzunk, fel ne cseréljük a használatukat!

EGYSZERŰ TEVÉKENYSÉGEK

A gép bekapcsolása után (amint „A TV összehangolása a géppel" című fejezet leírja) a képernyőn a gép „bemutatkozása" jelenik meg, majd az OK szöveg, ami azt jelenti, hogy minden rendben van, a gép kész a parancsaink fogadására. Az OK alatti sorban egy kis fénylő négyszög villog: ez a kurzor. A kurzor jelzi, hogy hol tartunk a sorban: minden egyes jel begépelésekor egyet lép jobbfelé, a jel pedig ott lesz látható, ahol a kurzor az előbb volt. Most tehát bármilyen szöveget begépelhetünk, betűket, számjegyeket, írásjeleket, műveleti jeleket, és mindegyik sorra megjelenik a képernyőn.

Speciális jel: CR

Jelentése: vége a sornak. (Carriage return: kocsi vissza.) Ezt a jelet a jobb szélső, alulról a második gomb lenyomásával adjuk ki.

A képernyőn lévő sor akkor jut el a központi egységhez, ha a CR gombot lenyomjuk. Amíg ezt meg nem tettük, addig a képernyőn (a később ismertetendő módon) még következmények nélkül tetszőlegesen javíthatjuk a sort.

Szokták a CR gomb lenyomását úgy is nevezni, hogy ezzel „küldjük el" a sort a gépnek.

Természetesen kicsi a valószínűsége, hogy a próbaképpen, találomra lenyomott jelek a gép számára „értelmes" üzenetet fognak jelenteni. Ha például az ABCD betűket gépeltük be, majd lenyomjuk a CR-t, akkor a képernyőn a

? SN ERROR
OK

szöveg jelenik meg, ami azt jelenti, hogy a gép nem érti a begépelt szövegünket.

A kérdőjel és az ERROR szó jelzi, hogy hiba történt; a kettő közti rövidítés (jelen esetben az SN) a hiba jellegére utal. Az SN ERROR azt jelenti: a beírt szöveg, úgy, ahogy van, a gép számára értelmetlen. A következő sorban megjelenő OK pedig azt jelzi, hogy a gép ismét alapállapotba került: újra készen áll arra, hogy fogadja a következő parancsot.

PRINT 15 CR hatására a gép (a képernyő következő sorába) kiírja a 15-ös számot, s utána természetesen az OK üzenetet.

PRINT 15*3/2–8 CR vagy hasonló hatására – tehát ahol a PRINT után valamilyen matematikai kifejezést írunk – a gép (a képernyő következő sorába) kiírja a PRINT után álló teljes kifejezés kiszámolt értékét, jelen esetben azt, hogy 14.5. A gép alapállapotba kerülését ismét az OK jelzi.

Kulcsszó: PRINT
Jelentése: írd

A PRINT betűsorozatnak tehát már van értelme a gép számára.

(Ha elírnánk, például RPINT-et gépelnénk, persze megint az SN ERROR üzenet jelenne meg.) A BASIC nyelv teljes egészében ilyesfajta kulcsszavakból áll.

Mit tegyünk, ha PRINT-et akartunk írni és véletlenül az R betűt nyomtuk le? Az AIRCOMP-on is létezik az a gomb, ami az írógépeken az „egy betűnyit vissza" funkciót valósítja meg. A jobb felső sarokban két vízszintes nyíl van egy gombon; ha ezt a gombot lenyomjuk, a kurzor egyet jobbra lép anélkül, hogy bármit írna vagy törölne, ha pedig valamelyik SH-gombot is lenyomva tartjuk ennek a gombnak a lenyomása közben, akkor a kurzor balra lép. Így közlekedhetünk a sorban előre-hátra, és ha a kurzort rámozgattuk egy jelre és begépelünk egy másik jelet, akkor ezzel a régit már töröltük is és helyettesítettük az újjal. Ha a nyilakkal jelzett gombot nemcsak lenyomjuk, hanem lenyomva is tartjuk, ennek a hatása (mint a többi jel gombjainál) olyan, mintha egymás után sokszor nyomnánk le. Kis gyakorlattal ott állíthatjuk meg, ahol akarjuk, és a sor bármelyik részét felülírhatjuk – mielőtt a sort a CR gombbal elküldtük volna. (Az alatta levő, függőleges nyilakkal jelzett gombbal hasonlóképpen mozgathatjuk a kurzort fel és le is.)

A lehetséges műveletek: + – * / ↑
A jelentésük: összeadás, kivonás, szorzás, osztás, hatványozás.
Vigyázat, a hatványozást jelentő nyíl a FRE és TRC feliratú gombon található, nem azonos a kurzor fel-le gombbal. A hatványozás jele a képernyőn ^ alakban jelenik meg.

Például	2↑1ϕ	jelentése	$2^{1\phi}$
	5*3–1	jelentése	5 és 3 szorzatából le kell vonni 1-et
	(5*3)–1	jelentése	ugyanaz
	5*(3–1)	jelentése	3 és 1 különbségével kell 5-öt megszorozni.

A felesleges többlet-zárójelek használata nem hiba: (((5)*(3))–1) ugyanaz, mint 5*3–1.

*Vigyázat: 6/2*3 nem azt jelenti, hogy 6-ot osszuk el 2*3-mal, hanem 6-ot osszuk el 2-vel és az eredményt szorozzuk 3-mal. (Tehát nem* $\frac{6}{2*3}$, *hanem* $\frac{6}{2}*3$.*) Ha* $\frac{6}{2*3}$ *értékét akarjuk a géppel kiszámoltatni, írjuk így: 6/(2*3) vagy 6/2/3.*

Vigyázat: nincs tizedesvessző, hanem tizedespont van. A számok ezresek szerinti szokásos tagolása tehát tilos: 1.234.567,75 helyesen: 1234567.75.

A műveleteket a gép a matematikában megszokott módon, vagyis a következő sorrendben végzi el:

először a hatványozásokat,
utána a szorzásokat-osztásokat,
utána az összeadásokat-kivonásokat.

Ha azonos szintű utasítások állnak egymás mellett, azokat a gép szigorúan balról jobbra haladva hajtja végre.

Zárójelezéssel a fenti sorrendet tetszőlegesen befolyásolhatjuk, a matematikában megszokott módon.

Ha egy számnak előjele van, az a szám részének minősül, tehát a gépi „tudomásulvétele" minden műveletet megelőz. Így például $2\uparrow-1\phi$ jelentése: $2^{-1}\phi$, 5--4 jelentése 5–(–4), 5-------4 jelentése 5–(–(–(–(–(–(–4))))))) és így tovább.

A számokat normálalakban is begépelhetjük és a túl nagy vagy túl kis számokat a gép is normálalakban jeleníti meg. Például PRINT $1\phi\phi\phi\phi\phi\phi\phi\phi\phi$ hatására 1E+9 jelenik meg, ami azt jelenti, hogy $1\cdot1\phi^{9}$. A géppel feldolgozható legnagyobb szám $1\phi^{38}$ (normálalakban 1E38), a legkisebb $1\phi^{-38}$ (normálalakban 1E–38).

A normálalak három részből áll: két számból és köztük az E betűből. A jelentése: első szám * $1\phi^{\text{második szám}}$, ahol a második szám csak egész lehet.

Az AIRCOMP-BASIC csak hat számjegyet tud megjeleníteni: PRINT 1234567 hatására 1.23457E+6 jelenik meg, vagyis egyegész 23457 százezred szorozva tíz a hatodikonnal. (Az utolsó jegy a kerekítés miatt lett hetes.) Ez azt is jelenti, hogy a túl sok számjegyet tartalmazó számokat a gép akkor is pontatlanul veszi tudomásul, tárolja, ha akármilyen pontosan is gépeljük be. A legnagyobb ábrázolható szám 1.06338 E +38, a legkisebb ábrázolható szám pedig 9.999999 E –38.

Megvan a lehetőség arra, hogy a kulcsszavakat egyetlen gombnyomással írjuk le. A PRINT az egyik leggyakoribb kulcsszó, ezért ennek rövid megjelenítésére két lehetőség is van: valamelyik SH-gombot lenyomva tartva nyomjuk meg a RUN/BRK gombot, vagy gépeljünk be egyszerűen egy kérdőjelet.

A többi kulcsszó rövid begépelése általában a SH-gombok segítségével történik. Ha egyik SH-gombot (shift-gombot, váltógombot) sem tartjuk lenyomva, akkor a kétfeliratos gombok alsó felirata, a háromfeliratos gombok középső felirata érvényes. A jobboldali SH-gombot lenyomva tartva a többi gomb jelei közül mindenütt a felső az érvényes. Ha pedig a baloldali SH-gombot tartjuk lenyomva, akkor a háromjelű gombok alsó jelei érvényesek; kétjelű gombok esetében a kétféle SH ugyanazt eredményezi: a felső jel megjelentetését. Ha SH nélkül nyomunk le egy gombot, mélyebb, SH-val magasabb csipogó hangot hallunk; ez annak a visszajelzése is, hogy az SH-t elég erősen nyomtuk-e le. Ha valamelyik gombot hosszabb ideig tartjuk lenyomva, akkor úgy viselkedik, mintha többször nyomtuk volna le.

A függvényhasználatról külön fejezet szól. Annyit azonban már most is érdemes tudni, hogy sokféle függvényt használhatunk, a matematikában megszokott módon. Például

```
PRINT SIN(1)CR
```

hatására

```
.841471
```

jelenik meg, ami azt jelenti, hogy 1-nek a színusza ϕ,841471. (1 radiánban értendő!). A függvények kifejezések részét is képezhetik, pl.

```
PRINT 1-SIN(3.1416/6)
```

hatására

```
.5
```

jelenik meg.

```
AR=15CR
PRINT ARCR
```

hatására a gép (a következő sorba) kiírja a 15-ös számot.

Kulcsszó: =
Jelentése: *legyen* egyenlő

Az első parancs hatására a gép „tudomásul veszi", hogy az AR értéke 15. A második parancs hatására kiírja az AR értékét.

Az adatoknak tehát elnevezéseket adhatunk. Az adatnevek egy vagy két betűből (csak betűkből!) állhatnak. Pontosabban: hosszabb adatnevet is begépelhetünk, de a gép csak az első két betű alapján különbözteti meg a neveket, így például a SZAMLALO és a SZORZAT nevű adatokat azonosaknak tekinti.

Vigyázat: az adatnév nem tartalmazhat kulcsszót! Hibás például az EZAZAMITKIKELLPRINTELNI *adatnév, mert a* PRINT *betűsorozatot tartalmazza.*

Az adatnevek kötelező rövidsége nem teszi lehetővé, hogy értelmes elnevezéseket használjunk. Annál fontosabb, hogy legalább értelmes rövidítéseket igyekezzünk kitalálni. Néhány hét elteltével már a legrövidebb programunkat is csak nehezen fogjuk tudni megérteni, ha nem alakítunk ki saját magunk számára követhető adat-elnevezési szokásokat.

Mind az értékadás, mind a kiírás tetszőleges matematikai kifejezésre is vonatkozhat. Így például

```
AR=15*3/2-8[CR]
PRINT AR[CR]
```

ugyanaz, mint

```
AR=15[CR]
PRINT AR*3/2-8[CR]
```

vagyis ugyanaz, mint

```
PRINT 15*3/2-8[CR]
```

Figyelemre érdemes, hogy az egyenlőségjel itt nem egyenlőséget jelent, hanem az egyenlővé tétel műveletét. Az I=I+1 például matematikailag értelmetlen, de mint parancs nagyon értelmes. Azt jelenti, hogy az I nevű adat (az I változó) eddigi értékéhez hozzá kell adni 1-et és mostantól kezdve az I új értéke *legyen* ennek az összeadásnak az eredménye.

Eddig tehát minden megoldható a gépünkkel, amit egy zsebszámológépen is meg tudnánk oldani; olyan zsebszámológépen persze, amelynek annyi memóriája van, ahányféle különböző változónevet csak ki tudunk találni. Például:

```
DB=6
AR=3.5ϕ
KP=DB*AR
PRINT KP
```

A [CR] jelet itt és a továbbiakban már elhagyjuk a leírásból, hacsak nem akarjuk valami miatt külön hangsúlyozni.

A fenti parancs-sorozat azt számolja és írja ki, hogy egy DB darabszámú és AR egységárú termékért mekkora KP összeget kell fizetni. A parancsok végrehajtása után természetesen mindhárom felhasznált változó értéke még rendelkezésre áll tetszőleges további műveletek számára, amíg meg nem változtatjuk, amíg nem töröljük őket, vagy amíg ki nem kapcsoljuk a gépet.

Ezekből a műveletekből teljes számolási programot is összeállíthatunk a gép számára.

Egy *program* lényege az, hogy milyen műveleteket milyen sorrendben kell elvégezni. A műveleteket a kulcsszavak azonosítják; a sorrend meghatározása végett meg kell számoznunk őket.

```
1 DB=6
2 AR=3.5ϕ
3 KP=DB*AR
4 PRINT KP
```

Ez a számozás nagyon célszerűtlen. Mi történik, ha kiderül, hogy az érték előtt a darabszámot és az egységárat is ki kell írni? Az eddigi 4-es sort újra kell számoznunk, és még szerencse, hogy csak ezt az egyet. Számozzunk inkább így:

```
1ϕ DB=6
2ϕ AR=3.5ϕ
3ϕ KP=DB*AR
4ϕ PRINT KP
```

Ha a fentebbi bővítési igény felmerül, gépeljük be akár utólag, hogy

```
33 PRINT DB
36 PRINT AR
```

A sorok számai ϕ-tól kezdődhetnek és tetszőleges kihagyásokkal 32 759-ig haladhatnak.

Az egyes sorszámozott sorok begépelése után a gép semmiféle műveletet nem végez azon kívül, hogy a sort elhelyezi saját memóriájában. A sorokban leírt műveleteket tehát ilyenkor a gép még nem hajtja végre; mi több: nem is ellenőrzi őket. A hibás sort (pl. PRINT helyett RPINT) is elfogadja, csak a *végrehajtáskor* jelez hibát, amikor a rossz sorhoz ér.

A létrehozott program új kulcsszóval indítható el.

Kulcsszó: RUN
Jelentése: fuss

A kulcsszó betű szerinti begépeléssel vagy a RUN/BRK gombbal állítható elő.

A kulcsszó hatására a korábban begépelt, számozott sorokban lévő műveletek sorra végrehajtódnak, a számozás sorrendjében (tehát *nem* a begépelés sorrendjében). Újabb és újabb RUN hatására a műveletsorozat újra és újra megismétlődik.

A BASIC-programozók a sorszám nélküli műveleteket parancsoknak, a sorszámozottakat utasításoknak nevezik. Tehát

```
PRNT KP
```

parancs,

```
4ϕ PRINT KP
```

pedig utasítás. A parancs a begépeléskor azonnal végrehajtódik, majd elvész, és ha újból végre akarjuk hajtatni, újból be kell gépelnünk. Az utasítás csak RUN parancs hatására hajtódik végre, (valamennyi többi begépelt utasítással együtt), és megőrződik: ha újból végre akarjuk hajtatni, akkor csak újabb RUN parancsot kell kiadnunk.

A LIST parancs hatására a memóriában lévő program megjelenik a képernyőn.

Kulcsszó: LIST
Jelentése: listázd ki

A programsorok a sorszámok sorrendjében íródnak ki.

Ha a programba kettőnél több betűből álló változónevet írtunk, listázáskor már csak az első két betű jelenik meg. Ha a programba a PRINT kulcsszó helyett kérdőjelet írtunk, listázáskor már a PRINT kulcsszó jelenik meg.

Még egy tekintetben különbözhet a kilistázott programszöveg attól, amit begépeltünk: a szóközök számában. Kulcsszó belsejébe nem írhatunk szóközt, vagyis PRINT helyett nem írhatjuk azt, hogy P R I N T. Bárhol másutt, akár a változónév két betűje közt, akár adat kellős közepén akárhány szóköz lehet a begépeléskor – de sehol nem kötelező egy sem. Nem vét tehát hibát az, aki ezt írja:

```
1ϕ   D B  =  6
2ϕAR=3.5      ϕ
```

és így tovább. A gép mindenesetre egységes formában tárolja és listázza ki az utasításokat, egy-egy szóközzel értelmesen tagolva.

Ha hosszabb programot listázunk ki, akkor nem fér rá az egész program a képernyőre: az első sorait megnézni nincs is időnk, mire már el is tűnnek. Részletekben a következőképpen tudunk egy programot kilistázni:

A LIST 2ϕ–4ϕ parancsra a 2ϕ-as, a 4ϕ-es és minden köztük lévő sor jelenik meg a képernyőn; a LIST 2ϕ – parancsra a 2ϕ-as és minden azt követő sor; a LIST –4ϕ parancsra a 4ϕ-es és minden azt megelőző sor; a LIST 4ϕ parancsra csak a 4ϕ-es sor. Ha a LIST-ben olyan sorszámot adunk meg, amilyen a programban nem létezik, a gép a megadott sorszámot akkor is olyan határként értelmezi, amely alá ill. fölé nem szabad a listázással mennie. Például ha 19-es és 41-es sor nincs a programunkban, akkor a LIST 19–41 ugyanazt jelenti, mint a LIST 2ϕ–4ϕ.

Új sort beszúrni egyszerűen úgy kell, hogy begépeljük az új sort, természetesen új sorszámmal.

Ha olyan sort gépelünk be, amivel azonos sorszámú sor már van a programunkban, akkor az eredeti sor törlődik, és ezentúl az a sor lesz érvényes, amit begépeltünk.

Sort törölni úgy lehet, hogy begépeljük a sorszámot és rögtön utána a CR jelet. Vigyázzunk: ha a sorszám és a CR közé szóköz is kerül, ezt a gép begépeléskor elfogadja, de futás közben hibásnak találja és ettől a programfutás megszakad!

A programot teljes egészében is ki lehet törölni.

Kulcsszó: NEW
Jelentése: új

A NEW parancs a teljes programot törli, megkezdhetjük az új program begépelését. NEW végrehajtása után az első bejelentkezésnél megismert felirat jelenik meg.

A meglévő sor kijavításának legnyilvánvalóbb módja az, hogy a sort újragépeljük. Ez a módszer azonban nemcsak kényelmetlen, hanem veszélyes is: minél hosszabb sort gépelünk újra, annál nagyobb a valószínűsége annak, hogy miközben egy hibát kijavítunk, újabb hibát követünk el.

Sokkal biztonságosabb a meglévő sorba belejavítani. Ehhez az kell, hogy a sor látsszon valahol a képernyőn. (Ha nem látszik, ki kell listáznunk.) Ezután a billentyűzet jobb felső sarkában és az az alatt lévő gombok segítségével a begépelések helyét jelző világító kis téglalapot, a kurzort a javítás helyére kell mozgatnunk. Most felülírhatjuk, amit elrontottunk. A sor javítását a CR gombbal kell befejeznünk.

Megjegyzés: Ha a sor közepén javítottunk, nem szükséges a kurzort a sor végére kivinni; a CR lenyomásakor a gép a *teljes* sort a memóriába teszi.

Ilyen javításkor van szerepe a ϕ számjegy gombján a két feliratnak: INS (insert: beszúrás, betoldás;) és DEL (delete: törlés). Ahányszor az INS jelet kiadjuk, annyi új szabad betűhely nyílik a sorban, hogy oda új szöveget írhassunk; ahányszor a DEL jelet kiadjuk, annyi betűhelynyi szöveg tűnik el, annyival húzódik összébb a sor. (Az INS úgy működik, hogy az éppen alatta lévő betűt vagy más jelet és a sor teljes további részét eggyel jobbfelé tolja. A DEL úgy működik, hogy az éppen alatta lévő betűt vagy más jelet elnyeli és a sor további részét eggyel balra húzza; ha pedig tőle jobbra már nem folytatódik a sor, akkor ő maga lép egyet balra és elnyeli azt a betűt vagy más jelet, amire rálép.)

Vigyázzunk: ha a kurzorral a képernyő felsőbb soraiban járunk és alulra akarunk visszatérni, ezt nem célszerű a [CR] *gomb nyomogatásával megtennünk, mert ahány régi sort, parancsot, eredményt stb. átlépünk, a gép mindegyiket úgy veszi, mintha most újonnan megint begépeltük volna! Használjuk a ↑↓ kurzor mozgató parancsot.*

A progamírás értelme persze nem az, hogy egymásután százszor írathatjuk ki a géppel a 6·3,5ϕ-et, hanem az, hogy egymásután száz különféle darabszámú és egységárú termék összértékét is kiszámoltathatjuk.

```
1φ INPUT DB
2φ INPUT AR
3φ KP=DB*AR
4φ PRINT KP
```

Kulcsszó: INPUT
Jelentése: (adat)bevitel

Ha a fenti programot a RUN paranccsal elindítjuk, akkor kérdőjel jelenik meg a képernyőn, mutatva, hogy a program adatot vár: a darabszámot. Gépeljünk be egy számot és nyomjuk le a [CR] gombot; erre új kérdőjel jelenik meg, mutatva, hogy a program a második adatra, az egységárra vár. A második szám begépelése és a [CR] lenyomása után megjelenik az eredmény.

Ha a RUN parancsot újból kiadjuk, a program újból elindul és újból végigfut, most már újabb adatokkal.

Érdemes észrevenni, hogy a gépünkből most szorzógépet csináltunk. Ha elindítjuk és begépelünk két számot, kiírja a két szám szorzatát. Az a gép tehát, amely program nélkül egyszerű kalkulátorként működött, most szorzó célgéppé vált. Ugyanígy ha például számlakészítő programot írunk, akkor a gép számlakészítő géppé alakul át.

A számítógép nem gép, hanem csupán egy gép lehetősége. Működőképes célgéppé a program teszi. Mintha létezne egy univerzális szerszámgép-alap, amelyre különféle modulokat illesztve hol fúrót, hol marót, hol esztergapadot, hol mást kapnánk, de maga a gép-alap a rárakott modulok nélkül még használhatatlan – így válik ugyanaz az üres számítógép más-más programok hatására hol szorzógéppé, hol könyvelőgéppé, hol folyamatirányító-géppé, hol épülettervező-géppé, hol játékká és így tovább.

```
1φ INPUT DB
2φ INPUT AR
3φ KP=DB*AR
4φ PRINT DB;"*";AR;"=";KP
```

Egy-egy PRINT-utasításban több kiírnivalót is felsorolhatunk. Kiírnivaló lehet névvel jelölt adat vagy konstans. A konstans lehet szám vagy szöveg. A számot a szokásos módon írjuk le, a szöveget két idézőjel között. (A sor végén a záró idézőjel elhagyható, de nem tanácsos elhagyni, mert programjavításkor, ha a sort bővítjük, zavart okozhat.)

Általános, a teljes BASIC-nyelvre érvényes szabály, hogy ahol adatnév vagy konstans állhat, ott mindenütt állhat adatnevekből és konstansokból felépülő kifejezés is. A 3ϕ-as utasítást tehát el is hagyhattuk volna és írhattuk volna a 4ϕ-es utasítást így:

4ϕ PRINT DB;"*";AR;"=";DB*AR

Ennek a megoldásnak egyedüli hátránya az, hogy ha a program további részeiben még sok helyen fel akarnánk használni a DB*AR szorzat értékét, újból és újból ki kellene számoltatnunk a géppel, ami időveszteséget jelent. Ha viszont a 3ϕ-as utasítást meghagyjuk, akkor a KP nevű változó megőrzi az egyszer kiszámolt értéket. Általában tehát: ha egy értékre többször is szükségünk van a programvégrehajtás során, akkor érdemes tárolnunk, ha viszont csak egyszer, például ha csak egyetlen kiíráshoz kell, akkor jobban megéri a felhasználás helyén kiszámoltatni, rögtön felhasználni és elfelejteni.

Az idézőjelek közé zárt szövegben nem lehet idézőjel. Például az "A CSO 2"–OS" szöveget a gép így érti: "A CSO 2" egy szöveg, amelyből le kell vonni egy számot, tudniillik az OS nevű változó értékét. Ez tehát (szövegből számot levonni) hibás művelet.

A szövegen belül bármilyen betű vagy más jel lehet, ami a billentyűzeten létezik. Ha két idézőjel között vagyunk, akkor a négyféle kurzormozgató nyíl lenyomása, valamint az INS és a DEL gomb lenyomása nem ezeknek a gomboknak az eredeti funkcióit jelenti, hanem (speciális jelentéssel) ezek a jelek bekerülnek a szövegbe. Ha vissza akarunk térni az eredeti jelentésükhöz, előbb ki kell írnunk a záró idézőjelet vagy a CR gombbal be kell fejeznünk a sort.

Ha a kiírnivaló sorban idézőjelek között jobbra, balra vagy felfelé mutató nyíl van, az a kurzor léptetését jelenti a megjelölt irányba, a már kiírt szöveget azonban csak akkor törli, ha valamit a helyére írunk:

```
    PRINT "ABC←DEF"
```

hatása:

```
ABDEF
```

```
    PRINT "ABC→DEF"
```

hatása:

```
ABC DEF
```

```
    PRINT "ABC←←D"
```

hatása:

```
ADC
```

és így tovább. Ha nyilakkal túlléptetjük a sor elejét vagy végét, a kiírás zavartalanul folytatódik a megelőző illetve a következő sorban.

Ha az idézőjelek között lefelé mutató nyíl van, akkor alapértelmezés szerint ezen a ponton a kurzor egy sorral lejjebb lép, és a kiírás onnan folytatódik. Például a

```
PRINT " ↓ABC DEF"
```

hatása:

```
ABC
    DEF
```

Vigyázat!
Ez az alapértelmezés nem érvényesül, ha a kiírás a képernyő legalsó sorában kezdődött el. Ilyenkor a teljes kép egy sorral feljebb mozog, és a nyíl új sor kezdését eredményezi. Így a fenti példa szerinti kiírás

```
ABC
DEF
```

lesz.

Ha az idézőjelek között DEL jel van, ez ugyanazt jelenti, mint a balra mutató nyíl, de a visszafelé átléptetett jeleket mind törli.

PRINT "ABC DEL DEL D" (a képernyőn "ABC<<D" jelenik meg)

hatása:
AD

Ha az idézőjelek között INS (a képernyőn >) jel van, ennek nincs hatása.

Változó is kaphat szöveg-értéket, az ilyen változókat azonban meg kell különböztetnünk a többitől azáltal, hogy a nevük után dollárjelet írunk: SZ$="JONAPOT!" vagy UJ$ =O$. Ha a programban van például X nevű változó, attól még lehet X$ nevű változó is.

Szövegek között egyetlen művelet végezhető: egyiknek a másik után való illesztése. Jele a pluszjel. Például az

```
E$="EZ A"
V$="Z EGESZ"
SZ$=E$+V$
```

parancssorozat hatására SZ$ tartalma ez lesz: EZ AZ EGESZ.

A PRINT utasításban felsorolt kiírnivalókat el kell választani egymástól. Kétféle elválasztójel van: a pontosvessző és a vessző.

Ha pontosvesszőt írunk, akkor a következő kiírnivaló közvetlenül az előző után jelenik meg (nemnegatív szám esetén elől egy szóközzel, amely az előjel helye). Például

```
PRINT "A";"B";"C"
```

hatása:

```
ABC
```

```
PRINT 3;-6;"*"
```

hatása:

```
3-6*
```

(a hármas előtt a sor elején egy szóközzel).

Ha vesszőt írunk, akkor a BASIC a képernyőt nyolckarakteres mezőkre osztja, összesen öt darabra, és az új adat vagy konstans kiírását mindig a soronkövetkező szabad mező elején kezdi. (A mező akkor számít szabadnak, ha az előző mezőnek legalább a legvége üres.) Például

```
PRINT "1234567", "8"
```

hatása:

```
1234567 8
```

```
PRINT "1234567",, "8"
```

hatása:

```
1234567         8
```

mert a két adat közt egy üres mező is van,

```
PRINT "12345678", "9"
```

hatása:

```
12345678    9
```

mert a második mező nem üres, hiszen az őt megelőző mező legvége nem üres,

```
PRINT "A", "B", "C", "D", "E", "F"
```

hatása:

```
A    B    C    D    E
F
```

mert a képernyő 40 karakternyi széles, tehát a 41. karakter már az új sor elejére kerül. A nemnegatív számot itt is szóközzel kezdődnek, vagyis maga a szám az általa elfoglalt mező második karakterén kezdődik el.

Ugyanezek érvényesek az utasítás végére is. Tehát

```
10 PRINT "A",
20 PRINT "B";
30 PRINT "C"
```

ugyanazt jelenti, mint

```
10 PRINT "A", "B"; "C"
```

feltéve, hogy az első megvalósítás három PRINT-utasítása között sem beolvasó, sem kiíró utasítás nem volt; másmilyen utasítás akármennyi lehetett közöttük.

A magában álló, kiírnivaló nélküli PRINT-utasítás, például

```
15φ PRINT
```

egy üres sor kiírását eredményezi. Ha előtte elválasztójellel végződő PRINT volt, vagyis most egy korábbi sor folytatását kellene az új PRINT-nek kiírnia, akkor a magában álló PRINT ennek a sornak a befejezését, lezárását jelenti.

```
1φ INPUT "DARAB=";DB
2φ INPUT "EGYSEGAR=";AR
3φ KP=DB*AR
4φ PRINT DB;"*";AR;"=";KP
```

Az INPUT-utasítás sem csak beolvasnivaló adatot tartalmazhat, hanem bármi mást is, amit a PRINT-utasítás. Ha az INPUT-utasításban más van, mint beolvasnivaló adat, akkor az megjelenik a képernyőn. Ha beolvasnivaló adat következik, akkor a gép vár az adat begépelésére. A példánk 1φ-es utasításának hatására tehát ez lesz a képernyőn:

DARAB=

És a program várja az adatot. Így oldhatjuk meg tehát, hogy a program minden adatbeolvasás előtt közölje a felhasználóval, hogy most melyik adatra várakozik. Ha tehát a beolvasnivaló adat előtt közvetlenül valami kiírnivaló volt, akkor nem jelenik meg a beolvasáskor eddig megszokott kérdőjel. Ugyanígy nem jelenik meg akkor sem, ha a beolvasnivaló adat és az őt közvetlenül megelőző másik adat neve közt pontosvessző van. Ha azonban adatnév, vessző, majd újabb adatnév következik az INPUT-utasításban, akkor a második adat beolvasása előtt megjelenik a kérdőjel. Egyebekben a vessző és a pontosvessző szerepe itt is ugyanaz, mint a PRINT esetében, annyi különbséggel, hogy az adat beolvasása mindig soremeléssel jár. Például

```
INPUT "←";A$
```

hatására a kurzor a következő sor legelejére áll (hiszen kiíródott az egybetűnyi visszalépés, majd egy szóköz, és így marad a kurzor a sor elején), és itt várja a szöveges adat begépelését. Kiírt szöveg elé például így kérhetünk adatbegépelést a program kezelőjétől:

```
INPUT "AR=→→→→→→FT←←←←←←";AR
```

Az INPUT X,(X),2*X,X*2 parancs hatására a következők történnek:

- a gép beolvassa az X értékét,
- (X) hatására ki is írja az X értékét, (mert ami zárójelben van, az biztosan nem lehet beolvasnivaló, tehát ki kell írnia,)

– 2*X hatására X értékét megszorozza 2-vel és kiírja az eredményt, (mert ami konstanssal kezdődik, az biztosan nem lehet beolvasnivaló, tehát ki kell írnia,)
– majd újból beolvassa X értékét, majd pedig hibát jelez (mert amikor „észreveszi" az X-et, azonnal értelmezni próbálja, *anélkül, hogy végigolvasná a sor hátralévő részét;* ha pedig még csak az X-et olvasta el, értheti úgy, hogy beolvasni kell; ezt végre is hajtja; majd elválasztójelet vár, de csillagot talál, tehát hibát jelez.)

Ha egy számadatot kérő INPUT-utasítására csak a CR-t nyomjuk le, a programfutás leáll.

Ha egy szövegadatot kérő INPUT-utasításra csak a CR-t nyomjuk le, a gép úgy értelmezi, hogy üres (ϕ hosszúságú) szöveget gépeltünk be.

Adat helyett tetszőleges kifejezést is begépelhetünk; a gép kiszámolja a kifejezés értékét és úgy tesz, mintha ezt az értéket gépeltük volna. Ha hibás kifejezést írunk, a gép megismétli a kérdést.

Vigyázzunk: ha az INPUT-utasításra begépelt szöveg idézőjelet tartalmaz, ez megszakítja a begépelt szöveget. Például ha az INPUT A$ *utasításra vagy parancsra az* ABC"DEF *szöveget gépeljük be,* A$ *tartalma csak ABC lesz.*

Egy INPUT-ban több adatbevitel is lehet: INPUT A,B,C

Egészítsük ki programunkat az

5ϕ GOTO 1ϕ

sorral! A program lefuttatásakor azt tapasztaljuk, hogy az eredmény kiírása után a program újból felteszi az első kérdést és így tovább.

Kulcsszó: GOTO
Jelentése: menj

A GOTO kulcsszó hatására a programfutás nem a soronkövetkező utasítás végrehajtásával folytatódik, hanem azzal az utasításéval, amelynek sorszámát a GOTO után írtuk. A GOTO tehát feltétlen vezérlésátadást hajt végre. (Szokás ugróutasításnak is nevezni.)

Természetesen a GOTO után szám helyett kifejezés is állhat, de ez nem kezdődhet számmal. Pl. GOTO I+5 jó, GOTO 5+I nem jó, GOTO (5)+I jó.

A GOTO is használható parancsban is. A mi programunkban, ahol a legkisebb előforduló sorszám a 1ϕ-es, a

GOTO 1ϕ

parancs egyenértékű azzal, mintha a RUN parancsot adtuk volna ki – egy igen fontos szemponttól eltekintve: a RUN hatására valamennyi változó értéke törlődik, a GOTO 1ϕ hatására viszont megmaradnak mindazok az értékek, amelyek egy esetleges korábbi programfutás során a változókba kerültek.

(Mind a GOTO, mind a RUN parancs, lehetővé teszi, hogy a programunkat ne az első, hanem bármelyik másik utasítástól indítsuk el. Ha például a 4ϕ-es sorszámú utasítástól akarunk indulni, kiadhatjuk a GOTO 4ϕ parancsot, és tapasztalhatjuk, hogy minden változó-érték megmaradt, vagy kiadhatjuk a RUN 4ϕ parancsot, és tapasztalhatjuk, hogy minden változó-érték törlődött, elveszett.)

A GOTO kulcsszó eredeti írásmódja GO TO. (GO=menj; TO=-hoz, -hez, -höz.) Az AIRCOMP-BASIC azonban kulcsszó belsejében nem engedi meg a szóközt.

Az így kiegészített program tehát olyan szorzóprogram, amely újra és újra szoroz, anélkül, hogy újra és újra el kellene a folyamatot indítanunk. Kikapcsolni csak erőszakos úton lehet. (Ennek a módjairól később szó lesz, mint ahogyan arról is, hogy hogyan kerülhetjük el az ilyesfajta végtelen ciklusok véletlen létrehozását.)

Elvileg nincs akadálya annak, hogy egy sorba több utasítást vagy parancsot írjuk. Ezek elválasztására a kettőspont szolgál. Például

```
4ϕ PRINT ERTEK
5ϕ GOTO 1ϕ
```

helyett írhatjuk azt, hogy

```
4ϕ PRINT ERTEK : GOTO 1ϕ
```

(Az átalakítás módja például az lehet, hogy a 4ϕ-es sor végére kettőspontot gépelünk, majd annyi szóközt, hogy még a következő képernyő-sorban lévő 5ϕ-es sorszámot is töröljük vele. Ha ezután a CR-t lenyomjuk, a két sor összeolvadt.)

Az AIRCOMP-on egyetlen sorszámhoz 1ϕϕϕ karakternyi utasítássorozat tartozhat és egy-egy parancssorozat is 1ϕϕϕ karakteres lehet, vagyis akár az egész képernyőt is betöltheti. Egyetlen sorszám felhasználásával így viszonylag bonyolult programot is megírhatunk. Az utasítások egy sorba írása kisebb tárigényű, mintha minden utasításhoz új sorszámot írnánk. A program áttekinthetősége és főleg javíthatósága azonban jelentősen leromlik. Képzeljük el, amikor futás közben olyan hibajelzést kapunk, hogy elírás van a 2ϕ1ϕ-es sorszámú sorban – és ez éppen egy teljes képernyőnyi sor! Ilyen sorban hibát keresni, ebbe új utasítást betoldani, innen törölni nemcsak kényelmetlen, hanem veszélyes is.

Azt tanácsoljuk, hogy a programozó csak akkor írjon utasításokat egy sorba, ha erre külön oka van, és akkor is csak logikailag összetartozó utasításokat, amelyeket a program megváltozása esetén amúgy is egyszerre kellene módosítani.

FELTÉTELES TEVÉKENYSÉGEK

A programutasítások végrehajtása a sorszámok sorrendjében folyik, kivéve a vezérlésátadó utasítás hatására, amelyben előírhatjuk a következő végrehajtandó sor sorszámát.

Az előbb bemutatott programunk azonban meglehetősen ügyetlen; sokkal jobb lenne, ha egyszerű adatbevitellel vezérelhetnénk, hogy mikor hagyja abba a feldolgozást. Ehhez azonban azt a vezérlésátadó utasítást kell használnunk, amely egy feltételtől függően tudja megváltoztatni a programfutás menetét.

```
1ϕ INPUT "DARAB=";DB
2ϕ INPUT "EGYSÉGAR=";AR
3ϕ KP=DB*AR
4ϕ PRINT DB;"*";AR;"=";KP
5ϕ INPUT "FOLYTASSAM? ";F$
6ϕ IF F$="IGEN" THEN GOTO 1ϕ
```

Kulcsszavak: IF = THEN
Jelentésük: ha, egyenlő, akkor

Az első számolás végrehajtása után a program a

FOLYTASSAM?

kérdést írja ki. Ha azt gépeljük be, hogy IGEN, akkor a végrehajtás a 1ϕ-es sorszámú sortól folytatódik, azaz kezdődik az új számolás. Ha bármi mást gépelünk be a gép kérdésére, akkor az IF-utasításban álló feltétel nem teljesül, ezért a THEN utáni utasítás (a vezérlésátadás) nem hajtódik végre, vagyis a vezérlés egyszerűen továbbhalad, mintha az IF ott sem lett volna. Mivel pedig a programnak nincs több utasítása, a feldolgozás leáll.

Figyeljük meg, hogy itt az egyenlőségjel jelentése valóban az, hogy valami valamivel egyenlő, nem pedig az, hogy valami valamivel egyenlő *legyen.*

A THEN kulcsszó után bármilyen utasítás vagy utasítássorozat is állhat, például

```
21ϕ IF A=B THEN PRINT A, : X=Y+3
147ϕ IF C=1ϕ THEN INPUT C : C=C/1.8 : GOTO 157ϕ
```

Az IF érvénye a teljes sorra kiterjed: ha a feltétel nem teljesül, akkor a THEN utáni utasítások egyike sem hajtódik végre. Például az

```
5 Z=2 : IF I=1 THEN A=—A : IF J=1 THEN B=ϕ
```

utasítássorban Z=2 mindenképpen végrehajtódik, A=–A csak akkor, ha I=1, és B=ϕ csak akkor, ha I=1 és J=1. (Ha I nem egyenlő 1-gyel, akkor a J=1 feltételt a gép már meg sem vizsgálja.)

Az A=B vagy más hasonló állításoknak is „számértékük" van a gépen belül: az 1-es jelenti azt, hogy „igaz", a ϕ azt, hogy „nem igaz". Ezt kihasználva ilyen feltételeket is írhatunk:

```
583 IF X THEN ...
```

A gép akkor tekinti úgy, hogy a feltétel nem teljesült, ha X értéke nulla, és akkor tekinti a feltételt teljesültnek, ha X értéke más: 1 vagy bármi, de legfeljebb 255.

Az ilyen utasítás végrehajtási menete az, hogy HK-t a gép logikai változónak tekinti és ezért a lefelé kerekített értékét vizsgálja meg (–1,5-öt például –1-re kerekíti). Ha ez az érték ϕ, akkor úgy tekinti, hogy a feltétel nem teljesült. Ha ez az érték 1 és 255 közé esik, akkor úgy tekinti, hogy a feltétel teljesült. Ha pedig ez az érték ϕ-nál kisebb vagy 255-nél nagyobb, akkor hibajelzést kapunk.

Erről a lehetőségről és a célszerű használatáról még lesz szó; most csak annyi a lényeg, hogy egyfajta veszélyre hívjuk fel a figyelmet. Tegyük fel, hogy a

```
117 A=B:C=D
```

utasítást a programozó elgépeli és a gépbe ez kerül:

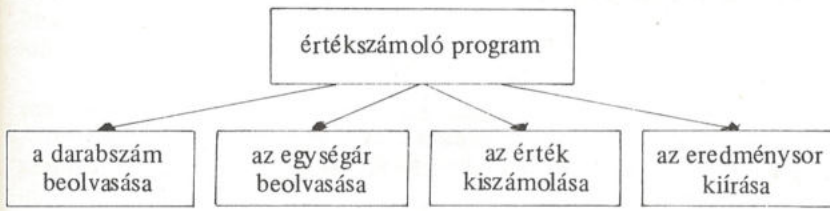
```
117 A=B=C=D
```

A gép ezt elfogadja és így értelmezi: B=C egy logikai kifejezés, amelynek az értéke 1, ha B egyenlő C-vel és ϕ, ha B nem egyenlő C-vel. B=C=D egy újabb logikai kifejezés, amelynek az értéke 1, ha D egyenlő a B=C kifejezés értékével és ϕ, ha B nem egyenlő a D=C kifejezés értékével. Ezt az értéket (tehát a B=C=D logikai kifejezés értékét) kell A-nak felvennie. Vigyázat tehát: a BASIC itt új lehetőséget nyújt a programozónak, amellyel élni jó, hatékony lehet, de egyben veszélyhelyzetet is teremt, amikor egy egyszerű elgépelés miatt a programunk látszólag jól, valójában rossz eredményekkel hajtódik végre.

Az alábbiakban – eddigi programjaink összefoglaló áttekintése kapcsán – a programtervezés néhány eszközét és alapfogalmát mutatjuk be.

Az IF és a GOTO nélküli programjaink nagyon primitív szerkezetűek voltak: az utasítások minden feltételtől függetlenül mindig azonos sorrendben hajtódtak végre, például így:

Amint a program *folyamatábrájából* látszik, ez a program *lineáris.*



Amint a *struktúraábrából* látszik, ez a program csak egymást követő blokkokat, úgynevezett *szekvenciákat* tartalmaz.

A felrajzolt kétféle ábra a programtervezés és a programdokumentálás két leginkább elterjedt grafikus eszköze.

A kettő között nagy a különbség: az első a műveletekre és ezek végrehajtási sorrendjére koncentrál, a második pedig a tevékenységek egymáshoz való viszonyára. Az elsőnél, ha új nyilat húzunk, ezt a kérdést tesszük fel: mi következik ez után? A másodiknál: milyen részekből áll ez a tevékenység?

A lineáris programokat kétféleképpen is kódolhatjuk: vagy úgy, ahogy eddig tettük, hogy egyszerűen egymás után írjuk az utasításokat, vagy úgy, hogy a jobban összetartozó utasításokból külön, önálló rész-eljárást szervezünk és arra a végrehajtás helyén csak hivatkozunk.

Ezzel a második módszerrel a korábbi programunkat így is leírhatjuk:

ahol a bemenő adatok megszerzésének folyamata a következő:

A program szerkezete:

Maga a program:

```
1φ GOSUB 1φφφ
2φ KP=DB*AR
3φ PRINT DB;"*";AR;"=";KP
4φ END
1φφφ INPUT "DARAB=";DB
1φ1φ INPUT "EGYSEGAR=";AR
1φ2φ RETURN
```

Kulcsszavak:	GOSUB	RETURN	END
Jelentésük:	hajtsd végre,	térj vissza,	vége

A GOSUB (a GO=menj és a SUBROUTINE=alprogram, szubrutin szavak összevonása) azt jelenti, hogy a programvégrehajtásnak a GOSUB-bal megjelölt sorszámú sortól kell folytatódnia.

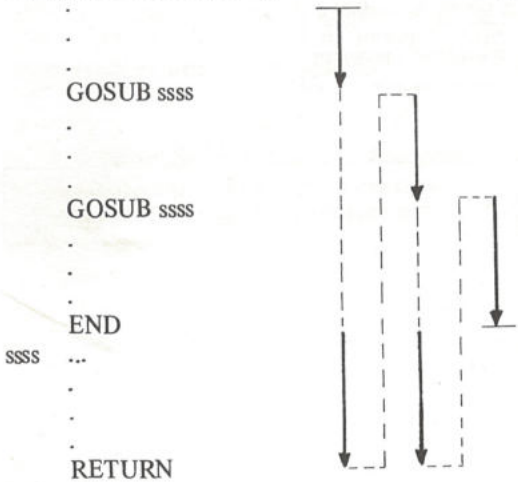
A RETURN jelentése: a legutóbb végrehajtott, még érvényes GOSUB utáni utasítással folytatódjon a programvégrehajtás.

Természetesen a GOSUB után szám helyett kifejezés is állhat, de ez nem kezdődhet számmal. Pl. GOSUB I+5 jó, GOSUB 5+I nem jó, GOSUB (5)+I jó.

A GOSUB után lista is lehet: GOSUB 5φφ, 7φφ, 6φφ azt jelenti, hogy az 5φφ-as, a 7φφ-as és a 6φφ-as soroktól kezdődő szubrutinokat kell sorban végrehajtani.

A szubrutinhasználat lehetővé teszi, hogy a programunk több pontján hajtsuk végre ugyanazt a tevékenységet anélkül, hogy az egészet többször kelljne

leírnunk. A végrehajtás ilyenkor a következőképpen folyik:



A GOSUB és a RETURN használatát a következő példa az elképzelhető legbonyolultabb eset kapcsán mutatja be:

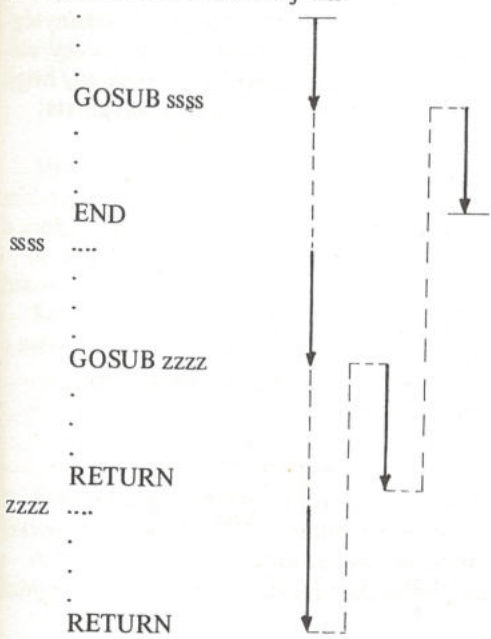
```
1 X=1
2 GOSUB 1ϕ
3 PRINT 4
4 END
1ϕ PRINT X
2ϕ IF X=2 THEN RETURN
3ϕ X=2
4ϕ GOSUB 1ϕ
5ϕ PRINT 3
6ϕ RETURN
```

A program először is az 1-es utasítást hajtja végre: X értéke 1 lesz. Ezután a 2-es utasítás következik: GOSUB-os vezérlésátadás a 1ϕ-esre. Ezután a 1ϕ-es következik: X (vagyis most az 1) kiírása. Ezután a 2ϕ-as következik: de X nem egyenlő 2-vel, így a sor további része nem érdekes. Ezután a 3ϕ-as következik: X értéke 2 lesz. Ezután a 4ϕ-es következik: GOSUB-os vezérlésátadás a 1ϕ-esre. Ezután a 1ϕ-es következik: X (vagyis most a 2) kiírása. Ezután a 2ϕ-as következik: a feltétel teljesül, tehát végrehajtódik a RETURN, azaz a visszatérés a legutóbbi érvényes GOSUB utáni utasításra. A legutóbbi érvényes GOSUB (amelyiknek a végrehajtása hatására a vezérlés a jelenlegi helyére került) a 4ϕ-es, tehát a RETURN hatására most ez után, az 5ϕ-es utasítással folytatódik a programvégrehajtás. Most tehát az 5ϕ-es sor következik: a 3-as szám kiírása. Ezután a 6ϕ-as következik: RETURN, azaz visszatérés a legutóbbi ér-

vényes GOSUB utánra. A legutóbb végrehajtott GOSUB a 4ϕ-es volt, de ez már nem érvényes, mert a hatására egy teljes, lezárt eljárást hajtottunk végre, vissza is tértünk. Az ezelőtti GOSUB a 2-es volt, amelyre még nem volt visszatérés, tehát ez még érvényes. A legújabb RETURN nem vonatkozhat másra, mint erre. Ezután tehát a 3-as sor következik: a 4-es szám kiírása. Ezután a 4-es következik: vége.

Az END hatására a programfutás leáll. Ilyenre eddig azért nem volt szükség, mert a programvégrehajtás vége úgyis a program fizikai végén volt. Az END szerepe az, hogy a programfutást bárhol meg tudja állítani.

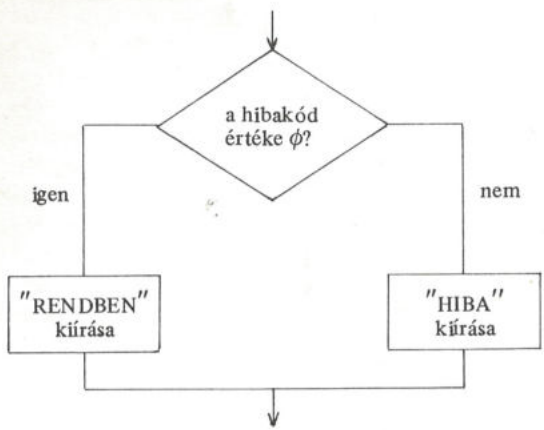
Reméljük, hogy a fenti rövid példa elegendő volt a GOSUB és a RETURN rugalmas felhasználási lehetőségeinek szemléltetésére – és arra is, hogy az Olvasó elborzadjon a GOSUB és a RETURN kusza használatától. Még a gyakorlott felhasználónak is azt tanácsoljuk, hogy a GOSUB-ot lehetőleg ne használja másra, mint egy-egy önálló funkciót megvalósító programrészletnek, modulnak a különválasztására és ennek a megfelelő helyen való aktivizálására. A GOSUB-bal elérhető trükkök sok lehetőséget nyújthatnak, de sok zavart is okozhatnak, ezért óvatosan bánjunk velük! Ez természetesen nem zárja ki azt, hogy GOSUB-bal aktivizált modulból újabb GOSUB-bal más, a hierarchia még alacsonyabb szintjén lévő modult aktivizáljunk és így tovább, majd ugyanezen a hívási láncon át visszatérjünk:



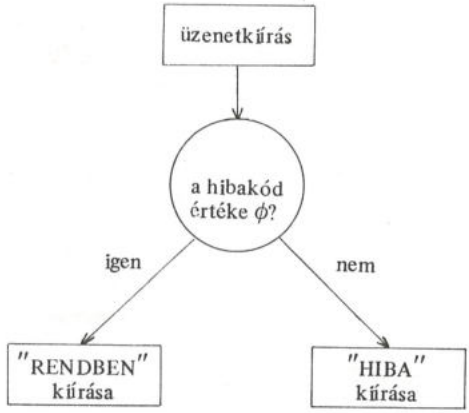
Súlyos bonyodalmat okozhat, ha két programegység között GOSUB-kapcsolat és GOTO-kapcsolat is van. A GOSUB-RETURN ugyanis úgy működik, hogy GOSUB hatására nemcsak a vezérlésátadás történik meg, hanem a visszatérési cím (a GOSUB-ot követő utasítás sorszáma) is megőrződik a memóriában. RETURN hatására a program a legutóbbi megőrzött címre tér vissza és ugyanakkor ezt a címet törli a nyilvántartásból. A GOSUB-oknak és RETURN-oknak tehát mindig párokat kell alkotniuk – de ezt a gép nem tudja ellenőrizni. Pontosabban: csak annyit tud ellenőrizni a gép, hogy nem akarunk-e RETURN-utasítást végrehajtani olyankor, amikor egyetlen érvényes GOSUB sincs, azaz amikor a visszatérési címek nyilvántartása üres.

Az AIRCOMP lehetővé teszi, hogy kiküszöböljük a hibák egy fajtáját: erre szolgál a POP kulcsszó. Ha egy GOSUB-bal aktivizált eljárásból GOTO-val lépünk vissza, a POP utasítás segítségével töröltethetjük a visszatérési címek nyilvántartásának legutóbb bekerült elemét, azaz olyan helyzetet állíthatunk elő, mintha RETURN segítségével történt volna a vezérlés visszaadása. (A POP használatára szemléletes példát mutat „Az AIRCOMP mint fejlesztőgép" című fejezet.) A POP használata azonban nem javítja ki a különböző vezérlési szerkezetek keveredésének hibáját, csak (a rendszerbe való belenyúlással) eltünteti a hiba következményét.

Most már, az IF és a GOTO segítségével, feltételes szerkezetű tevékenységcsoportot is programozni tudunk. Például: legyen a programunkban egy változó, amit hibakódnak használunk, s ha ennek az értéke ϕ, írjuk ki, hogy "RENDBEN", egyébként írjuk ki, hogy "HIBA". A végrehajtás folyamata:



Az üzenetkiíró eljárás szerkezete:



BASIC nyelven:

```
2φ1φ IF HK=φ THEN PRINT "RENDBEN"
2φ2φ HK<>φ THEN PRINT "HIBA"
```

Jelölések:	=	<	>	<=	=<
Jelentésük:	egyenlő,	kisebb,	nagyobb,	kisebb vagy egyenlő,	
	>=	=>		<>	><
	nagyobb vagy egyenlő,			nem egyenlő	

Mint látható, a jelpárral megjelölt három reláció esetében a két jelet fel szabad cserélni. Ha esetleg begépeléskor szóközt írunk közéjük, listázáskor ez a szóköz már nem jelenik meg.

A BASIC-ben nemcsak számok, hanem szövegek is nagyság szerint összehasonlíthatók.

Két szöveg közül az minősül kisebbnek, amelyik ABC-sorrendben megelőzi a másikat, tehát

- a rövidebb szöveg mindig kisebb a hosszabbnál,
- ha két szöveg első n karaktere azonos és az (n+1)-edik karakterük különböző, akkor az a szöveg kisebb, amelyiknek az (n+1)-edik karaktere az ABC-ben előbb áll.

Ezt felhasználhatjuk pl. arra, hogy tetszés szerinti sorrendben begépelt szavakat a géppel ABC-sorrendbe rendeztessünk.

A betűkön kívüli karakterek egymás közti és a betűkhöz képesti sorrendje a könyv végén lévő karakterkód-táblázatból látható.

A fentebbi második utasítást így is írhattuk volna:

```
1ϕ2ϕ IF HK THEN PRINT "HIBA"
```

Ez a két forma teljesen egyenértékű, de a második több, mint kétszer olyan gyorsan hajtódik végre, és még a program olvashatóságát is növeli a használata. Itt ugyanis nem a változót, hanem tulajdonképpen az eseményt jelöltük meg egy névvel. Ezt az utasítást nem úgy olvassuk, hogy „Ha a hibakód értéke nem egyenlő nullával", hanem úgy, hogy „Ha van hiba". Erre a megoldásra tehát akkor van mód, ha valamilyen esemény bekövetkezését úgy jelöljük egy változó különböző értékeivel, hogy nullától eltérő érték tartozzon ahhoz az eseményhez, amelyikre majd rá akarunk kérdezni.

A hibaüzenetek fenti programozási módja jó, de csak a mi speciális esetünkben, amikor mind a két tevékenység egy-egy sorban kifér és nincs is semmilyen akadálya annak, hogy egy-egy sorba írjuk őket. Most olyan példa következik, ahol a feltétel egyes ágaihoz tartozó tevékenységek tetszőlegesen bonyolultak lehetnek.

```
1ϕ1ϕ IF HK THEN GOTO 2ϕ1ϕ
1ϕ2ϕ PRINT "RENDBEN"
        .       egyéb tevékenységek, amiket
        .       akkor kell végrehajtani, ha
        .       a feltétel 2. ága teljesül
199ϕ GOTO 3ϕ1ϕ
2ϕ1ϕ PRINT "HIBA"
        .       egyéb tevékenységek, amiket
        .       akkor kell végrehajtani, ha
        .       a feltétel 1. ága teljesül
3ϕ1ϕ .
     .
     .
```

Persze megtehetjük azt is, hogy a bonyolultabb programrészletet is az egyszerűbb formában írjuk le:

```
1ϕ1ϕ IF HK=ϕ THEN GOSUB 1ϕ11ϕ
1ϕ2ϕ IF HK THEN GOSUB 1ϕ21ϕ
```

és a bonyolultabb utasítássorozatokat külön eljárások formájában adjuk meg, a program egy elkülönített helyén.

A fenti eredményt éri el az alábbi utasítás is:

```
1ϕ1ϕ IF HK THEN PRINT "HIBA" 'PRINT "RENDBEN"
```

Az IF-utasításban felsővesszővel jelölhetjük meg a két ág közti határt. A THEN kulcsszó és a felsővessző között írhatjuk le azt az utasítást (vagy kettősponttal tagolt utasítássorozatot), amelynek akkor kell végrehajtódnia, ha az IF és a THEN közti feltétel teljesül, majd a felsővessző után írhatjuk le

azt az utasítást (vagy kettősponttal tagolt utasítássorozatot), amelynek akkor kell végrehajtódnia, ha az IF és a THEN közti feltétel nem teljesül. Maga a THEN kulcsszó is helyettesíthető egy felsővesszővel; ilyenkor a felsővesszők a programágak elejét jelölik.

Itt hívjuk fel a figyelmet arra, hogy a gép számábrázolási pontossága véges, és a számolások szükségképpen kerekítésekkel járnak. Az X=3.1415927 : PRINT X parancssorozat hatására a képernyőn 3.14159 jelenik meg és így tovább. Sok számolás esetén a kerekítési hibák összegződhetnek is. Nemigen számíthatunk tehát arra, hogy két szám (főleg két nem-egész szám), ha különböző eljárásokkal számoltuk ki őket, egyenlő lehet.

Ha valamiféle hibakódot vagy más hasonló jelet vizsgálunk, amelyik csak egyszerű értékadással kaphatott értéket, nincs veszély. De ha például A és B egy-egy hosszabb számolás eredménye, akkor gyakorlatilag biztos, hogy az IF A=B feltétel soha nem teljesül, hiszen a különböző számolási eljárásokban a kerekítési hibák is különböznek. Javasoljuk, hogy IF A=B helyett (főleg, ha A és B nem egész) inkább olyan feltételt írjunk, ahol azt vizsgáljuk, vajon A és B különbsége A-hoz és B-hez képest eléggé kicsi-e. A könyv elején, az egyszerű tevékenységek közt megemlített függvényhasználati példa mintájára, az abszolútértéket képző ABS függvény felhasználásával például így oldhatjuk meg a vizsgálatot:

```
IF ABS(A-B)<ϕ.ϕϕ1 THEN ...
```

ha egy meghatározott hibakorlát alá akarunk eljutni, vagy

```
IF ABS(A-B)<ABS(A)/1ϕϕϕ THEN ...
```

ha A pillanatnyi értékéhez akarunk viszonyítani.

A feltételes utasítások alkalmazásának tekintjük az alábbi példát.

Döntsük el, hogy egy begépelt számpár (például 2 és 3ϕ) helyes dátum-e.

A dátum a naptárunk szerint akkor helyes,

- ha a hónap 1, 3, 5, 7, 8, 1ϕ vagy 12 és a nap legfeljebb 31,
- ha a hónap 4, 6, 9 vagy 11 és a nap legfeljebb 3ϕ,
- vagy ha a hónap 2 és – az egyszerűség kedvéért – a nap legfeljebb 29.

A programunk olvassa be a három számot és írja ki a "ROSSZ" vagy a "JO" üzenetet!

Hasonlítsuk össze a következő két megoldást!

```
1ϕϕ INPUT "HONAP=";HO
11ϕ INPUT "NAP=";N
12ϕ IF HO=2 THEN GOTO 3ϕϕ
13ϕ IF HO=4 OR HO=6 OR HO=9 OR HO=11 THEN GOTO 4ϕϕ
2ϕϕ IF N<=31 THEN PRINT "JO" : GOTO 5ϕϕ ' "31-NAPOS.
```

```
21ϕ PRINT "ROSSZ" : GOTO 5ϕϕ
3ϕϕ IF N<=29 THEN PRINT "JO" : GOTO 5ϕϕ ' "FEBRUÁR.
31ϕ PRINT "ROSSZ" : GOTO 5ϕϕ
4ϕϕ IF N<=3ϕ THEN PRINT "JO" : GOTO 5ϕϕ ' "30-NAPOS.
41ϕ PRINT "ROSSZ" : GOTO 5ϕϕ
5ϕϕ PRINT "VEGE"
51ϕ END

1ϕϕ INPUT "HONAP=";HO
11ϕ INPUT "NAP=";N
2ϕϕ MX=31 ' " A MAXIMUM.
21ϕ IF HO=2 THEN MX=29
22ϕ IF HO=4 OR HO=6 OR HO=9 OR HO=11 THEN MX=3ϕ
3ϕϕ IF N<=MX THEN PRINT "JO" : GOTO 4ϕϕ
31ϕ PRINT "ROSSZ"
4ϕϕ PRINT "VEGE"
41ϕ END
```

Az egymás után írt felsővessző és idézőjel után bármit írhatunk, az a programfutást nem befolyásolja. Így helyezhetünk el magyarázatokat a programjainkban.

Kulcsszavak:	OR	AND	NOT
Jelentésü:	vagy,	és,	nem

A kulcsszavak jelentése köznapi értelmükhöz hasonló:

– A OR B teljesül, ha A *vagy* B teljesül;
– A AND B teljesül, ha A *és* B teljesül;
– NOT A teljesül, ha A *nem* teljesül;

(A és B logikai értékek). A példából látható, hogy a logikai műveletekkel kifejezések is képezhetők (1. a 220-as sort). A kifejezésekben gép először a NOT, majd az OR műveletet végzi el, pl:

IF NOT A=B AND C=D OR E=F THEN ...

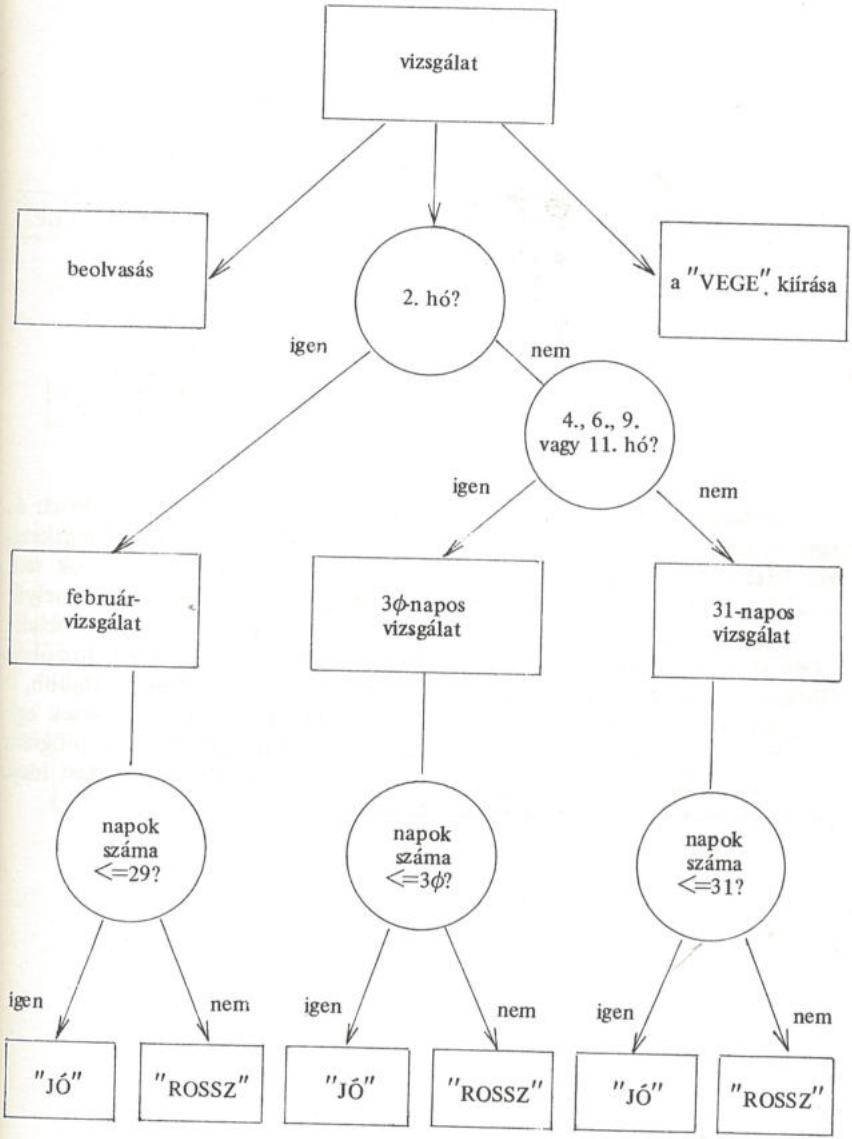
feltétel akkor teljesül, ha A=B nem igaz és ugyanakkor C=D igaz, vagy pedig ha E=F igaz, mintha tehát így lenne zárójelezve:

IF((NOT(A=B)AND B=C) OR E=F) THEN ...

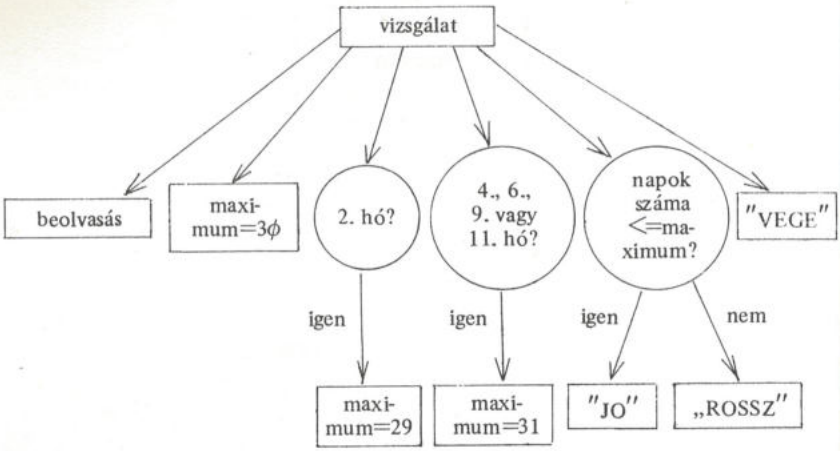
Zárójelzéssel a sorrend tetszőlegesen befolyásolható.

Megjegyezzük, hogy ha A,B,C stb. értéke szám, akkor a műveletek a számok egész részének 2 bájtos fixpontos kettes komplemens alakjai bitjeire vonatkoznak, tehát pl. 63 AND 16 értéke 16,4 OR 2 értéke 6, NOT 1 értéke –2.

Az első megoldás szerkezete:

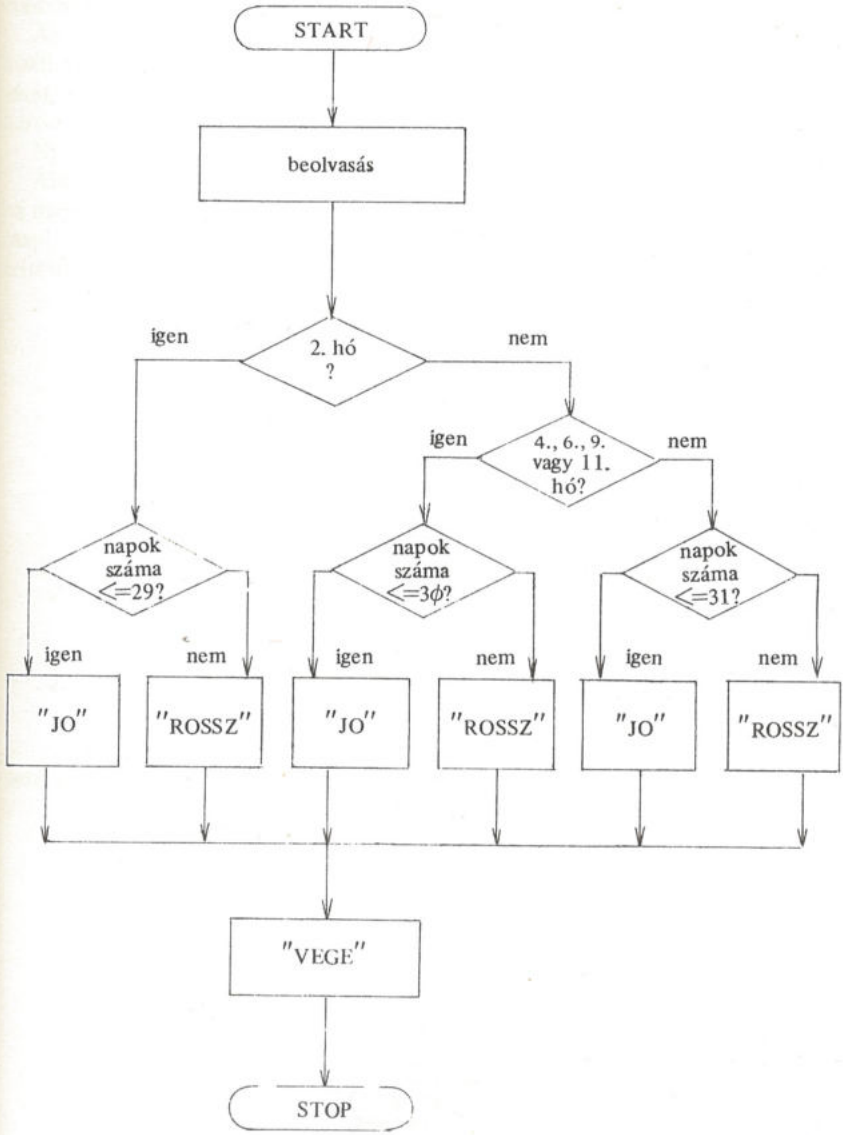


A másodiké:

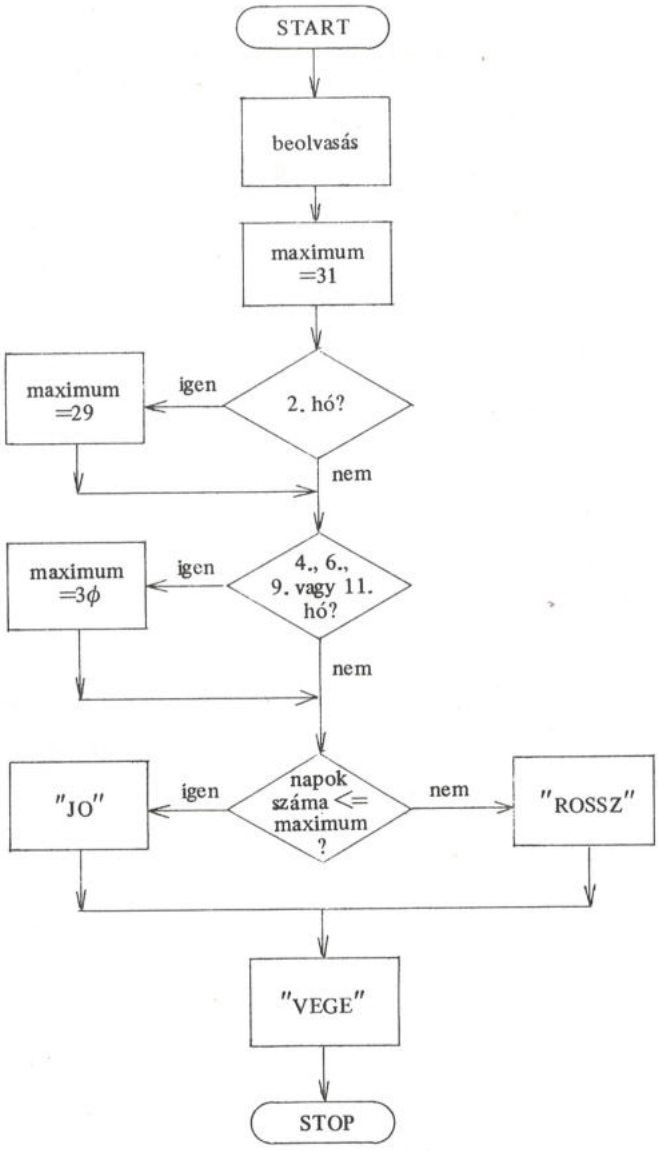


Vegyük észre, hogy a második megoldásban ha a hónap *nem* február és *nem* 3ϕ-napos, akkor a maximum értéke 31 marad, hiszen a soronkövetkező két feltétel egyike sem teljesül, tehát a hozzájuk tartozó értékadások sem hajtódnak végre. Ha pedig a hónap február vagy 3ϕ napos, akkor valamelyik későbbi értékadás végrehajtódik, MX új értéket kap, és az egyszer beleírt 31-es szám törlődik. Megszokott programozói fogás, hogy egy változóban elhelyezik azt az értéket, amelyiknek a bekövetkezése a legvalószínűbb, a leggyakoribb, és csak a többi esetre tesznek fel kérdést. Így az esetek egy részében egy értékadással több hajtódik végre, mint kellene, de a program áttekinthetőségét ez azért még nem zavarja, az értékadás végrehajtási ideje pedig törtrésze csak a feltételes értékadásének.

Az első megoldás végrehajtási folyamata:



A másiké:


START
beolvasás
maximum =31
2. hó?
igen
maximum =29
nem
4., 6., 9. vagy 11. hó?
igen
maximum =3ϕ
nem
napok száma <= maximum ?
igen
"JO"
nem
"ROSSZ"
"VEGE"
STOP

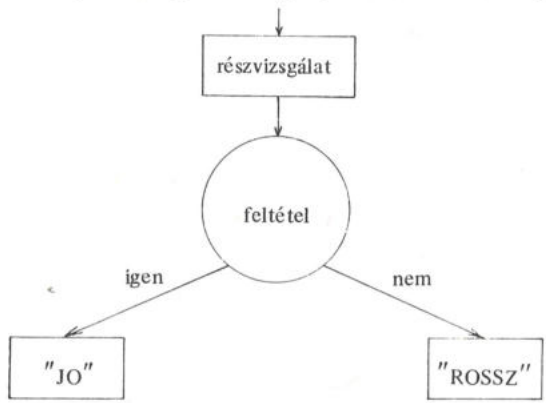
A két program pontosan ugyanazt a feladatot oldja meg, pontosan azonos eredménnyel. Különbség csak a megoldás menetében van.

Az első program pontról pontra követi a feladatleírást; a másik az eljárások közti válogatás helyett inkább adatbeállítással oldja meg az esetek szétválasztását, majd ugyanazzal az eljárással (a $3\phi\phi$-as utasítással) dolgozza fel mind a három esetet.

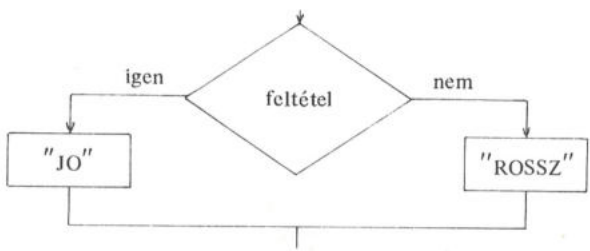
Nyilvánvaló, hogy a második típusú megoldás egyszerűbb, rövidebb.

Általában bátran megtehetjük, (és általában a programunk előnyére válik, ha megtesszük,) hogy az esetszétválasztásokat adatértékek megváltoztatásával, majd egységes eljárás alkalmazásával oldjuk meg – ha biztosak vagyunk két feltétel teljesülésében: hogy

– a különféle esetek valójában teljesen azonos szerkezetű megoldást illetve azonos felépítésű folyamatot igényelnek, mint a fenti példában a



szerkezet, illetve a



folyamat, ahol csak a feltételben és ezen belül is csak adat-értékekben volt különbség az esetek között; továbbá hogy

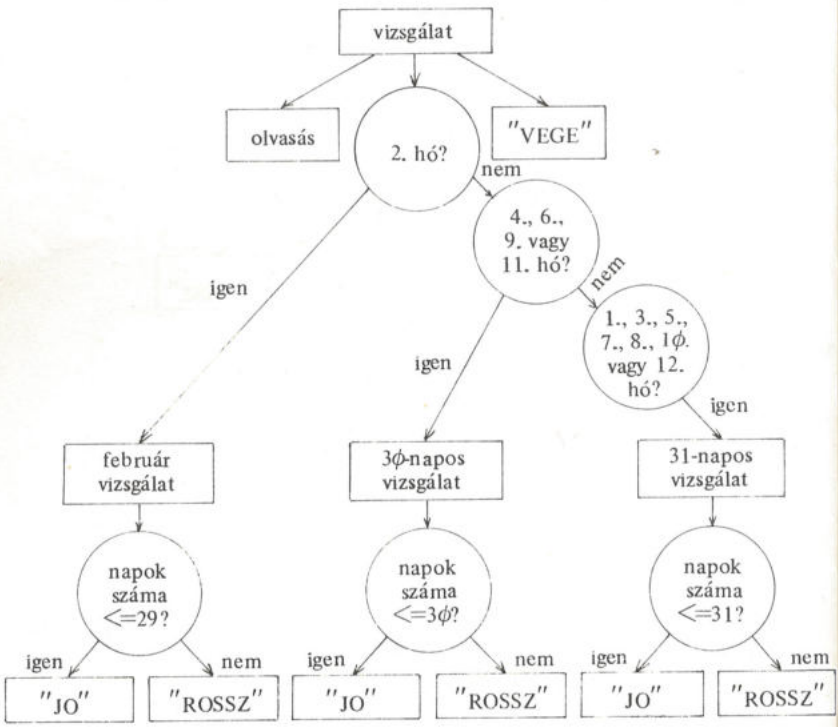
– ezen az azonosságon a feladat semmiféle várható módosulása nem változtat.

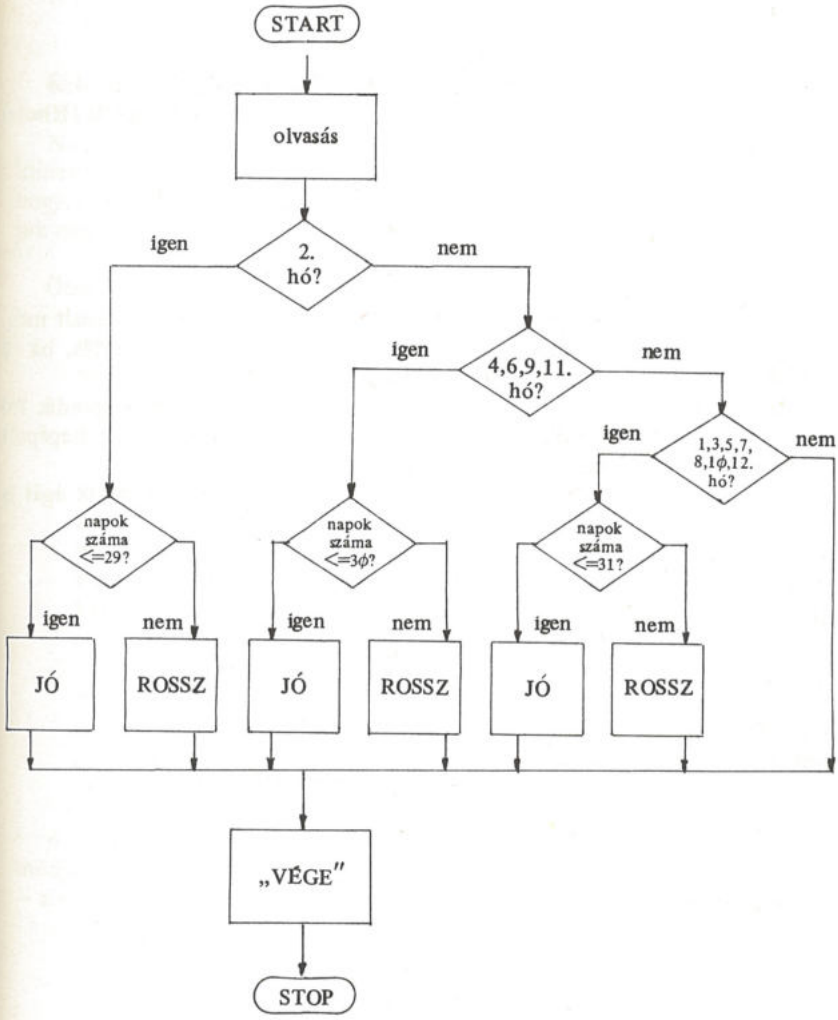
A fenti programjainkkal messze nem lehetünk elégedettek. Igaz, hogy jó adatbevitel esetén jól működnek, de vajon mi történik az általunk eddig figyelembe nem vett hibák esetében? Arra felkészültünk, hogy valaki második hónap 31-ét gépel be – de gépeljünk csak be harmincegyedik hó másodikát! Ezt a dátumot mindkét fenti program elfogadja és "JO"-nak minősíti.

A programtervezés, programírás *alapszabálya*, hogy a programnak a hibás adatbevitelre is fel kell készülnie. A hibás adatbevitel, elgépelés miatti rossz programfutás, téves eredmény nem az adatbevívő, hanem a program hibája! Nem kívánhatjuk meg, hogy az adatrögzítő, aki esetleg egyhuzamban többezer adatot gépel be, végig egyetlen hiba, elütés nélkül dolgozzon.

Az említett rossz válasz (harmincegyedik hó másodikának helyes dátumként való elfogadása) tehát programhiba. Annak a következménye, hogy a programban csupán *részleges* vizsgálatot végeztünk. Ha egy hónap-szám nem 2 és nem is 4, 6, 9 vagy 11, akkor csakis valamelyik 31-napos hónap helyes száma lehet – ez a téves meggondolás van beépítve mind a két programunkba.

Az első program esetében ezt a hibát így javíthatjuk ki:




START
olvasás
2. hó?
igen
nem
4,6,9,11. hó?
igen
nem
1,3,5,7, 8,1ϕ,12. hó?
igen
nem
napok száma <=29?
napok száma <=3ϕ?
napok száma <=31?
igen
nem
JÓ
ROSSZ
„VÉGE"
STOP

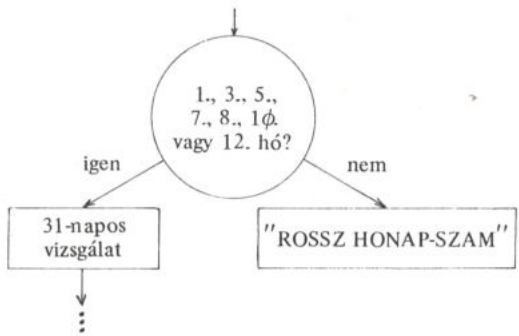
```
1ϕϕ INPUT "HONAP=";HO
11ϕ INPUT "NAP=";N
12ϕ IF HO=2 THEN GOTO 3ϕϕ
13ϕ IF HO=4 OR HO=6 OR HO=9 OR HO=11 THEN GOTO 4ϕϕ
14ϕ IF HO=1 OR HO=3 OR HO=5 OR HO=7 OR HO=8 OR HO=1ϕ
OR HO=12 THEN GOTO 2ϕϕ
15ϕ GOTO 5ϕϕ
      .
      .
      .
```

A program további része változatlan.

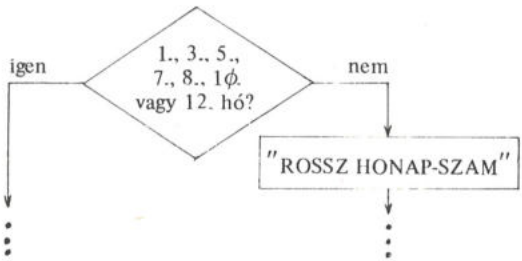
Ez a program a hónapok szempontjából *kimerítő* vizsgálatot valósít meg. Az egyes feldolgozó programágakra csakis akkor érkezhet a vezérlés, ha az ahhoz szükséges feltételek valóban teljesülnek.

Mi történik, ha ennek a programnak gépeljük be a harmincegyedik hó másodikát? A program kiírja a "VEGE" feliratot, semmi mást. A begépelt adathármast sem JO, sem ROSSZ dátumnak nem minősíti.

A programot tovább javíthatjuk, ha az eddig egyágú feltétel másik ágát is megrajzoljuk:



illetve a folyamatban:



A programban ez csak ennyi változást jelent:

15φ PRINT "ROSSZ HONAP-SZAM"

16φ GOTO 5φφ

Ezzel a változtatással a hónapok szempontjából már *teljes* vizsgálatot hajtunk végre.

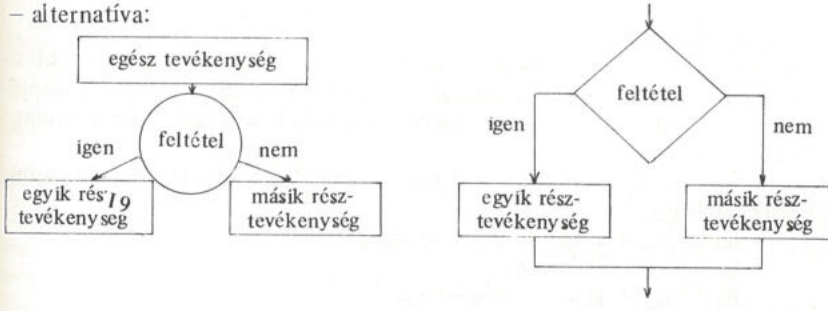
Nem állítjuk, hogy a teljes vizsgálat mindig jobb, mint a kimerítő, és a kimerítő mindig jobb, mint a részleges. Az adott konkrét feladat dönti el, hogy mikor melyiket érdemes használnunk. A fontos az, hogy mindig gondoljuk meg, melyiket alkalmazzuk és miért.

Összefoglalva tehát:

- részleges a vizsgálat, ha valahány lehetséges eset közül nem mindegyiket vizsgáljuk meg, mert feltételezzük, hogy ha a megvizsgált esetek egyike sem következett be, akkor csakis a még meg nem vizsgált eset valósulhatott meg. Például: ha egy számról kiderült, hogy nem pozitív és nem is nulla, akkor tudjuk, hogy csakis negatív lehet.
- kimerítő a vizsgálat, ha a kérdéseinkkel valamennyi lehetséges esetet kimerítjük, azaz minden feldolgozó eljárás csak akkor hajtódhat végre, ha a rá vonatkozó feltétel bizonyosan teljesül; de nem törődünk azzal, hogy mi történjék, ha nem valósul meg a vizsgált esetek egyike sem. Például ha a beolvasott szám egyenlő az eheti lottószámok valamelyikével, akkor növeljük a találat-számláló változó értékét, ha nem, akkor nem csinálunk semmit.
- teljes a vizsgálat, ha a program megfelelő módon reagál arra is, amikor egyik várt eset sem valósul meg. Például ha a beolvasott személyi szám első jegye 2 vagy 4, növeljük a nő-számláló értékét, ha 1 vagy 3, növeljük a férfi-számláló értékét, ha más, akkor hibajelzést írunk ki, mert hibás személyi szám érkezett.

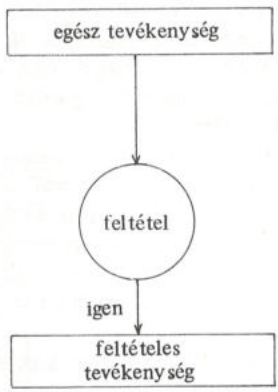
A feltételes szerkezetek legbiztosabb, mindig alkalmazható programozási módja a következő:

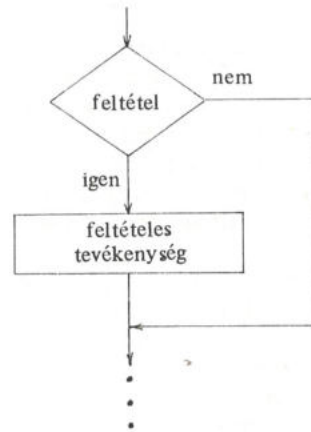
– alternatíva:



2ϕϕϕ IF feltétel THEN 3ϕϕϕ
. itt írjuk le
. a *másik* résztevékenységet
.
299ϕ GOTO 4ϕϕϕ
3ϕϕϕ . itt írjuk le
. az *egyik* résztevékenységet
.
4ϕϕϕ ... innen folytatódik a program

– feltétel:





2ϕϕϕ IF NOT feltétel THEN 3ϕϕϕ
. itt írjuk le
. a feltételes tevékenységet
.
3ϕϕϕ ... innen folytatódik a program

– esetszétválasztás:

Az esetszétválasztás (amikor egyazon feltételtől függően kettőnél több tevékenység között kell választanunk) a fentiek és a dátumos példa alapján könnyen megvalósítható. Az eset-szétválasztásra a könyv igen részletes mintapéldát is tartalmaz.

Van egy speciális esete, amikor még a vártnál is könnyebb a helyzetünk. Tegyük fel, hogy

– ha az I adat értéke 1, akkor az ezres,
– ha 2, akkor a kétezres,
– ha 3 vagy 4, akkor a háromezres

sorszámú utasítástól kell a végrehajtást folytatnunk. Az IF-ek helyett itt egyetlen ON-utasítást írhatunk:

1φφ ON I ' GOTO 1φφφ ' GOTO 2φφφ ' GOTO 3φφφ ' GOTO 3φφφ

Kulcsszó: ON
Jelentése: esetében

Az utasítás ezt jelenti: folytatódjon a programvégrehajtás azzal az utasítással illetve utasítássorozattal, amelyik az I-edik felsővessző után következik.

Összesen 255 felsővessző lehet egy ON-kulcsszó után. (Mivel az egész sor legfeljebb 1φφφ karakternyi lehet, ez nem jelent megkötést, mert 1φφφ karakteren belül úgysem igen lehetne a 255 programágnál többet elérni.)

Ha az ON után következő változó (vagy kifejezés) értéke φ, akkor a vezérlés átlépi az ON sorát. Ugyanez történik akkor is, ha a változó (vagy kifejezés) értéke több, mint a felsővesszők száma, (a mi példánkban 5 vagy több,) de nem nagyobb 255-nél. Ha viszont a változó (vagy kifejezés) értéke negatív vagy 255-nél nagyobb, akkor hibajelzést kapunk és a programfutás megszakad. Ha a változó (vagy kifejezés) értéke nem egész, akkor a gép abszolút értékben mindig lefelé kerekít: 1.999 még 1-nek számít, −φ.999 nullának.

Ha például az I változó értéke bizonyosan csak a 2, a 3 és a 6 értékek valamelyikét veszi fel, a közbülső számokra nem kell az ON-t kitöltenünk:

2φφ ON I '' GOTO 2φφφ ' GOTO 3φφφ ''' GOTO 6φφφ

Vigyázzunk ezzel a lehetőséggel, mert ha I mégiscsak felveszi a kimaradt értékek valamelyikét, akkor a programfutás hibajelzéssel megszakad.

Az ON utáni utasításlistában bármi lehet, csak újabb ON és felsővesszős IF nem.

Mind az IF, mind az ON rendelkezik egy különleges lehetőséggel: a különböző programágak közös kezdeteinek kiemelési lehetőségével. Az előbbi utasítás-példa így is írható:

1φφ ON I GOTO ' 1φφφ ' 2φφφ' 3φφφ' 3φφφ

hiszen a GOTO (mint kezdet) valamennyi programágban azonos.

Ugyanígy például a

2φφ IF A=B PRINT A, ''='' : X=X+1 ' PRINT A, ''NEM =''

utasítással egyenértékű a következő:

2φφ IF A=B PRINT A, ''='' : X=X+1 'A, '' NEM =''

Minden kiemelhető, csak értékadás nem.

Ez a lehetőség kényelmesebbé teszi a program begépelését, de a használatát csak akkor ajánljuk, ha biztos, hogy semmiféle majdani programmódosítás nem fogja azt igényelni, hogy új, másképp kezdődő programágat is beillesszünk – ekkor ugyanis visszamenőleg át kellene írnunk az egész utasítást.

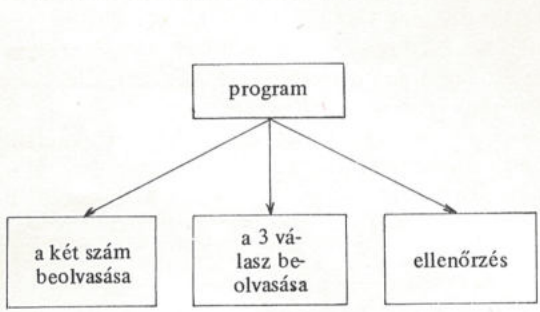
Eddig csak egyetlen feltételtől függő tevékenységekről volt szó, még ha ezek a feltételek bonyolultak is lehettek. Nem biztos azonban, hogy egy-egy összetett feltételt mindig így érdemes megvizsgálnunk. Itt volt például a dátumvizsgáló program: ha a hónap-szám hibás, akkor a nap-számot már felesleges is ellenőrizni. Elképzelhető azonban olyan feladat is, ahol egy adatsor minden egyes adatát mindenképpen ellenőriznünk kell, akár jó volt a többi, akár hibás.

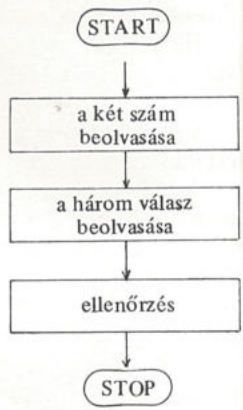
Írjunk egyszerű programot, amely gyerekek számolási készségét fejleszti és ellenőrzi! A program olvasson be két számot, majd kérdezze meg a két szám összegét, különbségét és szorzatát. A válaszokat ellenőrizze és írja ki azt, hogy helyesek-e.

Ezt a kűrást kétféleképpen képzelhetjük el:

- vagy minősítsen a gép minden választ külön-külön,
- vagy pedig csak egyetlen üzenetet írjon, jónak minősítve a választ, ha mindhárom jó, és hibásnak, ha van közöttük hibás.

A program váza egyöntetűen így fest:

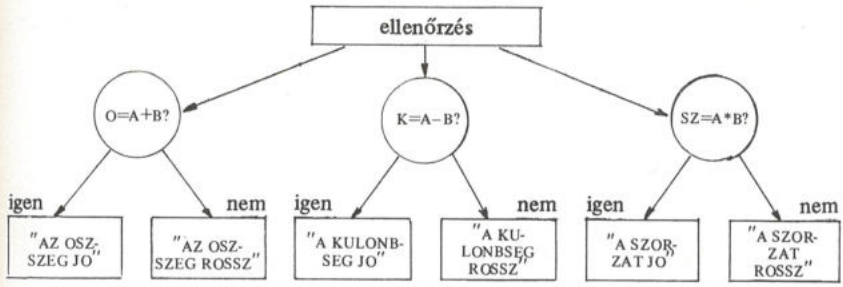


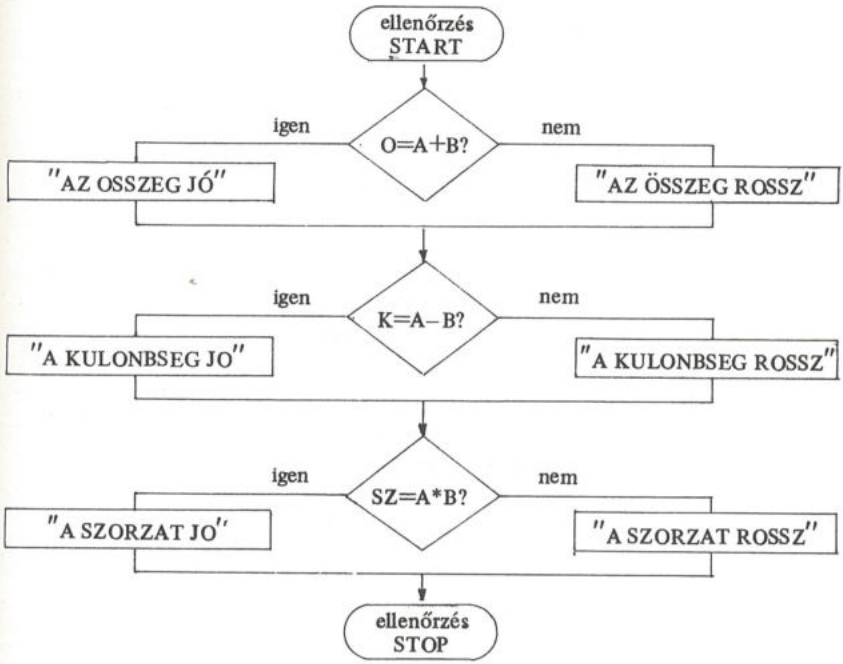



```
1ϕ INPUT "MI A KET SZAM? ";A,B
2ϕ INPUT "MENNYI AZ OSSZEGUK?"; O
3ϕ INPUT "MENNYI A KULONBSEGUK?"; K
4ϕ INPUT "MENNYI A  SZORZATUK?"; SZ
5ϕ GOSUB 1ϕϕ
6ϕ END
```

Különbség csak az ellenőrzésben van.

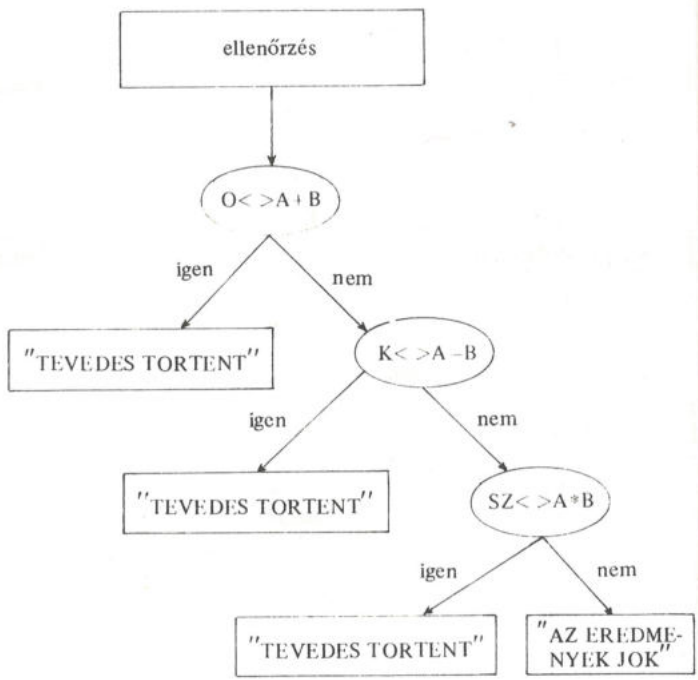
Az első esetben, válaszonként külön üzenettel:

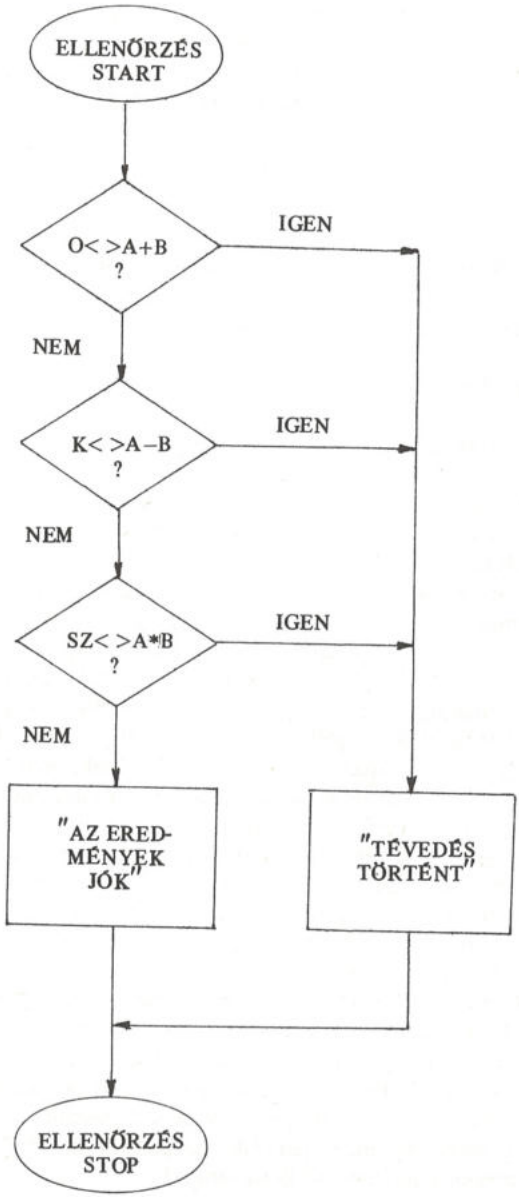




```
1ϕϕ IF O< >A+B THEN GOTO 15ϕ
11ϕ PRINT "AZ OSSZEG JO"
12ϕ GOTO 2ϕϕ
15ϕ PRINT "AZ OSSZEG ROSSZ"
2ϕϕ IF K< >A–B THEN GOTO 25ϕ
21ϕ PRINT "A KULONBSEG JO"
22ϕ GOTO 3ϕϕ
25ϕ PRINT "A KULONBSEG ROSSZ"
3ϕϕ IF SZ< >A*B THEN GOTO 35ϕ
31ϕ PRINT "A SZORZAT JO"
32ϕ GOTO 4ϕϕ
35ϕ PRINT "A SZORZAT ROSSZ"
4ϕϕ RETURN
```

A második esetben pedig, összesen egy üzenettel:




ELLENŐRZÉS START
O< >A+B ?
IGEN
NEM
K< >A–B ?
IGEN
NEM
SZ< >A*B ?
IGEN
NEM
"AZ EREDMÉNYEK JÓK"
"TÉVEDÉS TÖRTÉNT"
ELLENŐRZÉS STOP

```
1φφ IF O<>A+B THEN GOTO 2φφ
11φ IF K<>A–B THEN GOTO 2φφ
12φ IF SZ<>A*B THEN GOTO 2φφ
13φ PRINT "AZ EREDMENYEK JOK"
14φ GOTO 3φφ
2φφ PRINT "TEVEDES TORTENT"
3φφ RETURN
```

A kétféle feltételszerkezet közül az első *lineáris*, a második *kivezérelt* volt.

A hasonló feladatokat természetesen másképp is meg lehet oldani, például a legutóbbi programrészletben

```
14φ GOTO 3φφ
```

helyére írhattuk volna az

```
14φ RETURN
```

utasítást is és sok más részletben is eltérhetünk a bemutatott sémától. A séma előnye ugyanakkor, hogy egyetlen, jól felismerhető kezdete és vége van; ha változtatni kell, egyetlen blokként kivehető a programból és újraírható, anélkül, hogy ez a módosítás a program más helyein zavart okozna.

Erre a sémára is igaz tehát, ami a könyv többi sémáira: felépítésében figyelembe veszi a majdani könnyű módosíthatóság, javíthatóság szempontjait, s ezek érdekében inkább lemond néhány program-rövidítési lehetőségről.

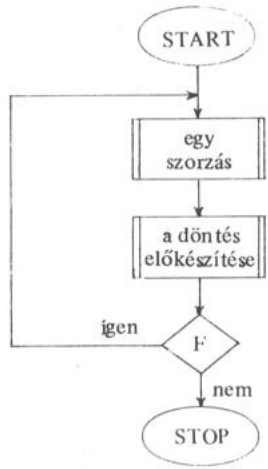
Külön felhívjuk a figyelmet a feltételek tagadó megfogalmazásaira. Általában megfigyelhető, hogy sokkal áttekinthetőbb az az ábra, az a program, ahol a feltételek igen-ágaiban már nincs újabb feltétel. Érdemes tehát a feltételeket úgy megfogalmazni, hogy ha teljesülnek, már csak egyféle dolgot kelljen tenni; ha nem teljesülnek, akkor még nyugodtan rátérhetünk a további vizsgálatokra. A példa kedvéért íme a legutóbbi program leírása a kétféle módon:

ahogyan megoldottuk: meg kell vizsgálni, O különbözik-e A+B-től; ha igen, hibajelzést kell adni, ha pedig nem, meg kell vizsgálni, K különbözik-e A–B-től; ha igen, hibajelzést kell adni, ha pedig nem, meg kell vizsgálni, SZ különbözik-e A*B-től; ha igen, hibajelzést kell adni, ha pedig nem, helyeslő üzenetet kell adni.

ahogyan célszerűtlen: meg kell vizsgálni, O egyenlő-e A+B-vel; ha igen, meg kell vizsgálni, K egyenlő-e A–B-vel; ha igen, meg kell vizsgálni, SZ egyenlő-e A*B-vel; ha igen, helyeslő üzenetet kell adni, ha pedig nem ... és most vajon mire vonatkozik ez a „ha pedig nem"? A legutóbbi feltételre vagy mindegyikre együtt? Csak a programunk primitívségének köszönhető, hogy ezek az esetek most ugyanazt jelentik, de gondoljuk végig, hogy más esetben milyen nehezen fejezhetnénk ki ugyanezt!

ISMÉTLŐDŐ TEVÉKENYSÉGEK

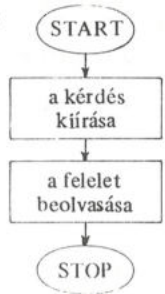
Az a szorzóprogramunk, amelyik minden szorzás után megkérdezte, kell-e folytatnia, ezt a folyamatot hajtotta végre:



Az F feltétel teljesül, ha a kérdésre adott válasz „IGEN".
A döntés előkészítése a válasz megkérdezését és beolvasását jelenti.

Egy szorzás:

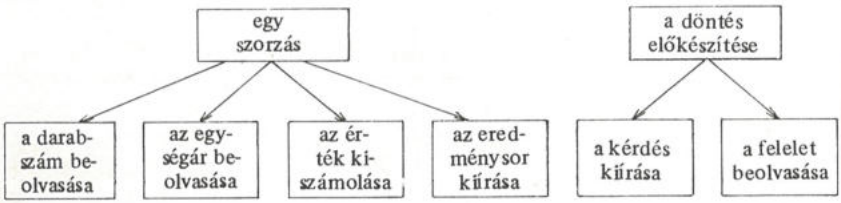
A döntés előkészítése



Technikai részletkérdés, hogy a kérdés kiírását és a felelet beolvasását egyetlen utasítással is meg tudtuk oldani.

A program szerkezete a következő:

ahol:

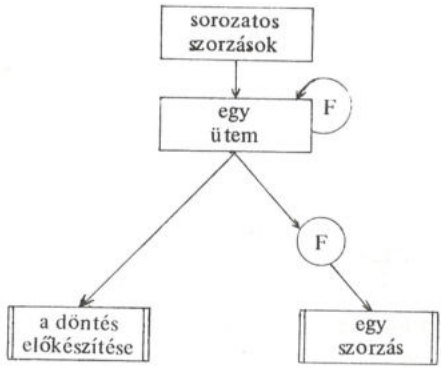


A tevékenységek ilyen ismétlődését *ciklusnak* nevezzük. A példában bemutatott ciklusnak az a jellegzetessége, hogy a folytatás feltételének a vizsgálata a lényegi eljárás, a *ciklusmag* után következik. Ez azt jelenti, hogy ha egyszer a program elindult, akkor *legalább egyszer* mindenképpen végrehajtja a ciklusmagot és csak utána vizsgálja meg, vajon végre kell-e ismét hajtania. Az ilyen szervezésű ciklust *hátultesztelőnek* nevezzük.

Ugyenez a feladat megoldható volna *elöltesztelő ciklus* segítségével is:

F jelentése, a döntés előkészítése és a szorzás ugyanaz, mint az előbb, hiszen nem a feladat változott, csak a műveletek szervezése. A programban is csak sorrendváltozások lesznek; legfeljebb még a kérdés szövege változhat meg.

Az új program szerkezete:



Maga a program:

```
1φ INPUT "KELL SZOROZNI? ";FS
2φ IF FS<>"IGEN" THEN GOTO 8φ
3φ INPUT "DARAB=";DB
4φ INPUT "EGYSEGAR=";AR
5φ KP=DB*AR
6φ PRINT DB;"*";AR;"=";KP
7φ GOTO 1φ
8φ END
```

Ebben az esetben tehát előfordulhat, hogy a program a ciklusmagot *egyszer sem* hajtja végre (ha már az első kérdésre igentől különböző választ kap).

Lehetett volna a 2φ-as utasítás IF FS<>"IGEN" THEN END is; mindig érdemes betartani azonban azt az íratlan szabályt, hogy a programnak (és lehetőleg minden önállóbb programegységnek is) ott legyen a logikai vége, ahol fizikailag is vége van.

Ugyanennek a szabálynak a betartását jelenti az is, hogy csak elől- és hátultesztelő ciklusról beszélünk, noha semmiféle elvi akadálya nincs annak, hogy egy ciklus középentesztelő legyen vagy hogy a döntéselőkészítést és magát a döntést két, egymástól különálló helyre írjuk. Egy-egy trükkösebb megoldás pillanatnyi előnyénél azonban többet ér, ha a programjainkat mindig olyan áttekinthető, egyszerű, sokszor begyakorolt, bombabiztos panelekből építjük fel, mint amilyen például az elől- és a hátultesztelő ciklus.

A ciklusszervezés gyakori speciális esete az, amikor valamilyen változó különböző, egyenletesen növekvő vagy csökkenő értékeivel kell például ugyanazt a műveletet elvégeznünk. Írjuk ki az első tíz pozitív egész köbét, először növekvő, majd csökkenő sorrendben!

Mindkét feladat annyira egyszerű, hogy sem folyamatábrát, sem struktúraábrát rajzolni nem érdemes.

```
1ϕ SZ=1
2ϕ PRINT SZ↑3
3ϕ SZ=SZ+1
4ϕ IF SZ<=1ϕ THEN GOTO 2ϕ
```

illetve

```
1ϕ SZ=1ϕ
2ϕ PRINT SZ↑3
3ϕ SZ=SZ−1
4ϕ IF SZ>=1 THEN GOTO 2ϕ
```

Ezekre az esetekre a BASIC külön utasításpárral rendelkezik.

A fenti két programocskát így is írhatjuk:

```
1ϕ FOR SZ=1 TO 1ϕ
2ϕ PRINT SZ↑3
3ϕ NEXT SZ
```

illetve

```
1ϕ FOR SZ=1ϕ TO 1 STEP −1
2ϕ PRINT SZ↑3
3ϕ NEXT SZ
```

Kulcsszavak:	FOR	TO	STEP	NEXT
Jelentésük:	-tól, -től,	óta, -ig,	lépés,	soronkövetkező

A FOR-utasítás formája

sorszám FOR ciklusváltozó=kezdőérték TO végérték STEP növekmény

vagy

sorszám FOR ciklusváltozó=kezdőérték TO végérték

A ciklusváltozó értéke minden végrehajtás *után* a növekménnyel (ha az utasítás rövid alakját használjuk, akkor eggyel) növekszik. FOR I=J TO K tehát ugyanazt jelenti, mint FOR I=J TO K STEP 1. Ha már *túljutott* a végértéken, akkor a ciklus nem hajtódik végre többször.

Így például a

```
FOR I=1 TO 1ϕ STEP 2
```

kezdetű ciklus 5-ször hajtódik végre, mialatt I értéke rendre 1, 3, 5, 7 és 9; a

```
FOR I=ϕ TO 1ϕ STEP .5
```

kezdetű ciklus 21-szer hajtódik végre, mialatt I értéke rendre ϕ, $\phi.5$, 1, 1.5, ... , 9.5 és 1ϕ; a

FOR I=3 TO −3 STEP −1

kezdetű ciklus 7-szer hajtódik végre, mialatt I értéke rendre 3, 2, 1, ϕ, −1, −2 és −3; a

FOR I=5 TO ϕ

kezdetű ciklus 1-szer hajtódik végre, (mert ez a ciklus hátultesztelő, tehát legalább egyszer akkor is végrehajtódik, ha a ciklusváltozó eleve túl van a végértéken,) mialatt I értéke 5.

A NEXT után nem kötelező a ciklusváltozó nevének a kiírása; ez inkább csak önellenőrzési célt szolgál, mert ha kiírjuk, akkor a gép ellenőrzi, valóban nem párosítottuk-e rosszul a FOR és a NEXT utasításokat.

Ismét felhívjuk a figyelmet a számok (főleg a nem-egész számok) számolási pontatlanságaira, kerekítési hibáira. A

FOR I=$1\phi\phi\phi\phi\phi\phi\phi$ TO $1\phi\phi\phi\phi\phi\phi\phi1$

kezdetű ciklus *végtelen ciklus,* mert a gép csekély számábrázolási képessége (7 számjegy) miatt a ciklusváltozó megnövelt értéke $1\phi\phi\phi\phi\phi\phi\phi+1$ kerekítve megintcsak $1\phi\phi\phi\phi\phi\phi\phi$ lesz. A

FOR I=ϕ TO 1 STEP $.\phi\phi\phi1$

kezdetű ciklus pedig nem $1\phi\phi\phi1$-szer, hanem csak $1\phi\phi\phi\phi$-szer hajtódik végre, mert I minden egyes növelése újabb és újabb kerekítési hibával jár. I értéke a ciklus utolsó végrehajtása után, ugyanezen okból, nem $1.\phi\phi\phi1$, hanem $1.\phi\phi\phi\phi5$ lesz.

A kezdőérték, a végérték és a növekmény természetesen változó vagy kifejezés is lehet. Az ilyen változóknak vagy magának a ciklusváltozónak a cikluson belüli megváltoztatása, bár szabályos és megengedett, a program átlátekinthetőségét és biztonságát rontja, ezért rendszerint nem célszerű.

A ciklus belsejéből a cikluson kívülre történő vezérlésátadás is szabályos, de nagy körültekintést igényel. Például: írjuk ki az első tíz pozitív egész köbét, de legfeljebb $5\phi\phi$-ig:

```
1φ FOR SZ=1 TO 1φ
15 IF SZ↑3>5φφ THEN GOTO 4φ ← Hibás megoldás
2φ PRINT SZ↑3
```

```
3ϕ NEXT SZ
4ϕ END
```

A helyes megoldás megtalálásához már ismerni kell a gépen belüli adatszervezés egy fontos részletét is. A visszatérési címeket (ahonnan a NEXT elérése után a végrehajtást folytatni kell) a gép a memóriában tárolja. Ha egy ciklus rendesen végetér, akkor törli a legutóbbi visszatérési címet. Ha viszont ezt a normális befejezést nem engedjük meg, (azaz ha átlépjük,) akkor magunknak kell a címtörlésről gondoskodnunk a POP utasítás segítségével:

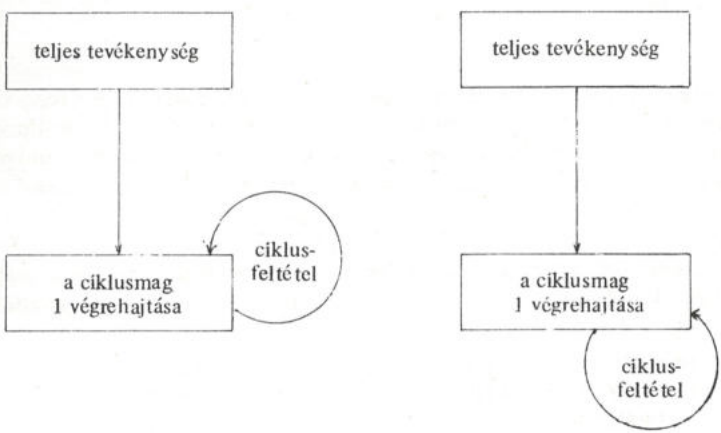
```
1ϕ FOR SZ=1 TO 1ϕ
15 IF SZ↑3>5ϕϕ THEN POP : GOTO 4ϕ ← Helyes megoldás
2ϕ PRINT SZ↑3
3ϕ NEXT SZ
4ϕ    itt folytatódhat a program akárhogyan,
      mert a cím-táblázatot már rendbetettük.
```

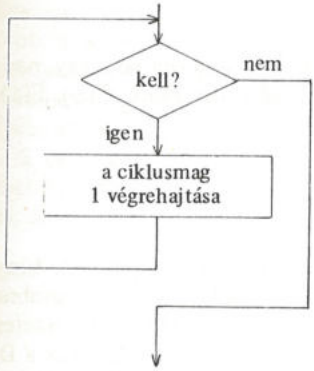
A ciklusból való ilyesfajta kilépéskor a ciklusváltozó értéke információt hordozhat. Például: ha végigolvassuk egy táblázat elemeit és hiba esetén elhagyjuk a ciklust, akkor a ciklusváltozó értéke azt mutathatja, hogy hányadik elem volt hibás.

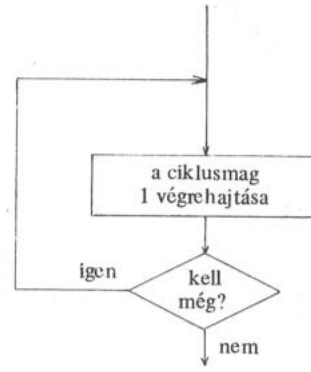
Összefoglalva a kétféle ciklusszervezési elvet:

Elöltesztelő ciklus:

Hátultesztelő ciklus:







kezdőértékek

IF ... THEN GOTO
ciklusmag
GOTO

kezdőértékek

ciklusmag
IF ... THEN GOTO

vagy:

FOR ...
ciklusmag
NEXT ...

Nincs feladat, amit az egyik fajta ciklussal meg lehetne oldani, de a másikkal nem. Bármelyiket alkalmazhatjuk, ha tudatosan, a tulajdonságait figyelembevéve alkalmazzuk.

Írjunk totószelvény-kiértékelő programot! A program olvassa be a heti eredményeket, majd olvassa be sorban a szelvények tippjeit és írja ki rendre, hogy melyik szelvényen hány találat van! (Az egyszerűség kedvéért ne vegyük figyelembe a mérkőzések elmaradásának lehetőségét. Vegyük figyelembe viszont azt, hogy a 13+1-edik tippet csak akkor kell számításba venni, ha az első 13 mind talált.)

Ennek a feladatnak a megoldása semmi elvi nehézséget nem okozhat, technikai nehézséget annál többet: tizennégy eredmény-adatot és szelvényenként tizennégy tipp-adatot kell páronként összehasonlítanunk. Ez szelvényenként tizennégy beolvasást és tizennégy összehasonlítást, vagyis tizennégy INPUT-utasítást és tizennégy IF-et jelent. Egyszerű ciklussal a program nem egyszerűsíthető, mert az összehasonlítást mindig más és más adatokkal kell elvégezni. Ezt a feladatot *tömbök* segítségével lehet egyszerűen megoldani.

A program elején elhelyezhető

1ϕ DIM E(13)

utasítás azt jelenti, hogy ebben a programban E nevű adatból nem egy, hanem éppen tizennégy van. A programban így hivatkozhatunk rájuk: E(ϕ), E(1) és így tovább (az E tömb nulladik, első stb. eleme).

Kulcsszó: DIM
Jelentése: kiterjedés

A DIM a dimension szó rövidítése. Segítségével azt írhatjuk elő, hogy a programunkban használni kívánt adatsorozatnak (adattömbnek, tömbnek) melyik legyen az utolsó eleme; a kezdőelem mindig a nulladik. Természetesen változó vagy kifejezés is szerepelhet a tömbné utáni zárójelben, akár a DIM utasításban, akár később, az egyes tömbelemekre való hivatkozáskor. Ha több tömböt kívánunk használni, ezeket akár külön-külön DIM-utasításokban, akár egyben is szerepeltethetjük:

25 DIM A(18), B(29), C$(33)

formában. Vigyázzunk, nehogy véletlenül egyazon programfutás során kétszer hajtsunk végre ugyanarra a tömbre vonatkozó DIM utasítást! Ez még akkor is hiba, ha a tömböt mindkétszer egyforma méretűre írjuk elő. Ennek elkerülésére jó, ha valamennyi DIM-utasításunkat a program elejére tesszük. Ez annál is indokoltabb, mert a DIM utasításnak mindenképpen előbb kell végrehajtódnia, mint ahogy a tömbre hivatkozunk.

Ha DIM utasítás nem volt a programban, és egy változónevet mégis úgy használunk, mintha tömbnév volna, például ha a programban egyszercsak megjelenik az

55 X(3)=ϕ

utasítás, akkor a BASIC úgy viselkedik, mintha lett volna egy

54 DIM X(1ϕ)

utasítás is, azaz a BASIC ilyenkor automatikusan előírja magának, hogy létezzen egy X nevű tömb összesen 11 elemmel (a nulladiktól a tizedikig).

Tömb nemcsak ilyen elemsorozat (vektor) lehet, hanem elemek táblázatszerű készlete (mátrix) is.

1 DIM TA(2ϕ,14)

egy kétdimenziós tömböt hoz létre, amely tömbnek 21 sora és soronként 15 oszlopa van, összesen TA(ϕ,ϕ)-tól TA(2ϕ,14)-ig tehát 315 eleme.

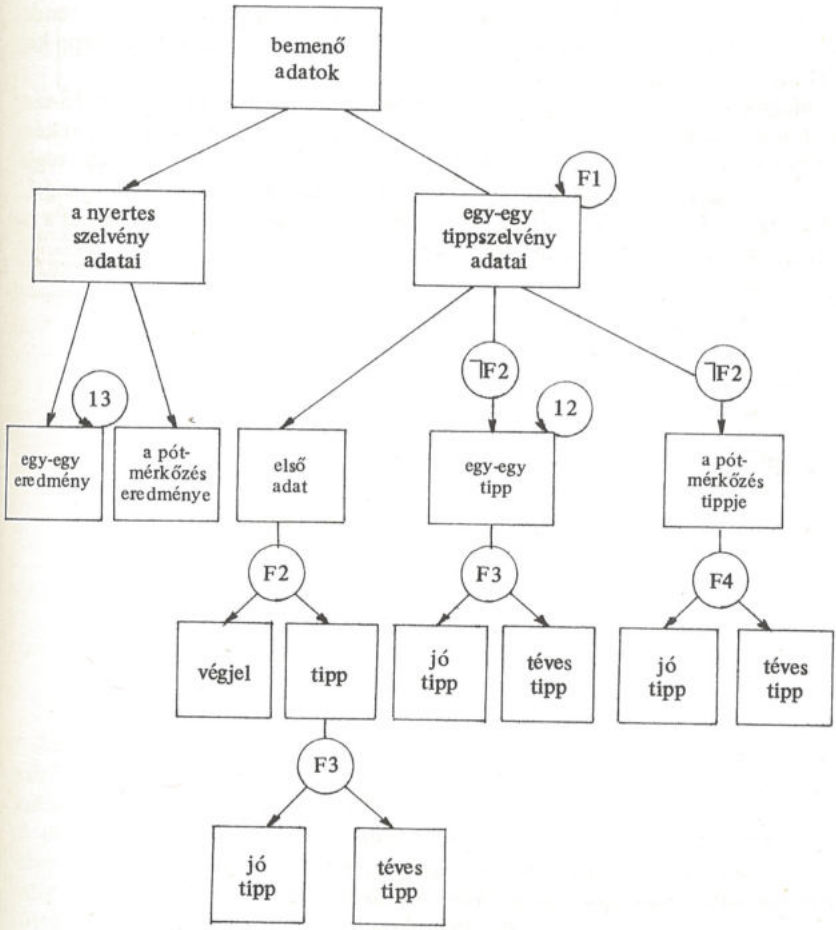
Tömbnév lehet bármilyen betű ill. kétbetűs kombináció (mint a változóké).

Tömbnévnek nyugodtan választhatunk olyan nevet, amilyen nevű nemtömb adat (skalár) változó már van a programban. Egy egydimenziós tömb neve azonban nem lehet azonos egy ugyanabban a programban szereplő két-

dimenziós tömbével. Például egyazon programban létezhet X változó és X(I) tömb, létezhet X változó és X(I,J) tömb, de X(I) tömb és X(I,J) tömb egyazon programon belül nem lehet.

Hadd szolgáljon ez a totószelvény-kiértékelő program arra, hogy a példáján lépésről lépésre mutassuk be a programtervezés és programírás menetét!

A bonyolultabb programok szerkezetét, végrehajtási folyamatát nemigen lehet egy lépésben kitalálni. Rajzoljuk fel először a program bemenő adatainak szerkezetét:



F1 jelentése: tippszelvények addig érkeznek, amíg valamilyen, megállapodás szerinti végjelet be nem gépelünk. Legyen ez a mi esetünkben kötőjel az első tipp helyén.

F2 jelentése: ha az első begépelt adat kötőjel.

¬F2 jelentése: ha F2 nem teljesült.

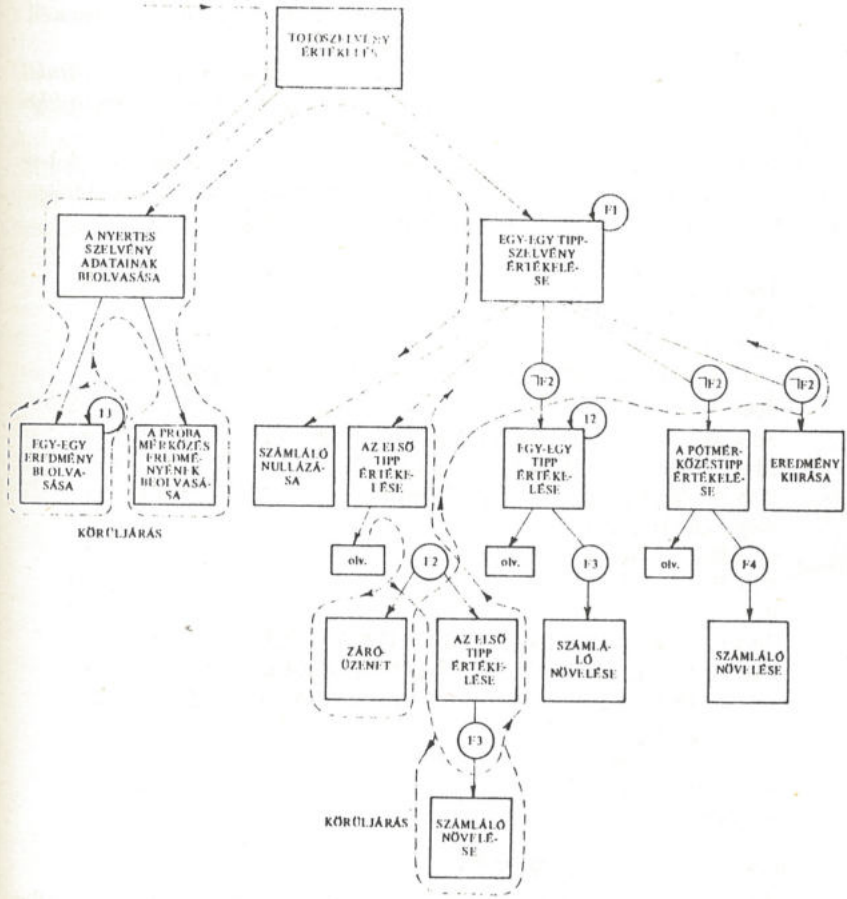
F3 jelentése: egy tipp akkor jó, ha megegyezik az ugyanannyiadik eredménnyel.

F4 jelentése: a pótmérkőzés-tipp akkor jó, ha megegyezik a pótmérkőzés-eredménnyel és minden tipp jó volt.

A rendes (nem pótmérkőzéshez tartozó) eredmények száma 13, a rendes (nem pótmérkőzéshez tartozó) tippek száma: az első tipp (amit másképp kell elbírálni, mert végjel is lehet) és még 12.

Magyarul: elől állnak a nyertes szelvény adatai, ezen belül először 13-szor 1–1 eredmény, majd a pótmérkőzés eredménye. Ezután tippszelvényenként rendre következnek az adatok, kezdve az első adattal, amelyik vagy végjel (kötőjel), vagy már maga is tipp. Ha az első adat tipp, akkor utána még 12 tipp majd egy pótmérkőzés-tipp is következik. Minden tipp kétféle lehet a mi feladatunk szempontjából: jó vagy téves.

A program szerkezete így fest:



Az adatszerkezet és a programszerkezet közti hasonlóság, reméljük, meggyőző. A gyakorlott programozónak is érdemes a programszerkezet megtervezése előtt az adatszerkezetet felrajzolnia. (Ha nem túl egyforma vagy nem túl primitív, akkor a bemenő adatok és a kimenő adatok szerkezetét különkülön is.) Ezt sokkal könnyebb megragadni, mint a programszerkezetet; ha pedig már felrajzoltuk, az esetek döntő többségében alig néhány átalakítással előttünk áll maga a kész, biztos programszerkezet.

Ez a módszer a könnyű javíthatóság, módosíthatóság titka is: ha a követelményekben valami módosul, ennek nyilván adat-kihatásai is vannak, amelyeknek a helye, jellege azonnal megmutatja, hogy a program melyik részein kell a módosítást átvezetni.

Figyelem! Hasonló programok leggyakoribb programozói hibája a számláló nullázásának kifelejtése! Enélkül pedig, ha az első szelvényen 7, a másodikon 8 találat van, a program a második szelvényt 15-találatosnak minősíti!

A programszerkezetből könnyen írhatjuk meg magát a programot. A bejelölt körüljárási vonal mentén kell a műveleteket leírni. (Ennek a körüljárásnak a pontos végiggondolása egyébként a tervünk jó ellenőrzése is.)

```
1φ DIM E$(13)
1φφ FOR I=1 TO 13
11φ INPUT (I);". EREDMENY=";E$(I)
12φ NEXT I
13φ INPUT "A POTMERKOZES EREDMENYE=";PE$
2φφ SZ=φ
3φφ INPUT "1. TIPP VAGY VEGJEL:";T$
31φ IF T$="-" THEN GOTO 8φφ
32φ IF T$=E$(1) THEN SZ=SZ+1
4φφ FOR I=2 TO 13
41φ INPUT (I);". TIPP=";T$
42φ IF T$=E$(I) THEN SZ=SZ+1
43φ NEXT I
5φφ INPUT "A POTMERKOZES TIPPJE=";T$
51φ IF SZ=13 AND T$=PE$ THEN SZ=SZ+1
6φφ PRINT "A SZELVENYEN";SZ;" TALALAT VAN."
61φ PRINT
7φφ GOTO 2φφ
8φφ PRINT "VEGE"
81φ END
```

Néhány megjegyzés a programhoz:

- E$ I-edik eleme az I-edik eredmény. A nulladik elem kihasználatlan maradt; ezzel viszont azt nyertük, hogy a programbeli sorszám mindig egyezik a szelvényen szereplő sorszámmal.
- (I) persze ugyanaz, mint I, (11φ. és 41φ. számú sor,) de ha I-t írtunk volna, a gép azt hinné, hogy azt is be kell olvasnia. Így viszont 11φ PRINT I; : INPUT ". EREDMENY=";E$(I) és 41φ PRINT I; : INPUT ". TIPP=";T$ hajtódik végre.
- ha T$ értékét megvizsgáltuk, többé nincs szükség rá. A következő beolvasáskor nyugodtan elfelejthetjük, ezért nem kell a tippek számára

külön tömböt szerveznünk. Az eredményeket viszont sokszor kell felhasználnunk, tehát mindegyik eredményt kénytelenek vagyunk megőrizni.

Minden magyarázat nélkül álljon itt példaként ugyanez lottószelvényre, ahol tehát lehet, hogy az első tipp az ötödik nyerőszámmal egyezik meg vagy fordítva, tehát minden tippet minden nyerőszámmal össze kell vetni, és az eredmények és a tippek is számok. A program leállításának módja legyen most az, hogy első tippként nullát gépelünk be.

```
1ϕ DIM E(5)
1ϕϕ FOR I=1 TO 5
11ϕ INPUT (I);". NYEROSZAM=";E(I)
12ϕ NEXT I
2ϕϕ SZ=ϕ
3ϕϕ INPUT "1. TIPP VAGY VEGJEL=";T
31ϕ IF T=ϕ THEN GOTO 7ϕϕ
32ϕ FOR I=1 TO 5
33ϕ IF T=E(I) THEN SZ=SZ+1
34ϕ NEXT I
4ϕϕ FOR J=2 TO 5
41ϕ INPUT (J);". TIPP=";T
42ϕ FOR I=1 TO 5
43ϕ IF T=E(I) THEN SZ=SZ+1
44ϕ NEXT I
45ϕ NEXT J
5ϕϕ PRINT "A SZELVENYEN";SZ;" TALALAT VAN."
51ϕ PRINT
6ϕϕ GOTO 2ϕϕ
7ϕϕ PRINT "VEGE"
71ϕ END
```

A program példát mutat egymásba skatulyázott ciklusokra (400–450 sz. utasítások). A cikluszáró utasítások (44ϕ–45ϕ-es sorok) helyett írhattuk volna ezt is:

```
44ϕ NEXT
45ϕ NEXT
```

vagy

```
44ϕ NEXT I,J
```

vagy a két NEXT-et egybeírva, de a változóneveket mégis kihagyva

44ϕ NEXT,

Vigyázzunk: ha egymást tartalmazó ciklusokban véletlenül ugyanazt a változót használjuk ciklusváltozónak (pl. ha most 42ϕ FOR J=1 TO 5-*öt írtunk volna), ezzel súlyos programhibát idézünk elő, amit futás közben általában igen nehéz megtalálni.*

TOVÁBBI LEHETŐSÉGEK

FÜGGVÉNYHASZNÁLAT

BELSŐ ADATOLVASÁS

FÜGGVÉNYHASZNÁLAT

A matematikában megszokott függvényjelölést a BASIC-ben is használhatjuk. Például az

113ϕ A=SIN(B)

utasítás hatására a gép kiszámolja B színuszát és ez lesz az A új értéke.

A használható numerikus függvények és jelentésük:

A=ABS(B)	B abszolútértéke. (Például A=ABS(5.3) hatására A értéke 5,3 lesz, A=ABS(−3.1) hatására A értéke 3,1 lesz.)
A=ATN(B)	B arkusz tangesének főértéke, radiánban. (Például A=ATN(−1) hatására A értéke −.785398 lesz, ami éppen $-\pi/4$.)
A=COS(B)	a radiánban megadott B koszínusza. (Például A=COS(3.14159/3) hatására A értéke .5 lesz.)
A=EXP(B)	e^B, azaz a természetes logaritmusalap B-edik hatványa. (Például A=EXP(1) hatására A értéke 2,71828 lesz.)
A=FRE(B)	B értékétől függetlenül a még szabad memódiakapacitás adja meg. (Például a frissen bekapcsolt gépen A=FRE(B) hatására A értéke 16189 lesz.) B értéke teljesen közömbös; csupán formai követelmény, hogy itt meg kell adni egy változónevet.
A=INT(B)	B egész része (a legnagyobb egészszám, ami B-nél nem nagyobb). Például A=INT(3) hatására A értéke 3 lesz, A=INT(3.14) hatására A értéke 3 lesz, A=INT(−1.9) hatására A értéke −2 lesz.)
A=LOG(B)	B természetes alapú logaritmusa. (Például A=LOG(2.71828) hatására A értéke .999999 lesz.)
A=PEEK(B)	a B-edik memóriacím tartalma. B-t kettes komplemens számmá alakítja át. Ha például az elérni kívánt cím hexadecimális 8$\phi\phi$F, akkor B értékének −32752-nek kell lennie. Ha B kisebb, mint 8192, azaz hex. 8$\phi\phi\phi$, akkor a megadott B-vel IN gépi utasítást hajt végre.

A=POINT(B,C)	a (B,C) képpont színe. (Lásd a grafikus lehetőségekről szóló részt.)
A=RND(B)	ha B nagyobb, mint ϕ, akkor véletlenszám a ϕ-tól B-ig terjedő intervallumban. (Például A=RND(1ϕ) hatására A értéke valamelyik ϕ és 1ϕ közti, nem feltétlenül egész szám lesz; lehet éppen ϕ vagy éppen 1ϕ is.) Ha B értéke ϕ, akkor A értéke az a szám lesz, amelyből a gép a soronkövetkező véletlenszámot elő fogja állítani. Ha B értéke negatív, akkor az a szám, amelyből a gép a soronkövetkező véletlenszámot elő fogja állítani, értékként felveszi B abszolútértékét. A véletlenszámok jellegzetes használatára „Az AIRCOMP-16 és a gyerekek" című fejezet mintaprogramja jól kidolgozott példát tartalmaz.
A=SGN(B)	B előjele: 1, ha B pozitív; −1, ha B negatív és ϕ, ha B nulla.
A=SIN(B)	a radiánban megadott B színusza.
A=SQR(B)	B négyzetgyöke.
A=TAN(B)	a radiánban megadott B tangense.
A=USR(B)	gépi kódú szubrutin aktivizálása. (Lásd a gépi kódú programozás lehetőségeiről szóló részt.)

A használható szövegfüggvények és jelentésük:

A=ASC(B$)	a B$ szöveg első karakterének kódja. (Például A=ASC("?") hatására A értéke 63 lesz.)
A$=CHR$(B,C,...)	a B,C,... kódú karakterekből álló szöveg. (Például A$=CHR$(97,61,81,84,89,65) hatására A$ értéke (gép által megjelenített kutya-figura) =QTYA lesz.)
A=LEN(B$)	a B$ szöveg hossza. (Például az A=LEN ("VALAMI") hatására A értéke 6 lesz, az A=LEN("") hatására A értéke ϕ lesz.)
A$=LFT$(B$,C)	a B$ szöveg baloldali C számú karaktere. (Például A$=LFT$("KUTYA",3) hatására A$ értéke KUT lesz.)
A$=MID$(B$,C,D)	a B$ szöveg C-edik karakterétől kezdődő D számú karakter. (Például az A$=MID$

("RUHAFOGAS",5,3) hatására A$ értéke FOG lesz.)

A$=RGH$(B$,C) a B$ szöveg jobboldali C számú karaktere. (Például A$=RGH$("KALAP",3) hatására A$ értéke LAP lesz.)

A$=STR$(B) a B kifejezés számértékének nyomtatási képe. (Például A$=STR$(15) hatására A$ értéke 15 lesz.)

A=VAL(B$) ha B$ egy matematikai kifejezés (vagy szám) szövegképe, akkor A a kifejezés értéke. (Például A=VAL("1.3*2") hatására A értéke 2,6 lesz.)

A programozó új, saját függvényeket is előállíthat saját használatára. A saját függvények neve az FN betűpárral kezdődik és még egy betűt tartalmaz. (Egy programban tehát annyi, a programozó által előírt új függvény lehet, ahány betűt az AIRCOMP-16 ismer, vagyis 26.) Az új függvény jelentését maga a programozó írhatja elő a program elején. Például a

3ϕ DEFFNU(A)=A↑3

utasítás azt eredményezi, hogy a végrehajtásától kezdve a programban az FNU betűcsoport, mint függvénynév, az utána zárójelben álló konstans, változó vagy kifejezés értékének a harmadik hatványra emelését jelenti. Z=FNU(9) végrehajtása után például Z új értéke 729 lesz.

Kulcsszavak: FN_ DEFFN_

Jelentésük: a _ nevű függvény, a _ nevű függvény meghatározása (mindkét esetben _ helyén bármilyen betű állhat)

Megismételjük: ahol a BASIC-ben konstans vagy változó állhat, ott állhat változókból és konstansokból összetett tetszőleges kifejezés is.

PRINT CHR$(ASC(MID$("MILESZ",SQR(3+INT(1.21)),1))+128)

hatására például fehér alapon fekete I betű jelenik meg a képernyőn, mert

INT(1.21) : értéke 1,

3+INT(1.21) : értéke 4,

```
                    SQR(3+INT(1.21))                    : értéke 2,
MIDS("MILESZ",SQR(3+INT(1.21)),1)                       : értéke 1,
ASC(MIDS("MILESZ",SQR(3+INT(1.21)),1)                   : értéke 73,
ASC(MIDS("MILESZ",SQR(3+INT(1.21)),1))+128              : értéke 2ϕ1,
CHRS(ASC(MIDS("MILESZ",SQR(3+INT(1.21)),1))+128         : értéke fehér
                                                          alapon feke-
                                                          te I betű.
```

A függvénydefiníció jobboldalán álló kifejezés a baloldalon megnevezett független változó mellett tartalmazhat egyéb, a programban szereplő változókat is. Pl:

```
DEFFNU(A)= A↑N
```

az A változót N-ik hatványra emeli, ahol N az FNU(A) fv. kiértékelésekor az N változó pillanatnyi értéke.

BELSŐ ADATOLVASÁS

Írjunk programot, amely beolvassa valakinek a havi jövedelmét és kiírja az illető szakszervezeti tagdíját! A programot többször is lehessen továbbindítani!

A program egyszerű; bonyodalmat csupán az okoz, hogy a jövedelmi kategóriák határai és a tagdíjösszegek teljesen kiszámíthatatlan sorozatát hogyan helyezzük el a programunkban.

A gondolatmenet a következő: a tagdíj havi összege 12ϕ Ft, de ha a jövedelem 1ϕϕϕ1 Ft alatti, akkor a tagdíj csak 1ϕϕ Ft, de ha a jövedelem 8ϕϕ1 Ft alatti, akkor a tagdíj csak 8ϕ Ft, és így tovább, ha a jövedelem az I-edik jövedelemhatárt nem haladja meg, akkor a tagdíj a tagdíj-sorozat I-edik eleme, végül ha a jövedelem 6ϕ1 Ft alatti, akkor a tagdíj csak 5 Ft. (Az egyszerűség kedvéért a nyugdíjasok tagdíjaival most nem foglalkozunk.)

Ennek a közvetlen programozása 16 IF-utasítást igényelne:

```
... TD=12ϕ
... IF J<=8ϕϕϕ THEN TD=8ϕ
... IF J<=7ϕϕϕ THEN TD=7ϕ
```

vagy egy-egy tömböt kellene felvennünk a jövedelemhatárok illetve a tagdíjak számára, és akkor egyetlen ciklusban megoldható a vizsgálat egy

```
... IF J<=JH(I) THEN TD=T(I)
```

utasítással; ekkor viszont 32 értékadásra van szükség:

```
... JH(ϕ)=1ϕϕϕ
... JH(1)=8ϕϕ
.
.
.
... JH(14)=9ϕϕ
... JH(15)=6ϕϕ
... T(ϕ)=1ϕϕ
... T(1)=8ϕ
.
.
.
... T(14)=7
... T(15)=5
```

A feladat legegyszerűbb megoldása a következő:

```
1ϕ  DATA  12ϕ,1ϕϕϕ,1ϕϕ,8ϕϕϕ,8ϕ,7ϕϕϕ,7ϕ,6ϕϕϕ,6ϕ,5ϕϕϕ,5ϕ,42ϕϕ,
      4ϕ,37ϕϕ,35,32ϕϕ,3ϕ,27ϕϕ,25,22ϕϕ,2ϕ,19ϕϕ,17,16ϕϕ,15,14ϕϕ,
      13,12ϕϕ,1ϕ,9ϕϕ,7,6ϕϕ,5
1ϕϕ INPUT "JOVEDELEM=";J
2ϕϕ RESTORE
21ϕ READ TD
3ϕϕ READ JH
31ϕ IF J>JH THEN GOTO 4ϕϕ
32ϕ READ TD
33ϕ IF JH=6ϕϕ THEN GOTO 4ϕϕ
34ϕ GOTO 3ϕϕ
4ϕϕ PRINT "JOVEDELEM=";J,"TAGDIJ";TD
41ϕ INPUT "KELL TOVABB SZAMOLNI? ";FŞ
42ϕ IF FŞ="IGEN" THEN GOTO 1ϕϕ
5ϕϕ PRINT "VEGE"
51ϕ END
```

Kulcsszavak:	DATA	RESTORE	READ
Jelentésük:	adatok,	állítsd vissza,	olvasd

A RESTORE kulcsszó a betűnkénti begépelésen kívül a RESTR gombbal állítható elő.

A program tárolja a jövedelemhatár-adatokat és a tagdíjadatokat. Beolvassa a jövedelmet. A RESTORE hatására a tárolt adatsorozat elejére áll, majd a tagdíj-értéket beállítja 12ϕ-ra. A READ mindig a soronkövetkező adatot veszi

elő a DATA-ban leírt sorozatból: most 1φφφφ lesz JH értéke. Ha a jövedelem ennél nagyobb, ki kell írni TD jelenlegi tartalmát, a 12φ-at, mert ennyi most a tagdíj. Ha viszont a jövedelem nem nagyobb a legutóbb beolvasott JH-értéknél, akkor be kell olvasni a következő DATA-adatot, amely a következő tagdíjösszeg, és kezdődhet az egész elölről, a következő jövedelemhatár beolvasásával. Ha viszont a legutóbbi jövedelemhatár 6φφ volt, akkor több nincs, tehát mindenképpen a kiírás következik. A TD változóban tehát minden pillanatban az a legutóbbi tagdíjérték van, amelyiknek a jövedelemhatára még nem volt kisebb, mint a beolvasott jövedelem.

Elvileg szebb lett volna a továbbfolytatásra vonatkozó kérdés után megvizsgálni, hogy a válasz az "IGEN" vagy "NEM" valamelyike-e, és ha nem, újból kérdezni – de a választott megoldás hibás felelet esetén legfeljebb leállítja a futást és újabb RUN-t kell kiadni, vagyis a hibás feleletnek semmi helyrehozhatatlan következménye nincs, csak kényelmetlenséget okozhat.

Az adatokat több DATA-utasításba is írhattuk volna:

```
1φ DATA 1φφφφ
11 DATA 1φφ
```

és így tovább vagy tetszőleges csoportonként. Ugyanígy egy READ is akárhány adatra vonatkozhat, (READ A,B,C,... formában,) amíg a programban lévő összes DATA együttes listája ki nem merül. Természetesen a DATA-ra is igaz, hogy a konstans adatok helyén változók és kifejezések is állhatnak. Bár nem szerepel a könyvben ilyen példa, de a DATA és a READ szöveges adatokat, változókat is tartalmazhat.

EGYÉB SZOLGÁLTATÁSOK

GÉPI PROGRAMOZÁS

KÉPERNYŐFRISSITÉS

HANGGENERÁLÁS

GRAFIKUS LEHETŐSÉGEK

MAGNETOFON HASZNÁLAT

GÉPI PROGRAMOZÁS

Az AIRCOMP-16 Z8ϕ-as mikroprocesszor-kódban programozható. Jelen könyvnek nem feladata a gépi kódú programozás módjának ismertetése, csupán a BASIC nyelvű és a gépi kódú programok kapcsolatának leírása.

A BASIC-terület felső határának címét a HM nevű változó tartalmazza. Ennek lejjebb állításával memóriaterületet zárhatunk el a BASIC elől, hogy ezáltal tárterületet biztosítsunk az adott esetben szükséges gépi kódú programok számára. A FRE függvény megadja a szabad (mármint a BASIC számára szabad) memóriaterület nagyságát.

Vigyázzunk: HM nem kaphat értéket INPUT *utasítással. Az* INPUT HM *parancs vagy utasítás nem olvassa be HM-et, hanem kiírja az aktuális értékét. Ha be akarjuk olvasni, közvetítő változót kell használnunk, például:* INPUT H : HM=H *formában.*

A memóriát a POKE kulcsszó segítségével tölthetjük fel. Poke A,B,C,... jelentése: az A-adik bájtra kerüljön B értéke, az A+1-edikre C értéke és így tovább. (Ha A kisebb, mint 8192, akkor OUT gépi utasítást hajt végre a gép.) A PEEK(B) függvény értéke a B-edik bájt tartalma. (Ha B kisebb, mint 8192, akkor IN gépi utasítást hajt végre a gép.)

A CALL A,B gépi kódú szubrutinhívást jelent az A,B (decimális) címre. Visszatérni a RET gépi utasítással lehet. Ha a vezérlésátadási címet (a POKE utasítással vagy másképp) a 16384. és 16385. címre tettük, akkor az A=USR(B) is gépi szubrutinhívást jelent. B értékének egész része a HL regiszterpárba kerül, aktívizálódik a gépi szubrutin, majd visszatéréskor a HL tartalma A-ba kerül.

A CALL után hosszabb címlistát is írhatunk, a címeket egymástól vesszővel elválasztva. Ilyenkor a felsorolt címeken lévő gépi szubrutinok a felsorolás sorrendjében sorra aktívizálódnak.

A CALL és az USR használatakor a CPU IX regisztere kivételével valamennyi regiszter használható.

Az AIRCOMP-16 monitora CALL 892 hatására jelentkezik be. A monitorszolgáltatások leírása meghaladja könyvünk hatáskörét.

A KÉPERNYŐFRISSÍTÉS VEZÉRLÉSE

A képernyőn legfeljebb 25 sor, soronként 4ϕ karakter jelenhet meg. Ez 2ϕϕx32ϕ képpontból áll. A képpont-sorok számát a DL nevű változó tartalmazza.

A képernyőtartalmat a gépnek pillanatonként fel kell frissítenie, ami természetesen időt vesz igénybe. Ha a programunkban az utolsó adat beolvasása után kiadjuk a DL=ϕ utasítást, (vagyis előírjuk, hogy ϕ sort kell felfrissíteni,) majd a feldolgozás végén a DL=2ϕϕ utasítást, (vagyis előírjuk, hogy ismét fel kell frissíteni valamennyi sort,) akkor a következőket tapasztaljuk:

- a DL=ϕ hatására a képernyő elsötétedik,
- a DL=2ϕϕ hatására a képernyő ismét kivilágosodik és ugyanazt láthatjuk, amit akkor láttunk volna, ha DL értékéhez nem nyúlunk,
- a kettő között lezajlott műveletek végrehajtási sebessége pedig mintegy három és félszeresére nő.

DL=ϕ hatására tehát a képernyőtartalom nem vész el, csak nem látszik, de DL visszaállítása után megint látható, a közben született kiírásokkal együtt.

Vigyázzunk: DL=ϕ esetén a kép INPUT utasítás végrehajtásakor, vagy az editor módban való visszatéréskor visszaáll, de a DL az értékét megtartja, az mindaddig nulla marad, amíg át nem állítjuk. Ez az egyetlen olyan eset, amikor DL értéke és a valójában látható kép mérete nem felel meg egymásnak.

Megtehetjük azt is, hogy DL értékét például 4ϕ-re állítjuk be, mire a képernyő tetején 5 sor megmarad, (mert 1 szöveges sor 8 képpont-sor,) ami elegendő ahhoz, hogy a programfutás eseményeit nyomonkövessük, de elég kicsi is ahhoz, hogy a programfutási sebesség jelentősen növekedjen.

Vigyázzunk: DL *nem kaphat értéket* INPUT *utasítással. Az* INPUT DL *parancs vagy utasítás nem olvassa be* DL-*t, hanem kürja az aktiális értéket. Ha be akarjuk olvasni, közvetítő változót kell használnunk, például:*

INPUT D : DL=D

formában.

HANGGENERÁLÁS

Az AIRCOMP-16 hanggenerálásra is képes. A

POKE 16384,132,29

parancs vagy utasítás kiadása után, amíg a 16384. és 16385. bájtokba más nem kerül, az

A=USR(256*B+C)

parancs vagy utasítás hatására hangot hallhatunk. (A biztonság kedvéért érdemes minden USR előtt újra és újra kiadni a POKE-ot.)

Az USR függvényben B határozza meg a hallható hang magasságát. B=ϕ esetén halljuk a létrehozható legmagasabb, B=255 esetén a létrehozható legmélyebb hangot. C értéke dönti el a hang hosszát. C=ϕ esetén halljuk a létrehozható legrövidebb, C=127 esetén a létrehozható leghosszabb hangot. (Körülbelül C=5ϕ az a legkisebb érték, amelytől felfelé a hang már jól hallható.)

A hallott hang recsegő, dudaszerű, mert a gép közben a képernyő állandó frissítgetésével is foglalkozik, így pillanatonként elhallgat. Ha azonban DL=ϕ paranccsal vagy utasítással a képernyő felfrissítését letiltjuk, akkor tiszta, elektromos síp-szerű hangot hallunk. (Igaz, hogy közben a kép eltűnik.) Az ilyen hang ugyanazon C-érték mellett természetesen sokkal rövidebb, mint ha van képernyőfrissítés.

A programutasítások végrehajtása annyira gyors, hogy a hangot nemigen szakítja meg érezhetően. Például a

DL=ϕ : FOR I=1 TO 5$\phi\phi$: A=USR(1$\phi\phi$*256+ϕ) : NEXT I

sor hosszabb, egybefüggő hangot szólaltat meg.

A hanggenerálás gyakorlati alkalmazására érdekes példát mutat be „Az AIRCOMP-16 és a gyerekek" című fejezet mintaprogramja.

GRAFIKUS LEHETŐSÉGEK

A képernyőterület képpontonként is hozzáférhető, így ábrákat is megjeleníthetünk. A GL nevű változó tartalmazza a képrajzolási célra rendelkezésre álló pontsorok számát. Amikor a gépet bekapcsoljuk, ez az érték ϕ. GL értéke soha nem lehet nagyobb, mint DL.

GL változtatásakor HM is változik, mert a grafikus kép előállításához is memória kell, amit a gép a BASIC elől vesz el. Erre nagyon vigyázzunk: ha GL értékét programon belül növeljük, a BASIC menetközben veszít el memóriaterületet, ami bajt okozhat az ott lévő adatok megsemmisülése miatt. GL csökkentésekor viszont a BASIC nem kapja az elveszett memóriaterületet vissza, csak ha a HM értékét magunk visszaállítjuk. A biztonságos eljárás tehát az, hogy a program elején állítsuk be GL értékét a programban előforduló legnagyobb értékre; így a BASIC az elvesző memóriaterületre eleve nem is ír adatot. Ezután már szükség szerint változtathatjuk GL-t.

Ha GL értékét növeljük, akkor a szöveges képernyő-rész feljebb csúszik és alatta megjelenik az a terület, amire rajzolhatunk. Ha GL értékét csökkentjük, akkor a szöveges képernyő-rész lejjebb csúszik és a tetejét eltakarja. Úgy működik, mintha egy teljes szöveges képernyő fel-le csúszkálna egy teljes

rajzos képernyőn, miközben mindkettőnek megmarad az éppen nem látható tartalma is.

Vigyázzunk: GL *nem kaphat értéket* INPUT *utasítással. Az* INPUT GL *parancs vagy utasítás nem olvassa be a* GL*-t, hanem kiírja az aktuális értékét. Ha be akarjuk olvasni, közvetítő változót kell használnunk, pl:* INPUT G : GL=G *formában.*

A rajzolás lényeges adata a szín-változó, a CR. (Nem azonos a CR gombbal!) Ennek értéke a gép bekapcsolásakor 1. Minden parancssor végrehajtása után és minden program lefutásakor automatikusan ismét felveszi az 1-es értéket, így biztosan számíthatunk arra, hogy az értéke 1 minden új parancssor végrehajtásának az elején és minden programindításkor is.

Vigyázzunk: CR nem kaphat értéket INPUT-utasítással. Az INPUT CR parancs vagy utasítás nem olvassa be CR-t, hanem kiírja az aktuális értékét. Ha be akarjuk olvasni, közvetítő változót kell használnunk, pl. INPUT C : CR=C *formában.*

A PLOT A,B utasítás vagy parancs hatására a B-edik sor A-adik képpontjának színe, bármi is volt eddig, ezentúl

- fehér lesz, ha CR értéke éppen páratlan,
- fekete lesz, ha CR értéke éppen páros.

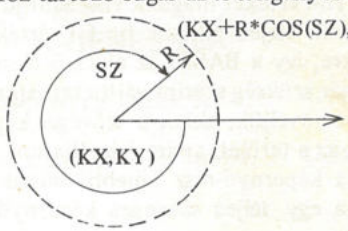
B felülről lefelé, A balról jobbra növekszik.

Természetesen B (vagy a helyére írt szám, kifejezés) nem lehet ϕ-nál kisebb vagy (GL-1)-nél nagyobb, A (vagy a helyére írt szám, kifejezés) értéke nem lehet ϕ-nál kisebb vagy 319-nél nagyobb. Vigyázzunk: ha ezt az utasítást egyetlen képpont esetében próbáljuk ki, valószínűleg semmit sem látunk, mert egyetlen magában álló képpont nagyon kicsi.

A CALL 7535 vagy PRINT CHR (5) utasítás vagy parancs hatására az egész aktuális képterület

- fehér lesz, ha CR értéke éppen páratlan,
- fekete lesz, ha CR értéke éppen páros.

Egyszerű példaként rajzoljuk ki fehér alapon fekete színnel azt a kört, amelynek középpontja az 5ϕ-edik sor 11ϕ-edik pontja, sugara pedig 3ϕ egység! Ha KX és KY a középpont x- illetve y-koordinátája és R a sugár, akkor a kör pontjainak koordinátái KX+R*COS(SZ) és KY+R*SIN(SZ), ahol SZ az a szög, amelyet az adott ponthoz tartozó sugár az x-tengellyel bezár:



Ha a kör valamennyi pontját meg akarjuk kapni, SZ-nek valamennyi lehetséges értéket fel kell vennie, ϕ-tól 2π-ig (mivel a BASIC-szögfüggvények a szög megadását radiánban igénylik).

Maga a program így fest:

```
1φ GL=2φφ
2φ CALL 7535
3φ CR=φ
1φφ KX=11φ: KY=5φ: R=3φ
2φφ FOR SZ=φ TO 2*3.14 STEP .φ1
21φ PLOT KX+R*COS(SZ),KY+R*SIN(SZ)
22φ NEXT SZ
```

A ciklus növekménye azért lett éppen ϕ,ϕ1, mert ez elég kicsi ahhoz, hogy a pontsorozatot összefüggő vonalnak lássuk, de elég nagy ahhoz, hogy a program viszonylag hamar lefusson. A szöveges képernyőt úgy állíthatjuk vissza, hogy begépeljük a GL=ϕ[CR] parancsot.

Az A=POINT(B,C) utasítás vagy parancs hatására A értéke

- 1 lesz, ha a C-edik sor B-edik pontja éppen fehér,
- ϕ lesz, ha a C-edik sor B-edik pontja éppen fekete.

Példaként álljon itt a könyv címlapábráját létrehozó mintaprogram, amelynek szerzője Lukács Endre (HOMELAB HARD–SOFT Elektronikai GMK):

```
5 PRINT" MINTAPROGRAM"
1φ PRINT CHR$(12)'"– A KEPERNYO TORLESE
12 FOR X=φ TO 23 : PRINT : NEXT
2φ HM=24φφφ'" LEGYEN SZABAD HELY A BASIC FELETT
22 POKE 16384,132,29'" EZ A HANGGENERALASHOZ KELL
3φ PRINT"FELRAJZOLOM AZ Y=A*EXP(B*R)*COS(C*R)"
4φ PRINT"FUGGVENYT; DIKTALJA BE A PARAMETEREKET!"
5φ PRINT"JAVASOLT ERTEKEK:" : PRINT ,"A=1φφ", "B=–4",
"C=12"
6φ INPUT"A=";A, "B=";B, "C=";C
7φ D=16φ : E=111 : F=39'" SEGEDVALTOZOK A RAJZOLASHOZ
72 P=.125 : L=.φφ9'" LEPESKOZOK
74 GL=φ : DL=7φ
8φ FOR X=HM TO HM+32φ : POKE X,φ : NEXT
82 DL=2φφ : GL=2φφ
9φ CR=φ : CALL 7535'" A GRAFIKUS MEZO TORLESE
92 CR=1
1φφ FOR Z=–1 TO 1 STEP P : FOR X=–1 TO 1 STEP L
11φ R=X*X+Z*Z : G=E*X+F*Z+D : Y=F*Z+F+A*EXP(B*R)*COS
(C*R)+4
```

```
12ϕ IF PEEK(HM+G)<Y THEN POKE HM+G,Y : PLOT G,D–Y
13ϕ NEXT ,
132 DL=ϕ
14ϕ C=1ϕϕϕ
142 A=2ϕ : B=5 : GOSUB C ' " EGYIK HANG
144 A=1ϕϕ : B=ϕ : GOSUB C ' " MASIK HANG
15ϕ DL=2ϕϕ
152 FOR X=ϕ TO C : NEXT ' " VARAKOZAS
154 END
1ϕϕϕ FOR X=ϕ TO B : FOR Z=ϕ TO A : F=USR(Z+1ϕ23) : NEXT ,
1ϕϕ2 RETURN
```

MAGNÓHASZNÁLAT

A kazettás magnóról, mint a géphez csatlakoztatható egységről, a gép általános leírásában már szóltunk. Itt csak arra térünk ki, hogy mire, és hogyan használható.

A magnó használatának első és legfontosabb feltétele, hogy a gép és a bekapcsolt magnó össze legyen kötve. Gyakorlott felhasználókkal is előfordult már, hogy a hosszasan keresett hibát csupán az okozta, hogy a csatlakozó nem volt bedugva.

Akkor is a hálózatról működtessük a magnót, ha egyébként elemmel is üzemeltethető lenne. A hálózat biztosabb, a TV-hez és a géphez pedig amúgyis kell.

A magnót a hálózatra kapcsolni még a gép bekapcsolása előtt célszerű. A csatlakoztatásnál ugyanis olyan tranziens jelenségek léphetnek fel (különösen ha ugyanabba az elosztóba csatlakoztatunk), amelyektől a rendszer elszáll. Súlyosbítja a helyzetet, hogy ez a jelenség nem rendszeresen, hanem véletlenszerűen jelentkezik.

A MAGNÓ ÉS A GÉP ÖSSZEHANGOLÁSA

Minthogy a különböző gyártmányú kazettás magnók szerkezetileg és minőségileg is különbözők, minden magnót külön össze kell hangolni a géppel.

A magnót akkor tudjuk csak használni, ha képes a gép által kiadott 300 mV erősségű jelek felvételére, és az így rögzített jelekből lejátszáskor legalább 500 mV erősségű kimenő jelet tud előállítani.

(a) A bemenő jel beállítása (felvételkor)

– Ha a magnónk felvételszabályozása nem automatikus, akkor a megfelelő

erősségű jel biztosítása érdekében célszerű a felvételeket enyhén túlvezérelni.
- Automata felvételű magnók esetén a megfelelő minőségű felvételről az automatika maga gondoskodik.

(b) A kimenő jel beállítása (lejátszáskor)
- Ha a magnónk átjátszó kimenete független a hangerőszabályozótól, nem kell tennünk semmit, feltéve, hogy a kimenet jelszintje elegendően nagy.
- Általában azonban a hangerő befolyásolja a kimenetet. Ilyenkor a magnót és a gépet feltétlenül össze kell hangolnunk, éspedig a következőkben leírt módon.

Az összehangoláshoz használjuk a géphez adott mintaprogram kazettáját. Ezen mindig valamilyen ismert (mondjuk ALFA) nevű program van. Kapcsoljuk be a magnót és a gépet, és kössük őket össze. Ezután

(1) Csévéljük vissza a kazettát az elejére,
(2) adjunk ki egy LOAD "? " parancsot,
(3) nyomjuk meg a CR gombot és indítsuk el a magnót lejátszásra.

Ekkor a képernyő elsötétedik, és amikor a magnó eléri a kazettára rögzített programot, hangos és kellemetlen fütty hallatszik.

Ha a hangerő jól van beállítva, akkor néhány másodperc múlva a kép újra megjelenik, és rajta a szalagon lévő program neve olvasható. Esetünkben tehát ALFA.

Ha a hangerő nincs jól beállítva, akkor ez a név hibás. Például: A2–%. Még olyan karaktereket is tartalmazhat, amelyek billentyűzetről be sem vihetők. Előfordulhat, hogy a kép vissza sem jön. Ilyenkor 3–4 másodpercig érdemes várni, majd a rendszert újra kell indítani. Ezt a legegyszerűbben úgy tehetjük meg, hogy a hálózati kapcsolóval a tápegységet először ki-, majd ismét bekapcsoljuk.

Akár sikeres, akár sikertelen beolvasás után a magnót állítsuk le, mert erről a gép nem gondoskodik!

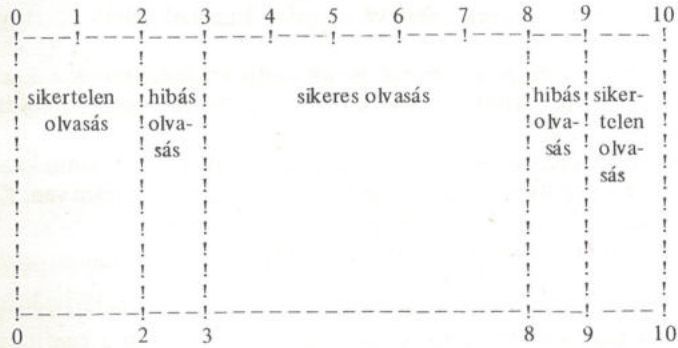
A gép és a magnó akkor van összehangolva, ha ismerjük azt a hangerőtartományt, amelyen belül a beolvasás teljes biztonsággal sikeresen végrehajtható.

Ezért függetlenül az első kísérletünk eredményétől, a fenti három lépést a hangerőskála minden fokozatára hajtsuk végre, mondjuk alulról fölfelé haladva.

Azt fogjuk tapasztalni, hogy kezdetben a gép képtelen beolvasni a programot. Majd elérünk egy olyan pontot, ahol már be tudja olvasni, de hibásan. Tovább haladva lesz egy olyan pont, ahol a beolvasás sikeres. Ezt jelöljük meg. Majd menjünk tovább. Egy darabig sikeres beolvasások következnek, de

el fogunk érni egy pontot, ahol a beolvasás ismét hibás lesz. Ezt is jelöljük meg. Mégtovább haladva eljutunk egy olyan hangerőig, amelyen a beolvasás már lehetetlen. A helyes beállítás a két megjelölt pont között van, mondjuk félúton.

Például az alábbi képzeletbeli hangerőskálán:



a helyes beállítás 5 és 6 között van.

A beszabályozás után ellenőrzésképpen olvastassuk be a teljes programot. Ilyenkor ugyanúgy az (1)–(2)–(3) lépéseket kell végrehajtanunk, mint a beállításkor, de azzal a különbséggel, hogy itt a (2) lépésben

LOAD "ALFA"

parancsot kell adnunk. (Feltéve persze, hogy ez volt a program azonosítója a szalagon.)

A kép ugyanúgy elsötétedik, mint a beállításkor, a sípoló hang is ugyanúgy hallatszik, de a kép csak a program beolvasása után, a sípolás megszűntekor jelenik meg.

Ne felejtsük el leállítani a magnót!

Ha a képernyőn az OK üzenet olvasható, a beolvasás rendben megtörtént.

Hiba esetén vagy ERROR üzenetet kapunk, vagy a BASIC újra bejelentkezik. Ilyenkor ismételten meg kell próbálni a beolvasást. Ha ez többszöri kísérletre sem sikerül, akkor a beállítást újra végre kell hajtani.

Nem érdemes kísérletezni, ha a sikertelenséget nyilvánvaló szalaghiba okozza. Például ha látjuk, hogy a szalag gyűrődött, vagy halljuk, hogy nyávog, el-elhalkul, stb. Ilyenkor ezeket a programokat többé már nem lehet beolvasni.

Megjegyzések a beállításhoz.

– Elvileg bármilyen készülék használható, de ne várjunk megbízható adatrögzítést a kopott, használt, régi magnóktól, és az agyonnyúzott kazettáktól.

– Automata felvételű magnók esetén előfordulhat, hogy egészen rövid (1 soros) programok még beolvashatók, de hosszabbak már nem. Ennek oka, hogy a felvételek eleje lényegesen hangosabb. A problémát csak szakember tudja megoldani, a felvételi automatika kikötésével.

– A beállításhoz nem feltétlenül kell a LOAD "?" parancsot használni, minden LOAD "XXXX" megfelel, amelyben olyan programnév szerepel, amely biztosan nincs a beállításhoz használt szalagon. A "?" azért jó, mert rövid, és semmilyen gyári programnév nem kezdődik kérdőjellel. Ha minden felhasználó betartja ezt a konvenciót, nem is kell ismernünk a próbakazetta tartalmát.

– A gép a LOAD parancsban megadott szövegkonstanst összehasonlítja a szalagról beolvasott névvel, és ha nem egyeznek meg, a rekordot nem olvassa be. Ezért gyorsabb a beállítás, ha nemlétező nevet, például "?"-et használunk.

– Ne lepődjünk meg, ha egyes magnókon a hangerő beszabályozásakor a sikertelen-hibás-sikeres-hibás-sikertelen olvasási fázisok valamelyike nem jelenik meg. Például a szerzők által használt magnók egyikén a skála feléig nem lehetett beolvasni, attól kezdve viszont végig sikeres volt minden beolvasás.

– *Figyelem! A hangerőbeállítást nemcsak új magnó, hanem új típusú kazetta esetén is érdemes újra elvégezni. Ugyanis a kazetták sem egyformák. Egy típuson belül azonban nincsenek lényeges eltérések.*

A MAGNÓ HASZNÁLATA PROGRAMOK TÁROLÁSÁRA.

Mindenekelőtt felhívjuk a figyelmet arra, hogy a szokásos AIRCOMP-16 konfigurációkban ez a magnó elsődleges szerepe.

A továbbiakban feltételezzük, hogy a magnó üzemkész állapotban van, a géppel összekötöttük, és az előbb ismertetett behangolása megtörtént.

(a) Programok kimentése kazettára.

Ha a képernyőre begépeltünk egy programot, az a gép kikapcsolásakor elvész. Ha később is használni szeretnénk, a programot ki kell mentenünk egy kazettára. Ehhez az alábbi lépéseket kell végrehajtani:

(1) Helyezzük el azt a kazettát a magnóban, amelyre a programot fel kívánjuk venni.

(2) Előre vagy hátracsévéléssel állítsuk be a szalagot arra a pontra, ahol a program rögzítését el akarjuk kezdeni.

(3) Adjunk ki egy SAVE "programnév" parancsot, de a CR gombot ne nyomjuk meg.

(4)Indítsuk el a magnót felvételre.

(5)Nyomjuk meg a CR gombot.

Ekkor a képernyő elsötétedik, és a gép halk fütyülő hangot ad. (Ezt a hangot a magnó felveszi, a CR gomb megnyomásával járó hanggal együtt.) A képernyő mindaddig sötét marad, és a hangjelenség mindaddig tart, amíg az átvitel be nem fejeződik, amikoris a hang megszűnik, és a kép visszaáll. Ekkor leállíthatjuk a magnót.

Ha a képernyőn nem jelenik meg az OK üzenet, a kimentés biztosan sikertelen volt. Ha viszont OK üzenetet kapunk, az csak azt jelenti, hogy a gép az adatátvitelt befejezte, de ez az OK nem vonatkozik az átvitel minőségére. Például megkapjuk az OK üzenetet akkor is, ha a magnót be se kapcsoltuk. A gép semmilyen információt nem ad arra vonatkozóan, hogy a program hibátlanul vagy hibásan került fel a szalagra, vagy hogy felkerült-e egyáltalán.

Erről csak úgy tudunk meggyőződni, hogy a programot a kazettáról visszaolvassuk, és kilistázzuk vagy lefuttatjuk. Minthogy a visszaolvasás a képernyőre begépelt programot megsemmisíti, a programunkat mindig több példányban mentsük ki. Így elkerülhető, hogy a teljes programot újra be kelljen gépelnünk.

Sorozatos sikertelenség esetén elsősorban a magnó, a kazetta, vagy azok beállításának hibájára gyanakodjunk.

Vigyázat: Bármi is történjék, soha ne kapcsoljuk ki a gépet, mert a begépelt program elvész. Ehelyett használjuk a (korábban ismertetett) RESET *gombot.*

Hosszabb programokat begépelés közben többször is mentsünk ki szalagra. Ugyanis hardver hiba, rendszerelszállás, hálózati hiba (fáziskiesés, feszültségingadozás stb.) esetén a képernyőre begépelt információ elvész. A legcélszerűbb, hogy minden alkalommal, amikor a képernyő megtelik, a programot kimentjük. Minthogy a kimentés mindig elölről indul, és a teljes programra vonatkozik, a szalagot mindig úgy pozícionáljuk, hogy az aktuális kimentéssel felülírjuk az előzőket. Így bármilyen hiba esetén mentesülünk a teljes program újrabegépelésétől.

Megjegyzések a programok kimentéséhez.

– A gép akkor is végrehajtja az átvitelt, ha a magnót elfelejtjük elindítani, de természetesen a program nem kerül a szalagra.

– A programnév tetszőleges szövegkonstans lehet, de fontos, hogy a programot egyértelműen azonosítsa, és konvenciónk értelmében ne kezdődjön kérdőjellel. Tapasztalatunk szerint legalkalmasabbak a 4–6 karakteres, lehetőleg értelmes programnevek. Csupán megszokásból jobb, ha a programnevek csak betűkből állnak, és már az első két karakterükben eltérnek egymástól.

– Egy kazettára több program is kimenthető, elvileg akárhány.

– Sokkal kényelmesebb azonban a programok kezelése, ha egy kazettán csak egyetlen program van. Sajnos a kereskedelmi forgalomban lévő kazetták túl nagyok, de azért saját kényelmünk érdekében próbáljunk meg beszerezni rövid, legfeljebb tízperces kazettákat. Ilyenek házilag is előállíthatók a rokonoktól, ismerősöktől begyűjtött használaton kívüli kazettákból, de ezt csak azoknak ajánljuk, akiknek megvan a technikai felszerelésük és a szakértelmük ehhez. Ugyanis már kis ügyetlenkedéssel is tökéletesen használhatatlan kazettákat lehet előállítani, ha például a szalagot megsértjük.

– A gép a magnót nem kezeli, minden szalagkezelő funkciót magunknak kell, manuálisan ellátnunk. Így nekünk kell gondoskodnunk a kazetta behelyezéséről és pozícionálásáról, elindításáról és megállításáról, továbbá arról, hogy a felvétel a szalag aktív részére kerüljön, ne pedig a befűzőszalagra, hogy a program kiférjen a kazettára, hogy ha a kazettán több program van, azok ne fedjék át egymást stb.

– A programok tárolására lehetőleg mindig ugyanolyan típusú kazettákat használjunk. Így megkímélhetjük magunkat attól, hogy a gépet és a magnót állandóan újra és újra össze kelljen hangolnunk.

– Minden kész, vagy egyelőre késznek tekintett programot két példányban mentsünk ki, külön-külön kazettára. Célszerű az egyik másolatot egy önálló kis kazettára venni, ez az üzemi példány; míg a másik másolatot egy nagy kazettára vigyük. Ezen lehetőleg legyen rajta annyi program, amennyi csak ráfér. Nevezhetjük programkönyvtárnak is. A program módosításakor ne felejtsük el a programkönyvtárban lévő változatot is módosítani.

– Különösen kényes a szalag pozícionálása, ha a kazettán már több program van, és most egy újabbat akarunk felvenni rá. Ha nem figyelünk eléggé, könnyen megsemmisíthetjük valamelyik régi programunkat. Viszonylag egyszerűbb a helyzet, ha a magnó olyan kijelző berendezéssel van felszerelve, melynek segítségével az egyes felvételek pozíciója egyértelműen nyomonkövethető. Ilyenkor csak az szükséges, hogy tudjuk, mettől meddig van szabad hely a kazettán. Ilyen számlálóberendezés hiányában a legbiztosabb módszer az, hogy beolvassuk a kazettáról az utolsó fentlévő programot, majd a szalagon 5–10 másodperces hézagot hagyunk. Ilymódon garantált, hogy értékes programot nem semmisítünk meg. Ezt a pozícionálást azonban csak akkor hajthatjuk végre, ha a kimentendő program még nincs a képernyőn. Ha már ott van, akkor csak a fülünkre hagyatkozhatunk. A magnó ugyanis lejátsza a jellegzetes füttyöt akkor is, ha a kazettát nem a géppel olvastatjuk be, hanem attól függetlenül csak lejátszuk, mintha egyszerű műsoros kazetta lenne. Ilyenkor a "belehallgatásos" módszerrel megkísérelhetjük megtalálni azt a csendes helyet, ahová a felvétel elkészíthető. Ez persze csak akkor lehetséges,

ha a kazettán nincsenek régi, részben felülírt felvételek, vagyis ha a kazetta új, vagy lelkiismeretesen végig van törölve. A programok elejének és végének megtalálását, valamint a programok azonosítását nagyon megkönnyíthetjük, ha minden program felvétele előtt egyszerűen mikrofon segítségével rámondjuk a szalagra a programra vonatkozó lényeges információkat; a nevét, a dátumot stb. És a programot csak ezután mentjük ki. Szövegünk a gépet semmilyen tekintetben nem befolyásolja.

– Minden kazettáról és minden kazettán lévő programról vezessünk pontos nyilvántartást. A kazettákat egyértelmű azonosítóval (például sorszámmal) lássuk el, úgy hogy az ne legyen könnyen eltávolítható. Vigyázat! Ne a kazetta dobozát jelöljük meg, hanem magát a kazettát. Minden egyes kazettához fektessünk fel egy pontos nyilvántartást, amely megadja, hogy a kazettán milyen program vagy programok vannak, és azt vagy azokat milyen szövegkonstanssal azonosítottuk a felvitelkor, továbbá ha a kazettán több program van, akkor azok milyen sorrendben követik egymást, és ha lehetőségünk van rá, azt is rögzíthetjük, hogy az egyes programok milyen pozíciókon találhatók meg. Ezt a nyilvántartást rávezethetjük magára a kazettára is, aminek az az előnye, hogy csak akkor veszhet el, ha a kazetta is elvész. Hátránya, hogy külön gondoskodnunk kell arról, hogy a nyilvántartás módosítását a kazetta rongálása nélkül is végrehajthassuk. Általában elegendő egy névjegykártya nagyságú papírlapot használni a nyilvántartás céljaira, és azt a kazetta dobozában tartani. Ha a kazettáinkat gondosan tároljuk és kezeljük, az összecserélés valószínűsége nem túl nagy. Általános jótanács, hogy tartsunk rendet magunk és gépünk körül. Egyébként személyes tapasztalatunk, hogy már öt kazettát sem lehet fejben nyilvántartani rendesen.

(b)Programok betöltése kazettáról.

A korábban kazettára kimentett programjainkat az alábbi módon olvashatjuk vissza a szalagról:

(1)Helyezzük el azt a kazettát a magnóban, amelyről a programot be kívánjuk tölteni a gépbe.
(2)Előre vagy hátracsévéléssel állítsuk be a szalagot arra a pontra, amely a kazettán tárolt program kezdete előtt van.
(3)Adjunk egy LOAD "programnév" parancsot.
(4)Nyomjuk meg a CR gombot.
(5)Indítsuk el a magnót lejátszásra.

A CR gomb megnyomása után a képernyő elsötétedik, és a gép türelmesen vár a magnó bekapcsolására. A magnó elindítása után lejátssza a szalagon lévő hangokat, így az általunk szalagra mondott szöveget, a felvitelkor a gép által keltett sípolást, illetve az ilyenkor lenyomott billentyűk adta hangjelzéseket is.

A szalagon lévő kimentett adatok elérésekor a gép ellenőrzi, hogy a kimentéskor felvett programazonosító megegyezik-e a LOAD parancsban megadott programnévvel.

Ha ezek megegyeznek, a gép beolvassa a programot. Sikeres beolvasás esetén a képernyőn újra megjelenik a kép, és rajta az OK üzenet olvasható. A magnó tovább megy, ne felejtsük el leállítani.

Sikertelen olvasáskor, vagy ha a LOAD parancsban megadott programnév nem egyezik meg a szalagon tárolttal, ugyanazt a jelenséget tapasztaljuk, mint amit a magnó beállításával foglalkozó részben hasonló esetre leírtunk.

Vigyázat! Minden egyes LOAD *parancs csak egyetlen olvasást hajt végre, annak ellenére, hogy a sikeres vagy sikertelen beolvasás után a magnó nem áll meg, és tovább sípol. Ha tehát egy kazettán több programunk van, például egy* LOAD "ALFA" *parancs hatására a gép nem fogja addig olvasni a szalagot, amíg meg nem találja az* "ALFA" *jelű programot, hanem csak az éppen soronkövetkező program beolvasását fogja megkísérelni. Ha ez nem az* "ALFA" *volt, akkor annak beolvasásához a* LOAD "ALFA" *parancsot ismételten ki kell adni. (Lásd még a program betöltéséhez fűzött megjegyzéseink utolsó bekezdését!)*

Megjegyzések a programok betöltéséhez.

Az alábbiakban nem ismételjük meg azokat a megjegyzéseket, amelyeket a programok kimentésénél már leírtunk, de a betöltésre is vonatkoznak. Csak a különbségekre és az esetleges eltérő kezelési módokra mutatunk rá.

– Ügyeljünk arra, hogy a magnót kimentéskor a CR gomb megnyomása előtt, betöltéskor pedig a CR után kell elindítani.

– Míg a SAVE parancs nem adható ki "programnév" nélkül, a LOAD paranccsal ez megtehető. Ilyenkor az éppen soronkövetkező programot fogja ellenőrzés nélkül beolvasni. Ne éljünk ezzel a lehetőséggel, mert több időt veszíthetünk a program újraolvasásával, ha nem a megfelelőt olvastuk be, mint a 4–6 karakternyi programnév begépelésével.

– Ha ERROR üzenetet kapunk, mindenképpen olvassuk újra a programot. Ne bízzunk abban, hogy a kilistázott programban úgyis felfedezzük az olvasáskor torzult részeket, és hogy ezeket javítással korrigálhatjuk.

Vigyázat! Akkor is sikertelen beolvasás áll elő, ha a program a SAVE parancs hatására hibásan került fel a kazettára. Vagyis az olvasáskor kapott ERROR *üzenetet nemcsak olvasási hiba, hanem közvetve szalagraírási hiba is okozhatja. Ha ilyesmire gyanakszunk, akkor próbáljunk meg más programokat is beolvastatni, vagy próbáljuk meg a sikertelenül olvasott programnak egy másik szalagra kimentett példányát beolvastatni. Ha ezek az olvasások sikeresek, akkor valószínű, hogy íráshibával van dolgunk.*

– Ha a sikertelen olvasás miatt elszáll a rendszer, nem kell bíbelődnünk a jól álcázott RESET gombbal, nyugodtan ki- és bekapcsolhatjuk a gépet. Nem veszítünk semmit.

– A szalag pozícionálásának betöltéskor nincs olyan kritikus jelentősége, mint kimentéskor. Olvasással ugyanis a szalag tartalmát elvileg nem ronthatjuk el. Akármilyen rosszul pozícionáljuk a szalagot, soha nem olvasódhat be más program, mint amelyiknek a nevét a LOAD parancsban megadtuk. Az sem baj, ha a szalag egészen ütközésig vissza van csévélve, mert a befűzőszalagot úgysem olvassa el a magnó.

– Ismert pozíciójú, de ismeretlen nevű program nevét a LOAD "?" paranccsal tudakolhatjuk meg.

– Ismert nevű, de ismeretlen pozíciójú programot a legnagyobb biztonsággal úgy találhatunk meg, hogy kiadjuk a LOAD "programnév" parancsot, majd ha nem a keresett programot találja a gép a beállított pozíción, addig hagyjuk futni a magnót, amíg a program végére nem ér. Ekkor leállítjuk a magnót, és újra kiadjuk ugyanazt a LOAD "programnév" parancsot, majd elindítva a magnót, ezt az eljárást addig ismételjük, amíg a kívánt programot meg nem találjuk. Ha ez a kazetta végéig nem következik be, kezdhetjük az egészet elölről egy másik kazettával. A módszer lassú és elég zajos, de bombabiztos. Egyébként használhatjuk a "belehallgatásos" módszert is, ami általában gyorsabb, és ha a programok közötti hézagokba mikrofonról felvettük a programneveket, majdnem ugyanilyen biztos is.

A MAGNÓ HASZNÁLATA ADATOK TÁROLÁSÁRA.

A kazettán nemcsak programok, hanem adatok is tárolhatók. Ezeket onnan programmal be is tudjuk olvastatni, és így feldolgozhatók. Az AIRCOMP gépen tehát kötegelt (batch) feldolgozást is megvalósíthatunk, természetesen bizonyos korlátok között. Hangsúlyozzuk azonban, hogy a gép nem erre való, ne is próbáljuk ilyen célokra használni.

Az AIRCOMP azonban kiváló fejlesztő gép, és mint ilyen, arra igenis nagyon alkalmas, hogy más gépekre, batch környezetre tervezett programjainkat és algoritmusainkat rajta kipróbáljuk, leteszteljük.

Minthogy az "idegen" programok tesztelési célú szimulálásával külön fejezet foglalkozik, itt csak a kazettán tárolt adatállományok kezelésével és a velük kapcsolatos magnókezeléssel foglalkozunk.

Noha előbb kellene foglalkoznunk az állományok létrehozásával, és csak ezután a feldolgozásukkal, mivel ez utóbbi csak akkor hajtható végre, ha a feldolgozandó állományt már létrehoztuk, az állományok kezelését mégis fordított sorrendben tárgyaljuk. Ennek egyetlen oka, hogy az állományok feldol-

gozása technikailag lényegesen egyszerűbb, mint a létrehozásuk, és így az állományok kezelésének alapelvei a feldolgozásukon keresztül – különösen a kezdők számára – könnyebben bemutathatók és megérthetők. Tudjuk, hogy némi türelmet és önfegyelmet igényel az Olvasótól, hogy ne ugorjon előre az állományok létrehozásához, mégis kérjük, hogy előbb a feldolgozással foglalkozó részt tanulmányozza át, majd áttérve a létrehozásra állítsa elő a bemutatott tesztállományt, és ezután térjen vissza a feldolgozás kipróbálására.

(a) Állományok feldolgozása kazettáról.

Példaként tekintsünk egy rendkívül leegyszerűsített programot, amely kötegelt feldolgozást végrehajtva a mellékelt tesztállományból az ugyancsak mellékelt listát állítja elő.

TESZTADATOK A BÉRLISTÁZÓ PROGRAMHOZ

```
név         = TOTH ISTVAN
órabér      = 1ϕ.1ϕ
órák száma  = 11ϕ
név         = KISS JOZSEF
órabér      = 2ϕ.2ϕ
órák száma  = 12ϕ
név         = NAGY JANOS
órabér      = 3ϕ.3ϕ
órák száma  = 13ϕ
név         = KOVACS PAL
órabér      = 4ϕ.4ϕ
órák száma  = 14ϕ
név         = ***
órabér      = ϕ
órák száma  = ϕ
```

A BÉRLISTÁZÓ PROGRAM TABLÓJA

```
--------BERLISTA------
DOLGOZO NEVE:    FIZETESE:
-------------------------
TOTH ISTVAN          1111
KISS JOZSEF          2424
NAGY JANOS           3939
KOVACS PAL           5656
=========================
OSSZESEN:           1313ϕ
-------------------------
```

**** bérlistázó program ****

```
----
 110   SS=ϕ
 120   M$="   "
 130   C$="----------BERLISTA----------"
 140   A$="DOLGOZO NEVE:          FIZETESE:"
 150   E$="============================"
 160   O$="OSSZESEN:                    "
 170   K$="------------------------------"
 180   U$="                             "
---- előkészítés
1010   PRINT M$;C$
1020   PRINT M$;A$
1030   PRINT M$;K$
---- bérlistázás
2010   INPUT #1;NV$,OB,SZ
2020   IF NV$="***" THEN GOTO 3010
2030   FI=OB*SZ :SS=SS+FI
2040   NV$=LFT$ (NV$+U$,27-LEN(STR$(FI)))
2050   PRINT M$;NV$;FI
2060   GOTO 2010
---- befejezés
3010   PRINT M$;E$
3020   S$=LFT$(O$,27-LEN(STR$(SS)))
3030   PRINT M$;S$;SS
3040   PRINT M$;K$
---- program vége
4010   END
```

Megjegyzések az állományok feldolgozásához.

– A kazettán tárolt állomány adatait olyan INPUT utasítással olvashatjuk be, amelyben a kazettás magnó egységszáma szerepel. Ez mindig #1. A numerikus konstans helyett megadhatunk ugyanilyen tartalmú változót is. Ezután pontosvessző következik, majd egymástól vesszővel elválasztva fel kell sorolni mindazokat a változókat, amelyeknek a beolvasással értéket akarunk adni.

– Noha nemcsak a magnónak, hanem a képernyőnek is van egységszáma (éspedig #ϕ), az onnan adatot váró INPUT utasításban ne használjuk, hiányával is hangsúlyozva a képernyő kitüntetett szerepét.

– A kazettán tárolt állomány adatai között kell, hogy legyen egy kitüntetett adat, amelynek valamely speciális értéke az állomány végét (end of file) jelzi. Ennek figyeléséről a programnak kell gondoskodnia.

– A gép maga nem figyeli az állomány végét, mivel számára az állomány nem is létezik, minden egyes INPUT utasítás egy fizikai rekordot olvas be. Ha a beolvasási ciklusból nem lépünk ki idejében, a gép gátlástalanul tovább olvas, még akkor is, ha az adatok már régen elfogytak, vagyis hát keresi a következő rekordot.

– A programban előforduló legelső INPUT utasítás végrehajtásakor a képernyő elsötétül, és ettől kezdve nem használható. A program várja a magnó elindítását. Ilyenkor lejátszásra kell elindítani.

– A magnó elindítása után hasonló hangjelenségeket észlelünk, mint a program beolvasásakor.

– A program sikeres befejezésekor visszaáll a kép, amely már tartalmazni fogja a program végrehajtása közben PRINT utasítással kiírt információkat is. Ekkor a magnót leállíthatjuk.

– Olvasási hiba esetén a rendszer automatikusan generál egy GOSUB 32000 utasítást az INPUT utasítás helyett. Így erre az utasításszámra kódolhatjuk a hibakezelő tevékenységünket, amelyben a 32759-ig terjedő sorszámokat használhatjuk fel. Szabályos szubrutin lévén, RETURN utasítással kell befejeznünk. Például:

```
----  megszakítás lekezelése
32000 H=1
32010 RETURN
----
```

A programot ilyenkor így módosítjuk:

```
----
2015  IF H=1 THEN GOSUB 10010 :GOTO 4010
----  hibarutin
10010 PRINT M%;"OLVASASI HIBA!"
10020 PRINT M%;"UTOLSO FELDOLGOZOTT REKORD=";NV%
10030 RETURN
----
```

Ez a megoldás csak akkor használható, ha az olvasási hiba megszakítást (interrupt) eredményez. Megjegyezzük, hogy az ilyen megszakítások gépi tudomásulvétele a POKE 16443,0 utasítással vagy paranccsal letiltható; a POKE 16443,1 utasítással vagy paranccsal újra engedélyezhető.

– Látható tehát, hogy mivel a képernyő a teljesfeldolgozás alatt sötét, és a feldolgozás eredményét csak a program befejezése után láthatjuk, csak annyi adatot érdemes feldolgozni, amennyinek megfelelő eredmény a képernyőn elfér, ugyanis a képernyőről kifutott eredmények jóságát nem tudjuk ellenőrizni. Így ez a lehetőség valóban csak tesztelésre jó.

(b) Állományok létrehozása kazettán.

Némi nehézséget okoz az a tény, hogy az I/O azaz író/olvasó műveletek végzése közben a képszerkesztés szünetel. Amikor az adatátvitel a magnóra elkezdődik, a képernyő elsötétül, és úgy is marad a program végéig. A billentyűzet természetesen aktív marad, így onnan adatok bevihetők, de a képernyőn nem jelenik meg semmi, a programból oda kiírt, vagy a billentyűzetről begépelt szövegek csak a program lefutása után válnak láthatóvá.

Ezt a kényelmetlenséget kétféle módon is megszüntethetjük. Vagy minden egyes szalagraíró művelet után visszaállítjuk a képet, vagy a képernyőt használó, és a magnókezelő tevékenységeket a programban szétválasztjuk egymástól úgy, hogy az előbbiek megelőzzék az utóbbiakat.

A tevékenységek szétválasztásával a már bemutatott bérlistázó program tesztadatait például az alábbi módon vihetjük fel kazettára.

```
**** adatfelvívő program ****
----
----  deklarációk
 110  DIM NV$(15),OB(15),SZ(15)
----  adatok bekérése terminálról
1010  PRINT ""
1020  PRINT "   KEREM A DOLGOZOK ADATAIT!"
1030  FOR I=1 TO 15
1040  PRINT ""
1050  PRINT "   -";I;"-"
1060  INPUT "   DOLGOZO NEVE=? ";NV$(I)
1070  INPUT "      ORABERE=? ";OB(I)
1080  INPUT "      ORAK SZAMA=? ";SZ(I)
1090  IF NV$(I)="***" THEN POP :GOTO 1110
1100  NEXT I
1110  PRINT ""
1120  PRINT "   ADATBEKERES VEGE."
----  átállás
2010  PRINT ""
2020  PRINT "   KEREM A MAGNOT FELVETELRE ELINDITANI!"
2030  INPUT "   MEHET AZ ADATFELVITEL? ";VA$
----  adatok felvitele kazettára
3010  FOR J=1 TO I
3020  PRINT #1;NV$(J),OB(J),SZ(J)
3030  NEXT J
3040  PRINT ""
3050  PRINT "   ADATFELVITEL KESZ."
3060  PRINT "   A MAGNOT LE LEHET ALLITANI!"
----  program vége
4010  END
```

Megjegyzések az állományok létrehozásához.

– Adatokat a kazettára felírni a PRINT utasítással lehet. Ebben meg kell adni a kazettás magnó egységszámát, ami mindig #1. A numerikus konstans ugyanilyen tartalmú változóval helyettesíthető. Ezután pontosvessző következik, majd egymástól vesszővel elválasztva fel kell sorolni mindazokat a változókat, amelyeknek tartalmát egy fizikai rekordban a kazettára fel akarjuk vinni.

– Noha nemcsak a magnónak, hanem a képernyőnek is van egységszáma (éspedig #0), a képernyőre való kiíráshoz ezt ne használjuk, hiányával is hangsúlyozva a képernyő kitüntetett szerepét.

– Az adatfelvitel megkezdésekor a képernyő elsötétül, és a szokásos hangjelenségeket halljuk, de nem folyamatosan, hanem szaggatottan, ahogyan a név-órabér-órakszáma adatokból képzett rekordok sorban egymás után felkerülnek a szalagra. (Itt, és mostantól fogva rekordnak nevezzük az adatoknak olyan csoportját, amelyek a szalagon összefüggően helyezkednek el, úgy hogy közöttük nincs hézag.)

– A kép csak a program lefutása után áll vissza, és ugyanúgy mint az állományok feldolgozásánál, megjelennek rajta az időközben végrehajtott PRINT utasítások eredményei.

– Ha olyan íráshiba fordul elő, amely megszakítást okoz, ugyanúgy járhatunk el, mint az állományok feldolgozásánál. A gép generálni fogja a GOSUB 32000 utasítást, és erre az utasításszámra kódolhatjuk a megszakítás lekezelését végrehajtó saját eljárásunkat.

Megjegyzések az adatfelvívő programhoz.

– Jól látható, ahogyan az adatok bekérése és felvitele el van választva egymástól. Közöttük egy átállási tevékenység van.

– Az átállási tevékenységben nem a 2020 PRINT utasítás a fontos, annak csak tájékoztató szerepe van. Igazi jelentősége a 2030 INPUT utasításnak van, mert hatására megáll a program, és a válaszra vár. Így van időnk a magnó elindítására. Nélküle a felvitel olyan gyorsan elindulna, hogy lehetetlen lenne időben bekapcsolni a magnót, és az adatok egy része elveszne. Vagy ha ezt el akarjuk kerülni, már előre, jóval korábban el kellene indítani a magnót.

– Minthogy a felvitel előtt minden adatot be kell kérnünk, gondoskodnunk kell ezek tárolásáról, mindaddig, amíg a felvitelükre sor nem kerülhet. Ezt a legegyszerűbben tömbökkel oldhatjuk meg. A tömbhasználatnak a gép méretéből fakadóan vannak fizikai korlátai, de ha csak néhány tucat tesztadatról van szó, ezeket a korlátokat nemigen érhetjük el.

– A példaprogramban akármit válaszolunk a 2030 INPUT kérdésre, a CR gomb megnyomása után elindul a felvitel, sőt ezt teszi akkor is, ha a kérdésre nem válaszolunk, csak a CR gombot nyomjuk meg. Ha el akarjuk kerülni,

hogy a CR gomb véletlen megnyomásától a felvitel szándékunk ellenére elinduljon, akkor bővítsük ki a programot például az

2040 IF VA$ <> "IGEN" THEN GOTO 2030

utasítással.

– Nincs kapcsolatban az adatfelvitellel, mégis felhívjuk a figyelmet a 1090 sorban lévő POP utasításra. Minthogy ezt az utasítást már korábban tárgyaltuk, itt annyit jegyezzünk meg, hogy a POP hiánya általában nem okoz problémát, de fennáll az a veszély, hogy a visszatérési címeket tároló verem, véges mélysége folytán, előbb utóbb megtelik.

Az alábbiakban bemutatjuk az adatfelvitelnek a fentitől eltérő változatát, amely az adatokat rekordonként viszi fel. Előnye, hogy nem kell tömböket használnunk.

```
**** rekordonkénti adatfelvitel ****
----
----  adatok bekérése terminálról
1010  INPUT " DOLGOZO NEVE=? ";NV$
1020  INPUT "       ORABERE=? ";OB
1030  INPUT "       ORAK SZAMA=? ";SZ
----  átállás
2010  INPUT " A REKORD FELVIHETO? ";VA$
----  adatok felvitele kazettára
3010  PRINT #1;NV$,OB,SZ
----  program vége
4010  END
```

A programot az alábbi módon használhatjuk:

(1)Kiadunk egy RUN parancsot, amivel elindítjuk a programot.
(2)Begépeljük a válaszokat az 1000-es sorozat INPUT kérdéseire.
(3)Elindítjuk a magnót felvételre.
(4)Megadjuk a választ a 2010 INPUT kérdésre. Elég egy CR.
(5)A program lefutása után megállítjuk a magnót, de nem csévéljük vissza.

A fenti öt lépést minden egyes rekordra külön-külön végre kell hajtani.

Megtehetjük azt is, hogy a magnót az első rekord felvitele előtt elindítva állandóan bekapcsolva tartjuk, és csak az utolsó rekord felvitele után állítjuk le. Ekkor megtakaríthatjuk a (3) és (5) lépések állandó ismétlését, és nincs szükség a 2010 utasításra sem.

A programot akkor célszerű használni, ha sok adatot tartalmazó, hosszú rekordokból álló állományt kívánunk kazettán rögzíteni. Egyébként csakis ezt a programot használhatjuk, ha a felvívendő adatok együttes tárolásához nem tudunk elegendő mennyiségű és méretű tömböt deklarálni.

Adatfelvitel a kép helyreállításával.

Ha az eddig ismertetett felviteli lehetőségek közül egyik sem nyeri meg a tetszésünket, megtehetjük, hogy az adatfelvitel után mindig visszaállítjuk a képet. Így felváltva gépelhetünk be adatokat, és vihetjük fel azokat kazettára, anélkül, hogy a sötét képernyő kényelmetlenségét el kellene tűrnünk.

```
**** adatfelvitel a kép helyreállításával ****
----
----  adatok bekérése terminálról
1010  INPUT " NEV=? ";NV$
1020  INPUT " BER=? ";OB
1030  INPUT " ORA=? ";SZ
----  adatok felvitele kazettára
3010  PRINT #1;NV$,OB,SZ
----  visszaállítás
4010  DL=2ϕϕ
4020  IF NV$<>"***" THEN GOTO 1010
----  program vége
5010  END
```

Megjegyzések a kép visszaállításához.

– A kép a 4010 DL=2ϕϕ utasítás hatására visszaáll. Megjelenik rajta a képernyő elsötétülése előtti kép, kiegészítve az időközben esetlegesen oda kiírt szövegekkel.

– Nincs szükség átállási tevékenységre. Ha a magnót az utolsó (1030) INPUT utasításra kiadott CR előtt elindítjuk, folyamatosan működhet mindaddig, amíg a program végén le nem állítjuk.

– Ha így teszünk, a szalagon a rekordok közötti hézag (gap) meglehetősen nagy lesz. Ez egyrészt szalag pazarlás, másrészt visszaolvasáskor a feldolgozást is lelassítja. Különösen akkor, ha nagyon sok és/vagy nagyon hosszú adatot kell a képernyőről beadnunk. Mellesleg ilyenkor a magnó felveszi az adatok begépelésekor keletkező csipogó zajt is.

– A rekordhézagok csökkenthetők, ha a magnót minden egyes adatátvitel után, amikor a kép visszajön, leállítjuk, majd sorra begépeljük az INPUT utasításokra a válaszokat, de az utolsó (1030) INPUT utasításra kiadott CR előtt a magnót újra elindítjuk. Természetesen mindig felvételre. Minderre bőven van időnk, mialatt a gép az INPUT utasításoknál a válaszra vár.

– Ha valamelyik alkalommal elfelejtenénk újraindítani a magnót, a hiányzó rekord adatait ismét begépelve, és most már elindított magnóval a felvitelt a program megszakítása nélkül végrehajthatjuk.

A kazettát illetve a szalagot érő mechanikus roncsolásoktól eltekintve, programjainkban és adatainkban kétféle beavatkozás okozhat közvetlen vagy közvetett kárt:
– a véletlen megsemmisítés, és
– az illetéktelen hozzáférés.

(a) A véletlen megsemmisítés elleni védelem.

Ugyanazt a módszert alkalmazhatjuk, amit a műsoros kazettákkal is tenni szoktunk, azaz kitörjük a kazetta sarkán lévő műanyag pöcköt. Ettől kezdve a kazetta csak olvasható, írni rá nem lehet. Ilymódon mind programjainkat, mind adatainkat megvédhetjük a felülírástól. (Ha később mégis át akarjuk írni a kazettát, a kitört pöcök helyét Cellux-szal be kell ragasztani.)

(b) Az illetéktelen hozzáférés elleni védelem.

Kazettán lévő adatainkat nem védhetjük meg, csak lakattal; bárki, aki hozzáférhet egy adatkazettánkhoz, el is olvashatja azt.

RENDKÍVÜLI ESEMÉNYEK

MEGSZAKÍTÁS, TOVÁBBINDÍTÁS, TÖRLÉS

HIBAKEZELÉS

MEGSZAKÍTÁS, TOVÁBBINDÍTÁS, TÖRLÉS

Ha a RUN/BRK gombot programfutás *közben* nyomjuk meg, akkor a programfutás megszakad.

Kulcsszó: BREAK
Jelentése: szakítsd meg

Ez a kulcsszó csak a RUN/BRK gombbal állítható elő.

Az így leállított program a CONT paranccsal továbbindítható
- a gép legközelebbi kikapcsolásáig,
- a legközelebbi NEW parancsig,
- a legközelebbi GOTO parancsig,
- a legközelebbi RUN parancsig,
- a programba való legközelebbi belejavításig.

Kulcsszó: CONT
Jelentése: folytasd

A CONT kulcsszó (a CONTINUE rövidítése) csak parancsban használható, utasításban nem.

Akár normálisan, akár hiba miatt, akár BREAK hatására áll le a program, az adatok értékei megmaradnak
- a gép legközelebbi kikapcsolásáig,
- a legközelebbi NEW parancsig,
- a legközelebbi CLR parancsig,
- a legközelebbi RUN parancsig,
- a programba való legközelebbi belejavításig.

Ilyenkor tehát az értékek kiírathatók, parancsokkal megváltoztathatók.

Akár normálisan, akár hiba miatt, akár BREAK hatására áll le a program, maga a program a memóriában marad
- a gép legközelebbi kikapcsolásáig,
- a legközelebbi NEW parancsig.

Kulcsszó: CLR
Jelentése: töröld

A CLR (a CLEAR rövidítése) az adattartalmakat törli, de a programot változatlanul hagyja, ellentétben a NEW paranccsal, amely törli az adattartalmakat és a programot is.

Vigyázzunk: a BREAK *hatására leállt program a* CLR *parancs kiadása után* CONT *paranccsal továbbfuttatható, de a program valamennyi addigi adata elveszett, ezért az eredmények hibásak lehetnek!*

Ha valamelyik SH-gomb lenyomva tartása közben lenyomjuk a CR gombot, akkor a képernyő tartalma eltűnik, de a memóriában minden program és minden adat megmarad.

Van ezeken kívül egy „vészkapcsoló", – amint erről már esett szó – amelyet akkor használhatunk, ha semmi más nem segít.

Ennek a kapcsolónak (RESET-gomb, visszaállítógomb) a megnyomására
– minden futó program azonnal leáll,
– megjelenik az – AIRCOMP – felirat, miközben a memóriában minden program és minden adat megmarad,
– vagy (ritkán) megjelenik az a teljes felirat, ami a gép bekapcsolásakor szokott megjelenni, miközben a teljes memóriatartalom elvész.

HIBAKEZELÉS

A hiba miatt leállt program leállásakor hibajelzés jelenik meg és a képernyőre kiíródik az a sor, amelyikben a hibát a gép észrevette. (Ha egy sorban több utasítás van, semmi sem jelzi, hogy melyiknek a végrehajtása közben következett be a leállás.) Ugyanilyen hibajelzés jelenik meg a parancsokban előforduló hibák hatására is.

Fontos, hogy a hibajelzés nem feltétlenül arra a pontra mutat, ahol a hiba van, hanem ahol azt a gép észlelte. Ha például egy húszelemű tömb értékeit a

```
1 FOR E=ϕ TO 2ϕϕ
2 PRINT T(E)
3 NEXT E
```

sorozattal akarjuk kilistázni, akkor a hiba nyilvánvalóan a FOR utasításban van, ahová végértékként 2ϕϕ-at írtunk 2ϕ helyett, de hibajelzés a 2-es sorra érkezik, amikor E felveszi a 21-es értéket. (A gép ugyanis semmi kivetnivalót nem talál abban, hogy egy E nevű változó a 21-es értéket felvegye; csak az a hiba, amikor a húszelemű tömb ennyiedik elemére hivatkozunk.)

Vannak hibák, amelyek csak programozói hiba miatt fordulhatnak elő, és vannak, amelyek kezelési hiba miatt is.

Csak programozói hibára utalhat, ha a következő hibajelzések valamelyike jelenik meg:

CD CONT parancsot adtunk ki, de a program nem indítható tovább (mert végetért, mert más hiba miatt állt le vagy mert bekövetkezett a CONT ismertetésekor felsorolt valamelyik, a továbbindítást megtiltó esemény.)

OD több READ-utasítást adtunk ki, mint amennyit a DATA utasítás(ok)ban rendelkezésre álló adatok lehetővé tettek volna.

PP NEXT utasítást adtunk ki, pedig nem volt FOR; RETURN utasítást adtunk ki, pedig nem volt GOSUB; POP utasítást adtunk ki, pedig nem volt sem FOR, sem GOSUB.

SN szintaktikus hibát követtünk el (például PRINT helyett RPINT-et írtunk, a nyitó- és csukózárójelek száma nem egyezik stb.)

TM nem megfelelő típusú kifejezést használtunk: számot, numerikus változót vagy numerikus kifejezést írtunk oda, ahová szöveget kellett volna vagy fordítva.

UF definiálatlan függvényre hivatkoztunk (például végre akartuk hajtani az A=FNX(B) utasítást, noha a programban DEFFNX utasítás nem volt vagy még nem hajtódott végre.)

US GOTO, GOSUB vagy RUN olyan programsorra hivatkozott, amilyen számú sor nincs a programban

DD egyszer már dimenzionált tömbre másodszor is végre akartuk hajtani a DIM utasítást.

Kezelési hibából is előfordulhatnak a következő hibák:

Jel:	Sz.:	Magyarázat:
/ϕ	1	ϕ-val való osztást kíséreltünk meg.
IQ	2	olyan adatot adatot használtunk, amely kívül esik a megengedett értelmezési tartományon (például DL=5$\phi\phi$ utasítást akartunk végrehajtani, pedig DL értéke legfeljebb 2$\phi\phi$ lehet.)
OM	3	elfogyott az engedélyezett memóriaterület.
OV	4	túlcsordulás következett be (mert ábrázolhatatlanul nagy számot akartunk használni.)
ST	5	egyetlen szövegváltozóba 255-nél több karaktert próbáltunk betölteni.
BS	6	nemlétező tömbelemre hivatkoztunk (például DIM T(15) után hivatkoztunk T(16)-ra.)

A programfutás a fenti hibák bármelyikének a bekövetkezésekor abbamarad. A második felsorolásban szereplő hibák esetében azonban megvan a módunk arra, hogy magunk döntsük el, hogy mi történjen a hiba bekövetkezésekor (például, hogy milyen üzenet jelenjen meg a kezelő előtt, milyen módosított adatokkal folyjon esetleg mégis tovább a programfutás stb.)

Ha a programban kiadjuk a

... POKE 16457,216,27

utasítást, akkor attól kezdve a második felsorolásban szereplő hat hiba nem szakítja meg a programfutást, hanem a gép úgy viselkedik ilyen hiba bekövetkezésekor, mintha GOTO 31$\phi\phi\phi$ utasítást adtunk volna ki. 31$\phi\phi\phi$-es sorszámmal kezdve (legfeljebb 31999-es sorszámig) tehát írhatunk egy olyan programrészletet, amelyik az általunk kívánt módon válaszol a hibajelenségre. Az ebben a programrészletben elhelyezett

... változó= PEEK(16463)

utasítás hatására a változó értéke az a szám lesz, ami a felsorolásban a hiba jele mellett szerepelt: Iϕ esetén 1, IQ esetén 2, OM esetén 3, OV esetén 4, ST esetén 5, BS esetén 6. Az ebben a programrészletben elhelyezett

... változó=PEEK(16464)+256*PEEK(16465)

utasítás hatására a változó értéke annak a sornak a sorszáma lesz, amelyikben a gép a hibát észlelte.

Vigyázzunk: ilyen esetben az eredeti programba csak GOTO *utasítással térhetünk vissza, és nem térhetünk vissza sem ciklus belsejébe, sem* GOSUB*-bal aktivizált szubrutin belsejébe, különben* PP *jelű hibaüzenetet kapunk.*

Kellemetlen kivétel az ON-utasítás egy speciális hibája: ha például az

... ON X ' GOTO 11 ' ' GOTO 22

utasítás végrehajtásakor X értéke mégis 2, ez kezelési hibából is adódhat, (téves adatbevitelből,) a gép mégis SN jelű hibát jelez, ami mindenképpen programleállást eredményez.

Kulcsszó: TRC
Jelentése: kövesd nyomon

A TRC (a TRACE szó rövidítése) csak utasításban használható; parancsban hatástalan.

Ha programban szerepel, attól kezdve minden végrehajtott sor száma megjelenik a képernyőn. Ha a sorban több utasítás van, akkor a sorban lévő második és minden további utasítás végrehajtásakor egy-egy kettőspont is megjelenik. Az IF és az ON külön utasításoknak számítanak, így például az

55 IF A=11 THEN X=3 : Y=X+A

végrehajtásakor, ha volt korábban TRC utasítás,

55

jelenik meg, ha A nem volt egyenlő 11-gyel, vagyis ha csak a vizsgálat zajlott le, és

55 : :

jelenik meg, ha A egyenlő volt 11-gyel, tehát lezajlott a másik két művelet is.

A TRC hatása a program végéig vagy újabb TRC utasítás végrehajtásáig tart.

A TRC szerepe az, hogy nyomonkövethessük a programfutás menetét, ha nem tudjuk biztosan, hogy egy ciklus hányszor is hajtódik végre, egy utasításra rákerül-e a vezérlés, egy többutasításos sor melyik utasításában következett be a leállás stb. Mivel a képernyőn csak 25 sor fér el, a TRC-ket úgy kell elhelyezni, hogy csak a legfontosabb, legkényesebb néhány utasítást fogják közre. A túl sok nyomkövetési információ, még ha el is tudjuk olvasni, nem nyújt annyi segítséget, mint ha gondosan megválasztjuk a nyomkövetés helyét.

A PROGRAMKÓDOLÁS TECHNIKÁJA

KÓDOLÁSI TANÁCSOK

A MEGJEGYZÉSEK KÓDOLÁSA

A KÓD OPTIMALIZÁLÁSA

KÓDOLÁSI TANÁCSOK

Az alábbiakban néhány hasznos kódolási szabályt mutatunk be. Ezek betartását a gép nem ellenőrzi, tehát nem kötelezőek, de a programozó érdeke, hogy ne puszta ajánlásnak tekintse őket.

A program belsejében lehetőleg ne használjunk konstans adatokat, hanem adjunk nekik nevet és a program elején kapják meg az értéküket. Például

```
11ϕ HO$="APRILIS"
12ϕ KP=3 : VP=11 : NP = 2
      .
      .
      .
2ϕ1ϕ PRINT HO$
      .
      .
      .
3ϕ8ϕ FOR I=KP TO VP STEP NP
```

Ezzel elkerülhetjük, hogy egy-egy adat megváltozásakor az adat valamennyi előfordulását meg kelljen a programunkban keresnünk; elég a program elején az értékadást kijavítani.

Mindig szabad a vezérlést egy modul (pl. szubrutin) belsejéből a modul kilépési pontjára, az END vagy a RETURN utasításra átadni.

Mindig szabad a vezérlést egy szerkezeti elem belsejéből a szerkezeti elem kilépési pontjára (például a FOR utasítással szervezett ciklusból a NEXT utasításra) vagy a következő szerkezeti elem kezdőpontjára átadni.

Tilos a vezérlést egy modulból (pl. egy szubrutinból) a modulon kívülre átadni.

Tilos a vezérlést egy szerkezeti elemből a szerkezeti elemen kívülre átadni, kivéve a fentebb említett esetet: a következő szerkezeti elem kezdőpontjára való vezérlésátadást. (Például, ha egy feltétel nem teljesül, akkor a vezérlést a feltételes szerkezetet követő első utasításra adhatjuk át.)

Tilos a vezérlést egy modul vagy szerkezeti elem belsejébe átadni; a moduloknak, szerkezeti elemeknek csak egyetlen kezdőpontja lehet.

Minden egyes vezérlésátadást aszerint kell megítélni, hogy megbontja-e a program szerkezetének egységét. Ha igen, az a vezérlésátadás tilos. (Ez nem új szabály, hanem az eddigiek összefoglalása.)

Általában minden utasítást külön sorba írjunk, ha nincs nyomós okunk az ellenkezőjére. A modulok, szerkezeti elemek kezdő- és záróutasításai (pl. END, RETURN, GOSUB, FOR, NEXT, IF, ON stb.) azonban mindenképpen külön sorban legyenek, hogy a modulok, szerkezeti elemek eleje és vége feltűnő legyen.

A MEGJEGYZÉSEK KÓDOLÁSA

Mielőtt a megjegyzések kódolásának szintaktikus szabályaira kitérnénk, néhány szót szeretnénk szólni magukról a megjegyzésekről. Következzék tehát a megjegyzések megjegyzése.

Ha a megjegyzéseket csoportosítani akarjuk, alapvetően két típust találhatunk. Ugymint

– szükséges megjegyzések, és
– felesleges megjegyzések.

Például ez a bekezdés a második kategóriába tartozik.

Már most leszögezzük, hogy a programban felesleges megjegyzések ne legyenek.

A szükséges megjegyzések mindig valamilyen információt hordoznak. Ez lehet fontos, vagy kevésbé fontos, de mindig olyan információ, ami a program kódjából vagy egyáltalán nem nyerhető ki, vagy kinyerhető ugyan, de nem könnyen, nem félreérthetetlenül, nem szemléletesen.

A megjegyzések funkciójuk szerint lehetnek

– magyarázó, vagy
– dokumentatív

jellegűek.

A magyarázó jellegű megjegyzések általában kiegészítik, megmagyarázzák, szemléltetik a programnak azokat a részeit, amelyeket a nyelvi eszközök szegényessége miatt nem tudtunk áttekinthetően, egyszerűen, közérthetően kódolni.

Például a

```
5410 FOR I=1 TO 6
5420 J=INT(I/4)
5430 K=I-3*J
5440 J=J+1
5450 M(J,K)=M(J,K)+V(I)
5460 NEXT I
```

programrészlet funkcióját és működését meg lehetne fejteni csupán a kódból is. De valljuk be őszintén, hogy nem tűnt fel első látásra, hogy tulajdonképpen egy hatelemű vektor elemeit adja hozzá egy kétszer hármas mátrix elemeihez, sorfolytonosan.

Erre a tényre például az alábbi megjegyzéssel hívhatjuk fel a program olvasójának figyelmét:

```
---- gyűjtés vektorból mátrixba sorfolytonosan
```

vagy esetleg röviden utalhatunk arra is, hogy melyik vektorelemet melyik mátrixelembe gyűjti az eljárás. Mondjuk így:

```
---- 1,1=1 1,2=2 1,3=3 2,1=4 2,2=5 2,3=6
```

Ugyanis az ilyen jellegű megjegyzéseknek nem kell szószátyáraknak lenniök. Néha elegendő egy-két, a lényegre mutató szó, és minden megvilágosodik.

Mindig tartsuk szem előtt, hogy megjegyzéseink nem a laikus olvasónak, hanem egy másik programozónak, egy szakembernek szólnak, aki tudja mindazt, amit mi a programozásról tudunk, sőt még többet is. De ha mégsem, nem a mi programunk kötelessége, hogy erre őt megtanítsa.

Annak eldöntésére, hogy a program mely részeit szükséges kommentálni, és hogy mi legyen a megjegyzés tartalma, a legjobb módszer az, hogy elolvastatjuk a programunkat egy kollégánkkal. Amikor ő feltesz egy kérdést, akkor amit erre válaszolunk, azt kell a programba megjegyzésként beletenni.

De hova tegyük a megjegyzést? A lehető legközelebb ahhoz a helyhez, ahol a kérdés felmerült. Ugyanakkor a kód folyamatos olvashatóságát lehetőleg ne bontsuk meg. Ezért a megjegyzéseket mindig a problémát okozó szerkezeti elem vagy egység elejére tegyük. A mi példánkban az 5010 sorba, vagy az elé.

A megjegyzéseket jól láthatóan különböztessük meg, és különítsük el a kódtól. Ne legyenek azzal összetéveszthetők vagy összeolvashatók. Például:

```
2020 IF TR$="T" THEN GOTO 4010 === törlés
```

vagy

```
3030 GOSUB 12010 === beszúrás
```

Ezek egyébként tipikus magyarázó jellegű megjegyzések. A kód nélkülük is érthető lenne, de velük sokkal könnyebb.

A dokumentatív megjegyzések általában nem a kódra vonatkoznak, hanem olyan információkat tartalmaznak a programról, vagy magáról a programmal megoldott problémáról, amelyek kóddal nem adhatók meg, vagy nem eléggé figyelemfelkeltően. Például:

```
---- "SZAMLAZO" program: 83/05/12 változat
```

A program neve és az utolsó módosítás dátuma lényeges információ, kódolásukat a BASIC mégsem támogatja külön erre a célra rendszeresített utasítás-

sal. Más példa:

```
13070 RETURN
------ átlagszámítás
14010 A=φ
```

Itt a kódból egyértelműen kiderül, hogy hol kezdődik az átlagszámító eljárás, de ha "elbújik" a környező utasítások között, és nehezen található meg, célszerű egy megjegyzéssel felhívni rá a figyelmet.

Tipikusan dokumentatív jellege van az adatok leírásának. Sajnos erre nagyonis szükségünk van, mert a BASIC adatnévképzési szabályai miatt az adatnevek ritkán fejezik ki azt, amit jelölnek. Az adatleírásokat ugyan külön dokumentációban szokás megadni, de ha valamely adatot a programban amúgyis deklarálni kell, például dimenzionálás vagy kezdőértékadás céljából, akkor csökkenthetjük a külön dokumentáció terjedelmét, ha az adatokat a programban kommentáljuk. Például:

```
130 ER=φ --- hibakód, tartalma = φ, ha nem volt hiba
140 DIM UZ$(12) --- hibaüzenetek
```

Esetenként, noha egyébként nem is lenne rá szükség, megéri az adatot csak azért deklarálni, hogy melléje elhelyezhessük a megjegyzésünket.

A dokumentatív megjegyzések formájára és elhelyezésére ugyanaz vonatkozik, amit a magyarázó jellegű megjegyzésekről elmondtunk: ott és annyit kell a programot kommentálni, ahol és amennyire az olvasó a kódot információszegénynek érzi.

Általános alapelvnek tekinthetjük, hogy a program szerkezeti elemeit célszerű dokumentatív megjegyzésekkel megjelölni. Elsősorban az elejüket, de esetenként akár a végüket is.

Ezután a bevezetés után térjünk rá arra, hogy hogyan lehet a megjegyzéseinket a programban elhelyezni, kódolni.

Nos, az AIRCOMP-16 BASIC-ben nem lehet önálló megjegyzéssorokat képezni. Megjegyzések csak olyan sorokban lehetnek, amelyekben utasítás is van. Ha a sorban több utasítás szerepel, a megjegyzés csak a legutolsó utasítás után következhet.

A megjegyzéseket úgy kódoljuk, hogy a sor utolsó utasítása után egy aposztrófot, majd egy idézőjelet teszünk ki. Minden, amit ezután írunk le, megjegyzés lesz, egészen a sor végéig. Például:

```
130 ER=φ'" --- HIBAKOD = φ, HA NEM VOLT HIBA
140 DIM UZ$(12)'" --- HIBAUZENETEK
2020 IF TR$="T" THEN GOTO 4010 '" === TORLES
3030 GOSUB 12010 '" === BESZURAS
13070 RETURN
14010 A=φ'" --- ATLAGSZAMITAS
```

Az utóbbit úgy is kódolhatjuk, hogy a megjegyzést egy sorral előbbre tesszük, de a tulajdonképpeni szöveg elé annyi szóközt írunk, hogy a megjegyzés látszólag külön sorban jelenjen meg a képernyőn:

```
13070 RETURN ' "
------ ATLAGSZAMITAS
14010 A=ϕ
```

Ez a módszer, noha nagyon látványos, nem elég kényelmes, különösen karbantartáskor.

De mit tegyünk a program elején lévő megjegyzésekkel? Nem sok választásunk van. Javasoljuk, hogy minden programot egy PRINT utasítással kezdjünk, amely tömören kiírja a program főbb adatait. Például:

```
1 PRINT "*** SZAMLAZO PROGRAM *** 83ϕ512 ***"
```

A megjegyzések használatára vonatkozóan elegendő példát találhat az Olvasó a könyvünkben elszórt mintaprogramokban, ahol a megjegyzéseket soha nem kódolva adtuk meg, hanem csak megjelöltük a helyüket, és megadtuk a tartalmukat. Ennek elsődleges oka, hogy azt szeretnénk, hogy az Olvasó ne a megjegyzéseinket másolja le, hanem a fenti alapelveket fogadja el. És ezek szellemében maga készítse el a saját megjegyzéseit, esetenként döntve el, hogy elfogadja-e a mi megjegyzéseinket, vagy más megjegyzéseket alkalmaz helyettük, netán akár mellettük is. Sőt arról is szabadon dönthet, hogy milyen formában kódolja a programjában a megjegyzéseket, ha egyáltalán alkalmazza.

Végezetül a legfontosabb szabály: a programunkban a lehető legkevesebb megjegyzés legyen. A megjegyzések nemcsak tárhelyet foglalnak el, hanem interpretálásuk is időt vesz igénybe. Ezen kívül veszélyesek is, mert a gép nem ellenőrzi őket, így nagyobb valószínűséggel fordulhatnak elő hibák bennük, mint a kódban, különösen a program többszöri módosítása után. A megtévesztő megjegyzés pedig rosszabb, mint a semmillyen. Programjainkban tehát csak a valóban szükséges megjegyzések legyenek. De azok viszont ne hiányozzanak.

A KÓD OPTIMALIZÁLÁSA

Soha ne optimalizáljunk hibás, még nem működő programot, csak a kész, működő változatot, és azt is csak akkor, ha elkerülhetetlen!

A programméret csökkentése:

– legtermészetesebb módszer a felesleges modulok, elemek, utasítások elhagyása.

– elhagyhatjuk a felesleges megjegyzéseket; a megmaradóakat lerövidíthetjük.

– elhagyhatjuk a felesleges kulcsszavakat, például az IF-utasításban THEN helyett írhatunk felsővesszőt.

– rövidebb lesz a program, ha több utasítást vonunk össze egy sorba.

A futási idő csökkentése:

– az utasítások egy sorba való összevonása már önmagában is eredményez némi időmegtakarítást. Ha például 4 egyszerű értékadó utasítást vonunk össze egy sorba, mintegy 1ϕ %-kal rövidebb idő alatt hajtódnak végre.

– sok időt nyerhetünk az esetszétválasztás megfelelő kódolásával. Ha 3 eset közül például a másodikat kell kiválasztanunk, 4ϕ % időmegtakarítást érünk el, ha IF helyett ON-t használunk. Ha sok a szétválasztandó esetek száma, a különbség még nagyobb.

– ha IF-eket használunk, igyekezzünk őket gyakorisági sorrendbe tenni, hogy lehetőleg minél kevesebbet kelljen végrehajtani közülük, mert a végrehajtási idő jelentősen függ attól, hogy hányadik kérdéssel találunk rá a keresett esetre.

– mintegy 3ϕ %-kal gyorsabban hajtódik végre az IF A THEN ... feltételvizsgálat az IF A<>ϕ THEN ... feltételvizsgálatnál.

– a GOSUB + RETURN végrehajtása gyorsabb a GOTO +GOTO-vissza utasításpárénál. Ha a vezérlésátadás távolsága kicsi, (azaz ha alig néhány utasítást kell átlépni,) akkor 5%, nagyobb távolság esetén 25% a különbség a GOSUB + RETURN javára.

– az IF A=B THEN ... végrehajtási ideje a kétszerese az IF A$=B$ THEN ... végrehajtási idejének.

– az A=123 végrehajtási ideje másfélszerese az A$="123" végrehajtási idejének.

– a logikai műveletek (NOT, AND, OR) nagyon lassúak.

– A↑5 lényegesen lassabb, mint A*A*A*A*A.

– a FOR + NEXT utasításpárral vezérelt ciklus háromszor-négyszer gyorsabb az IF + GOTO utasításpárral vezérelt ciklusnál.

A fenti adatok csak a leírt utasításokra vonatkoznak. Tehát mondjuk a legutóbbi példában, a FOR + NEXT utasításpárral vezérelt ciklus ciklusmagja természetesen ugyanannyi ideig fut, akárhogyan is vezéreljük; az időcsökkenés csak magára a ciklus-ismétlés megszervezésére fordított idő csökkenését jelenti.

Soha ne optimalizáljunk becslés, mindig csak mérés alapján!

A méréshez az is hozzátartozik, hogy például felgyorsítani nem azt a műveletet kell, amelyik a programfutás során csak egyszer hajtódik végre, hanem azt, amelyik többezerszer.

Befejezésül pedig: a program igazi optimalizálása nem az utasításonkénti vagy utasításcsoportonkénti csiszolgatás, hanem a tevékenységek rendszerének optimális megtervezése és az eleve megfontolt kódolás.

MINTAPÉLDÁK

ESETSZÉTVÁLASZTÁS
AZ AIRCOMP-16 ÉS A GYERMEK
AZ AIRCOMP-16 MINT FEJLESZTŐGÉP

A
– a
– a
– a
– a

Az
közép
Csı

formá
A
egésze
lehetn
A
együtt
öreg"
dig ga
másod
az esei
Nos
határo

A következőkben három mintapéldát mutatunk be, amelyek illusztrálják
– az AIRCOMP-16 felhasználási területeit,
– az AIRCOMP-16 BASIC nyelvének hatékonyságát,
– a korszerű programtervezés technikáját,
– a leírt kódolási elvek alkalmazását.

ESETSZÉTVÁLASZTÁS

Az esetszétválasztás szemléltetésére egy közismert példát mutatunk be, a középiskolában is tanult másodfokú egyenlet gyökeinek meghatározását.

Csupán emlékeztetőül, az ilyen egyenletek általános alakja

$$ax^2+bx+c=O$$

formában írható fel.

A példánk szempontjából elméletileg lényegtelen, hogy az együtthatók egészek vagy valósak, de a számítógépes megvalósítás miatt csak racionálisak lehetnek.

A megoldás szempontjából azonban nagyonis lényeges, hogy melyik együttható nulla, és melyik nem az. Ugyanis az együtthatóktól függően a "jó öreg" megoldóképlet nem mindig használható, más esetekben pedig nem mindig gazdaságos, a benne lévő "drága" műveletek miatt. Ezért ha mindenféle másodfokú egyenlet megoldására fel kell készülnünk, mindenképpen célszerű az eseteket szétválasztani.

Nos, egyenletünkben az együtthatók eloszlása elsődlegesen nyolc esetet határoz meg.

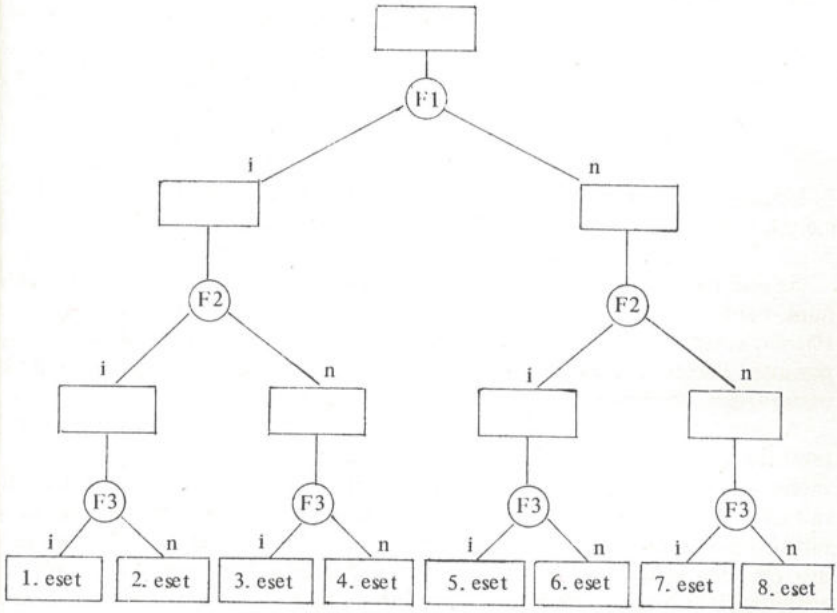
AZ EGYÜTTHATÓK ELOSZLÁSÁNAK TÁBLÁZATA

eset	együtthatók			megoldás
	a	b	c	
1	ϕ	ϕ	ϕ	az egyenletnek minden x megoldása
2	ϕ	ϕ	c	az egyenlet értelmetlen
3	ϕ	b	ϕ	az egyenlet nem másodfokú, $x = 0$
4	ϕ	b	c	az egyenlet nem másodfokú, $x = \frac{c}{b}$
5	a	ϕ	ϕ	$x1 = 0, x2 = 0$
6	a	ϕ	c	lásd (6.1) és (6.2)!
7	a	b	ϕ	$x1 = 0, x2 = \frac{b}{a}$
8	a	b	c	lásd (8.1) és (8.2) és (8.3)!

Egyelőre ne törődjünk azzal, hogy a (6) és a (8) esetek további aleseteket implikálnak, koncentráljunk csupán a nyolc fő eset szétválasztására!

Ez kétféleképpen történhet: vagy egymásbaágyazott, vagy sorozatos feltételekkel. Kérdés, hogy melyiket válasszuk.

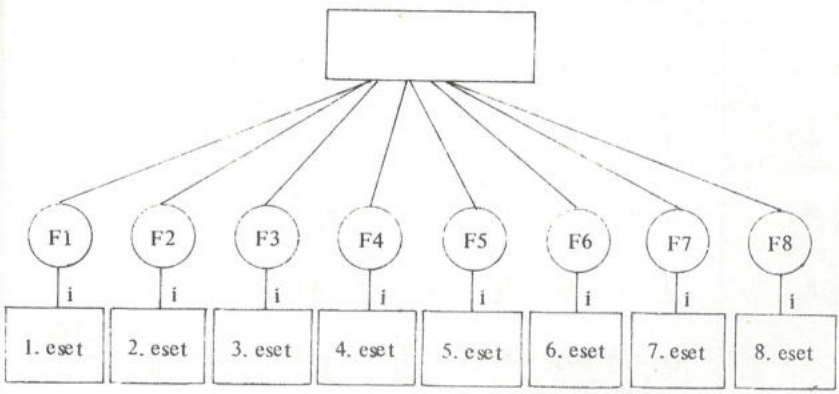
ESETSZÉTVÁLASZTÁS EGYMÁSBAÁGYAZOTT FELTÉTELEKKEL



Ahol a feltételek:

– F1: a=ϕ
– F2: b=ϕ
– F3: c=ϕ

ESETSZÉTVÁLASZTÁS SOROZATOS FELTÉTELEKKEL



Ahol a feltételek:

- F1: $a=\phi$ és $b=\phi$ és $c=\phi$
- F2: $a=\phi$ és $b=\phi$ és $c\neq\phi$
- F3: $a=\phi$ és $b\neq\phi$ és $c=\phi$
- F4: $a=\phi$ és $b\neq\phi$ és $c\neq\phi$
- F5: $a\neq\phi$ és $b=\phi$ és $c=\phi$
- F6: $a\neq\phi$ és $b=\phi$ és $c\neq\phi$
- F7: $a\neq\phi$ és $b\neq\phi$ és $c=\phi$
- F8: $a\neq\phi$ és $b\neq\phi$ és $c\neq\phi$

Megjegyezzük, hogy itt a feltételek egymást kölcsönösen kizárják, így bármelyik bekövetkezése esetén az összes többi vizsgálata elhagyható.

Az első megoldás előnye, hogy mindig csak egyszerű feltételeket kell kezelnünk benne. Az összetett feltételek kiértékelése ugyanis elég "drága" lehet. Hátránya viszont, hogy bármely eset egyértelmű meghatározásához mindig pontosan három döntést kell elvégeznünk. További hátrány, hogy a feltételek összefüggők, ezért cserélésük, módosításuk nehézkes.

A második megoldás előnye, hogy mind a feltételek, mind a tőlük függő tevékenységek egymástól függetlenek, tehát könnyen módosíthatók. Továbbá, az esetek szétválasztásához szükséges döntések száma 1 és 8 között változik, így több egyenlet megoldása esetén a vizsgálatok sorrendjének megváltoztatásával mindig elérhetjük, hogy a leggyakrabban előforduló eseteket már egyetlen vizsgálattal határozhassuk meg, feltéve, hogy ismerjük a várható adateloszlást. A megoldás hátránya azonban, hogy összetett feltételeket kell kezelnünk.

Most vegyük figyelembe az aleseteket is!

AZ ALESETEK

aleset	diszkrimináns	megoldás
6.1	$D=-\frac{c}{a}>0$	$x1=\sqrt{D}\quad x2=-x1$
6.2	$D=-\frac{c}{a}<0$	$x1=\sqrt{\lvert D\rvert}\ \text{i}\quad x2=-x1$
8.1	$D=b^2-4ac>0$	$x1=\frac{-b+\sqrt{D}}{2a}\quad x2=\frac{-b-\sqrt{D}}{2a}$
8.2	$D=b^2-4ac=0$	$x1=\frac{-b}{2a}\quad x2=x1$
8.3	$D=b^2-4ac<0$	$x1=\frac{-b}{2a}+\frac{\sqrt{\lvert D\rvert}}{2a}\ \text{i}\quad x2=\frac{-b}{2a}-\frac{\sqrt{\lvert D\rvert}}{2a}\ \text{i}$

Látható, hogy az egyenletnek a (6.2) esetben imaginárius, a (8.3) esetben komplex gyöke van. Minden más esetben a gyökök (ha vannak) valósak.

Az alesetek további esetszétválasztást igényelnek. Itt is bármelyik módszert alkalmazhatjuk a fent bemutatott kettő közül. Minthogy összetett feltételeket nem kell kezelnünk, egyértelműen a második megoldás a jobb.

A továbbiakban be fogjuk mutatni mind a két megoldást, de a következetesség kedvéért az alesetek szétválasztásához mindkét esetben ugyanazt a módszert fogjuk alkalmazni, amelyiket a megfelelő főesetek meghatározásához is használtunk.

Ha az algoritmusokat ki akarjuk próbálni, a következő teszteseteket és tesztadatokat ajánljuk:

A TESZTESETEK TÁBLÁZATA

teszteset	együtthatók			eset	eredmény	
	a	b	c			
I.	ϕ	ϕ	ϕ	(1)	az egyenletnek minden x megoldása	
II.	ϕ	ϕ	1	(2)	az egyenlet értelmetlen	
III.	ϕ	1	ϕ	(3)	az egyenlet nem másodfokú	
IV.	ϕ	1	1	(4)	az egyenlet nem másodfokú	
V.	1	ϕ	ϕ	(5)	x1 = ϕ	x2 = ϕ
VI.	1	ϕ	–4	(6.1)	x1 = 2	x2 = –2
VII.	1	ϕ	9	(6.2)	x1 = 3i	x2 = 3i
VIII.	1	5	ϕ	(7)	x1 = ϕ	x2 = –5
IX.	2	–4	–6	(8.1)	x1 = 3	x2 = –1
X.	1	8	16	(8.2)	x1 = –4	x2 = –4
XI.	1	–8	17	(8.3)	x1 = 4+1i	x2 = 4–1i

Most pedig következzék az esetszétválasztás az egymásbaágyazott feltételek segítségével!

A programrészletbe csupa kisbetűvel gépelt megjegyzéseket is elhelyezünk, de csak ott, ahol véleményünk szerint a kód magyarázatra szorul. A kód

folyamatos olvashatóságát megzavaró megjegyzéseinket a program végén közöljük.

A program egyes szerkezeti egységei között a jobb olvashatóság érdekében üres sorokat hagytunk.

```
----- esetszétválasztás egymásbaágyazott feltételekkel -----
----
 510  PRINT "=================================="
----
1010  IF A <> φ THEN GOTO 5010
1020  IF B <> φ THEN GOTO 3010
1030  IF C <> φ THEN GOTO 2010
1040  PRINT "ESET=1 AZ EGYENLETNEK MINDEN X MEGOLDASA."
1050  GOTO 9510
----
2010  PRINT "ESET=2 AZ EGYENLET ERTELMETLEN."
2020  GOTO 9510
----
3010  IF C <> φ THEN GOTO4010
3020  PRINT "ESET=3 AZ EGYENLET NEM MASODFOKU."
3030  GOTO 9510
----
4010  PRINT "ESET=4 AZ EGYENLET NEM MASODFOKU."
4020  GOTO 9510
----
5010  IF B <> φ THEN GOTO 7010
5020  IF C <> φ THEN GOTO 6010
5030  PRINT "ESET=5 X1=φ, X2=φ"
5040  GOTO 9510
----
6010  D=-C/A
6020  IF D<0 THEN GOTO 6210
----
6110  XE=SQR(D):XK=-XE
6120  PRINT "ESET=6.1 X1=";XE;", X2=";XK
6130  GOTO 9510
----
6210  IE=SQR(ABS(D)) :IK=-IE
6220  PRINT "ESET=6.2 X1=(";IE;")I, X2=(";IK;")I"
6230  GOTO 9510
-----
```

```
7010    IF C <> φ THEN GOTO 8010
7020    XK=B/A
7030    PRINT "ESET=7 X1=φ, X2=";XK
7040    GOTO 9510
----
8010    D=B*B-4*A*C :AK=2*A
8020    IF D <= φ THEN GOTO 8210
----
8110    SD=SQR(D)
8120    XE=(-B+SD)/AK :XK=(-B-SD)/AK
8130    PRINT "ESET=8.1 X1=";XE;", X2=";XK
8140    GOTO 9510
----
8210    IF D < φ THEN GOTO 8310
8220    XE=-B/AK :XK=XE
8230    PRINT "ESET=8.2 X1=";XE;", X2=";XK
8240    GOTO 9510
----
8310    SD=SQR(ABS(D))
8320    VE=-B/AK :VK=VE :IE=SD/AK :IK=-IE
8330    PRINT "ESET=8.3"
8340    PRINT "X1=(";VE;")+(";IE;")I"
8350    PRINT "X2=(";VK;")+(";IK;")I"
8360    GOTO 9510
----
----    esetszétválasztás vége
9510    a program folytatása
```

A programban több műveletet, illetve munkaváltozót megtakaríthattunk volna a PRINT utasítások alkalmas megválasztásával. Például ha a 8230 sort

```
8230 PRINT "ESET=8.2 X1=";XE;", X2=";XE
```

alakban kódoljuk, akkor nincs szükségünk az XK változóra, és az XK=XE műveletre.

Az ilyesfajta "optimalizálás" azonban nagyon rontja a program áttekinthetőségét, és nagyonis kérdéses, hogy előnyei arányban állnak-e azzal a kárral amit okoz.

Nem zavaró azonban a 2*A szorzat "kiemelése" a 8000 sorszámmal kezdődő utasítások moduljának az elejére.

Megjegyezzük, hogy a 9510 utasítás az esetszétválasztás végét jelenti.

Most pedig lássuk összehasonlításul az esetszétválasztást a feltételsorozat segítségével!

```
---- esetszétválasztás feltételsorozattal ----
----
 510  PRINT "=====================================";
 511  ES=φ :XE$="" :XK$="" :UZ$=""
 521  IF A=φ   AND B=φ   AND C=φ   THEN GOTO 1010
 522  IF A=φ   AND B=φ   AND C<>φ  THEN GOTO 2010
 523  IF A=φ   AND B<>φ  AND C=φ   THEN GOTO 3010
 524  IF A=φ   AND B<>φ  AND C<>φ  THEN GOTO 4010
 525  IF A<>φ  AND B=φ   AND C=φ   THEN GOTO 5010
 526  IF A<>φ  AND B=φ   AND C<>φ  THEN GOTO 6010
 527  IF A<>φ  AND B<>φ  AND C=φ   THEN GOTO 7010
 528  IF A<>φ  AND B<>φ  AND C<>φ  THEN GOTO 8010
 530  PRINT "EZ LEHETETLEN ESET!"
 540  GOTO 9510
----
1010  ES=1 :UZ$="AZ EGYENLETNEK MINDEN X MEGOLDASA."
1020  GOTO 9010
----
2010  ES=2 :UZ$="AZ EGYENLET ERTELMETLEN."
2020  GOTO 9010
----
3010  ES=3 :UZ$="AZ EGYENLET NEM MASODFOKU."
3020  GOTO 9010
----
4010  ES=4 :UZ$="AZ EGYENLET NEM MASODFOKU."
4020  GOTO 9010
----
5010  ES=5 :XE$="φ" :XK$="φ"
5020  GOTO 9010
----
6010  D=-C/A
6020  IF D>φ THEN GOTO 6110
6030  IF D<φ THEN GOTO 6210
6040  PRINT "EZ LEHETETLEN ESET!"
6050  GOTO 9510
----
6110  ES=6.1 :XE=SQR(D) :XK=-XE
6120  XE$=STR$(XE) :XK$=STR$(XK)
6130  GOTO 9010
----
```

```
6210    ES=6.2 :IE=SQR(ABS(D)) :IK=-IE
6220    XE$="("+STR$(IE)+")I"
6230    XK$="("+STR$(IK)+")I"
6240    GOTO 9010
7010    ES=7 :XK=-B/A
7020    XE$="φ" :XK$=STR$(XK)
7030    GOTO 9010
8010    D=B*B-4*A*C :AK=2*A
8020    IF D>φ THEN GOTO 8110
8030    IF D=φ THEN GOTO 8210
8040    IF D<φ THEN GOTO 8310
8050    PRINT "EZ LEHETETLEN ESET!"
8060    GOTO 9510
8110    ES=8.1 :SD=SQR(D)
8120    XE=(-B+SD)/AK :XK=(-B-SD)/AK
8130    XE$=STR$(XE) :XK$=STR$(XK)
8140    GOTO 9010
8210    ES=8.2 :XE=-B/AK
8220    XE$=STR$(XE) :XK$=XE$
8230    GOTO 9010
8310    ES=8.3 :SD=SQR(ABS(D))
8320    VE=-B/AK :VK=VE :IE=SD/AK :IK=-IE
8330    XE$="("+STR$(VE)+")+("+STR$(IE)+")I"
8340    XK$="("+STR$(VK)+")+("+STR$(IK)+")I"
8350    GOTO 9010
```

---- esetszétválasztás vége ----

Vegyük észre, hogy szándékosan úgynevezett kimerítő vizsgálatsorozatokat kódoltunk, vagyis az összes lehetséges esetre külön-külön rákérdeztünk, így lehetővé téve a lehetetlen esetek figyelembevételét is. Ennek a program biztonságosságán túlmenően dokumentatív szerepe is van. Lehet, hogy ez a szerep itt nem látszik jelentősnek, de más programokban nagyonis fontos lehet, ezért jobb, ha már most hozzászokunk.

Ez a programrészlet egyébként abban is eltér az előzőtől, hogy az eredmények megjelenítését kiemeltük az esetszétválasztásból, ahol a tényleges számítások elvégzésén kívül csak a kiírások előkészítése maradt meg.

```
----- esetszétválasztás vége -----
9010  PRINT "ESET=";ES
9020  PRINT UZ$
9030  IF XE$="" AND XK$="" THEN GOTO 9510
9040  PRINT "X1=";XE$ :PRINT "X2=";XK$
----- nyomtatás vége -----
```

Az I/O műveleteknek egyetlen modulba való csoportosítása egyrészt logikus, mivel az eredmények kiírása mindig valamilyen feldolgozási menet lezáró, befejező tevékenysége, így jobb ha nem keveredik össze a feldolgozást magát végrehajtó tevékenységekkel, hanem követi azokat; másrészt hasznos, mert így az I/O tevékenységek és azok utasításai lényegesen könnyebben karbantarthatók, módosíthatók, az íráskép sokkal egyszerűbben megváltoztatható. Ugyanakkor hátrányosan érinti a programozót, hogy ilyenkor többnyire szükség van kiírásokat előkészítő tevékenységekre, amelyek esetenként bonyolultak is lehetnek.

Ha visszalapozunk az előző megoldáshoz, láthatjuk, hogy ott a PRINT utasítások a program területén szét vannak szórva. Ennek előnye, hogy rendkívül rugalmas kiírás valósítható meg, a sajátos egyedi esetek könnyedén, a többitől függetlenül elintézhetők. Viszont egy esetleges módosítás igen kényelmetlen lehet, hiszen előbb meg kell találnunk, hogy melyPRINT utasítást vagy utasításokat kell megváltoztatnunk, majd még külön ellenőriznünk kell, hogy e változtatásnak milyen kihatása lehet az összes többi kiírásra.

Mindazonáltal kultúrált írásképet létrehozni, bonyolult kiírást végrehajtani biztonságosan (ismételjük, biztonságosan!) szinte kizárólag csak külön modulba szervezett I/O utasításokkal lehet.

Ha a fenti algoritmusokat ki szeretnénk próbálni, akkor az alábbi programba ágyazzuk be:

```
****  másodfokú program ****
----
 110  PRINT "MEGOLDHATOK ONNEK EGY"
 120  PRINT "   A*X↑2+B*X+C=φ"
 130  PRINT "ALAKU MASODFOKU EGYENLETET? "
 140  INPUT "VALASZ =I/N=";VA$
 150  IF VA$<>"I" AND VA$<>"N" THEN GOTO 140
 160  IF VA$="N" THEN GOTO 9910
 170  PRINT "KEREM AZ EGYUTTHATOKAT!"
 180  INPUT "A=";A
```

```
190     INPUT "B=";B
200     INPUT "C=";C
----
----    esetszétválasztás ----
----
----    esetszétválasztás vége ----
----
9510    PRINT "=====================================
9520    GOTO 110
----
9910    PRINT "SEBAJ, MAJD LEGKOZELEBB!"
9920    PRINT "VISZONTLATASRA!"
9930    END
```

Vigyázat! A program nincs felkészítve a hibás adatok feldolgozására, tehát az együtthatókat különös gondossággal adjuk meg. Természetesen a hibavizsgálat az előkészítő részbe könnyedén beépíthető; mi csak azért nem tettük meg, mert nem kívántuk elterelni az Olvasó figyelmét az esetszétválasztás algoritmusáról.

Kérdés, hogy milyen hibás adat fordulhat elő az együtthatók megadásakor. Legfőképpen az, hogy elgépeljük az adatot: például 123 helyett 213-at írunk. Ez ellen a program nem tud védekezni. Furcsább eset áll elő, ha nem numerikus adatot gépelünk be, például azért, mert véletlenül rákönyökölünk az SH gombra. Ha mondjuk a "C=? " kérdésre 125 helyett 12% választ adunk, a C együttható értéke 12 lesz, és a gép semmilyen hibát nem jelez. A program pedig egészen más egyenletet fog megoldani.

Megjegyezzük, hogy a fenti programok gazdaságosságára vonatkozó korábbi elméleti megfontolásainkat méréseink a várakozásnak megfelelően jól igazolták. Az egymásbaágyazott feltételekkel megírt programban az esetszétválasztás végrehajtási ideje gyakorlatilag konstans volt. Magának a programnak a sebessége azonban nem volt állandó. Azokban a tesztesetekben, amelyekben gyökvonást is végre kellett hajtani, a végrehajtási idő durván 60%-kal hosszabb volt, mint azokban, amelyekben ilyen tevékenységre nem volt szükség. Ez utóbbi jelenség megfigyelhető volt a sorozatos feltételekkel megírt program esetén is. Ott azonban az esetszétválasztásra fordított idő nem volt állandó; ugyanannak az egyenletnek a megoldása mintegy 20%-kal kevesebb időt vett igénybe, ha az esetet a legelső lépésben sikerült kiválasztani, mint amikor ez csak a legutolsó vizsgálatra sikerült. Látszik tehát, hogy a jól megszerkesztett feltételsorozat nemcsak áttekinthetőbb és könnyebben karbantartható, hanem gazdaságosabb is lehet.

A feltételsorozatos változat végrehajtási ideje (azonos tesztesetek és megtalálási feltételek mellett) mégis mintegy 80%-kal volt hosszabb, mint az első változaté. Ennek legfeljebb a negyede származott az összetett feltételek használatából. A programot a kiírás átszervezése, és az ebből fakadó előkészítő tevékenységek beépítése fékezte le. Különösen a karakterláncok kezelése, az üzenetek áttöltése és az eredmények konvertálása volt drága.

Nos, ez az ára a kultúrált kiírásnak, az áttekinthető, korszerű kódolásnak. Ennyibe kerül, hogy ha a programunkat mondjuk fél év múlva módosítanunk kell, nem lesz szükségünk arra, hogy napokat töltsünk el a módosítással és az újratesztelléssel, sőt ami még rosszabb lenne, hogy újra kelljen írnunk az egész programot, csak azért, mert nem igazodunk ki rajta. Kérdés, hogy nem fizetünk-e túl magas árat ezért? Ha tekintetbe vesszük, hogy a program ebben a formájában 0,3–0,7 másodperc alatt oldott meg egy másodfokú egyenletet, kiírással együtt, akkor azt mondhatjuk, hogy megéri. Ugyanis ha nem kell rendszeresen többezer másodfokú egyenletet megoldanunk, ez a sebesség tökéletesen kielégítő.

AZ AIRCOMP-16 ÉS A GYERMEKEK

Személyi számítógépünk kiválóan alkalmas arra, hogy elszórakoztassuk vele gyermekeinket, egyben támogatva iskolai tanulásukat.

A gyerekek ösztönösen érdeklődnek a technika csodái iránt, így kiváló lehetőségünk adódik a számítástechnikai kultúra terjesztésére, szinte már attól a perctől kezdve, hogy a gyerek már tud írni és olvasni. Ha engedjük érdekfeszítő és tanulságos programokkal játszani, akkor amellett, hogy elősegítjük iskolai tanulását, felkelthetjük érdeklődését a programozás iránt, amivel elérhetjük, hogy viszonylag korán megtanulja gondolatait összerendezetten, logikusan megfogalmazni.

Az életkori sajátosságoktól és az iskolai tananyagtól függően számos jól programozható témát találhatunk. Itt most csak egyet mutatunk be, elsősorban azért, mert talán ez az, amit a legkorábban használhatunk. Ugyanis olyan programról van szó, amely a szorzótábla gyakorlásához teremt lehetőséget és kedvet. Ezt pedig már másodikos gyerekek használhatják.

A program olyan kérdéseket tesz fel a gyereknek, mint "MENNYI 3.2=?" vagy "MENNYI 9.7=?" stb. A kérdésre begépelt választ a program összehasonlítja a helyes eredménnyel, és informálja a gyermeket arról, hogy válasza helyes volt-e avagy rossz. Természetesen a program a helyes választ is megadja, hogy ha a gyerek mégegyszer ugyanazt a kérdést kapja, legközelebb már ne rontsa el. Ezen kívül a program a kérdésekből kérdéssorozatokat képez, és minden sorozat után értékeli a gyerek teljesítményét. Attól függően, hogy

hány helytelen válasz volt, más és más bíztató üzenetet ír ki a képernyőre. Hibátlan sorozat esetén megdicséri a gyereket. Ime:

```
****    szorzótábla program ****
----
----    deklarációk
1010    N=5
1020    K$="-" :FOR I=1 TO 36 :K$=K$+"-" :NEXT I
1030    E$="=" :FOR I=1 TO 36 :E$=E$+"=" :NEXT I
1040    M$="  "
----    előkészítés
2010    PRINT M$;"TUDOD MAR A SZORZOTABLAT? "
2020    PRINT M$;"VALASZOLJ! I VAGY N";
2030    INPUT V$
2040    IF V$<>"I" AND VS<>"N" THEN GOTO 2020
2050    PRINT     :PRINT
2060    PRINT M$;"AKKOR MOST  ";
2070    IF V$="I" THEN GOTO 2100
2080    PRINT "GYAKOROLHATOD."
2090    GOTO 2110
2100    PRINT "KIPROBALHATOD."
2110    PRINT     :PRINT
----    játék a szorzótáblával
3010    PRINT M$;"VALASZOLJ A KOVETKEZO KERDESEKRE:"
3020    PRINT M$;K$
3030    GOSUB 5010 === kérdések feltétele, feleletek fogadása
3040    PRINT M$;K$
3050    GOSUB 6010 === kérdéssorozat kiértékelése
3060    PRINT M$;E$
3070    PRINT     :PRINT
----    befejezés
4010    PRINT M$;"OHAJT MEG VALAKI JATSZANI? "
4020    PRINT M$;"VALASZOLJ! I VAGY N";
4030    INPUT V$
4040    IF V$<>"I" AND V$<>"N" THEN GOTO 4020
4050    IF V$="N" THEN GOTO 4080
4060    FOR I=1 TO 3ϕ :PRINT     :NEXT I
4070    GOTO 2010
4080    PRINT     :PRINT M$;"VISZONTLATASRA!"
4090    END
----
```

```
----  kérdések feltétele, feleletek fogadása
5010  H=ϕ
5020  FOR I=1 TO N
5030  R=RND(1ϕϕ) :R$=STR$(R)
5040  W$=RGH$(R$,2)
5050  A$=LFT$(W$,1)
5060  B$=RGH$(W$,1)
5070  IF A$<"2" THEN GOTO 5030
5080  IF B$="ϕ" THEN GOTO 5030
5090  A=VAL(A$) :B=VAL(B$)
5100  X=A*B :X$=RGH$("  "+STR$(X),2)
5110  PRINT M$;"MENNYI";M$;A$;".";B$;"=";
5120  INPUT Y$ :Y$=RGH$("  "+Y$,2)
5130  PRINT M$;"          ";M$;A$;".";B$;"=";X;
5140  IF X$<"1ϕ" THEN PRINT " ";
5150  IF X$=Y$ THEN GOTO 5180
5160  PRINT " --VALASZOD ROSSZ--"
5170  GOTO 5200
5180  PRINT " ++VALASZOD HELYES++"
5190  H=H+1
5200  PRINT
5210  NEXT I
5220  RETURN
----
----  kérdéssorozat kiértékelése
6010  ON H 'GOTO 6110 'GOTO 6210 'GOTO 6310 'GOTO 6410 'GOTO 6510
6020  PRINT M$;"NE KOMOLYTALANKODJ!"
6030  GOTO 6610
----  elégtelen
6110  PRINT M$;"EZ BIZONY NAGYON SZOMORU!"
6120  GOTO 6610
----  elégséges
6210  PRINT M$;"MEG SOKAT KELL GYAKOROLNOD!"
6220  GOTO 6610
----  közepes
6310  PRINT M$;"SZORGALMASAN GYAKOROLJ TOVABB!"
6320  GOTO 6610
----  jó
6410  PRINT M$;"EZ MAR MAJDNEM KIVALO!"
6420  GOTO 6610
----  jeles
```

```
6510    PRINT M$;"HOGY HIVNAK";
6520    INPUT T$
6530    FOR I=1 TO 3ϕ
6540    PRINT M$;"GRATULALOK";
6550    PRINT M$;T$;
6560    PRINT M$;"OKOS VAGY"
6570    NEXT I
----    kiértékelés vége
6610    RETURN
```

Néhány megjegyzés a programhoz:

– Az M$ változó egyetlen szerepe, hogy a kiírás ne kezdődjön közvetlenül a képernyő szélén, ahol nehezen olvasható, különösen ha a TV nincs gondosan beállítva.

– A program 5 kérdést tesz fel egy sorozatban a szorzótáblából. A kérdések számát az 1010 N=5 utasítás határozza meg. Átállításával a kérdések száma tetszőlegesen változtatható.

– A 2010 PRINT M$;"TUDOD MAR A SZORZOTABLAT?" kérdést csak bemelegítés céljából teszi fel a program, hogy lélegzetvételnyi időt biztosítson a feladatra való ráhangolódásra.

– A szorzótényezők kiválasztásához az 5030 R=RND(1ϕϕ) utasítás generál egy véletlenszámot, amelynek utolsó számjegye a szorzó, az utolsó előtti pedig a szorzandó. Ebből következik, hogy egyikük sem lehet nagyobb 9-nél.

'GOTO 6 – A feladat nehezítése céljából a szorzandó nem lehet kisebb mint kettő, a szorzó pedig nem lehet nulla. Ha ez mégis bekövetkezne, a program új szorótényezőket generál.

– A program a "MENNYI A.B=?" kérdésre begépelt válasz utolsó két karakterét értékeli. A hibás begépelés tehát egyszerűen javítható; csupán legalább egy szóközt kell leütni, majd utána megadható a helyesnek vélt válasz.

– Az 5140 IF X$<"1ϕ" THEN PRINT " "; utasításra csak azért van szükség, hogy a választ minősítő üzenetek pontosan egymás alá essenek, akár egyjegyű, akár kétjegyű a szorzat.

– A program a helyes válaszokat külön modulban értékeli ki, ami azért is célszerű, hogy a kiértékelési szempontok kívánságra könnyen megváltoztathatók legyenek.

– A minősítés csak 5 kérdésből álló sorozatokra működik. Ha a kérdések számát megnöveljük, akkor vagy teljesen újra írjuk a kiértékelő modult, vagy módosítanunk kell azt. Az utóbbi esetben, ha minimális módosításra törekszünk, két lehetőségünk is van:

Az egyik az, hogy a 6010 ON H... utasításba újabb GOTO utasításokat építünk be. Például tíz kérdés esetén:

```
6010 ON H'GOTO 6110 'GOTO 6110 'GOTO 6210 'GOTO 6210
        'GOTO 6310 'GOTO 6310 'GOTO 6310 'GOTO 6410
        'GOTO 6410 'GOTO 6510
```

A második lehetőség választása esetén százalékos kiértékelést valósíthatunk meg. Például így:

```
6010 S=H*1ϕϕ/N
6011 IF S>99 THEN GOTO 6510
6012 IF S>8ϕ THEN GOTO 6410
6013 IF S>6ϕ THEN GOTO 6310
6014 IF S>4ϕ THEN GOTO 6210
6015 IF S> ϕ THEN GOTO 6110
```

Ez nem függ a kérdések számától.

A kérdések számának csökkentését nem javasoljuk.

– A 6530 FOR I=1 TO 3ϕ utasítás által szervezett ciklus szerepe az, hogy miután a program (tökéletes feladatmegoldás esetén) bekérte a válaszadó nevét, a képernyőt teleírhassa dícsérő szavakkal.

– A program minden feladatmegoldó menet végén megkérdezi, hogy óhajt-e még valaki tovább játszani. Ha a válasz nemleges, a program udvariasan elköszön. Igenlő válasz esetén a 4060 FOR I=1 TO 3ϕ :PRINT :NEXT I utasításokkal kifuttatja a képernyőről a rajta lévő szöveget, majd tiszta képernyővel újabb játszmára készen áll. (Ennek célja, hogy a képernyőn bentmaradó régi kérdések ne zavarják a gyereket az új válaszok megadásában.)

– Fokozhatjuk gyermekeink örömét, ha a programhoz hanggeneráló modult írunk:

```
----  hanggenerálás
7010  L=LEN(T$)
7020  IF L>14 THEN L=14
7030  FOR J=1 TO L
7040  D=ASC(MID$(T$,J,1))
7050  IF D<65 OR D>9ϕ THEN D=77
7060  POKE 16384,132,29
7070  F=USR(256*D+D/3)
7080  NEXT J
7090  RETURN
----  hanggenerálás vége
```

Ez esetben az eredeti programunkat is módosítanunk kell, mégpedig az alábbi utasításokkal:

6530 FOR I=1 TO 5
6565 GOSUB 7010 === hanggenerálás

Ennek a kis modulnak a hatása az lesz, hogy a program a válaszadó gyerek nevéből egy kellemes hangzású dallamot képez (különböző nevekre természetesen különbözőt), és a gép ezt a dícsérő szöveg kinyomtatása közben elfütyüli. Ebben a jutalomban csak azok részesülhetnek, akik a sorozat minden kérdésére hibátlanul válaszoltak.

Megjegyezzük, hogy a programot egy általános iskola második osztályában, szakkör keretében kipróbáltuk, és hatalmas sikert aratott. Egy kicsit magunk is megdöbbentünk azon a hatáson, amit a gyerekekből kiváltott.

Ez a fejezet

a

SZORZÓTÁBLA

programot használó

GYEREKEKNEK

szól!

Olvassátok el figyelmesen, mielőtt a géphez nyúltok!

Ha
váncs
aki é
kozó

A
Sc
Ne
Ke
zettá
Ug
A
ad. A
Ha
villog
Ne
Me
CR g
Ek
Je
képe
áll. Il
Ha
bizon
nyom
az N
Az
befeje
Ha
nik s
helye
Ez

A vill

Ha már másodikos vagy, és szeretnéd gyakorolni a szorzótáblát, vagy kíváncsi vagy, hogy vajon már jól tudod-e, akkor kérj meg egy olyan felnőttet, aki ért a számítógép kezeléséhez, hogy kapcsolja be neked, a géphez csatlakozó TV-vel és magnóval együtt.

VIGYÁZZ!

A gépet mindig csak felnőttek jelenlétében használd!
Soha ne kapcsold be egyedül!
Ne nyúlj a TV-hez, a magnóhoz, és a konnektorhoz!

Kérd meg azt a felnőttet, aki bekapcsolta neked a gépet, hogy a magnókazettádról olvastassa be a "SZORZÓTABLA" programot.

Ugye van egy saját kazettád, amin a programjaidat tartod?

A program beolvasásakor a magnó elég hangos és kellemetlen sípoló hangot ad. A képernyő is elsötétül. De ez nem baj.

Ha a képernyőn megjelenik az OK felirat, és alatta egy kis négyzet alakú jel villog, ez azt jelenti, hogy minden rendben van.

Ne felejtsétek el leállítani a magnót!

Most elindíthatod a programot. Nyomd meg először a RUN, majd utána a CR gombot. Ezeket ott találod a jobb kezednél.

Ekkor a gép megkérdezi, hogy tudod-e már a szorzótáblát.

Jegyezd meg, hogy ha a gép kérdez, azt onnan lehet megismerni, hogy a képernyőn látható szöveg végén kérdőjel van, és a villogó jel a kérdőjel után áll. Ilyenkor a gép türelmesen várja a válaszodat.

Ha úgy érzed, hogy tudod a szorzótáblát, nyomd meg az I betűt. Ha bizonytalan vagy, akkor az N betűt nyomd meg. Akármilyen más betűt nyomsz meg, a program nem fogja elfogadni, mert az I azt jelenti, hogy igen, az N pedig azt, hogy nem. Ezután nyomd meg a CR gombot.

Azt is jegyezd meg, hogy amikor a gép kérdésére válaszolsz, a válaszod befejezése után mindig meg kell nyomnod a CR gombot.

Ha véletlenül rossz betűt nyomtál meg, például N helyett M-et, nem történik semmi baj, a program újra felteszi a kérdést, és most már válaszolhatsz helyesen.

Ezután a gép kérdéseket fog kiírni a képernyőre. Például ilyeneket:

MENNYI 5·4= ?

A villogó jel a kérdőjel mögött áll, a gép várja a válaszodat.

Gépeld be, hogy szerinted mennyi ötször négy. A számokat a legfelső sorban találod. A nulla a 9 után következik, és úgy jelölik, hogy ϕ, azért hogy ne lehessen összetéveszteni az O betűvel. Ezután nyomd meg a CR gombot.

Közben a gép is kiszámolja, hogy szerinte mennyi ötször négy, és az eredményt kiírja a képernyőre. Így:

$5 \cdot 4 = 2\phi$

Ha ugyanezt válaszoltad te is, a gép kiírja, hogy a válaszod helyes. Ha nem ezt válaszoltad, akkor bizony a válaszod rossz. Jegyezd meg azt az eredményt, amit a gép kiírt, hogyha legközelebb ugyanezt kérdezi, helyesen tudjál majd válaszolni.

Ezután a gép még több ilyen kérdést fog neked feltenni, persze általában nem ugyanezt. Válaszolj ezekre is ugyanilyen módon.

Ha egy kérdésre nem tudsz válaszolni, akkor nyomd meg a ϕ gombot, vagy találomra gépelj be egy tetszőleges számot. Ezután nyomd meg a CR gombot.

Mindig figyeld, hogy mit válaszol a gép, mert tanulhatsz tőle. Jegyezd meg, hogy a gép soha nem téved. Ő tudja a szorzótáblát. A kérdésekre lehetőleg gyorsan válaszolj. Ha sokat töprengsz, lassan tanulsz.

Ha egy kérdésre véletlenül rosszul válaszoltál, például nem azt a gombot nyomtad meg, amelyiket akartad, de a CR gombot még nem nyomtad meg, akkor a válaszodat még kijavíthatod. Ezt úgy csináld, hogy nyomd meg a szóköz billentyűt. Ez az a hosszú, amelyikbe nincs írva semmi, és ott van középen legalul, a hasadnál. Legalább egyszer kell megnyomnod, de megnyomhatod többször is. Ezután gépeld be a helyes választ. És most nyomd meg a CR gombot.

A program összesen öt szorzást kérdez, ezután minősíti az eredményedet. Olvasd el, hogy mit írt ki, és szívleld meg.

Ha a gép megkérdezi, hogy hogyan hívnak, akkor gépeld be a nevedet. Ha kész, nyomd meg a CR gombot. Kellemes meglepetésben lesz részed.

Ha a gép megkérdezi, hogy óhajt-e még valaki játszani, gondold meg, hogy igen vagy nem. Ha igen, nyomd meg az I betűt, ha nem, akkor az N betűt. Ezután nyomd meg a CR gombot.

Ha I, azaz igen választ adtál, a program újra kezdi a játékot, és ismét felteszi a kérdéseit. Válaszolj rájuk ugyanúgy, mint eddig.

Ha N, azaz nem választ adtál, a program elbúcsúzik, majd leáll. Ekkor megjelenik az OK felirat, és alatta a villogó jel.

Ha később meggondolod magad, és ismét játszani szeretnél, egyszerűen elindíthatod a programot: nyomd meg a RUN, és utána a CR gombot.

Ha többen vagytok másodikosok, rendezhettek versenyt is. Ki tud rövidebb idő alatt több helyes választ adni? Ki lesz a szorzótábla bajnoka?

Végül néhány jótanács:

– Mielőtt a géphez nyúlsz, mossál kezet. A gombok nem szeretik a maszatos ujjakat.

– Ne egyél a gép mellett, mert köszöni szépen, de nem kér sem zsíros, sem vajas kenyeret, de még tejet vagy málnaszörpöt sem. Nincs rá szüksége. Kizárólag áramot fogyaszt. Abból se sokat.

– Használd bátran a gépet, mert ha nem csinálsz vele mást, csak a gombokat nyomogatod, akkor nem tudod elrontani. Ne ijedj meg, ha rossz gombot nyomtál meg, ettől a gép nem megy tönkre. Csak szándékos rosszasággal tudsz kárt okozni.

– Vigyázz a gépre, ne lökd le az asztalról, ne tenyerelj vagy könyökölj rá, ne üsd meg, ne önts rá semmilyen folyadékot.

– A gombokat finoman nyomd meg, és figyeld a képernyőt. Ha ott megjelenik az a betű vagy szám, aminek a gombját megnyomtad, akkor jó. Ha nem, nyomd meg a gombot mégegyszer, most már kicsit erősebben.

– A gombokat határozottan, egyetlen mozdulattal, röviden nyomd meg, mintha az ujjaddal át akarnád szúrni az iskolatej tetejét. Ha sokáig tartod például az I betűn az ujjadat, akkor nem azt fogja a gép kiírni, hogy I, hanem azt hogy, IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII egészen addig, amíg az ujjadat le nem veszed a gombról. Hasonló történik, ha remeg a kezed. Ne félj a gomboktól, akármelyiket nyomod is meg, nem fog elromlani a gép.

– Egyszerre csak egy gombot nyomjál meg!

– Ha a játék közben a gép nem úgy viselkedik, ahogy szokott, például a képernyőn furcsa, eddig soha nem látott jelek jelennek meg, vagy a képernyő elsötétül, vagy akármit nyomsz meg, nem történik semmi, vagy a program nem azt kérdezi vagy válaszolja amit kellene, vagy egészen mást csinál, vagy nem az történik, aminek a fenti leírás szerint történnie kellene, vagy nem abban a sorrendben jönnek a kérdések, amilyenben szoktak, és egyátalán, ha bármilyen furcsát és szokatlant tapasztalsz, azonnal szólj a legközelebbi felnőttnek, és mondj el mindent, amit észleltél. Ne tagadj le semmit, mert majdnem biztos, hogy olyan hibáról van szó, amit nem te okoztál, amiről nem te tehetsz.

BÚCSÚZÓUL JÓ SZÓRAKOZÁST ÉS JÓ TANULÁST KIVÁNUNK!

AZ AIRCOMP MINT FEJLESZTŐGÉP

Feltételezzük, hogy az AIRCOMP tulajdonosok és felhasználók egy része nemcsak saját maga szórakoztatására ír programokat, hanem más felhasználók részére is programoz, programozott, vagy programozni fog, éspedig általában az AIRCOMP-tól eltérő gépeken és eltérő felhasználói környezetben. Nekik szól ez a fejezet, amely olyan számítógépes alapismeretekre is épít, amelyeket e könyv korábbi fejezeteiben nem tárgyaltunk.

Az AIRCOMP csak akkor játékszer, ha annak használjuk. A gépünk ennél jóval többet tud. Kiválóan alkalmas arra, hogy segítségével alapszoftver vagy felhasználói szoftver programokat tervezzünk, fejlesszünk, ilyeneket, vagy ilyeneknek az algoritmusát a gépen kipróbáljuk, teszteljük, azok helyességét bizonyítsuk. Röviden az AIRCOMP kiváló fejlesztőgép.

A programfejlesztés különösen nagy gépeken és kötegelt feldolgozás mellett igen gazdaságtalan, mert egyrészt drága gépidőt fogyaszt, másrészt időigényes a hosszú átfordulási idő miatt. Emellett Magyarországon ritka, hogy a programozók külön fejlesztőgépet kapjanak, így az üzemi gépen kell fejlesszteniök, amivel üzemeltetésre vagy eladásra fordítható gépidőt emésztenek fel. A számítóközpontok általában igyekeznek is korlátozni a programozók tesztelésre fordított gépidejét.

Személyi számítógépünk azonban mindig a kezünk ügyében van, a nap 24 órájában a rendelkezésünkre áll, a szokásos egy-két nap helyett néhány perces átfordulási idővel számolhatunk, és a felhasznált gépidő gyakorlatilag ingyen van. Mindezen előnyök mellett olyan tulajdonságokkal rendelkezik, amelyek lehetővé teszik, hogy nemcsak az AIRCOMP-hoz hasonló elvű mikrogépekre, hanem akár az egészen nagy gépekre is programokat fejleszthessünk rajta.

Természetesen arról külön gondoskodnunk kell, hogy az idegen gép és környezet olyan funkcióit, amelyek az AIRCOMP-on nem léteznek, megfelelően szimuláljuk.

A szimulálás lényege az, hogy az AIRCOMP-on nem az idegen gépre írt programot, hanem egy azzal ekvivalens programot futtatunk. Fontos alapelv azonban, hogy két program akkor ekvivalens, ha funkcióik és a szerkezetük teljesen megegyezik, és csak a funkciók megvalósítási módjában, a kódolásban különböznek.

Például vegyünk egy jól definiált egyszerű funkciót: adjuk meg egy I változóban egy adott tízelemű S karakterlánc azon elemének a pozícióját, amely

balról jobbra haladva első a karakterlánc olyan elemei között, amelyeknek tartalma az "A" betű.

A feladat megvalósítása BASIC-ben:

```
45010  FOR I=1 TO 1ϕ
45020  IF MID$(S$,I,1)="A" THEN POP: GOTO 45050
45030  NEXT I
45040  I=ϕ
45050  RETURN
```

És ugyanez például COBOL-ban:

```
32010  77  S PICTURE X(1ϕ).
32020  77  I PICTURE 9(ϕ2).

45010  P-KEZD.
45020      EXAMINE S
45030          TALLYING UNTIL FIRST "A".
45040      MOVE TALLY TO I.
45050  P-VEGE.
45060      EXIT.
```

Látható, hogy az utasítások még csak nem is hasonlítanak egymásra. A funkciók viszont teljesen azonosak. (A szerkezetek azonossága nem merülhet fel, mivel kiragadott funkcióról van szó).

Ha két program a mi értelmezésünk szerint ekvivalens, akkor, ha az AIRCOMP-on futó változat jól működött, a vele ekvivalens változatban sem lehet nagy hiba. Mert mit ronthatunk el benne? Legfeljebb utasításokat. Ezeknek hibája pedig a tesztelés során nagyon hamar, többnyire már a fordításkor kiderül. Így soha nem kell a jól működő AIRCOMP programunkkal ekvivalens programban nagy szerkezeti egységeket kicserélni, újraírni, módosítani, esetleg az egész szerkezetet áttervezni, legfeljebb néhány utasítást kell csak kijavítani.

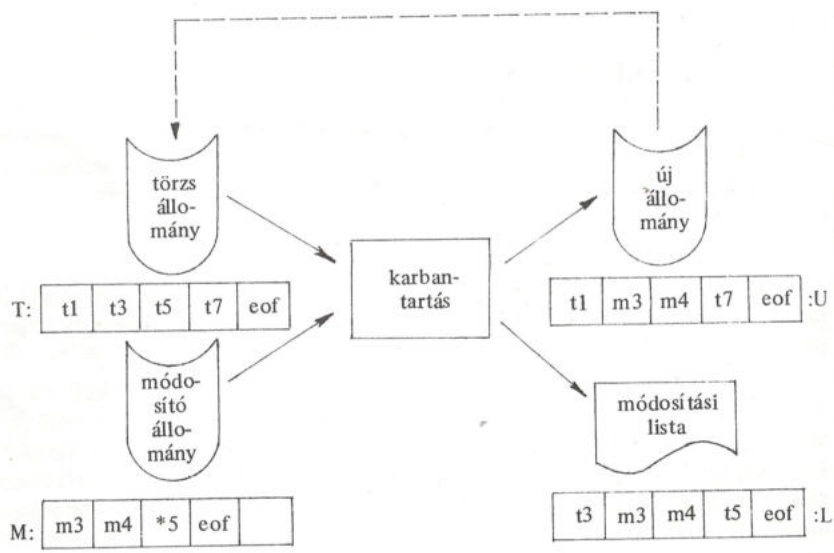
Tapasztalataink szerint az AIRCOMP-on letesztelt átlagos méretű adatfeldolgozó programjaink a tárgygépre telepítve az adatrögzítésből származó szintaktikus hibák kiküszöbölése után elsőre futottak, és azóta is működnek.

A szimulálásra vonatkozó jótanácsok tételes felsorolása helyett a szimulálás mikéntjét inkább egy konkrét példán keresztül mutatjuk be.

Ehhez egy gyakori és jellegzetes adatfeldolgozási feladatot választottunk ki: a rendezett soros állományok karbantartását, éppen gyakorisága, bonyolultsága, és tipikus volta miatt.

A soros állományok karbantartása (update vagy updating) olyan adatfeldolgozó tevékenység, melynek során adott egy úgynevezett törzsállomány, valamint egy ezt módosító állomány, mindketten sorosak, és ugyanazon kulcs szerint azonos módon rendezettek. Ezekből kell létrehozni egy új állományt, amely szerkezetében formailag azonos az előbbi állományokkal, de tartalmazza változatlan formában mindazokat a törzsrekordokat, amelyekhez nem volt módosítórekord, beszúrva mindazokat a módosítórekordokat, amelyekhez nem volt törzsrekord, és a megfelelő módosítórekorddal módosítva mindazokat a törzsrekordokat, amelyekhez volt módosítórekord.

A KARBANTARTÁS ALAPELVE



A programnak tehát egyidejűleg négy állományt kell kezelnie, esetünkben ezeknek egyike sem olyan, amely az AIRCOMP-on használható lenne.

A karbantartást a párosítás alapelvének felhasználásával hajtjuk végre, amit csakis az állományok rendezett, sőt azonos rendezettségű volta tesz lehetővé.

A párosítás lényege, hogy a törzs és a módosító állomány egy-egy rekordjából egy párt képezünk, és megnézzük, hogy összetartoznak-e. Ha nem, meg-

keressük, hogy kettőjük közül melyiknek nincs párja. Ily módon összesen három lehetséges eset van: a törzs és a módosító rekord összetartozik, a törzsrekordnak nincs párja, a módosítórekordnak nincs párja.

A PÁROSÍTÁS MENETE

1. párosítás:

T:	t1	t3	t5	t7	eof
M:	m3	m4	*5	eof	

t1 < m3, tehát átvitel

U: | t1 | | | | |

2. párosítás:

T:	t3	t5	t7	eof	
M:	m3	m4	*5	eof	

t3 = m3, tehát módosítás

U: | t1 | m3 | | | |

3. párosítás:

T:	t5	t7	eof	
M:	m4	*5	eof	

t5 > m4, tehát beszúrás

U: | t1 | m3 | m4 | | |

4. párosítás:

stb.

A három esetnek megfelelően három feltétel, és három, ezektől függő tevékenység származik a párosításból.

F1 feltétel: Ha ti<mk, akkor

T1 tevékenység: a ti törzsrekordra módosítás nem érkezett, ezt a rekordot változtatás nélkül kell az új állományba átvinnünk.

F2 feltétel: Ha ti=mk, akkor

T2 tevékenység: a ti törzsrekord az mk szerint módosítandó. Az mk-tól függően vagy törölnünk kell a ti rekordot, ekkor nem visszük fel az új állományba; vagy (a legegyszerűbb esetben) ki kell cserélnünk a ti rekordot mk-ra, ekkor az új állományba az mk rekordot kell felvinnünk.

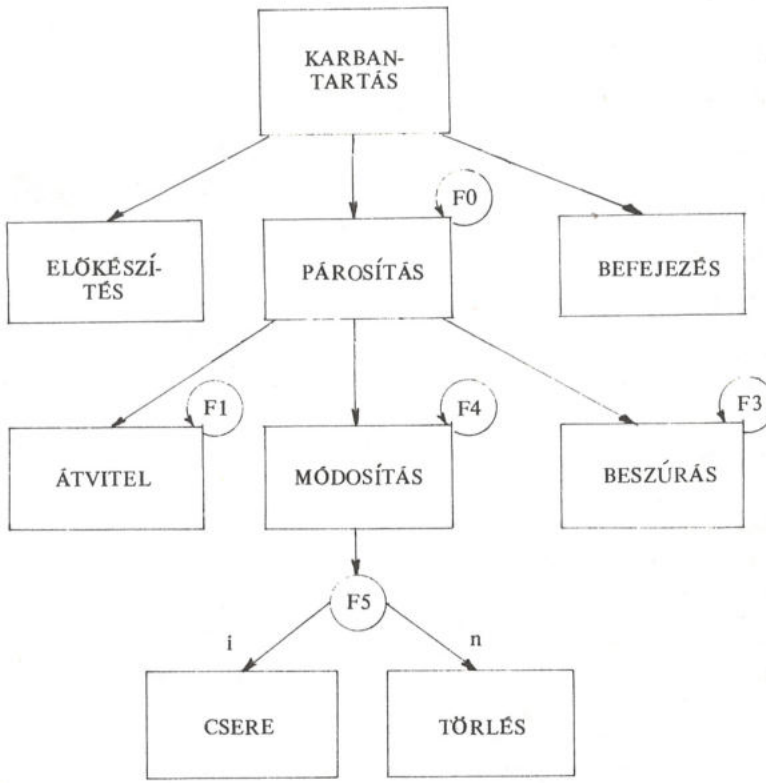
F3 feltétel: Ha ti>mk, akkor

T3 tevékenység: az mk rekordnak nincs párja a törzsben, és mint ilyet, beszúrandónak tekintve fel kell vinnünk az új állományba.

Ezeket a tevékenységeket ismételten mindaddig végre kell hajtanunk, amíg a rájuk vonatkozó feltételek fennállnak, vagy amíg el nem érjük mindkét bemenő állomány végét. Ha a pár bármelyik elemét feldolgoztuk, helyére azonnal új rekordot kell behozni.

A párosításon alapuló karbantartásra ezekből a tevékenységekből jellegzetes programszerkezetet építhetünk fel, amely gerincét képezi egy általános karbantartó programnak, általában mindenféle karbantartásnak, és kisebb-nagyobb módosításokkal egészen speciális igényű karbantartásokra is alkalmazható.

A KARBANTARTÁS SZERKEZETE PÁROSÍTÁSSAL



Feltételek:

F0: TI ≠ EOF és MK ≠ EOF
F1: TI < MK
F2: TI = MK
F3: TI > MK
F4: F2 és nem F0
F5: MA ≠ " "

A párosítás eljárásának a kódját is bemutatjuk:

```
----- párosítás -----
2010  IF TI$=EO$ AND MK$=EO$ THEN GOTO 7010
----- átvitel -----
3010  IF TI$>=MK$ THEN GOTO 4010
3020  GOSUB 17010 === törzs felvitele
3030  GOSUB 13010 === törzs olvasása
3040  GOTO 3010
----- módosítás -----
4010  IF TI$<>MK$ OR TI$=EO$ THEN GOTO 5010
4020  IF MA$=" " THEN GOTO 4210
----- csere -----
4110  GOSUB 18010 === törzs kinyomtatása
4120  GOSUB 19010 === módosító felvitele
4130  GOSUB 20010 === módosító kinyomtatása
4140  GOTO 4310
----- törlés -----
4210  GOSUB 18010 === törzs kinyomtatása
4220  GOTO 4310
----- módosítás vége -----
4310  GOSUB 13010 === törzs olvasása
4320  GOSUB 15010 === módosító olvasása
4330  GOTO 4010
----- beszúrás -----
5010  IF TI$<=MK$ THEN GOTO 6010
5020  GOSUB 19010 === módosító felvitele
5030  GOSUB 20010 === módosító kinyomtatása
5040  GOSUB 15010 === módosító olvasása
5050  GOTO 5010
----- párosítás vége -----
6010  GOTO 2010
-----
```

A kódból azonnal felismerhető a szimulálás egyik szabálya: az AIRCOMP-on meg nem valósítható tevékenységek helyére egy GOSUB utasítást kell kódolnunk, és a valódi tevékenység hatását a hívott belső eljárásban kell szimulálnunk.

A továbbiakban nem az eddigiek alapján már könnyen elképzelhető általános karbantartó programot mutatjuk be, hanem egy speciálisat, melynek sajátossága, hogy kisgépen valósult meg, és kihasználva azok egyik legjellegzetesebb tulajdonságát, az interaktivitás lehetőségét, a karbantartást az operátor felügyelete, vezérlése alatt hajtja végre.

A program funkciója az, hogy egy már rögzített törzsállományba átvezesse a szükséges javításokat. Az egyes rekordok sorszámmal vannak azonosítva.

A javítórekordok jelenthetnek beszúrást vagy cserét, de törlést soha. Bármely rekord törölhető, de csak külön operátori parancsra. Ugyanakkor az új állományba is csak az operátor parancsára kerülhetnek föl rekordok, akár változatlanul hagyva, akár módosítva, akár beszúrva.

VG
ER
VM$
VT$

A PROGRAMBAN HASZNÁLT ADATOK LEÍRÁSA

AJ$	:	Az új állomány megnyitott vagy lezárt állapotának jelzője. Tartalma lehet "NYITVA" vagy "ZARVA".
EG	:	Az állományokat tartalmazó lemezegység száma. Tartalma lehet ϕ vagy 1 vagy 2.
EO	:	Állományvége kód. Tartalma ϕ, ha nincs vége, 1 ha vége van.
EO$	:	Az elvileg legnagyobb, de gyakorlatilag soha el nem érhető sorszám. Tartalma "999".
JS$	:	A módosítórekord sorszáma. Tartalma " 1" és "998" között lehet.
LF$	:	A lemezfej azonosítója. Ha a tartalma "NAPI", a lemez jó, egyébként nem jó.
NJ	:	Annak a sornyomtatónak a száma, amelyre a lista elkészítendő. Tartalma lehet ϕ vagy 3 vagy 4.
SS$	:	A törzsrekord sorszáma. Tartalma u.a. mint JS$.
UH$	:	Az új állomány legutolsó lezárása előtt felvitt rekord sorszáma. Tartalma u.a. mint JS$.
USS	:	Az új állományba felvitt utolsó rekord sorszáma. Tartalma u.a. mint JS$.
VA$	:	Az operátor válasza egy kérdésre. Lehet "I" vagy "N".
VB$	:	Az operátor válasza arra, hogy egy beszúrás végrehajtható-e avagy sem. Tartalma lehet "I" vagy "N" betű.
VF$	:	Az operátor válasza arra, hogy egy felvitel végrehajtható-e avagy sem. Tartalma "I" vagy "N" betű, amit vagy nem követ semmi,

vagy tetszőleges számú szóköz, majd egy legfeljebb háromjegyű pozitív egész szám követ.

VG : Egy adatcsoport végének jelzője. Ha a tartalma ϕ, vége az adatcsoportnak, egyébként nem.

ER : Hibakód. Tartalma ϕ, ha nem volt hiba, 1 ha volt.

VM$: Azt a sorszámot tartalmazza, ameddig a sorok felvitele a VT$ változó "I" vagy "N" tartalmától függően végrehajtható vagy nem hajtható végre. Ha ezt az operátor a VF$ válaszban meghatározta, akkor a VM$ nem más, mint a VF$ utolsó három számjegye. Ha nem, akkor VM$=SS$, az aktuális sorszám. Igy " 1"<=SS$<=VM$<="998"

VT$: A VFS válasznak azon része, amely meghatározza, hogy valamely sor felvihető, avagy sem. Vagyis a VT$ nem más, mint a VF$ első karaktere. Így a tartalma lehet "I" vagy "N" betű.

```
    1  PRINT " ** UNICOMP ** === KARBANTARTO === ** VER. 2 **"
    2  PRINT " **      BOTI  **                    ** 83φ413 **"
    3  PRINT " ------------------------------------------------ "
----
---- előkészítés ----
2010  INPUT " TORZSADATOK LEMEZEGYSEGE =0/1/2= ";EG
2020  IF EG<>0 AND EG<>1 AND EG<>2 THEN GOTO 2010
2030  GOSUB 10010 === lemez ellenőrzése
2040  GOSUB 17110 === törzs megnyitása
2050  IF ER THEN GOSUB 18010 :GOTO 6070 === nyitási hiba
2060  GOSUB 14010 === törzs olvasása
2070  IF EO THEN PRINT " NINCSENEK TORZSADATOK!"
2080  GOSUB 17210 === javító megnyitása
2090  IF ER THEN GOSUB 18010 :GOTO 6070 === nyitási hiba
2100  GOSUB 14510 === javító olvasása
2110  IF EO THEN PRINT " NINCSENEK JAVITASOK!"
2120  INPUT " KARBANTARTHATO =I/N= ";VA$
2130  IF VA$="I" THEN GOTO 2150
2140  PRINT " SEBAJ, MAJD LEGKOZELEBB!" :GOTO 6070 === leállás
2150  INPUT " A KARBANTARTAS ELOLROL INDUL =I/N= ";VA$
2160  IF VA$<>"I" AND VA$<>"N" THEN GOTO 2150
2170  IF VA$="N" THEN GOTO 2210
2180  GOSUB 17310 === új állomány kreálása
2190  IF ER THEN GOSUB 18210 :GOTO 6070 === kreálási hiba
2200  GOTO 2320
```

```
2210    INPUT "  MI AZ UTOLSO FELVITT SOR =NNN= ";US$
2220    US$=RGHS("  "+US$,3)
2230    IF US$<"  1" OR US$>"999" THEN GOTO 2210
2240    IF SS$>US$ THEN GOTO 2280
2250    PRINT "  ";SS$;" ATLEPVE."
2260    GOSUB 14010 === törzs olvasása
2270    GOTO 2240
2280    IF JS$>US$ THEN GOTO 2320
2290    PRINT "  ";JS$;" KIPORGETVE."
2300    GOSUB 14150 === javító olvasása
2310    GOTO 2280
2320    INPUT "  MELYIK NYOMTATO SZABAD =0/3/4= ";NJ
2330    IF NJ<>ϕ AND NJ<>3 AND NJ<>4 THEN GOTO 2320
2340    IF NJ=ϕ THEN GOTO 2360
2350    GOSUB 12010 === fejléc nyomtatása
2360    PRINT "  ----------------------------------------"
----    karbantartás ----
3010    IF SS$=EO$ AND JS$=EO$ THEN GOTO 6010
3020    PRINT "  ";SS$;
3030    INPUT "  FELVIHETO";VF$
3040    IF LEN(VF$)>1 THEN GOTO 3080
3050    VT$=VF$
3060    VM$=SS$
3070    GOTO 3120
3080    VTS=LFT$(VFS,1)
3090    VM$=RGH$(VF$,LEN(VF$)-1)
3100    VM$=RGHS("  "+VM$,3)
3110    IF VMS<"  1" OR VM$>"999" THEN GOTO 3010
3120    IF VT$="I" THEN GOTO 4010
3130    IF VT$="N" THEN GOTO 5010
3140    GOTO 3010
----    felvitel ----
4010    IF SS$=EO$ AND JS$=EO$ THEN GOTO 6010
4020    IF SS$<JS$ THEN GOTO 4110
4030    IF SS$=JS$ THEN GOTO 4210
4040    IF SS$>JS$ THEN GOTO 4310
4050    GOTO 4410
----    felvisz ----
4110    GOSUB 15010 === törzs felvitele
4120    PRINT "  ";SS$;" FELVIVE." :US$=SS$
4130    GOSUB 14010 === törzs olvasása
```

```
4140    GOTO 4410
----    módosít ----
4210    IF SS$=EO$ OR JS$=EOS THEN GOTO 6010
4220    GOSUB 16010 === törzs kinyomtatása
4230    GOSUB 16150 === javító kinyomatása
4240    GOSUB 15510 === javító felvitele
4250    PRINT "  ";JS$;" JAVITVA." :US$=JS$
4260    GOSUB 14010 === törzs olvasása
4270    GOSUB 14510 === javító olvasása
4280    GOTO 4410
----    beszúr ----
4310    GOSUB 13010 === beszúrás
4320    GOTO 4410
----    felvitel vége ----
4410    IF SS$<=VM$ THEN GOTO 4010
4420    GOTO 3010
----    törlés ----
5010    IF SS$=EO$ AND JS$=EO$ THEN GOTO 6010
5020    IF JS$>SS$ THEN GOTO 5110
5030    IF JS$=SS$ THEN GOTO 5210
5040    IF JS$<SS$ THEN GOTO 5310
5050    GOTO 5410
----    töröl ----
5110    GOSUB 16010 === törzs kinyomtatása
5120    GOSUB 17410 === üzenet nyomtatása
5130    PRINT "  ";SS$;" TOROLVE."
5140    GOTO 5410
----    kipörget ----
5210    GOSUB 14510 === javító olvasása
5220    GOTO 5010
----    beszúr ----
5310    GOSUB 13010 === beszúrás
5320    GOTO 5010
----    törlés vége ----
5410    IF SS$=EO$ THEN GOTO 5430
5420    GOSUB 14010 === törzs olvasása
5430    IF SS$<=VM$ THEN GOTO 5010
5440    GOTO 3010
----    befejezés ----
6010    IF NJ=ϕ THEN GOTO 6030
6020    GOSUB 17510 === lábléc nyomtatása
```

```
6030    GOSUB 17610 === az új állomány lezárása
6040    IF ER THEN GOSUB 18150 :GOTO 6070 === zárási hiba
6050    PRINT "  UTOLSO HASZNALHATO SOR = ";US$
6060    PRINT "  VISZONTLATASRA!"
6070    END
----
```

A program érdekessége, hogy a karbantartás mindvégig az operátor vezérlése alatt folyik. Ebből több dolog következik:

– A feldolgozás tetszés szerint újraindítható. Mégpedig az operátor dönthet, hogy elölről kezdi a karbantartást, vagy onnan folytatja, ahol az a program előző futása során megszakadt. Utóbbi esetben csak az utolsó felvitt rekord sorszámát kell megadnia.

– A program minden rekord (akár módosítatlan, akár javított) felvitele előtt felteszi a kérdést, hogy az felvihető-e, illetve a beszúrandó rekordok esetén megtudakolja, hogy azok beszúrhatók-e. Az operátor mindegyik esetben igennel vagy nemmel válaszolhat. Ennek következtében a programnak nem kell törlési funkciókkal foglalkoznia, mivel ezek az operátor kezében vannak.

– Hogy az operátornak ne kelljen minden egyes rekorddal külön-külön foglalkoznia, az "igen" és a "nem" értelmű válaszaihoz intervallumot is megadhat, megjelölve azt a rekordszámot, ameddig válasza – az aktuális sorszámtól kezdődően – érvényes. Például a "123 FELVIHETO" kérdésre "1215" választ adva, a program fel fogja vinni az összes rekordot a 123-astól a 215-ös sorszámig, a határokat is beleértve, felvitel előtt értelemszerűen végrehajtva az összes szükséges javításokat is, ha vannak. A program közben új kérdéseket nem tesz fel, csak a 215. sor felvitele után, kivéve ha közben beszúrások fordulnak elő.

– A beszúrások mindegyikéről külön-külön kell nyilatkoznia az operátornak. Tehát a fenti intervallum feldolgozása közben előforduló beszúrásokra is megkapjuk a "BESZURHATO" kérdést. Az erre adott igenlő vagy tagadó válasz csak az adott beszúrásra vonatkozik, az intervallum többi elemére nem.

```
----  beszúrás ----
13010  PRINT "  ";JS$;
13020  INPUT " BESZURHATO";VB$
13030  IF VB$<>"I" AND VB$<>"N" THEN GOTO 13010
13040  IF VB$="N" THEN GOTO 13210
----
```

```
13110  GOSUB 17710 === üzenet nyomtatása
13120  GOSUB 16510 === javító kinyomtatása
13130  GOSUB 15510 === javító felvitele
13140  PRINT "  ";JS$;" BESZURVA." :US$=JS$
13150  GOTO 13310
---
13210  GOSUB 17410 === üzenet nyomtatása
13220  GOSUB 16510 === javító kinyomtatása
13230  PRINT "  ";JS$;" TOROLVE."
----
13310  GOSUB 14510 === javító olvasása
13320  RETURN
----
```

A program természetesen nem felel az operátor hibás döntéseiért, vagyis az operátort terhelik a következmények, ha igenlő válasz helyett nemlegeset ad vagy fordítva, illetve ha akár újraindításkor, akár intervallum megadásakor rossz sorszámot gépel be.

Mindazonáltal a program nem semmisíti meg a bemenő adatokat, így az operátor hibájából soha nem keletkezik helyrehozhatatlan kár. Bármilyen hiba esetén a programot legrosszabb esetben elölről újraindíthatja és újra elvégezheti – mostmár helyesen – a teljes karbantartást.

Tapasztalataink szerint a program nagyon jól bevált olyan környezetben, ahol a karbantartás során a beszúrások és törlések száma elenyésző, ugyanakkor a karbantartásra gyakran (naponta többször) van szükség, és az technikai okokból ritkán hajtható végre egyetlen menetben.

De szakadjunk el most a program diszkutálásától, és térjünk rá a szimulálási lehetőségek tanulmányozására. Ennek érdekében bemutatunk néhányat a program által hívott belső eljárásokból. Azért nem mindegyiket, mert felesleges volna a könyvet és az Olvasót terhelni olyan modulok leírásával, melyekben a már látottakon kívül újabb szimulálási lehetőség nem fordul elő.

```
----  törzs felvitele ----
15010  PRINT "  --- TORZS FELVITELE ---"
15020  IF AJ$="NYITVA" THEN GOTO 15060
15030  GOSUB 17410 === az új állomány megnyitása
15040  IF ER THEN GOSUB 18010 :GOTO 6070 === nyitási hiba
15050  AJ$="NYITVA"
---
```

```
15060  INPUT "  ERR=";ER
15070  IF ER THEN GOSUB 18410 :GOTO 6070 === írási hiba
15080  INPUT "  VEG=";VG
15090  IF VG=ϕ THEN GOTO 15120
15100  PRINT "  SS$=";SS$
15110  GOTO 15170
-----
15120  IF AJ$="ZARVA" THEN GOTO 15170
15130  GOSUB 17610 === az új állomány lezárása
15140  IF ER THEN GOSUB 18510 :GOTO 6070 === zárási hiba
15150  AJ$="ZARVA"
15160  UH$=SS$
-----
15170  US$=SS$
15180  RETURN
-----
```

Megjegyezzük, hogy az általunk használt fájlkezelő rendszer alatt az új állománynak a meghatározott sorok utáni lezárására és újra megnyitása azt eredményezi, hogy a lemezen mindig van feldolgozható állapotban lévő állományunk, így a program megszakításakor nem kell a teljes állományt újra feldolgozni, hanem a feldolgozás az utolsó lezárást követő sortól folytatható, a program újraindítása után. Ezeket az újraindítási pontokat adják meg az UH$ és US$ változók. Az AJ$ jelző szerepe csak annyi, hogy az új állományba író két eljárás (a törzs felvivő és a javító felvivő) ezen keresztül informálják egymást, hogy az új állományt milyen állapotban hagyták maguk után.

Egyébként ez a példa azt is illusztrálja, hogy a szimuláláskor milyen különböző gépi sajátosságokat kell figyelembe venni. Ilyen például az, hogy csak lezárt adatállományok dolgozhatók fel. Ez nem minden gépre, illetve fájlkezelő rendszerre jellemző, de arra a gépre igen, amelyen a szimulált programunk eredetije fut. Így a szimulálásban sem hagyhatjuk figyelmen kívül.

```
-----  lemezellenőrzés -----
10010  PRINT "  --- LEMEZELLENORZES ---"
10020  GOSUB 17ϕ10 === lemezfej megnyitása
10030  IF ER THEN GOSUB 18010 :GOTO 6070 === nyitási hiba
10040  INPUT "  ERR=";ER
10050  IF ER THEN GOSUB 18110 :GOTO 6070 === olvasási hiba
10060  INPUT "  EOF=";EO
```

```
10070  IF EO=ϕ THEN GOTO 10090
10080  PRINT "  A LEMEZEN NINCSENEK ADATOK!" :GOTO 6070
10090  INPUT "  LF$=";LF$
10100  IF LF$="NAPI" THEN GOTO 10120
10110  PRINT "  EZ NEM A NAPI LEMEZ!" :GOTO 6070
10120  PRINT "  LEMEZTARTALOM=";LF$
10130  RETURN
-----
----  nyitási hiba ----
18010  PRINT "  *** NYITASI HIBA ***"
18020  PRINT "  HIBAKOD=";ER;"  EGYSEG=";EG
18030  GOTO 18910
----
----
18910  PRINT "  --- KARBANTARTAS MEXAKADT ---"
18920  PRINT "  UTOLSO HASZNALHATO SOR = ";UH$
18930  RETURN
----


----  törzs olvasása ----
14010  PRINT "  --- TORZS OLVASASA ---"
14020  INPUT "  ERR=";ER
14030  IF ER THEN GOSUB 18110 :GOTO 6070 === olvasási hiba
14040  INPUT "  EOF=";EO
14050  IF EO THEN SS$=EO$ :GOTO 14070
14060  INPUT "  SS$=";SS$
14070  RETURN
----


----  javító kinyomtatása ----
16510  PRINT "  --- JAVITO NYOMTATASA ---"
16520  PRINT "  JS$=";JS$
16530  RETURN
----


----  az új állomány lezárása ----
17610  PRINT "  --- UJ ZARASA ---"
17620  INPUT "  ERR=";ER
17630  RETURN
----
```

Megjegyzések a hívott belső eljárásokhoz.

– Látható, hogy az END utasítást a GOTO helyettesíti, ahol a 6070 az END utasítás sorszáma.

– Minden szimulálásra használt eljárásba kódoljunk olyan utasításokat, amelyek lehetővé teszik a nyomkövetést, annak megállapítását, hogy éppen melyik funkció szimulálása folyik. A PRINT különböző fajtái általában megfelelnek erre a célra.

– Vannak tisztán szimulációs eljárások, amelyek kizárólag az idegen programban elő sem forduló tevékenységeket tartalmaznak, sőt általában maga az eljárás nem is szerepel az idegen programban. Egyetlen szerepe egy nálunk nem létező funkció szimulálása. Ilyen például az új állomány lezárása.

– Vannak olyan eljárások, amelyeknek minden tevékenysége az idegen program megfelelő eljárásában is szerepel, vagyis a mi eljárásunkban nincs egyetlen egy szimulációs tevékenység sem, de az eljárás csak eredeti környezetében hajtódhatna végre, ha nem gondoskodnánk valahol máshol olyan adatok szimulálásáról, amelyek a programot erre az ágra vezérlik. Ilyen például a nyitási hiba.

– Vannak olyan eljárások, amelyekben az idegen programhoz tartozó, és a szimulációs tevékenységek vegyesen fordulnak elő. Ilyen például a törzs felvitele, vagy a lemezellenőrzés. Az előbbiben a tisztán szimulált tevékenység jól elkülönül: a 15060 és 15110 sorok között van. A többi már valódi tevékenység. Az utóbbiban ezek a tevékenységek már jobban keveredtek.

– Külön említésre méltók azok az eljárások, amelyek későbbi feldolgozás céljából adatokat állítanak elő, és adnak át a többi eljárásnak. Ilyen például a törzs olvasása. Az idegen programban ez egy teljes rekordot olvas be, akár több tucat adattal és több száz karakter hosszúságban. Ezekből az adatokból azonban nem szükséges valamennyit szimulálni, azaz INPUT utasítással bevenni, hanem csak azokat, amelyeket a program ténylegesen fel is használ, azaz amelyekkel műveleteket végez, vagy amelyek alapján vezérlésátadásokat hajt végre. Általában nem kell a programon változtatás nélkül keresztülfutó adatokat szimulálni.

Összefoglalva a szimulálást, alapszabályként rögzíthetjük, hogy általában nem magukat a funkciókat, és soha nem azoknak a programlépéseit kell szimulálnunk, hanem a funkciók hatását, éspedig a többi eljáráshoz képest relatívan. Vagyis az idegen programban egy eljárás hívása ne ugyanazt, hanem ugyanolyan hatást keltsen az őt hívó eljárásban, mint amilyet a mi programunk megfelelő hívott eljárása kelt a mi programunkban lévő hívó eljárásban. Különös gondot kell fordítani az adatok szimulálására, minthogy az eljárások általában ezeken keresztül kapcsolódnak egymáshoz. Programunknak tükröznie kell az idegen program szerkezetét.

AZ AIRCOMP–16 TÖRTÉNETE

s:
r
s:
u
g

p
a
is
k
a

z
A
C
j
á
a
g

ć
k

e
t
r

S

I
t
r
k
n

KÉT DIÁK AMERIKÁBAN

1976-ban egy amerikai egyetemi városban két diák garázsban épített egy szellemesen kialakított kiskomputert, amely hardver- és szoftvertulajdonságai révén olyan újszerű alkotás volt, hogy egy menedzser érdemesnek találta ezt szériában is gyártani. Ezt a számítógépet – megfelelő formatervezés és reklám után – Cupertino-ban kezdtek gyártani. Ebből lett az APPLE II. amely a maga kategóriájában páratlan sikert aratott az egész világon.

KÉT DIÁK MAGYARORSZÁGON

Ez a történet megismétlődött itthon. Lukacs József és Endre egy zuglói panelház 3x3 méteres szobájában dolgozott egy olyan mikroszámítógépen, amely a hazai alkatrészválaszték figyelembevételével olyan tulajdonságokkal is rendelkezik, amelyet hasonló kategóriájú nyugati kisszámítógépeken hiába keresnénk. József a hardvert és a monitort, Endre pedig a BASIC-interpretert alkotta.

Az első példány éppen akkor készült el, amikor Budapesten a Szakszervezetek székházában rendeztek A SZÁMITÁSTECHNIKAI MINDENKIÉ – A SZÁMITÁSTECHNIKA MINDENKIÉRT című kiállítást. A Házikészítésű Computer-Club standjain először mutatták be a gépüket. A szakemberek teljes meglepetésére a gép nagyfelbontóképességű grafikában háromdimenziós ábrákat rajzolt. Ezt akkor semmilyen magyar kisszámítógép nem tudta, és az egészet ráadaásul olyan áron, amely messzemenően alatta maradt az egyéb gépek árszínvonala alatt.

AZ AIRCOMP–16 MŰSZAKI FELÉPÍTÉSE

Az AIRCOMP–16 szíve a Z80A mikroprocesszor, amely teljes 4 MHz-es órajellel működik. Ez a processzor végzi a képszerkesztést is, azaz gondoskodik arról, hogy a képernyőn megjelenjen a 25 sorban 48 karakter.

Ez egy 1,5 K memóriaigényű monitorral történik. A monitor tartalmaz ezenkívül olyan rutinokat, amelyek lehetővé teszik az alapkommunikációt a billentyűzettel, a képernyővel, a magnóval, vmt. a 6,5 K-s BASIC-interpreterrel.

A szabad memóriaterület 16115 bytes. Tehát majdnem az egész RAM BASIC-terület lehet.

De van olyan lehetőség, hogy a képernyőn maximálisan 320x200 raszterpontos grafikát lehessen megjeleníteni. A grafika képet természetesen a RAM-ban kell összeállítani. Ez azt jelenti, hogy ilyenkor nem illik a grafikai RAM-területet programterületként használni; hogy ez ne történjen meg, erről gondoskodik a speciális HM-változó, amely beállítja a BASIC-program számára a legmagasabb használható memóriacímet.

A GL speciális változó beállítja a grafikai terület nagyságát. GL a maximá-

lis grafikai sorok számának megfelelően legfeljebb 200 lehet, de lehet ennél kisebb értéket is megadni. Ilyenkor a grafikusan használható képernyő arányosan csökken. Pl. GL=188 beállításával csak a képernyő alsó fele használható grafikusan, a felső fele viszont megmarad szöveges résznek. Igy tetszőleges arányban lehet a képernyőt grafikus és szöveges részre osztani, ami azt jelenti, hogy az alkotott ábrák fölött magyarázó szöveget is lehet elhelyezni. Ilyenkor célszerű a DL speciális változót is beállítani, mert egyébként a szöveges képernyőn a grafikus rész alatt nem lenne látható az oda írt szöveg. A DL értékének csökkentésével a szöveges képernyő is csökken annyira, hogy a mikroprocesszor csak a látható sorokat kezeli.

A BASIC-interpreter néhány olyan lehetőséggel rendelkezik, amely ebben a gépkategóriában egyáltalán nem szokványos.

Igy pl. megengedi a kiszámított ugrások végrehajtását, mert a GOTO utasítás után elfogad egy változónevet, amelynek értékét előzőleg ki kell számolni.

Az ON X utasítás után következő szokványos sorszámlistának a tartalma úgy lett megváltoztatva, hogy sorszámok helyett aposztrofok között elhelyezett egész programrészeket lehet végrehajtani az x érték függvényében. Így keletkezhetnek olyan tömör programok, amelyek lényegesen elősegítik a memóriagazdálkodást.

További újdonság az, hogy külső periféria által generált interrupt hatására a BASIC-interpreter megszakítja működését és elugrik egy előre meghatározott sorra, ahol a megfelelő – BASIC-ben írt – interrupt-alprogramot dolgozhatja fel. Ezek után a gép folytathatja az előzőleg megszakított BASIC-programot.

A gép – az olcsó hardver érdekében – csak 32K-ig kibővíthető. De ha figyelembe vesszük, hogy a legtöbb BASIC-program bőven elfér ebben, nem látszik szükségesnek a további bővítés, hisz a tömör BASIC-program miatt olyan programok is elférnek az AIRCOMP-ban, amelyek pl. egy VT28-ba is csak cipőkanállal férnének bele.

A GYÁRTÓ

A kiállításon részt vett a BOSCOOP Agráripari Közös Vállalat Számítástechnikai és Információs Főosztálya. A főosztály vezetőjének – Cseres Pálnak – is tetszett a kis gép. Felkereste a Lukács testvéreket, hogy nem látnának-e lehetőséget az együttműködésben, hogy a BOSCOOP gyártana ezt a kisgépet.

A tárgyalások végül sikeresen zárultak, és még 1982-ben kikerültek a nullszéria példányai.

Ezt követte az első kisszéria, amelyet a BOSCOOP már saját üzemeiben és néhány oktatási intézményben értékesített.

A szériagyártás a barcsi UNITECH szövetkezettel kooperálva indult meg. De elég hamar lehetett látni, hogy a laza kooperáció nem elegendő a termelés biztonságos elindítására. Igy 1983. május 1-én alakítottak a PERSONAL Agroelektronikai Gazdasági Társaságot, amelyben az eddigi kooperációs partnerek egyesültek.

AZ AIRCOMP–16 MA

A géppel szerzett tapasztalatok egyértelműen mutatták, hogy néhány műszaki változást kell végrehajtani. A legfőbb gond volt a gép belsejébe bezárt tápegység, amely olyan hőmennyiséget adott le, hogy a felforrósodott alkatrészek már nem működtek rendesen. Igy szét kellett választani a gép elektronikáját a tápegységtől. Ezenkívül szükségesnek látta a BOSCOOP a gépet formatervezővel is átterveztetni.

Az új köntösben kihozott gép első szériája az idei őszi BNV-n jelent meg először a nyilvánosság előtt és általános tetszést aratott. A külön tápegység nagyobb költséget okozott ugyan, de mégis megmaradt az a célkitűzés, hogy a gépet tartós fogyóeszközként is be lehessen szerezni. Ez úgy sikerült, hogy a számlázásnál az alapgép ára 19988,– Ft, a tápegységé pedig 7188,– Ft. Ezzel az AIRCOMP–16 továbbra is a legolcsóbb hazai számítógép, sőt ha összevetjük a gép árát az egyéb itthon kapható – nyugatról behozott – számítógépek áraival, akkor így is a legolcsóbb személyi számítógép.

VÁRHATÓ FEJLESZTÉSEK

Elsősorban a gépet olyan felhasználói szoftver-rel kell ellátni, amely a gép szélesebb körű alkalmazását teszi lehetővé.

Ezenkívül lesznek olyan alapszoftverbővítések is, amelyek elsősorban a grafika használatát könnyítik.

A cél olyan általánosan használható számítógép alkotása, amely a felhasználók igényeinek megfelelően lehet bővíteni. Emiatt tervez a PERSONAL GT bővítőkártyát, amely kivezetett busz sokszorosítását teszi lehetővé. Ezen keresztül lehet majd többféle perifériát csatlakoztatni, sőt a felhasználó által fejlesztett, ill. használt készülékeket is.

További terv olyan játékvezérlő doboz, amellyel főleg az ifjúság számára lehet élvezetesebbé tenni a számítógéppel való foglalkozást.

Elkészült a párhuzamos interfész, amely nyomtató csatlakoztatását teszi lehetővé. Nyomtatót viszont a PERSONAL GT sem tud szállítani, hisz a kisszámítógép gyártása már súrolja a gazdaságosság határát. A nyomtató gyártása viszont itthon teljesen kilátástalan dolognak tűnik, mert a progálkozások a magas finommechanikai munkaarány miatt eddig nem vezettek eredményre. Itt is majd csak olyan megoldás segít, mint maga az AIRCOMP–16. Reméljük, hogy nem sokára ez a probléma is megoldódik, talán úgy, hogy lesz két diák. . .

AAAABCCCCCCCEEEEEEEEFFC

AZ AIRCOMP-16 BASIC-NYELVÉNEK FOGLALT SZAVAI

Ezek a szavak változónévként, változónév részeként nem használhatók!

ABS	(86)	GUSUB	(45)	POP	(48, 74)
AND	(52)	GOTO	(39)	PRINT	(27, 110)
ASC	(87)	HM	(95)	READ	(89)
ATN	(88)	IF	(41)	RESTORE	(89)
BRK	(117)	INPUT	(34, 107)	RETURN	(45)
CALL	(93)	INS	(33)	RGH$	(86)
CHR$	(87)	INT	(86)	RND	(86)
CLR	(117)	LEN	(87)	RUN	(31)
CONT	(117)	LFT$	(87)	SAVE	(101)
COS	(86)	LIST	(32)	SGN	(87)
CR	(95)	LOAD	(104)	SIN	(87)
DATA	(89)	LOG	(86)	SQR	(87)
DEFFN	(88)	MID$	(87)	STEP	(72)
DEL	(33)	NEXT	(73)	STR$	(87)
DIM	(76)	NEW	(33)	TAN	(87)
DL	(94)	NOT	(52)	THEN	(41)
END	(45)	ON	(63)	TO	(72)
EXP	(86)	OR	(52)	TRC	(120)
FOR	(72)	PEEK	(86)	USR	(87)
FN	(88)	PLOT	(96)	VAL	(87)
FRE	(86)	POINT	(87)		
GL	(95)	POKE	(93)		

TARTALOM

Közös képviselő:

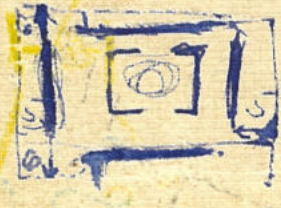
BOSCOOP

AGRÁRIPARI KÖZÖS
VÁLLALAT
2040 Budaörs
Nefelejcs u. 2.

Felelős kiadó: Cseres Pál igazgató
Felelős szerkesztő: Molnár Gáborné
Készült: Ipari Informatikai Központ nyomdájában
Példányszám: 1000
Terjedelem: 8,5/A5 ív
Engedélyszám: 45 962
ISBN 963 01 5547 8

Terjedelem: 8,5/aA5 ív

INFORMATIK 83.273



PERSONAL

PERSONAL AGROELEKTRONIKAI
GAZDASÁGI TÁRSASÁG
2040. BUDAÖRS, MOLNÁR P. U. 1.
TELEFON: 260–612
TELEX: 22–5962