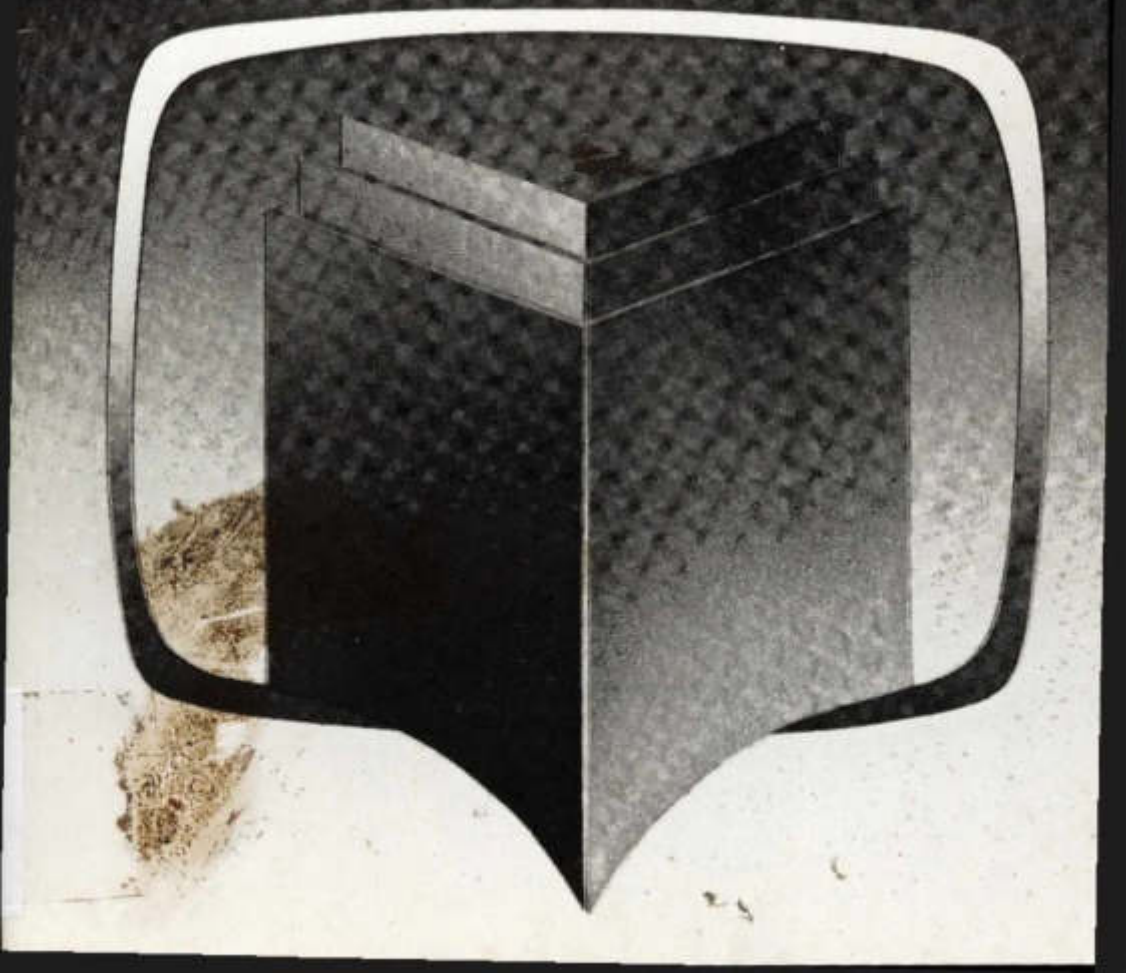
DR. KOCSIS ANDRÁS

TV-BASIC

SZÁMALK



ITA/432

TV-BASIC

Dr. András Kocsis

TV–BASIC

Computing Applications and Service Company
Budapest, 1984

Dr. Kocsis András

TV–BASIC

Számítástechnika-alkalmazási Vállalat
Budapest, 1984

Lektorálta: Danis Miklós

Supervised by Miklós Danis

© *Dr. Kocsis András, 1984*

ISBN 963 553 102 8

TARTALOM

Köszöntöm

a tv-nézőket és a könyv olvasóit! Szokatlan vállalkozásba kezdett a Neumann János Számítógéptudományi Társaság, a SZÁMALK és a Magyar Televízió, tanfolyamsorozatot indít a számítástechnika népszerűsítésére és főleg tanítására. A szó nemes értelmében **népoktatásra** *vállalkozunk, arra, hogy mindazok, akik eddig nem kerültek vagy nem kerülhettek közelebbi kapcsolatba a számítástechnikával, a televízió segítségével szerezhessék meg a legfontosabb ismereteket. Indul a* **Tv-BASIC**, *s a tanfolyam végén — először a Magyar Televízió történetében — a szorgalmas néző vizsgát tehet, és sikeres vizsga esetén bizonyítványt kap.*

A tanfolyam anyagát és a könyvet Kocsis András, a SZÁMALK munkatársa állította össze, a kéziratot, majd az ebből készült forgatókönyvet az NJSZT tagjaiból, a SZÁMALK és az MTV munkatársaiból álló munkabizottság vitatta meg és fogadta el.

Nyilvánvalóan tökéleteset szerettünk volna alkotni, ám nagyon optimistának kellene lennünk, hogy hihessünk, ez sikerült is. Reméljük, hogy a tv-sorozatot és a könyvet is a néző—olvasó könnyen megérti. A feltétlenül szükséges BASIC ismeretek mellett olyasmit is bemutatunk, ami nemcsak a kezdők, de a már bizonyos ismeretekkel rendelkező szakemberek érdeklődését is felkelti.

Amit elmondunk, ahhoz feltétlenül elegendő, hogy bárki több-kevesebb sikerrel programozni tudja a munkahelyi vagy az otthoni számítógépet.

A tanfolyam elsősorban azoknak a felnőtt — ha szabad azt mondani, hogy talán nem is túl fiatal — szakembereknek szól, akiket a személyi számítógépek programozása érdekel, de eddig nem volt alkalmuk ezt a „tudományt" elsajátítani. Mindannyian tudjuk, hogy a mai egyetemisták, középiskolások, de talán az általános iskolai tanulók nagy része is „tudósa" a számítástechnikának, reméljük, azért nekik is tudunk újat mondani.

A BASIC-nek — bármennyire is szabványosított programozási nyelv — különböző típusú gépeken különböző változatai vannak forgalomban. Mi úgy gondoltuk, hogy a mikroszámítógépes klubokban,

iskolákban, művelődési és ifjúsági házakban, általában mindazokon a helyeken, ahol szabadidőben számítógéppel lehet dolgozni vagy szórakozni, leginkább Commodore–64, HT–1080Z, PRIMO vagy Sinclair-gépek találhatók. Ezért e könyvben az ezeken a gépeken használt BASIC utasításokat mutatjuk be. Minden új utasítást példákkal illusztrálunk. A feladatokat igyekeztünk változatosan összeállítani. Nem is egyszerű elmemunka lesz például a Commodore-mintaprogramot átírni Sinclair-gépre. Arra számítunk, hogy a tanfolyam nézői erre a „kalandra" is vállalkoznak. Itt hívjuk fel figyelmüket a Mikroszámítógép Magazin 1985. évi számaira, amelyekben bőségesen szeretnénk további példákat bemutatni a tanfolyam nézőinek.

A könyvet, az adást kísérletnek szánjuk, az eredményt a vizsgák mutatják meg, illetve a nézők véleményéből szeretnénk kiszűrni. Ezért nemcsak várjuk a véleményeket, de kérjük is, hogy a tanfolyam ismétlésekor, illetve a következő tanfolyam (Számítástechnikai alapismeretek) összeállítása során a javaslatokat már figyelembe tudjuk venni.

Szeretnénk a tanfolyam anyagát később is hasznosítani, ezért azt videokazettán a tankönyvvel együtt árusítani fogjuk. Úgy véljük, hogy hasznos segédeszköze lesz a pedagógusoknak, de a vállalatok, intézmények belső tanfolyamaikhoz is jól tudják használni.

E nagy vállakozásért köszönetet mondok az MTV Ifjúsági és Oktatási Főosztály munkatársainak: Kovács Béla, Kelemen Endre, Gál Mihály, Albert József, Hegyi István elvtársaknak, az anyag kidolgozását támogató, lektorálását és megvitatását, valamint a tv-sorozat bírálatát vállaló bizottság tagjainak: Ada-Winter Péternek, Faragó Sándornak, Meskó Andornak, Pál Lászlónak, Páris Györgynek, a könyv gondozásáért felelős Pálfy Adorjánnak, a könyv gyors elkészítéséért a Dabasi Nyomda dolgozóinak és nem utolsósorban a szerzőnek, Kocsis Andrásnak.

Végül előre megköszönöm a néző–olvasó szíves közreműködését, véleményét és javaslatait, amelyeket – mint említettem – megpróbálunk a további adásokban és könyvekben minél hamarabb hasznosítani.

Kovács Győző
az NSZJT főtitkára

ELŐSZÓ

Manapság, amikor a személyi számítógépek robbanásszerűen terjednek hazánkban, érthető módon egyre többen szeretnék megtanulni a módját, hogyan lehet ezeket a gépeket minél jobban hasznosítani akár a gazdasági munkában, akár a magánélet különböző területein. A számítástechnika fejlődése úgy hozta, hogy valamennyi mikro- és személyi számítógép programozható BASIC nyelven. Ez természetesen nem véletlen. A BASIC nyelv a magyar származású John G. Kemeny munkájának eredményeként húsz éve indult el hódító útjára. Ez az egyszerű, könnyen megtanulható és széles körben alkalmazható nyelv rendkívül népszerű lett. Éppen ezért vált a személyi számítógépek alapnyelvévé. Ennek köszönhetően a BASIC nyelv egy általános eszköz, amellyel bármely személyi számítógépet lehet használni. Megjegyezzük, hogy az egyes géptípusok BASIC-jei között kisebb-nagyobb eltérések, „tájszólások" vannak. Sorozatunkban az ún. „közös rész"-t igyekeztünk tárgyalni, amely minden gépnél azonos. Emellett elkerülhetetlen, hogy a nyelv eltérő sajátosságait is feltárjuk az egyes megoldásoknál.

A BASIC nyelv alkalmazásához a számítógép használatát is meg kell ismerni, ez viszont már kevésbé általánosítható. Ez is szükségessé teszi, hogy konkrét géptípusokon mutassuk be a nyelvet. Úgy gondoljuk, hogy a ma Magyarországon legelterjedtebb négy géptípus, a Commodore–64, a HT–1080Z, a PRIMO és a Sinclair-gépek felelnek meg a legjobban ennek a célnak. Mivel a Commodore–64 gépből van a legtöbb, könyvünkben ez az alapgép, a többi gépre vonatkozó megállapításokat megkülönböztető jelzéssel látjuk el. A HT gépe jele: ■, a PRIMO-é ▲, a Sinclair-gépeké pedig ● jel, amely mindig a megfelelő szöveg kezdetén jelenik meg.

A tanuláshoz segédanyagunkat szánjuk segítségül. A fejezetek a sorozat tagolódását követik: tartalmazzák az egyes részekben tárgyalni kívánt elméleti anyagot és a feladatok megoldási lépéseit. A tv-ben látható műsor csak akkor segíti a BASIC nyelv gyors megtanulását, ha a kedves Néző a műsor előtt már elolvassa az elméleti részt.

A fontosabb fogalmak magyarázatát az 1. függelék tartalmazza ábécérendben, ezenkívül a tárgymutató segíti a fogalmak keresését a könyvben. A 2. függelékben találhatók a számítógép és a nyelv használatához szükséges kulcsszavak (parancsok, függvények és utasítások,

valamint rövid, tájékoztató jellegű leírásuk). Itt utalunk arra is, hogy az illető utasítást melyik oldalon tárgyaljuk. Természetesen könyvünk nem tartalmazza az összes parancs, függvény és utasítás leírását, mert erre itt nincs lehetőségünk. Ha olyan utasítást vagy függvényt kíván alkalmazni, amelyről itt nincs szó, javasoljuk, hogy a gép kézikönyvét is lapozza fel.

A 3. függelékben a gyakrabban előforduló hibákat és javításukat, valamint a négy gépre vonatkozó programjavítási lehetőségeket részletesen ismertetjük.

A Commodore-gép állománykezelési műveletei a 4. függelékben találhatók.

Az 5. függelékben példaként a Commodore-gép kódtáblázatát adjuk közre, ez a „profi" programozáshoz jól alkalmazható. Az egyes gépek programozói kézikönyvében a megfelelő táblázat megtalálható.

Ugyancsak a függelékben kaptak helyet az egyes feladatokat megoldó programok listái. Ezeket csak végső esetben érdemes használni, inkább saját megoldásokra törekedjünk.

1. BEVEZETÉS

Ismerkedés a számítógéppel. Az első programok elkészítése

Ez az első alkalom, hogy számítógépet használunk. Van aki tartózkodóan, van aki kíváncsian közelít a géphez. Szeretnénk megértetni, hogy nem kell előismeret vagy különleges képesség ahhoz, hogy valaki értelmesen tudja használni a számítógépet. Már az első műveletekkel megpróbáljuk ezt az állítást bizonyítani. Remélhetőleg mindenki hamarosan érezni fogja, hogy ha nem is magas fokon, de uralja a gépet. Ne keseredjen el senki, ha valamilyen fogalmat nem ért meg első hallásra. Inkább a műveletekre figyeljen, és abból próbálja megmagyarázni, hogy mi is történik. Egyébként megjegyezzük, hogy a továbbiakban minden itt felmerülő fogalmat részletesen és teljeskörűen megmagyarázunk.

Először is nézzük meg, hogy milyen eszközök szükségesek a programozáshoz! Kell egy számítógép, amely a műveleteket végzi el, továbbá egy olyan eszköz, amelynek segítségével meg tudjuk mondani a számítógépnek, hogy mit csináljon, s végül egy kijelző eszközre is szükségünk van, amelyen az eredmények megjelennek. Egy írógép-billentyűzethez hasonló berendezésen lehet jeleket beírni a számítógépbe, a kijelző eszköz pedig egy tv-készülék (1. ábra). Ez a programozáshoz nélkülözhetetlen **minimális** eszközkészlet.

Kapcsoljuk most be az eszközeinket; először a tv-t, majd a gépet. Kisvártatva megjelenik egy üzenet a tv-n, amely mutatja, hogy dolgozhatunk. Az üzenet utolsó szava a

READY

- A HT gépen

READY?
>

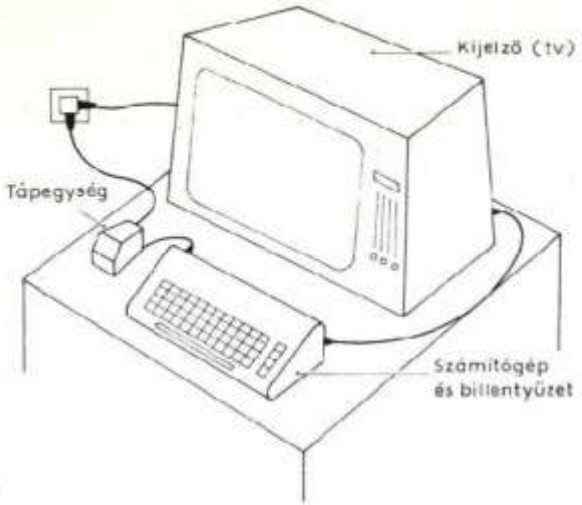
(nyomjuk le a NEW LINE gombot és akkor a képernyő alján jelenik meg a READY üzenet)

- a PRIMO-nál pedig

Ok

jelenik meg.

A kiírás alatti sor elején az ún. **helyőr** (pozíciómutató, kurzor; angolul cursor) villog.



1. ábra. A programozáshoz szükséges eszközök

- A Sinclair-gépeknél a képernyőn a

  © 1982 Sinclair Research Ltd.

  felirat olvasható, majd az ENTER gomb lenyomása után a felirat eltűnik és helyette a képernyő alján bal oldalt egy K betű (K helyőr) jelenik meg.

Ha ezektől eltérő kiírás olvasható, akkor hiba van.

Az üzenet és a villogó helyőr azt jelenti, hogy a gép készen áll BASIC programok befogadására, végrehajtására stb. Írjuk most be az alábbi szöveget:

PRINT "ITT VAGYOK!"

és nyomjuk le a RETURN gombot.

- ■ A HT gépen a beírás végén a NEW LINE gombot kell lenyomni,
- ▲ a PRIMO-nál a beírás kezdete előtt a nagybetű írásához szükséges UPPER gombot nyomjuk le, s a sort itt is a RETURN-nel zárjuk le.

Felhívjuk a figyelmet, hogy az idézőjelek és a felkiáltójel az 1 és 2 feliratú gomb felső részén találhatók, ezért csak akkor jelennek meg a képernyőn, ha a SHIFT billentyűt előzetesen lenyomjuk, és utána ütjük le a kívánt jel gombját.

- A Sinclair-gépeknél a PRINT szót nem betűnként kell begépelni, hanem alulról a harmadik sor jobb szélső billentyűjét kell lenyomni, s a gomb feliratának megfelelően megjelenik a PRINT

szó. A PRINT után újra villog a helyőr, de most már nem K, hanem L helyőr. Ez azt jelenti, hogy most betűket, számokat és egyéb írásjeleket lehet beírni, vagyis a billentyűkön lévő betűk „élnek". Alaphelyzetben a kisbetűk, de ha a CAPS SHIFT és CAPS LOCK gombot együtt lenyomjuk, ettől kezdve nagybetűket írhatunk. Az idézőjelet úgy lehet kiírni, hogy lenyomjuk a SYMBOL SHIFT billentyűt és a P gombot. A ! jelet a SYMBOL SHIFT és az 1 gomb (piros jel!) lenyomásával írhatjuk ki. A beírás végén az ENTER gombot kell lenyomni. Hatására az alsó sorból eltűnik a szöveg.

Ha eltévesztjük a gépelést, ne keseredjünk el, nyomjuk le a

RETURN (▲PRIMO-nál is!)
- NEW LINE
- ENTER

gombot és kezdjük elölről. Hiba esetén mindig megjelenik egy hibaüzenet, ezzel azonban most még ne törődjünk.

Ha hibátlanul beírtuk a fenti szöveget, akkor egy sorral lejjebb megjelenik az

ITT VAGYOK!

szöveg.

Mi ennek a magyarázata?

- Megjegyezzük, hogy a Sinclair-gépeknél a szöveg a képernyő legfelső sorában jelenik meg. A képernyő alján pedig több jel látható, amelyek az elvégzett műveletre vonatkoznak. Ebből számunkra az

OK

a legfontosabb, amely azt jelenti, hogy a művelet „oké". Ezután ismét meg kell nyomni az ENTER gombot, hogy visszatérjen a K helyőr.

A begépelt sorral utasítást adtunk a számítógépnek, hogy írja ki az ITT VAGYOK! szöveget, és a gép ezt végrehajtotta. Az erre vonatkozó utasítást a PRINT szóval közöltük a géppel, a kiírandó szöveget pedig idézőjelbe tettük. Ez jelzi a gépnek, hogy mit kell megjelenítenie. Próbáljuk meg más szövegek kiírását is!

Például

```
PRINT"GABOR"
PRINT"EZ JOL MEGY!"
```

Ne felejtsük el minden sor (vagy utasítássor) begépelése után a

RETURN,
- NEW LINE,
- ENTER

gombot lenyomni! A gép csak akkor hajtja végre az utasítást, ha ezt az ún. **sorzáró** billentyűt lenyomtuk. Erről úgy győződhetünk meg, hogy megnézzük, hol a helyőr. Ha a helyőr az utolsónak begépelt jel után villog, akkor nem nyomtuk le.

Most egy más jellegű utasítássort gépeljünk be:

```
PRINT 6*8
```

és nyomjuk le a sorzáró billentyűt. A 6 és 8 közötti ún. csillagjel a BASIC nyelvben a szorzás jele. A helyes begépelés után a gép a következő sor elejére kiírja azt, hogy

```
48
```

Nyilvánvalóan arra utasítottuk a gépet, hogy a 6*8 szorzat eredményét írja ki.

Figyeljük meg a lényeges különbséget az előző példához viszonyítva: idézőjelet nem írtunk! Ha a szorzatot idézőjelbe tesszük, akkor a kiírás eredménye más lesz.

```
PRINT"6*8"
6*8
```

Ha az idézőjeleket elhagyjuk, a gép először elvégzi a kijelölt számtani műveletet, majd kiírja az eredményt. Próbáljuk meg ezt az utasítást más számokkal is!

A fenti két feladatban a számítógépet utasítottuk valamilyen művelet elvégzésére. Amikor idézőjelek közé írt szöveg kiírására adtunk utasítást, akkor lényegében egy műveletet hajtott végre a gép. A számolási műveleteknél pedig előbb a gépnek ki kellett számítania az eredményt, majd a következő lépésben ezt ki kellett írnia. A lényeg tehát az, hogy mi a gépet utasítottuk valamilyen művelet végrehajtására és az elvégezte.

Látható, hogy ilyen módon a számítógépet úgy lehet használni, mint egy kalkulátort, ezért az ilyen használatot kalkulátormódnak is nevezik. Ez a mód azonban lényegesen több szolgáltatást tud nyújtani a programkészítéshez, mint amit itt bemutattunk, ezért a neve is az általánosabb célt megjelölő **közvetlen mód**. (Erre később még visszatérünk.)

Oldjunk meg egy nagyon egyszerű feladatot!

Írassuk ki a számítógéppel két szám összegét és szorzatát. Legyen a két szám 6 és 8. Jelöljük a számokat A-val és B-vel:

```
A=6
B=8
```

Ebből következik, hogy az összeg kiírásakor az

```
A+B
```

műveletet, a szorzat kiírásakor pedig a

A * B

műveletet kell elvégezni.

A feladat megoldásához négy műveletet kell elvégezni:

1. Az A értéke legyen 6
2. A B értéke legyen 8
3. Számítsa ki az A + B összeget, és az eredményt írja ki
4. Számítsa ki az A * B szorzatot, és az eredményt írja ki

Valójában ez a feladat is megoldható az előzőleg bemutatott módon, de ennek a módszernek korlátai vannak. Ha ez a feladat még nem is lépi át ezeket a korlátokat, előbb-utóbb találunk olyan nagyobb feladatot, amely már biztosan nem oldható meg így. Ezt a másfajta feladatmegoldást azonban könnyebben megérthetjük egy kisebb feladaton.

Mindenekelőtt azt nézzük meg, hogyan érhetjük el, hogy egy A vagy B nevű változó valamilyen értéket kapjon. Erre a

LET

szó való. A szó után meg kell adni a változót és az értéket egyenlőségjellel összekapcsolva:

LET A=6

A gép végrehajtja ezt az utasítást, és eredményeként az A értéke 6 lesz.

- A Sinclair-gépen a LET szót nem betűnként kell begépelni, hanem az L gombot kell lenyomni (a K helyőrnél).

A feladat megoldására tehát új módszert kell keresnünk. Ez a módszer a műveletek **programban** való leírása. Ha nem egy műveletet akarunk a géppel végrehajtatni, akkor az utasításokat a végrehajtás sorrendjében sorszámozva gépeljük be:

```
10 LET A=6
20 LET B=8
30 PRINT A+B
40 PRINT A*B
```

Az utasítások elé írt számok mutatják a végrehajtási sorrendet. Először a legalacsonyabb sorszámú utasítást kell elvégezni (10), majd a következő sorszámút (20) és így tovább.

Ne felejtsük el, hogy minden utasítás után le kell nyomni a sorzáró billentyűt! A következő utasítás csak így kerül a következő sorba a képernyőn.

Már most jegyezzük meg, hogy amikor változóknak értéket adunk, akkor az egyenlőségjel nem a hétköznapi értelemben vett „ténymegállapítás", hanem az egyenlőségjel bal oldalán álló változót egyenlővé teszi a jobb oldalon álló értékkel.

- A Sinclair-gépeknél minden sor begépelése után a beírt sor a képernyő felső részére kerül át a legfelső, még üres sorba. Az utolsónak beírt sorban a szám után > jel látható. Ez csak arra figyelmeztet, hogy ez az utolsónak beírt sor.

Ha begépeltük a fenti négy sort, akkor jól láthatjuk, hogy sem az egyes sorok begépelésének végén

(a RETURN,
- NEW LINE vagy
- ENTER

lenyomása után), sem a teljes feladat begépelése után nem történik semmi. Ez azért van, mert most az utasítássorok elé sorszámot is írtunk. Az ilyen sorokat a gép a begépelésnél nem hajtja végre, hanem csak együttesen az egészet. Külön meg kell viszont „mondani", hogy mikor kell a műveleteket elvégezni. Ehhez be kell gépelni a

RUN

szót

- (a Sinclairnél csak az R betű gombját kell leütni),

le kell nyomni a sorzáró gombot, ezután a számítógép a műveleteket elvégzi. Ha hibátlanul gépeltük be a négy sort, akkor a következő eredmények jelennek meg a képernyőn:

14
48

Ha valamelyik sort tévesen gépeltük be, írjuk be még egyszer. A végrehajtás sorrendje ugyanis független a begépelés sorrendjétől, egyedül a sorszámtól függ.

A RUN hatására a számítógép elvégzi a műveleteket. Ilyenkor nem egyetlen, hanem négy művelet elvégzésére adtunk utasítást.

Gépeljük be ismét a RUN szót, nyomjuk le a sorzáró gombot, és figyeljük, mi történik! Az eredmény ugyanaz, mint az előbb:

14
48

A számítógép megint végrehajtotta a megadott utasításokat anélkül, hogy az utasításokat újra begépeltük volna. Akárhányszor is írjuk be a RUN szót, az eredmény nem változik. Ebből az a következtetés vonható le, hogy a sorszámozott utasítássorokat a számítógép „megjegyzi", és ezek akárhányszor végrehajtathatók. Egy ilyen sorszámozott utasítássorozat egy **program.** A program eddig megismert néhány jellemzője:

– sorszámozott utasításokból áll,

– végrehajtását a RUN szóval lehet kezdeményezni, ahányszor a felhasználó kívánja,
– a számítógép „megjegyzi".

A kiírás formailag nem a legszebb: nem lehet tudni, hogy melyik érték mit jelent. Ha valaki nem sajnálja a fáradságot, akkor értelmesebbé teheti úgy, hogy az érték elé kiíratja az első esetben az

A+B=

a második esetben az

A*B=

magyarázó szöveget. Ezt úgy lehet elérni, hogy a PRINT szó után idézőjelek közé írjuk a fenti két szöveget a megfelelő utasítássorban. Ezután viszont még az értéket is ki kell írni. Ha egy sorba több dolgot akarunk kiíratni, akkor ezeket pontosvesszővel kell elválasztani. Ezek szerint a módosított műveletsorozat (program) a következő lesz:

```
10 LET A=6
20 LET B=8
30 PRINT"A+B=";A+B
40 PRINT"A*B=";A*B
```

▲ A PRIMO gépnél a 30-as és 40-es sorszámú utasítássorban a ; elhagyható.

A módosítást az új sorok begépelésével hajthatjuk végre. Ekkor a korábban beírt azonos sorszámú sor elvész.

Oldjunk meg egy újabb példát! Öt szám átlagát kell kiszámítani és az eredményt kiírni. Legyen az öt szám:

B = 6
C = 8
D = 12
E = 2
F = 5

Mint ismeretes, az átlagot úgy kell kiszámítani, hogy a számokat összeadjuk:

S = B+C+D+E+F = 6+8+12+2+5

és az összeget elosztjuk a tagok számával:

A = S/5

Eddigi ismereteink szintjén a feladat könnyen megfogalmazható számítógép-utasításokkal. Először a változók értékeit kell beállítani, majd

az összeget kell kiszámítani. Vegyük észre, hogy itt nem elég csupán összeadni az öt számot, mert az összeget meg is kell őrizni az átlagszámításhoz. Tehát egy újabb változót kell bevezetnünk, és az összeget értékeként megadnunk. Ez az S változó. Az átlagra is szükségünk van, ezért az S értékét 5-tel elosztjuk (erre szolgál a / jel), és az A változót a hányadossal tesszük egyenlővé. A kiíráshoz pedig az A-t használjuk fel.

Végül is az alábbi műveleteket kell elvégezni:

1. B értéke legyen 6
2. C értéke legyen 8
3. D értéke legyen 12
4. E értéke legyen 2
5. F értéke legyen 5
6. S (összeg) értéke legyen B+C+D+E+F
7. A (átlag) értéke legyen S/5
8. Írja ki A értékét

Joggal merül fel az olvasóban, hogy most miért választjuk szét például az átlag kiszámítását és kiírását. Az előző példákban ezt egyetlen műveletben végeztettük el a géppel. Azért célszerűbb ez a módszer, mert ha például a szórást is ki kellene számolni a feladatban, vagy más számításnál fel kellene használni az átlagot, akkor az itt bemutatott módszer szerint nem kell minden egyes felhasználásnál külön kiszámítani, hanem csak az értékét átvenni. A korábban bemutatott módszer szerint az átlagot minden felhasználásnál ki kellene számítani.

Ezek után hozzá is kezdhetnénk az utasítássorok begépeléséhez, de valamit szem előtt kell tartanunk. Az előző négy műveletsort (program) a számítógép még őrzi. Most viszont egy más feladat műveleteit akarjuk begépelni, az előzőekre már nincs szükségünk. Mit tegyünk? Azt már láttuk, hogy ha egy olyan utasítássort gépelünk be, amelynek a sorszáma megegyezik egy méglévőével, akkor csak az újonnan begépelt utasítássor marad meg. Ha tehát az új feladat utasításorait ugyanazokkal a sorszámokkal gépeljük be, mint az előzőekét, akkor az előző utasítássorokat „töröljük". Ez célravezető megoldás, de sok hibalehetőséget tartalmaz. Mi van akkor, ha a másodiknak beírt program kevesebb utasítássorból áll, mint az előző? Mi van akkor, ha az új programban nincs olyan sorszám, mint ami az előzőben volt (pl. 25)? Ha ugyanis az előző program valamelyik utasítássorát nem töröljük, akkor az megmarad, az új program része lesz, és ezt zavarhatja. Ez a megoldás tehát sok veszélyt rejt magában. Szerencsére van rá lehetőség, hogy elkerüljük.

Ha a

NEW

szót begépeljük,

- a Sinclair-gépeknél az A betű gombját nyomjuk le és a sorzáró billentyűt,

a meglévő program megszűnik, vagyis a NEW hatására a gép törli (elfelejti) a meglévő programot, és ezután lehet kezdeni az új begépelését. Esetünkben ez a következő lesz:

```
10 LET B=6
20 LET C=8
30 LET D=12
40 LET E=2
50 LET F=5
60 LET S=B+C+D+E+F
70 LET A=S/5
80 PRINT A
```

A RUN begépelése után a számítógép végrehajtja a program műveleteit (ha jól gépeltük be őket), és kiírja az átlagot. Ezt a programot is akárhányszor végre lehet hajtatni, az eredmény mindig ugyanaz lesz.

Mi történik viszont akkor, ha azt akarjuk, hogy minden esetben más legyen az az öt szám, aminek az átlagát ki akarjuk számoltatni? Ilyenkor minden végrehajtás előtt meg kell adni azt az öt számot, aminek az átlagát ki kell számolni. Hogy oldható ez meg? Már ismerünk rá egy megoldást. Eszerint az első öt sort át kell írni az új számértékeknek megfelelően. Ha például azt akarjuk, hogy B 115 legyen, akkor az első utasítássort így kell begépelni:

```
10 LET B=115
```

és így tovább. Ez meglehetősen fárasztó, és a gépelés eltévesztése is sok hibát okozhat. Ebben az esetben is van a BASIC-nek olyan lehetősége, amely ennél egyszerűbb megoldást eredményez. Kiadhatunk ugyanis olyan utasítást, hogy a gép a billentyűzeten begépelt értéket adja meg egy változónak.

Hogy ezt jobban megértsük, töröljük ki a meglévő programot a NEW begépelése és a sorzáró billentyű lenyomása segítségével. Írjuk be az alábbi programműveleteket:

```
10 INPUT B
20 PRINT B
```

- A Sinclair-gépeknél az INPUT szót az I betű, a PRINT szót a P betű lenyomásával írhatjuk be.

Írjuk be a RUN szót is a végrehajtáshoz. Azt fogjuk látni, hogy a gép a következő sor elejére kiír egy kérdőjelet:

```
?
```

Ez a 10-es sorszámú utasítássor hatása, és azt jelenti, hogy a gép valamilyen érték begépelésére vár. A program végrehajtása mindaddig fel-

függesztve marad, amíg be nem gépelünk egy értéket. Írjunk be egy számot:

115

és nyomjuk le a sorzáró billentyűt. A 10-es sorszámú utasításban lévő B változó értéke 115 lesz, és a program folytatódik. A folytatás egyetlen műveletből áll, a B értékének kiírásából:

115

Ezt a gép el is végzi.

Az INPUT szó tehát azt eredményezi, hogy a program végrehajtása az INPUT-ot tartalmazó utasítássornál leáll, és a gép mindaddig vár, amíg be nem gépelünk egy értéket. Ezt az értéket az INPUT után írt változónak adja, és a végrehajtás folytatódik.

Ezzel lehetővé válik, hogy minden végrehajtáskor új adatokat adjunk meg. Az adatokat a billentyűzeten kell begépelni.

Most már láthatjuk, hogy az átlagszámító feladat első öt sorát ki kell cserélni, és mindegyiket egy-egy beolvasó utasítássorrá kell átalakítani.

A NEW művelet végrehajtása után gépeljük be a programot még egyszer, de most már adatbeolvasó utasításssorokkal:

```
10 INPUT B
20 INPUT C
30 INPUT D
40 INPUT E
50 INPUT F
60 LET S=B+C+D+E+F
70 LET A=S/5
80 PRINT A
```

Az utolsó három sor változatlan maradt.

Hajtsuk végre most a programot többször egymás után! Látni fogjuk, hogy minden alkalommal új adatokat adhatunk meg, és az eredmény is eszerint alakul.

Ismereteink alapján meg lehet oldani az előző (összeg, szorzat) feladatot is úgy, hogy az A és B értékét a billentyűzetről adjuk meg minden esetben.

Végül nézzünk egy játékosabb példát! Próbáljuk ki, mennyire tudunk fejben szorozni 1 és 12 közé eső egész számokat. Építsük fel úgy a feladatot, hogy a gép írjon ki két, 1 és 12 közé eső egész számot (A és B), szorozzuk őket össze fejben, és gépeljük be az eredményt. Ezután a gép írja ki a pontos eredményt. Így ellenőrizni tudjuk, hogy jól számoltunk-e.

Milyen műveleteket kell végrehajtani? Először is a két számnak (A és B) 1 és 12 közötti egész értéket kell adni (pl. A=4, B=12), majd

ezeket ki kell írni, hogy elolvashassuk. Ezután nekünk kell „dolgozni". Ki kell számítani a szorzatot, és be kell írni. Vagyis a két szám kiírása után a gépnek be kell olvasni a mi eredményünket. Ezek után a gép kiszámítja a helyes eredményt, és ki kell írni. Lépésekre bontva a következő műveleteket kapjuk:

1. A értéke legyen 4
2. B értéke legyen 12
3. Írja ki A-t
4. Írja ki B-t
5. Olvassa be a szorzatot (C)
6. Szorozza össze A-t és B-t (D)
7. Írja ki a szorzatot (D)

A műveleteket átalakítjuk BASIC utasítássorokká, és az előző program törlése (NEW szó) után gépeljük be.

```
10 LET A=4
20 LET B=12
30 PRINT A
40 PRINT B
50 INPUT C
60 LET D=A*B
70 PRINT"SZERINTEM: ";D
```

Hajtassuk végre a programot!

A gép először kiírja a két tényezőt:

```
4
12
```

majd egy kérdőjellel jelzi, hogy egy érték — jelen esetben a szorzat — begépelésére vár. Begépeljük az eredményt:

```
? 48
```

Ezután a program kiírja a saját eredményét:

```
SZERINTEM: 48
```

Ezzel a végrehajtás befejeződik. Csalódottan állapítjuk meg, hogy a program ugyan korrektül működik, a feladatát mégsem látja el. Minden végrehajtásnál ugyanazt a kérdést teszi fel, tehát a második futás már teljesen érdektelen mindenki számára. Így nem értük el célunkat. Számunkra ugyanis az lenne a kedvező, ha minden alkalommal más, előre ki nem számítható számokat adna meg. Ekkor ugyanis minden végrehajtás új és érdekes lenne. Eddigi ismereteink alapján ez nem oldható meg, de van ismét olyan lehetőség a BASIC-ben, amivel ez sikerül.

Van egy függvény (részletesen lásd a 8. részben), amely véletlen-

számot állít elő. Ez a függvény lehetővé teszi, hogy egy változó ne ugyanazt az értéket kapja minden végrehajtáskor, hanem valamilyen véletlenszerű értéket. Ekkor a LET szó utáni részben az egyenlőségjel jobb oldalára nem egy konkrét értéket kell írni, hanem ezt a véletlenszám-előállító függvényt. Azt is meg kell adni, hogy a véletlenszám milyen tartományba essen. Kívánságunk szerint 1 és 12 közötti egész véletlenszámokat kell előállítani. A véletlenszám előállításának részleteit a 12. részben ismerjük meg, itt csak a függvény számunkra érdekes végső formáját mutatjuk be. A Commodore esetén 1 és 12 közötti véletlen egész számot következőképpen állíthatunk elő:

```
INT(RND(1)*12)+1
```

Ezt a bonyolult kifejezést az A és B változó értékének meghatározásánál a konkrét érték helyére kell írnunk:

```
10 LET A=INT(RND(1)*12)+1
20 LET B=INT(RND(1)*12)+1
```

A módosítás eredményeképpen minden végrehajtás alkalmával mind az A, mind a B valamilyen 1 és 12 közé eső egész értéket kap, amely véletlenszerű (megjósolhatatlan).

■ A HT gépen a véletlenszám előállítása formailag egyszerűbben érhető el:

```
10 LET A=RND(12)
20 LET B=RND(12)
```

▲ A PRIMO-nál a véletlenszám-előállítás formája:

```
10 LET A=RND(12)
20 LET B=RND(12)
```

● A Sinclair-gépeknél a véletlenszám-előállítás formája:

```
10 LET A=INT(RND*12)+1
20 LET B=INT(RND*12)+1
```

Az INT szó begépeléséhez előbb nyomjuk le a CAPS SHIFT és a SYMBOL SHIFT gombot. Ekkor megjelenik az E helyőr, ilyenkor billentyűk **feletti** zöld feliratok „élnek". Most nyomjuk le az R betű gombját, és megjelenik az INT a képernyő alsó sorában. A begépelés után visszaáll az L helyőr. Az RND ugyanezzel a módszerrel gépelhető be, de a T gomb lenyomása kell hozzá.

A két első utasítássor módosításával a módosítás be is fejeződik, hiszen az egyetlen hiányosságot pótoltuk. Ha most adunk ki végrehajtási parancsot, akkor a két megadott szám minden alkalommal más lesz, a játék pedig „élvezetessé" válik.

Munkánk befejeztével az eszközöket kikapcsolhatjuk (először a gé-

pet, utána a tv-t). Ezzel az utoljára begépelt program is elveszett. Erről meggyőződhetünk, ha újra bekapcsoljuk a gépet, s a RUN begépelése hatására nem történik semmi. Jó lenne, ha nem veszne el a munkánk! Ezzel majd a 2. részben foglalkozunk.

A következőkben rendszeresen fogunk előrehaladni, építve a most megismert fogalmakra, és jelentősen kibővítve, hogy összetett feladatokat is könnyen oldhassunk meg.

2. A SZÁMÍTÓGÉP HASZNÁLATA

Milyen részekből áll a számítógép? A billentyűzet leírása. Programok másolása kazettára és lemezre

Az első részben megismertük azokat az eszközöket, amelyek ahhoz szükségesek, hogy feladatokat tudjunk megoldani BASIC nyelven (1. ábra). A számítógépből, billentyűzetből és kijelzőből álló (tv) eszközkészlet elegendő ahhoz, hogy programokat (sorszámozott utasítássorozatokat) készítsünk, azokat végrehajtsuk. Ezek az eszközök lehetővé teszik, hogy a program végrehajtása közben a feladatokhoz számértékeket adjunk meg, s a feladatmegoldás eredményei a tv képernyőjén megjeleníthetők legyenek.

Az első rész végén kiderült az is, hogy ennek az ún. minimális eszközkészletnek korlátai is vannak. Mint emlékszünk rá, minden új feladat megoldásánál az előző feladatot megoldó program „elveszett", mert helyére kellett beírni az új programot. Ezért ha egy korábbi feladatot újra meg akarunk oldani, akkor annak az utasítássorait ismét be kell gépelni. Ha pl. a szorzatkitaláló program használata után újra szeretnénk átlagot számolni, akkor be kell gépelni előbb a NEW szót (parancsot), ezzel kitörölnénk a szorzatkitaláló programot, majd be kell írni a teljes átlagszámító programot. Ez nyilvánvalóan fáradságos és nem lelkesítő munka. Mit tegyünk azért, hogy az újra begépeléstől megmeneküljünk?

Először vizsgáljuk meg röviden, hogy mi is a jelenség oka! Eddig nem beszéltünk róla, de a számítógépnek van egy nagyon fontos része, ahol tárolni lehet mindent, amit mi begépelünk, illetve egyéb adatokat. Ez a számítógép **tárja.** A tár feladata az adattárolás. A tár tárolási egységekből épül fel. Egy tárolási egység egy betűt vagy számot, vagy írásjelet képes megőrizni. A betűt, számot és írásjelet együttesen **karakternek** vagy jelnek nevezzük, a tárolási egységet **bájtnak.** A bájt is még tovább bontható, de ezzel itt még nem foglalkozunk. Csupán annyit jegyezzünk meg, hogy egy bájtban 256 különböző karakter tárolható.

Ha például beírjuk az alábbi BASIC utasítássort:

```
10 INPUT B
```

akkor ennek a tárolásához néhány bájtra van szükség.

Látható, hogy a program egy meghatározható helyet foglal el a tárból. De nemcsak a programot kell tárolni, hanem a változókat, vagyis a programban előforduló adatokat is. Milyen hosszú programokat és

mennyi adatot tudunk tárolni a számítógép tárjában? Ez a tár méretétől függ. A legkisebb gépekben 1000 bájt (= 1 kilobájt = 1 kbájt) áll a programok rendelkezésére. A nagyobb gépeken 8, 16, 32, 48, 64 kbájt vagy ennél jóval nagyobb méretű tár van. Nekünk ekkorára nincs szükségünk.

Megjegyezzük, hogy a programkészítő nem használhatja a teljes tárkapacitást, mivel a gép saját maga is leköt egy bizonyos bájtmennyiséget.

A tár egyes bájtjai sorszámmal vannak ellátva, ez a bájtok **címe**. Ez lehetőséget ad arra, hogy szükség esetén az egyes bájtokat egyénileg is tudjuk kezelni, azaz valamit bele tudjunk írni, illetve a tartalmát ki tudjuk olvasni. Ezt a BASIC nyelv is lehetővé teszi. (Erre majd még később visszatérünk.) Nagyon fontos, hogy tulajdonképpen nem is kell azzal törődnünk, mi hova kerül a tárban, vagy honnan lehet kiolvasni, mivel ezeket a feladatokat a gép (illetve az ún. operációs rendszer) helyettünk elvégzi.

Az általunk használt kis számítógépekben általában kétféle tár van:

— ROM: csak kiolvasható tár
— RAM: írni is lehet bele, és olvasható is

A ROM számunkra nem hasznosítható, mivel nem tudunk beleírni. Mi csak a RAM-ot használhatjuk. Ennek viszont az a kellemetlen tulajdonsága, hogy tartalma a tápfeszültség kikapcsolásakor elvész. Ez az oka annak, hogy kikapcsoláskor a gépben, illetve a tárban lévő program elvész. Ezt a hiányosságot úgy tudjuk megszüntetni, ha a tárnak azokat a részeit, ahol a program elhelyezkedik, egy olyan eszközre másoljuk át, amely a beírt adatokat sokáig képes megőrizni. Ilyen eszköz a közismert **magnókazetta** vagy a kevésbé közismert, de hasonló fizikai elv szerint működő **mágneslemez.** Itt bemutatjuk ezeknek az eszközöknek a használatát.

Előbb azonban térjünk vissza még egyszer az előző rész feladataihoz. Joggal merül fel olyan igény, hogy a programok által szolgáltatott eredményeket is meg lehessen őrizni valahogy, hiszen a képernyőről előbb-utóbb elvész. Az lenne a jó megoldás, ha a számunkra fontos eredményeket (adatokat) papírra is kiírná a gép, így ezek megmaradnának nekünk. Vagyis jó lenne, ha a géphez egy **kiíró** is tartozna, amelylyel adatokat lehet kiírni.

Ebben a részben a már röviden bemutatott vagy említett eszközök használatát ismerjük meg részletesen:

— billentyűzet
— mágneslemezegység
— kazettás tároló

A nyomtató használatát a 4. részben mutatjuk be.

A BILLENTYŰZET

A billentyűzetek felépítése nagyjából hasonló, általában csak apróbb részletekben térnek el egymástól. A billentyűzet nagyon emlékeztet az írógépek billentyűzetére. Az a legnagyobb különbség a kettő között, hogy a terminál jelkészlete nagyobb.

A billentyűket három csoportra oszthatjuk:

- **karakterbillentyűk:** lenyomásukkor a rájuk rajzolt jel (betű vagy szám) megjelenik a megjelenítőn,
- **funkcionális billentyűk:** lenyomásukkor önállóan vagy a számítógép segítségével valamilyen műveletet végeznek el (soremelés, törlés, bevitel stb.).
- **vegyes használatú billentyűk:** önállóan használva karakterbillentyűk, valamely funkcionális billentyűvel együttesen lenyomva a funkció pontosítására szolgálnak (a későbbiekben mutatunk be erre példákat).

A billentyűk többsége két állásban használható (két jel van rajtuk). A normál eset az alsó állás (megfelel az írógépek „kisbetűírás" állásának). Ilyenkor, ha olyan billentyűt nyomunk le, amelyen két jel van, az alsó jel íródik ki. Ha viszont olyan karakterbillentyűt nyomunk le, amelyen csak egy jel van (többnyire betű), akkor a billentyűre rajzolt betű jelenik meg. A SHIFT billentyű lenyomásával a felső állásba kapcsolunk. Ilyenkor a billentyűk felső részén látható jelek írhatók ki.

Mivel a számítástechnika angolszász nyelvterületen alakult ki, a billentyűkön az angol ábécé betűi vannak. Magyar ékezetes betűk általában nincsenek a terminálokon (a PRIMO kivételével). Mi azt javasoljuk – és ezt is fogjuk követni –, hogy azoknál a gépeknél, amelyeken nincs ékezetes betű, az **ékezetes betűk** helyett ezek **ékezet nélküli változatát** írjuk ki a szövegekben (pl. É helyett E-t). Bizonyára zavaró és szokatlan első látásra, de véleményünk szerint ezt könnyebb megszokni, mint a kettőzött magánhangzó írását (pl. É helyett EE). A magánhangzó megkettőzése a szavak hosszát is megnöveli, ami a táblázatok fejrovat-beosztásának megtervezésében okoz gondot.

Megfigyelhetjük, hogy a terminál billentyűzetén mind a 10 számjegy szerepel a legfelső sorban. Vigyázzunk, hogy számok írásához csak ezeket a billentyűket használjuk! Aki tud gépelni, annak kezdetben figyelnie kell, hogy külön van O betű és nulla számjegy, valamint hogy az L nem 1-et jelent.

BASIC programokban a billentyűzetnek csak meghatározott karakterei használhatók (pl. betűk és számok). Ezeket részletesen az egyes gépek billentyűzetének ismertetésekor soroljuk fel.

Érdemes minél előbb megtanulni, hogy a tizedesjel – az európai szabványtól eltérően, amerikai szokás szerint – a pont, tehát mindig

tizedespontot kell használni! A vessző funkciója más, tizedesként nem használható.

A billentyűk fontos szolgáltatásokat tudnak nekünk nyújtani. Ilyen a **képernyő törlése,** amelynek eredményeként a képernyőről eltűnik minden kiírás, és a helyőr az 1. sor 1. karakterpozíciójára (bal felső sarok) áll. Ez nem jelenti a program törlését, csak a képernyőt „tisztítja meg".

A funkcionális billentyűk teszik lehetővé, hogy a begépelt szöveget módosítsuk, a téves **begépeléseket kijavítsuk.** Itt csak az egyszerűbb lehetőségeket tekintjük át, a függelékben részletes leírás található a programok javítási módjairól.

A billentyűk között vannak egyéb speciális funkciókat végrehajtók is. Ezeket, valamint a jelkészletet, a képernyő törlését és a gépelési hibák kijavítását az alkalmazott négy géptípusnál külön-külön mutatjuk be.

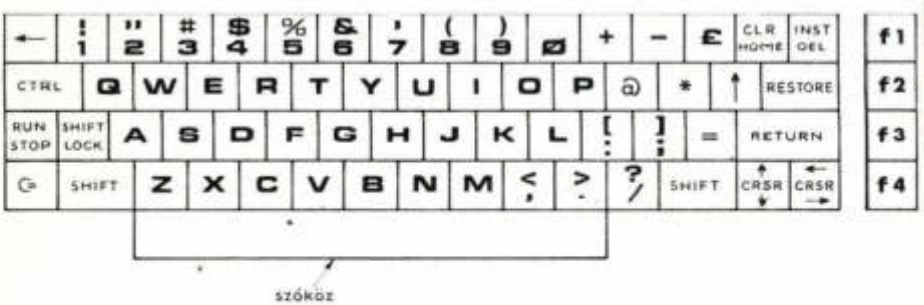
Megjegyezzük, hogy a billentyűk egy része ismételni is tud, azaz ha tartósan lenyomva hagyjuk, akkor mindaddig újabb karakter kerül a képernyőre, amíg a billentyűt lenyomva tartjuk.

A COMMODORE–64 BILLENTYŰZETE

A Commodore–64 billentyűzete a 2. ábrán látható. A gépen írt programokban következő karakterek használhatók:

Betűk: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Számjegyek: 0 1 2 3 4 5 6 7 8 9
Speciális jelek: + – * / ↑ = < > $ " ' ( ) : ; ? , . @ # ! % és a szóköz

A képernyő a SHIFT és a CLR/HOME billentyű együttes lenyomásával **törölhető.** Ha csak a CLR/HOME gombot nyomjuk le, akkor a gép csupán a „HOME" funkciót hajtja végre. Ennek eredményeként a helyőr a bal felső pozícióba ugrik, de a képernyő nem változik meg.



2. ábra. A Commodore–64 billentyűzete

Begépelés közbeni hibát az INST/DEL nyomógombbal javíthatunk ki, ha még a RETURN gombot nem nyomtuk le. Tegyük fel, hogy az alábbi sort kell begépelnünk:

```
100 PRINT A
```

de eltévesztettük, és az alábbi szöveget gépeltük be:

```
100 PRIT A
```

Mielőtt a RETURN gombot lenyomtuk, észrevesszük a hibát. Ekkor az INST/DEL gombot lenyomjuk, amely kitörli az utolsónak begépelt karaktert. Ha többször lenyomjuk, többet töröl visszafelé haladva. A begépelt szöveg nem teljesen rossz, az eleje még helyes:

```
100 PRI
```

Elég tehát idáig visszalépegetni az INST/DEL gombbal, majd innen folytathatjuk a beírást, most már helyesen:

```
100 PRINT A
```

Ha olyankor vesszük észre a hibát, amikor a RETURN gombot már lenyomtuk, akkor a javítást a függelékben leírt módon kell végrehajtani.

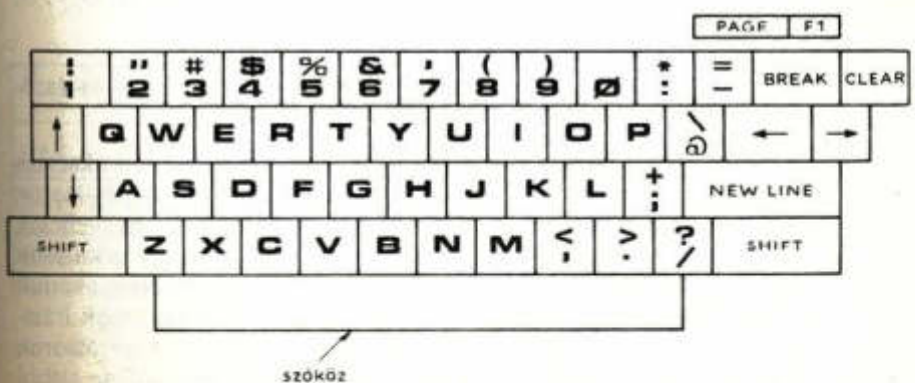
A Commodore-on van egy STOP RUN feliratú billentyű, amellyel a programok végrehajtását le lehet állítani. A lenyomás pillanatában még folyamatban lévő utasítást a gép végrehajtja, és a

BREAK IN *sorszám*

üzenet kiírása után a program végrehajtása leáll.

A HT–1080Z SZÁMÍTÓGÉP BILLENTYŰZETE

- A gép billentyűzete a 3. ábrán látható.



3. ábra. A HT–1080Z számítógép billentyűzete

A gép BASIC jelkészletének karakterei:

Betűk: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Számjegyek: 0 1 2 3 4 5 6 7 8 9
Speciális jelek: + − * / ↑ = <> $ " ' ( ) : ; ? , . @ # & ! % és a szóköz

A képernyőt a Clear gombbal lehet törölni. A gomb lenyomásakor a képernyőről minden jel eltűnik, és a helyőr a bal felső sarokba ugrik.

Begépelés közben a hibát a Commodore-nál leírt módon lehet javítani. A visszalépés és karaktertörlés itt a ← jelű gomb hatására megy végbe.

Ha a hiba a sor elején van, vagy túl sok hiba van a begépelendő sorban, akkor nem érdemes a visszalépéses törlést végrehajtani, mert a SHIFT és a ← gomb együttes lenyomása törli a teljes sort, és a beírást újra lehet kezdeni.

A program végrehajtása a BREAK gombbal szakítható meg. A gomb lenyomásakor végrehajtás alatt álló utasítás befejeződik, és utána a program leáll a

BREAK IN *utasítássorszám*

üzenet kiírása után. Az utasítássorszám a leállás helyét jelöli.

A PRIMO SZÁMÍTÓGÉP BILLENTYŰZETE

▲ A PRIMO számítógép billentyűzete a 4. ábrán látható.
A gép BASIC jelkészlete a következő karakterekből áll:

Betűk: a, á, b, c, d, e, é, f, g, h, i, í, j, k, l, m, n, o, ó, ö, ő, p, q, r, s, t, u, ú, ü, ű, v, w, x, y, z,
A, Á, B, C, D, E, É, F, G, H, I, J, K, L, M, N, O, Ö, P, Q, R, S, T, U, Ü, V, W, X, Y, Z
Számok: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Speciális jelek: ↑ , ! " # $ % & ' ( ) * + , − . / : ; < > = és a szóköz

A billentyűzet nagy előnye, hogy a magyarban használt ékezetes betűk is — a nagy Ó, Ő, Ú, Ü és Í kivételével — megtalálhatók rajta, és ezek a programokban alkalmazhatók, valamint nemcsak a nagybetűk, hanem a kisbetűk is használhatók. Így a kiírások a magyar nyelvben szokásos módon elvégezhetők. Nem akarjuk befolyásolni a PRIMO-felhasználókat, hogy a programok írásához kis- vagy nagybetűt használjanak. A program-utasítássorok megírásához ugyanis mindkettő használható. Például az alábbi két sor egyenértékű:

150 print a+b
és 150 PRINT a+b

A kis- és nagybetű váltás a SHIFT gombbal végezhető el az írógéphez hasonlóan. Ha valaki a többi géphez hasonlóan csak nagybetűket akar használni, akkor (a bekapcsolás után) nyomja le az UPPER billentyűt.
Ennek hatására csak a nagybetűk jelennek meg. A többi gombról, ahol két jel van, továbbra is az alsó jelenik meg. A felső jel csak akkor kerül a képernyőre, ha a SHIFT-et is lenyomtuk. Ha még egyszer lenyomjuk az UPPER-t, akkor a gép visszatér a kisbetűs üzemmódba.

A képernyőt a CLS gomb lenyomásával lehet törölni. Ilyenkor a képernyő törlődik, és a helyőr a bal felső sarokba ugrik.

Begépelés közbeni hibát a ← gomb segítségével javíthatunk ki, ha a sorzáró RETURN gombot még nem nyomtuk le. A ← gomb lenyomásával a helyőr balra megy, és minden lenyomás után törli az útjában álló jeleket. Vissza kell menni a helyes rész végéig, és onnan a bevitelt helyesen meg kell ismételni.

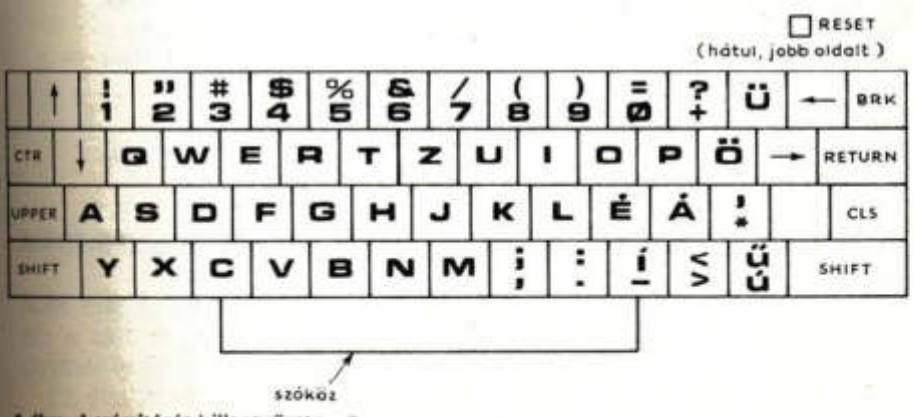
Ha egy sorban sok a hiba, vagy a legelején van hiba, akkor célszerű a sort a ← és a SHIFT együttes lenyomásával törölni és a bevitelt megismételni.

Ha a RETURN-t már lenyomtuk, akkor a függelékben található módon lehet a hibát kijavítani.

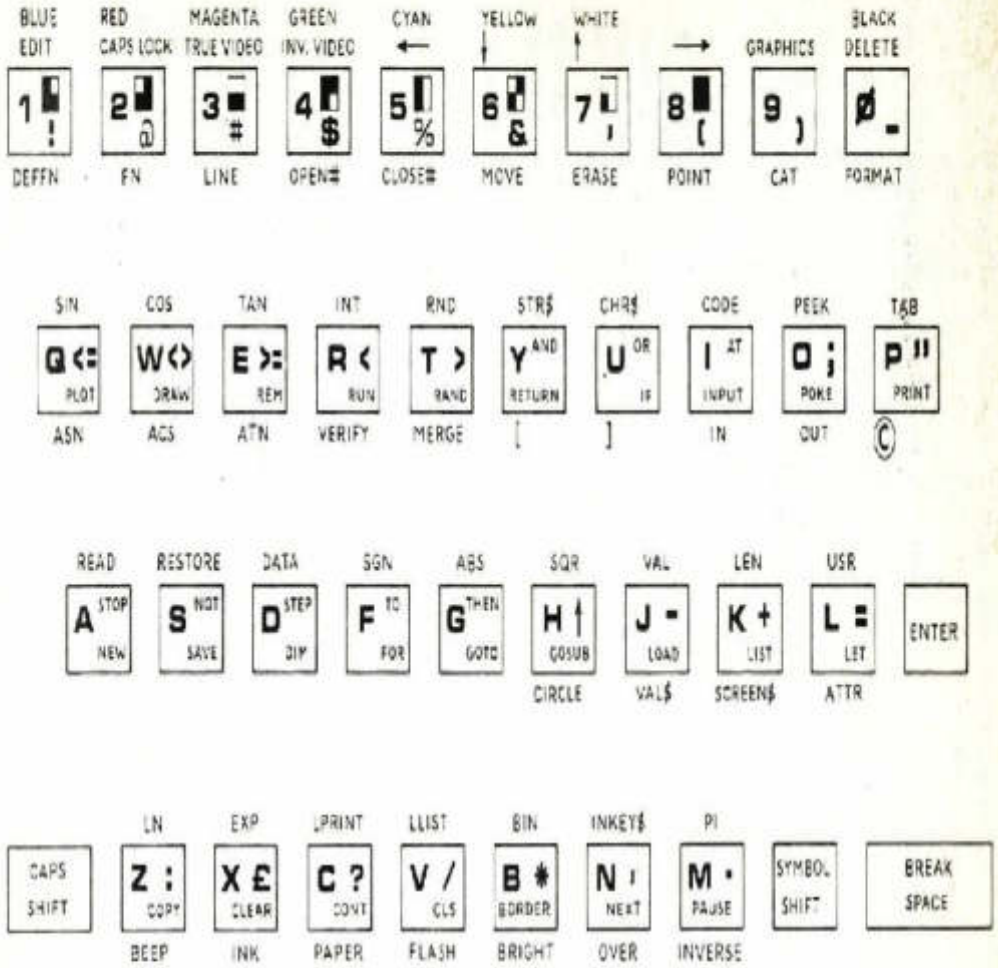
A BRK billentyű lenyomásával a program végrehajtása az éppen végrehajtott utasítás befejezése után leáll, és a

Break in *utasítássorszám*

szöveget írja ki. Az utasítássorszámmal jelöli, hogy hol állt le.



4. ábra. A számítógép billentyűzete



5. ábra. A Sinclair billentyűzete (Spectrum)

- A Sinclair-gépek billentyűzete az 5. ábrán látható.

A gépen írt programokban a következő **karakterek** használhatók:

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

Mint látható, a kisbetűk is használhatók a gépen.

A **képernyőt** a CLS (V gomb) lenyomásával lehet **törölni**, ha a K helyőr villog.

Az alsó, beviteli sorba beírt szöveget az ENTER gomb lenyomása előtt a többi géphez hasonlóan lehet **javítani.** A begépelt szöveg utolsó jeleit a CAPS SHIFT és a 0 gomb egy-egy lenyomásával lehet kitörölni.

Itt is addig kell törölni (visszafelé haladni), amíg a maradék rész hibátlan nem lesz. Figyeljük meg, hogy ha olyan szövegrészhez ér a helyőr visszafelé haladtában, amelyet egyetlen billentyű lenyomásával írhatunk le (pl. PRINT), akkor ennek a törlése is a CAPS SHIFT és a 0 egyetlen lenyomására megy végbe.

A Sinclair-gépek formailag hibás szöveget nem fogadnak el. Egy kérdőjel jelzi az ENTER lenyomása esetén, hogy hiba van a sorban. A hibát a bemutatott visszafelé haladó törléssel és újra begépeléssel javíthatjuk ki.

A program végrehajtása a CAPS SHIFT és a BREAK gomb együttes lenyomásával szakítható meg. A lenyomás hatására a végrehajtás alatt álló utasítás befejeződik, utána a program leáll, és az

L Break into program *sorszám*

üzenet jelenik meg.

MÁGNESLEMEZ ÉS KAZETTA HASZNÁLATA

Mint már korábban láttuk, a mágneslemezegység és a kazettás magnó azt a célt szolgálja, hogy a számítógépben tárolt programokat a gép tárjából lemezre vagy kazettára kimásoljuk megőrzés végett.

Ez általában akkor szükséges, ha készítettünk egy új programot, és miután használtuk, vagy ki akarjuk kapcsolni a gépet, vagy más progra-

mot kívánunk írni. A tárban lévő program mindkét esetben elveszne, ezért előbb kimásoljuk egy ún. **külső tároló** berendezésre (mágneslemez-egység vagy magnó).

Ha újra akarjuk használni a már kimásolt programot, akkor a külső tárolóeszközről a programot be kell másolni a gép tárjába. Mivel az ilyen másolás hatására a program nem törlődik a mágneslemezről vagy a kazettáról, a tárba bemásolt programot szabadon lehet törölni, nem kell ismételten visszamásolni.

Tegyük fel, hogy mind az átlagszámító, mind a szorzatkitaláló programot kazettára másoltuk. Ha az átlagszámító programot akarjuk használni, akkor a programot bemásoljuk. Ha már nem akarjuk tovább végrehajtani, és helyette a szorzatkitalálót szeretnénk használni, akkor nyugodtan másoljuk be ez utóbbi programot anélkül, hogy az átlagszámítót kimásolnánk (kimentetnénk), mivel az megmaradt a külső tároló-egységen.

Rögtön felmerül a kérdés, hogy mi történik akkor, ha már több programunk van egy kazettán vagy mágneslemezen, hogyan tudjuk kiválasztani a kívánt programot a sok közül. Ez csak akkor lehetséges, ha a programnak valamilyen nevet adunk a másoláskor. A program a megadott névvel kerül a külső háttértárolóra, és ha vissza akarjuk másolni, akkor a nevével tudjuk kiválasztani.

Egy jó tanács. Akinek nincs saját számítógépe, és kölcsöngépet tud használni, vegyen magának 1–2 magnókazettát vagy lemezes gép esetén lemezt (a pontos típust a kölcsönadótól meg kell kérdezni!), amelyre a saját programjait gyűjtheti. Mágneslemez esetén a lemezt a használat előtt inicializálni kell (lásd a függelékben).

A kazettás magnó ugyanúgy tárolja a programokat, mint a zeneszámokat. Egy kazettán több program is lehet mindkét oldalon. Ha több kazettánk van, érdemes valamilyen megkülönböztető nevet adni nekik, és feljegyezni, hogy melyiken mi van.

Most nézzük meg, hogy az egyes gépeknél a műveletek hogyan hajthatók végre. Itt nem részletezzük, de nyilvánvaló, hogy a kazettára írás vagy a kazettáról való másolás előtt a kazettát be kell helyezni a magnóba, a kazettaajtót be kell zárni stb.

A MAGNÓ ÉS MÁGNESLEMEZEGYSÉG HASZNÁLATA A COMMODORE-ON

A Commodore-nál mind a mágneslemez, mind a kazettás magnó használatát bemutatjuk.

Kazettára való másoláskor a következő szöveget kell begépelni:

SAVE *"programnév"*, 1

ahol

SAVE — a tárolásra utaló szó,
programnév — a program neve, amellyel majd visszamásolhatjuk a programot,
1 — a magnó eszközszáma.

Jegyezzük meg, hogy a programnév legfeljebb 16 tetszőleges jelből állhat (pl. MINTA1).

A szöveg (parancs) begépelése és a RETURN lenyomása után megjelenik a

PRESS RECORD & PLAY ON TAPE

üzenet a képernyőn, ami azt jelenti, hogy nyomjuk le a felvételt indító gombot a magnón. A másolás végén megjelenik a képernyőn a

SAVING *programnév*
READY

üzenet, ekkor a magnót le lehet állítani.

A kazettára vagy lemezre másolás közben előfordulhatnak olyan hibák, amelyek miatt a kimásolt program használhatatlanná válik. Ezért jó meggyőződni arról, hogy a kimásolás hibátlan-e. Van a BASIC-ben olyan lehetőség, amellyel meg lehet vizsgálni, hogy a kimásolt és tárban levő program egyezik-e. Ezt a műveletet akkor kell elvégezni, amikor a másolás befejeződött, de a program még a tárban van.

A vizsgálathoz a következő parancsot kell kiadni:

VERIFY *"programnév"*, 1

ahol

VERIFY — a vizsgálat parancsszava,
programnév — a kimásolt program neve,
1 — a kazettás magnó száma.

Ha a parancs kiadása előtt a magnó a vizsgálandó program kezdetén áll, akkor a programnév és a kazettás magnó száma elhagyható:

VERIFY

A parancs kiadása után a

PRESS PLAY ON TAPE

üzenet kéri, hogy indítsa el a magnót. Ekkor megjelenik a

SEARCHING FOR *programnév*

üzenet, amely jelzi, hogy a gép keresi a megadott nevű programot.

Ha megtalálta, akkor ezt is tudatja:

FOUND *programnév*

Ha a két program egyezik, akkor az OK és a READY üzenet, másolási hibánál a

? VERIFY ERROR
READY

jelenik meg. Ilyenkor a kimásolást meg kell ismételni.

Arra már nekünk kell vigyázni, hogy a gép ne másoljon egymásra programokat. A kimásolás előtt ezért a magnószámláló segítségével vagy más módon a szalagot szabad helyre kell állítani.

A szalagról való programbeolvasáshoz az alábbi parancsot kell begépelni:

LOAD *"programnév"*, 1

ahol

LOAD	— a bemásolásra utaló szó,
programnév	— a beolvasni kívánt program neve,
1	— a magnó eszközszáma.

A szöveg begépelése és a RETURN lenyomása után a

PRESS PLAY ON TAPE

üzenet jelenik meg, ami azt jelenti, hogy a magnón meg kell nyomni a lejátszó gombot. Ha ezt megtettük, akkor a gép elkezdi olvasni a szalagon levő programokat, illetve azok neveit.

Ha megtalálta a kívánt nevű programot, akkor kiírja a

FOUND *programnév*

üzenetet, amellyel jelzi a megtalálás tényét. Ahhoz, hogy a bemásolás is megtörténjék, meg kell nyomni a C=, CTRL, ← vagy szóköz gombot. Ha ezt megtettük, akkor elkezdődik a másolás. Erre utal a megjelenő

LOADING

üzenet. A bemásolás végét a

READY

üzenet jelzi; a magnót le lehet állítani. Ekkor a program bent van a tárban, és a

RUN

paranccsal a program végrehajtható, mintha most gépeltük volna be. A művelet eredményeként a program természetesen a kazettán marad.

A **mágneslemezre** másolás és az onnan való visszamásolás valamivel egyszerűbben megy végbe, mivel a helykeresést a mágneslemezegység maga intézi, továbbá az indítás és a leállítás is felesleges.

A mágneslemezre másolás a következő paranccsal hajtható végre:

SAVE *"programnév"*, 8

ahol

SAVE — a tárolásra utaló szó,
programnév — a program neve, amellyel majd visszamásolhatjuk a programot,
8 — a mágneslemezegység száma.

A programnév itt is 16 jel hosszúságú lehet.

Megjegyezzük, hogy a mágneslemez száma 9, 10, 11 is lehet. A másolás megkezdésekor a

SAVING *programnév*

üzenet, a végén pedig a

READY

üzenet jelenik meg. Ez jelenti a másolás befejezését.

A másolás helyességét itt is ellenőrizzük a tárban levő program törlése előtt. A parancs formája hasonló a kazettás magnónál bemutatott parancshoz:

VERIFY *"programnév"*, 8

Hibás másolásnál itt is a

? VERIFY ERROR

ellenkező esetben csak a READY üzenet jelenik meg. Hiba esetén a másolást meg kell ismételni.

A visszamásolás az alábbi paranccsal végezhető el:

LOAD *"programnév"*, 8

ahol

LOAD — a beolvasás parancsszava,
programnév — a beolvasni kívánt program neve,
8 — a mágneslemezegység száma.

A parancs kiadása után a

SEARCHING FOR *programnév*

üzenet jelenik meg, amely arra utal, hogy a gép keresi a kívánt programot. Ha már megtalálta, akkor a

LOADING

üzenet mutatja, hogy a beolvasás megkezdődött. A beolvasás végét a

READY

üzenet jelenti. Ekkor a program már végrehajtható!

A mágneslemezegységnek olyan szolgáltatása is van, hogy meg lehet tudni, milyen nevű programok vannak egy lemezen, tehát nekünk nem kell erről nyilvántartást vezetni.

A lemezen levő programok nevét két lépésben lehet kiíratni. Az első lépésben beolvastatjuk a lemez tartalomjegyzékét a már ismert LOAD paranccsal:

LOAD "$", 8

ahol

$ — a tartalomjegyzék neve.

A READY üzenet megjelenése után a

LIST

szó begépelése és a RETURN lenyomása után (listázási parancs) a képernyőn megjelenik a lemezen levő programnevek listája.

Felhívjuk a figyelmet arra, hogy a lemez érzékeny eszköz. Nem szabad erős mágneses térbe helyezni (villamos, trolibusz), nem szabad hajlítgatni, a lemez felületét valamilyen kemény eszközzel megérinteni, beszennyezni stb. Bármilyen sérülés a lemez tönkremenését okozhatja! Célszerű kemény lapok között tárolni.

MAGNÓHASZNÁLAT A HT GÉPEN

- A HT gépen a mágneskazettára kimásolás a következő paranccsal végezhető el:

CSAVE#−1,*"programnév"*

ahol

CSAVE#−1 — a kimásolásra utaló szó,
programnév — a program neve, amellyel a programot vissza lehet másolni.

A program neve 1 karakter lehet, illetve egynél hosszabb névből csak az első karaktert használja. A másolási parancs begépelése után le kell nyomni együtt a REC és a PLAY gombot, majd a NEW LINE gombot. Ezzel a másolás elkezdődik. A másolás befejezését a

READY

üzenet jelzi.

A kimásolás a HT gépnél is lehet hibás. Ilyenkor a kimásolt program használhatatlan. Ezért a program kitörlése előtt érdemes összehasonlítással ellenőrizni a másolás helyességét. Erre van lehetőség a gép BASIC-jében. Az összehasonlítás előtt a kazettát visszacsévéljük a kimásolt program elejére, elindítjuk a magnót a PLAY gomb lenyomásával, és kiadjuk a parancsot:

CLOAD?*"programnév"*

ahol

CLOAD? — az ellenőrzés parancsszava,
programnév — az ellenőrzendő program neve.

A parancs begépelése és a NEW LINE lenyomása után a gép elkezdi keresni a megadott nevű programot. Ha megtalálta, a képernyő jobb felső sarkában két csillag jelenik meg. Ezután a kazettán és a tárban levő programokat összehasonlítja. Ha a kazettán levő program hibátlan, akkor a jobb oldali csillag villog. Hibás esetben a végén a

BAD

hibátlan esetben a

READY

üzenet jelenik meg. Ha a másolt program hibás, a kimásolást meg kell ismételni.

A beolvasás a következő paranccsal hajtható végre:

CLOAD#−1,*"programnév"*

ahol

CLOAD#−1 — a beolvasás parancsszava,
programnév — a beolvasni kívánt program neve.

Ezután le kell nyomni a magnó PLAY gombját és a NEW LINE gombot. Ekkor a gép elkezdi keresni a megadott nevű programot. Ha megtalálta, két csillag jelenik meg a képernyő jobb felső sarkában. A jobb oldali csillag időről időre felvillanással jelzi, hogy folyik a program bemásolása. A művelet végén a

READY

üzenet jelenik meg. Ezután a program végrehajtható.

MAGNÓHASZNÁLAT A PRIMO-NÁL

A PRIMO-nál egy program kimásolása az alábbi paranccsal hajtható végre:

SAVE *"programnév"*

ahol

SAVE — a kimásolásra utaló szó,
programnév — a program neve, amellyel a programot vissza lehet másolni.

A programnév legfeljebb 16 jelből állhat (pl. SZORZAT).

Ezután el kell indítani a magnót felvétel állásban, és le kell nyomni a RETURN gombot.

A másolás befejeztével az

Ok

üzenet jelenik meg.

Másolás közben előfordulhatnak hibák, amelyek azt eredmé nyezik, hogy a kimásolt program használhatatlanná válik. E nagyon kellemetlen, mert ha a tárból is kitöröljük a programo a program elvész, és újra be kell gépelni. Ezért célszerű a máso programot ellenőrizni, hogy hibátlan-e.

A PRIMO-nál az ellenőrzés nem a tárban levő progra összehasonlításával megy végbe, hanem a kazettán levő progra formai helyességének ellenőrzésével. Ha a kazettára máso program formailag helyes, akkor tartalmilag is hibátlan. A ellenőrzés előtt a kazettát legalább az ellenőrzendő progra elejére kell állítani, és be kell gépelni az alábbi parancsot:

TEST *"programnév"*

ahol

TEST	– az ellenőrzés parancsszava,
programnév	– az ellenőrzendő program neve.

Majd lenyomjuk a RETURN gombot és a magnó PLAY gombját Ezután ugyanaz a folyamat zajlik le, mint a LOAD paranc begépelésekor (lásd később). Ha a *hsz* (hibaszám) értéke 0 akkor a kazettán levő program hibátlan, egyébként hibás. A ellenőrzés végén az Ok jelenik meg.

A program visszaolvasását a következő parancs hajtja végre:

LOAD *"programnév"*

ahol

LOAD	– a beolvasás parancsszava,
programnév	– a beolvasni kívánt program neve.

Majd le kell nyomni a RETURN gombot, és el kell indítani a magnót lejátszásra. Ezután a gép keresni kezdi a megadott nevű programot. Ha közben más nevű programot talál, akkor azt kiírja:

SKIP:*programnév*

Ha a kívánt programot megtalálja, akkor elkezdi a betöltést, és a művelet végén üzenetet ír ki:

rsz hsz FOUND:*programnév*

ahol

rsz	– a betöltött program méretére vonatkozó szám (rekordok száma),

hsz — a hibák száma,
FOUND — a megtalált (program).

A betöltés csak akkor helyes, ha a hibaszám nulla. Ellenkező esetben a betöltést meg kell ismételni.

MAGNÓHASZNÁLAT A SINCLAIR-GÉPEKNÉL

- A Sinclair-gépeknél egy program kimásolása a következő paranccsal hajtható végre:

SAVE *"programnév"*

ahol

SAVE — a kimásolásra utaló szó,
programnév — a program neve, amellyel a programot vissza lehet másolni.

A programnév legfeljebb 10 jelből állhat.

A magnó csatlakozóját előzőleg a számítógép EAR-kivezetésébe kell bedugni. Lenyomjuk az ENTER gombot és ezután megjelenik a

Start tape, press any key

üzenet. Ekkor el kell indítani a magnót felvétel állásban, és a gépen valamelyik gombot le kell nyomni. A másolás végén a 0 OK, üzenet jelenik meg.

A kimásolt programban lehet hiba, ilyenkor használhatatlan. Ezért a programot kitörlése előtt hasonlítsuk össze a kimásolt programmal. A kazettát állítsuk vissza a program kezdetére, a magnó csatlakozóját a MIC-kivezetésbe dugjuk be, és adjuk ki a

VERIFY

parancsot. A parancs formája:

VERIFY *"programnév"*

ahol

VERIFY — az ellenőrzés parancsszava,
programnév — az ellenőrzendő program neve.

A parancs begépelése után el kell indítani a magnót lejátszásra. Ha a kimásolt és a tárolt program egyezik, akkor az OK üzenet, hibánál az R hibaüzenet jelenik meg. Az utóbbi esetben a kimásolást meg kell ismételni.

Kazettán levő programot a következő paranccsal lehet bemásolni:

LOAD *"programnév"*

ahol

LOAD — a beolvasás parancsszava,
programnév — a beolvasni kívánt program neve.

A bemásolás előtt a magnócsatlakozót dugjuk be a MIC-kivezetésbe. A parancs beírása után nyomjuk le az ENTER sorszáró billentyűt, és indítsuk el a magnót lejátszásra. A bemásolás végén megjelenik a K helyőr. Ezután a program végrehajtható.

NÉHÁNY SZÓ AZ OPERÁCIÓS RENDSZERRŐL

Végül nézzünk még valamit! Az eddigiekben láthattuk, hogy attól kezdve, hogy a gépet bekapcsoljuk,

READY

vagy

Ok

üzenet, vagy a K helyőr jelzi, hogy a gép készen áll valamilyen feladat végrehajtására. Ha van valamilyen, a gép által megoldható feladatunk, akkor egy megfelelő parancsot beírunk, és a gép azt végrehajtja. Hogyan lehetséges ez? A gépben már a bekapcsoláskor vannak programok, amelyek ezeket a parancsokat végrehajtják. Azokat a programokat, amelyek a számítógéprendszer egészét irányítják, elemeinek működését összehangolják, a felhasználót a program készítésében támogatják, együttvéve **operációs rendszernek** nevezzük.

Az operációs rendszer gondoskodik arról, hogy a begépelt program bekerüljön a tárba, hogy a RUN parancs hatására a program végrehajtódjék, hogy a SAVE jellegű parancs hatására a program kimásolódjék a kazettára, és így tovább.

Az operációs rendszert **paranccsal** utasítjuk valamilyen művelet végrehajtására. Ilyen parancs az eddig megismertek közül a NEW, a SAVE, a RUN stb. A programon belüli műveletek parancsait megkülönböztetésül **utasításnak** nevezzük.

Az operációs rendszer **üzenetekkel** ad választ egy parancs végrehajtása után, valami hiba esetén, vagy üzenettel tudatja, hogy éppen mit csinál (például a bemásolás közben kiírja, hogy LOADING a betöltés folyamatban van).

Az operációs rendszer szolgáltatásai megkönnyítik a felhasználó munkáját. Több szolgáltatáshoz több program kell, több programhoz nagyobb tár. Azt mondhatjuk tehát, hogy kis tárkapacitású gépben kis operációs rendszer van, nagyobb tárkapacitású gépben nagyobb operációs rendszer lehet, amely több szolgáltatást tud nyújtani, mint egy kisebb gépé. (Azért ne higgyük, hogy az operációs rendszer szolgáltatásainak színvonala egyedül az elfoglalt tár méretétől függ.)

Minden gépnek van saját operációs rendszere. Vannak általános operációs rendszerek, amelyek különböző gépeken (majdnem) ugyanazt a szolgáltatást tudják nyújtani. Ilyen operációs rendszer például a személyi számítógépeken használható CP/M. Ennek néhány fontosabb szolgáltatását a függelék tartalmazza.

Ellenőrző kérdések és feladatok

1. Magyarázza meg, hogy mi a különbség az alábbi két szöveg között:

```
10 PRINT"VEGRE!"
PRINT"VEGRE!"
```

Mi történik, ha az elsőt gépeljük be, és mi akkor, ha a másodikat? Minek a hatására lesz az eredmény ugyanaz mind a két szövegnél?

2. Mi a különbség az alábbi két program között?

a)
```
10 LET A=8
20 PRINT A
```

b)
```
10 INPUT A
20 PRINT A
```

3. Mi a különbség az alábbi két sor között?

a) PRINT"5+3"
b) PRINT 5+3

4. Gépeljük be a szorzatkitaláló programot, és másoljuk ki kazettára vagy lemezre! Ellenőrizzük, hogy jó-e a másolat!
Módosítsuk a programot az 1. részben leírtak szerint úgy, hogy A és B véletlenszerű legyen. A módosítást a 3. függelékben leírtak szerint végezzük el! Az új programot szintén másoljuk ki (de más névvel)!

3. A PROGRAMOZÁS MÓDSZERE I.

A programkészítés módszere és lépései,
a 1. feladat megoldásának első része

Ebben a részben a programkészítés alapelveit ismerjük meg. Számítógépet nem fogunk használni csak a következő részben, amikor az itt elkezdett feladat megoldását befejezzük.

A PROGRAM

Vizsgáljuk meg az 1. részben megoldott átlagszámítási feladatot megoldó programot! Nézzük meg, hogy az egyes utasítássorok milyen feladatot látnak el:

```
10 INPUT B
20 INPUT C
30 INPUT D
40 INPUT E
50 INPUT F
60 LET S=B+C+D+E+F
70 LET A=S/5
80 PRINT A
```

Az első öt utasítás hasonló feladatot lát el: adatokat kér be a billentyűzetről. Az itt beolvasott adatokat a 60-as és közvetve a 70-es sorszámú utasítás felhasználja, ezért ezek az adatok a megoldás **kiinduló** vagy **bemeneti** adatai. Ha ezeket nem adjuk meg, akkor a feladatmegoldás elveszíti az értelmét.

A program további két utasítása – mint már említettük, a 60-as és 70-es sorszámú – a kiinduló adatokkal egy meghatározott műveletsort végez el, ez az öt szám összeadása és öttel való elosztása. De könnyen beláthatjuk, hogy feladattól függően itt más műveletek is állhatnának. Az adatokkal való műveletvégzést általános érvénnyel **feldolgozásnak** nevezzük. Az adatok feldolgozása során a kiinduló vagy bemeneti adatokból új adatok (értékek) készülnek. Ezek az adatok alkotják a feldolgozás célját. A bemutatott példában a feladat célja az átlag kiszámítása volt. Ezért van szükség az egész műveletsorozatra, hogy a végén ezeket az új adatokat megkapjuk. Ezek a feldolgozás **eredményei** vagy másképpen a megoldás **kimeneti** adatai.

Bármilyen más, számítógéppel megoldott feladatot vizsgálunk me azt látjuk, hogy a feladat a kívánt eredményadatok előállítása valam lyen kiinduló adatból valamilyen feldolgozási műveletsorral:

Kiinduló adat – Feldolgozás – Eredmény

Ez természetesen nemcsak a programozásra jellemző, hanem bármilye termelőfolyamatra is.

Számítógépes feladatmegoldásnál a kiindulás és az eredmény is adat valamilyen számérték, valamilyen szöveg, ezért is nevezzük az ilyen feladatmegoldást adatfeldolgozásnak. Ezt a három művelettípust jó meg kell jegyeznünk, mert a további feladatok megoldásának magvá fogják alkotni.

A feldolgozás műveletek sorozatából áll, amely a bemeneti adatokból elkészíti az eredmény-(kimeneti)adatokat. A feldolgozás műveleteit é sorrendjüket a feladatnak megfelelően kell összeállítani. Az egy adot feladat megoldását szolgáló műveletsorozatot **algoritmusnak** nevezzük A feldolgozási algoritmust a feladat alapján lehet meghatározni.

Egy feladat több különböző algoritmussal, hasonló feladatok ugyan azzal az algoritmussal is megoldhatók, vagyis nem kell minden feladat hoz új algoritmust kitalálni.

Az algoritmust a feladat megoldójának kell megfogalmaznia. Ez az jelenti, hogy a megoldónak kell meghatároznia a műveleteket és sor rendjüket úgy, hogy számítógéppel el lehessen végeztetni őket.

A számítógépi műveletekből összeállított algoritmus a **program.** Más képpen: a program a számítógép nyelvére lefordított algoritmus.

A program – mint láttuk – **utasításokból** épül fel. Amikor a gép a programot végrehajtja, akkor ezeket a műveleteket a kijelölt sorrend ben elvégzi, és hatásukra elkészül az eredmény. Ezekkel a megállapí tásokkal kiegészíthetjük az 1. részben felsorolt programjellemzőket.

A következőkben ismerjünk meg egy módszert, amely alkalmas arra hogy feladatokat számítógép segítségével megoldjunk. Figyeljük meg hogy a cél sohasem egy program megírása, hanem egy feladat megoldá sa, amelyben egy eszköz a program, így a számítógép is.

A FELADATMEGOLDÁS MÓDSZERE

Ha felmerül egy feladat (pl. a babkávé fogyasztását kell megjósolni az elkövetkező öt évre, vagy egy termékszerkezet rendelésállománya alap ján a gyártókapacitást és az alkatrész-szükségletet kell meghatározni vagy egy vállalat dolgozóinak bérét kell kiszámítani a teljesítmények alapján), amelyet számítógéppel akarunk megoldani, akkor a feladat megfogalmazása után a megoldásig a következő lépéseket kell elvégezni

– a feladat elemzése,
– a program tervezése,
– a modulok tervezése,
– a program kódolása (BASIC nyelvű utasítássorok megírása),
– a program kipróbálása (tesztelése), javítása,
– a program végrehajtása (futtatása),
– a kapott eredmények értékelése.

Már itt megemlítjük, hogy valamennyi lépésben írásos anyagok is készülnek a megoldásról, amelyek együttesen alkotják a program dokumentációját. Az első három lépéssel itt, a többivel a 4. részben foglalkozunk egy egyszerű feladat megoldásán keresztül.

A programozási feladatok megoldásához olyan módszert mutatunk be, amely az elemzési, tervezési és kódolási szakaszban is hasznosan alkalmazható. Az alapmódszer: a **moduláris programozás.**

A moduláris programozás lényege, hogy a feladatot részfeladatokra (funkciókra) kell felbontani, ezeket önállóan kell kódolni, majd az egyes modulokból össze kell állítani a kész programot. Először tisztázzuk a modul fogalmát.

A **modul** a programnak egy része, amely

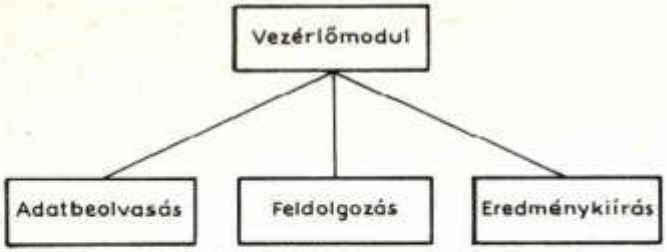
– az egész feladaton belül mint elkülönített részfeladat jelenik meg (logikai egység);
– funkciója, hatása elkülönítve is értelmes, ami az ellenőrzés szempontjából jól hasznosítható;
– mint részfeladat külön is meghatározható, és a főfeladattól elkülönítve is elkészíthető;
– egy vagy több, de 20–30-nál kevesebb utasításból áll.

A moduláris programozás leglényegesebb lépése a feladat modulokra bontása. Ez azt jelenti, hogy a feladatot funkciókra (részfeladatokra) kell bontani, és ezek fogják a modulokat alkotni.

A modulok a feladat megoldásában betöltött szerepük szerint három csoportba oszthatók:

– vezérlőmodul,
– adatmodul,
– eljárásmodul.

A **vezérlőmodul** irányítja egy vagy több modul, vagy az egész program végrehajtását (6. ábra). A vezérlőmodul a benne meghatározott sorrend szerint az 1. alárendelt modul végrehajtására tér át (ez a bal oldali szélső modul). Ha ezzel elkészült a gép, akkor ismét a vezérlőmodul veszi át a szerepet, és a következő (balról a második) modulra tér át a végrehajtás. Ez addig ismétlődik, míg valamennyi alárendelt modul végrehajtódik. A vezérlőmodulból való áttérést nevezzük **hívásnak.**



6. ábra. A vezérlőmodul működése

Példaként nézzünk meg egy olyan feladatot, amelyben egy vezérlőmodul van, és ennek három modult kell vezérelnie. Az első modulnak öt számértéket kell beolvasnia, a másodiknak ki kell számítania a beolvasott öt adat átlagát, a harmadiknak pedig ki kell írnia az átlagot (7. ábra). A modulok a végrehajtás sorrendjében helyezkednek el a vezérlőmodul alatt.

A vezérlőmodul az alábbiak szerint működik ebben a példában: Először hívja az *adatbeolvasást*. Ezután ismét a vezérlőmodul fog működni, ami abban áll, hogy hívja az *átlagszámítást*. Az *átlagszámítás* végrehajtása után újra a vezérlőmodul működik az *átlagkiírás* hívásával.

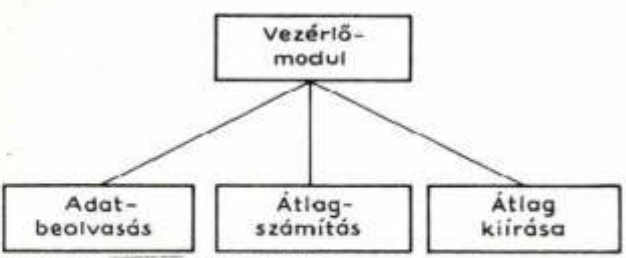
A vezérlőmodul tehát úgy működik, mint egy karmester: az alárendelt modulok valamelyikét végrehajtja. A későbbiekben ennél bonyolultabb vezérlési műveleteket is fogunk látni.

A hívás helyett vezérlésátadást is szoktak mondani. Például a vezérlőmodul átadja a vezérlést az *adatbeolvasásnak*, majd a végrehajtás után a vezérlés ismét a vezérlőmodulra kerül vissza.

Egyszerű szerkezetű programoknál a vezérlőmodul el is hagyható. Ilyenkor a modulok a megoldásnak megfelelő logikai sorrend szerint egymást hívják. A 6. ábrán bemutatott példa vezérlőmodul nélküli felépítése a 8. ábrán látható. A modulokat a végrehajtásnak megfelelő sorrendben láthatjuk, és a szemléletesség kedvéért a végrehajtás (és a vezérlés) felülről lefelé halad.

A modulszerkezeti ábrán azonban a vezérlőmodult érdemes meghagyni, mert így jobban látszódik a feladat logikai szerkezete.

Az **adatmodul** a program adatainak meghatározására szolgál. Lénye-



7. ábra. A vezérlőmodul működése

8. ábra. A végrehajtás sorrendje

gében passzív modul, a feladat érdemi megoldásához szükséges műveleteket nem tartalmaz, csupán adatokat. A BASIC nyelvben egyszerű az adatmeghatározás: szigorúan véve csupán tömbök (több adatból álló adathalmazok) terjedelmét határozza meg. Adatmodulnak tekinthetjük azonban az olyan modulokat is, amelyekben kiinduló adatokat kell beolvasni, vagy változóknak induló értéket kell adni. Adatmodul például a 7. ábrán az *adatbeolvasás* modul.

Adatmeghatározás két módon lehetséges a programban:

– minden műveletet végző modulhoz külön-külön,
– együtt az egész program számára.

Tanácsos a program elején az összes adatot egyszerre definiálni, mert így az adatmeghatározások egy helyen lesznek, de adott esetben ettől el is térhetünk.

Az **eljárásmodulok** valamilyen tényleges műveletet tartalmazó modulok. Ezek a modulok végzik el pl. a különböző feldolgozási műveleteket és a kiírásokat. Egy programon belül eljárásmodulból van a legtöbb, és ezek összessége hajtja végre a tulajdonképpeni feldolgozást. Eljárásmodul a 7. ábra *átlagszámítás* és *átlagkiírás* modulja.

Hogyan kapjuk meg egy feladat megoldásához szükséges modulokat? Nézzünk erre egy példát! Tegyük fel, hogy a feladat egy órabéres dolgozó havi bérének kiszámítása és az eredmény kiírása (bizonylat!). Vegyük sorra, hogy egy dolgozó bérének meghatározásához milyen feladatokat kell elvégezni. A feladatot első lépésben két részfeladatra bonthatjuk (9. ábra):

– meg kell határozni az összeadandó tételeket, amelyekből a bér összetevődik;
– meg kell határozni a levonásokat.

A részfeladatok azonban még nem oldhatók meg egyszerűen, mert

több különböző dolgot foglalnak magukban: a levonásokba beletartozik a nyugdíjjárulék levonása, az esetleges letiltások, CSÉB stb. Ugyanígy a bérösszetevők között van időbér, prémium, családi segély stb. Tehát egy következő lépésben a két részfeladatot bontsuk szét további részfeladatokra. Az összetevőkre ezt kapjuk (nem teljeskörűen):

- időbér kiszámítása
- prémium
- bérkiegészítés kiszámítása
- családi segély kiszámítása

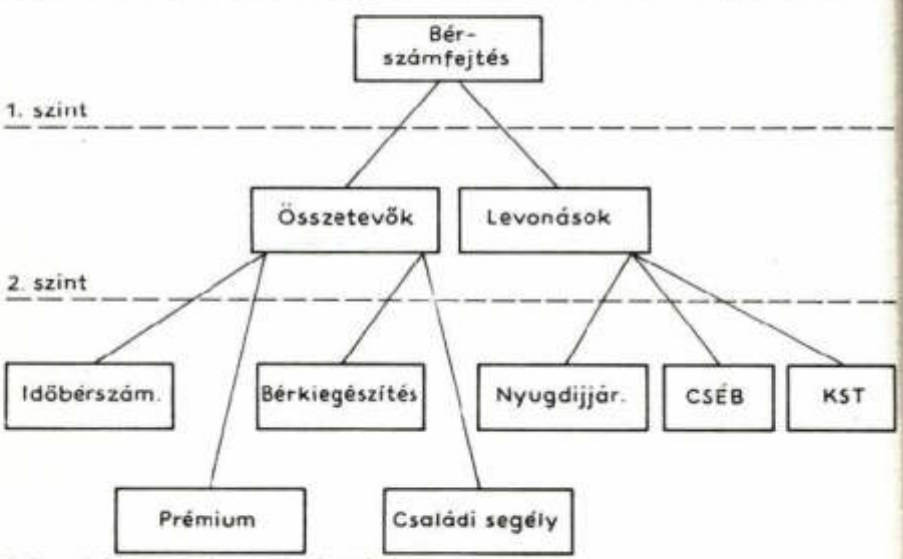
.
.
.

A levonásokat – szintén nem teljeskörűen – az alábbi részfeladatokra lehet bontani:

- nyugdíjjárulék kiszámítása
- CSÉB kiszámítása
- KST kiszámítása

.
.
.

A részfeladatokra bontást addig érdemes csinálni, amíg nem kapunk olyan egyszerű, a többitől jól elkülöníthető részfeladatokat, amelyeket



9. ábra. A feladat részfeladatokra bontása

moduloknak tekinthetünk. Ezek után meg kell határozni a modul műveleteit, és a feladatot megoldó programot ezekből a modulokból kell felépíteni.

A fent bemutatott eljárás módszerünk másik fontos eleme, ti. a **felülről lefelé** haladás a megoldás során. Tehát az első lépésben meghatározzuk a feladatot. Ez a fő, a legmagasabban álló feladat. A következő lépésben megvizsgáljuk, milyen részfeladatot kell elvégezni, hogy a feladat meg legyen oldva. Ekkor megkapjuk a feladat egészének alárendelt részfeladatokat. Ezeket a feladat alá rajzoljuk, hogy jelöljük a felbontást. A részfeladatokat tovább lehet bontani újabb részfeladatokra, amelyeket a részfeladatok alá lehet rajzolni. Lényegében a feladat egészét bontjuk fokozatosan részfeladatokra. Közben szakaszosan haladunk a feladattól lefelé a legapróbb részfeladatokig. Ezért nevezzük az eljárást felülről lefelé haladásnak. A felülről lefelé való lebontás (top-down) során a részfeladatok „szinteket" alkotnak. A lebontást a feladat összetettségétől függően különböző számú szinten végezzük addig, amíg nem kapunk egyszerű, jól elkülöníthető, önállónak tekinthető részfeladatokat, amelyek már egy modulnak foghatók fel.

A módszer előnye, hogy a lebontás eredményeként jól látszik a **részfeladatok** – modulok – közötti **összefüggés.** A feladat módszeres felbontása abban is segít, hogy **ne hagyjuk ki** a feladat megoldásához szükséges részfeladatokat.

A FELADATMEGOLDÁS FOLYAMATA

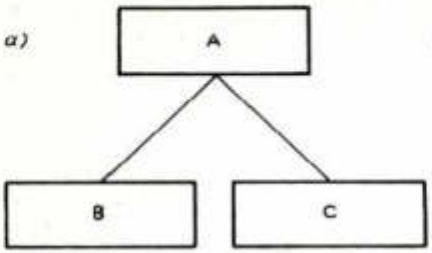
A FELADAT ELEMZÉSE

Az első lépésben a feladatot **fel kell bontani** részfeladatokra (elegendő egy szinten), hogy lehessen látni, milyen jellegű tevékenységeket kell elvégezni a feladat megoldása érdekében.

Elemezni kell a feladat számítógépben való **megoldhatóságát.** Ha a feladat egyébként megoldható (ismertek a kiinduló adatok, a megoldási algoritmus, kiszámítható az eredmény stb.), akkor még mindig felmerülhet olyan akadály, amely részben vagy egészben meghiúsíthatja ezt. Ilyen ok például, ha a kiinduló adatok nem kaphatók meg időben az eredményadatok elkészítéséhez, illetve a számítógép képességeit meghaladó bonyolultság vagy adatmennyiség.

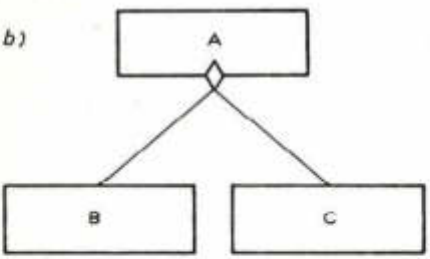
Itt kell meghatározni, hogy **milyen műveletek nem végezhetők el** a számítógéppel (pl. az eredményadatok elszállítása), vagy melyeket **nem érdemes számítógéppel** elvégezni (pl. túlságosan hosszú listák, amelyeket nem rendszeresen használnak). Az elemzés eredménye a megoldás **algoritmusának** pontos vagy nagyvonalú kialakítása is.

a)



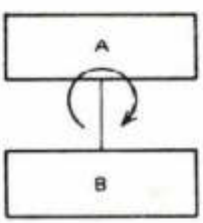
A hívja B-t, majd C-t

b)



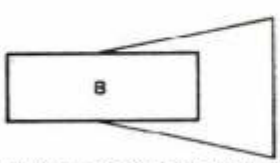
A egy feltételtől függően vagy B-t, vagy C-t hívja

c)



A ciklikusan hívja B-t

d)



B önmagát hívja (rekurzív hívás)

10. ábra. A modulok közötti kapcsolatok rajzjelei

A programtervezés a feladat elemzésének eredményeit használja fel. Fontos feladata a program szerkezetének további finomítása a feladatelemzés során kapott modulszerkezet tovább bontása révén. A meglevő funkciókat újabb részfunkciók megállapításával újabb szinteken bontjuk tovább. A bontást addig célszerű folytatni, amíg minden szinten lehetőleg egyszerű funkciót ellátó modulokat kapunk. Ezek a funkciók alkotják a **program moduljait.**

A bontás eredményeként kapott **modulok között kapcsolat** van. Ezeknek a kapcsolatoknak kell biztosítaniuk, hogy a modulok végrehajtása a helyes sorrendben történjen.

A programtervezés eredményeként létrehozott modulokat és a közöttük fennálló kapcsolatokat (struktúrát) egyezményes jelekkel ábrázoljuk (10. ábra).

a) Az A modul előbb a B, majd a C modult hívja. A szerkezet gyakorlati alkalmazására a 7. ábrán láthatunk példát.

b) Az A modul egy feltételtől függően vagy a B, vagy a C modult hívja. Az A modul feltételvizsgálati feladatot lát el a programon belül. A B és C modul végrehajtása a vizsgált feltételtől függ.

c) Az A modul ciklikusan hívja a B modult. Erre akkor van szükség, ha egy adatcsoport valamennyi elemén ugyanazt a műveletet kell elvégezni (pl. egy vállalat dolgozóinak bérét ugyanazzal az eljárással kell kiszámítani). Az A modul annyiszor hívja a B modult, ahányszor a műveleteket végre kell hajtani.

d) A B modul önmagát hívja (rekurzív hívás). Ez lényegében a *c)* pontban leírt szerkezet egyszerűsített változata, ha a vezérlő A modult elhagyjuk, és a ciklikus irányítást a B modulba építjük be.

A 12. ábrán bemutatott modulszerkezeti jelekkel szinte valamennyi feladat szerkezete ábrázolható.

A modulok **végrehajtásának sorrendjét** a vezérlőmodul, illetve ha több van, akkor a vezérlőmodulok megfelelő kialakításával kell biztosítani. A funkciókat megvalósító modulokat — mint már korábban utaltunk rá — balról jobbra haladva kell berajzolni a modulszerkezeti ábrába, mert a végrehajtás is majd balról jobbra halad.

A programtervezéshez tartozik annak az eldöntése is, hogy a program egyetlen program legyen, vagy több önálló programból álljon. Felmerül a kérdés, hogy mikor melyik megoldás a célszerűbb. Bonyolult, nagy feladatnál érdemes több programra felosztani az egészet.

Hasonló vagy azonos funkciók ellátására érdemes típusprogramot tervezni. Ezt csak egyszer kell elkészíteni, és ezek után több feladat

megoldásában is alkalmazható. Sok fáradságot és időt lehet megtakarítani vele. A többször felhasználható modulprogramok **modulkönyvtárat** alkotnak.

Összefoglalva megállapíthatjuk, hogy a program modulokra bontásának számos előnye van:

- a feladatot könnyen kezelhető, zárt egységekre bonthatjuk fel;
- a program(ok) terve jól áttekinthető;
- a feladat változásánál a program könnyen módosítható, mert ez rendszerint csak modulcserét vagy modulmódosítást jelent;
- a program elkészítéséhez szabványos modulokat is használhatunk, s ez időmegtakarítást eredményez.

A MODULOK TERVEZÉSE

Az egész program megtervezése után a program „építőköveinek" – a moduloknak – a tervezése következik. A tervezés során meghatároztuk a feladat megoldásához szükséges modulokat. Láttuk, hogy a modul egy részfeladatot old meg. Az egész feladat megoldása szempontjából azt is meg kellett határozni, hogy egy modul hogyan kapcsolódjék a többihez, mikor kell feladatát megvalósítania, honnan kapjon adatokat stb. Ez a modul külső specifikációja. Továbbá le kell írni azokat a belső folyamatokat, amelyek végrehajtásával a modul teljesíti funkcióját.

A **külső specifikációhoz** tartozik a **modul neve,** amellyel a programozó az adott modult azonosítja. Ide tartozik a modul típusának (vezérlő-, adat-, eljárás-) meghatározása is. Végül azt is definiálni kell, hogy a modul milyen módon **kapcsolódik a környezetéhez,** vagyis hogyan lehet **hívni,** és mi történik a **befejezéskor.**

Általánosan követendő szabályként javasoljuk az olyan modulok tervezését, amelyek fő jellemzője, hogy **egyetlen** bejáratuk és **egyetlen kijáratuk** van. Ez azt jelenti, hogy a modul végrehajtása mindenféleképpen ugyanazon a kezdő utasításssoron indul, és ugyanazon a záró utasításssoron fejeződik be. Ez a modulok zártságát, áttekinthetőségét fokozza, és nem lebecsülendő az sem, hogy ilyenkor a módosítás könnyebben elvégezhető.

A modul **belső tervezése** három fő lépésből áll:

a) A kiinduló adatok leírása
Részletesen le kell írni minden tárolt vagy az előző modulok által készített adatot, amelyet a modul a végrehajtáskor felhasználhat. A meghatározás tartalmazza a kiinduló adatok tárolási módját (külső tárolón, tárban stb.), jelölését, típusát (szöveges, számérték vagy mindkettő) és értéktartományát.

b) Az eredményadatok leírása
A modul funkciója az eredményadatok előállításával valósul meg. A modul tervezésekor ezeket is részletesen meg kell határozni (pl. a kiírás formája, a tárolt adatok felépítése, jelölése stb.).

c) A feldolgozás folyamatának meghatározása
Meg kell határozni azokat az adatokon végrehajtott műveleteket és végrehajtási sorrendjüket, amelyeknek eredményeként megjelennek a szükséges kimeneti adatok.

A feldolgozási folyamatot folyamatábrával tesszük képszerűbbé és könnyebben áttekinthetővé. Sok olyan művelet van, amely rendszerint több feldolgozási folyamatban is előfordul. Ezeket célszerű egységes folyamatábrajellel jelölni. Így alakultak ki a folyamatábra-szimbólumok, amelyek egy-egy művelettípust vagy eszközt jelölnek.

A következőkben áttekintjük a gyakrabban használt folyamatábra-szimbólumokat:

Tárolóeszközök

Mágnesszalag-állomány

Mágneslemez-állomány

Perifériák és a velük kapcsolatos műveletek

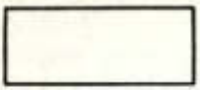
Sornyomtató, bemeneti bizonylat, kiírás (lista)

Képernyős terminál

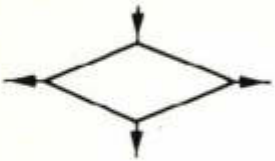
Billentyűzet, kézi adatbevitel

Műveletek

Bemeneti vagy kimeneti művelet

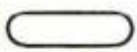


Tetszőleges műveleti jelkép. A művelet meghatározását a téglalapba kell beírni



Döntés. A döntés valamilyen vizsgálaton alapszik. A vizsgálat eredményeként a program végrehajtása 2—3 irányban haladhat tovább

Külön meghatározott művelet, amely önmaga is egy több lépéses folyamatábra lehet. Akkor használjuk, ha a folyamatábrában egy, már máshol meghatározott műveletsorozatot nem akarunk részletezni



Határoló jelkép. A folyamat kezdetén és végén alkalmazzuk

Rajztechnikai jelek

Folyamatvonal a folyamat-végrehajtás irányával

Folyamatvonalak keresztezése

Magyarázó jelkép, amely valamilyen folyamatábrajelhez kapcsolódik. Akkor használjuk, ha a kívánt szöveg nem fér el a szimbólumban. A kiegészítő szöveget a magyarázó jelkép jobb oldalára írjuk

A folyamatábra-jelölések egyszerűek és könnyen érthetők. Rajzolásuk megkönnyítésére a kereskedelemben folyamatábra-sablonok kaphatók.

A modul funkcióját megvalósító folyamat sokféleképpen mehet végbe. A programozók körében elfogadottá vált az az elv, hogy az algoritmusokban három algoritmustípust elegendő alkalmazni. Ezeket a folyamat-„építőköveket" fogjuk mi is felhasználni:

SZEKVENCIA

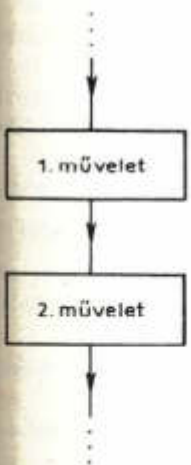
Két vagy több utasításból álló utasítássorozat (lásd 11. ábra). Egy szekvencia tetszőleges típusú és számú műveletet tartalmazhat.

FELTÉTELES ELÁGAZÁSOK

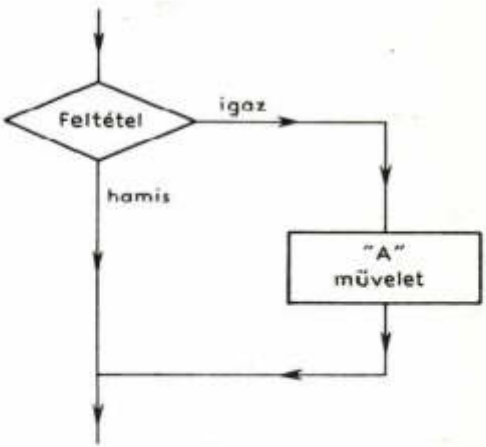
Valamilyen feltételtől függően

– egy műveletet vagy végrehajtunk, vagy nem (12. ábra);
– két művelet közül csak az egyiket hajtjuk végre (13. ábra);
– kettőnél több művelet közül valamelyiket végrehajtjuk (14. ábra).

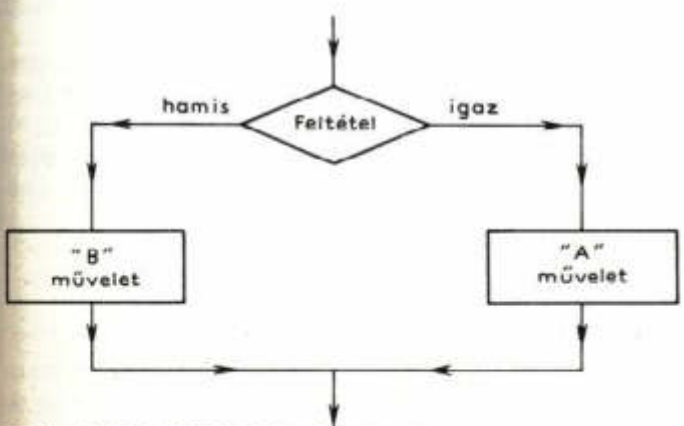
Példaként említjük a szorzatkitaláló feladatot.



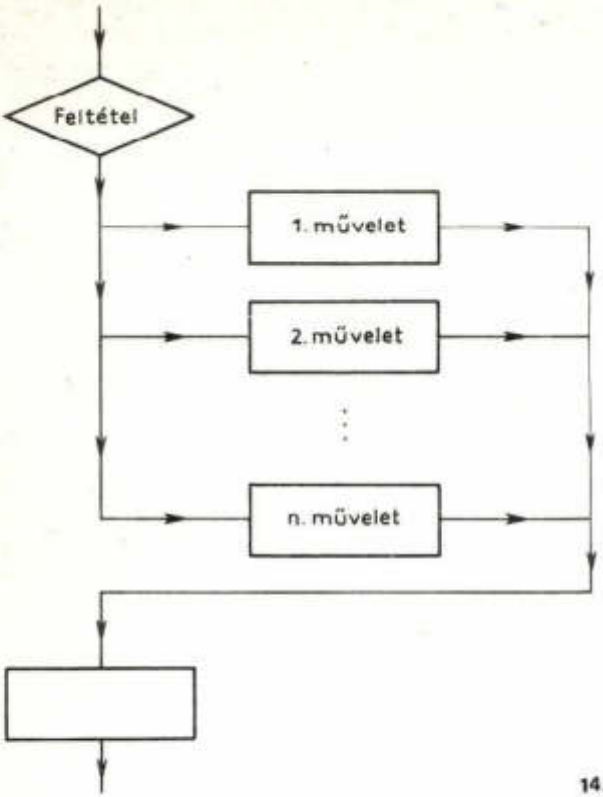
11. ábra. Szekvenciális műveletek



12. ábra. Az IF THEN típusú szerkezet



13. ábra. Az IF THEN ELSE típusú szerkezet



14. ábra. A CASE típusú szerkezet

Megfogalmazhatunk egy olyan részfeladatot, hogy a gép által kiszámított helyes szorzatot csak akkor írja ki a gép a képernyőre, ha a játékos helytelen értéket gépelt be. Ez az első típusú folyamat. Ilyenkor a kiírási műveletet csak akkor kell végrehajtani, ha a számított és a begépelt összegek eltérőek, egyébként nem.

Megfogalmazhatjuk olyan formában is a részfeladatot, hogy ha a játékos eltalálja az eredményt, akkor elismerésként a „GRATULALOK" üzenetet, ha nem találta el, a „SAJNOS, EZ ROSSZ" üzenetet kell kiírni. Ez a második típusú folyamat alkalmazását teszi szükségessé, mivel valamelyik üzenetet mindig ki kell írni. A több műveletből való kiválasztásra a 6. részben fogunk látni egy feladatot.

CIKLUS

Ha például egy raktárban levő anyagok értékét kell kiszámítani, akkor minden anyag esetében az egységár és a mennyiség szorzatát kell kiszámolni és az eredményt összegezni. Ilyenkor a művelet elvégzését nem

érdemes annyiszor leírni, ahányszor végre kell hajtani, hanem a műveletet ciklikusan kell megismételni az összes anyag adataival. A ciklikus műveletek végét valamilyen feltétel határozza meg (pl. ha minden anyag értékét kiszámítottuk).

A ciklikus műveletek a végfeltétel-vizsgálat helyétől függően két formában hajthatók végre:

– elöl tesztelő,
– hátul tesztelő.

Az elöl tesztelő típusú szerkezetben a ciklikus művelet elvégzése **előtt** kell megvizsgálni azt a feltételt, amely alapján el lehet dönteni, hogy szükség van-e még a művelet elvégzésére (15. ábra).

A hátul tesztelő típusú szerkezetben előbb a műveletet kell végrehajtani, utána kell vizsgálni a további ismétlés feltételét (16. ábra).

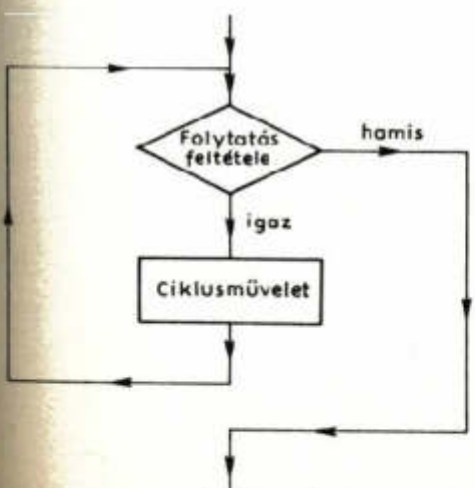
A „tiszta" esetek mellett vannak olyanok is, amikor az abbahagyás (vagy folytatás) feltételét sem elöl, sem hátul, hanem valahol középen ellenőrizzük.

Most pedig nézzük meg a feladatmegoldás első három lépését egy konkrét feladaton!

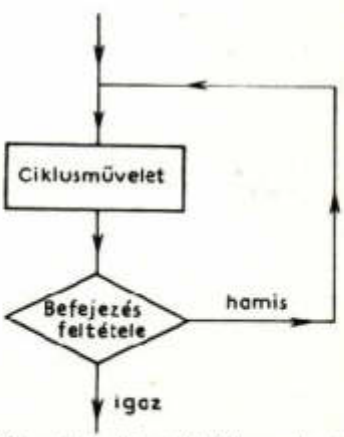
1. feladat

Adva van a 17. ábrán látható ellenállásokból álló hálózat. Meg kell határozni az A és B pont közötti eredő ellenállást. Az ellenállások értékei:

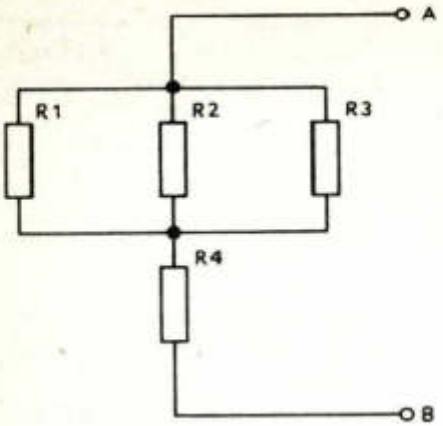
R1 = 1000 ohm
R2 = 2000 ohm
R3 = 500 ohm
R4 = 200 ohm



15. ábra. Az elöl tesztelő ciklusszerkezet



16. ábra. A hátul tesztelő ciklusszerkezet



17. ábra. Az I. feladat

A program az alábbi formában írja ki az eredő ellenállás értékét:

AZ EREDO ELLENALLAS (OHM): X

A feladat elemzése

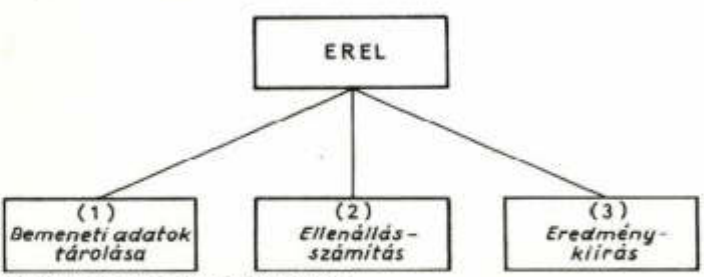
Nézzük meg, mit eredményez a feladat elemzése! A feladat az eredő ellenállás meghatározása ismert adatokból. A feladat számítógéppel megoldható. A feladatot megoldó programnak az alábbi főbb funkciókat kell megvalósítania:

- a bemeneti adatok (az ellenállásértékek) tárolása,
- az eredő ellenállás kiszámítása,
- az eredő ellenállás értékének kiírása.

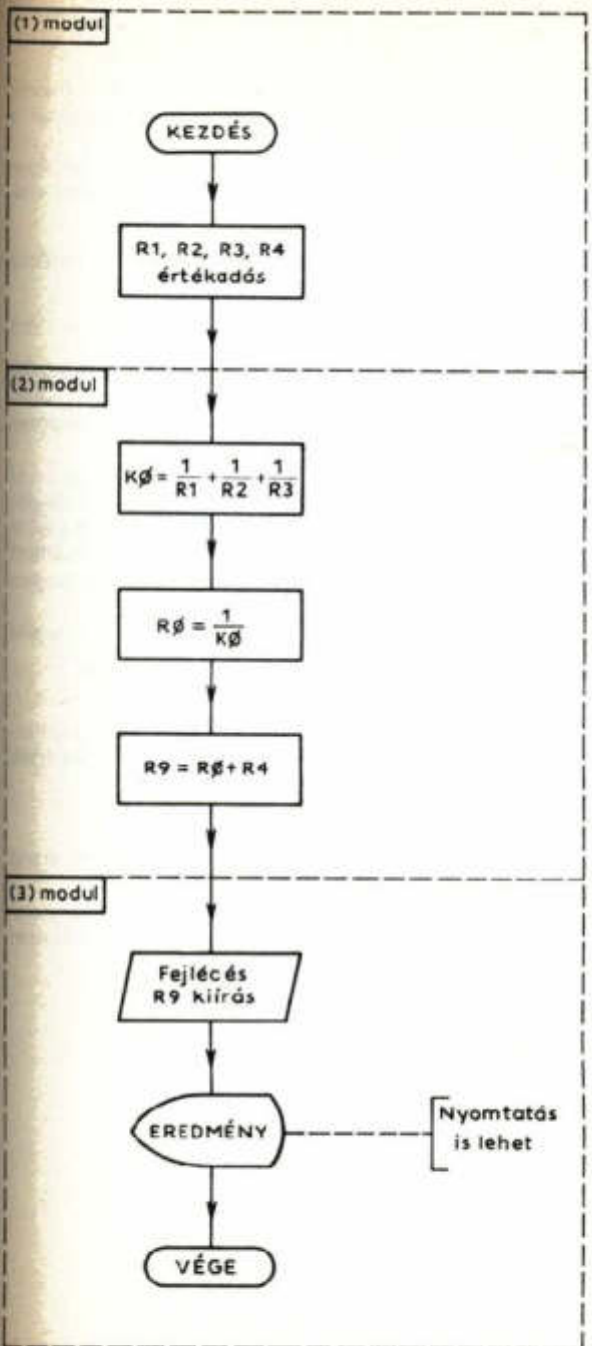
A feladatban felmerült eredő ellenállást az ismert összefüggés alapján lehet kiszámítani:

$$R9 = \frac{1}{\frac{1}{R1} + \frac{1}{R2} + \frac{1}{R3}} + R4$$

Végül elkészítjük a feladat modulokra bontásának első közelítését (18. ábra).



18. ábra. A feladat modulszerkezete



19. ábra. A modulok folyamata

A program tervezése

Most pedig végezzük el a programtervezés folyamatát a feladaton!

A probléma egyszerűsége miatt nincs szükség a modulszerkezet további finomítására. Így az elemzéskor kapott modulszerkezetet a programtervezés alapjának is tekinthetjük.

Látható, hogy a modulokat egyszerű szekvencia szerint [(1), (2), (3)] kell végrehajtani, ezért a feladat egyszerűsítése végett a vezérlőmodul alkalmazásától eltekintünk (18. ábra).

A modulok tervezése

(1) *A bemeneti adatok tárolása*

A modul **adat** típusú, mert egyetlen funkciója a négy ellenállásérték tárolása. Önálló bemeneti adatai nincsenek, a kimeneti adatokat az ellenállások értékei adják. A modul algoritmusa az R1, R2, R3, R4 változó értékeinek megadása (19. ábra).

(2) *Ellenállás-számítás*

A modul **eljárás** típusú, mert feldolgozást végez. Bemenete a négy ellenállásérték, kimeneti adata az R9 eredő ellenállás. Az alkalmazott algoritmus a soros és párhuzamos ellenállásokból álló hálózat eredőellenállás-számítására vonatkozó ismert kifejezésre épül:

$$R9 = \frac{1}{\frac{1}{R1} + \frac{1}{R2} + \frac{1}{R3}} + R4$$

Először határozzuk meg a tört nevezőjében álló kifejezést, jelöljük K0-val, majd ennek vegyük a reciprokát. Ezzel a jobb oldal első tagját kiszámítottuk. Az R4 érték hozzáadásával megkapjuk R9-et. Az algoritmust a 19. ábra mutatja.

(3) *Eredménykiírás*

Ez is **eljárás** típusú modul. Bemeneti adata a (2) modulból származó R9 eredő ellenállás értéke. Az eredményadat ugyanez, de kiírva.

A specifikáció szerint a modulnak egy megadott szöveg után ki kell írnia az eredő ellenállás értékét. Ezt egy lépésben el lehet végezni (lásd 19. ábra). A kiírás után a program befejeződik.

Ellenőrző kérdések és feladatok

1. Egy számítógépes feladat megoldása milyen főbb lépésekből áll?

2. Mi a program?

3. Milyen lépésekből áll egy feladat megoldása?

4. Munkahelyén vagy otthon keressen egy megoldandó feladatot, és végezze el a feladatmegoldás első három lépését!
   Milyen feltételek teljesülése mellett oldható meg a feladat számítógéppel?
   Mennyivel nyújt többet a számítógépes megoldás, mint a jelenlegi ún. „kézi" megoldás?

5. Milyen rajzjeleket használt a modulok műveleteinek ábrázolásához?

4. A PROGRAMOZÁS MÓDSZERE II.

A programkészítés módszere és lépései,
az első feladat megoldásának befejező része

Ebben a részben folytatjuk a programkészítés lépéseinek részletes bemutatását és a lépések illusztrálásaként befejezzük az 1. feladatot.

A FELADATMEGOLDÁS FOLYAMATA

(FOLYTATÁS A 3. RÉSZBŐL)

A KÓDOLÁS

A kódolás a folyamatábrában rögzített műveletek megfogalmazása az adott programnyelven, jelen esetben BASIC nyelven. A kódolást modulonként külön-külön, a folyamatábra alapján kell elvégezni. A kódolás eredményei az utasítássorok vagy röviden utasítások. A folyamatábra lefordításánál figyelembe kell venni az adott nyelv lehetőségeit, sajátosságait. Ezért nem csupán szóról szóra kell fordítani, hanem a programnyelv szerkezeti sajátosságaira is ügyelni kell.

Egy utasítás egy mondatnak felel meg. Az utasítások a BASIC nyelvben „mondattanilag" két részből állnak: a **kulcsszóból**, amely a mondat állítmányának felel meg, és az **utasítás tárgyából**, amely azt mutatja, hogy mire vonatkozik az utasítás.

A végrehajtás sorrendjének meghatározásához az utasítások **sorszámot** is kapnak. A sorrend kijelölésén kívül ez az utasítás azonosítója is, ezzel hivatkozhatunk az utasításra.

Egy BASIC utasítás tehát három részből áll:

sorszám, utasításkulcsszó, az utasítás tárgya.

Célszerű, de nem kötelező az egyes részek között **legalább egy szóközt** hagyni, de a sorszámon és az utasításkulcsszón belül **nem lehet szóközt** írni. Az elmondottak értelmében formailag helyesek a következő utasítások:

```
100␣LET␣A=8
```

vagy

```
100LETA=8
```

(A␣jelet egy szóköz jelölésére használjuk a kritikus esetekben, amikor a szóköznek nagy jelentősége van.)

Formailag hibásak az alábbi sorok:

100␣L␣E␣T␣A=8

vagy

1␣0␣0 LET␣A=8

A BASIC utasításokat sorokba írjuk. Általában **egy sorba egy utasítást** írunk. Könyvünk további részében – néhány esetet kivéve – ezt a szabályt fogjuk alkalmazni.

Felhívjuk a figyelmet, hogy egy sor nem azonos egy, a képernyőn megjelenített sorral (ez a képernyősor). Egy sor addig tart, amíg le nem nyomjuk a sorzáró karaktert. Egy utasítássor tehát több sort is elfoglalhat a képernyőn. Természetesen a sorhossznak van felső határa. Ez a különböző gépeknél a következő:

Commodore: 80 jel (2 teljes képernyősor),
- HT: 240 jel,
- ▲ PRIMO: 210 jel (5 teljes képernyősor),
- Sinclair-gépeknél a tár szab határt.

Egy sorba legfeljebb ennyi jelet írhatunk. Ha egy sor betelik az utasítássoron belül, akkor a gép automatikusan sort emel, és a helyőr a sor elejére ugrik.

Megjegyezzük, hogy egy utasítássorba több utasítást is írhatunk, de egy utasítást sohasem szabad két utasítássorba írni! Ha egy sorba több utasítást írunk, akkor ezeket kettősponttal kell elválasztani. Ha egy sorban több utasítás van, akkor ezt **többszörös utasításnak** nevezzük. A program áttekinthetősége kedvéért a többszörös utasítások használatát kerülni kell, különösen kezdőknek.

Most pedig vizsgáljuk meg az utasítások részeit!

A SORSZÁM

Mint már láttuk, a program utasítássorait sorszámmal kell ellátni. A program futásakor az egyes utasítások a legalacsonyabb sorszámtól a sorszámok növekvő sorrendjében hajtódnak végre. Egy sorban csak a sor elején lehet egy sorszám. Többszörös utasításoknál a sorban levő első utasítás kap sorszámot, a többi nem.

Említettük már, hogy a sorszám az utasítás azonosítója, ezzel hivatkozhatunk rá. Ha egy sorban például valamilyen szintaktikai (formai) hiba van, akkor a gép a hibaüzenetben a hibás sor számát is kiírja. Így a programozó rögtön tudja, hol keresse a hibát.

A sorszámot a program készítője határozza meg. A program első utasítása kapja a legalacsonyabb sorszámot, a többi utasítást a végrehajtás

sorrendjének megfelelően növekvő sorszámmal kell ellátni. A sorszám téves begépelése az utasítás végrehajtási sorrendjét megváltoztathatja, és a program hibásan fog működni.

Ha két utasítást ugyanazzal a sorszámmal gépelünk be, akkor a másodszorra beírt utasítás az elsőt törli, mivel a gép azt az illető utasítás javításának tekinti.

A sorszám szerinti végrehajtást néhány utasítás megváltoztathatja. Ezeket – a többi utasítással együtt – a következőkben részletesen bemutatjuk.

Bár az utasítások végrehajtási sorrendjét nem a begépelésük sorrendje határozza meg, hanem a sorszámuk, mégis célszerű az utasításokat a végrehajtás sorrendjében begépelni.

Az utasítások sorszámozása a következő lehet:

- Commodore: 1-től 32767-ig,
- ■ HT és ▲ PRIMO: 1-től 65529-ig,
- • Sinclair: 1-től 9999-ig.

Ez a szám elég nagy ahhoz, hogy ne okozzon gondot.

Ajánlatos az utasításokat ötösével vagy tízesével sorszámozni. Ha ugyanis a helyes működés végett az utasítások közé újabb utasítást kell beszúrni, akkor ez az utasítás (vagy utasítások) megkaphatja a fel nem használt sorszámot. Ha egyesével számozzuk az utasításokat, akkor egy kényszerű beszúrásnál át kell sorszámozni az egész programot.

■ A HT és ▲ a PRIMO gépeknél automatikus sorszámozó lehetőség is van, amely programíráskor maga ad sorszámot az utasításoknak. Ezt az

AUTO *sorszám, növekmény*

paranccsal lehet működtetni. A *sorszám*-mal kezdő sorszámot lehet megadni, a *növekmény* a két egymás után következő utasítás sorszámának különbsége. Ha a növekményt nem adjuk meg, akkor a gép 10-nek veszi. Ha a sorszámot nem adjuk meg, akkor a sorszámozás a 10-nél fog kezdődni.

A parancs kiadása után hozzákezdhetünk a program begépeléséhez. A ■ NEW LINE, illetve a ▲ RETURN lenyomása után a következő sor elején megjelenik a következő utasítás sorszáma. Így ezt már nem kell begépelni. Ha beírtuk a programot, ■ a HT esetében a BREAK, ▲ a PRIMO esetében a BRK gombot kell lenyomni, és ezzel az automatikus sorszámozás befejeződik, a gép visszakerül a szokásos üzemmódba.

AZ UTASÍTÁSKULCSSZÓ

Az utasításkulcsszó műveletet határoz meg. Az utasításkulcsszavak meghatározott szavak, a függelékben megtalálhatók. Az utasításkulcsszavakat csak a megadott formában – szintaktika szerint – lehet használni. Mindemellett minden utasításnak a Commodore-on és

- ▲ a PRIMO-n van úgynevezett rövidített – két karakterből álló – formája is.
- ● A Sinclair-gépeknél az utasítások billentyűkön vannak, így egyetlen gomb lenyomásával egy utasítás beírható, elgépelni nem lehet.

Itt most egyetlen utasítást mutatunk be, amely műveletet nem hajt végre, de dokumentációs szempontból fontos. Ez a megjegyzést jelölő REM utasításkulcsszó:

```
30 REM ITT KEZDODIK A SZAMOLAS
```

Ez az utasítás azt jelenti, hogy a számítógépnek semmilyen műveletet sem kell elvégeznie, áttérhet a következő utasítás végrehajtására. Az utasítás tárgyában levő szöveget a programozó saját magának beírja, hogy a program nevezetes pontjait megjelölje, dokumentálási céllal.

AZ UTASÍTÁS TÁRGYA

Az utasítás tárgya legtöbbször valamilyen adat, amellyel valamit elvégez a gép, de lehet utasítássorszám is. Van olyan típusú utasítás is, amelyhez nem tartozik tárgy. Az utasítás tárgyára majd az egyes utasítások bemutatásánál láthatunk példát.

A KÓDOLÁS MÓDSZERE

A programkódot tanácsos először kódlapra (vagy más papírra) megírni, és csak ellenőrzés után begépelni. Nem javasoljuk a rögtönzött kódolást a terminálon a folyamatábra alapján, mert az még rövid, egyszerű moduloknál is nagyon sok hibát okozhat. A kódolásnál érdemes néhány szót szólni a dokumentációról.

A **programdokumentáció** a programról rendelkezésre álló írásos információ. A dokumentációnak írásban kell rögzítenie a program célját, a feladat megoldási és az adatok tárolási módját, a bemeneti, az eredményadatokat, és általában minden olyan adatot, amely lehetővé teszi, hogy más szakember is megismerje, megértse programunkat. Lényeges, hogy a dokumentáció segítse a program készítőjét vagy javítóját, hogy a programot – ha módosítani kell – 1–2 év után is megértse.

Az alkalmazott módszernek megfelelően a következő dokumentumok készülnek a programmal párhuzamosan:

- feladatleírás,
- a feladat elemzése és a szerkezet,
- a program terve,
- az egyes modulok terve,
- az adatok tárolási módja (állományleírások, tömbök),
- a programkód listája,
- a felhasználói utasítás (kézikönyv).

Valamennyi programdokumentummal kapcsolatban általános követelmény, hogy az érdekeltek hozzáférhessenek. Ezért a felhasználók számának és igényeinek megfelelő példányszámban és minőségben kell a dokumentumokat elkészíteni. A dokumentáció legyen **tömör, világos szerkezetű.**

A dokumentumok közül kettőt emelünk itt ki, a többit a feladatok megoldása során ismerjük meg. A **programkód listája** a program egyes sorainak a listája abban a formában, ahogy a program tárolódik. A programlista önmagában nem beszédes dokumentum. Olvashatóságát növeli, ha a REM utasítással különböző magyarázatokat, közléseket helyezünk el benne. REM utasításokkal a program elejére beírhatjuk a program nevét. Egy ilyen megoldás:

```
10 REM**************
20 REM*              *
30 REM*    TREND     *
40 REM*              *
50 REM**************
```

Sormintának a csillag helyett bármilyen más, BASIC által ismert karaktert alkalmazhatunk. A REM utasítással üzeneteket helyezhetünk el a program kritikus pontjain vagy az egyes programmodulok elején:

```
200 REM***** OSSZEG KIIRAS *****
```

Változók jelentését is megadhatjuk:

```
30 REM IL: IDOLEPTEK
```

A BASIC-ben a programot a

LIST

paranccsal listázhatjuk ki a képernyőre, vagy a program egy részét, ha megadjuk a rész kezdő- *(n)* és a zárósorának *(m)* sorszámát a LIST parancsban:

LIST *n–m*

Ha csak a kezdő sorszámot adjuk meg, akkor a gép a megadott sorszámtól kezdve kiírja az egész programot:

LIST *n*–

Ha viszont csak a záró sorszámot adjuk meg, akkor a gép a program kezdőrészét írja ki az *m* sorszámig:

LIST –*m*

Végül egyetlen sort is ki lehet listázni, ha megadjuk a sorszámot:

LIST *n*

▲ A PRIMO gépen a RETURN lenyomása után csak az első kiírandó sor jelenik meg. A listázás csak akkor folytatódik, ha a RETURN gombot lenyomva tartjuk és csak addig listáz a gép, amíg az ujjunk a RETURN gombon van.

● A Sinclair-gépek a LIST parancs hatására egy képernyőnyi részt írnak ki. Ekkor megjelenik a

scroll?

kérdés. Ha a listázást folytatni akarjuk, akkor az Y-t, ha nem, akkor az N betűt kell leütni. A LIST *n* parancs hatására a gép legalább az *n*. sorszámú sortól kezdve egy képernyőnyi listát készít.

A képernyőn megjelenő lista nem marad meg, és így mint dokumentum nem használható. A lista akkor látja el legjobban feladatát, ha a gép maradandó formában tudja elkészíteni. Erre a legjobb megoldás, ha a lista nyomtatva készül. Ha a számítógéphez nyomtató is tartozik, akkor a listát kinyomtathatjuk. Commodore-géppel a program listáját az alábbi többszörös paranccsal lehet elkészíteni:

```
OPEN3,4:CMD3:LIST
```

ahol

OPEN3	– megnyitja a nyomtató és a gép közötti utat (csatornát) 3 hivatkozási számmal,
4	– a nyomtató száma,
CMD3	– a képernyőről átirányítja a kiírást a 3 hivatkozási számmal megnyitott eszközre,
LIST	– a listázás parancsszava.

A parancs hatására a gép kiírja a tárban levő program listáját a nyomtatóra. A lista kiírása után a kiírást vissza kell állítani a képernyőre, ezért a gép és a nyomtató közötti utat le kell zárni a következő paranccsal:

PRINT#3:CLOSE3

PRINT#3 – egy üres kiírási parancs a sornyomtató csatornájának lezárása előtt,
CLOSE3 – lezárja a csatornát.

A HT gép, ▲ a PRIMO és ● a Sinclair-gépek esetében a tárban levő programot az

LLIST

parancs kiadásával lehet nyomtatóra kíiratni. A parancs ugyanúgy használható, mint a LIST utasítás. (Megjegyezzük, hogy a HT géphez csak egy külön csatolóegységen keresztül lehet nyomtatót csatlakoztatni.)

felhasználói kézikönyv a program felhasználójának készül. Megutatja, hogy a programot hogyan lehet használni az adott feladat megdására. Tartalmaznia kell a végrehajtáshoz szükséges bemeneti adatoit és bevitelük módját, a futtatás (végrehajtás) parancsait, az eredményatok kezelését, a program üzemszerű szolgáltatásait. Meg kell adnia a ülönböző hibáknál kiadott hibaüzenetek értelmezését és a hiba elhárísához szükséges tevékenységeket.

PROGRAM KIPRÓBÁLÁSA

tesztelés a program elkészülése utáni munkafázis, célja, hogy megállaítsa, helyesen működik-e a program. Ha kiderül, hogy nem, akkor a rogram hibás. Ekkor a hibát meg kell keresni, ki kell javítani, és a rogramot újra kell futtatni.

A tesztelés általánosan elfogadott elve, hogy

- előbb modulonként, majd
- az egész programra és
- a megtervezett próbaadatokkal

ell elvégezni. A modulonkénti tesztelés azért kedvező, mert egy vizonylag kicsi, jól áttekinthető részt fog át. Valamennyi modul ellenrzése után az egész programot is tesztelni kell. A megtervezett próbadatok jelentősége abban áll, hogy minél kevesebb próbaadattal minél eljesebb ellenőrzést lehessen elvégezni.

A tesztelés lényege, hogy különböző bemeneti adatokkal próbáljuk i a programot (illetve a modulokat), és közben figyeljük, hogy a specikáció szerint működik-e. A hibakeresés nehezen formalizálható, sok ntuíciót követelő tevékenység. Még könyörtelenebb logikát kíván a rogram készítőjétől, mint a program tervezése vagy írása. Egy program lkészítése során a hibakeresés szokta okozni a legnehezebb perceket, rákat. Gyakran maga a program készítője is „ciklusba esik", és nem

dolni, hogy mi a jelenség oka, mi idézi elő, és hogyan javítható ki. Bármilyen hibakeresés alapja, hogy a felhasználó a programlista alapján tájékozódjék, mi lehet a hiba.

Az első futások eredménye a szintaktikai (formai) hibák kimutatása lesz. A szintaktikai hibák kijavítása után olyan tesztadatokat kell megadni, amelyek lehetőleg minden utasítást kipróbálnak, eközben figyelni kell a működés helyességét. Ebbe a fázisba tartoznak a szélsőséges adatok is, amelyek a legszigorúbb követelményeket jelentik (pl. mit csinál a modul akkor, ha a felhasználó tévesen negatív számot ad meg egy dolgozó bérének). Ha a modul nem a specifikáció szerint működik, akkor a hibakeresésre térünk át.

A hibakeresés egyik eszköze a **program listája.** Hiba esetén a programlista alapján ellenőrizni kell, hogy melyik utasítás vagy utasításcsoport idézi elő a hibát. Lehet, hogy csak egy elgépelés a hiba oka, de lehet az is, hogy valamit nem vettünk figyelembe, valamit kifelejtettünk stb. Ezt kell megkeresni a lista alapján. Ha nem sikerül, akkor más eszközhöz is folyamodhatunk.

Egy másik eszköz a *STOP utasítás* és a *közvetlen mód* együttes használata, amit az 1. részben ismertünk meg. A STOP utasítás a BASIC programokban használható utasítás. Hatására a program leáll, és a

BREAK IN LINE 500

üzenetet,

- a Sinclair-gépeknél a 9 STOP statement üzenetet írja ki.

A LINE szó után álló szám a STOP utasítás sorszáma, ahol leállt a program.

Ekkor **közvetlen módban** ki lehet íratni az egyes változók aktuális értékét. Legyen például az alábbi program:

```
10 LET A=5
20 STOP
30 LET B=4
```

A program végrehajtása után a

BREAK IN LINE 20

üzenet jelenik meg, és a program leáll.

Ekkor közvetlen módban ki lehet íratni az A változó értékét:

```
PRINT A
5
```

Ezzel a módszerrel meg tudjuk vizsgálni, hogy a program egy adott pontján a változóknak mi az értéke. Ebből nagyon sok következtetést le lehet vonni. Kiderülhet, hogy valamilyen műveletet kihagytunk, valamilyen műveleti jelet rosszul írtunk be stb.

tud megszabadulni attól a gondolattól, hogy a program biztosan jó, valami rejtélyes dolog okozza a hibát. Ilyenkor logikusan végig kell gon-

A STOP utasítással leállított program a

CONT

parancs kiadásával folytatható a leállás helyétől. Egy programot több helyen is meg lehet szakítani szükség esetén, és mindenütt folytatható a CONT paranccsal. A közvetlen mód fenti alkalmazása a program végén is lehetséges.

A felfedett hiba jellegétől függően javítható ki. Ha elemzési hiányosság okozta, akkor vissza kell térni az elemzésre, és a programot újra kell tervezni, illetve a tervet módosítani. Ugyanez érvényes a modulokra és a kódra is. Ha a hibát a tervezésben követtük el, akkor csak ide kell visszatérni. A legegyszerűbb esetben pedig csak a kódot kell kijavítani.

A PROGRAM VÉGREHAJTÁSA

Az elkészült (vagy félkész) programot a

RUN

paranccsal futtathatjuk le. A parancs hatására a számítógép a program utasításait a sorszámoknak megfelelő sorrendben végrehajtja. Ha eközben formai (szintaktikai) hibát talál, a futás leáll, és hibaüzenet jelenik meg. A normális végrehajtást az jelzi, hogy a gép nem ad ki hibaüzenetet, és megjelenik a READY üzenet.

A KAPOTT EREDMÉNYEK ÉRTÉKELÉSE

Ennél a lépésnél feltételezzük, hogy programunk hibátlanul működik. Azt kell vizsgálni, hogy helyes volt-e a megoldás elve, valóban a várt eredményt kaptuk-e stb. Emellett a működés körülményeit is ellenőrizni kell, nevezetesen azt, hogy nem lehet-e lerövidíteni a program végrehajtását, nem tudunk-e biztonságosabb megoldást találni. Ez az utolsó lépés az előzőektől eltérően kevésbé formalizálható, és nagymértékben függ a feladattól.

KÓDOLÁSI ALAPISMERETEK

A programkészítés lépéseinek áttekintése után térjünk vissza az 1. feladat megoldásához, és folytassuk ezt a programkód elkészítésével. Előbb azonban a program elkészítéséhez szükséges **kódolási alapismeretekkel** és utasításokkal foglalkozunk.

A VÁLTOZÓ

A számítástechnikában — a matematikában kialakult fogalomhoz hasonlóan — a **változó** olyan adat, amelynek értéke változhat a feldolgozás közben. Ezért jellemzője nem egy konkrét érték, hanem a logikai funkció, amit a feldolgozásban betölt. A változóra egy szimbolikus névvel hivatkozunk, amely önállósítja és elkülöníti a többi adattól.

A számítástechnikában a változónak mindig konkrét értéke van, és ez a tárban tárolódik. Ha a változóval végzett műveletek során az értéke megváltozik, a korábbi érték elvész, és csak az új marad meg.

A változók tartalmuk szerint három típusba sorolhatók:

— numerikus
— egész értékű (numerikus)
— szöveges

Ebben a részben az első kettőt mutatjuk be, a szöveges változókat a 6. részben ismerjük meg.

A **numerikus** változók értéke egy valós szám lehet. A BASIC nyelvben betűkből és számokból álló neveket kaphatnak. Ez a név nem lehet utasításkulcsszó vagy bármilyen védett BASIC üzenet, parancs, függvénynév. A változónév hossza BASIC-változatonként eltérő, az általunk használt három gépen (Commodore–64, ■ HT, ▲ PRIMO) legfeljebb 16 karakter lehet, amelyből az első csak betű lehet. A gép viszont csak az első két karaktert jegyzi meg. Nézzük például az alábbi két változónevet:

ABLAK
ABBA

A gép nem tesz különbséget a két változó között, mivel a két első karakter egyezik mindkét névben. A következőket azonban különbözőnek tekinti:

A1BEM
A2BEM

A tévesztés lehetőségének elkerülésére a továbbiakban csak legfeljebb két karakter hosszúságú változóneveket fogunk használni. Az egész értékű változó csak egész értéket vehet fel bármilyen művelet eredményeként.

A változónévben kikötéseket tehetünk a változó típusára is. Így például előírhatjuk, hogy egy változó csak egész (integer) legyen. Ilyen esetben a változó nevéhez a % jelet kapcsoljuk. Ha ilyen megkülönböztetést nem teszünk, akkor a változó értéke vegyes szám lesz.

A fentiek értelmezésére nézzünk néhány helyes, illetve helytelen változónevet:

Helyes változónevek	Helytelen változónevek	
AB	A.B	(pontot tartalmaz)
AB%	1F	(számmal kezdődik)
X1	F%2	(a % után még áll valami)
X%	ABBA	(formailag helyes, de kerülendő, mivel 4 karakterből áll)

A Sinclair-gépeken a numerikus változók neve betűvel kezdődik, és betűkből, valamint számokból állnak. Hosszukat a tárméret korlátozza. Egész típusú változó nincs ezeken a gépeken. Érdemes megjegyezni, hogy ugyan kis- és nagybetűket lehet alkalmazni a változók nevéhez, de ugyanannak a betűnek a kis és nagy változatát azonos betűnek tekinti. Néhány példa:

Helyes változónevek	Helytelen változónevek	
OSSZEG	1OSSZEG	(számmal kezdődik)
EGY ES KETTO	EE—KK	(a kötőjel nem megengedett)
Vacak3		
VACAK3		

A gép a két utóbbi helyes változónevet nem tekinti különbözőnek.

A BASIC nyelvben a változókat nem szükséges a program elején meghatározni. A programban bárhol lehet új változókat bevezetni. A BASIC minden numerikus változónak automatikusan 0 értéket ad, ha valamelyik utasításban nem kap értéket.

FELDOLGOZÁSI LÉPÉSEK

A feldolgozás során az adatokkal számításokat, illetve logikai műveleteket végzünk. A számítási műveletek a BASIC-ben egyszerűen megadhatók:

A művelet matematikai jele	A művelet BASIC-ben
A+B	A+B
A—B	A—B
A·B	A*B
A:B	A/B
A^2	A↑2

Gyökvonás törtkitevős hatványozással végezhető el.

A változók és a közöttük álló műveleti jelek kifejezést alkotnak, amellyel valamit ki tudunk számítani. Egy kifejezés értékének kiszámítását és egy változóban való megőrzését **értékadásnak** nevezzük. Ezt szolgál a már látott

LET

utasítás.

Ha az A·B szorzatot egy C változóban akarjuk tárolni (megőrizni), akkor ezt a következő utasítással érhetjük el:

70 LET C=A*B

A LET utasításkulcsszót el is hagyhatjuk a Sinclair-gépek kivételével:

70 C=A*B

Az értékadó utasításban tehát egy egyenlőségjel fejezi ki az értékadást (egyenlővé tevést). Az egyenlőségjel bal oldalán mindig egyetlen változó állhat, a jobb oldalán pedig egy tetszés szerinti kifejezés. Megjegyezzük, hogy a bal oldali változó valamelyik jobb oldalon álló változó is lehet:

70 A=A*B

Ez azt jelenti, hogy A értéke felveszi az A*B szorzat értékét. Nagyon fontos, hogy az egyenlőségjel jobb és bal oldala akkor sem cserélhető fel, ha mindkettőn egy-egy változó áll. Például a

80 A=B

utasítás azt jelenti, hogy A felveszi B értékét. Ha

B=5,

akkor az utasítás hatására

A=5

lesz. Ha felcseréljük az egyenlőség jobb és bal oldalán álló változókat, akkor az eredmény más lesz.

80 B=A

Eszerint B veszi fel A értékét. Ha A=10 volt, akkor az utasítás hatására B is 10 lesz. A két eredmény tehát nem egyezik.

A bonyolultabb kifejezések sok műveletet tartalmaznak, ezért az egyértelműség kedvéért a műveletek között elsőbbségi rangsort kell kialakítani. A BASIC nyelvben is megengedett a zárójelek használata (figyeljük meg, hogy van nyitó és záró zárójel). A zárójelek ugyanolyan műveletvégzési sorrendet írnak elő, mint ahogy azt a matematika is előírja: a legbelső zárójeles kifejezést számítja ki először a program, majd innen fokozatosan halad kifelé:

((A1 + B1) * (A2 + B2)) ↑ K

1. lépés 2. lépés

3. lépés

4. lépés

A zárójeles kifejezésrészen belül vagy a zárójel nélküli kifejezésben az egyes műveletek sorrendjében a következő prioritási szabályok érvényesülnek:

1. Hatványozás.
2. Szorzás és osztás (ha több van, akkor ezek elvégzése egymás után, balról jobbra halad.
3. Összeadás és kivonás azonos prioritással (ha több van ezekből a műveletekből, akkor a kiszámítás itt is balról jobbra halad).

Ha egy kifejezésben több azonos szintű szabályba tartozó művelet van, akkor a műveletek kifejtése balról jobbra halad:

A*B/C

Értelmezése:

1. D=A*B
2. E=D/C

vagy

A*B+C*D
1. E=A*B
2. F=C*D
3. G=E+F

vagy

A+B
C*D
1. E=A+B
2. F=C*D
3. G=E/F

Vigyázzunk! Az alábbi értelmezés helytelen:

1. E=A+B
2. F=E/C
3. G=F*D

AZ EREDMÉNYEK KIÍRÁSA

Mint már az 1. részben láttuk, a feldolgozás eredményeit a PRINT kulcsszavú utasítással lehet kiírni. A PRINT utasítás tárgyában idézőjelbe tett karaktereket a program változatlanul kiírja, az idézőjel nélkülieket változóknak tekinti, s ezek tartalmát írja ki. Ha az utasítás tárgya kifejezés, akkor ennek értékét előbb kiszámítja és utána írja ki.

Ha például az *eredő ellenállás* szöveget akarjuk kiíratni, akkor ezt a

```
100 PRINT"EREDO ELLENALLAS"
```

utasítással lehet elérni.

Ha a feladat a C változó értékének kiíratása, akkor a

```
110 PRINT C
```

utasítást kell megadni. Ha az érték mellett a C változó nevét is ki akarjuk íratni, akkor a

```
120 PRINT"C=" ;C
```

utasítással oldhatjuk meg a feladatot. Ennek hatására a program a következő kiírást végzi el (tegyük fel, hogy C értéke 125):

```
C=125
```

Figyeljük meg, hogy a kiíratni kívánt adatelemeket pontosvesszővel választottuk el.

A BASIC nyelv csak bizonyos intervallumba tartozó számokat ír ki **fixpontos** (valós) formában. A Commodore esetében ezt az intervallumot a

```
0.01, 999999999
```

számok határolják. Az intervallumon kívül eső értéket **lebegőpontos** formában írja ki a gép:

(előjel) X.XXXXXXXX$\pm^n$

általános formában. A **mantissza** (a szám jegyei) 9 jegyű, és az n (kitevő) két szélső értéke a

```
−39 és +38
```

A Commodore BASIC esetében a legkisebb szám, amellyel még számolási művelet végezhető, a

```
+2,93873588E−39
```

és a legnagyobb a

```
+1,70141183E+38
```

szám.

Egész értékű változó

−32768 és +32767

közötti értéket vehet fel.

- A HT gép

0.01 és 999999

intervallumban jeleníti meg a számokat valós formában.
A legkisebb érték: −1,701411E±38
A legnagyobb érték: 1,701411E±38
Az egész értékű változók intervalluma megegyezik a Commodore-éval.

- A PRIMO a

0.01 és 999999

intervallumban ír ki számokat valós formában.
A legkisebb értéke: 9.9E−38
A legnagyobb érték: 1.7E+38
Az egész értékű változók intervalluma itt is megegyezik a Commodore-éval.

- A Sinclair-gépek legfeljebb 8 jeggyel írnak ki számokat. Olyan számokat, melyek kiírásakor ennél több jegy szükséges, lebegőpontos formában ír ki a gép.
A legkisebb érték: 10^{-38}
A legnagyobb érték: 10^{38}

A PROGRAM BEFEJEZÉSE

A program végét az END utasítással lehet jelölni, de használata nem kötelező. Az END hatására a programvégrehajtás megáll, ezért az END legyen a legnagyobb sorszámú utasítás. Használata egyszerű:

500 END

- A Sinclair-gépeken nincs ilyen utasítás.

Az 1. feladat befejezése

Ezek után térjünk vissza az 1. feladathoz, és oldjuk meg az itt bemutatott ismeretek begyakorlására.

A kódolás

(1) *A bemeneti adatok tárolása*
Ez a program első modulja, ezért a modul elejére dokumentációs célból elhelyezzük a program nevét REM utasítások között. Ezután a modul kezdetén a modul

```
1Ø REM* * * * * * * * * * * * * * * *
2Ø REM*                             *
3Ø REM*          EREL               *
4Ø REM*                             *
5Ø REM* * * * * * * * * * * * * * * *
6Ø REM* * * * * BEMENO ADATOK TAROLASA * * * * *
7Ø LET    R1 = 1ØØØ
8Ø LET    R2 = 2ØØØ
9Ø LET    R3 =  5ØØ
1ØØ LET   R4 =  2ØØ
11Ø REM* * * * * ELLENALLAS-SZAMITAS * * * * *
12Ø LET   KØ = 1/R1+1/R2+1/R3
13Ø LET   RØ = 1/KØ
14Ø LET   R9 = RØ+R4
15Ø REM* * * * * EREDMENYKIIRAS * * * * *
16Ø PRINT "AZ EREDO ELLENALLAS (OHM):"; R9
17Ø END
```

20. ábra. Az 1. feladat kódja

nevét adjuk meg. A modul tényleges kódjában pedig az egyes ellenállások értékeit kell megadni. Ezt a LET értékadó utasítással végezhetjük el. Ezek alapján a kódot kézzel megírhatjuk (20. ábra).

(2) *Ellenállás-számítás*
Ez a modul is megoldható értékadó utasításokkal. Figyeljük meg azonban azt a különbséget, hogy itt nem közvetlenül számértéket rendelünk hozzá egy változóhoz, hanem az érték egy kifejezés kiszámításával keletkezik. A folyamatábra tagjait egy-egy utasítással lehet kódolni. A modul elejére itt is elhelyezzük a modul nevét. Ez a modul az (1) modul után hajtódik végre, ezért az utasításokat az (1) modul végéhez folyamatosan sorszámozhatjuk. Ezek után a modul kézzel írt kódja a 20. ábrán látható.

(3) *Eredménykiírás*
A modul kiírási feladatát egyetlen PRINT utasítással el lehet végezni. Ezt a modult is célszerű a modul nevével bevezetni. Mivel ez a program utolsó modulja, a modul végére END utasítást kell írni. A *kiíró* modul a *számítási* modul után következik, ezért a sorszámozást a (2) modultól lehet folytatni (20. ábra).

A program bevitele, kipróbálása és a megoldás értékelése

A program teljes tervezése után a programot be kell gépelni:

```
10 REM*************
20 REM*             *
30 REM*     EREL    *
40 REM*             *
50 REM*************
60 REM***** BEMENO ADATOK TAROLASA *****
70 LET R1=1000
80 LET R2=2000
```

A program bevitele után a program tesztelési célokra futtatható:

```
RUN
?SYNTAX ERROR IN 70
READY
```

Az első futás után hibaüzenet jelent meg, amely szerint a programban szintaktikai hiba van. Ha megnézzük a bevitel listáját, rögtön szembeötlik, hogy a 70-es sorban az R1 után kihagytuk az egyenlőségjelet, s mivel nem volt egyenlőségjel, a program nem tudta értelmezni ezt a sort. Ekkor a programfutás leállt, és READY üzenet jelent meg. A hibás sort ki kell javítani. Ezután újra megkíséreljük a futtatást.

```
AZ EREDO ELLENALLAS (OHM) :485.714
READY
```

- A Sinclair-gépeknél ilyen formai hiba nem fordulhat elő, mivel a gép a formailag hibás utasításokat nem fogadja el. Ilyenkor egy kérdőjel jelenik meg az ENTER lenyomásakor. A sort csak a javítás után fogadja el a gép.

A futás most már eredményes. Csupán azt kell ellenőrizni, hogy a program logikailag is helyesen működik-e. Példánkban ennek legegyszerűbb módja, ha az eredő ellenállás értékét magunk is kiszámítjuk:

$$R9 = \frac{1}{\frac{1}{1000} + \frac{1}{2000} + \frac{1}{500}} + 200 = 485.714 \text{ ohm}$$

Megállapíthatjuk, hogy programunk helyesen működik, a feladatot megoldja. A hibátlan programot ezután a megismert módon kazettára vagy lemezre másolhatjuk.

Ellenőrző kérdések és feladatok

1. Mi a különbség a képernyősor és az utasítássor között?

   Ha lenyomjuk a sorzáró gombot, akkor új képernyősorba vagy új utasítássorba megyünk?

2. Egy programot az AUTO segítségével kezdett el beírni. Majd abbahagyta a beírást (megszakította az AUTO módot), és most folytatni akarja. Lehet AUTO-val folytatni egy program beírását?

3. Ha van rá lehetősége, készítse el a meglevő programjai listáit!

4. Készítse el az alábbi két programot!
   Magyarázza meg, mi a különbség a kettő között!

   a)
   ```
   10 LET A=20
   20 LET B=15
   30 LET A=B
   40 PRINT A
   ```

   b)
   ```
   10 LET A=20
   20 LET B=15
   30 LET B=A
   40 PRINT A
   ```

5. Készítsen BASIC programot az A kiszámítására!

$$A = \frac{(A+B)^2}{C \cdot D} \cdot K$$

5. ELÁGAZÁSOK I.

Az elágazásos (IF THEN, IF THEN ELSE, CASE) szerkezet megvalósítása.
A 2. feladat megoldása

Az előzőekben olyan feladatokat oldottunk meg, amelyekben az utasítások egy meghatározott sorrendben hajtódtak végre minden esetben. Vannak azonban olyan feladatok is, amelyek így nem oldhatók meg. Nézzünk egy példát!

ELÁGAZÁSOS SZERKEZETEK

Egy szállítóvállalatnak vissza kell igazolnia a rendelések értékét. A vállalat egyetlen terméket állít elő, és egységárát a rendelt mennyiségtől teszi függővé. Két árat határoz meg. Ha egy vevő egy meghatározott (M) mennyiséget vagy ennél kevesebbet rendel, akkor a termék egységára A1 lesz. Ha viszont e feletti mennyiséget rendel, akkor alacsonyabb (A2) egységárat számít fel. Foglaljuk össze az elmondottakat egy táblázatban:

Mennyiség (R)	$R \leqslant M$	$R > M$
Ár (A)	A_1	A_2

ahol

M — az árengedmény határát rögzítő mennyiség,
R — a rendelt mennyiség,
A — a visszaigazolt ár,
A_1 és A_2 — a rendelt mennyiségtől függő árak.

Ahhoz, hogy az egységárat megállapítsuk, a rendelt mennyiséget (R) és az árengedmény határát rögzítő mennyiséget össze kell hasonlítani. Ha:

$R \leqslant M$, akkor az egységár (A) A1
$R > M$, akkor az egységár (A) A2

Ebben a feladatban a rendelt mennyiség teljes értékét ($R \cdot A$) vagy az A1, vagy az A2 értékkel kell kiszámítani. Vegyük észre, hogy mindig csak az egyik egységárral kell számolni, sohasem mind a kettővel. Ezt a feladatot tehát soros műveletekkel nem lehet megoldani, mert amikor arra kerül a sor, hogy melyik egységárat vegyük figyelembe, akkor el

kell dönteni, hogy a kettő közül melyiket válasszuk. Ha ezt a feladatot program formájában is meg akarjuk oldani, akkor az egységár kiválasztása előtt szükség van egy műveletre, amely meghatározza, hogy mi a teendő.

Honnan lehet tudni a példánkban, hogy melyik egységárat kell tekintetbe venni? Jól érzékelhető a példákból, hogy az R és M értéke határozza ezt meg. Tehát a döntés az R és M viszonyától függ, amint már be is mutattuk.

Azt is tudnunk kell, hogy a programba mind az A1-gyel, mind az A2-vel való értékszorzást bele kell írni, mivel mindkettő előfordulhat. A program tehát mind a két alábbi műveletet tartalmazni fogja:

$$E=R\cdot A1$$
$$E=R\cdot A2$$

ahol

E — a rendelés értéke.

Most már pontosan látható, hogy hogyan kell ezt a feladatot megoldani:

1. Össze kell hasonlítani az R-et M-mel.
2. Ha $R \leqslant M$, akkor $E=R\cdot A1$
   egyébként $E=R\cdot A2$
3. Visszaigazolás készítése E összeggel

Mindhárom lépést el kell végezni, de a másodikban levő kettő közül csak az egyiket. A második műveletben a program végrehajtása elágazik vagy az egyik, vagy a másik műveletre. Ezért az ilyen szerkezetű műveleteket **elágazásnak** nevezzük. A harmadik műveletet mindenképpen el kell végezni, ez már független az R és M viszonyától.

Látható, hogy egy **feltétel** határozza meg, hogy a két műveletcsoportból melyiket választjuk ki. Ez a feltétel

$$R \leqslant M$$

vagy

$$R > M$$

formában fogalmazható meg. Bármelyik jó, csak arra kell figyelni, hogy melyik műveletet kell a **feltétel teljesülése** és melyiket **nem teljesülése** esetén elvégezni.

Akármelyiket is alkalmazzuk, a feltétel teljesülése a megrendelt mennyiségtől függ, mivel M értékét (legalábbis egy elég hosszú időre) konstansnak tekintjük. Ezért R a **feltétel változója.** Az elágazás tehát az R feltételváltozó értékétől függ.

A BASIC nyelvnek van olyan lehetősége, amely lehetővé teszi az elágazás megvalósítását a programban. Erre szolgál az IF utasítás. Az IF utasítás felépítése a következő:

x IF *1. kifejezés* összehasonlítás *2. kifejezés* { THEN *utasítás* / THEN *sorszám* / GO TO *sorszám* }

Az utasítás kulcsszavai az IF és a THEN vagy a THEN helyett álló GO TO.

Az utasítás lényege, hogy két kifejezést hasonlít össze, és az összehasonlítás eredményétől függő utasítássorozatot hajtja végre. Ha az 1. és 2. kifejezés között az utasításban foglalt feltétel **teljesül**, akkor a program a THEN utáni részt, illetve a GO TO utasítást hajtja végre, majd a következő sorszámú sorra lép. Ha a feltétel **nem teljesül**, akkor a program a következő sorszámú utasításra tér át.

Az utasításban kifejezések (változókból és műveletekkel összekapcsolva) és egyedül álló változók lehetnek. Az egyik kifejezés helyett konstans is állhat. Ha van konstans, akkor az rendszerint a második kifejezés helyén áll.

Az összehasonlítást mindig valamilyen **relációval** fejezzük ki. Ha például a reláció az =, akkor az összehasonlítás tartalmilag azt jelenti, hogy a program egyenlőséget vizsgál a két kifejezés között. Ha teljesül (egyenlőek), akkor a THEN-ág hajtódik végre, ellenkező esetben az utasítás utáni sorszámú utasítást hajtja végre a gép.

Az elvégezhető összehasonlítások műveleti jelei megegyeznek a matematikában használatos relációkkal, csupán a formájukban van kisebb eltérés.

Matematikai relációk	BASIC-relációk
=	=
<	<
⩽	=< vagy <=
>	>
⩾	=> vagy >=
#	<> vagy ><

A THEN után állhatnak utasítások (kettősponttal elválasztva), vagy állhat egy utasítássorszám. Ilyenkor a programot az itt megadott utasítássorszámtól kell folytatni. Tartalmilag ugyanez érvényesül a GOTO-nál is. Ekkor a GOTO után írt utasítássorszámnál folytatódik a program. A GOTO leírására később visszatérünk.

- A Sinclair-gépeknél a THEN után mindig utasítást kell megadni, a sorszám nem elegendő.

Jól érzékelhető, hogy a bemutatott IF utasítás IF THEN változata akkor használható a legjobban, ha valamilyen feltételtől függően egy műveletet vagy el kell végezni, vagy nem. Például a szorzatkitaláló feladat egyik megoldási módjánál a gép által kiszámított eredményt csak akkor íratjuk ki, ha az nem egyezik meg az általunk begépelttel. Ekkor

a feltétel a két szorzat egyenlősége. Ha ez nem áll fenn, akkor ki kell írni a helyes szorzatot, és a program befejeződik. Ha fennáll reláció, akkor a kiírást nem kell végrehajtani, és a program befejeződik.

Ilyenkor tehát nem két művelet közül kell valamelyiket kiválasztani, hanem csak **egy** művelet elvégzéséről kell dönteni. Megjegyezzük azonban, hogy ezzel az utasítással is meg lehet valósítani a két művelet közül az egyik kiválasztását.

Az IF utasításnak van egy több szolgáltatást nyújtó változata is, amely a

- HT és a
- PRIMO gépen használható.

Az utasítás formája:

$$x\ \text{IF}\ \textit{1.kif.}\ \text{összehas.}\ \textit{2.kif.}\ \left\{\begin{matrix}\text{THEN}\ \textit{utasítás}\\ \text{THEN}\ \textit{sorszám}\\ \text{GOTO}\ \textit{sorszám}\end{matrix}\right\}\ \text{ELSE}\ \left\{\begin{matrix}\textit{utasítás}\\ \textit{sorszám}\\ \text{GOTO}\ \textit{sorszám}\end{matrix}\right\}$$

Az utasítás kulcsszavai az IF THEN és ELSE, ezért hogy az IF THEN utasítástól megkülönböztessük, IF THEN ELSE az utasítás neve.

Az utasítás lényegében ugyanúgy működik, mint az IF THEN utasítás. A legnagyobb eltérés az, hogy ha a reláció nem teljesül, akkor a gép az ELSE után álló utasításokat hatja végre.

Megjegyezzük, hogy mind az IF THEN, mind az IF THEN ELSE utasításba további ugyanolyan típusú utasítások is elhelyezhetők:

```
100 IF A=7 THEN IF B=3 THEN A=B
```

Ez esetben ha az A=7 és B=3, akkor az A=B értékadás megy végbe: A értéke 3 lesz.

Ismerjük meg még a GO TO utasítást is. A GO TO utasítás segítségével lehet elérni, hogy a program a sorrendben következő helyett a GO TO utasítás tárgyában megadott sorszámú utasításra ugorjon, és onnan folytassa a végrehajtást. A GO TO utasítás felépítése:

x GO TO *sorszám*

Amikor a program a végrehajtáskor eléri az *x* sorszámú utasítást, akkor a GO TO utasítás végrehajtásának eredményeként nem az *x* után következő sorszámú utasítás hajtódik végre, hanem a GO TO utasítás tárgyában megadott sorszámú utasítás. A GO TO tehát a programon belül ugrást hajt végre.

Ezután azt kell megvizsgálni, hogy az elágazásos szerkezeteket hogyan lehet megvalósítani a programokban.

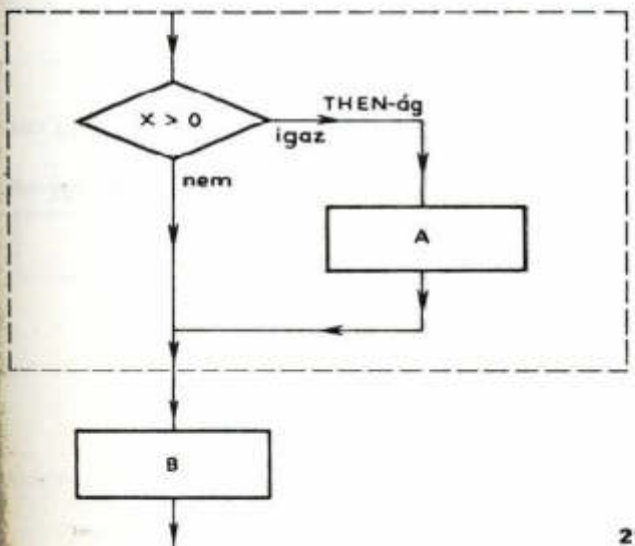
AZ IF THEN SZERKEZET MEGVALÓSÍTÁSA

Az IF THEN szerkezetet olyan algoritmusban kell használni, amelyben a feltétel teljesülésekor valamilyen A műveletsorozatot kell végrehajtani, és utána a B műveletsorozatra kell áttérni. Ha a feltétel nem teljesül, akkor rögtön a B műveletre kell áttérni. A B műveletsorozatot mindenképpen végrehajtja a program, ezért ezt feltétel nélkül végrehajtandó műveletnek nevezzük. Az IF THEN szerkezet folyamatát a 21. ábra mutatja.

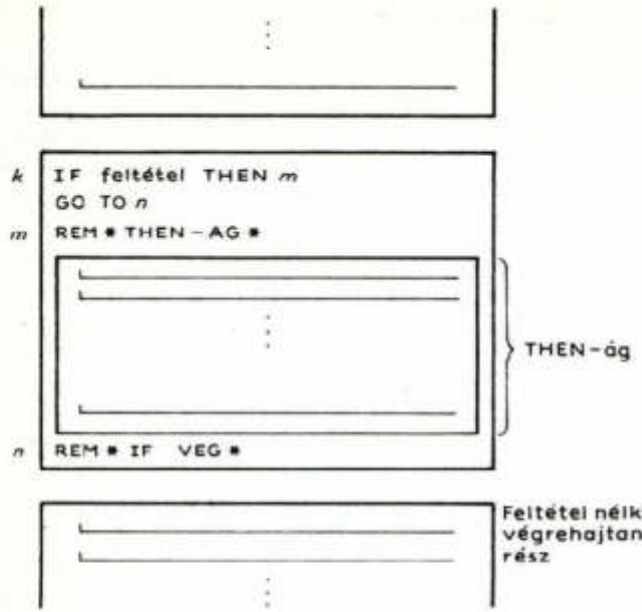
Az IF THEN szerkezet kódolására két változatot mutatunk be. Az első formát akkor célszerű használni, ha az A programrész sok utasításból áll. A kódrészt az IF utasítás vezeti be, és az IF VEG megjegyzéssel (REM) zárjuk le (22. ábra). A THEN után következő műveletsorozatot az *m* sorszámtól kezdve, két sorral az IF utasítás alatt (így $m=k+20$ lehet) helyezzük el. A műveletet egy REM utasítással nyitjuk meg, s ide beírjuk, hogy itt kezdődik a THEN-ág. A THEN-ág utolsó utasítása után az *n* sorszámú, az IF VEG-et tartalmazó REM utasítás következik, és a vezérlés átkerül a közös műveletsorozat kezdetére.

Ha a feltétel nem teljesül, akkor a program a következő sorra tér át. Logikailag azonban a B műveletsorozatnak kell következnie, amit úgy érhetünk el, hogy egy GO TO utasítást helyezünk az IF utasítás utáni (például $k+10$ sorszámú) sorba, s ez az *n* sorszámú utasításra — gyakorlatilag a közös műveletsorozatra — helyezi át a vezérlést.

Ez utóbbi lépésnek az a lényege, hogy az IF THEN zárt szerkezeti egységnek egyetlen „kijárata" legyen, az *n* sorszámú utasítás. Ez az „egykijáratúság" módosításnál nagyon megkönnyíti a helyzetünket.



21. ábra. Az IF THEN szerkezet



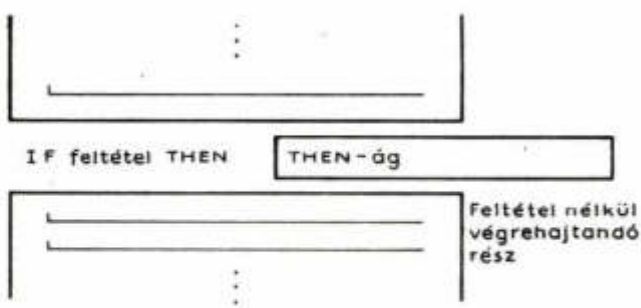
22. ábra. Az IF THEN szerkezet a BASIC-ben

Példaként nézzük meg a szorzatkitaláló program eredményértékelő részének kódolását. A begépelt szorzat változója az A, a gép által kiszámított szorzat a B változóban van. A kód a következő:

```
100 IF A<>B THEN 120
110 GO TO 140
120 REM* THEN AG *
130 PRINT"A SZORZAT: ";B
140 REM* IF VEG *
150 END
```

A második szerkezet akkor használható ésszerűen, ha a THEN-ág csupán egy-két utasításból áll.

Ezt a kódolási szerkezetet a 23. ábra mutatja. A THEN-ág egy-két



23. ábra. Az egyszerűsített IF THEN szerkezet

utasítását a THEN kulcsszó után, kettősponttal elválasztva helyezzük el. Az utolsó THEN-ági utasítás után a feltétel nélkül végrehajtandó ág következik, ezért GO TO utasításra nincs szükség, mert a program ettől függetlenül is a következő sorban levő B műveletsorozat első utasítására lép. Ha a feltétel nem teljesül, akkor is a B műveletsorozatra tér át a program, tehát a kódolás helyesen valósítja meg az IF THEN szerkezetet.

Nézzük meg az előző példát ezzel az eljárással kódolva! Megtehetjük, mert a THEN-ágban csak egy utasítás lesz:

```
100 IF A<>B THEN PRINT"A SZORZAT: ";B
110 END
```

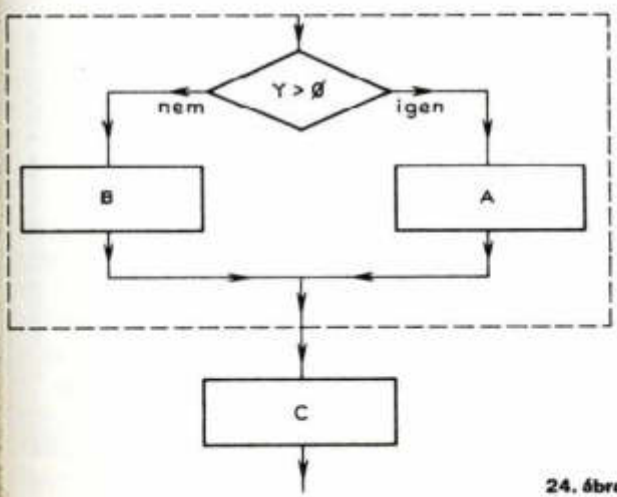
AZ IF THEN ELSE SZERKEZET MEGVALÓSÍTÁSA

Az IF THEN ELSE szerkezet akkor használható, amikor egy feltétel teljesülésekor a THEN-ágban levő A műveletsort kell elvégezni, és a C feltétel nélkül végrehajtandó műveletre kell áttérni. Ha a feltétel nem teljesül, az A helyett egy B műveletsort kell elvégezni; ez az ELSE-ág. Ezután C részre kell áttérni (24. ábra).

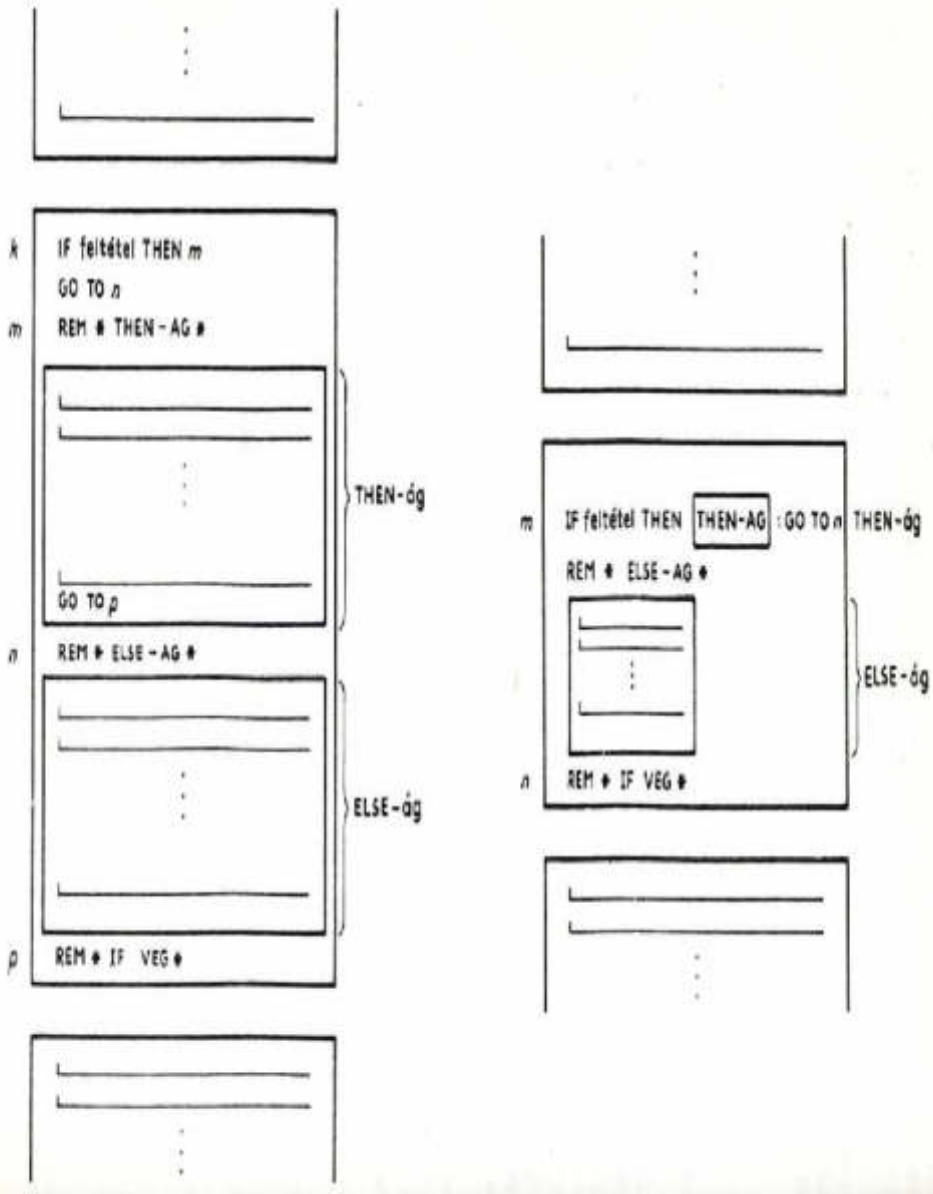
Az IF THEN ELSE szerkezet kódolására is két megoldást mutatunk be. Az első megoldás a 25. ábra bal oldalán látható.

A THEN-ág kódolása az IF THEN szerkezetnél bemutatott módszerhez hasonlóan megy végbe, csak a befejezés tér el. A THEN-ág után most nem a C műveletsor, hanem az ELSE-ág kódja következik, ezért egy GO TO utasítással át kell ugrani a *p* sorszámú utasításra, ahol az IF THEN ELSE kijárata (a C rész kezdete) van.

Ha a feltétel nem teljesül, akkor az ELSE-ágra kell áttérni. Ezt az IF



24. ábra. Az IF THEN ELSE szerkezet



25. ábra. Az IF THEN ELSE szerkezet a BASIC-ben

utasítás után következő sorban elhelyezett GO TO utasítással lehet elérni. Ekkor az ELSE-ág végrehajtódik, és a vezérlés közvetlenül átkerül a *p* sorszámú megjegyzésre, illetve innen a feltétel nélkül végrehajtandó ágra.

Készítsük el a termékrendelési és visszaigazolási feladatrészlet kódolását az itt bemutatott módszer szerint:

```
100 IF R<M THEN 120
110 GO TO 150
120 REM* THEN AG *
130 E=R*A1
140 GO TO 170
150 REM* ELSE AG *
160 E=R*A2
170 REM* IF VEG *
180 PRINT"A RENDELES ERTEKE:␣";E
```

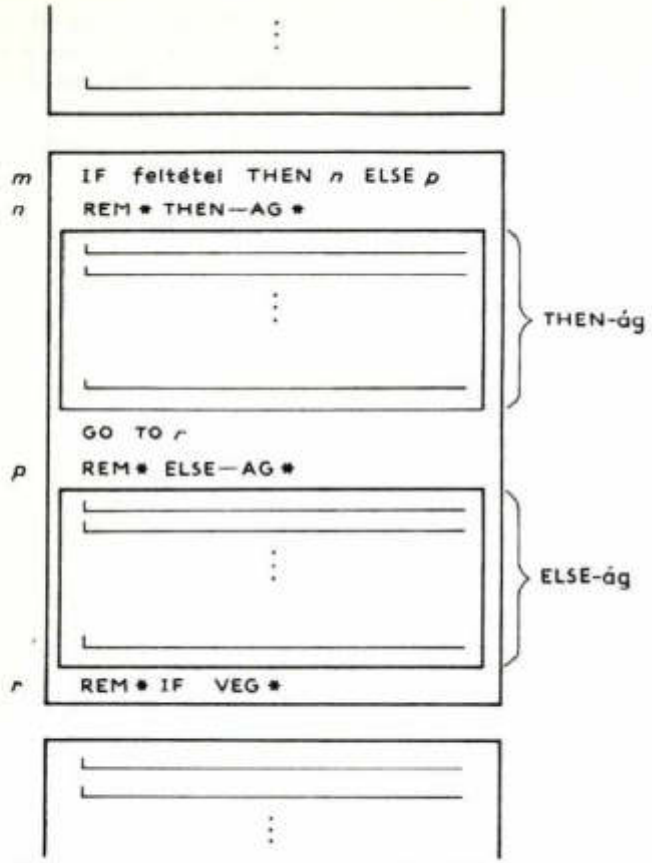
A 180-as sorszámú sorban a ␣ jel egy szóközt jelent.

■ A HT és a
▲ PRIMO BASIC-változatában az ilyen műveletet IF THEN ELSE típusú utasítás felhasználásával kódolhatjuk. Itt a THEN és az ELSE után meg kell adni, hogy melyik sorszám alatt kezdődnek az egyes ágak. A THEN- és az ELSE-ágat hasonló, strukturált módon lehet kódolni. Ilyenkor az IF utáni GO TO utasítás feleslegessé válik (26. ábra).

A feladatot most az IF THEN ELSE utasítással kódoljuk:

```
100 IF R<M THEN 110 ELSE 140
110 REM* THEN AG *
120 E=R*A1
130 GO TO 160
140 REM* ELSE AG *
150 E=R*A2
160 REM* IF VEG *
170 PRINT"A RENDELES ERTEKE: ";E
```

Ha a THEN-ág csupán egy-két utasítást tartalmaz (a feltétel jó megválasztásával eldönthető, hogy melyik ág legyen a THEN-ág), akkor az IF THEN szerkezet megvalósításánál bemutatott egyszerűsített szerkezet alkalmazható. A THEN-ág utasításait a THEN kulcsszó után, kettősponttal elválasztva írjuk le, majd a sor végén egy GO TO utasítással az IF szerkezet végére kell ugrani, s ezután már a közös rész következik. Ha a feltétel nem teljesül, akkor az IF THEN ELSE szerkezetnek megfelelően az ELSE-ág hajtódik végre. Az ELSE-ág befejezése után pedig a feltétel nélkül végrehajtandó rész kerül sorra (a 25. ábra jobb oldala).



26. ábra. IF THEN ELSE szerkezet IF THEN ELSE utasítással

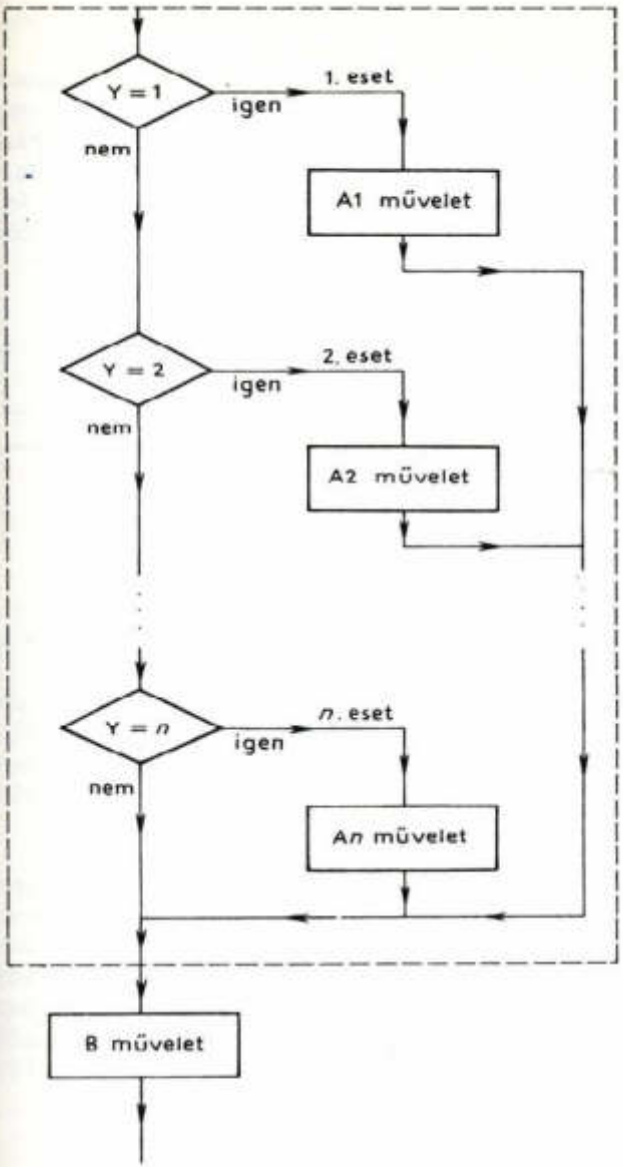
Ezzel példánk kódja a következő lesz:

```
100 IF R<M THEN E=R*A1: GO TO 130
110 REM* ELSE AG *
120 E=R*A2
130 REM* IF VEG *
140 PRINT"A RENDELES ERTEKE: ";E
```

Az IF THEN ELSE utasítással pedig az alábbi lesz az egyszerűsített kód:

```
100 IF R<M THEN E=R*A1 ELSE E=R*A2
110 PRINT"A RENDELES ERTEKE: ";E
```

Ha a feladat olyan, hogy egy feltételtől függően nemcsak kétfelé lehet elágazni, hanem többfelé, akkor a feladat folyamatábrája a 27. ábra szerint alakul. *Az ilyen típusú elágazást CASE szerkezetnek nevezzük.*



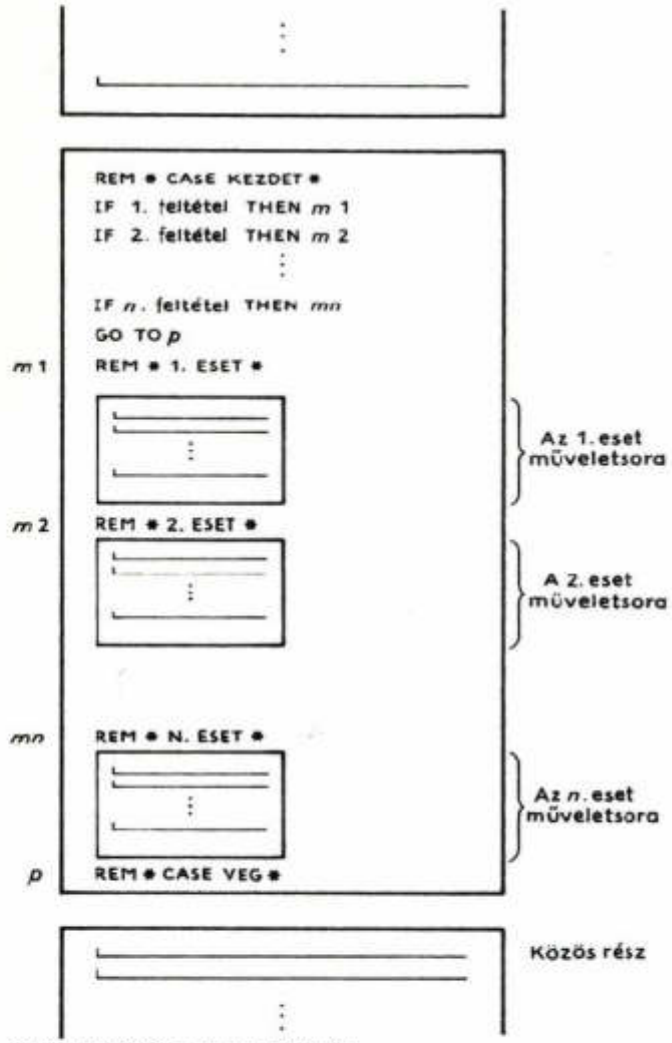
27. ábra. A CASE szerkezet

A BASIC-ben a CASE szerkezet az ON GO TO vagy az IF THEN utasítással kódolható. Az utóbbi nyújtja az általánosabb és rugalmasabb megoldást, ezért ennek a használatát mutatjuk be. A CASE szerkezet kódolására is két megoldást ismertetünk.

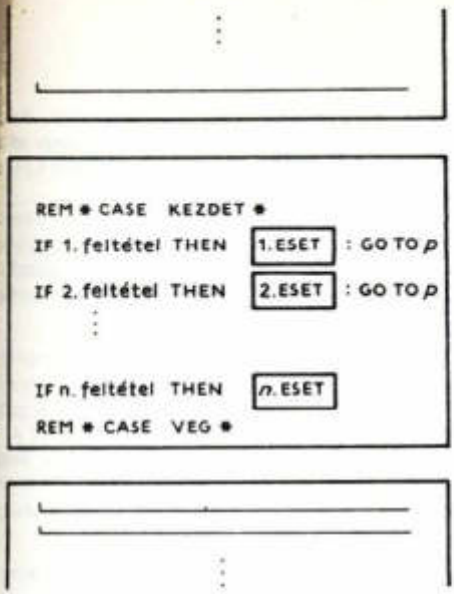
A CASE szerkezetet az 1. feltétel IF utasítása nyitja meg, és a CASE

VEG megjegyzés zárja le. A CASE szerkezet elején egymás után helyezkednek el az egyes feltételeket vizsgáló IF utasítások.

Ha a változó értéke valamelyik feltételnek nem tesz eleget, akkor a vezérlés a következő vizsgálatra kerül át. Ha a változó egyik feltételnek sem felel meg, akkor a feltétel nélkül végrehajtandó rész végrehajtása következik, tehát a *p* sorszámú – utolsó – utasításra kell ugrani. Ha a változó valamelyik feltételnek eleget tesz, akkor a vezérlés az adott feltételhez tartozó műveletsorra ugrik át. Ennek végrehajtása után a közös rész következik, erre egy GO TO utasítással kell áttérni (28. ábra).



28. ábra. A CASE szerkezet kódolása



29. ábra. Az egyszerűsített CASE szerkezet

A CASE szerkezet egyszerűsített változatában (29. ábra) az egyes esetek kódját a megfelelő feltételt vizsgáló IF utasításban, a THEN kulcsszó után helyezzük el. Ha az *i*-edik feltétel teljesül, akkor az *i*. esethez tartozó műveletsor hajtódik végre. Ha valamelyik *i* eset végrehajtódott, akkor a CASE részből egy GO TO utasítással ki kell lépni.
Most pedig ismerjük meg részletesen a billentyűzetről adatot beolvasó INPUT utasítást! Az utasítás formája:

x INPUT"*szöveg*";1. *változó*, 2. *változó*, . . .

Hatására az idézőjelbe tett szöveg kiíródik, és utána egy kérdőjel jelenik meg, annak jeléül, hogy a gép adatot vár a billentyűzetről. Ilyenkor olyan típusú (pl. egész típusú vagy numerikus) és annyi adatot kell vesszővel elválasztva begépelni, amilyen és ahány az utasítás tárgyában szerepel. Ha az adatbevitel befejezése előtt nyomjuk le a sorzáró gombot, akkor a következő sorban újra kérdőjel látható, és a gép a hiányzó adatok begépelésére vár.

- A Sinclair-gépeknél az INPUT utasítás formája kismértékben eltér a fentitől:

  x INPUT"*szöveg1*",*változó1*;"*szöveg2*",*változó2*; . . .

  Látható, hogy egy INPUT utasítással több változó értékét be lehet olvasni, de mindegyikhez külön szöveget kell megadni. A szöveg tudatja a felhasználóval, hogy a gép milyen adatokat kér be. Az utasítás befejeztével az utasítás tárgyában felsorolt változók a begépelt értékeket kapják meg.

2. feladat

Állóeszközök amortizációval csökkentett nettó értékét kell meghatározni. Az állóeszköz beszerzésének éve és ára (bruttó ár) alapján ki kell számítani, hogy jelenleg mennyi az állóeszköz nettó értéke. A két kiinduló adatot a végrehajtás közben billentyűzetről kell beadni.

A vizsgált állóeszközök leírási időszaka 5 év. A leírás lineáris, vagyis az állóeszközök nettó értéke az eredeti (bruttó) érték évenként 20%-kal csökkentett értéke. 5 év után, amikor a nettó értéknek nullára kellene csökkennie, az állóeszközöket könyvelési okokból 100,– Ft eszmei értékre írják le, és ezután ezen az értéken tartják nyilván.

Az eredményt az alábbi formában kell kiírni:

```
NETTO ERTEK SZAMITASA

BRUTTO ERTEK          X  EFT
A BESZERZES EVE    19XX  EV
ELETKOR               X  EV
NETTO ERTEK           X  EFT
```

vagy nullára leírt állóeszközöknél:

```
NETTO ERTEK SZAMITASA

BRUTTO ERTEK          X  EFT
A BESZERZES EVE    19XX  EV
ELETKOR               X  EV
NETTO ERTEK         0.1  EFT (ESZMEI ERTEK)
```

A feladat elemzése

A feladat leírása szerint az állóeszközök nettó értékét kell kiszámítani a bruttó érték és a beszerzés éve alapján az ide vonatkozó szabályok alkalmazásával. Az eredményt a megadott formában kell kiírni.

- A Sinclair-gépek képernyőjén az utolsó sor nem fér el, ezért az "ERTEK)" részt az "(ESZMEI" szöveg alá lehet kiírni.

A felhasználó az állóeszközök adatait kartonokon tárolja, ezért ilyenkor számítógépes adattárolásra nincs szükség.

A program elején a számításhoz szükséges bemeneti adatokat be kell olvasni, és ezekből kell a nettó értéket kiszámítani, végül az eredményt ki kell írni.

Az amortizációszámítást a következő képlet szerint végezhetjük el:

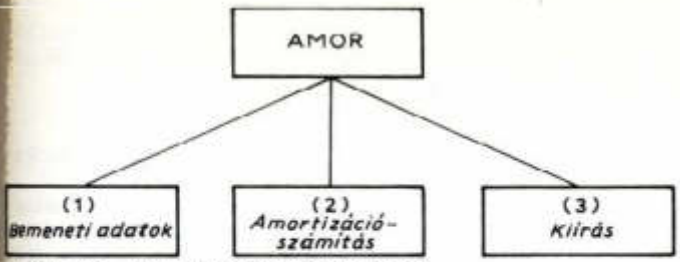
$$N = \frac{B[5-(J-E)]}{5} \qquad \text{ha } (J-E)<5$$

ahol

N – a nettó érték eFt-ban,
B – a bruttó érték eFt-ban,
J – a folyó év (csak kerek évszámot használunk),
E – a beszerzés éve.

A kiírásnál az állóeszköz életkorát is meg kell határozni, ezért a (J–E) helyett a K (életkor) változót vezetjük be. A nullára leírt (5 éves vagy régebbi) állóeszközök nettó értékét pedig az

$$N=0.1 \text{ eFt} \qquad (\text{ha } K \geq 5)$$



30. ábra. A 2. feladat programjának szerkezete

értékadással határozzuk meg. Az állóeszköz életkora dönti el, hogy melyik kifejezést alkalmazzuk a nettó érték kiszámításánál. A feltételváltozó tehát a K (életkor) változó lesz.

A fentiek alapján már meghatározható, hogy a feladat megoldásához milyen modulok kellenek. Először szükség van egy *bemeneti adat* modulra, az amortizációszámításhoz egy másik modulra, végül egy *kiírás* modulra. A modulok a 30. ábrán látható egyszerű szerkezetet alkotják. A programnak az AMOR nevet adjuk.

A program tervezése

A program szerkezete kialakult. Vizsgáljuk meg, hogy a **modulszerkezet** hogyan **finomítható** tovább!

A *bemeneti adatok* modulban csupán egy állóeszköz beszerzési évének és bekerülési árának adatait kell beolvasni. Ez elég egyszerű feladat ahhoz, hogy egyetlen modul lássa el. A modulban kell meghatározni a számításhoz szükséges J (folyó évszám) adatot is. Láthatjuk, hogy az amortizációszámítás is alapjában véve egyszerű, ezért további finomításra nincs szükség.

Ehhez hasonlóan könnyen belátható, hogy a *kiírás* modul bontására sincs szükség, mert csak a címet és a négy adatot kell egyszerű formában kiírni. Látnunk kell, hogy az utolsó sor formája nettó érték, és tartalma a K változótól függ.

Mivel a modulok szekvenciális szerkezetet alkotnak, a vezérlőmodul alkalmazásától eltekintünk.

A modulok tervezése

(1) *Bemeneti adatok*
A modul **adat** típusú. Bemeneti adatait a B és az E, kimenetét pedig három adat: B, E, J alkotja.

A modul folyamata a J konstans értékadásával kezdődik. A következő lépésben kell beolvasni a B és az E értékét INPUT utasításokkal. A modul müveletei a 31. ábrán láthatók.

(2) *Amortizációszámítás*
A modul **eljárás** típusú. Bemeneti adatai: a B, E és J változó az (1) modulból. A modul állítja elő az életkor (K) és a nettó érték (N) kimeneti adatokat.

Először a K (életkor) értékét kell meghatároznia, mert a K-ra a továbbiakban szükség lesz. Ezután következik a nettó érték kiszámítása (N), illetve az N=100 Ft eszmei értékadás 5 évnél régebbi állóeszközök esetén. Hogy melyik müveletet kell elvégezni, az a

$$K < 5$$

feltételtől függ.

Ha a feltétel teljesül, akkor az amortizációval csökkentett nettó értéket kell kiszámítani, egyébként a nettó értéknek 0.1 eFt értéket kell adni. A soron következő modul kezdő művelete független a K értékétől (31. ábra).

(3) *Kiírás*

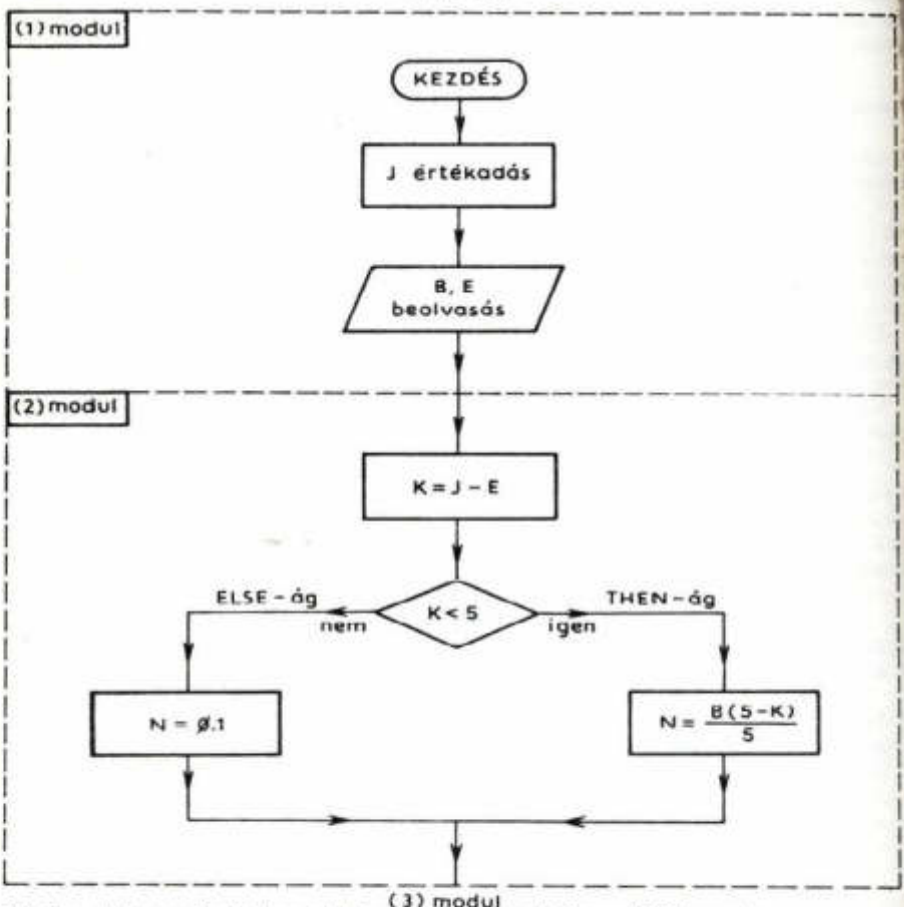
A modul **eljárás** típusú. Bemeneti adatai a B és E az (1) modulból, valamint K és N a (2) modulból.

A modul kimenete a cím és az eredmények kiírása. A műveletek végrehajtásakor először a címet (aláhúzással), majd a B, E és K adatot kell a megfelelő szöveggel kiírni. A következő sor kiírási formája attól függ, hogy az állóeszköz nettó értéke nagyobb-e nullánál, vagy eléri-e a nullát, illetve eszmei értéken tartják-e nyilván. A megfelelő kiírási formát az életkor határozza meg. Ha

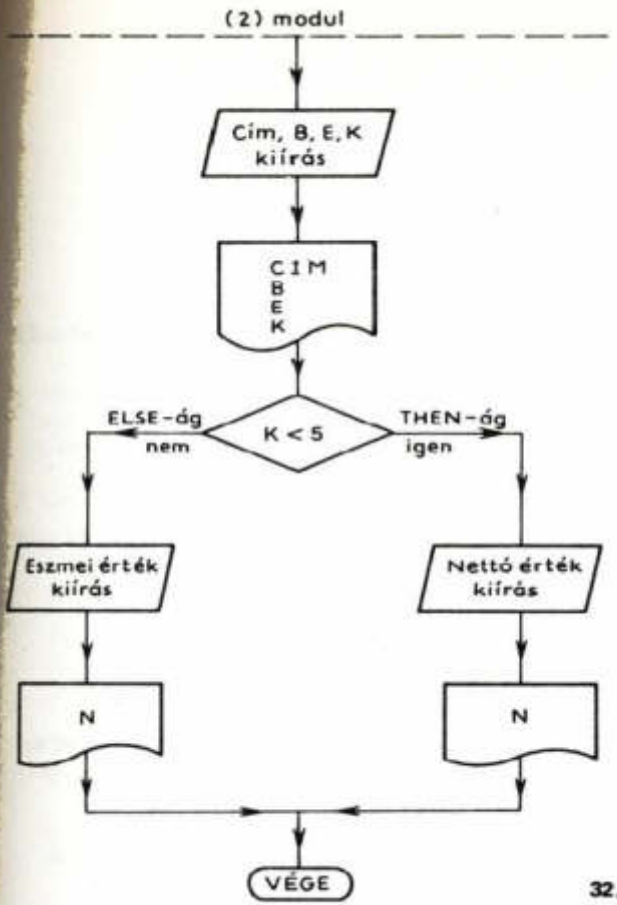
$$K < 5,$$

akkor a számított nettó értéket, ha

$$K \geq 5,$$



31. ábra. A bemeneti adatok modul és az amortizációszámítás modul folyamata



32. ábra. A (3) modul folyamata

akkor az eszmei értéket tartalmazó N változó értékét (100,— Ft) kell kiírni, de zárójelben meg kell jegyezni, hogy ez az eszmei érték. Ezután a program véget ér. A modul folyamatábráját a 32. ábra mutatja.

A kódolás

A program kezdetén helyezzük el a program nevét tartalmazó REM utasítássorokat. Az (1) modulban két INPUT utasítás van:

```
90 INPUT"BRUTTOERTEK: ";B
100 PRINT
110 INPUT"BESZERZES EVE: ";E
```

A (2) modulban egy IF THEN ELSE szerkezetet kell kódolni a bemutatott szabályok szerint. Amikor elkezdjük kódolni ezeket a sorokat, akkor még nem tudjuk kitölteni az ugrási címeket:

```
140 IF K<5THEN
150 GO TO
160 REM* THEN AG *
```

Most már befejezhetjük a 140. sort:

```
140 IF K<5THEN 160
```

Majd folytatjuk a THEN-ág kódolását:

```
170 N=(B*(5-K))/5
180 GO TO
190 REM* ELSE AG *
```

Az ELSE-ág kezdetének ismeretében az ide ugró utasítást fejezhetjük be:

```
150 GO TO 190
```

Ezzel folytatjuk az ELSE-ágat:

```
200 N=0.1
210 REM* IF VEG *
```

Az IF THEN ELSE szerkezetnek a végére jutottunk, és most már a THEN-ág utáni GO TO utasítást is befejezhetjük:

```
180 GO TO 210
```

A kiíró modulban először a címet írjuk ki, majd az aláhúzást

```
230 PRINT"          NETTO ERTEK SZAMITAS"
240 PRINT"          ----- ----- ---------"
```

Ezután a bruttó értéket, a beszerzés évét és az életkort kell kiírni. Majd ismét egy IF THEN ELSE szerkezet következik a nettó érték szövegének kiírásához. Végül a programot lezáró END utasítás következik.

Ellenőrző kérdések és feladatok

1. Magyarázza meg, hogy ha egy feltételtől függően vagy egy A műveletsort, vagy egy B műveletsort kell végrehajtani, akkor miért kell mind a két műveletsort belevenni a programba!

2. Keressen példát a CASE szerkezet alkalmazására a környezetéből!

3. Gépelje be az alábbi programot, és figyelje meg, mi történik!

```
10 A=1
20 PRINT A;
30 IF A<2 THEN GO TO 10
```

Elképzelhető olyan feladat, ahol ez a program használható?

4. Alakítsa át a szorzatkitaláló programot, hogy a program hibás szorzat begépelése esetén írja ki:

"NEM JO, PROBALJA MEG UJRA!"

és az eredményt lehessen újra beírni.
Ha a beírt szorzat jó, akkor az

"EZ JO!"

szöveget írja ki a gép, és a program fejeződjék be.

5. Készítsen programot a másodfokú egyenlet gyökeinek kiszámítására!

$$X_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

6. ELÁGAZÁSOK II.

Az elágazások folytatása, a vezérlőmodul kialakítása, a modulok mint szubrutinok. A módosított 2. feladat. A 3. feladat megoldása

A VEZÉRLŐMODUL ALKALMAZÁSA

Nézzük meg az előző részben megoldott amortizációszámítási feladat modulszerkezetét (30. ábra). A modulszerkezetből kitűnik, hogy a legegyszerűbb – a szekvenciális – esettel állunk szemben. Éppen ezért a feladat megoldásánál azt mondtuk, hogy a vezérlőmodul elhagyható, és a három modul kódját szekvenciálisan egymás után írjuk a programkódban. Ezzel viszont nem tartottuk be programfelépítési módszerünk egyik szabályát.

Készítsünk vezérlőmodult a feladat három moduljához! A vezérlőmodul a 30. ábra szerinti AMOR lesz, amelyet a megoldás során nem kódoltunk. Ennek az új modulnak az a részfeladata, hogy a feladat moduljainak végrehajtását vezérelje, ami abból áll, hogy a modulokat (1), (2), (3) sorrendben hívja. A modulok hívása után a folyamat befejeződik (33. ábra).

Kérdés az, hogyan kódolható ez a modul. Egy modul hívása azt jelenti, hogy a főmodulból ki kell ugrania a hívott modul első utasításához. Végre kell hajtani a modul egészét, és a befejezéskor nem a következő modulra kell áttérni, hanem a vezérlőmodulba kell visszaugrani, hogy az

33. ábra. Az 1. feladat vezérlőmoduljának folyamata

hívhassa a következő modult (34. ábra). Jelenlegi ismereteink alapj ezt a műveletet a GO TO utasítással meg lehet valósítani.
A program elején egy GO TO utasítással lépünk az (1) modul elejé majd ennek végéről másik GO TO utasítással visszatérünk a vezérl modulba.

Ha egy újabb modult beszúrunk a programba, akkor ennek a progra elején „helyet" kell csinálni. Ez azt jelenti, hogy a programnév utá minden utasítást hátrább kell tenni, vagyis az utasítássorszámokat me kell növelni. Tegyük fel, hogy az (1) modul kódja a 200-as, a (2) mo dulé a 300-as, a (3) modulé pedig az 500-as soron kezdődik majd. Ezz a program kiegészítő kódja (vezérlőmodul) a következő lesz:

```
60 REM***** AMOR *****
70 GO TO 200
80 GO TO 300
90 GO TO 500
100 END
200 REM***** BEMENETI ADATOK *****
    :
260 GO TO 80
300 REM***** AMORTIZACIO SZ. *****
    :
420 GO TO 90
500 REM***** KIIRAS *****
    :
670 GO TO 100
```

- A Sinclair-gépeknél nincs END utasítás, ezért a 100 sorszámú utasítás helyett

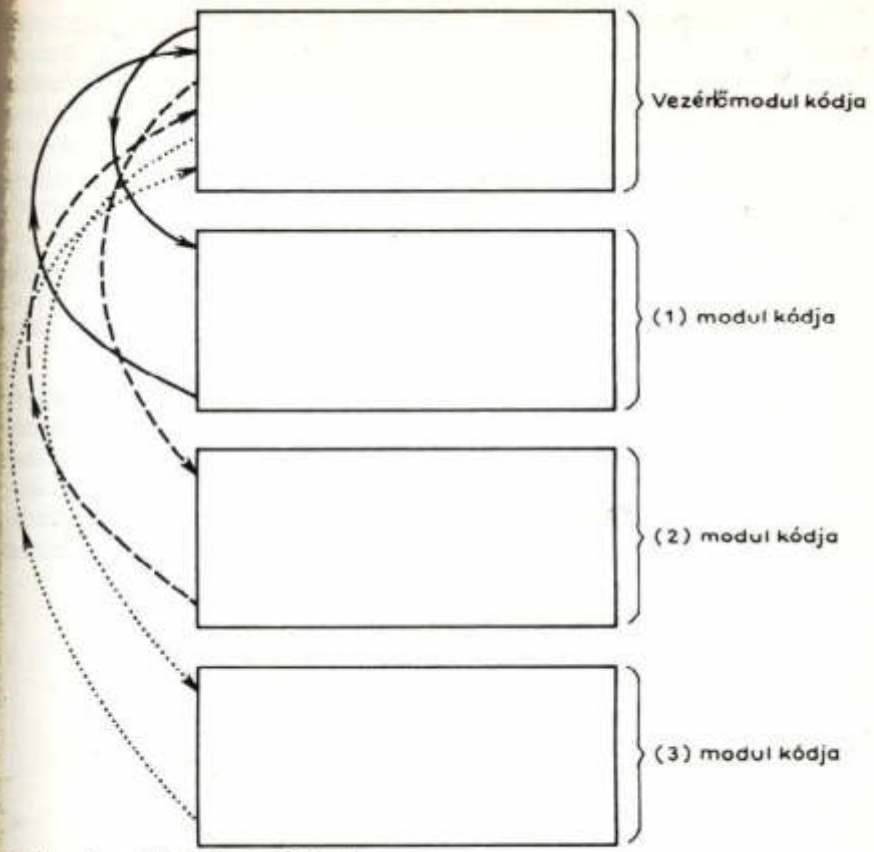
  100 GO TO 700

  utasítást írunk és a 700-as soron pedig egy „semleges" utasítást helyezünk el (pl. REM).

A feladatot ugyan helyesen oldottuk meg, de a GO TO utasítások nagyon merevvé teszik a programot, mert bármilyen módosításnál a sorszámokat át kell írni, és sok az ugrálás, ami kódoláskor könnyen elteveszthető. Van azonban ennél kellemesebb megoldás is, a következőkben ezt ismerjük meg.

A GOSUB, RETURN utasításpárral szubrutinokat tudunk kódolni a programban, és ezek végrehajtását lehet irányítani velük. A szubrutin egy alfeladat (vagy részfeladat), amely lehet egy modul, modulrészlet, vagy akár több modul. A szubrutin sajátossága az, hogy végrehajtását a GOSUB, RETURN utasításpár irányítja.

A szubrutin formában kódolt modulokat a program végére írjuk, hogy a szerkezet könnyebben áttekinthető legyen. A szubrutin kezdetét semmilyen szubrutin-irányító utasítás nem jelzi, így a program ol-



34. ábra. A modulok hívása GOTO-val

vasója nem veszi észre egy utasításról, hogy az szubrutinkezdés-e, vagy sem. A szubrutin kezdőutasítását sorszáma alapján határozhatjuk meg. A szubrutint minden esetben a RETURN utasítással kell befejezni.

A programnak azon a pontján, ahol egy szubrutint kell hívni,

x GOSUB *sorszám*

szubrutinhívó utasítást adunk meg.

A GOSUB utasítás tárgyában levő szám azt a sorszámot jelöli, ahol a szubrutin kezdődik. Az utasítás hatására a vezérlés átkerül a megadott sorszámra.

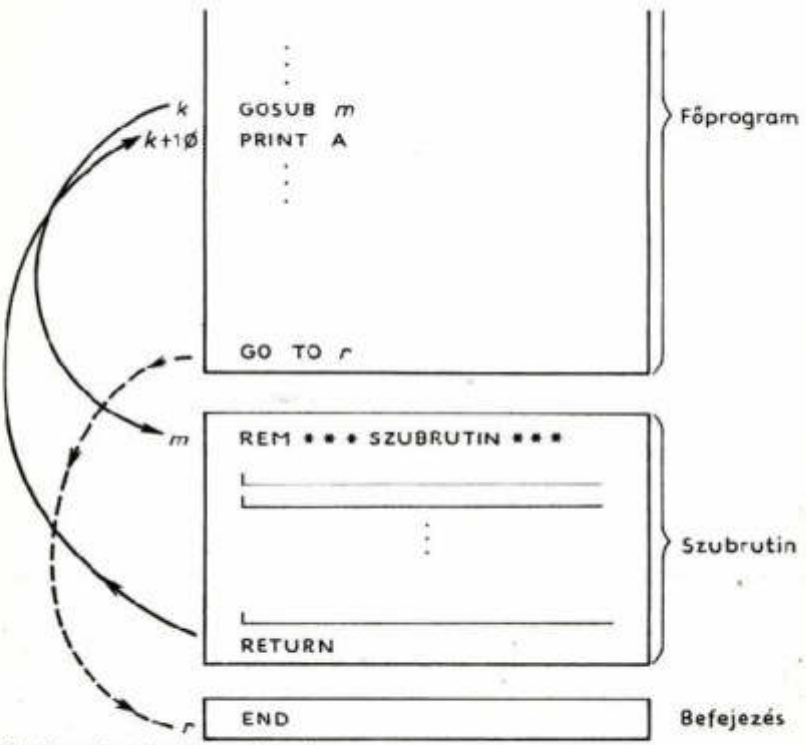
Ezután a hívott szubrutin végrehajtódik. Amikor a program eljut a szubrutin végét jelölő RETURN utasításhoz, akkor a szubrutin befejeződik, és a vezérlés visszakerül a **hívó GOSUB utasítást követő utasításra,** és ezután ott folytatódik. Láthatjuk, hogy a szubrutinhívást a GOSUB, a visszatérést a főágra pedig a RETURN utasítás valósítja meg.

A szubrutinhívási műveletet a 35. ábra mutatja. A GOSUB utasítás hatására a program az *m* sorszámú utasításnál folytatódik, amely a szubrutin kezdete. A gép végrehajtja a szubrutint, majd a RETURN utasítás hatására a hívó GOSUB utasítást követő (*k*+10) utasításra tér át a program.

Az ábrából az is látható, hogy ha egy program szubrutint is tartalmaz, akkor a programkód ezáltal két részre osztható:

- szubrutint nem tartalmazó rész, melyet **főprogramnak** vagy főágnak nevezünk;
- szubrutinokat tartalmazó rész.

Felhívjuk a figyelmet arra, hogy a főág végrehajtása után a program rátérne az első szubrutin végrehajtására anélkül, hogy erre szükség lenne. Ilyenkor eljutna az első RETURN-ig, de nem tudna sehova sem visszatérni, mert GOSUB hívás nélkül került a szubrutinba. Ezért a gép hibaüzenetet ad ki, és a végrehajtást befejezi. Ez a program helyes működését is befolyásolhatja, ezért a főág befejezése után GO TO utasítással a



35. ábra. A szubrutint tartalmazó program

program utolsó – END – utasítására adjuk át a vezérlést, hogy a szubrutinokat kikerüljük.

Az elmondottak alapján könnyű belátni, hogy a vezérlőszerkezet egyszerűbb lesz, ha a vezérelt modulokat szubrutinoknak tekintjük, és a vezérlőmodulban ezek hívását helyezzük el.

Ha visszatérünk a 2. feladat vezérlőmodullal való megvalósításához, de most már szubrutinokkal, akkor az (1), (2) és (3) modult szubrutin formájában kell kódolnunk, a vezérlőmodult pedig ezek hívásaiból kell összeállítanunk. Ha feltételezzük, hogy az egyes szubrutinmodulok ugyanazokon a sorszámokon kezdődnek, akkor az eredeti program kiegészítő kódja a következő lesz:

```
60 REM***** AMOR *****
70 GOSUB 200
80 GOSUB 300
90 GOSUB 500
100 GO TO 700
200 REM***** BEMENETI ADATOK *****
    :
260 RETURN
300 REM***** AMORTIZACIO SZ. *****
    :
420 RETURN
500 REM***** KIIRAS *****
    :
670 RETURN
700 END
```

- A Sinclair-gépeknél a korábbi megállapításunk értelmében

  700 REM

  utasítást írunk a program befejezésére.

2. feladat módosítása

A 2. feladat úgy módosul, hogy van egy szervezet, amelynek 10 állóeszköze van, és ezek amortizációját kell kiszámítani évente olyan módon, hogy az állóeszközök adatait ne kelljen minden alkalommal beírni. Ez így kevesebb kézi munkát igényel (és kevesebb hiba fordulhat elő).

Mielőtt a módosítást elemeznénk, azt vizsgáljuk meg, hogyan lehet adatokat a programmal együtt tárolni, és van-e olyan lehetőség, hogy az értékadásnál egyszerűbb megoldást kapjunk. A BASIC nyelvben erre szolgál a READ és a DATA utasításpár. Az adatokat a DATA utasítás tárgyában soroljuk fel, és a felsorolás sorrendjében a READ utasítással olvassuk ki. A **READ utasítás** a kiolvasott adat értékét a **READ utasítás tárgyában levő változóhoz rendeli.**

A DATA utasításokat célszerű a program végén elhelyezni. A DATA szó után vesszővel elválasztva olyan sorrendben soroljuk fel a használni kívánt adatokat, amilyen sorrendben ki akarjuk őket olvasni:

x DATA 6,210,45

A READ utasítást oda kell helyezni, ahol szükség van a változó értékének megadására. A READ utasítás tárgyában meg kell adni azt a változót (vagy változókat), amelynek értékét a DATA-ból kiolvasott adattal kell egyenlővé tenni:

110 READ N

Térjünk vissza a 2. feladat módosításához, és oldjuk meg. A megoldás folyamán az eredeti feladattól való eltéréseket emeljük ki!

A feladat elemzése

A feladat annyiban tér el az eredetitől, hogy az állóeszközök alapadatait (bruttó érték, a beszerzés éve) a programban kell tárolni és nem kartonokon. Ebből az is következik, hogy az alapadatokat a program futása során nem kell begépelni. Mivel a nettó érték számítását évente kell elvégezni – és eredménye is az évszámtól függ –, a számításokhoz mindig meg kell adni az évszámot (már csak azért is, mert ez változik, és az eredmények ettől is függnek).

A feladat megoldásához szükség van egy *évszámbeolvasó*, egy *amortizációszámítási*, egy *kiírási*, valamint egy *alapadat-tároló* modulra.

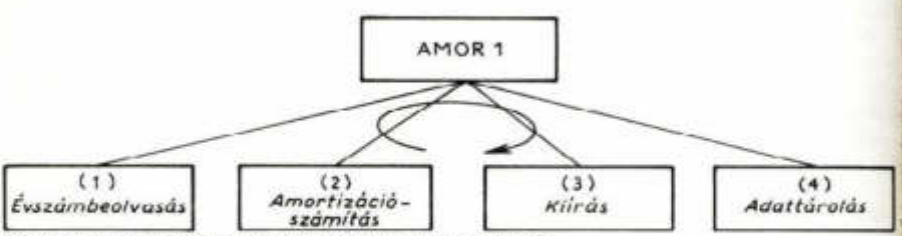
A program tervezése

A programban változatlanul szükség van *amortizációszámítási* és *kiírási* modulra. A *bemeneti adatok* modul helyett csak *évszámbeolvasó* modul szükséges, és új modulként az *adattároló* modullal bővül a program (36. ábra).

Az eredeti program egy futása egy állóeszköz nettó értékét számítja ki. A módosított feladatban pedig 10 állóeszközét kell kiszámítani. Ez csak úgy lehetséges, ha a (2) és a (3) modult 10-szer hajtja végre a program változó adatokkal.

Azt is figyelembe kell venni, hogy a 10 állóeszköz adatai nem férnek el együtt a képernyőn. A program futása során az állóeszközönkénti cím és a 4 adatsor végigszalad a képernyőn, és csak az utolsó két táblázatot lehetne elolvasni. Ez nyilván nem kívánatos, ezért állítsuk meg a kiírást állóeszközönként. Ezt úgy lehet elérni, hogy a táblázat kiírása után egy INPUT utasítást adunk ki. Amíg a felhasználó nem gépel be valamit, a program futása – és a kiírt kép – áll. Tulajdonképpen teljesen mindegy, hogy milyen adatot kérünk be. Nyilvánvaló azonban, hogy a helyzethez legjobban illő adatot kell kérni. Ha a felhasználó folytatni akarja a műveletet, akkor 1-et, ellenkező esetben 0-t kell begépelnie. Ez utóbbi esetben a program befejeződik.

A feladatot vezérlőmodul alkalmazásával oldjuk meg. Valamennyi modul végrehajtását a vezérlőmodul irányítsa! A többi modult szubrutin formájában kódoljuk.



36. ábra. A módosított 2. feladat programjának szerkezete

A modulok tervezése

(1) *Vezérlés*
A modul **vezérlő** típusú. Bemeneti adata a folytatást eldöntő T érték. Kimenete a három szubrutin hívása, illetve a program befejezése.

A modulban először a (2) modult kell hívni, utána pedig a (3) és a (4) modult. Ezután be kell olvasni a vezérléshez szükséges T adat értékét. Ha

$$T=1,$$

akkor ismét a (3) és a (4) modult kell hívni egy újabb amortizációszámításhoz és kiírásához. Ha

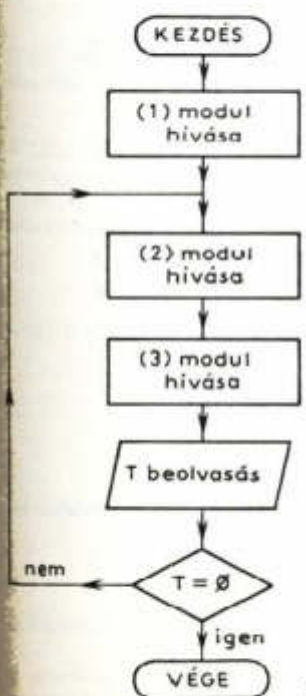
$$T=0,$$

akkor a program befejeződik. Ezt a folyamatot a 37. ábra mutatja.

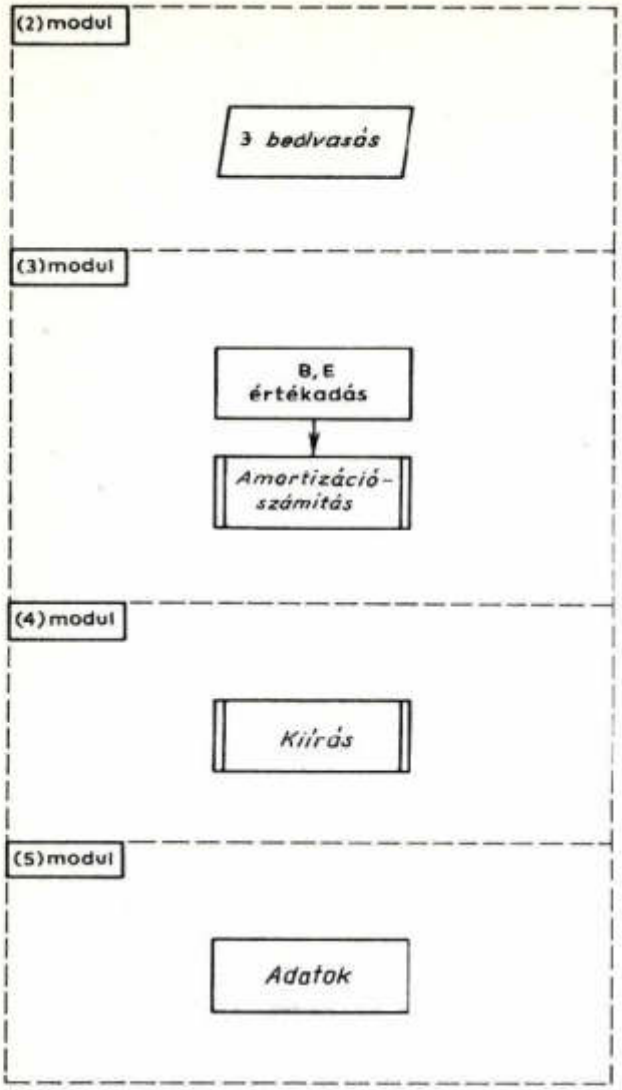
(2) *Évszámbeolvasás*
A modul **adat** típusú. Bemeneti adata a folyó év (J) adat billentyűzetről begépelve (38. ábra).

(3) *Amortizációszámítás*
A modul megegyezik az eredeti program azonos nevű moduljával, azzal az eltéréssel, hogy a modul elején minden végrehajtás előtt be kell olvasni a soron következő állóeszköz alapadatait (B és E – 38. ábra).

(4) *Kiírás*
Ez a modul változatlanul átvehető (38. ábra).



37. ábra. A (1) modul folyamata



38. ábra. A 2.1 feladat szubrutinjai

(5) *Adatok*
A modul **adat** típusú. Egyetlen feladata, hogy a 10 állóeszköz következő adatait tárolja (38. ábra):

	Bruttó érték (eFt) B	Beszerzés éve E
1.	2500	1982
2.	500	1984
3.	3050	1984
4.	162	1984
5.	520	1980
6.	1610	1978
7.	3960	1979
8.	865	1980
9.	32	1976
10.	785	1981

A program kódolása

Itt csupán a program módosult részeinek kódolását mutatjuk be.

Az (1) modul kódja a szubrutinhívásokat és a befejezés eldöntését tartalmazza:

```
60 REM***** VEZERLES *****
70 GOSUB 200
80 GOSUB 300
90 GOSUB 500
100 INPUT"TOVABB ? (IGEN=1, NEM=0) ";T
110 IF T=1 THEN 80
120 GO TO 860
```

A (3) modul kódjában az első utasítás a soron következő gép alapadatainak olvasása lesz:

```
310 READ B,E
```

A (2), (3) és (4) modulokat szubrutinokként kell megírni, ezért az utolsó utasítás mindegyikben a RETURN lesz.

A (4) modul kezdetén töröljük a képernyőt. Ezt a műveletet PRINT utasítással lehet elvégezni, ha a SHIFT lenyomásával együtt a CLR karaktert „nyomtatjuk ki".

■ A HT gép, ▲ a PRIMO és ● a Sinclair-gépek esetében a CLS utasítással lehet ezt elérni.

Végül a programot ki kell egészíteni az (5) modul DATA utasításaival.

```
800 REM***** ADATOK *****
810 DATA 2500,1982,500,1984
820 DATA 3050,1984,162,1984
830 DATA 520,1980,1610,1978
840 DATA 3960,1979,865,1980
850 DATA 32,1976,785,1981
860 END
```

A kapott eredmények értékelése

A program a feladatát hibátlanul ellátja, de ha az utolsó állóeszköz amortizációjának kiírása után T=1 választ adunk, akkor

```
OUT OF DATA
```

hibaüzenet jelenik meg. Ennek az az oka, hogy a 10. számítási és kiírási lépés után is a program újabb állóeszközadatait próbálja beolvasni. Mivel több adatot nem talál, a fenti hibaüzenettel figyelmezteti a felhasználót, hogy a READ utasítást és az utána következő részt nem tudja végrehajtani.

Felhívjuk az olvasó figyelmét, hogy a fenti ugyan egy lehetséges megoldás, de az ún. **veszélyes programszerkezetek** közé tartozik, mert a GO TO utasításciklust képez a programban, és a programozó nem ellenőrzi, hogy a ciklus mikor fejeződik be. Az ellenőrzést a gépre bízzuk, és az akkor állítja le a programot, amikor elfogynak az adatok. A 8. részben a ciklikus problémák megoldására szabályos szerkezetet mutatunk be, de más szerkezetek alkalmazásával is ki lehet kerülni az ilyen megoldást.

A 3. feladat megoldása előtt ismerjünk meg egy új fogalmat!

A SZÖVEGES VÁLTOZÓK

Egyes feladatok megoldása azt is igényli, hogy a program ne csak numerikus, hanem szöveges változókat is tudjon kezelni. A BASIC nyelvnek vannak ilyen lehetőségei. Egy szöveges adat korlátozott hosszúságú lehet.

A Commodore-nál egy szöveges változó 255 jel hosszúságú lehet.

- Ugyanez az értéke a HT gépeknél is.
- ▲ A PRIMO-nál csak 50 jelből állhat a sorozat (de kibővíthető max. 9450-re).

Szöveges adat például egy név:

KOVÁCS PÁL

A szöveges adatot meghatározáskor idézőjelek közé kell tenni, hogy a program tudja értelmezni.

A szöveges adatok a numerikus adatokhoz hasonlóan lehetnek: konstansok és változók.

A **szöveges konstans** a program során nem változtatja tartalmát. Ilyen szöveges konstans a PRINT utasítás tárgyában megadott szöveg:

```
110 PRINT"        NETTO ERTEK SZAMITAS"
```

Ha a szöveges adat tartalma a program során megváltozik, vagy értékét ugyan nem változtatja meg, de többször kell rá hivatkozni, akkor a **szöveges** adatot **változóként** kezeljük. Ekkor a szöveges adat változónevet kap. A szöveges adatok változónevét a numerikus változók nevéhez hasonlóan képezzük, azzal az eltéréssel, hogy a változónév végére $ jelet kell írni. Ilyen lehet például:

A$, C5$

- A Sinclair-gépeknél a szöveges változó neve egyetlen betű lehet (és utána a $ jel).

A szöveges változók a már ismert módokon kaphatnak értéket. Értékadás történhet a LET utasítás segítségével:

```
110 LET B$="␣␣␣␣␣NETTO ERTEK SZAMITAS"
```

3. feladat

Magánszemélyek nettó jövedelmét kell kiszámítani a bruttó jövedelemből a megfelelő jövedelemadó és községfejlesztési hozzájárulás levonása után. A feladatot a 45/1983. (XI. 20.) Minisztertanácsi rendelet előírásai szerint kell megoldani. Egyszerűsítésként személyenként és évente egy alkalommal kell a számolást elvégezni a kumulált éves bruttó jövedelem alapján.

A feladat megoldása során a következő adatokat kell kiírni:

BRUTTO JOVEDELEM	XXXXXX	Ft
JOVEDELEMADO	XXXX	Ft
KOFA	XXX	Ft
NETTO JOVEDELEM	XXXXXX	Ft

A feladat elemzése

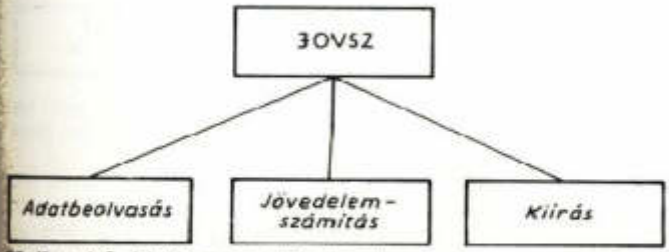
A feladat megoldási menete az idézett rendelet értelmében a következő: a bruttó jövedelemből le kell vonni 5% költséghányadot és a befizetett társadalombiztosítási összeget. A maradék a jövedelemadó alapja. A jövedelemadó mértékét az alábbi táblázat segítségével kell meghatározni:

Adóalap	Adó
20 000 Ft-ig	2%
20 001– 40 000 Ft	400 Ft és a 20 000 Ft-on felüli rész 6%-a
40 001– 60 000 Ft	1 600 Ft és a 40 000 Ft-on felüli rész 10%-a
60 001–100 000 Ft	3 600 Ft és a 60 000 Ft-on felüli rész 20%-a
100 001–200 000 Ft	11 600 Ft és a 100 000 Ft-on felüli rész 38%-a
200 001–400 000 Ft	49 600 Ft és a 200 000 Ft-on felüli rész 50%-a
400 001– 600 000 Ft	149 600 Ft és a 400 000 Ft-on felüli rész 60%-a
600 001–	269 600 Ft és a 600 000 Ft-on felüli rész 65%-a

A községfejlesztési hozzájárulás (KÖFA) mindig az adó 10%-a. A feladat egyszerű algoritmussal megfogalmazható, ezért számítógéppel megoldható.

A számítás elvégzéséhez két adat szükséges: a bruttó jövedelem és a társadalombiztosítási összeg. Az eredményadatok ezekből már kiszámíthatók.

A feladat megoldásához szükség van egy *adatbeolvasó* modulra, egy *számítási* modulra és egy *kiíró* modulra (39. ábra).



39. ábra. A 3. feladat programjának szerkezete

A program tervezése

Vizsgáljuk meg az egyes modulokat kicsit részletesebben! Az *adatbeolvasási* modulban két adatot (bruttó jövedelem = BJ és társadalombiztosítás = TB) kell beolvasni a billentyűzetről. Egyszerűsége miatt további bontása felesleges.

A nettó értékszámítási funkció két lépésre osztható: a bruttó jövedelemből a szükséges levonásokkal elő kell állítani az adóalapot, majd ebből az adót kell kiszámítani. Ez utóbbi művelet során meg kell vizsgálni, hogy az adóalap melyik sávba esik, majd ki kell számítani a levonásokat. Mivel ez a művelet elég terjedelmes CASE szerkezetet fog alkotni, tanácsos a többi számítást leválasztani róla.

Ennek figyelembevételével tekintsük önálló modulnak az *adóalap-számítást* (levonások), a *jövedelemadó-számítást* és ismét külön modulnak a KÖFA-számítást is magába foglaló *nettójövedelem-számítási* modult. A *kiírás* egy modulban megoldható (40. ábra). A feladatot 5 modulra bontottuk fel.

A feladat megoldása szempontjából nagyon lényeges a jövedelemadó-számítás, ezért alakítsuk ki ennek algoritmusát.

Az első lépésben meg kell vizsgálni, hogy az adóalap melyik sávba esik. Ezt egy 8-felé szétváló CASE szerkezettel lehet elvégezni. Egy intervallumba tartozást általános esetben két vizsgálattal lehet elvégezni (az értéke nagyobb az alsó határnál, kisebb a felső határnál). Amikor ilyen folyamatosan csatlakozó intervallumokba esést vizsgálunk, akkor az első intervallumnál még két, utána intervallumonként már csak egy vizsgálat is elegendő, ha a vizsgálat során vagy alulról felfelé, vagy felülről lefelé folyamatosan haladunk.

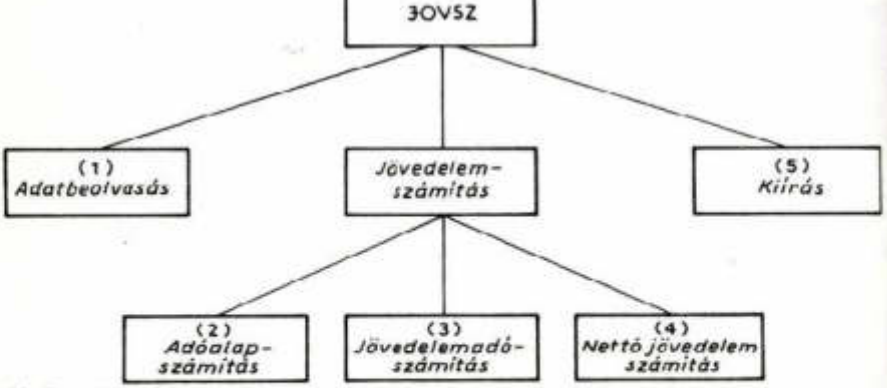
Ha a program kiválasztotta az intervallumot, akkor a következő lépésben az adóalap és az intervallum alsó határa közti különbséget kell képezni. A különbség előírás szerinti százalékához hozzá kell adni a sávra jellemző konstansot és a művelet befejeződik. Ezzel az adót kiszámítottuk, amelynek 10%-a a KÖFA. Az utolsó lépés a számított eredmények kiírása.

A feladat vezérlési szerkezetének egyszerűsége miatt tekintsünk el a vezérlőmodult alkalmazó szerkezettől.

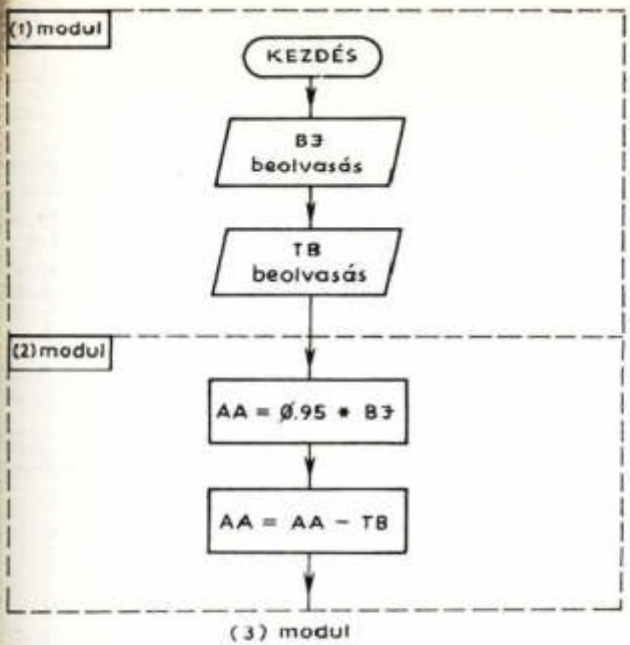
A modulok tervezése

(1) *Adatbeolvasás*

A modul **adat** típusú. A modulban két adatot (BJ és TB) kell billentyűzetről beolvasni INPUT utasítás segítségével (41. ábra).



40. ábra. A 3. feladat modul szerkezete



41. ábra. Az (1) és (2) modul folyamata

(2) *Adóalap-számítás*
A modul **eljárás** típusú. Bemeneti adatai az (1) modulban beolvasott BJ és TB értékek, kimenete pedig az adóalap (AA).

A számításkor a bruttó jövedelemből előbb levonjuk az 5% költséghányadot (0.95-tel való szorzás), majd ezt a társadalombiztosítási összeggel csökkentjük (41. ábra).

(3) *Jövedelemadó-számítás*
A modul **eljárás** típusú. Bemeneti adata az adóalap (AA) a (2) modulból. A modul kimeneti adata a jövedelemadó.

A modul kezdetén meg kell keresni, hogy az adóalap (AA) a 8 lehetséges intervallum közül melyikbe tartozik, majd az intervallumtól függően a következő képlettel lehet a jövedelemadót kiszámítani:

$$JA = K+(AA-AH)\ SZ$$

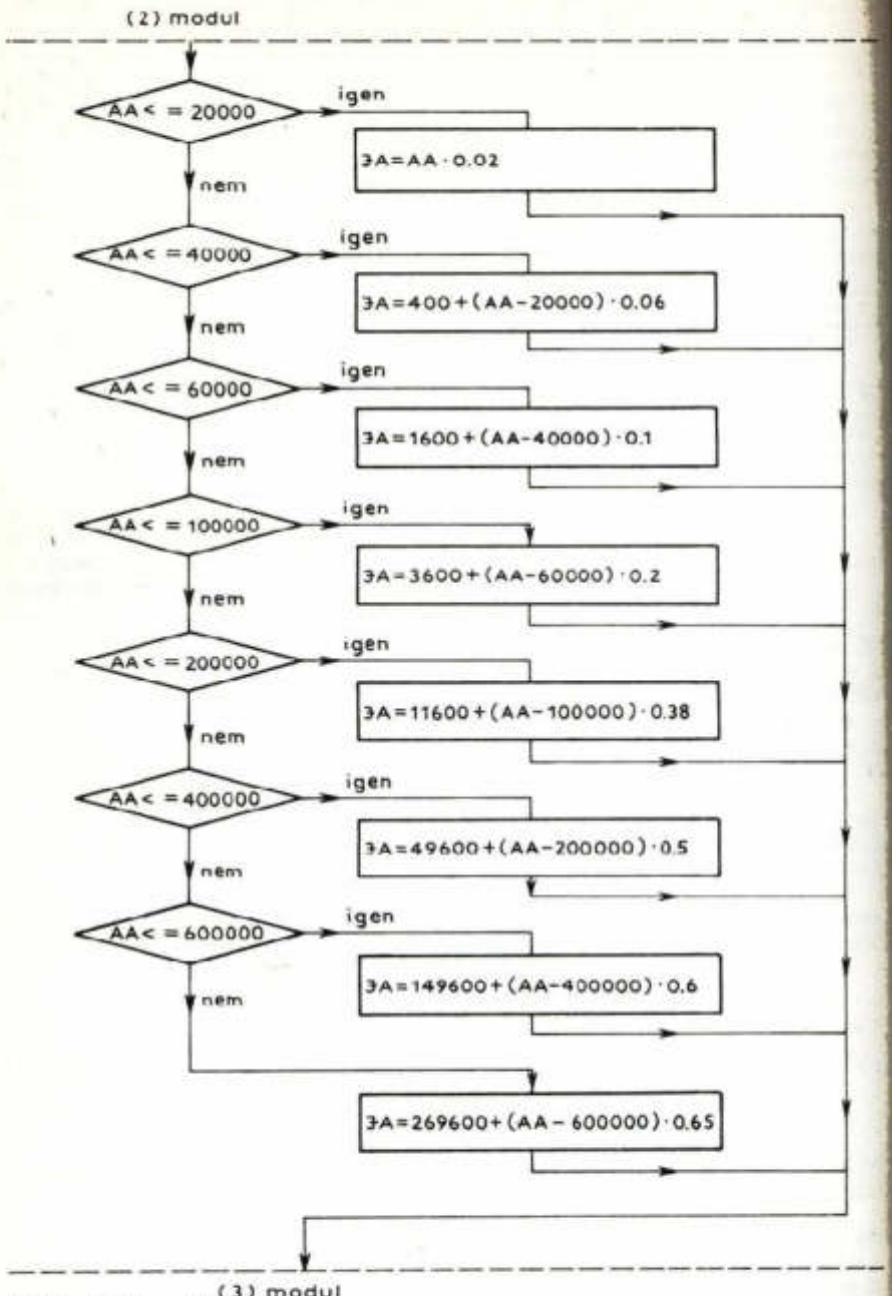
ahol

JA – jövedelemadó,
K – az intervallumhoz tartozó konstans adórész,
AH – az intervallum alsó határa,
SZ – Az adó arányos részének százalékértéke.

Az AH-t és SZ-t konstansnak tekintjük. Ezzel a modul algoritmusa már felépíthető (42. ábra).

(4) *Nettó jövedelem számítása*
A modul **eljárás** típusú. Bemenetei a bruttó jövedelem (BJ) és a levonások (TB, JA). A modulban először a KÖFA-t kell kiszámítani, amely a jövedelem 10%-a:

$$KF = 0.1 \cdot JA$$



42. ábra. A (3) modul műveletei

Majd a bruttó összegből le kell vonni valamennyi csökkentő tételt, és megkapjuk a nettó jövedelmet:

$$NJ = BJ-(TB+JA+KF)$$

Ezzel a modul terve elkészült (43. ábra).

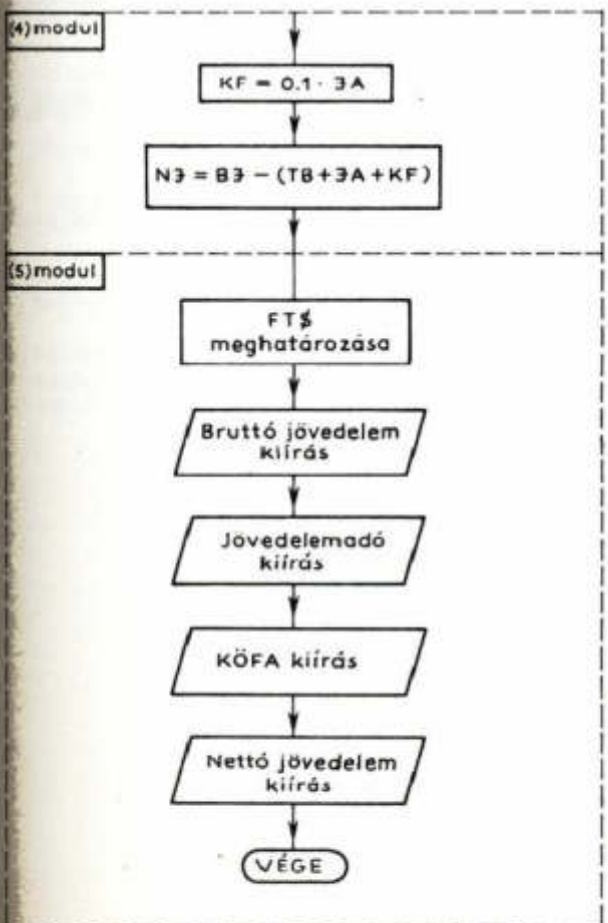
(5) *Nettó jövedelem kiírása*

A modul **eljárás** típusú. Bemenetét a kiírni kívánt adatok (BJ, JA, KF és NJ) alkotják, és ezek a kimeneti adatok is a kísérő szövegekkel.

A kiírás a korábban megismert módon elvégezhető. Hogy ne kelljen minden sorban az FT szövegrész kiíratásával konstansként foglalkozni, alakítsunk ki egy szöveges változót, amely az összeg utáni szóközt és az FT szöveget tartalmazza:

```
F$=" FT"
```

Ezzel elérjük, hogy minden sor végén ezt a szöveges változót kell kinyomtatni (43. ábra).



43. ábra. A (4) és (5) modulok folyamatai

A kódolás

Szokásunkhoz híven nem mutatjuk be a feladat teljes kódját, csupán néhány újszerű vagy kritikus kódszakaszt adunk közre.

A CASE szerkezet kódjánál először a hét IF utasítást írjuk le. A 8. felesleges, mert ha az adóalap nem esik bele a 400 001 és 600 000 forint közötti sávba, akkor a megoldás értelmében az adóalap 600 000 Ft-on felül van. Az IF utasítások leírásakor még nem tudjuk, hogy az egyes esetek kódja hová kerül, így a THEN utáni részt egyelőre üresen hagyjuk:

```
180 IF AA<=40000 THEN
```

Ezután az egyes esetekhez tartozó jövedelmadó-számítások kódolására térünk át:

```
280 REM* 2. ESET *
290 JA=400+(AA-20000)*0.06
300 GO TO
```

Látható, hogy a CASE szerkezet végére ugró utasítás ugrási címét egyelőre üresen kell hagyni. Az eset kódolása után kipótoljuk a 180. sor kódját:

```
180 IF AA<=40000 THEN 280
```

Amikor valamennyi eset kódját leírtuk, akkor a CASE szerkezet végéhez értünk, és ekkor már kiegészíthetjük a 300. sorszámú utasítást:

```
300 GO TO 480
```

Az (5) modul kódolásánál először a Ft-ot tartalmazó szöveges változónak adunk értéket:

```
540 F$=" FT"
```

Ennek felhasználásával kódolhatjuk a kiírási sorokat:

```
550 PRINT"BRUTTO JOVEDELEM    ";BJ;F$
```

Ellenőrző kérdések és feladatok

1. Mi a főág a programban?

2. A módosított 2. feladatban a *bemeneti adatok* modul a 200-as soron, az *amortizációszámítás* a 300-as soron kezdődik. Megcserélhető a két modul kódja a programban (az *amortizációszámítás* a 200-as soron, a *bemeneti adatok* a 300-as soron kezdődne)?
   Ha elvileg igen, akkor a program többi része változatlan maradhat a helyes működéshez?

3. Milyen típusú változókat ismer? Mi jellemzi ezeket?

4. Egészítsük ki a 3. feladatot úgy, hogy részfizetések esetén is helyesen működjék. Ehhez a bruttó jövedelem mellett az év során korábban kifizetett bruttó jövedelmek összegét is be kell kérni. A levonásokat és a nettó jövedelmet a teljes bruttó jövedelemből kell kiszámítani, de a korábbi levonásokat és a nettó jövedelmet le kell vonni a kapott eredményből. Készítsük el a programot!

5. Készítsen programot, amely 1 és 5 közötti számokat kér be. A beírt számnak megfelelő sorszámú sorba írjunk ki egy csillagot. Ha a felhasználó nem 1 és 5 közötti számot írt be, akkor írja ki a program:

   HIBAS ADAT; IRJA BE UJRA!

7. A SZUBRUTINOK ALKALMAZÁSA

A szubrutinok alkalmazása többször végrehajtható részfeladatok megvalósítására. A 4. feladat megoldása

Az előző részben megismert GOSUB, RETURN utasításpár nemcsak vezérlőszerkezetek kialakításához alkalmazható, hanem olyankor is, amikor egy műveletsorozatot vagy egy modult többször is végre kell hajtani. Itt nem valamilyen műveletsorozat ciklikus ismétlésére gondolunk, hanem olyan esetre, amikor ugyanazt a műveletsorozatot (modul) a program több pontján kell végrehajtani. A program készítőjében jogosan merül fel az az igény, hogy ilyenkor az ismétlődő modult csak egyszer kelljen kódolni, és a megfelelő helyekről — ahol végre kell hajtani — hívni lehessen.

A szubrutin valamilyen részfeladatot lát el a programon belül. Nem biztos, hogy a különböző hívási pontokon ugyanazokkal az adatokkal kell a feladatát végrehajtani. Ilyenkor a hívás előtt a szubrutin bemeneti változóinak aktuális értékét meg kell határozni értékadási utasításokkal. Ezt nevezzük a szubrutin **paraméterezésének.**

Mint már láttuk, egy programban több szubrutin is lehet, egy szubrutint több helyről lehet hívni, és szubrutinból is lehet szubrutint hívni. Az utóbbiakat egymásba ágyazott szubrutinnak nevezzük. Az egymásba ágyazás (egymásból hívás) szintjeinek száma nem tetszőleges. Ez a szám gépenként változik, de elég nagy ahhoz, hogy ne okozzon gondot.

4. feladat

Egy anyagot (ceruza) tartalmazó „mini" raktár készletnyilvántartását kell elvégezni. A számítógépben tárolni kell az anyag mindenkori készletét. Anyag be- és kivétel esetén ki kell írni a mozgás főbb adatait:

a) Anyagbevételezés esetén:

```
          ANYAGBEVETELEZES
A BEVETELEZETT MENNYISEG: XXX
A BEVETELEZETT ERTEK:     XXXX
```

b) Anyagkiadás esetén:

```
          ANYAGKIADAS
A KIADOTT MENNYISEG:      XXX
A KIADOTT ERTEK:          XXXX
```

A feladat elemzése

A feladat értelmében az anyag mindenkori készletét kell tárolni a számítógépben. Kis mennyiségű adatnál ez a programon belül megoldható a DATA utasítás segítségével. Ahhoz, hogy a DATA utasításban levő változó adatot változtatni lehessen, az utasítást minden programvégrehajtás után át kell írni (programmódosítás). Ezt tehát figyelmeztetni kell a felhasználót.

Alapvetően két tevékenységet kell elvégezni. Az egyik az anyagbevételezés, a másik az anyagkiadás. Mivel ezek vagylagosak, a felhasználóra kell bízni, hogy eldöntse, melyiket akarja elvégezni, vagy esetleg az egészet befejezi. Ezért a program elején egy „menüt" kell a felhasználó elé tárni, hogy a kívánt műveletet kiválaszthassa:

KESZLETNYILVANTARTAS
(1) BEVETELEZES
(2) KIADAS
(3) BEFEJEZES

Az egyes műveletek esetében a következő feladatokat kell elvégezni:

- Végre kell hajtani a mozgással kapcsolatos készletmódosítást. Kiadásnál meg kell vizsgálni, hogy megvan-e az illető anyagból a kívánt mennyiség. Ha nincs, akkor hibaüzenetet kell kiírni;
- Ki kell számítani a ki-, illetve bevételezett anyag értékét. Ehhez szükség van az anyag egységárára;
- Ki kell írni a mozgással kapcsolatos adatokat a specifikáció szerint;
- Végül a befejezés előtt fel kell szólítani a felhasználót arra, hogy írja át a készletet tartalmazó programsort;
- A bevételezés és kiadás után újra ki kell írni a menüt.

A modulok szerkezetét a 44. ábra mutatja.

Megállapíthatjuk, hogy a feladat elvégzéséhez az alábbi adatokat kell tárolni az anyaghoz:

- készletmennyiség,
- egységár.

A program tervezése

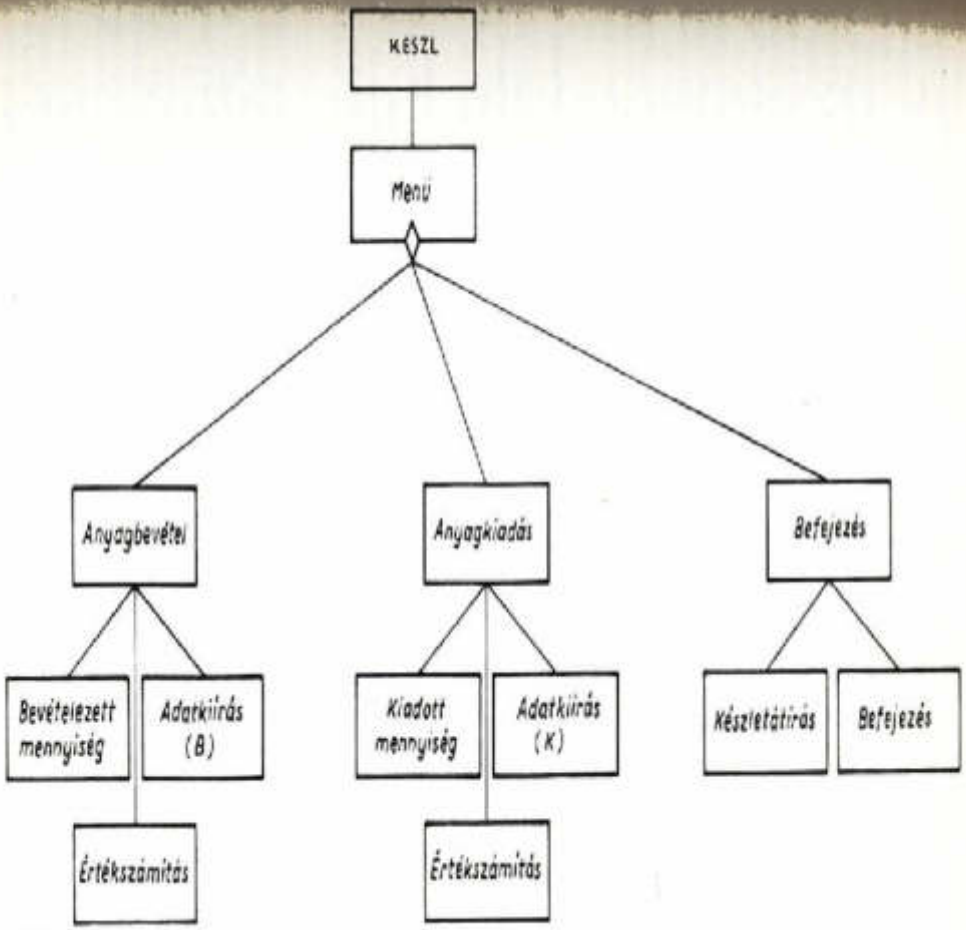
Vizsgáljuk meg a program egyes moduljait! A *menü* modul további bontása felesleges. A *bevételezett mennyiség* modul egyetlen összeadásból áll, míg a *kiadott mennyiség* modulnál előbb meg kell vizsgálni, hogy megvan-e a kivenni kívánt mennyiség az anyagból. Ha nincs, akkor hibaüzenetet kell kiírni:

ENNYI NINCS

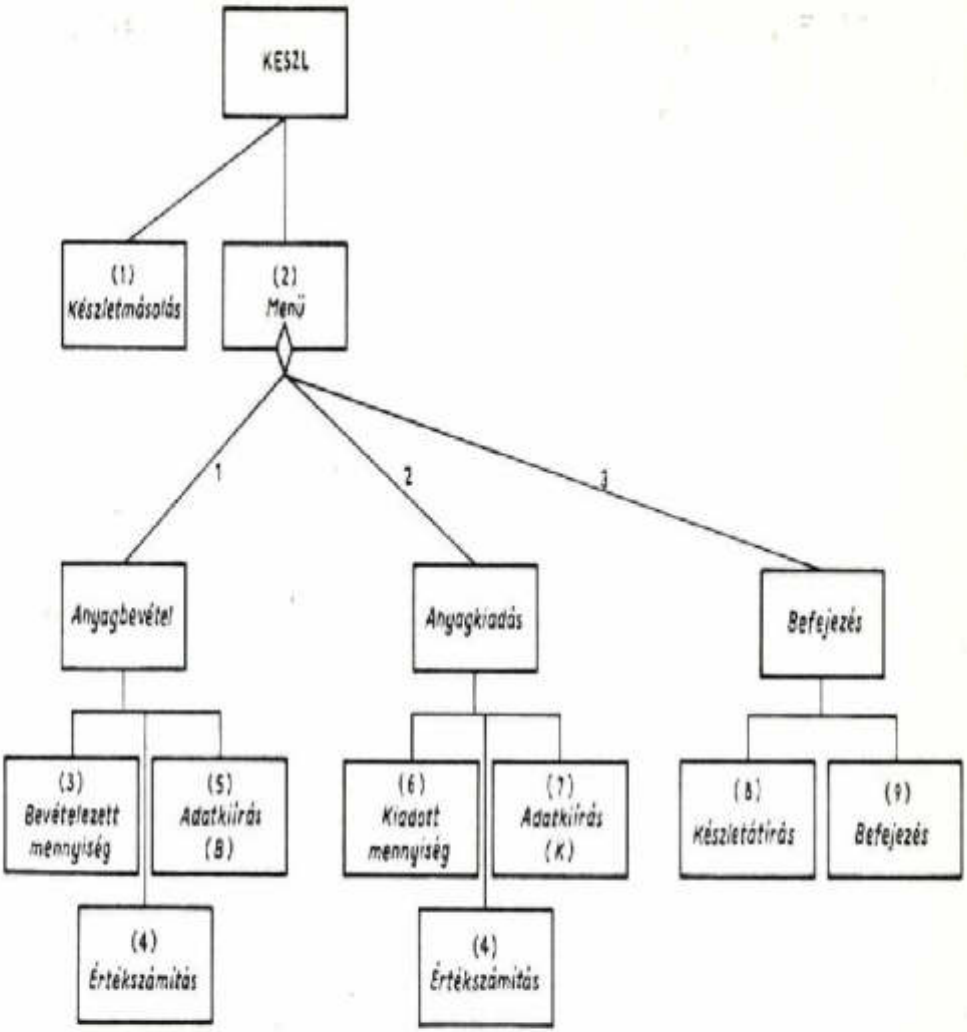
Ezután az anyagkiadási művelet elejére kell visszatérni, hogy a felhasználó módosíthassa a kiadni kívánt mennyiséget. Ezek a műveletek egy modulban elvégezhetők. A többi modult sem kell tovább bontani, mivel eléggé egyszerűek.

Mind a bevétel, mind a kiadás esetén a művelet befejeztével a *menü* modul kezdetére kell adni a vezérlést, hogy a felhasználó kiválaszthassa a következő műveletet (45. ábra).

Az adatokat DATA utasításban tároljuk. Célszerű vagy egyszerre beolvasni mind a két adatot, vagy ha egyenként olvassuk, akkor fokozottan kell ügyelni az olvasás sorrendjére, mivel az olvasó utasítások nem egymás után helyezkednek el. Az előbbi



44. ábra. A 3. feladat szerkezete



45. ábra. A 3. feladat programszerkezete

megoldást választjuk, mivel ez az egyszerűbb. Ezt a műveletet a program kezdetén kell elvégezni.

Az adatkezelésnél figyelembe kell venni, hogy a tárolt készletadatot a programfutás végén módosítani kell az aktuális értékre a ki- és bevételek miatt. A program a befejezés előtt a tényleges készletadat alapján felszólítja a felhasználót az adat átírására.

Mindkét készletmódosítási műveletnél az *értékszámítás* modul azonos, ezért ezt csak egyszer kódoljuk szubrutin formájában, és ahol végre kell hajtani, onnan hívjuk (45. ábra).

Vegyük szemügyre a modulszerkezetet! Jól felismerhető, hogy a (2) *menü* modul nem csupán a menükiírást végzi el, hanem vezérlő funkciót is ellát az anyagbevételezési, anyagkiadási és befejezési feladatok hívásával. Ezért ezt a modult vezérlőmodulnak alakítjuk ki, amely a 3 műveletcsoportot szubrutinként hívja.

Az *anyagbevételezés, anyagkiadás* és *befejezés* modulokat a tervezéskor további modulokra bontottuk, és csupán azért tartottuk meg őket, hogy az ábra jobban áttekinthető legyen. Mivel szekvenciálisan végrehajtandó modulokat vezérelnek, kódolásuktól most eltekintünk.

A feladat megoldásának egyszerűsége végett az *anyagbevételezés, anyagkiadás* és *befejezés* részfeladatait műveletenként egy-egy szubrutinban foglaljuk össze: a (3), (4) és (5) modul egy szubrutin, a (6), (4) és (7) egy másik, valamint a (8) és (9) egy szubrutin.

A modulok tervezése

(1) *Készletmásolás*

A modul **eljárás** típusú. Bemenetei a DATA utasításban tárolt két adat (készlet, egységár). Kimenete a progamban használt készletadat (K).

A modulban beolvassuk a tárolt adatokat (46. ábra).

(2) *Menü*

Vezérlő típusú modul. Bemenete a felhasználó által begépelt adat, amellyel a műveletet választja ki. Kimenete a kiválasztott modul hívása.

A modul kezdetén töröljük a képernyőt.

Ezután az üres képernyőre ki kell írni a menüt (lásd a feladatelemzésben), majd le kell olvasni a felhasználó döntését (D adat) a

MELYIKET VALASZTJA?

kérdésre. A D értéke dönti el, hogy melyik szubrutint kell hívni. A választott műveletet egy CASE szerkezettel lehet kiválasztani.

Itt kell figyelembe venni, hogy a CASE szerkezetben egy szubrutint hívunk. A szubrutin végrehajtása után a hívó utasítás utáni utasításra kerül a vezérlés. Ez pedig a CASE szerkezet értelmében a D értékének vizsgálata (46. ábra).

Ez nem okoz gondot, hiszen a szubrutin végrehajtása során D értéke nem változik meg, és így a következő vizsgálat eredménye nem lehet egy újabb szubrutinhívás. Ha például a felhasználó az anyagkiadási műveletet hívja, akkor 2-t gépel be (D=2). Ekkor a D=2 feltétel teljesül, a gép hívja az anyagkiadási szubrutint. Ennek végrehajtása után a D=3 feltétel vizsgálatára tér vissza a program. Ez a feltétel nem teljesül, vagyis a befejezést sem hajtja végre. Tehát a vezérlés helyesen fog működni.

A befejezés kivételével minden művelet után újra a menü kiírására kell visszatérni, hogy a felhasználó újabb műveletet hajthasson végre. Ezért a CASE szerkezet végén egy újabb vizsgálatot kell beiktatni, hogy előzőleg a felhasználó a befejezést kérte-e (D=3). Másik oldalról megközelítve erre azért van szükség, mert a két befejezési modul is szubrutint alkot. A szubrutin végrehajtása után a vezérlés visszakerül

a hívó utasítást követő utasításra. A modul akkor működik helyesen, ha itt egy olyan utasítás áll, amely még egyszer megvizsgálja D értékét. Ha

$$D \neq 3,$$

akkor újra ki kell írni a menüt, de ha

$$D = 3,$$

akkor a program végére kell ugrani.

Ha a második D=3 vizsgálat helyén egy GOTO utasítás állna, amely a menükiírásra vinné a vezérlést, akkor a programot sohasem lehetne befejezni, mert a CASE szerkezet után a program mindig újra kezdené a menükiadást (46. ábra).

(3) *Bevételezett mennyiség*

A modul **eljárás** típusú. Bemeneti adata a bevételezett mennyiség (BM). Kimeneti adata az új készlet (K).

A képernyő törlése után be kell olvasni a bevételezett mennyiséget. Az adatot hozzá kell adni a készlethez, hogy az új készletet megkapjuk (47. ábra).

(4) *Értékszámítás-hívás*

A modul **eljárás** típusú. Bemeneti adata az egységár (AR) és a mozgó anyagmennyiség (MA), kimenete pedig a kettő szorzata (ER).

Magát a műveletet szubrutinként kódoljuk, és mind a bevételezésnél, mind a kiadásnál használjuk, ezért az anyagmennyiség az egyik esetben a bevételezett mennyiség (BM), a másik esetben a kiadott mennyiség (KM). Hogy a modul mindkét esetben használható legyen, a mozgó anyagmennyiségnek a modul hívása előtt adunk értéket (47. ábra). Ezután a szubrutin befejeződik, és újra a (2) modul következik, amely a menüt kiírja. Ez a kiírás viszont törli a bevételezés kiírását, s a felhasználó el se tudná olvasni az eredményt, ezért a képet „le kell állítani". Csak akkor szabad folytatni a programot, ha a felhasználó már „eleget nézte" az eredményt. Ezt legegyszerűbben egy bármilyen jel beolvasásával lehet megvalósítani. (Lásd az AMOR1 programnál!) A program csak akkor folytatódik, ha a felhasználó beüt valamilyen billentyűt.

(5) *Adatkiírás* (B)

A modul **eljárás** típusú. Bemeneti adatai a bevételezett mennyiség (BM) és a bevételezett érték (ER). Ezeket az adatokat írja ki a modul (47. ábra).

(6) *Kiadott mennyiség*

A modul **eljárás** típusú. Bemeneti adata a kiírni kívánt mennyiség (KM). Kimeneti adata az új készlet (K).

A képernyő törlése után fel kell szólítani a felhasználót, hogy gépelje be a kiadni kívánt anyagmennyiséget.

Ellenőrizni kell, hogy a kívánt mennyiség kiadható-e. Ha igen, akkor a művelet a (4) modullal folytatódik, ellenkező esetben a modult újra kell hívni, hogy a felhasználó javíthasson (48. ábra).

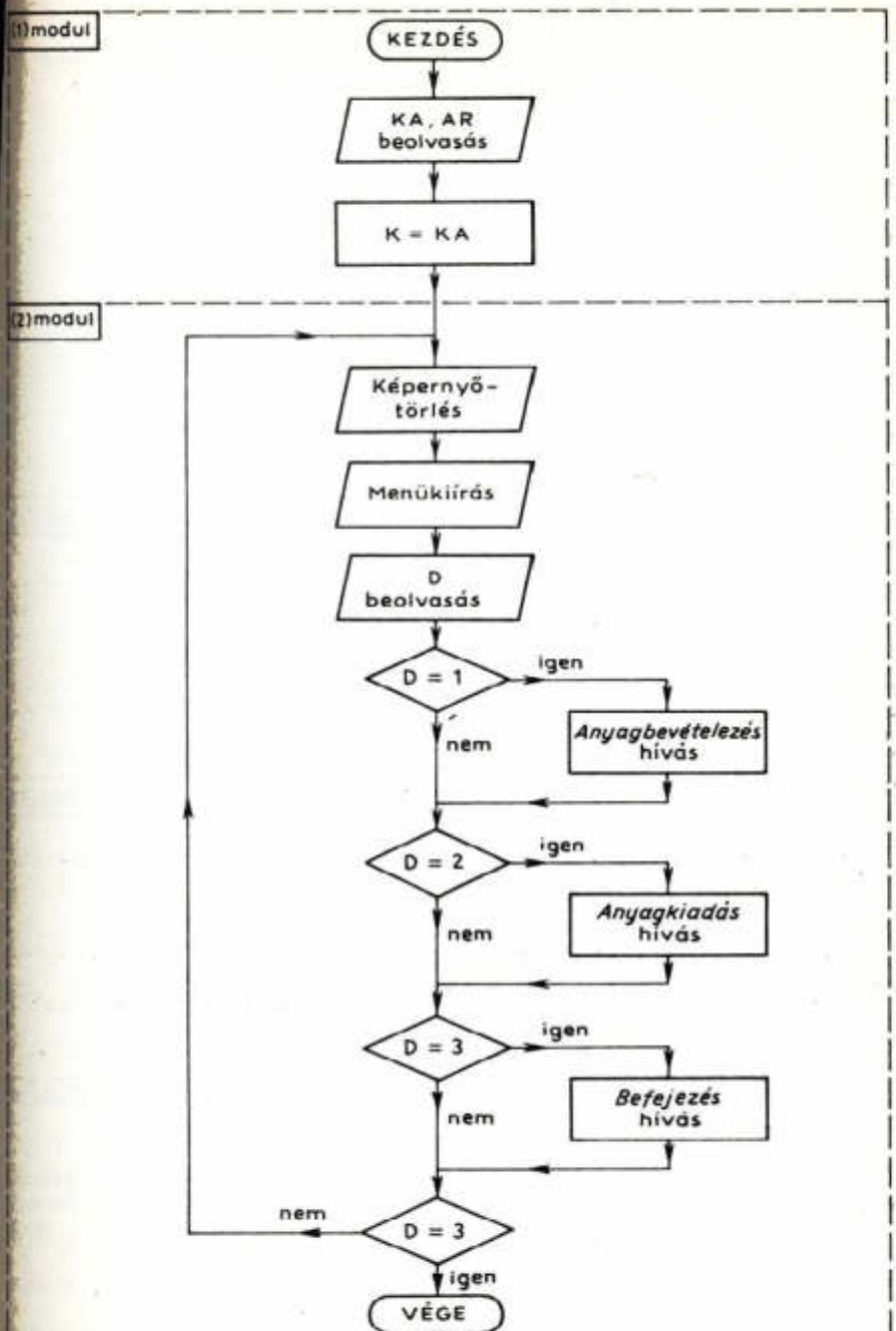
(7) *Adatkiírás* (K)

A modul **eljárás** típusú. Bemeneti adatai a KM és az ER. Ezek alkotják a modul kimenetét is.

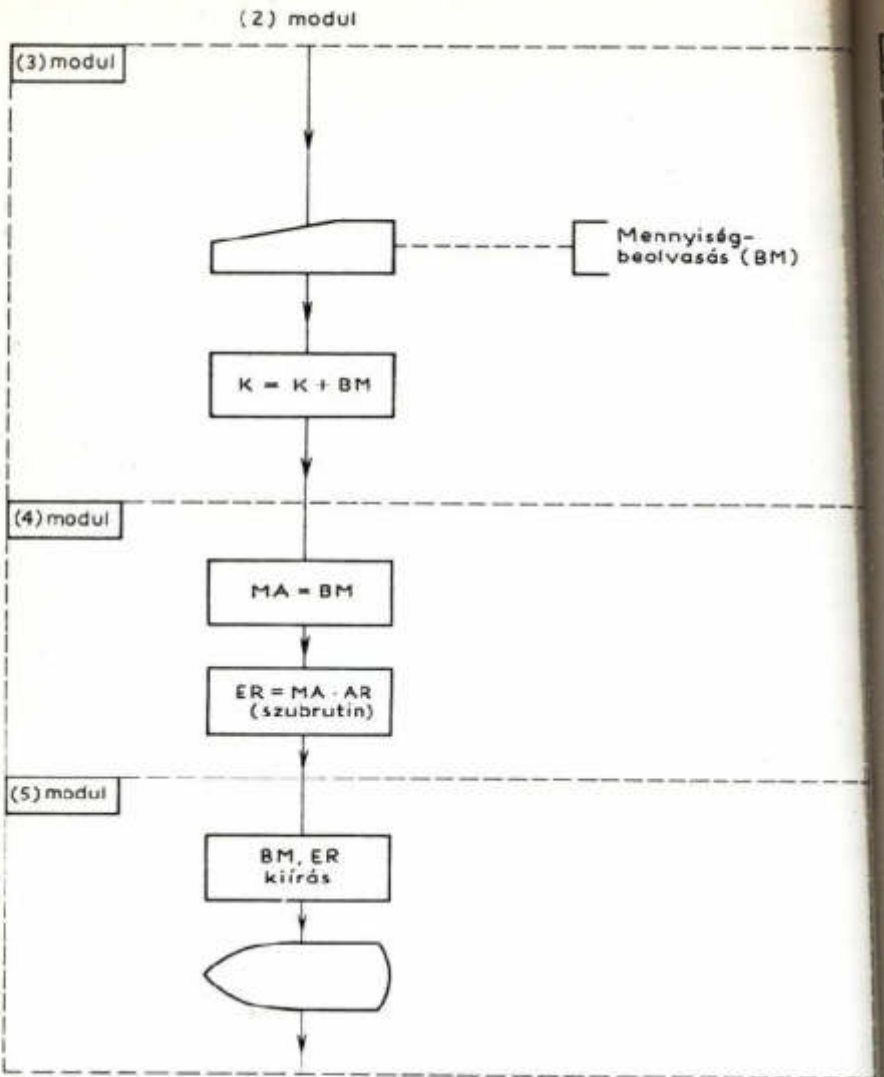
A modul az (5) modulhoz hasonlóan kiírja a kiadás adatait (48. ábra).

(8) *Készletátírás*

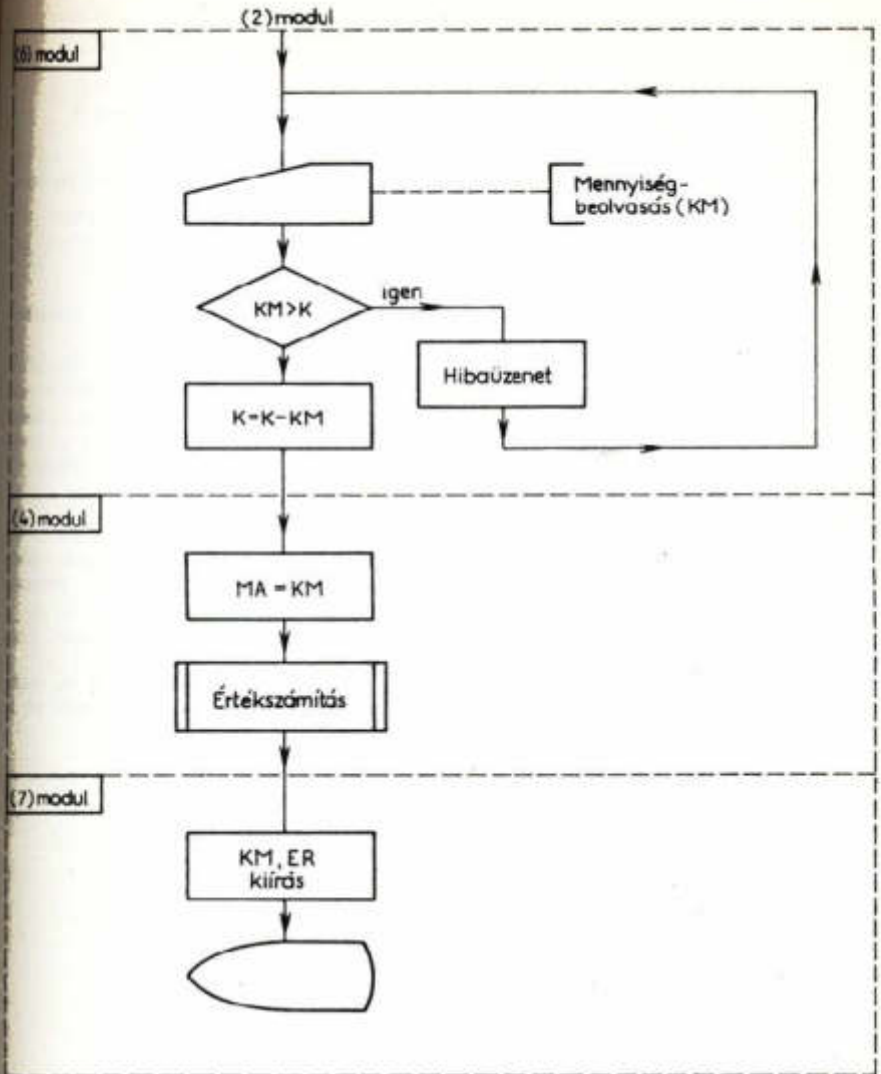
A modul **eljárás** típusú. Kimenete egyetlen emlékeztető kiírás a felhasználónak, hogy a készletet tartalmazó DATA utasítás tárgyában levő adatot az aktuálisra írja át. Ehhez természetesen ki kell írni az aktuális készletadatot is (49. ábra).



46. ábra. A (1) és (2) modul folyamata



47. ábra. A bevételi modulok folyamata

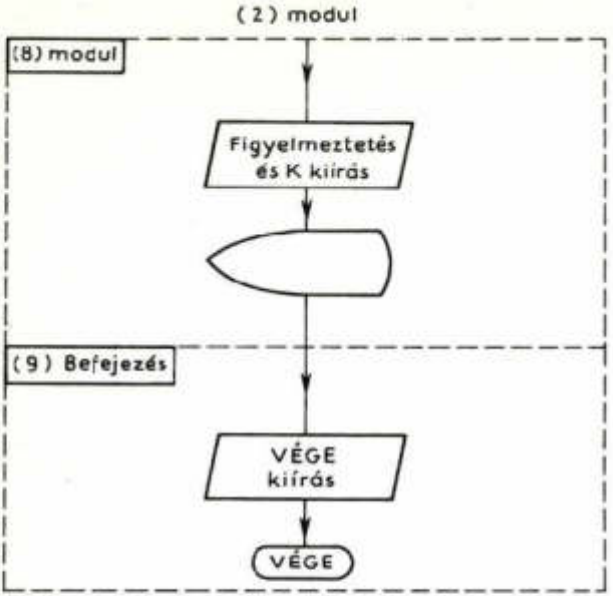


48. ábra. A kiadási modulok folyamata

(9) *Befejezés*
A modul **eljárás** típusú. A program végén kiírja, hogy a program befejeződött (49. ábra).

Kódolás

A program kódjából néhány szakaszt emelünk ki. A (2) *menü* modul kódja alább látható. a képernyőtörlő PRINT utasítást (110. sor) úgy kell begépelni, hogy a



49. ábra. A befejező modulok folyamata

PRINT után az idézőjel beírása után a SHIFT és CLR billentyűt ütjük le, majd újabb idézőjelet írunk. Ezzel a képernyő törlését programozzuk. A listán ez a törlőkarakter egy negatív szívként jelenik meg.

```
90 REM***** MENU *****
110 PRINT"♥"
120 PRINT: PRINT: PRINT: PRINT
130 PRINT"          KESZLETNYILVANTARTAS"
140 PRINT: PRINT
150 PRINT"(1) BEVETELEZES"
160 PRINT"(2) KIADAS"
170 PRINT"(3) BEFEJEZES"
180 PRINT
190 INPUT"MELYIKET VALASZTJA ";D
200 IF D=1 THEN GOSUB 250
210 IF D=2 THEN GOSUB 440
220 IF D=3 THEN GOSUB 750
230 IF D=3 THEN 960
240 GO TO 110
```

A (4) *Értékszámítás-hívás* szubrutin hívása és a szubrutin formája:

(anyagmennyiség
meghatározása)

A szubrutin:

```
900 REM***** ERTEKSZAM. SZUBRUTIN *****
910 ER=MA*AR
920 RETURN
```

Az *adatkiírási* modulok végén le kell állítani a gépet egy tetszés szerinti adat (F$) beolvasásával:

```
420 INPUT"HA BEFEJEZTE, NYOMJON LE EGY GOMBOT !";F$
430 RETURN
```

Értékelés

A feladat megoldásában alkalmazott programmódosítás (a DATA utasítás átírása) nem a legszerencsésebb módszer. Különösen, ha azt is figyelembe vesszük, hogy ezután a kazettán vagy a lemezen tárolt programot az újjal ki kell cserélni. Ez kazetta esetén a tárolt program újra kimásolásával végezhető el, a Commodore lemezegységén pedig a

SAVE" @ 0:*név*",8

paranccsal, ahol a *név* a program neve (a nevet nem kell változtatni). A parancs hatására a program előző változata törlődik, és csak a módosított változat kerül a lemezre.

A 13. és 14. részben megismerünk egy olyan megoldást, amellyel az ilyen feladat programmódosítás nélkül elvégezhető.

8. CIKLIKUS TEVÉKENYSÉGEK I.

Különböző ciklikus szerkezetek megvalósítása a programban. Néhány függvény. Az 5. feladat megoldása

A CIKLUS

Tegyük fel, hogy feladatunk egy raktárban levő anyagok értékének kiszámítása számítógépes program segítségével. Az anyagok értékét úgy lehet megkapni, ha az egyes anyagok egységárait megszorozzuk a mennyiséggel:

$$E=A\cdot M$$

ahol

E – egy anyag értéke,
A – az egységár,
M – a mennyiség.

Egy raktárban többféle anyag van, és az összértéket úgy kapjuk meg, hogy a fenti szorzatot minden anyagra kiszámítjuk:

$$E_1 = A_1 \cdot M_1$$
$$E_2 = A_2 \cdot M_2$$

Végül az E_i értékeket össze kell adni, és megkapjuk az összes értéket:

$$F=E_1+E_2+E_3+\ldots\ldots\ldots+E_n$$

Ha most ezt a feladatot egy programmal kívánjuk megoldani, akkor elvileg azt kellene csinálni, hogy az értékszámító szorzatokat annyiszor írjuk be a programba, ahányszor végre kell hajtani:

```
100  E1=A1*M1
110  E2=A2*M2
```

Nyilvánvaló, hogy ez nehézkes, ráadásul megvalósíthatatlan is. Képzeljük el ezt a megoldást egy 1000-féle anyagot tartalmazó raktárban! Ennyi utasítást a kisebb gépekbe be sem lehet írni. Ennyi változónevet sem tudunk adni. Azonfelül az anyagok is változhatnak (valamilyen anyag megszűnik, új anyagok jönnek), és a programot minden anyagféleség változása után módosítani kellene.

Észre lehet venni a szabályosságot a példákban. Az elvégzendő művelet mindig ugyanaz. Tehát az elvégzendő műveletet csak egyszer kódol-

juk, és minden alkalommal más adatokkal (a következő anyag) kell végrehajtani. Lépésekre bontva a következőt kapjuk:

1. adatmeghatározás (következő anyag A, M)
2. értékszámítás (E = A·M)

A 2. lépés után ismét az első, majd a következő anyag adataival a 2. lépés, utána megint az 1., és így tovább következik. A műveletet ciklikusan ismételve hajtja végre a program, ezért az ilyen szerkezetet **ciklus**nak nevezzük.

Az összértéket az egyes anyagok értékének kiszámítása után kell meghatározni. Ilyenkor minden anyag értékét meg kellene jegyezni egy egy változóban, és az egyedi értékszámítás után ezeket kell összegezni. Nem lehetne ezt valahogy egyszerűbben meghatározni? Az összegezést lépésenként is végre lehet hajtani, vagyis az F értékéhez minden értékszámítás után hozzáadjuk az aktuális értéket (E). Ennek eredményeként az összeg is rendelkezésre fog állni. Bővítsük ki ezzel a lépéssel a kiszámítás menetét:

1. adatmeghatározás
2. értékszámítás
3. az érték hozzáadása az összeghez (F+E)

Könnyen észrevehető, hogy az értékösszegezés csak akkor működik helyesen, ha a művelet elején F értéke nulla. Ellenkező esetben az induló érték hozzáadódna az összes értékhez. A műveletek folyamatát az 50. ábra mutatja.

Jól látható a ciklus előnye: ha egy műveletsort különböző adatokkal akarunk végrehajtani, akkor csak egyszer kell kódolni és minden adattal végrehajtani.

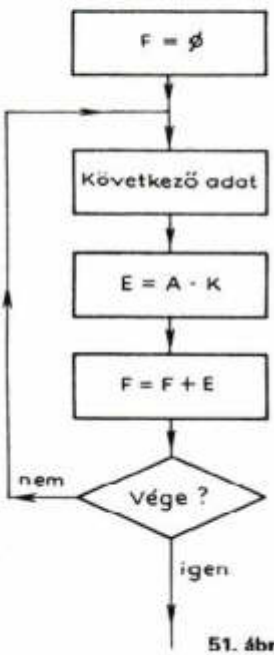
50. ábra. Raktárban lévő anyagok értékszámítása

Meddig kell a ciklust ismételni? Addig, amíg van adat, amellyel a műveletet el kell végezni. Ezért a ciklusban valahol meg kell vizsgálni, hogy van-e még adat, amellyel a műveletet végre kell hajtani. A vizsgálat a ciklus kezdetén vagy végén állhat (51. ábra). Akárhol is legyen a vizsgálat, a ciklusból tovább kell menni (be kell fejezni), ha a vizsgálat eredménye ezt mutatja.

A ciklus kezdete előtt van egy művelet (pl. F=0), amelyet csak egyszer kell ugyan végrehajtani, de a ciklushoz kapcsolódik. Az ilyen műveletek alkotják a **ciklus előkészítését**. Magát a műveletsort, amelyet ismételten végre kell hajtani, **ciklusmagnak** nevezzük. A ciklus folytatásával vagy befejezésével kapcsolatos vizsgálatot **ciklusfeltétel**-vizsgálatnak nevezzük. Az az adat, amelynek értéke meghatározza a ciklus folytatását vagy befejezését, **a ciklus változója.** Példánkban az anyagszám a ciklusváltozó. A ciklus addig kell ismételni, amíg minden anyagot sorra nem vettünk.

Összefoglalva, egy ciklus három részből áll:

- előkészítés,
- ciklusmag, amely magában foglalja
  - a ciklikusan végrehajtott műveletet, és ezzel együtt vagy ettől függetlenül,
  - a ciklusváltozó értékének módosítását,
- ciklusfeltétel-vizsgálat, vagyis a ciklusváltozó értékének vizsgálata.



51. ábra. A ciklus műveletei

A ciklusszerkezetet kétféle algoritmusban valósíthatjuk meg, attól függően, hogy hol helyezkedik a ciklusfeltétel-vizsgálat:

Az **elöl tesztelő típusú ciklusszerkezet:** Mint a nevében is benne van, az elöl tesztelő ciklusban a feltételvizsgálat a ciklusmag végrehajtása előtt történik meg (52. ábra). A ciklusmagot mindaddig végre kell hajtani, amíg a **végrehajtási feltétel fennáll.** Ha a feltétel már nem teljesül, akkor a ciklus befejeződik, és a vezérlés a ciklus után következő utasításra kerül.

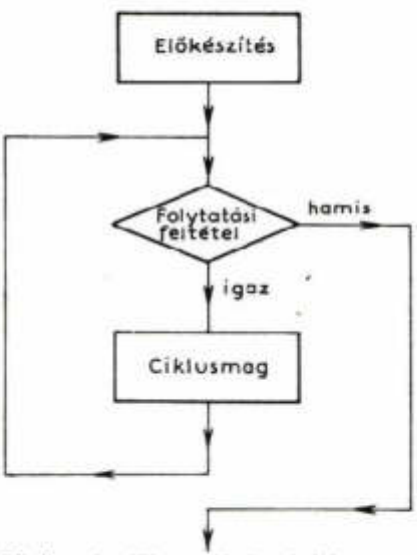
A **hátul tesztelő típusú ciklusszerkezet:** A feltételvizsgálat a ciklusmag után megy végbe (53. ábra). Azt kell vizsgálni, hogy a **befejezés feltétele fennáll-e.** Ha ez még nem áll fenn, akkor a ciklust folytatni kell. Ha a befejezés feltétele fennáll, akkor a ciklus befejeződött.

A két szerkezetet összehasonlítva látható, hogy a feltételvizsgálat helye és a vizsgálat jellege eltér egymástól. Az elöl tesztelő szerkezetben a **folytatás feltételét kell vizsgálni,** a hátul tesztelő szerkezetekben a **befejezés feltételét kell vizsgálni.**

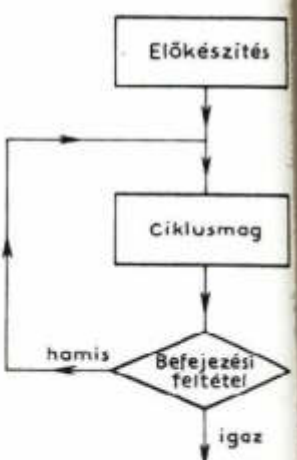
Többnyire bármelyik szerkezetet lehet használni. Vannak azonban olyan feladatok, ahol vagy csak az egyiket, vagy csak a másikat lehet használni.

Ezek a tiszta esetek néhányszor nem alkalmazhatók, mert arra kényszerülünk, hogy a ciklusfeltétel-vizsgálatot sem a ciklus elején, sem a végén, hanem valahol a közepén helyezzük el.

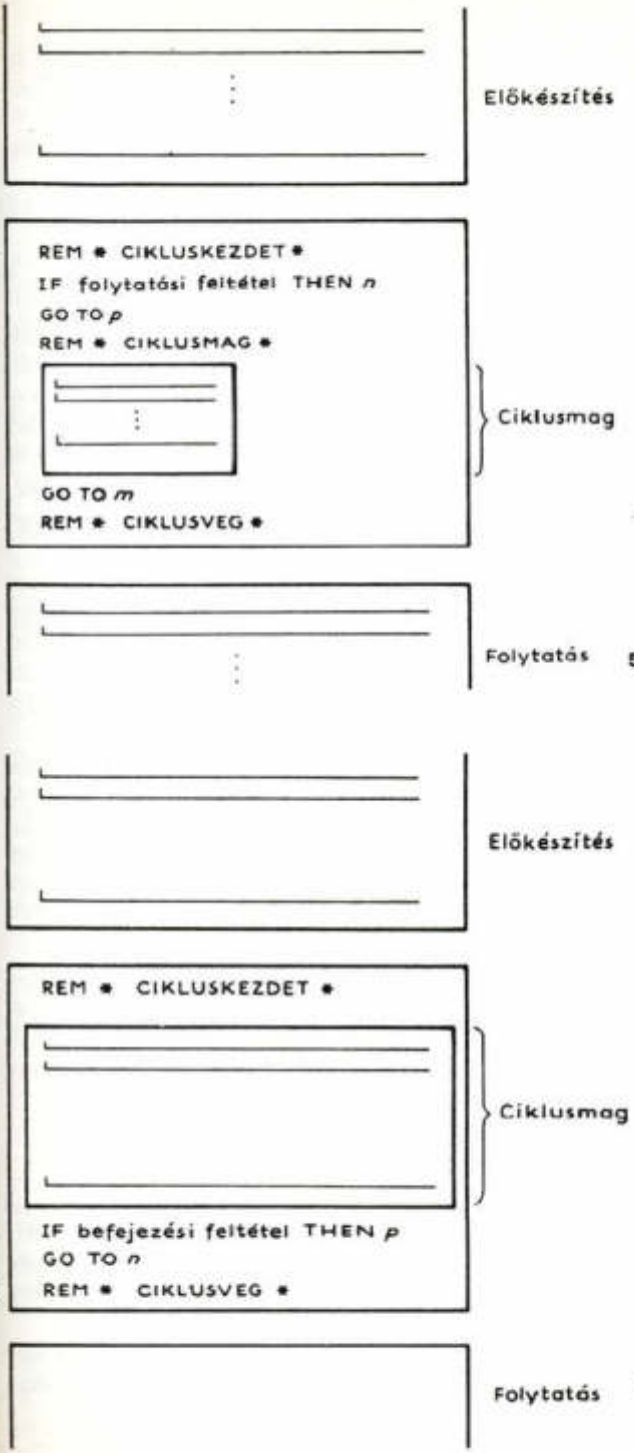
A ciklikus tevékenységek kódolásához is érdemes a ciklus szerkezetét kifejező kódolási formát alkalmazni. Az **elöl tesztelő** ciklusszerkezet kódolási formáját az 54. ábra mutatja. A **hátul tesztelő** ciklusszerkezet kódját az 55. ábrán láthatjuk.



52. ábra. Az elöl tesztelő típusú ciklusszerkezet



53. ábra. A hátul tesztelő típusú ciklusszerkezet



54. ábra. Az elöl tesztelő ciklus kódolási formája

55. ábra. A hátul tesztelő ciklus kódolási formája

A bemutatott két szerkezeten kívül a BASIC-ben van kifejezetten ciklusutasítás: ez a FOR, STEP, NEXT utasításhármas, amelyet azonban röviden csak FOR, NEXT utasításpárnak hívunk.

Nézzük meg, hogy a FOR és a NEXT utasításpárral hogyan lehet ciklust kialakítani.

A FOR, NEXT UTASÍTÁSPÁR

Ha a ciklusmagban van olyan változó, amelynek értéke minden ciklusmag végrehajtása során egyenletesen (lineárisan) változik (nő vagy csökken), és értéke alapjána ciklus befejezése eldönthető, akkor a ciklus kódolásához a FOR, NEXT utasításpár felhasználható. Az említett változó pedig a ciklusváltozó lesz. A FOR, NEXT utasításpárral felépített ciklusban azt kell meghatározni, hogy a ciklusváltozó milyen kezdőértékétől, milyen lépésenként haladva, milyen végfeltételértékig kell a ciklust végrehajtani. A program a ciklusmagot mindaddig végrehajtja, amíg a ciklusváltozó értéke meg nem haladja a záróértéket, ha a ciklusváltozó értéke növekszik, illetve ha a ciklusváltozó értéke csökken, akkor addig, amíg a ciklusváltozó a záróérték alá nem csökken. A ciklusváltozó értékének módosítását a program minden ciklusmag végrehajtásakor automatikusan elvégzi, és ha a folytatási feltétel nem teljesül, a gép a ciklust befejezi.

A FOR, NEXT utasításpárral kialakított ciklust **számlált menetű ciklusnak** is nevezzük.

A FOR utasítás vezeti be a ciklust. Definiálja a ciklusváltozót, valamint ennek kezdő- és záróértékét. A FOR utasítás formája:

x FOR *ciklusváltozó* = 1. *kifejezés* TO 2. *kifejezés* STEP ± 3. *kifejezés*

ahol

1. *kifejezés* – a ciklusváltozó indulóértéke,
2. *kifejezés* – a ciklusváltozó záróértéke,
STEP – a növekménymeghatározás kulcsszava,
3. *kifejezés* – növekmény.

A ciklusmagot a

NEXT *ciklusváltozó*

utasítás zárja le. Hatására a program visszaadja a vezérlést a FOR utasításnak, amely kiszámítja a ciklusváltozó következő értékét. A ciklusmag utolsó végrehajtása után a vezérlés a NEXT utasítás utáni utasításra kerül át.

A FOR, NEXT utasításokból felépített ciklus befejezésekor a ciklusváltozó értéke azt az értéket veszi fel, amellyel az utolsó ciklusmag végrehajtódott.

Ha a növekmény 1, akkor az utasításból a 2. kifejezés utáni rész (STEP . . .) elhagyható. A Sinclair-gépeken a ciklusváltozó csak egyetlen betű lehet.

FÜGGVÉNYEK

Tegyük fel, hogy egy feladaton belül van egy olyan részfeladat, amelyben meg kell határozni egy A változó abszolút értékét. Közismert, hogy az abszolút érték mindig nem negatív szám:

$$|A| \geq 0$$

Ezt ki lehet számítani az 56. ábrán látható algoritmussal, illetve az alábbi utasítással:

```
200 IF A<0  THEN A = -A
```

Mindezt el is hagyhatjuk, mivel a BASIC-ben van olyan **függvény**, amellyel az abszolút értéket közvetlenül meghatározhatjuk. Ez az ABS függvény. Az abszolút érték kiszámításához a bemutatott példában a következőképpen lehet felhasználni:

```
200  A = ABS(A)
```

- Sinclair-gépeknél: `200 A = ABS  A`

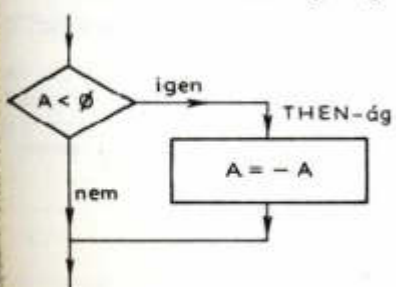
A BASIC nyelvben több ilyen „beépített" függvény van, amelyet a programozó programíráskor felhasználhat.

A **függvények** valamilyen szabály szerint egy független változó értékből (argumentumból) egy eredményértéket számítanak ki. Általános formában felírva:

Eredmény = *függvény* (argumentum)

Az eredmény és az argumentum mindig valamilyen konkrét számérték, a *függvény* pedig a kiszámítás módját határozza meg. Például az ABS függvény az argumentumból mindig az argumentum számértékének megfelelő pozitív eredményt állít elő.

De van ennél több segítséget nyújtó függvény is. Például a SIN függ-



56. ábra. Abszolút érték meghatározása

vény, amely az argumentum szinuszát számítja ki (az argumentumot ívmértékben feltételezi a gép):

```
200 A = SIN (B)
```

- Sinclair-gépeknél: `200 A = SIN B`

Az A (eredmény) értéke a B szinusza lesz. A függelék tartalmazza valamennyi függvényt. Számunkra azért előnyös a függvények használata, mert lehetővé teszik, hogy valamilyen eredményt egyetlen utasítással számítsunk ki ahelyett, hogy több lépéses műveletet végeznénk el az érték kiszámításához. Van olyan függvény is, amelyet más BASIC utasításokkal egyáltalán nem vagy csak roppant erőfeszítések árán tudunk pótolni.

A meglevő függvényeken kívül a felhasználó is definiálhat függvényeket az 5 alapműveletből és a BASIC könyvtári függvényeiből. Ezek a **felhasználói függvények.**

A felhasználói függvényeket használatuk előtt definiálni kell a programban az alábbi utasítással:

x DEF FN*név (változó) = kifejezés*

ahol

DEF	— a függvénydefiniáló utasítás kulcsszava,
FN*név*	— a felhasználói függvény neve, amelyben a név azonosítja a függvényt (2 karakter lehet, az első mindig betű),
változó	— a függvény független változója,
kifejezés	— azokat a műveleteket tartalmazza, amelyek segítségével a függvény értékét ki lehet számítani a független változóból.

- A HT és ▲ a PRIMO gépen ez a lehetőség nincs meg.
- A Sinclair-gépeknél a *név* egy betű lehet.

Tekintsünk most egy részfeladatot, amelynek értelmében egy valós számot kerekíteni kell! A részfeladat viszonylag egyszerűen megoldható eddigi ismereteink segítségével, de még jobban egyszerűsíthetjük, ha egy függvényt is felhasználunk a megoldásban.

Az INT függvény segítségével vegyes számokból le lehet „választani" a törtrészt. A függvény formája:

x eredmény = INT *(argumentum)*

A függvény eredménye az a legnagyobb egész szám (vagy zérus), amely nem nagyobb a kifejezésnél. Például:

```
INT (16.25) = 16
INT (0.47)  = 0
INT (–4.3)  = –5
```

- A Sinclair-gépeknél az argumentumot nem kell zárójelbe tenni:

x eredmény = INT *argumentum*

A függvény jól használható pozitív számok kerekítésére. Előtte azonban úgy kell alakítani, hogy a törtrész leválasztása után a kerekítésnek megfelelő értéket kapjuk meg. Ezt úgy érhetjük el, hogy a kerekítendő számhoz hozzáadunk 0,5-öt:

A+0,5

Ha az A törtrésze 0,5 volt vagy annál nagyobb, akkor az A egész része az összeadás következtében eggyel nagyobb lett. Ha ennek vesszük az egész részét, akkor az eredeti szám kerekített összegét kapjuk. Például:

A = 1.53
A = 1.53+0.5 = 2.03
INT(A) = 2

Ha az A törtrésze 0,5-nél kisebb volt, akkor az összeadás után az A egész része változatlan marad, és a törtrész leválasztása után a kerekített számot kapjuk. Például:

A = 2.27
A = 2.27+0.5 = 2.77
INT(A) = 2

A kerekítés elvégzésére definiálhatunk egy függvényt is. Ha a fenti A változó értékét kell kerekíteni, akkor ehhez például egy K nevű függvényt definiálunk:

100 DEF FN K (A) = INT(A+0,5)

Hatására a gép „ismerni fogja" az FN K felhasználói függvényt. Ha bárhol a programban kerekíteni akarjuk az A értékét, akkor az FN K függvénnyel meg lehet határozni a kerekített értéket:

500 B = FN K (A)

B értéke az A kerekített értéke lesz, amit a gép a függvénydefiniálásban megadott kifejezés szerint számít ki.

5. feladat

Ki kell írni egy egyéves lejáratú áruvásárlási kölcsön havi részletfizetési összegét, a hónapokra esedékes kamatot, az alapösszegből havonta visszafizetett összeget és a kölcsön maradék összegét a következő táblázatos formában:

RESZLET OSSZEG	KAMAT OSSZEG	VISSZA FIZ. OSSZEG	KOLCSON OSSZEG
:	:	:	:
XXXXX	*XXXXX*	*XXXXX*	*XXXXX*

Az összesítő eredményeket egy sor kihagyással a 14. sorba kell írni.

A feladat elemzése

A kamatos kölcsönök egyenlő részletben való visszafizetésekor az egyes részletek összegét a következő képlettel lehet meghatározni:

$$R = F \frac{K}{1 - \left(\frac{1}{1+K}\right)^T}$$

ahol

R – a részlet összege,
F – a felvett kölcsön összege,
K – a kamatláb (tizedesszám formájában),
T – a kölcsön visszafizetésének tartama években.

Mivel a feladatban egy 1 évre kapott kölcsönt kell havi részletekben törleszteni, a kifejezést módosítani szükséges. A kamat éves szinten értendő, ezért havi törlesztésre az 1/12-ét kell figyelembe venni. A tartam (T) évekre vonatkozik, tehát meg kell szorozni 12-vel, hogy a részlet havi összeg legyen. Ennek megfelelően a felhasznált képlet:

$$R = F \frac{\frac{K}{12}}{1 - \left(\frac{1}{1+\frac{K}{12}}\right)^{12T}}$$

Megjegyezzük, hogy negatív F vagy K esetén az eredmény is negatív lesz, ami nyilván értelmetlen a feladat szempontjából. Ezért csak pozitív F-et és K-t szabad elfogadni.

Havi törlesztés esetén az adott hónapra esedékes kamat összegét az előző havi maradék összeg alapján kell kiszámítani:

$$HK = F \cdot \frac{K}{12}$$

ahol

HK – a havi kamat összege,
F – az előző havi maradék kölcsön.

A kölcsön összege a havi részlet és a havi kamatösszeg különbségével csökken:

$$CK = F - HK$$

ahol

CK – a kölcsönből az adott hónapban visszafizetett összeg.

Ez nyilvánvaló, hiszen a kamat a kölcsönösszegen felüli összeg, így a kamat fizetése nem csökkentheti a kölcsönösszeget.

A maradék kölcsönösszeg tehát a tárgyhó végén:

$$FJ = FM - CK$$

ahol

FJ – a jelenlegi kölcsönösszeg (maradék),
FM – a múlt havi maradék kölcsönösszeg.

A fentiekből látható, hogy a havi részlet (F) kivételével mind a havi kamatösszeg (HK), mind a havi kölcsön-visszafizetés összege (CK) minden hónapban az előző havi maradék kölcsönösszegből származik.

A feladat elején meg kell ismerni a kölcsön adatait, a kölcsön összegét és a kamatot. A program ezeknek az adatoknak a beolvasásával fog kezdődni. Ezután ki kell számítani a havi részlet összegét, és rátérhetünk a táblázat adatainak kiszámítására és kiírására. Mint láthattuk, a havi változó adatok (HK, CK, FJ) az előző havi maradék kölcsönösszegből (FM) kiszámíthatók – mindig azonos kifejezések segítségével –, ezért ezt a műveletsort csak egyszer kell kódolni a programban, és annyiszor végrehajtani, ahányszor erre szükség van (12-szer). A művelet tehát ciklust fog alkotni.

A havi részlet-, a havi kamat- és a havi visszafizetés-összegeket a havi számítások eredményével meg kell növelni, és a havi adatok után ki is kell írni.

A feladat modulszerkezetét az 57. ábra mutatja.

A program tervezése

Tekintsük át az egyes funkciókat részleteiben! Az adatbeolvasás és a havi részlet kiszámítása zárt, egyszerű feladat, ezek önálló modulként megmaradhatnak.

Az (1) modulban az adatbeolvasást ellenőrzéssel kell kiegészíteni. Csak akkor szabad elfogadni a begépelt adatokat, ha pozitívak. Negatív adat esetén

CSAK POZITIV LEHET!

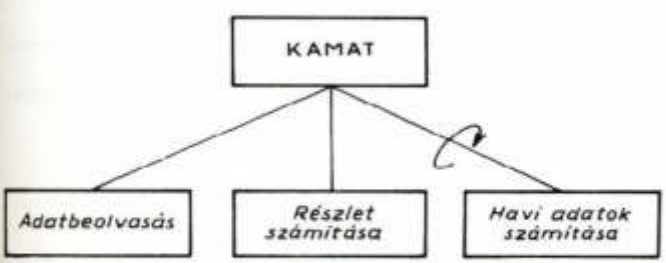
hibaüzenetet kell kiírni, és meg kell ismételni a beolvasást.

Alaposabb tervezést igényel azonban a havi adatok kiszámítása. Ez a funkció a következő műveleteket foglalja magában:

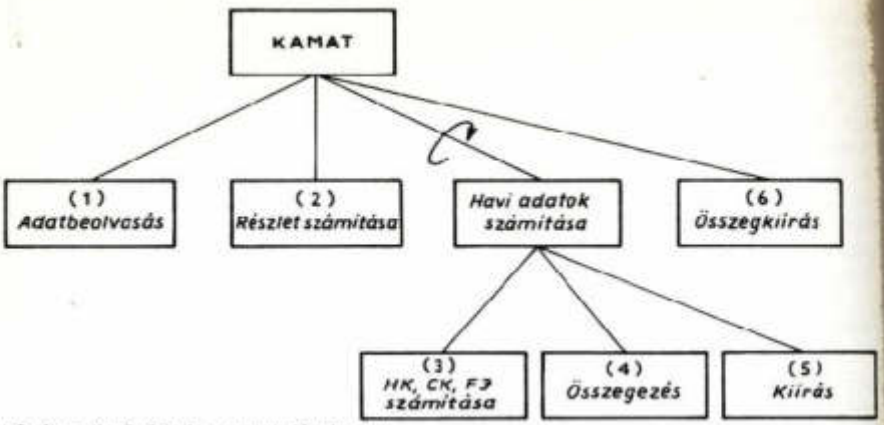
- a havi adatok (HK, CK, FJ) kiszámítása,
- az összegezések elvégzése,
- az adatok kiírása.

Ezt a modult tehát tanácsos három modulra tovább bontani (58. ábra). A (3), (4) és (5) modult annyiszor kell végrehajtani, ahány hónap van egy évben. Ez a három modul együtt egy **ciklus magját** alkotja. A **ciklusváltozó** a hónapok száma, a **ciklusfeltétel** pedig a 12. hónap adatainak feldolgozása. Ezután ugyanis a művelet véget ér. Ezt a ciklust a FOR, NEXT utasításpárral jól meg lehet valósítani. A ciklusműveletek számát a H (hónap) változó értéke szabja meg. A (3) modul számításait az elemzésnél megismertük.

A (4) összegezéseit az ún. „gyűjtő" módszerrel végezzük el. Minden összeghez kijelölünk egy változót, induló tartalmát nullázzuk (bár ez a BASIC-ben felesleges), és minden számítási művelet után a kapott eredményeket hozzáadjuk a megfelelő



57. ábra. Az 5. feladat szerkezete



58. ábra. Az 5. feladat programjának terve

változó értékéhez. Ezzel az összegek a ciklus befejezésére készen állnak a megfelelő gyűjtőkben.

Az (5) és (6) modul kiírásait a PRINT utasítással fogjuk megvalósítani. A (6) modult csak egyszer kell végrehajtani, és ezután a program befejeződik.

A kiírás nem tartalmazza a bemeneti adatokat, ezért csak a program elején célszerű a képernyőt törölni, a táblázat kiírása előtt nem, hogy a bemeneti adatok is láthatók legyenek.

A programban a ciklus vezérléséhez nem célszerű vezérlőmodult alkalmazni, mivel a ciklust FOR, NEXT utasításpárral kódoljuk. A FOR, NEXT utasításpár önmaga gondoskodik a ciklus vezérléséről. A cikluson kívüli 3 modul szekvenciális végrehajtása szintén megoldható vezérlőmodul nélkül.

A modulok tervezése

(1) *Adatbeolvasás*
A modul **adat** típusú. Bemeneti és kimeneti adatai a beolvasott és ellenőrzött F és K adatok.

A modulban kell beolvasni a kölcsönösszeget (F) és a kamatot (K). Mindkét adatot ellenőrizni kell, hogy pozitívak-e. Negatív vagy 0 bemenet esetén hibaüzenetet kell kiadni, és a beolvasást meg kell ismételni (59. ábra). A kamat összegét (K) el kell osztani 100-zal, hogy a számításhoz megfelelő értéket kapjunk. Megjegyezzük, hogy mindkét adatbeolvasás hátul tesztelő ciklusban megy végbe.

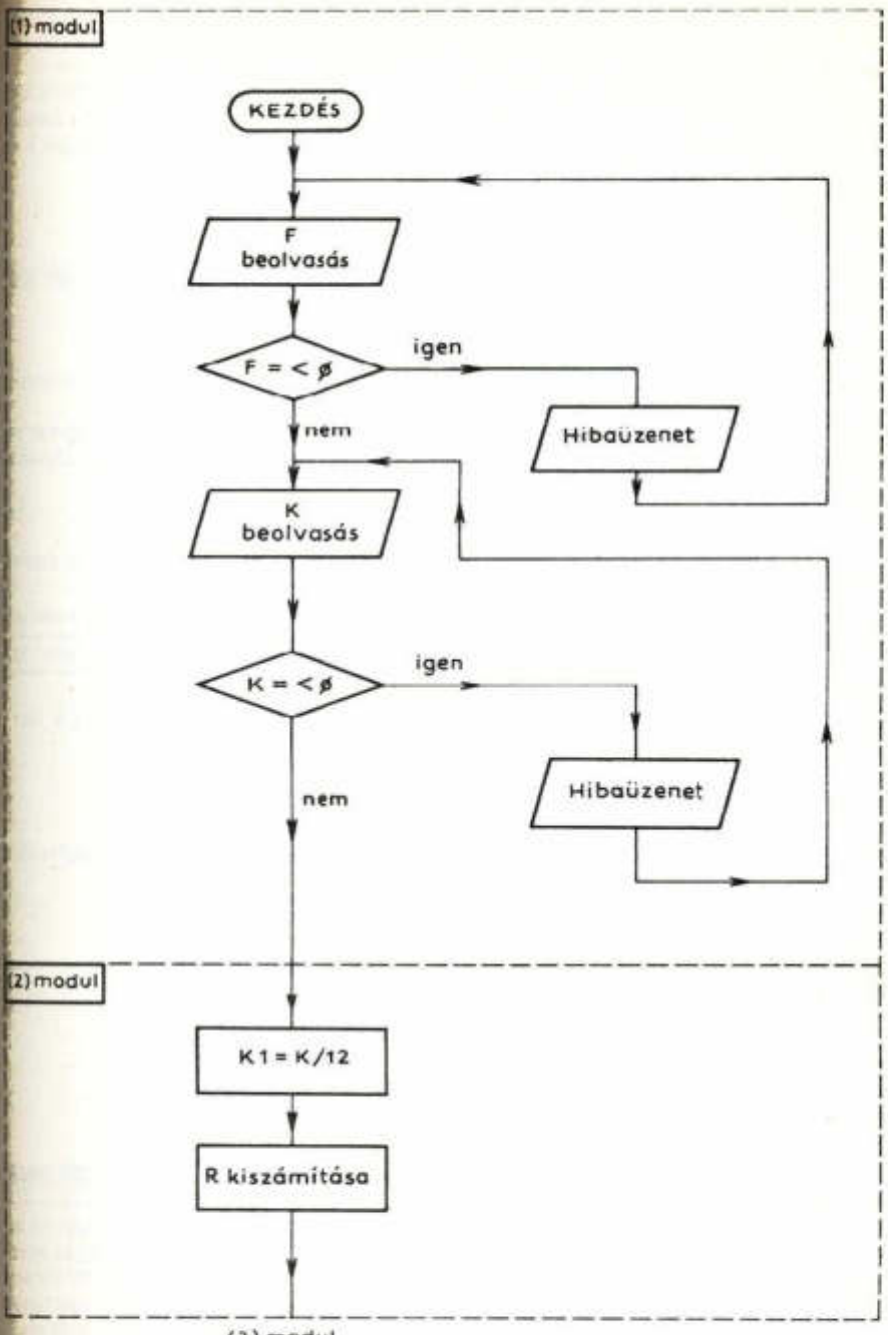
(2) *Részlet számítása*
A modul **eljárás** típusú. Bemeneti adatai az F és K érték az (1) modulból, kimeneti adata a havi részlet összege.

A modulban a kifejezés egyszerűbb kiszámítása végett a K/12-t egy K1 változóval helyettesítjük a

$$K1 = K/12$$

értékadás után. Majd egyetlen utasítással kiszámítható az R érték (59. ábra).

(3) *HK, CK, FJ szám*
A modul **eljárás** típusú. Bemeneti adatai a kölcsön összege (F) és a kamatláb (K) az (1) modulból.



59. ábra. Az (1) és (2) modul folyamata

Ez a ciklus első modulja, tehát a FOR utasítás ide kerül. Előtte azonban ki kell írni az oszlopok fejléceit. A modulban előbb a HK értéket számítjuk ki, majd ennek alapján a CK-t, végül az FJ-t.

Az első havi számításoknál az FJ a begépelt F értéke lesz. A továbbiakban az eredeti F értékre nem lesz szükségünk, ezért nem követünk el hibát, ha mindig az F változó tartalmazza a mindenkori FJ (a maradék kölcsön) összeget, vagyis F-et csökkentjük a visszafizetéssel.

A modul harmadik művelete tehát az F módosítása lesz:

$$F = F - CK$$

Így a következő számítási lépésnél már ez az új összeg lesz a kiinduló érték (60. ábra).

(4) *Összegezés*

A modul **eljárás** típusú. Bemeneti adatai az R, HK és CK értékek. A modul kimenetét három összeg alkotja.

A modulban három összegérték gyűjtését kell elvégezni. A részletösszegeket az SR, a kamatokat az SK, a visszafizetett összegeket az SC változó tartalmazza (60. ábra).

(5) *Kiírás*

A modul **eljárás** típusú. Bemeneti adatai az R, HK, CK és F adatok. A modul ezeket az adatokat egyetlen sorba írja ki.

Ebben a modulban fejeződik be a ciklusmag, ezért a modul végére kerül a ciklust záró NEXT utasítás (60. ábra).

(6) *Összegkiírás*

Hasonló az (5) modulhoz, azzal a különbséggel, hogy először egy üres sort ír (sor-emelés), majd az összegeket írja ki. Ezzel a program befejeződik (60. ábra).

Kódolás

A feladat kódjából néhány részletet mutatunk be. Az (1) modulban az F beolvasása és ellenőrzése a következő:

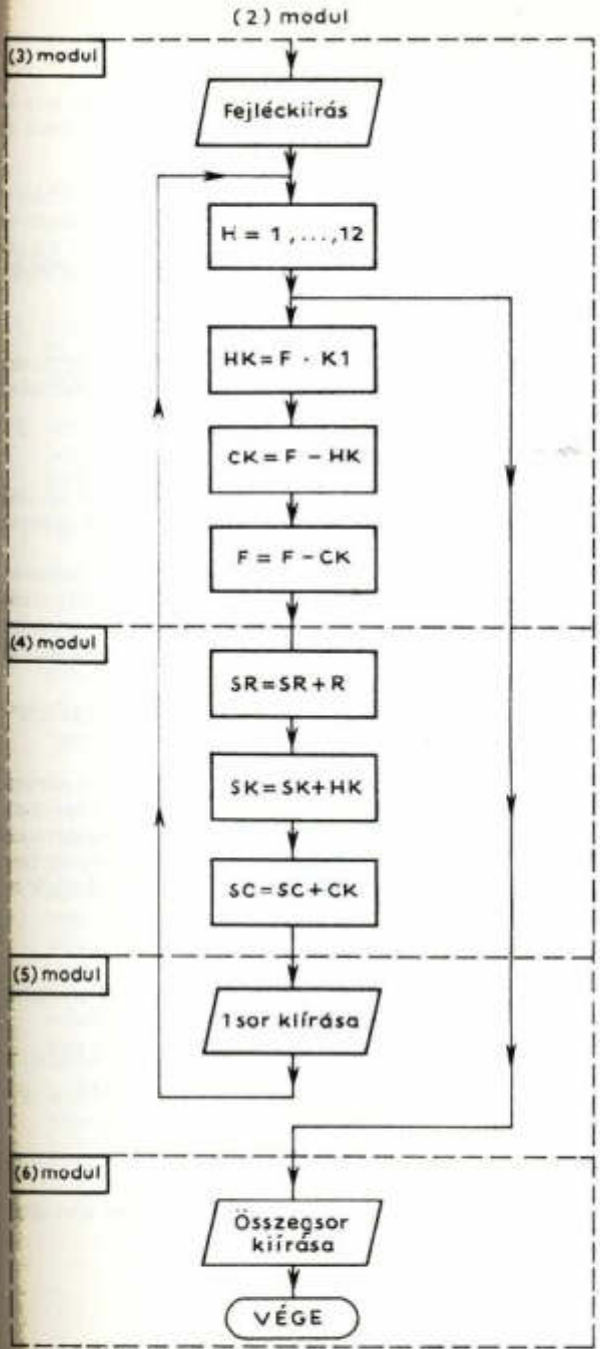
```
60 REM***** ADATBEOLV. *****
70 INPUT"A KOLCSON OSSZEGE: ";F
80 IF F<=0 THEN 100
90 GO TO 130
100 REM* THEN AG *
110 PRINT: PRINT"CSAK POZITIV LEHET !"
120 GO TO 70
130 REM* IF VEG *
```

Külön meg kell nézni a (4), (5) és (6) modulokban végbemenő négyoszlopos táblázat kiírását.

A Commodore-gépnél a PRINT utasításnak van olyan lehetősége, hogy ha az utasítás tárgyában levő adatelemek közé **vesszőt** teszünk, akkor a gép az egyes adatelemeket egy-egy tíz pozíció szélességű mező elejére írja ki. A Commodore 40 karakter hosszúságú sora tehát 4 oszlopra bontható, ezért a feladatunk így elvégezhető.

A fejlécet soronként kell kiíratni, és az oszlopok fejrovatait vesszővel kell elválasztani a PRINT utasításban:

```
280 PRINT"RESZLET","KAMAT","VISSZA-","KOLCSON"
290 PRINT"OSSZEG","OSSZEG","FIZ. O.","OSSZEG"
```



60. ábra. A (3), (4), (5) és (6) modulok folyamata

Ugyanígy kell az adatok kiírását is kódolni:

```
490 PRINT R,HK,CK,F
```

■ A HT gépen ugyanez a módszer követhető, mivel a képernyő egy sora 64 karakterből áll, és a PRINT utasításban az adatelemek közötti vessző az adatelemeket a 16 pozíció széles mező elejére írja ki.

▲ A PRIMO gép is – a PRINT utasítás adatelemei közötti vessző hatására – az adatelemeket a 16 pozíció széles mező elejére írja. A képernyősorba viszont csak 42 karakter fér el, így még 3 oszlop sem írható ki teljes biztonsággal. Ezért itt más módszert kell alkalmazni. A PRINT utasítással együtt használható a

TAB *(argumentum)*

alakú tabuláló függvény. Az argumentum csak pozitív és 42-nél kisebb lehet. A TAB függvényt csak a PRINT utasítással együtt lehet használni, önállóan nem. Az

x PRINT TAB *(argumentum)*; A

utasítás hatására az A értékének kiírása az argumentum értékével egyenlő sorszámú pozíción kezdődik. Például, ha a TAB argumentuma 10, akkor az A értékének nyomtatása a 10. pozíción kezdődik.

Ezzel az eljárással tehát tabulálni tudjuk a kiírást, vagyis az adatelemek kiírását a megfelelő pozícióra tudjuk állítani. A képernyőt itt is 10 pozíció széles mezőkre célszerű felbontani a TAB alkalmazásával. Az első adatelem kiírása a képernyő bal szélén kezdődik, ezért csak a második adatelemtől kezdve kell a TAB függvényt használni:

```
280 PRINT"RESZLET";TAB(10);"KAMAT";TAB(20);"VISSZA-";TAB(30);"KOLCSON"
290 PRINT"OSSZEG";TAB(10);"OSSZEG";TAB(20);"FIZ.O.";TAB(30),"OSSZEG"
```

● A Sinclair-gépeknél ugyanezt az eljárást kell követnünk, mivel az adatelemek közötti vessző alkalmazásával csak 2 oszlopot lehet írni a 32 helyérték hosszúságú képernyősorba. Ezért itt is a TAB függvényt kell alkalmaznunk (leírását lásd feljebb a PRIMO-nál), azzal a különbséggel, hogy itt csak 8 pozíció széles lehet egy oszlop. A Sinclair-gépeken a TAB függvény argumentumát nem kell zárójelbe tenni, például:

```
PRINT TAB 8
```

Ennek figyelembevételével a táblázat fejlécét a következő két utasítással lehet kiírni:

```
280 PRINT"RESZLET",TAB 8;"KAMAT",TAB 16;"VISSZA-",TAB 24,"KOLCSON"
290 PRINT"OSSZEG",TAB 8;"OSSZEG",TAB 16;"FIZ.O.",TAB 24,"OSSZEG"
```

Bár itt nem használjuk, de megjegyezzük, hogy a Commodore esetében az

x PRINT TAB *(n)*;*"szöveg"*

utasítás *n* szóközt hagy ki a képernyő bal szélétől vagy a legutolsónak kiírt jeltől jobbra, és ezután kezdi kiírni a *szöveget*.

Ellenőrző kérdések és feladatok

1. Alakítsa át az 5. feladat programját úgy, hogy hosszú lejáratú kölcsönök (20, 25 év) havi részletét lehessen vele kiszámítani. A kiírási képbe vegyen fel egy külön oszlopot (az első helyre), amelyben az év és a hónap van (pl. 1985. 6.).

2. Miért nem egyezik meg az 5. feladatban a visszafizetett részletek összege és a kölcsön összege (a kiíráson)? Mi az eredmény akkor, ha az összegeket kerekíti (999.99 Ft formában)?

3. Egy beruházás hozama a következő képlettel számítható ki:

$$H = (J-U)\,\frac{(1+K)^{N-1}}{J(1+K)^{N}}$$

ahol

H – a hozam,
J – az éves jövedelem a beruházásból,
U – az üzemeltetés éves költsége,
K – az érvényes kamatláb törtszám formájában,
N – az évek száma.

100 000 Ft-os beruházásnál az éves jövedelem (J) 10 000 és 20 000 Ft között változhat 1000 forintos lépésekben (10 000, 11 000, 12 000 . . .), az üzemeltetési költség pedig 3000 és 5000 Ft között, 500 forintos lépésekben. A kamat 6% vagy 8%. Milyen éves jövedelem és üzemeltetési költség mellett és melyik kamatlábbal lehet legalább 300 000 Ft hozamot elérni 10 év alatt?

4. Készítsen egy programot, amely a képernyő bal szélétől annyi karakterből álló vonalat ír ki, amilyen értéket a program a billentyűzetről beolvasott. Csináljuk meg a képernyő jobb szélétől számított vonalhosszra is!

5. Készítsünk programot, amely a billentyűzetről beolvasott szám (1 nem lehet) négyzetét kiírja, majd ennek a négyzetét, és így tovább! Mit tapasztal 1-nél nagyobb és kisebb számoknál?

9. ÖSSZETARTOZÓ ADATOK KEZELÉSE

Egyedi adatok – összetartozó adatok.
Tömbök és használatuk. Ciklikus műveletek tömbváltozókkal.
A 6. feladat megoldása

A SZÁMÍTÓGÉPES FELADATOK FELOSZTÁSA

A számítógéppel megoldható feladatok körében több szempontból is jól elkülöníthető csoportot alkotnak az **egyedi adatok** feldolgozását célzó feladatok. Erre a feladatcsoportra az jellemző, hogy a feladaton belül viszonylag **kevés** (legfeljebb néhányszor 10) és lényegében egymástól **független** adatot kell feldolgozni. A hangsúly inkább az utóbbin van, ami azt jelenti, hogy a feladat szinte valamennyi adatát külön kell kezelni és feldolgozni.

A feladatok másik csoportjába azok tartoznak, amelyekben több, valamilyen szempontból **összetartozó adatot** kell feldolgozni.

Az előző részekben **egyedi adatok** feldolgozására láttunk példákat, itt bemutatjuk az **összetartozó adatok** feldolgozását.

AZ ÖSSZETARTOZÓ ADATOK FELDOLGOZÁSA

Az egyedi adatok és az összetartozó adatok fogalmának pontos meghatározása nehézségekbe ütközik, jobb megértésükhöz három fogalmat ismerjünk meg. Ezek: az egyed, a tulajdonság és az érték.

Az **egyed** egy rendszernek olyan eleme, amelyet adatokkal kívánunk leírni. Egy vállalatnál egyed lehet egy termék, egy dolgozó, egy anyagféleség stb. A **tulajdonság** az egyedek jellemzője, amellyel leírhatók. Egy termék tulajdonsága pl. az ára, a súlya, a színe stb. Egy dolgozó tulajdonsága a szakmája, az életkora stb. Az **érték** egy tulajdonság konkrét megjelenése. Érték például egy termék számszerűen kifejezett ára, valamilyen mértékegységben meghatározott súlya, egy dolgozó szakmájának pontos megnevezése stb.

Egyedi adatok feldolgozásánál kevés egyed vesz részt a műveletben (pl. ellenállás-számítás). Általában a műszaki, tudományos számítások tartoznak a feladatoknak ebbe a körébe. Jellemzője ennek a feladatcsoportnak, hogy a feldolgozásban kevés számú (1–50) adat vesz részt.

Ezek az adatok egy, vagy kevés egyed tulajdonságainak az értékei. A kevés számú adat jelenlétéből következik, hogy a programban nem sok gondot kell fordítani az adatok beadására, ellenőrzésére, azonosítá-

sára, módosítására, rendszerezésére stb. Ezek a műveletek együttesen alkotják az **adatkezelést**. Egyedi feladatok esetében az adatkezelés kis helyet foglal el a programban, a nagyobb részt a feldolgozás teszi ki. Az adatok megőrzése sem merül fel problémaként, mivel ezek a programban változóként egyszerűen és problémamentesen tárolhatók.

Egészen más szemléletet igényel az **összetartozó adatok** feldolgozása. Ebben a feladatcsoportban sok egyed több-kevesebb tulajdonságának értékeit kell feldolgozni.

Ilyen feladat például egy raktárban levő termékek készletének nyilvántartása. Itt az egyes anyagféleségek az egyedek, és a mennyiségi tulajdonságaikat leíró értékeket (darabszám, súly stb.) kell feldolgozni. Az ilyen feladatokban sok adat szerepel (egyedenként legalább egy), és a sok adat kezelésére több energiát kell fordítani. Jelentős munka a részt vevő adatok számítógépbe vitele, a bevitel során végrehajtott ellenőrzések, az adatok megkülönböztetése (azonosítása). Ezt a későbbiekben alaposabban is szemügyre vesszük.

Nagy feladatot jelent továbbá az adatok tárolása, visszakeresése, törlése, módosítása is. Mindebből következik az is, hogy az összetartozó adatok kezelése a programra is hatással van, és a programok adatkezeléssel foglalkozó része lényegesen nagyobb, mint amit a feldolgozási műveletek foglalnak el. Ez az arányeltolódás azt is eredményezheti, hogy egy feldolgozási folyamatban olyan programok is vannak, amelyek csak adatkezelési lépést (pl. adatbeolvasás) hajtanak végre.

Az összetartozó adatok feldolgozása során egy fontos kérdés az adatok tárolása és a tárolt adatok megkülönböztetése. Hogyan tároljuk egy 5000-féle anyagot tartalmazó raktár 40–50 000 adatát? Hogyan tudjuk megmondani, hogy a 40 000 adat közül melyik tartalmazza a festékhígító aktuális mennyiségét? Először az első kérdést válaszoljuk meg. Ennyi adatot nem lehet az eddig megismert módon, azaz a programban tárolni. Nincs helyünk ilyen hosszú programok tárolására, de ennél prózaibb ok, hogy nincs annyi változónév, amit ennyi adatnak tudnánk adni. A programban néhány száz adatnál többet nem tudunk tárolni.

Nagy mennyiségű adatot úgy kell tárolni, hogy bármelyiket biztosan és könnyen megtaláljuk. Ez pedig csak úgy érhető el, ha az adatokat **rendszerezve** tároljuk. Ha például egy raktár anyagainak készletét kell nyilvántartanunk, akkor az egyedekről (anyagféleségek) legalább két tulajdonság értékét kell tárolni: az anyag azonosítására szolgáló számot (ez a „név") és a készlet tulajdonságértékét: a mennyiséget. Ezeket az adatokat valamilyen szempontból rendszerezni kell. Általában célszerű egy egyed tulajdonságainak értékeit együtt tárolni, ebből következik, hogy az anyagok azonosítóját és készletét együtt kell tárolni. Ezzel csoportokat hozunk létre, és tudjuk, hogy egy csoporton belül egy egyed adatai találhatók.

Tovább segítjük az adatok azonosítását, ha meghatározzuk, hogy a csoporton belül milyen sorrendben következnek az egyes adatok. Pél-

dánkban a sorrend lehet: azonosító – mennyiség. Így az adatcsoport első adata egy azonosító, az adathalmaz második eleme egy készletmennyiség. Ez ismétlődik minden egyed adatcsoportján belül. Ábrázolása a következő:

A1, M1
A2, M2
A3, M3
.
.
.

Látható, hogy egy sorban egy egyed tulajdonságértékei, egy oszlopban pedig azonos tulajdonságok értékei szerepelnek. Összetartozó adatok tárolására ez a fajta rendszerezés terjedt el.

Adatok fizikai tárolására két lehetőség adódik:

– programon belül,
– programtól függetlenül külső tárolón.

Az utóbbival a 13. és 14. részben foglalkozunk, itt a programon belüli adattárolást vizsgáljuk meg.

A programon belüli tárolási mód a tömb. A **tömb** egyedenként több, elvileg kötetlen számú tulajdonságértéket tartalmaz. A tömbben mindig van lehetőség az egyedi azonosító tárolására, ha szükséges. Tömbben lehet tárolni pl. egy méréssorozat adatait, ha mérésenként több jellemző értéket kell mérni. Megállapodás szerint **egy egyed adatai egy sorba, az azonos tulajdonságra** vonatkozó adatok pedig **egy oszlopba kerülnek.**

$$\begin{array}{lll} t_{1,1}=22.3 & t_{1,2}=20.8 & t_{1,3}=19.2 \\ t_{2,1}=20.8 & t_{2,2}=18.2 & t_{2,3}=17.6 \\ t_{3,1}=15.6 & t_{3,2}=14.1 & t_{3,3}=13.1 \end{array}$$

Az együvé tartozást az indexek is kifejezik. A kéttagú index első tagja az egyedre, a második a tulajdonságra utal. Ha az egyedek száma m, a tulajdonságok száma n, akkor m sorból és n oszlopból álló mXn méretű tömb szükséges az adatok tárolására. A BASIC általában több dimenziós tömböket tud kezelni.

A tömb egy konkrét elemére az indexeivel lehet hivatkozni. Annyi indexet kell megadni, ahány dimenziós a tömb. Egy kétdimenziós tömb elemére például az alábbiak szerint hivatkozunk:

$$t_{2,3}=17.6$$

A vektor egy speciális tömb, amelynek csak egy dimenziója van. A **vektor** elemei egyedenként egyetlen tulajdonság értékeit (az adatokat) tárolják. Ez lehet vagy az egyed azonosítója, vagy ha ez nem fontos, ak-

kor csak valamelyik tulajdonságának az értéke. Például olyan mérések nél, amikor mérésenként csupán egyetlen jellemzőt kell mérni, az adatokat tárolhatjuk egy vektorban. Ekkor a mérések azonosítóját külön nem lehet tárolni (mert nincs rá hely), de a tárolás sorrendje erre is utalhat. Tegyük fel, hogy a mérési eredmények a következők:

$$t_1 = 22.3$$
$$t_2 = 20.8$$
$$t_3 = 15.6$$
$$t_4 = 9.2$$
$$t_5 = 12.7$$
$$t_6 = 19.8$$

Ha t indexei a mérés sorszámára utalnak, akkor az adatokat a megadott sorrendben tárolva az azonosítókat is közvetve tároljuk. A vektorban csak az értékek szerepelnek:

$$t = \begin{bmatrix} 22.3 \\ 20.8 \\ 15.6 \\ 9.2 \\ 12.7 \\ 19.8 \\ . \\ . \\ . \end{bmatrix}$$

A vektor valamelyik adott elemére az elem indexével hivatkozunk:

$$t_5 = 12.7$$

Az összetartozó adatok **feldolgozását** jellemzi, hogy

- a program ciklust tartalmaz az azonos szempontú feldolgozások elvégzésére,
- sok az adatkezelés,
- a számítások viszonylag egyszerűek.

A sok összetartozó adatot tartalmazó problémánál az adatok szervezése, az adatállomány definiálása, értékének meghatározása külön tervezési lépést igényel. Először azt kell eldönteni, hogy milyen egyedek vannak, és ezek milyen tulajdonságai vesznek részt a feldolgozásban. Egy programban több tömb is szerepelhet. Mindegyik méretét, felépítését pontosan meg kell tervezni.

A tömbelemek ugyanazokkal a jelekkel jelölhetők, mint a változók. Az eltérés az, hogy a tömbnév után a zárójelbe tett számértékkel mutatjuk, hogy nem egyszerű változóról van szó. A zárójelbe írt szám az illető elem indexe.

Egyedi változó	Tömbváltozó	
A8	A8 (40)	az A8 vektor 40. eleme
C	C(200,50)	a C tömb 200. sorának 50. eleme
K2$	K2$ (2,20)	a K2$ tömb 2. sorának 20. eleme

Látható, hogy egy tömbön belül az elemek (változók) neve ugyanaz (tömbnév), és az indexszel azonosíthatók. A tömbök nevét ugyanolyan szabályok szerint lehet képezni, mint a változókét. A névvel a tömbben tárolt adatok típusára is utalhatunk. A tömbnév után írt $ jel azt jelenti, hogy a tömbben szöveges változókat, a % jel pedig azt, hogy a tömbben egész számokat tárolunk. Például:

A1% (5,7)
K$ (22,1)

- A Sinclair-gépeknél a tömbnév egyetlen betű lehet.

A program elején — vagy legalábbis a tömb használata előtt — a DIM utasítással kell helyet foglalni a tömbnek:

x DIM V (*egész szám*1, *egész szám*2, *egész szám*3, . . .)

ahol

DIM	— a helyfoglalás kulcsszava,
V	— a tömbnév,
egész számok	— a tömb mérete az egyes dimenziókban.

Ennek hatására a gép minden dimenzióban az egész számok által meghatározottnál eggyel nagyobb méretű V tömb számára foglal helyet, amelynek elemei közé tartozik a 0 indexű is. Például:

20 DIM T (15,3)

Ebben az esetben a sorok indexe 0-tól 15-ig, az oszlopoké 0-tól 3-ig van megengedve.

Egy DIM utasításban több tömböt definiálhatunk. Ekkor a tömbök nevét vesszővel kell elválasztani.

A tömb elemeire a vektor, illetve a tömb jelével és az elem sorszámával, illetve az indexeivel (zárójelben) hivatkozhatunk, amelyek nem lehetnek nagyobbak, mint a DIM utasításban meghatározott megfelelő méretek. Például:

T(3,2)

6. feladat

Legfeljebb 100, de esetenként változó számú mérési sorozat eredményeit (hőmérséklet) kell kiértékelni. Meg kell határozni a mérések átlagát és szórását. Az eredményeket a következő formában kell kiírni:

A MERESEK EREDMENYEI

A MERESEK SZAMA: X
AZ ATLAGOS HOMERSEKLET: X CELSIUS FOK
AZ ERTEKEK SZORASA: X.X

A feladat elemzése

A mérést — a probléma leírása alapján — nem a számítógép vezérli, ezért a mért értékeket a mérést végző olvassa le és adja a számítógépnek akár közvetlenül minden mérés végén, akár az adatok összegyűjtésével a méréssorozat befejezésekor. A mérési sorozat legfeljebb 100 mérésből állhat. Hogy a mérést végző ezt ne lépje túl, a mérések számát előre be kell kérni. Ezek alapján az adatbevitel programozható.

Az adatbevitel után az átlagot és a szórást a jól ismert kifejezések segítségével lehet meghatározni:

$$A = \frac{\sum_{i=1}^{m} t_i}{m}$$

ahol

A — átlag,
t_i — az i-edik mérés értéke,
m — a mérések száma.

$$D = \sqrt{\sum_{i=1}^{m} \frac{(A-t_i)^2}{m}}$$

D — a szórás.

A program adatait egy vektorban lehet tárolni. Mivel a vektor hőmérsékletadatokat tartalmaz, legyen a jele T. A vektor elemeit a felhasználó gépeli be. Minden esetben csak annyi adatot kell beírni, ahány mérést végez a felhasználó. Az adatokból előbb átlagot, majd szórást kell számítani.

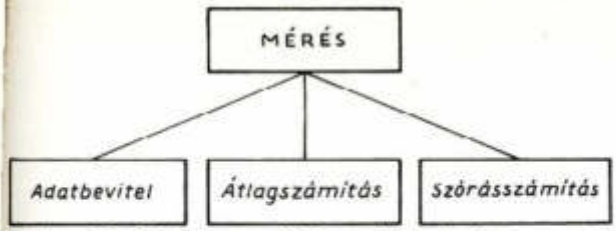
A számítógépes programban szükség van egy adatrögzítő modulra, amely rögzíti a bemeneti adatokat. Ezekből kell átlagot, majd szórást számítani. Az első modult kivéve mindegyik modulhoz tartozik kiírás is. Megoldásunkat úgy készítjük el, hogy a modulfunkciókhoz a kiírást is hozzávesszük (61. ábra).

A program tervezése

A program első modulja végzi az adatok bevitelét. Mérésenként legfeljebb 100 adat lehetséges. A felhasználó minden konkrét esetben megtervezi, hogy hány mérést végez, ezért az adatbevitel elején tudni lehet, hány adatot kell a modulnak beolvasnia a terminálról. Ezt a felhasználó közli. Az adatbevitelt ciklikusan kell elvégezni, mivel a feldolgozandó adatok beírása is hasonló. A modul további bontása felesleges.

A következő modul funkciója az átlagszámítás. Ehhez a mért értékeket összegezni kell, majd el kell osztani a mérések számával, a végén az eredményt ki kell írni. Tanácsos a modult két modulra bontani: az egyik végzi a feldolgozást, a másik pedig a kiírást.

A szórásszámítás a következő funkció, amely felhasználja az átlagszámítás ered-



61. ábra. A 9. feladat programjának modulszerkezete

ményét. Ezt a modult is célszerű kettévágni: az egyik modul végzi a számítást, a másik pedig a kiírást. A feladat finomított modulszerkezetét a 62. ábra mutatja.

A feladat programját vezérlőmodul nélkül készítjük el az egyszerű programszerkezet miatt.

A modulok tervezése

(1) *Az adatbevitel*

A modul **adat** típusú, mivel fő funkciója a mérési adatok beolvasása. Bemeneti adatai a mérési adatok, amelyeket a felhasználó ad be. Kimenete a számítógép tárjában levő adathalmaz, valamint a mérések száma.

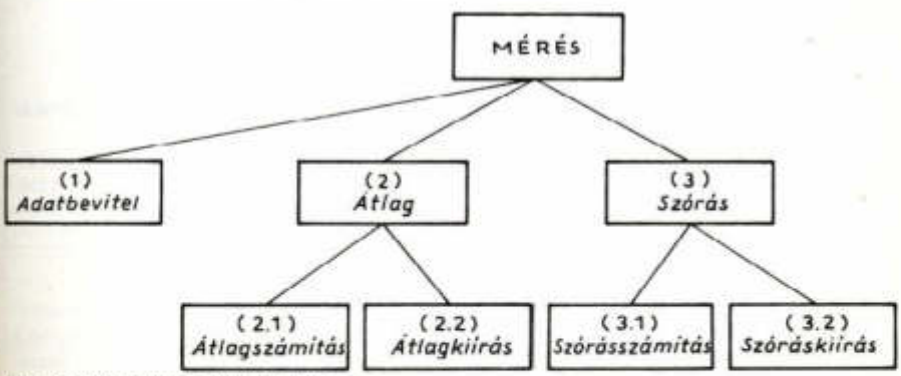
Ebben a modulban kap helyet a programnév, majd le kell kötni a mérési adatokat tároló 100 elemű, T nevű vektor helyét a megismert DIM T(100) utasítással. Ezután kezdődhet a modul „érdemi" feladatának ellátása, az adatok beolvasása. Első lépésként az adott mérési folyamat méréseinek számát kell bekérni:

A MERESEK SZAMA:

erre a felhasználó begépeli az M változó értékét.

A mérési adatok beolvasásakor minden egyes adat begépelése előtt tanácsos kiírni, hogy hányadik adat következik. Ezzel segíthetjük a felhasználót, hogy mindig helyes adatot írjon be. Az adatbevitelt **ciklusban** kell végezni, hiszen hasonló műveletet kell M-szer ismételni.

A **ciklusmagban** két művelet kap helyet: a következő adat sorszáma és az adatbeolvasás. A ciklust addig kell folytatni, amíg a beolvasott adatok száma eléri M-et. Ekkor a ciklusból ki kell lépni, és az adatbeolvasás befejeződött.



62. ábra. A 9. feladat programterve

63. ábra. Az adatbeolvasási ciklus szervezése

Az adatbeolvasási ciklusban fontos követelmény, hogy a beolvasott adatok a megfelelő indexű vektorelembe kerüljenek, ami azt jelenti, hogy az első beolvasott adat a T(1) értéke, a második adat pedig a T(2) értéke stb. legyen. Ez elérhető, ha az indexek értékét minden beolvasás előtt eggyel növeljük. Ez úgy oldható meg, hogy az indexet I változónak tekintjük, és értékét minden adatbevitel előtt eggyel növeljük (63. ábra).

Így biztosítjuk, hogy az adott sorszámú mérési adat az ugyanolyan sorszámú vektorelembe kerüljön. A ciklust egyszerűen meg lehet szervezni a FOR, NEXT utasításokból, ciklusváltozóként az I-t vezetjük be, és ez 1-től M-ig minden egész értéken végigfut a feldolgozáskor. Hogy mindig a ciklusváltozónak megfelelő vektorelemet dolgozza fel a program, a vektorelem indexét az I jelöli:

T(I).

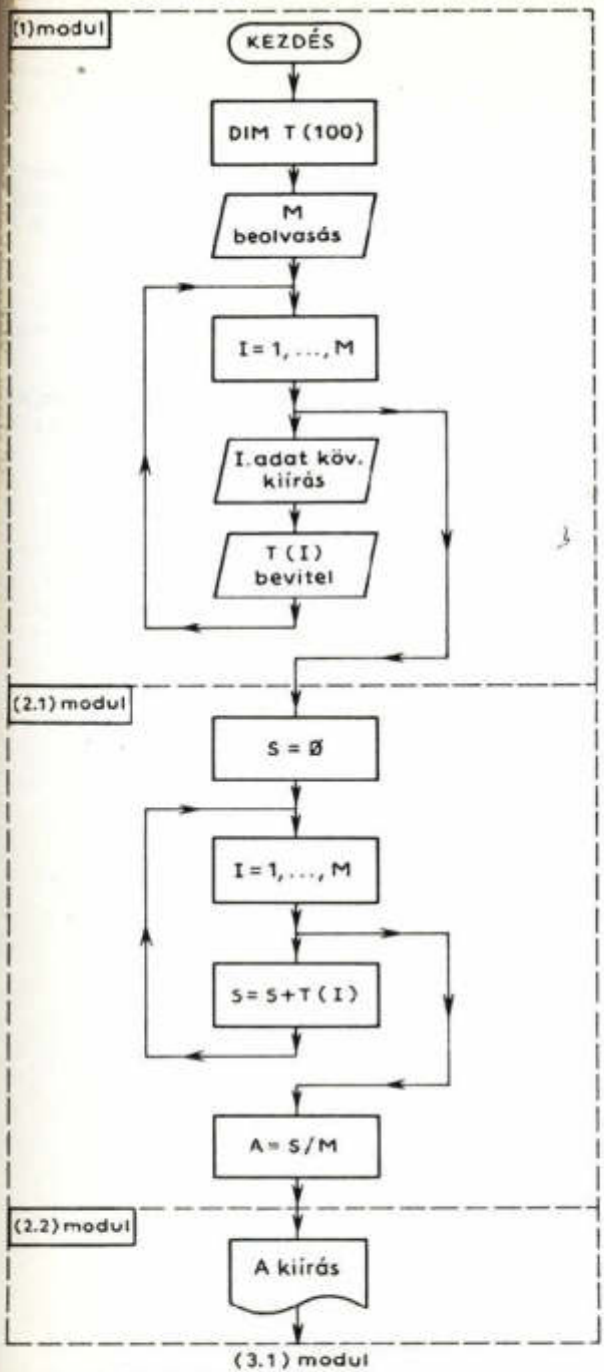
Ezzel a modul algoritmusát meghatároztuk (64. ábra).

(2.1) *Átlagszámítás*
A modul típusa: **eljárás.** Bemeneti adatai a T(I) mérési adatok és a mérések száma az (1) modulból. Kimeneti adata a mérési adatok átlaga.

Az átlagot a már ismert képlet alapján kell kiszámítani. Először a T(I) adatok értékeit kell összegezni. Ezt az összeget az S változó tartalmazza. Az összegezést ciklusban kell végrehajtani. A ciklusmagban az

$$S = S + T(I)$$

összegezést kell elvégezni. Ekkor is FOR, NEXT típusú feldolgozó ciklust alkalmazunk, amely biztosítja, hogy a program mindig a soron következő T(I) értéket adja hozzá az S-hez. A ciklus előkészítő műveletében az S gyűjtőt nullázzuk. Az összeget M-mel el kell osztani, és megkapjuk az átlagot (64. ábra).



64. ábra. Az (1), (2.1) és (2.2) modul folyamata

(2.2) *Átlagkiírás*
A modul **eljárás** típusú. Bemeneti adata az A átlagérték, amely egyúttal a kimenete is. A modul egyetlen művelete az átlag kiírása a specifikáció szerint (64. ábra)

(3.1) *Szórásszámítás*
A modul **eljárás** típusú. Bemeneti adatai a mérési adatok és az adatok száma az (1 modulból, valamint a szórás számításához szükséges A átlagérték a (2.1) modulból A modul kimenete a szórás (D2).

Először az

$$[A-T(I)]^2$$

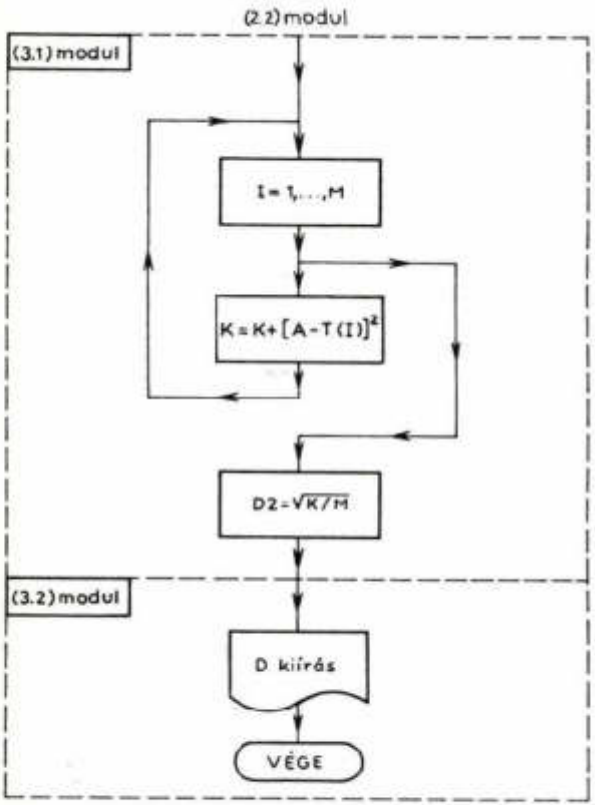
különbségek négyzetét kell képezni a K változóban:

$$K = K+[A-T(I)]^2$$

Ezt a műveletet is egy M-szer végrehajtott ciklusban célszerű elvégezni FOR, NEXT szervezéssel. A kapott eredményt el kell osztani M-mel, és négyzetgyökvonás után megkapjuk a szórás értékét:

$$D2 = \sqrt{K/M}$$

A modul műveleteit a 65. ábra mutatja.



65. ábra. A (3.1) és (3.2) modul folyamata

(3.2) *Szóráskiírás*
A modul **eljárás** típusú. Bemeneti adata a D2 jelű szórásnégyzetet tartalmazó változó a (3.1) modulból, amely egyúttal a modul kimeneti adata is (65. ábra).

A program kódolása

A program kódolásában az egyedüli újdonság a vektorméret meghatározása:

```
70 DIM T(100).
```

a többi művelet az eddig bemutatott kódolási szerkezetek felhasználásával megoldható.

10. CIKLIKUS TEVÉKENYSÉGEK

Adatok rendezése tömbben, egymásba ágyazott ciklusok.
A 7. feladat megoldása

Ebben a részben egy újabb feladatot oldunk meg a ciklikus tevékenységek gyakorlására és a tömbök alkalmazására.

7. feladat

Egy vállalat pénztára nem bérfizetési napokon a dolgozóknak nem bérjellegű kifizetéseket teljesít. Ezek a vállalat jellegéből fakadóan főleg helyi (taxi) és távolsági közlekedési költségek (kiküldetések útiköltsége, napidíj, szállásköltségek), valamint mások. A költségek fő jellegzetessége, hogy valamilyen munkaszámra rá kell őket terhelni. Naponta legfeljebb 100 ilyen kifizetés várható.

A könyvelés megkönnyítésére minden nap végén egy kimutatást kell készíteni, amelyben az azonos munkaszámra terhelhető költségek egy csoportban vannak. Így a könyvelőnek egy munkaszám könyvelési kartonját csak egyszer kell kézbe vennie. További segítséget jelent, ha a kimutatáson a munkaszámonként csoportosított bizonylati adatok sorrendje ugyanolyan, mint a könyvelési kartonoké. Ez utóbbiak növekedő sorrendben vannak, tehát a munkaszámoknak a kimutatáson is növekvő sorrendben kell szerepelniük. A kimutatás elején legyenek a legalacsonyabb munkaszámhoz tartozó kifizetési bizonylati adatok, utána a következő munkaszámhoz tartozók, és így tovább.

Mi legyen még a kimutatáson? Mivel könyvelési célra kell, a kifizetett összegnek feltétlenül szerepelnie kell, valamint a későbbi ellenőrizhetőség végett az adatok származási helyét – a bizonylat azonosítóját, azaz a bizonylat sorszámát – is fel kell tüntetni. Természetesen nem maradhat el a kifizetés kelte sem. Mivel a kimutatás naponta készül, elég a kimutatásra egyszer felírni az aznapi dátumot. A munkaszámok négyjegyű számok, a kifizetések nem haladják meg a tízezer forintot, és a bizonylatsorszám legfeljebb 8 jegyből áll.

A kimutatást a kifizetésekről sornyomtatóval kell elkészíteni a következő formában:

NAPI KIFIZETESEK

KELT 1985. EV 2. HO 23. NAP

MUNKA-SZAM	KIFIZETETT OSSZEG	BIZ. SORSZ.
9999	9999.99	99999999

(A 9-es számjegyek azt jelölik, hogy helyükre számjegyeket kell írni, és legfeljebb annyi helyértéken, ahány 9-es ott van.)

A feladat elemzése

A feladat értelmében egy vállalati pénztár napi kifizetéseiről kell naponta munkaszámonként, a munkaszámok emelkedő sorrendjében kimutatást készíteni.

A kimutatás elkészítéséhez kifizetésenként az alábbi adatok szükségesek:

- a terhelhető munkaszám,
- a kifizetett összeg,
- a bizonylat sorszáma.

A munkaszámot tehát mindig rá kell írni a bizonylatra. A kifizetett összeg természetesen rajta van a bizonylaton, hiszen éppen az a célja. Sorszáma is minden bizonylatnak van. Így a kimutatáshoz szükséges adatok biztosíthatók.

A nap végén a bizonylati adatokat a munkaszámok növekvő sorrendjében kell összegyűjteni és kiírni. A nap folyamán azonban a kifizetések nem a munkaszámok emelkedő sorrendjében történnek, hanem véletlenszerűen. A feladat egyik része ezért az, hogy a bizonylatok adatait a nap végén a munkaszámok szerint növekvő sorrendbe kell rendezni. Ahhoz, hogy ezt meg lehessen tenni, a bizonylatok adatait tárolni, és a nap végén az adatokat rendezni kell. A rendezett adatokat pedig ki kell írni.

A feladat számítógépes megoldásához az szükséges, hogy a pénztáros a kifizetések alkalmával ezek három adatát (munkaszám, összeg, bizonylatszám) beírja a számítógépbe, és a nap végéig megőrizze a gépen. Ekkor a kifizetésekhez tartozó adatokat munkaszámok szerint növekvő sorrendben össze kell rendezni, és ki kell írni egy kimutatásra. A kimutatásra az aznapi dátumot is fel kell írni, ezért a kiírás előtt valamikor ezt is be kell gépelni. A feladat elvileg megoldható, hiszen ezek a műveletek számítógéppel elvégezhetők.

Egy dolgot kell még megvizsgálni. Az aznapi bizonylati adatokat egész nap tárolni kell. Megoldható ez? Akkor igen, ha az adatok mennyisége nem lépi túl a gép adattárolási képességét. Esetünkben maximum 3×100=300 adatot kell tárolni. Egy numerikus adat tárolásához géptől függően 4–12 bájt szükséges. Ezek szerint legfeljebb 3600 (3,6 kbájt) szükséges az adatok tárolásához. Nem szabad elfelejteni, hogy a feladatot megoldó program is helyet foglal el. Azt azonban előre nem tudjuk megmondani, hogy mekkorát. Mégis azt mondhatjuk, hogy amelyik gépnek a tárkapacitása 8 kilobájt vagy e feletti, azzal a feladatot nagy valószínűséggel meg lehet oldani.

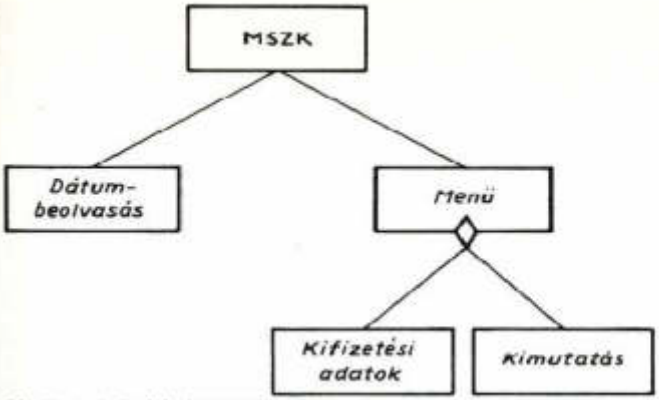
A feladat megoldásához nyomtatóra is szükség van, hogy a kimutatást el lehessen készíteni. A feladat tehát számítógépeinkkel megoldható.

Nézzük meg, milyen fontosabb részfeladatokat kell elvégezni a feladat megoldásához! A napi dátumot a nap kezdetén érdemes beolvasni. Egy részfeladat tehát a dátum beolvasása. Ezután következik a bizonylatonkénti három adat beolvasása, hogy a gép tárolni tudja az adatokat. A nap végén az adatokat sorrendbe kell szedni és ki kell írni.

Egy adatbeolvasás után így két dolog következhet:

- újabb adatbeolvasás,
- rendezés, kiírás.

Ezt a kiválasztást (döntés) a pénztárosnak kell megtennie. A beolvasások után ezért mindig ki kell írni egy menüt, hogy a pénztáros választhasson. A nap kezdetén, a



66. ábra. A 7. feladat szerkezete

dátum beolvasása után is ki kell írni ezt a menüt a képernyőre, hogy a műveletek egységesek legyenek (a nap kezdetén elég lenne rögtön az adatbevitelre rátérni).

Egy másik részfeladat a bemutatott menü kiírása. A menüből két további részfeladatot lehet hívni: az adatbeolvasást és a kimutatás készítését (66. ábra).

A program tervezése

Először vizsgáljuk meg, hogy lehet-e tovább bontani a részfeladatokat!

A dátum beolvasása egyszerű feladat, nem érdemes tovább bontani. A kifizetési adatok beolvasása három adat beolvasásából áll minden további művelet nélkül, ezért ez is lehet egy modul. A kimutatáskészítés két további feladatra bontható: először sorrendbe kell rendezni az adatokat, majd a rendezett adatokat ki kell írni. A kimutatás tehát két modulból fog állni:

- rendezés,
- kiírás.

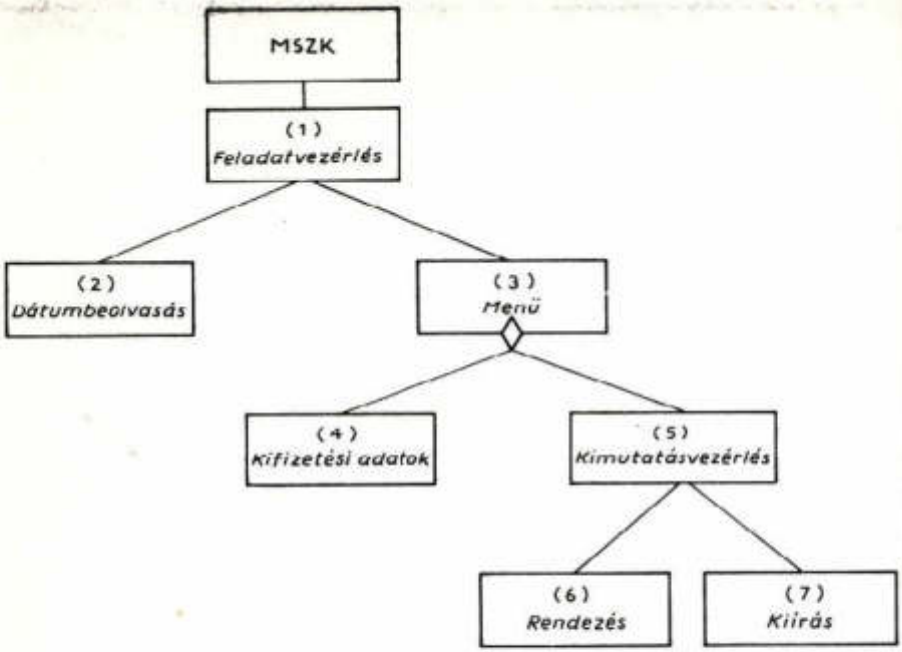
A feladat modulszerkezetéhez tartozik a program vezérlési módjának meghatározása is. Ebben a feladatban a bemutatott programtervezési módszerünk szerint járunk el.

A dátumbeolvasás és a menükiírás műveleteit egy vezérlőmodul irányítja, ez lesz a *feladatvezérlő* modul (67. ábra). A menükiíró modul a kifizetési adatok beolvasását és a kimutatáskészítő modult fogja irányítani. Ez utóbbi maga is vezérlőmodul, amely előbb a rendezés, majd a kiírás végrehajtásáról gondoskodik.

Ezzel a feladat modulszerkezete kialakult (67. ábra). A feladat egy programmal megoldható! Az (1) modul kivételével valamennyi modult szubrutinként fogjuk megvalósítani. Külön ki kell térni a feladatban tárolandó adatok tárolási módjára. A feladat megfogalmazásából nyilvánvaló, hogy a legfeljebb 100 kifizetési bizonylat munkaszám, összeg, bizonylatszám adatai összetartozó adatok. Az egyed itt a kifizetés, és három tulajdonságának értékeit kell megőrizni. A tulajdonságok:

- terhelhető munkaszám,
- kifizetett összeg,
- bizonylatszám.

Az adatok viszonylag alacsony száma miatt a programon belüli adattárolás megoldható. Az adatbeolvasás — mint látni fogjuk — a rendezés és a kiírás ciklus alkal-



67. ábra. A 7. feladat modulszerkezete

mazásával elvégezhető, ezért az adatokat tömbben kell tárolni. Legyen a tömb neve K.

A K tömbön belül egy sorban egy egyed adatai helyezkednek el. Egy egyedről három adatot kell tárolni, és mondjuk azt, hogy munkaszám, kifizetett összeg, bizonylatszám sorrendben.

Ebből következően az adattárolásra 100 soros és 3 oszlopos tömböt kell felvenni. A tömb 1. sorában lesznek az 1. bizonylat, második sorában a 2. bizonylat – és így tovább – adatai.

Ha 100-nál kevesebb kifizetés van, akkor nem célszerű a teljes tömböt kezelni a rendezés és a kiírás során, csak annyit, amennyi adat van. Ezért vegyünk fel egy N változót, amely a kifizetések számát tartalmazza.

A modulok tervezése

(1) *Feladatvezérlő*

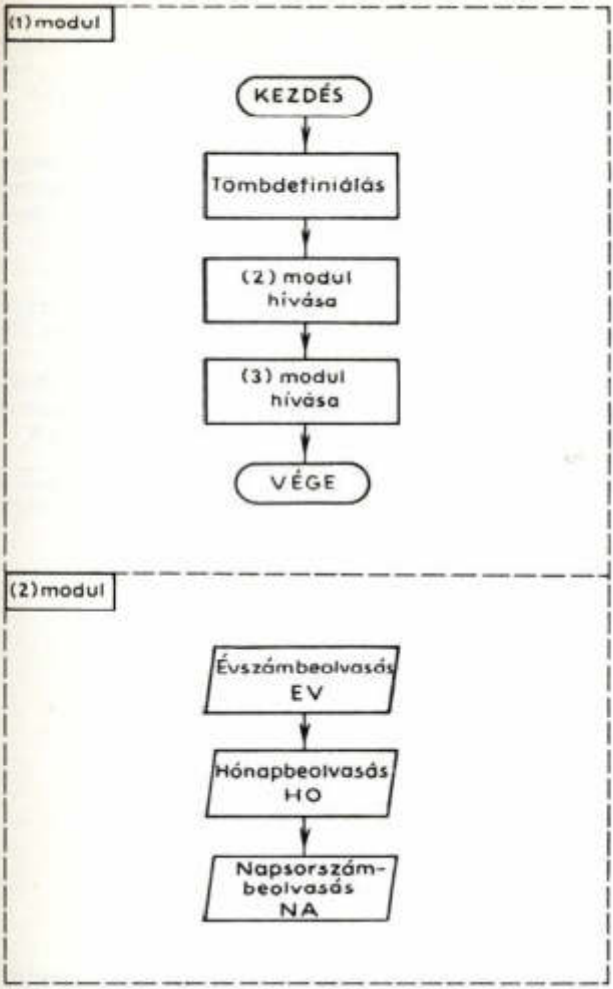
A modul **vezérlő** típusú. Bemeneti adata nincs. Kimenete a (2) és a (3) modul hívása.

A program első moduljában kell definiálni a felhasznált tömböt, majd két szubrutinhívással végre kell hajtani a (2) és a (3) modult. Ezután a program befejeződik (68. ábra).

(2) *Dátumbeolvasás*

A modul **eljárás** típusú. Bemenete az aznapi dátum, és ez a modul kimenete is.

A modulban be kell olvasni az évszámot, a hónapsorszámot és a nap számát (68. ábra).



68. ábra. Az (1) és (2) modulok folyamata

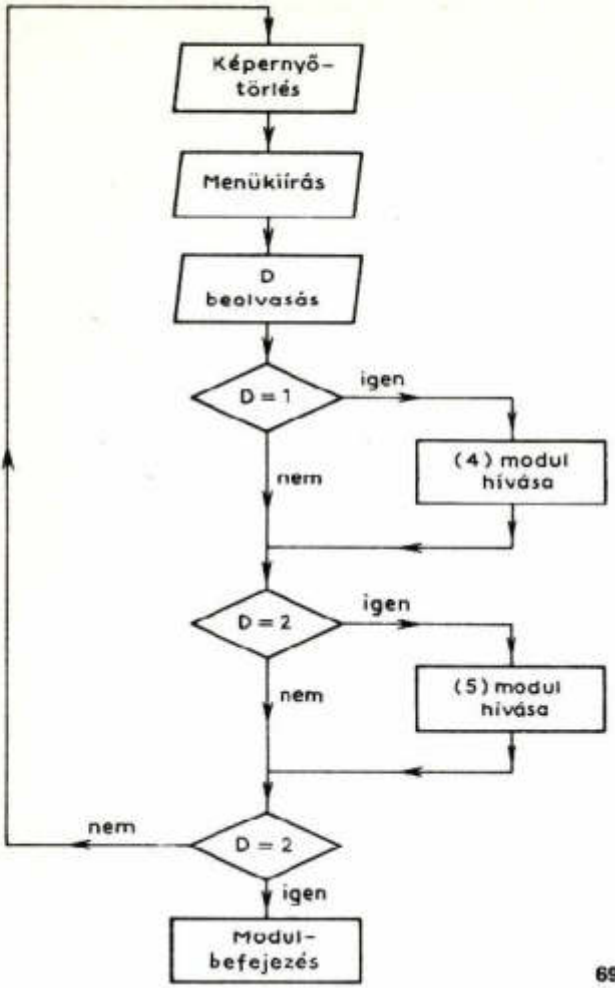
(3) *Menü*

A modul **vezérlést** lát el. Bemenete a kiválasztott müveletnek megfelelő adat. A modul kimenete vagy a (4), vagy az (5) modul hívása.

A modul kezdetén törölni kell a képernyőt, majd ki kell írni a menüt, és be kell olvasni a döntésnek megfelelő adatot (D). Ezután D értékének megfelelően kell a két modul közül valamelyiket hívni. Ha

D=1, akkor a kifizetési adatokat kell beolvasni,
D=2, akkor a kimutatást kell hívni.

Az utóbbi esetben a modul befejeződik. Egyébként újra ki kell írni a menüt az újabb döntéshez (69. ábra).



69. ábra. A (3) modul folyamata

(4) *Kifizetési adatok*

A modul **adat** típusú. Bemenete a kifizetés adatai: a munkaszám, a kifizetett összeg és a bizonylatszám, amelyeket a billentyűzetről kell beolvasni, és ezek a modul kimeneti adatai is.

A munkaszám mindig egy I sornak az 1., a kifizetett összeg a 2., a bizonylatszám a 3. eleme. Jelölésben:

munkaszám:	K(I,1)
kifizetett összeg:	K(I,2)
bizonylatszám:	K(I,3)

A modulban előbb törölni kell a képernyőt, majd a munkaszám, a kifizetett összeg és a bizonylat sorszámát kell beolvasni.

A beolvasott adatokat a K tömbben kell tárolni. Hogy a beolvasott adatok ne vesszenek el, minden adathármast a tömb következő üres sorában kell elhelyezni.

Ehhez fel kell venni egy változót (N), amelynek értéke biztosítja, hogy a begépelt adatok a következő üres sorba kerüljenek, és minden beolvasás után gondoskodni kell a növeléséről (felkészítés a következő beolvasásra).

Az első adat beolvasásakor

N=1

legyen, majd minden újabb beolvasáskor az N-t eggyel meg kell növelni, hogy a következő adatok már a következő tömbsorba kerüljenek. A növelést célszerű a modul elejére tenni, mert így az index a beolvasott adatok sorszámát jelöli.

A modul első hívásakor

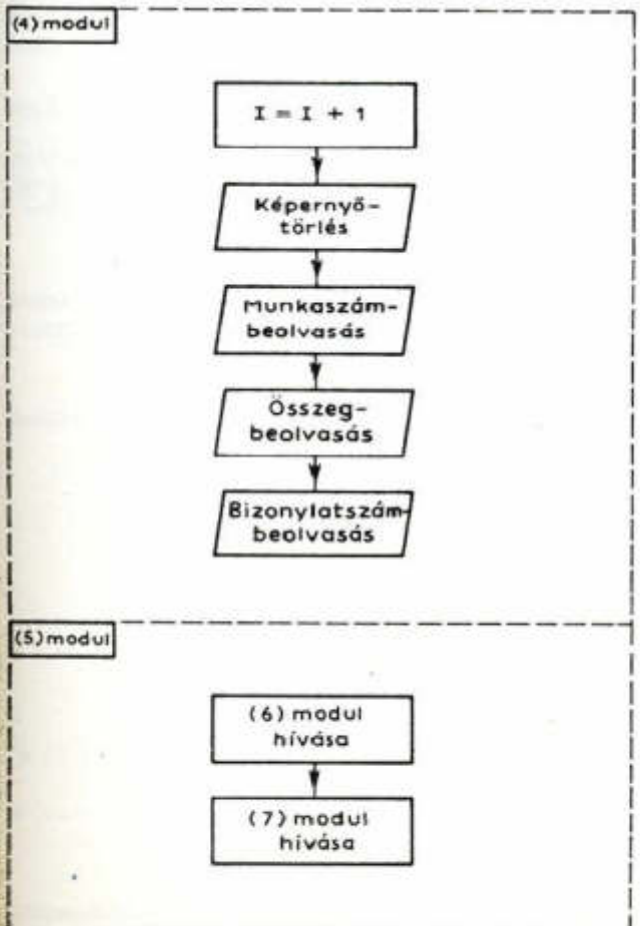
N=0

legyen (mert az N növelés után 1 lesz). A BASIC nyelvben ez automatikusan teljesül, tehát külön műveletet nem igényel (70. ábra).

(5) *Kimutatásvezérlő*

A modul **vezérlő** modul. Bemeneti adata nincs. Kimenete a (6) és a (7) modul hívása.

A modul előbb a (6), majd a (7) modult hívja (70. ábra).



70. ábra. A (4) és (5) modul folyamata

(6) *Rendezés*

A modul **eljárást** valósít meg. Bemenete az N kifizetésszám és a K tömb elemei. Kimenete a K tömb rendezett változata.

A rendezést csak az első N sorra kell elvégezni. Vizsgáljuk meg, hogy a kifizetések adatai milyen módszerrel rendezhetők! A rendezés szempontja a munkaszám. A sorba rendezést többféleképpen lehet végrehajtani. Ezek közül mutatunk be egy megoldást. A megoldás lényege a **páronkénti vizsgálat és csere.** Példaként tekintsük a 71. ábra bal oldali oszlopának két felső elemét. Tegyük fel, hogy a cél az oszlop (vektor) elemeinek növekvő sorrendű sorba rendezése.

Ha a két elem helyes sorrendben van (növekvő), akkor nem kell semmit sem tenni. Térjünk most át a 2. és 3. elemre (2. oszlop)! Ez a két elem nincs sorban, ezért fel kell cserélni őket (az eredmény a 3. oszlopban látható). Majd a 3. és 4. elemre térünk át. Azt látjuk, hogy itt is cserélni kell. A műveletet tovább folytatjuk, amíg el nem kezdjük elölről. Megint elkezdjük a szomszédos elemek összehasonlítását és szükség szerinti cseréjét.

A sorba rendezés akkor ér véget, ha a tömbön úgy mentünk végig, hogy nem kellett elemet cserélni. Hogy tudjuk, ez mikor következik be, fel kell venni pl. egy F változót, amelynek értéke megmondja, kellett-e elemet cserélni. Minden vizsgálatsorozat előtt adjunk F-nek nulla értéket. Ha a vizsgálatsorozatban nem kell cserét végrehajtani, akkor

$$F = 0$$

marad. Ha viszont egyetlen cserét is el kell végezni, akkor F értéke 1 legyen. Ez az 1 mutatja meg, hogy a sorba rendezés még nem ért véget.

Könnyen belátható, hogy a vizsgálatsorozat (páronkénti összehasonlítás és szükség szerint csere a tömb elejétől a végéig) ciklikus művelet. Ugyanis minden vizsgálatnál az I-edik sorszámú elemet hasonlítjuk össze az I+1-edikkel, jelen esetben a munkaszámot:

$$K(I,1) \leqslant K(I+1,1)$$

Az első esetben I=1, ezért az 1. és a 2. munkaszámot kell összehasonlítani. Utána a 2. és 3. elemet kell vizsgálni, tehát I értékét 1-gyel növelni kell. Ezt folytatjuk egészen

$$I = N-1\text{-ig.}$$

A műveletet I=1-től kezdve I=N−1-ig kell végrehajtani úgy, hogy az I növekménye 1 lesz. Ez a ciklus a FOR, NEXT utasításpárral oldható meg legegyszerűbben.

	1.	2.	3.	
1.	18.2	18.2	18.2	. . . .
2.	20.4	20.4	17.5	. . . .
3.	17.5	17.5	20.4	. . . .
4.	16.3	16.3	16.3	. . . .
5.	19.8	19.8	19.8	. . . .
6.	18.7	18.7	18.7	. . . .

71. ábra. Példa a sorba rendezésre

Nézzük meg, hogy két szomszédos elem vizsgálatát hogyan kell elvégezni! Ha

$$K(I,1) \leqslant K(I+1,1)$$

fennáll, akkor át kell térni a következő elempár vizsgálatára, vagyis az I-t eggyel meg kell növelni, és így kell a vizsgálatot megismételni. Ha a feltétel nem áll fenn, akkor végre kell hajtani az elemcserét. Ezt csak két lépésben lehet elvégezni, mert két értékadást kell végrehajtani:

$$K(I,1) = K(I+1,1)$$
$$K(I+1,1) = K(I,1)$$

Ha az első lépésben a K(I,1) felveszi a K(I+1,1) értékét, akkor a K(I,1) eredeti értéke elvész, és a második lépésben már nem történne semmi. Ezért mielőtt az első lépést végrehajtjuk, a K(I,1) értékét valahol meg kell őrizni. Legyen az SM változó:

$$SM = K(I,1)$$

Ha most végezzük el az első lépést:

$$K(I,1) = K(I+1,1)$$

akkor a K(I,1) értéke már nem vész el. A második lépésben a K(I+1,1) értékét az SM-mel kell egyenlővé tenni:

$$K(I+1,1) = SM$$

Ezt a cserét nemcsak a munkaszámoknál kell elvégezni, hanem a munkaszámokhoz tartozó másik két adatnál is (72. ábra).

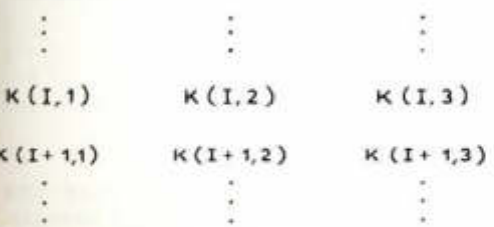
Itt is hasonlóan kell eljárni. A kifizetett összegeknél az SF, a bizonylatszámnál az SB változót használjuk fel:

$$SF = K(I,2)$$
$$K(I,2) = K(I+1,2)$$
$$K(I+1,2) - SF$$

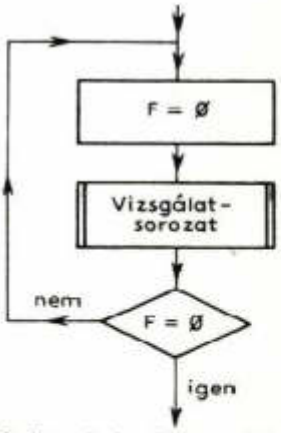
illetve:

$$SB = K(I,3)$$
$$K(I,3) = K(I+1,3)$$
$$K(I+1,3) = SB$$

Ezzel a cserét végrehajtottuk. Ez egyben a ciklusművelet is, vagyis ezt kell az I=1



72. ábra. A kifizetési adatok cseréje



73. ábra. A vizsgálatsorozatot tartalmazó ciklus

indextől N–1-szer végrehajtani. Ha cserét hajtunk végre, akkor az F=1 értékadást is el kell végezni.

A vizsgálatot tartalmazó ciklust addig kell ismételni, amíg minden elem a helyén nincs. Ilyenkor a vizsgálatsorozat végén

$$F = 0$$

marad, és ez jelzi, hogy az adatok sorba rendezése megtörtént.

Észrevehetjük, hogy maga a vizsgálatsorozat is egy cikluson belül van. Ennek a ciklusnak a ciklusművelete egy egész vizsgálatsorozat. Ezt addig kell ismételni, amíg az

$$F = 0$$

feltétel nem áll fenn. Ez tehát egy hátul tesztelő ciklus lesz (73. ábra).

A vizsgálatsorozatot ismétlő cikluson belül van az elemcserék ciklusa. Ezért ez a két ciklus egymásba ágyazott ciklust alkot. Ezek után a modul teljes folyamatát megterveztük (74. ábra).

(7) *Kiírás*

A modul **eljárás** típusú. Bemeneti adatai a K tömb sorba rendezett értékei, kimenetét is ezek az adatok képezik.

A modulban ki kell nyomtatni a K tömb első N sorának adatait a specifikáció szerinti formában. Először a fejlécet (dátum, cím és az oszlopok fejrovata), utána a K tömb rendezett elemeit kell kinyomtatni. Egy sorba három adatot kell kiírni, amelyek a K tömb egy sorát alkotják:

K(I,1) K(I,2) K(I,3)

Eddigi tapasztalataink alapján már tudjuk, hogy ezeket a sorokat is ciklusban lehet kiíratni.

A ciklusművelet egy sor kiírása. Ezt annyiszor kell ismételni, ahány érvényes sor (N) van a K tömbben. Erre is jól használható a FOR, NEXT utasításpárból felépített ciklus. A modul műveleteit a 75. ábra mutatja.

A modulok kódolása

A program tervezése során megállapítottuk, hogy az (1) modul kivételével minden modult szubrutinként kódolunk.

Az (1)–(5) modul kódolása eddigi ismereteink alapján mind a négy gépen elvégezhető. Különbség a képernyő törlésében és az INPUT utasítás formájában (veszsző, illetve pontosvessző) van. A kiírást viszont gépenként kell megvizsgálni.

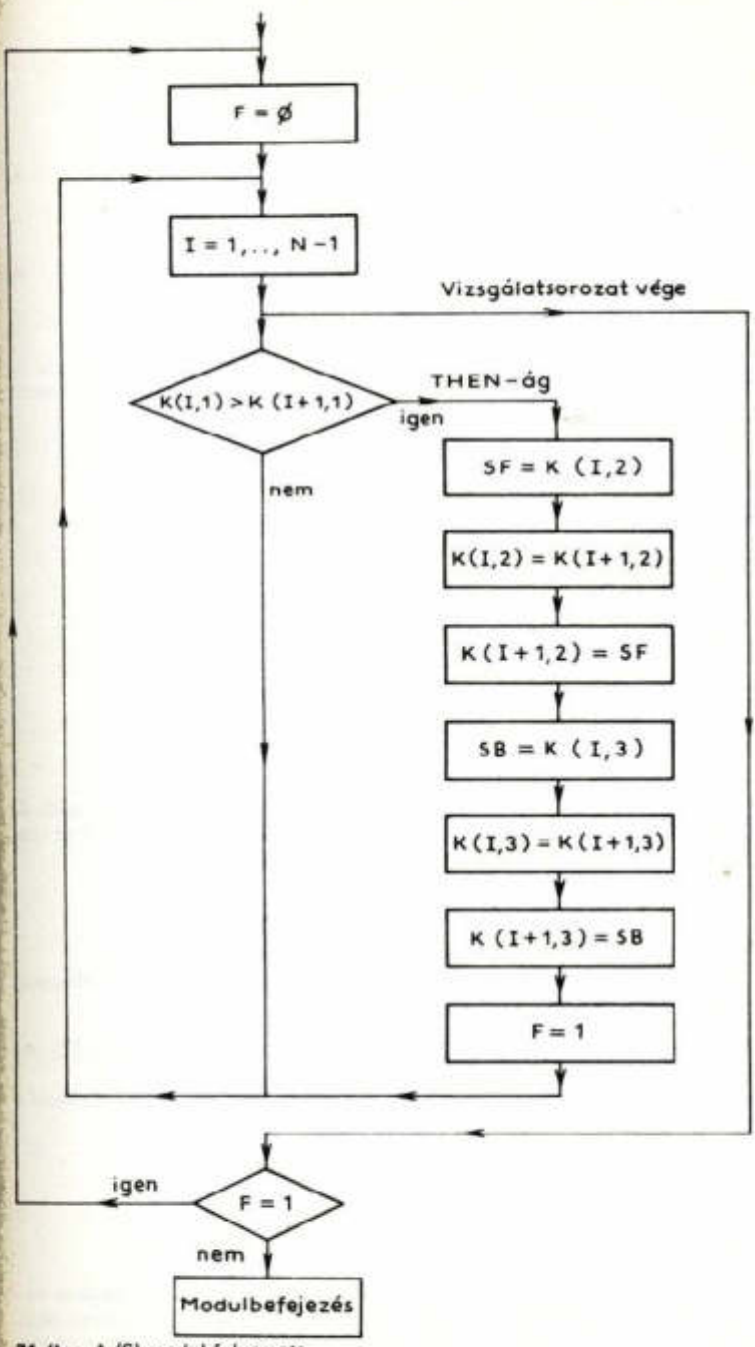
A Commodore-nál a nyomtató használatára külön utasítás van, a PRINT#. A gép a nyomtatót nem tekinti „természetes" tartozékának, ezért a használat előtt összeköttetést kell létrehozni a gép és a nyomtató között. Erre szolgál az OPEN utasítás:

x OPEN *állományszám*,4

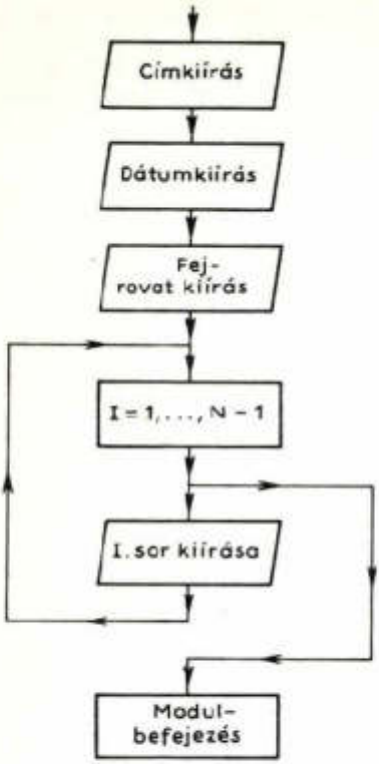
ahol

OPEN	– a kapcsolat-létrehozó utasítás kulcsszava,
állományszám	– 2 és 127 közötti kapcsolatazonosító,
4	– a nyomtatóeszköz száma.

Az állományszámnak akkor van jelentősége, ha több állományt használunk. Erre majd a 13. és 14. részben látunk példát. Esetünkben válasszuk a 2-t állományszámként.



74. ábra. A (6) modul folyamata



75. ábra. A (7) modul folyamata

A megnyitás után következhetnek a kiírási utasítások a PRINT # alkalmazásával. A PRINT # utasítás használati módja megegyezik a PRINT utasításéval. De az utasítás formája az eszköz miatt eltér a már ismert PRINT utasítás formájától.

x PRINT # *állományszám, adatelemek*

ahol

állományszám – annak az állománynak a száma, amelyre írni akarunk,
adatelemek – a kiírni kívánt adatelemek.

Az állományszám itt az a szám, amellyel a sornyomtatót megnyitottuk (2). Az adatelemeket ugyanúgy kell megadni, mint a képernyőre írás esetén.

A kiírás végén a gép és sornyomtató közötti kapcsolatot meg kell szakítani a CLOSE utasítással.

x CLOSE *állományszám*

ahol

CLOSE – az összeköttetést lezáró utasítás kulcsszava,
állományszám – az a szám, amellyel a kapcsolatot létrehoztuk.

Ezután a kódot megírhatjuk. Azt kell még meghatározni, hogy az oszlopokat milyen eljárással kell kiírni. Ez esetben a TAB(*n*) függvényt használjuk fel a kimutatás oszlopainak kiírásához. Az oszlopok 10 szóköznyi távolságban legyenek egymástól.

■ A *HT gépen* a nyomtatás az LPRINT utasítással elvégezhető. (Ismét megjegyezzük, hogy a nyomtató csak külön csatolóegységen keresztül kapcsolható a géphez.) Az utasítást ugyanúgy lehet használni, mint a PRINT utasítást. Az eltérés csak az utasítás hatásában van: a nyomtatás a sornyomtatón megy végbe. A kimutatás adatelemeit vesszővel választjuk el. Mivel a vessző hatására a gép az adatelemeket a 16 pozíció széles mezők kezdetére írja, az adatelemek között egy-egy vessző szükséges.

▲ A *PRIMO* esetében a nyomtatás ugyanúgy végezhető el, mint a HT esetében (lásd fent). Itt is használható az LPRINT utasítás, és az adatelemek elválasztását a TAB függvénnyel végezzük el a Commodore-hoz hasonlóan.

● A *Sinclair*-gépeknél is az LPRINT utasítással lehet elvégezni a nyomtatást a HT géphez hasonlóan. Az LPRINT utasítást ugyanúgy lehet alkalmazni, mint a PRINT utasítást.

A kimutatás adatelemeit a TAB függvény segítségével válasszuk el. Az egyes oszlopokra 16 pozíciót hagyjunk ki.

Ellenőrző kérdések és feladatok

1. Mi a BASIC nyelvben használható tömb előnye az egyedi adatnevekkel szemben? (Próbálja megoldani a 6. feladatot egyedi adatnevekkel!)

2. Rajzoljunk egy teljes szinuszgörbét a képernyőre! Az X tengely legyen függőleges, és a képernyő 16. oszlopán helyezkedjen el. Az X és Y tengely metszéspontja a képernyő tetején legyen. A rajzoláshoz 16 sort használjunk fel (1 sor = $\pi/16$), a függvényértékeket 14-szeresre nagyítsuk fel. A függvénypontokat csillaggal jelöljük.

3. Egészítsük ki a 6. feladatot azzal, hogy a kiírás végére kiírjuk a legkisebb és a legnagyobb mérési eredményt!

4. Állítsa össze ismerősei, barátai névsorát! Jegyezze meg mindenkinek a nevét és a testmagasságát. Készítsen egy programot, amely a névsort egyrészt ábécérendbe rakja, és kiírja a barátok nevét, valamint testmagasságát, másrészt a személyek nevét testmagasság szerint növekvő (vagy csökkenő) sorrendbe rendezi, és kiírja a neveket, valamint a testmagasságot. (A megoldáshoz használjuk fel azt az elvet, hogy szöveges változók közül az a „kisebb", amelynek kezdőbetűje az ábécében előbb van, mint a másiké.)

5. A 7. feladatot egészítsük ki azzal, hogy az azonos munkaszámokra való kifizetéseket összegezve írjuk ki:

   MUNKASZAM OSSZ: 99999.99

   formában. A kimutatás végén pedig a kifizetések összegét írjuk ki:

   OSSZES KIFIZETES: 99999.99

   formában.

11. JÁTÉKOK KÉSZÍTÉSE

A képernyőre „rajzolás" alapjai, véletlenszámok generálása.
A 8. feladat megoldása

A számítógépet nemcsak műszaki és gazdasági számítások vagy más adatfeldolgozási feladatok elvégzésére lehet felhasználni, hanem szórakoztató szerepet is játszhat. A számítógép mint „játszótárs" a kétszemélyes játékokban terjedt el. A mikroszámítógépek gyors térhódítása következtében a számítógépet mind nagyobb számban alkalmazták, és ez megnövelte a számítógéppel játszók táborát.

A játékokban interaktív használata miatt nagy szerepet kap a BASIC nyelv, és az is segíti a BASIC nyelvű játékok elterjedését, hogy nagyon sok „játékos kedvű" ember (egyetemi hallgató, kutató stb.) használja, aki napi feladatain kívül szívesen „elszórakozik" a számítógéppel. Ennek köszönhető, hogy a számítógépes játékok programozása és a BASIC nyelv összetartozó fogalmak.

A számítógép alkalmas játszótárs, mert interaktív (kérdést ír ki, a játékos válaszol), a játékos akcióira (lépéseire) gyorsan válaszol, statisztikával értékeli a játékos eredményeit, és végül: nem csal.

Nagyon sok számítógépes játék van forgalomban, és számuk egyre bővül. Programozásuknak néhány egyedi sajátossága közül az egyik leglényegesebb a nagyfokú interaktivitás, vagyis számos kérdés és válasz. A játékprogramok sok alternatívát kezelnek, és nagyon sok funkciót tartalmaznak, ami a program szerkezetét bonyolulttá teszi.

8. feladat

A közismert hadihajócsata-játék számítógépen játszható, módosított változatát kell elkészíteni. Most csak a számítógép jelölhet ki egy hajót egy 20×20 (a HT-nél és a PRIMO-nál 12×12, a Sinclair-gépeknél 18×18) négyzetből álló háló valamelyik pontjára, amelyet a játékosnak el kell találnia.

A játék **célja**, hogy a játékos minél kevesebb, de legfeljebb 15 (■ a HT-nél és ▲ a PRIMO-nál 10, ● a Sinclairnél 12) lövéssel eltalálja a hajókat.

A játékszabályok:

- a hajó helye véletlenszerű az adott csatamezőben;
- a hajó eltalálása győzelemnek számít;
- a hajó el nem találása sikertelenségnek számít;
- a képernyőn az 1–20.

(■ a HT-nél és ▲ a PRIMO-nál 1–12., ● a Sinclairnél 1–18.) sor és 1–20. (■ a HT-nél és ▲ a PRIMO-nál 1–12., ● a Sinclairnél 1–18.) oszlop által határolt területet kell csatatérnek tekinteni, és a lövés helyét itt meg kell jeleníteni egy csillaggal;

– a hajó és a becsapódás helye közötti távolságot a játékossal közölni kell a játéktér alatti 2. sorban.

A játék kezdetén ki kell írni a játékszabályokat a játékos számára:

EGY 20X20-AS (illetve 12X12-es vagy 18X18-as) CSATAMEZŐBEN KELL EGY ISMERETLEN HELYEN LÉVŐ HAJÓT ELTALÁLNI A CÉLPONT KOORDINÁTÁINAK MEGADÁSÁVAL. A PROGRAM A BECSAPÓDÁS ÉS A HAJÓ TÁVOLSÁGÁT KÖZLI: A HAJÓRA LEGFELJEBB 15 (illetve 10 vagy 12) LÖVEDÉKET LŐHET KI.

Ezután következik maga a vadászat. A játékos megadja az első becsapódás pontjának a koordinátáit az alábbi kérdésre:

AZ 1. LOVES KOORDINATAI (X, Y):

Az X, Y koordináta-rendszer 0,0 pontja a játéktér bal alsó sarkában van (77. ábra).

Ezután a játékos begépeli a lövés X, Y koordinátáit. Az ennek megfelelő ponton a csatatérben egy csillag karaktert kell kiírni, majd a program kiírja a becsapódási pont és a hajó közötti távolságot a játéktér alsó szélétől egy sorral lejjebb:

A BECSAPODAS TAVOLSAGA: X

Ha ez nulla, akkor a játékos eltalálta a hajót. Ezt külön is tudatni kell:

ON ELSULLYESZTETTE AZ ELLENSEG HAJOJAT.

A második lövés helye csak az első lövés valamelyik főirányú (fel, le, jobbra, balra) szomszédja lehet. Ugyanígy minden további lövés csak az előző lövés fő irányú szomszédja lehet. Minden lövés után továbbra is ki kell írni a becsapódás távolságát.

Mivel az első lövés után nem koordinátákat kell megadni, hanem a lövés mozgási irányát, a mozgatásra gombokat használunk fel. Úgy célszerű a négy iránynak megfelelő négy billentyűt kiválasztani, hogy azok utaljanak a mozgás irányára is. Ezért a gépeken az alábbi billentyűket válasszuk ki a mozgatásra:

	Commodore	HT	PRIMO	Sinclair
↑	P	P	Ö	I
←	L	L	Á	J
→	:	;	*	K
↓	.	.	>	M

Ha a 15. (illetve 10. vagy 12.) lövés sem talált, akkor a csata sikertelen volt. Ezt a játékossal közölni kell:

EZ NEM SIKERULT.
A HAJO AZ X, Y KOORDINATAKON ALLT.

A feladat elemzése

Ez a játék csak akkor élvezhető, ha az „ellenfél" (a számítógép) lépései kiismerhetetlenek, vagyis a játékos a korábbi játékok lefolyásából nem következtethet arra, hogy a program hol helyezi el a hajót a csatamezőben. A játékon belüli véletlenszerűség az RND véletlenszám-generáló függvénnyel érhető el.

A becsapódás helye és a hajó közötti távolság könnyen megállapítható, mivel minden pontot X és Y koordinátákkal adunk meg. Ügyelni kell azonban arra, hogy a program és a játékos is csak egész számmal jelölhet koordinátát. Ez különösen a találat érzékelésekor fontos.

A becsapódás és a lövés távolságát a koordináták segítségével lehet kiszámítani a 76. ábrán bemutatott ismert módszer alapján:

$$R = \sqrt{(X2-X1)^2 + (Y2-Y1)^2}$$

Az R-nek csak az egész részét szabad a játékossal közölni, mivel azonos egész részű R-ek esetében a törtrész értékéről az irányra lehet következtetni:

$$R = \text{int}\sqrt{(X2-X1)^2 + (Y2-Y1)^2}$$

Ha a törtrész 0, akkor vagy

$$X1=X2,$$

vagy

$$Y1=Y2.$$

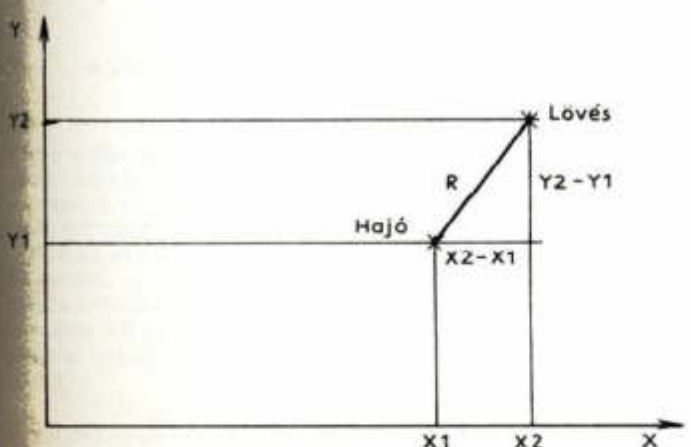
Tehát valamelyik koordinátát eltalálta a játékos. Minél nagyobb a törtrész, annál biztosabb, hogy a két koordináta eltérése megegyezik:

$$|X2-X1| \simeq |Y2-Y1|$$

A lövéseket számlálni kell, és a következő lövés sorszámát tájékoztatásul ki is kell írni. A 15. sikertelen lövés után a játékot be kell fejezni.

A lövés helyének megjelölése a képernyőn számítógéppel megoldható. A képernyő törlése után a helyőr a képernyő bal felső sarkában áll. Ha a játékos az

$$X = 5$$
$$Y = 17$$



76. ábra. A hajó és a becsapódás távolsága

lövést adta meg, akkor ez azt jelenti, hogy a helyőrt a 3. sor 5. pozíciójába kell állítani és ott egy csillag karaktert kinyomtatni. Azért kell így eljárni, mivel a mi szemléletünk szerint a koordináta-rendszer origója a játéktér bal alsó sarkában van (77. ábra). A helyőr ilyen jellegű mozgatását meg lehet valósítani mind a négy gépen. A Commodore-gépen a helyőrt úgy tudjuk a képernyő egy megadott pozíciójára állítani és ott valamilyen jelet kinyomtatni, hogy a PRINT utasításban idézőjelek közé helyőrt mozgató karaktert írunk be. Mivel előre nem tudjuk, hová kell állítani a helyőrt, mind a vízszintes, mind a függőleges mozgatást lépésenként kell végrehajtani a játékos által megadott koordinátáknak megfelelően. A fenti példában a helyőrt 5 lépésben kell 1–1 pozícióval jobbra tolni, és 3 lépésben kell 1–1 sorral lejjebb tolni. Ez sugallja, hogy a mozgatásra egy függőleges és egy vízszintes ciklust kell kialakítani, amelyeket annyiszor kell végrehajtani, ahányadik pozícióba vagy sorba kell a helyőrt mozgatni.

■ A HT, ▲ a PRIMO és ● a Sinclair-gépeken a képernyő bármely pontjára közvetlen címzéssel tudunk kiírni. Ez azt jelenti, hogy egyetlen utasításban meg lehet adni annak a pontnak a koordinátáit, ahová írni akarunk valamit.

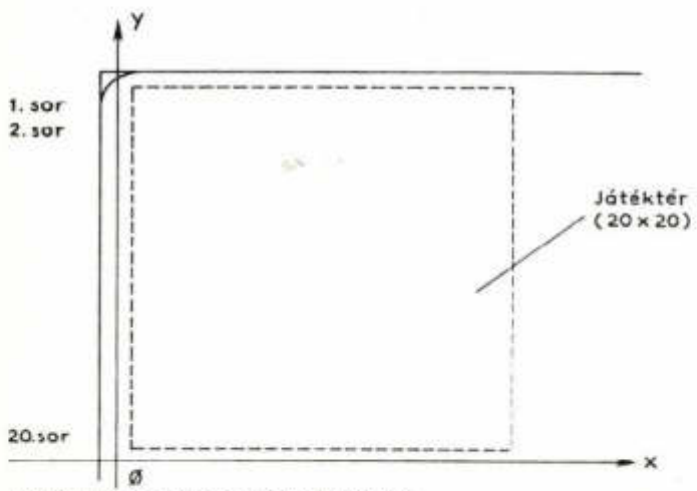
Felmerül a kérdés, hogy mi történjék akkor, ha a játékos bármely irányban eléri a játéktér szélét, és további lépést kísérel meg kifelé. Mivel a játéktér határai nem láthatók a képernyőn, nem tisztességes a kilépést megengedni vagy a lövésmódosítást letiltani. E helyett a „körben járást" végezzük el, vagyis ha a játékos bármely oldalon kilépne, akkor az ellenkező oldalon lépjen be ellenkező irányban a játéktérbe. Így a kilépés is hasznos lépés lehet.

A csillag kinyomtatása után a játéktér alatti 2. sorba kell menni, és ki kell írni a távolságot, majd még egy sorral lejjebb a következő lövés sorszámát.

A második lövéstől kezdve a játékos a 200. oldalon megadott gombok valamelyikével lő. A lövés beolvasására az INPUT utasítás nem használható, mivel a beolvasott adat a képernyőn megjelenne, s ezzel a kialakított képet megzavarná. Helyette más lehetőséget kell használnunk.

Először is azt kell tudni, hogy minden billentyűhöz egy kód tartozik (0–255 közötti szám), amely a billentyű lenyomásakor a tár meghatározott rekeszébe kerül, majd onnan a központi egységbe.

A leütött billentyű kódját (jelentését) egy utasítással ki lehet olvasni. Ezzel a



77. ábra. A játéktér pontjainak értelmezése

program számára meg lehet tudni, hogy a felhasználó melyik gombot nyomta be, és ezután mit kell tenni. A betűk és a speciális jelek gombjai esetében az elemzésben a kódot nem kell külön ismerni, elég csak a gombon levő karaktert megadni. Ha például azt kell figyelni, hogy a felhasználó A betűt ütött-e le, akkor meg kell vizsgálni, hogy a tár megfelelő helyén A betű van-e, és nem az A betűnek megfelelő kódszámot kell megadni (de azt is lehet). Erre a kiolvasásra használható a Commodore-nál a

GET

■ a HT gép, ▲ a PRIMO és ● a Sinclair-gépek esetében az

INKEY$

utasítás. Az utasítás formája:

x GET A$

■ A HT gép és ▲ a PRIMO esetében

x A$ = INKEY$

● a Sinclair-gépeknél pedig az INKEY$ önmagában állhat, mert nem tartozik hozzá az utasítástárgy.

Az utasítás hatására A$ felveszi a legutolsónak leütött billentyű kódját, egyben törli (nullát ír be) a tárnak azt a rekeszét, amelyből kiolvasta a kódot.

A GET, illetve az INKEY$ utasítás nem vár valamilyen billentyű leütésére, mint az INPUT utasítás, hanem az olvasás eredményétől függetlenül továbbmegy. Ha tehát azt akarjuk, hogy a program egy billentyű lenyomását megvárja a program egy adott pontján (pl. ez alapján kell eldönteni, hogy milyen műveletet kell a programnak végrehajtani), addig nem szabad továbbmenni, amíg ez meg nem történik. Ennek az a módja, hoy amíg nem ütünk le billentyűt, addig az olvasást meg kell ismételni:

```
20 GET A$
30 IF A$="" THEN 20
```

■ A HT gépnél és ▲ a PRIMO-nál:

```
20 A$=INKEY$
30 IF A$="" THEN 20
```

● A Sinclair-gépeknél:

```
20 IF INKEY$="" THEN 20
```

Vagyis a gép egy végtelen ciklust hajt végre mindaddig, amíg a beolvasott érték egy „üres" karakter, azaz nem nyomtunk le billentyűt. Ha bármelyik billentyűt lenyomjuk, a visszaugrás feltétele nem teljesül, ezért a program a következő utasítássoron folytatódik.

Példánk értelmében most azt kell megvizsgálni, hogy melyik gombot nyomta le a felhasználó, és ennek megfelelően melyik irányba kell mozgatni a kiírt karaktert. Mind a négy gépen azt kell vizsgálni, hogy melyik billentyű kódja jelenik meg az erre kijelölt A$ változóban. Ez pedig megegyezik a billentyűn levő jel (alsó állásban) kódjával, amit a jellel adhatunk meg. Eszerint a programnak az alábbi funkciói vannak:

– a játékszabályok kiírása és a hajó elhelyezése,
– a kezdőlövés koordinátáinak beolvasása,

– a lövések fogadása és értékelése
(2–15).
(■ HT-nél és ▲ PRIMO-nál 2–10., ● Sinclairnél 2–12.)
– a játék befejezése.

A feladat szerkezetét a 78. ábra mutatja.

A program tervezése

Vizsgáljuk meg az egyes funkciókat részletesen! A kezdőműveletek lényegében két lépésből állnak: ki kell írni a megadott játékszabályokat, és el kell helyezni a hajót a játéktér valamelyik pontjára. Tehát ezt a funkciót célszerű 2 modulra bontani (79. ábra).

A kezdőlövés két adat beolvasásából áll, ezért további bontása felesleges. Ezután és minden további lövés után a következő műveleteket kell elvégezni:

– ha a játékos nem az első lövést adta le, akkor várni kell a lövésre, és meg kell jegyezni a módosítás irányát;
– ki kell írni a képernyő megfelelő pontjára a csillagot;
– meg kell határozni a lövés és a hajó távolságát (értékelés);
– meg kell határozni a leadott lövések számát;
– a játéktér alatti 2. sorba ki kell írni a távolságot;
– a következő sorba ki kell írni a következő lövés sorszámát.

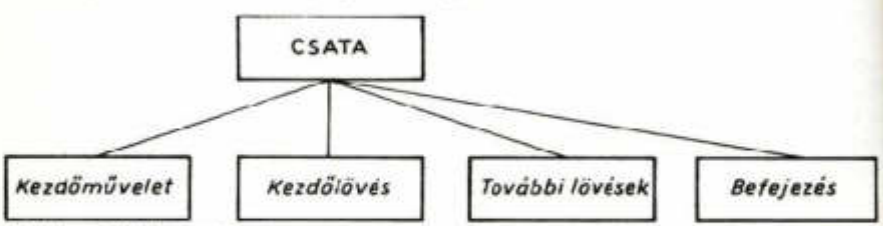
Az utolsó lépés után az elsőnek írt következik, és ez ismétlődik mindaddig, amíg a játékos vagy eltalálja a hajót, vagy leadta mind a 15 (illetve 10 vagy 12) lövést. Ezeket a lépéseket tehát ciklusban kell végrehajtani.

A ciklus változója ezek szerint a lövésszám lenne. A ciklusváltozót a **ciklusmag** végén ellenőrizzük, tehát a ciklus **hátul tesztelő** lesz. A ciklus befejezési feltétele akkor teljesül, ha a lövésszám már 15-nél nagyobb. A ciklus előkészítő művelete az első lövés leadása, amely induló értéket ad a lövési művelethez.

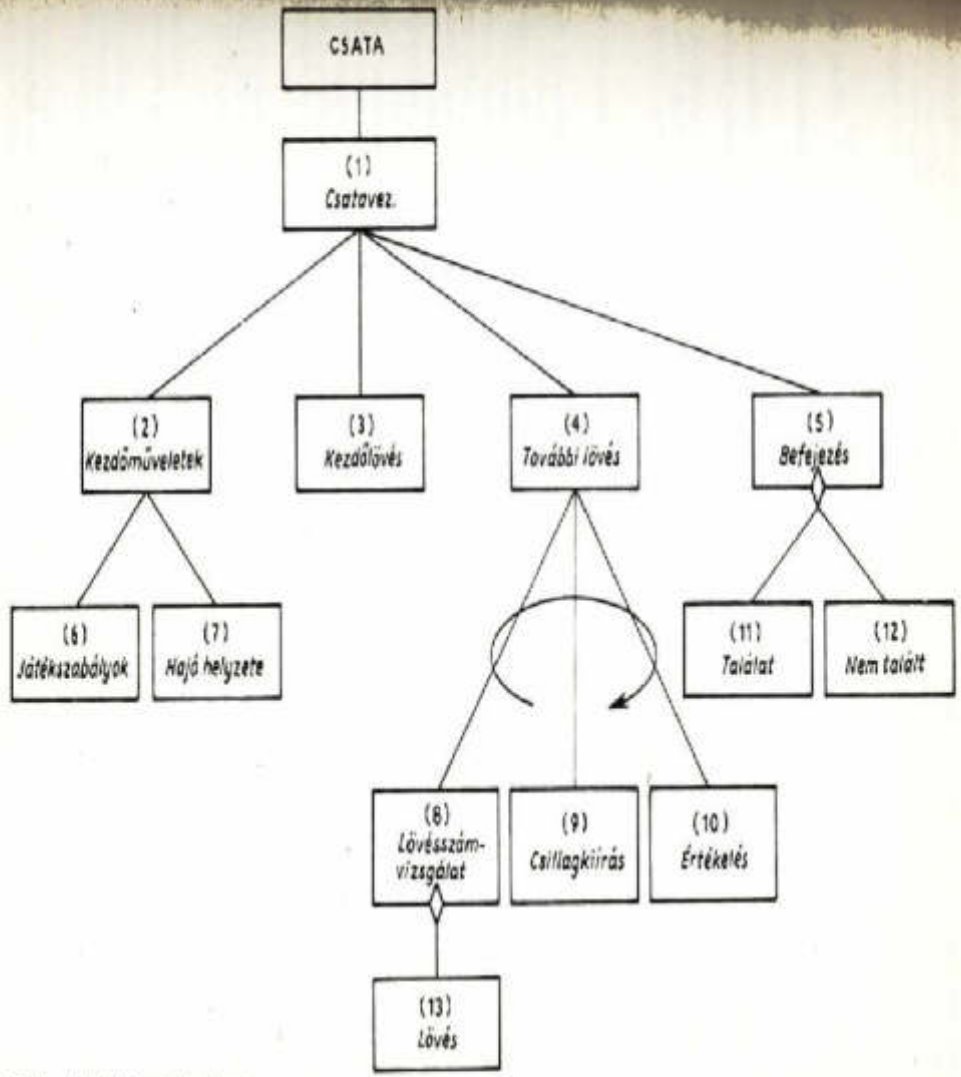
A ciklusból viszont akkor is ki kell lépni, ha találat következik be. Ezek szerint a ciklusmag végén nemcsak a lövésszámot kell ellenőrizni, hanem a találatot is. A ciklus **befejezési feltétele** tehát akkor teljesül, ha **vagy** minden lövést leadtunk, **vagy** találat következett be.

A további lövés részfeladatot ezzel négy modulra bontottuk. A (8), (9) és (10) modulok a lövési ciklus magját alkotják. A (13)-as modult csak akkor kell végrehajtani, ha nem a kezdőlövést adta le a játékos (vagylagos végrehajtás).

A *befejezésben* értékeljük a játékot. A következő sorba a találathoz tartozó vagy a sikertelen befejezés szövegét kell kiírni. A találat és a sikertelen vadászat kiírásait célszerű két külön modulba tárolni, és a (10) modulban elért eredménytől függően valamelyiket hívni. Ezután a program befejeződik.



78. ábra. A feladat szerkezete



79. ábra. A 8. feladat modulszerkezete

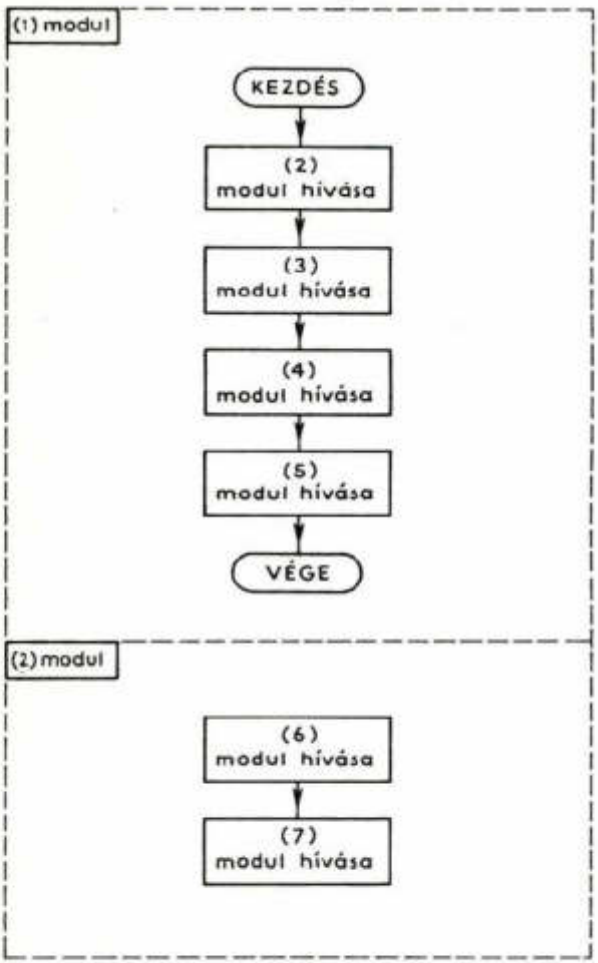
A programot a bemutatott programtervezési módszer szerint építjük fel (79. ábra). A programban ezek szerint összesen öt vezérlőmodul van: az (1), (2), (4), (5) és (8) modul.

A modulok tervezése

(1) *Csatavezérlés*
A modul **vezérlést** végez. Bemeneti adata nincs. Kimeneti adata a (2), (3), (4) és (5) modul hívása.

A modul a fenti négy modult szekvenciálisan hívja, utána a befejezésre tér át (80. ábra).

(2) *Kezdőművelet*
A modul **vezérlő** típusú. Bemeneti adata nincs, kimenete a (6) és (7) modul hívása. A modul a (6) és (7) modult hívja (80. ábra).



80. ábra. Az (1) és (2) modul folyamata

(3) *Kezdőlövés*
A modul **eljárás** típusú. Bemeneti adatai a játékos által begépelt X2, Y2 koordináták, és ezek képezik a modul kimenetét is.

A modul első művelete az X2 és Y2 beolvasása. Mindkettőt csak akkor lehet elfogadni, ha értékük az

(1,20),

■ HT-nél és ▲PRIMO-nál az

(1,12),

● Sinclair-gépnél pedig az

(1,18)

intervallumba esik. Ezt ellenőrizni kell, és hiba esetén a hibás adat nevét ki kell írni:

AZ X ERTEKE HIBAS, UJRA KEREM!

Ezután a beolvasást meg kell ismételni. A helyes értékű adatnak csak az egész értékét kell figyelembe venni, mert a találat nem lenne kimutatható, ha a játékos nem egész számot gépel be. Ebben a modulban kell a lövésszámlálást is elkezdeni az

$$L = 1$$

értékadással. A modul műveleteit a 81. ábra mutatja.

(4) *További lövés*
A modul **vezérlő** típusú. Bemeneti adata a lövésszám (2) és a találatjelző (T). Kimenete a (8), (9) és (10) modul hívása.

A modul a (8), (9) és (10) modult hívja mindaddig, amíg vagy találat nem következik be, vagy a játékos le nem adta az összes megengedett lövésszámot (Commodore-nál 15, ■ HT-nél és ▲ PRIMO-nál 10, ● Sinclair-gépeknél 12). Mint a tervezés során már megállapítottuk, a (8), (9) és (10) modulok egy ciklust alkotnak, amelyet ez a modul irányít. A ciklus hátul tesztelő, azaz a három hívott modul után kell vizsgálni a befejezés feltételét.

A ciklus akkor fejeződik be, ha a lövésszám elérte a megengedett maximumot (Commodore-nál 15, ■ HT-nél és ▲ PRIMO-nál 10, ● Sinclairnél 12), vagy találat következett be (T=1). A (13) lövés modulban minden lövés után meg kell növelni a lövések számát eggyel. Emiatt a lövésszámváltozó (L) már a következő lövés sorszámát tartalmazza. (Az utolsó megengedhető lövés után pedig ennél eggyel nagyobb számot.) Ezért az egyik befejezési feltétel a lövések maximuma +1 (82. ábra).

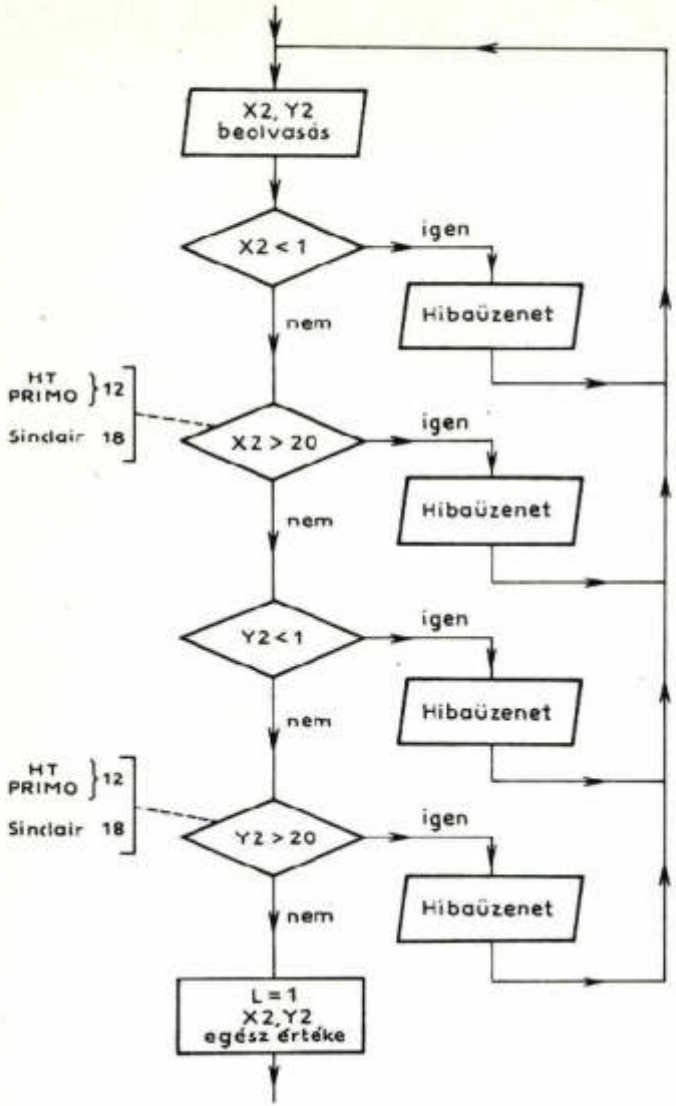
(5) *Befejezés*
Vezérlő típusú modul. Bemenete a T találatjelző értéke. Kimenete vagy a (11), vagy a (12) modul hívása. A modul a

$$T = 1$$

feltétel teljesülése esetén (találat következett be) a (11) modult, egyébként a (12) modult (nincs találat) hívja (82. ábra).

(6) *Játékszabályok*
A modul **eljárás** típusú. Kívülről nem kap adatot, kimenete a játékszabályok kiírása (83. ábra).

(7) *A hajó helyzete*
A modul **eljárás** típusú. Bemenete két véletlenszám a hajó helyzetének meghatározásához, kimeneti adata a hajó helye a csatatér területén.

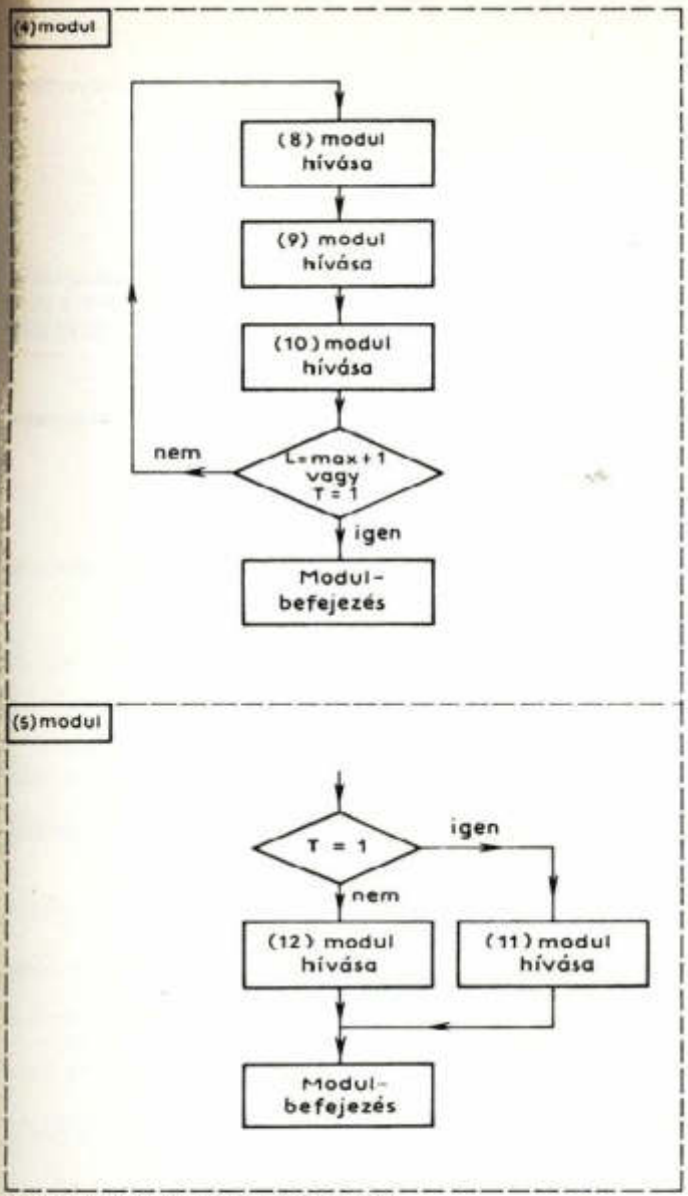


81. ábra. A (3) modul művelete

A hajó helyzetét a program az RND függvény segítségével véletlenszerűen úgy tudja kijelölni a játéktérben, hogy mind az X1, mind az Y1 koordinátát véletlenszám-generálással állítja elő. A Commodore-nál figyelembe kell venni, hogy az RND (0) utasítás csak a

$$[0,1]$$

intervallumba eső számot képes generálni. Ezért a kapott véletlenszámot meg kell szorozni 20-szal, és hozzá kell adni 1-et, hogy a kívánt intervallumba essen. Mindig



82. ábra. A (4) és (5) modulok folyamata

csak egész számú koordinátákat veszünk figyelembe, ezért a kapott értéknek csak az egész részét kell megtartani. Vagyis a koordináták kiszámítása:

```
X1 = INT (20*RND(0)) + 1
Y1 = INT (20*RND(0)) + 1
```

■ A HT gép esetében az

[1,12]

intervallumba eső véletlen egész számot kell generálni a hajó helyzetének meghatározásához. Ezt az

RND (12)

függvénnyel lehet elérni:

X1 = RND (12)
Y1 = RND (12)

Ha csak ezt a két utasítást alkalmaznánk, akkor a gép minden bekapcsolása után ugyanazt a véletlenszám-sorozatot állítaná elő. Így viszont a játék elveszítené érdekességét, mert megtanulható lenne. Ezért a koordináták meghatározása előtt egy

X RANDOM

utasítást kell megadni, amelynek eredményeként a gép minden alkalommal más véletlenszám-sorozatot állít elő egy játéksorozathoz.

▲ A PRIMO gépen a koordináták véletlenszámait szintén az

[1,12]

intervallumban kell előállítani. Ezt a feladatot az RND véletlenszám-generáló függvénnyel végezzük el:

X1 = RND(11)+1
Y1 = RND(11)+1

A játék teljes véletlenszerűségét itt is a RANDOM utasítás alkalmazásával lehet elérni a HT gépen való megoldáshoz hasonlóan.

● A Sinclair-gépeknél az

[1,18]

intervallumban kell a véletlenszerű koordinátákat előállítani. Itt is az RND véletlenszám-generáló függvényt alkalmazzuk:

X1 = 1+INT (RND *18)
Y1 = 1+INT (RND *18)

A teljes véletlenszerűséget ennél a gépnél a RANDOMIZE utasítás biztosítja, amelyet a véletlenszám-előállítások elé kell írni.

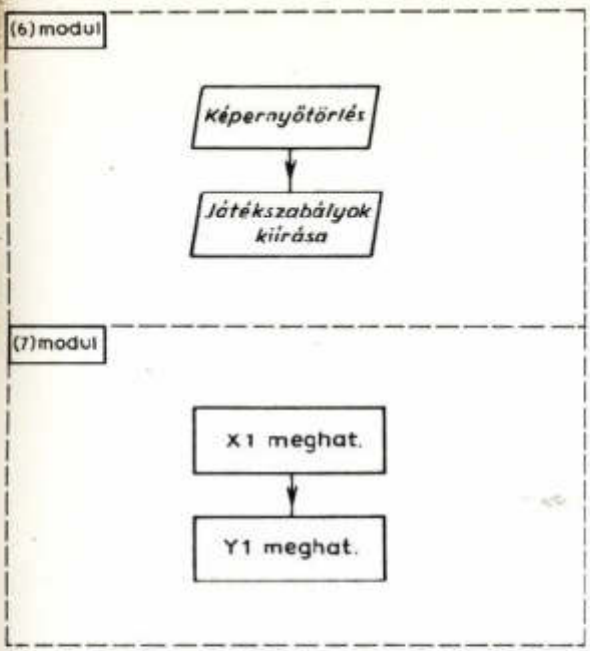
A modul műveleteit a 83. ábra mutatja.

(8) *Lövésszámvizsgálat*
A modul **vezérlést** lát el. Bemeneti adata a már leadott lövésszám (L). Kimenete a (13) modul hívása vagy nem hívása. Ha

$L > 1$,

akkor a (13) modult kell hívni, egyébként nem. A modul műveletei a 84. ábrán láthatók.

(9) *Csillagkiírás*
A modul **eljárás** típusú. Bemeneti adatai az X2 és Y2 koordináták, amelyek megmutatják, hová kell kiírni a csillagot. A modul kimenete a csillag megjelenítése a képernyőn.



83. ábra. A (6) és a (7) modul folyamata

A modult minden lövés után végre kell hajtani, mivel a csillag helyzete csak úgy módosítható, ha az egész kiírási műveletet megismételjük a módosított adattal.

A modul helyzetén a helyőrt a bal felső sarokba kell állítani, és a képernyőt törölni kell. Ezután a helyőrt a Commodore-nál:

$$Y3 = 20 - Y2$$

lépésben a megfelelő sorba kell állítani (az 1. sorban Y=20, a 20. sorban Y=1). Ha

$$Y2 = 20,$$

akkor a helyőrt az első sorban kell hagyni. Ilyenkor a lefelé mozgató ciklus műveleteit nem kell végrehajtani.

A sorbeállítás után a soron belüli helyzetet kell meghatározni. A helyőrt az adott soron belül az

$$X3 = X2$$

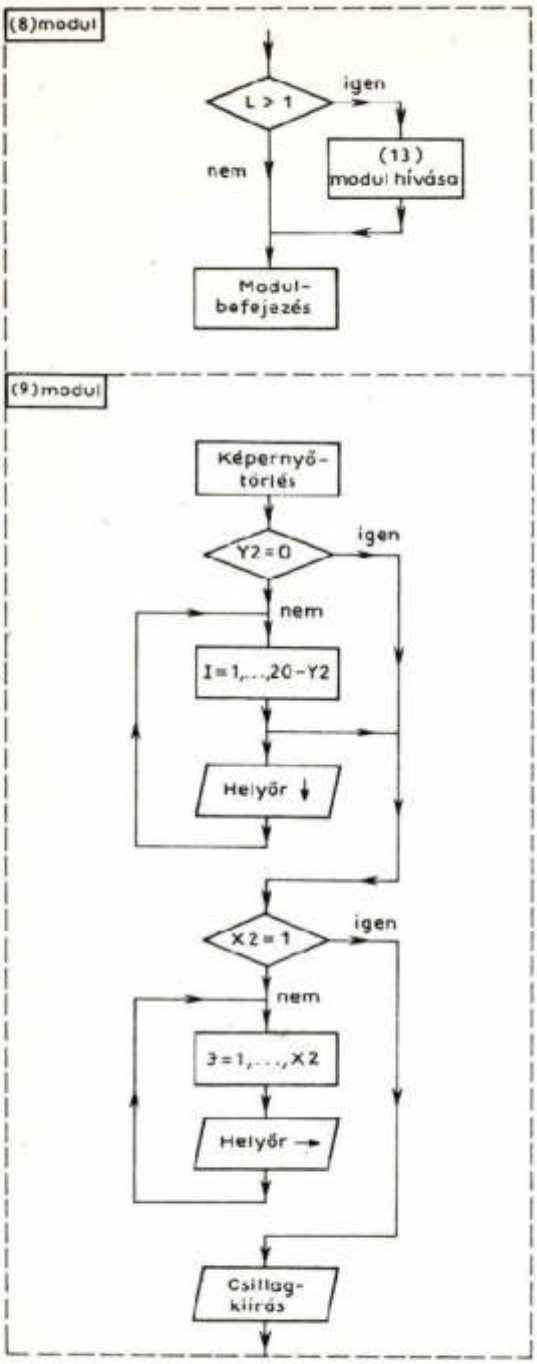
sorszámú oszlopba kell léptetni. Ehhez viszont csak X2−1 léptetést kell elvégezni, mivel a helyőr az 1. oszlopról indul. Ezt a műveletet is csak akkor kell végrehajtani, ha

$$X2 > 1.$$

Mindkét műveletet ciklusban kell végrehajtani. Mindkét ciklus magjában a helyőr 1−1 léptetése megy végbe.

A modul műveletei a 84. ábrán láthatók.

Magát a mozgatást a PRINT utasítás tárgyában levő helyőrmozgató karakterrel végezzük el.



84. ábra. A (8) és a (9) modul műveletei

■ A HT gépen a csillagot egyetlen utasítással ki lehet íratni, mivel a képernyő minden kiírási pontja közvetlenül címezhető az

x PRINT @ *mutató "szöveg"*

alakú utasítással. A PRINT @ a helyőrt állítja a mutató által meghatározott pontra a képernyőn, és itt kiírja a szöveget. A mutató

[0, 1023]

tartományba eső értéket vehet fel. A 0 a képernyő bal felső sarkát, az 1023 a jobb alsó sarkát jelöli. A szöveg tetszőleges BASIC szöveg lehet.

Az utasításban az okozza a kellemetlenséget, hogy a 16 sorra és 64 oszlopra felosztott képernyőn nekünk kell meghatározni a kiírási pozíciót.

Ha például az 5. sor 4. oszlopába akarunk valamit kiírni, akkor az első négy soron keresztül kell menni. Ez a

4×64=256

pozíció. De mivel a 0-ról indulunk, a 4. sor utolsó pozíciója a 255. Ehhez hozzá kell adni még 4-et (4. pozíció a sorban), ekkor a mutató értéke 259 lesz.

Az a kérdés, hogy tudjuk átszámolni a kezdőlövés X2 és Y2 koordinátáit egyetlen mutatóvá. Nézzük meg a 77. ábrát! A képernyő bal felső sarkában akkor jelenik meg a csillag, ha a mutató értéke 0. A lövés koordinátái viszont:

X2= 1
Y2=12

Ha X2-t eggyel növeljük, akkor a felső sor 2. pozíciójára kerül a csillag, amely mutató értéke 1. Azt látjuk tehát, hogy egy soron belüli pozíciót az

X2−1

értéke határoz meg. Ha

X2=1
Y2=11

(a második sor bal szélső pozíciója), akkor a mutató 64 (az első sorban a mutató 0–63 között változik). Ha egy sorral lejjebb megyünk, akkor a mutató értéke 64-gyel nő. Vagyis a soronkénti változás a

64 (12−Y2)

kifejezéssel írható le. Ezek az értékek különböző Y2-knél a sor bal szélén levő pozíciókat határozzák meg. A soron belüli helymegjelölés végett ehhez még hozzá kell adni az

X2−1

értéket. Ebből következően a mutató:

M=64(12−Y2)+X2−1

Ezzel a csillagkarakter már kiírható.

▲ A PRIMO gépen az

x PRINT$ *sorszám, oszlopszám, "szöveg"*

alakú utasítással a képernyő tetszés szerinti sorába és oszlopába a megadott szöveget ki lehet íratni.

A sorszám 0–15 közötti érték a képernyőn felülről lefelé haladva. Az oszlopszám pedig 0–41 közötti érték a képernyő bal szélétől jobbra növekedve. A sorszám

$$12-Y2$$

kifejezéssel számítható ki, mivel az utasításban levő sorszám a felső sorban 0, alatta 1, és így tovább. A játék legalsó sora viszont a 12. sor. Ezért kell a fenti különbséget számolni. Az oszlopszám (soron belüli pozíciószám) eggyel kisebb, mint az X2 koordináta (nulláról indul). Ezek szerint az

$$X2=10$$
$$Y2=6$$

lövéshez tartozó csillag az

x PRINT$ 2,5,"*"

utasítással írható ki.

- A Sinclair-gépeknél a képernyő megadott pozíciójára való kiírás elve megegyezik a PRIMO-nál bemutatottal, a különbség az utasítás formájában van:

x PRINT AT *sorszám, oszlopszám;"szöveg"*

A sorszám itt is 0-nál kezdődik, és 21-ig mehet. A 0 sorszámú a képernyő legfelső sorát jelenti. Az oszlopszám balról jobbra haladva 0-tól 31-ig nő. A lövés koordinátáit is hasonlóan lehet átszámítani sorszámmá és oszlopszámmá. A sorszám a

$$18-Y2$$

különbségből számítható. Az oszlopszám pedig az

$$X2-1$$

különbségből adódik.

Mindezek figyelembevételével az

$$Y2=13$$
$$X2=2$$

koordinátájú lövéshez a csillagot a következő tartalmú utasítással tudjuk kiíratni:

x PRINT AT 5,1;"*"

Megjegyezzük, hogy az AT függvény a TAB-hoz hasonlóan csak a PRINT utasítással együtt használható.

(10) *Értékelés*

A modul **eljárás** típusú. Bemeneti adatai a hajó koordinátái (X1, Y1) a (7) modulból, a lövés koordinátái (X2, Y2) a (3) vagy a (13) modulból. A modul kimenete a hajó és a becsapódás helye távolságának kiírása.

A távolságot a program terve szerint az

$$R = \text{int}\sqrt{(X2-X1)^2 + (Y2-Y1)^2}$$

képlettel kell kiszámítani. Ha

$$R=0,$$

akkor a találatjelző értéke

T=1

lesz, egyébként

T=0

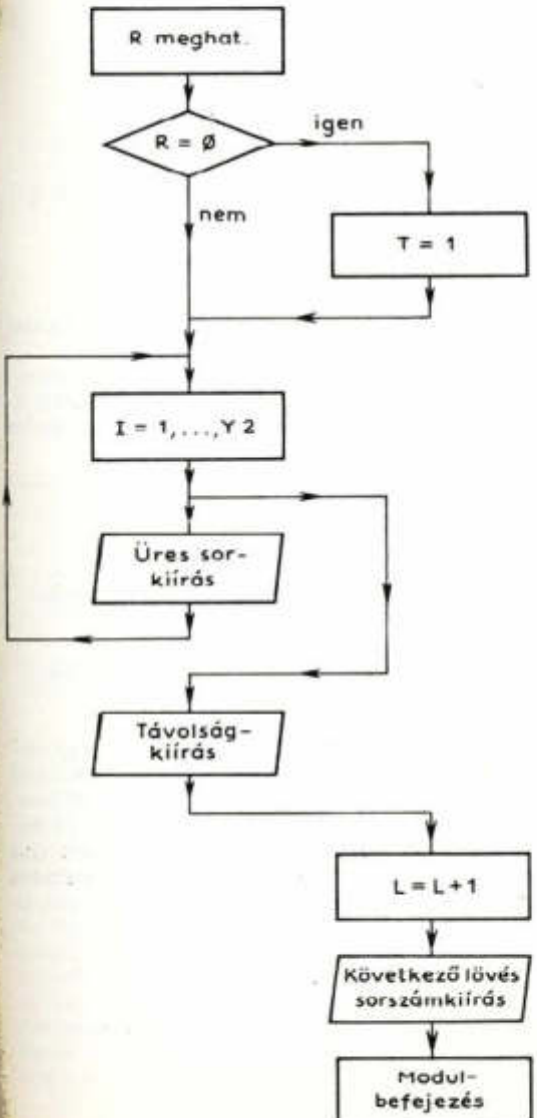
marad.

A távolság kiírása előtt az utolsó kiírás (csillag)

Y3=20–Y2+1

- a HT gép és ▲a PRIMO esetében az

Y3=12–Y2+1



85. ábra. A (10) modul folyamata

● a Sinclair-gépeknél az

Y3=18–Y2+1

sorban volt. (Ne tévesszük össze a kiírási sor számát a léptetés számával!)
Ebből következik, hogy

S=22–Y3=22–(20–Y2+1)–1=Y2

üres sort kell nyomtatni a csillag után (ekkor érünk a játéktér alatti 2. sorbal, mielőtt a távolságot kiírnánk. Az üres sorokat a már ismert ciklusos formában „írjuk ki". Itt kell növelni a lövések számát is (L=L+1), mivel ezt a modult minden lövés után végre kell hajtani. A távolság kiírása után a következő lövés sorszámát (L) is ki kell írni (85. ábra).

(11) *Találat*
A modul **eljárás** típusú. Bemeneti adata nincs. Kimenete a találat kiírása.
A modulban egyedül a találatot kell kiírni (86. ábra).

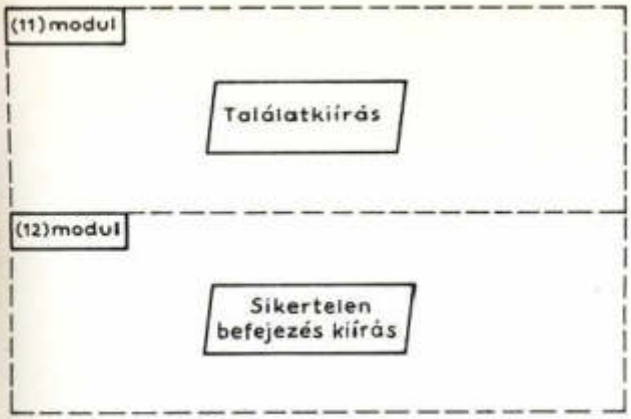
(12) *Nem talált*
A modul **eljárás** típusú. Bemeneti adatai a hajó helyzetének koordinátái (X1, Y1). Kimenete a sikertelenség kiírása.
A modulban csupán a sikertelen befejezés szövegét kell kiírni (86. ábra).

(13) *Lövés*
A modul **eljárás** típusú. Bemeneti adatai az előző lövés X2 és Y2 adata, valamint a játékos módosító lépése. Kimenete a módosított X2 és Y2 koordináta.
A modulban a játékos lépéseit kell várni. Műveletet csak akkor kell végrehajtani, ha a játékos lenyomja valamelyik lövésmódosító billentyűt. A billentyű lenyomásakor a billentyűtől függően más-más műveletet kell az alábbiak szerint elvégezni:

Lenyomott billentyű				Módosító koordináta
Commodore	■ HT	▲ PRIMO	● Sinclair	
P	P	Ö	I	Y2 növelése (sorszámcsökkentés)
L	L	Á	J	X2 csökkentése
:	;	*	K	X2 növelése
.	·	>	M	Y2 csökkentése (sorszámnövelés)

Ebből következik, hogy a billentyűlenyomás figyelési ciklusában mind a négy nyomógombot külön kell érzékelni, ezért ez egy CASE szerkezetet fog alkotni. A CASE szerkezet műveleteiben a fentiek szerint eggyel módosítani kell a kiválasztott koordinátát. Ezenkívül figyelni kell azt is, hogy a lövés nem esik-e a játéktéren kívülre.
Ha például X2=8 és Y2=1 (vagyis a csillag a játéktér legalsó sorában van), és a játékos a lefelé haladás gombját nyomja le (Y2-t tovább csökkenti, azaz a játékteret lefelé hagyná el), akkor a csillagnak a 8. oszlop tetején (1. sor) kell megjelennie, tehát Y2=20 (illetve 12 vagy 18) lesz.
A modul folyamata a 87. ábrán látható.

A Commodore-nál,
▲ a PRIMO-nál és ● a Sinclair-gépeknél a lövési koordináták módosítása közvetlenül elvégezhető az X2 és Y2 eggyel való csökkentésével vagy növelésével, illetve a „túlsó oldali" koordináta értékének megadásával.



86. ábra. A (11) és (12) modul folyamata

- A HT gépnél azonban ezt nem tudjuk közvetlenül elvégezni, mivel itt egyetlen számmal kell a lövés helyét megadni. Ha Y2-t növelni kell, azaz feljebb kell menni a képernyőn, akkor értékéből 64-et le kell vonni:

$$M = M-64$$

Ha Y2-t csökkenteni kell, akkor viszont hozzá kell adni 64-et:

$$M = M+64$$

Az X2 változtatása már csak eggyel módosítja az M értékét:

X2 csökkentése	M = M−1
X2 növelése	M = M+1

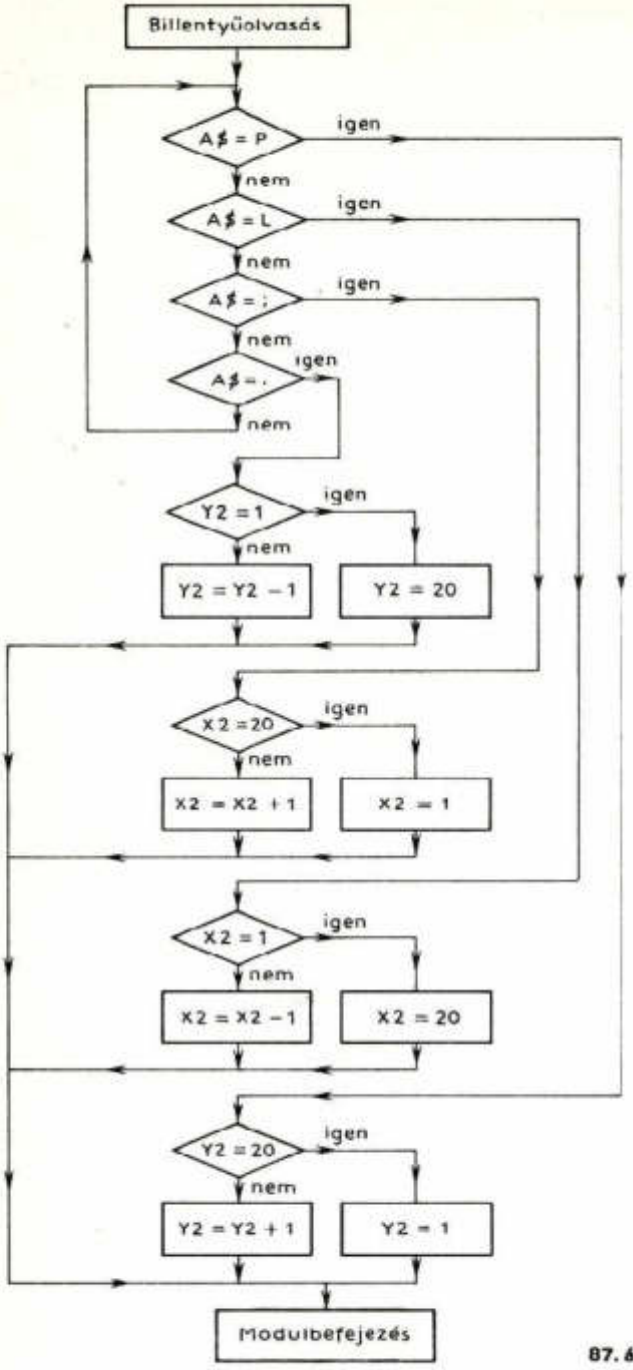
A HT gépnél a körben járás elég nehéz feladatot jelent, ezért ennek megoldásától eltekintünk.

Kódolás

A program kódjából néhány nevezetesebb részt bemutatunk. A *további lövés* modul vezérlési feladatot ellátó kódja:

```
510 GOSUB 850          (lövésszámvizsg.)
520 GOSUB 900          (csillagkiírás)
530 GOSUB 1100         (értékelés)
540 IF (L>15) OR T=1 THEN 560
550 GO TO 510
560 RETURN
```

Az első három utasítás a ciklusmag moduljait hívó utasítás. A negyedikben ellenőrizzük a ciklus befejezési feltételét. A ciklust a 15. lövés után, **vagy** találat esetén fejezzük be. Az IF THEN utasításban szerepel mind a két feltétel, ezeket az OR szóval kapcsoltuk össze. Ez azt jelenti, hogy ha a két feltétel közül bármelyik vagy mindkettő teljesül, akkor a THEN utáni rész hajtódik végre. Ha egyik sem teljesül, akkor a THEN utáni részt nem hajtja végre a gép. Ekkor az IF utasításban **logikai** feltételkapcsolatot vizsgálunk.



87. ábra. Az (7) modul folyamata

A csillagkiírást az alábbi ciklusok végzik el:

```
910 PRINT"■"                    képernyőtörlés
920 IF 20-Y2=0 THEN 960         a csillag a legfelső sorban marad,
                                nem kell lejjebb mozgatni
930 FOR I=1 TO 20-Y2
940 PRINT"■";                   a helyőr lefelé mozgatása
950 NEXT I
960 IF X2=1 THEN 1000           a csillag a bal szélső oszlopban
                                marad
970 FOR J=1 TO X2-1
980 PRINT"■";                   a helyőr jobbra mozgatása
990 NEXT J
1000 PRINT "*"
1010 RETURN
```

A 940-es sorban a PRINT utasításkulcsszó után egy inverz Q áll, amely a helyőrt lefelé mozgató billentyű leütésével írható be. Ugyanígy a 980-as sorban a PRINT után inverz szögletes záró zárójel áll, amely a helyőrt jobbra mozgató billentyű leütésével írható be.

Végül nézzük meg a mozgató billentyűk lenyomásának figyelését:

```
1410 GET A$
1415 IF A$="" THEN 1410          billentyűolvasás
1420 IF A$="P" THEN 1470         P gomb (felfelé)
1430 IF A$="L" THEN 1510         L gomb (balra)
1440 IF A$=":" THEN 1550         : gomb (lefelé)
1450 IF A$="." THEN 1590         . gomb (jobbra)
1460 GOTO 1410
1470 REM* F1 GOMB *              vissza az olvasáshoz,
                                 ha nem mozgató-
                                 gombot nyomtunk le
```

Ellenőrző kérdések és feladatok

1. Hogyan lehet még a csillagot kiírni a lövés helyére?

2. Alakítsa át úgy a 8. feladatot, hogy ha a játékos eléri a csatatér szélét, és ki akarna lépni, akkor a kifelé mutató lövést ne fogadja el! (Még szebb a megoldás, ha a játékos valamilyen figyelmeztetést is kap erről.)

3. Készítsen „fej vagy írás" (pénzdobálós) játékot úgy, hogy a gép „dobja fel" a pénzt! Lehessen fogadni is!

4. Készítsen lottójátékot, amelyben a gép „húzza ki" az öt számot, és kiírja a képernyőre!

5. Készítsen huszonegyező játékot, amelyban a gép a bankos (osztó), és az egyetlen játékos a felhasználó! Egyszerű változatában „pénz nélkül" (ütési lehetőség nélkül) játsszanak, de készítsünk olyan megoldást is, amelyben ütni is lehet!

12. TÖBB PROGRAMBÓL ÁLLÓ SZERKEZETEK

Miért kell egy feladatot több programmal megoldani?
Mi a feltétele a több programos szerkezetnek?
Néhány alapszerkezet

Egy feladat nem mindig oldható meg egyetlen programmal. Ennek egyik oka az lehet, hogy a feladat túl hosszú programot eredményezne, ezért inkább több különálló programból építjük fel. Egy másik ok pedig, hogy a feladat egy részére már van alkalmas kész program, így csak a hiányzó részekhez kell programo(ka)t írni.

Több programos szerkezet főleg csak lemezes tárolóval rendelkező gépnél valósítható meg, mivel a kazettáról való betöltés egy programnál nem jelent akadályt, de több program bemásolása már rendkívül nehézkessé tenné a programok használatát.

A **több programos szerkezet** jelentősen eltér az egyprogramos szerkezettől. Az egyprogramos megoldásnál a program teljes egészében a tárban van, így minden modul (vagy szubrutin) szabadon hívható. Több programos szerkezetnél viszont a programrendszernek mindig csak egyetlen programja – az aktuálisan futó program – tartózkodik a tárban. Így az sem lehetséges, hogy egy vezérlő-(fő-)program állandóan a tárban legyen, hogy a többi program futását irányítsa.

A programcserének az a következménye, hogy a bemásolás előtt a bent lévő program törlődik, és aztán csak az új marad bent a tárban. Az adatokat ilyenkor általában nem törli a gép, így a következő program elvileg használhatja az előző program adatait. Az előző program adatai akkor használhatók teljes biztonsággal, ha a második program rövidebb az elsőnél (kevesebb utasításból és adatterületből áll). Ez a feltétel viszont nem mindig teljesíthető.

De az sem jelent lényeges akadályt, ha a két program nem tud egymással kommunikálni a tárban tárolt adatok révén. Ilyenkor még mindig fennáll annak a lehetősége, hogy külső tárolóeszközön tároljuk (lásd a 13. részben) a közösen használt adatokat.

Egy programból egy másik programot a Commodore–64 esetében a LOAD utasítással lehet hívni:

x LOAD *"programnév"* ,8

ahol

programnév – a hívott program neve, amilyen néven a katalógusokban szerepel,
8 – a lemezegység sorszáma.

Az utasítás hatására a jelenleg futó program a tárból törlődik (tovább nem használható), a gép behívja a megadott nevű programot, és futtatni kezdi az első sortól kezdve. A felhasználó lényegében nem is veszi észre, hogy programcsere történt. Hogy elkerüljük az adatok megőrzésével kapcsolatos bizonytalanságot, a program bemásolása előtt, az előző program végén CLR utasítással törölni kell a változók tartalmát. Az utasítás formája:

x CLR

A törlőutasításnak nincs tárgya, de felesleges is, mivel valamennyi változó tartalmát nullázza. Így a programcsere kifogástalanul működik ugyan, de a programok közötti adatcsere megszűnik. Ez adott esetben a programrendszer működését is lehetetlenné teheti.

A probléma szerencsére nem megoldhatatlan. Egy kis többletráfordítás árán átmenthetjük a változók értékeit a programcsere és a törlés ellenére. Ezekkel az eljárásokkal itt nem foglalkozunk.

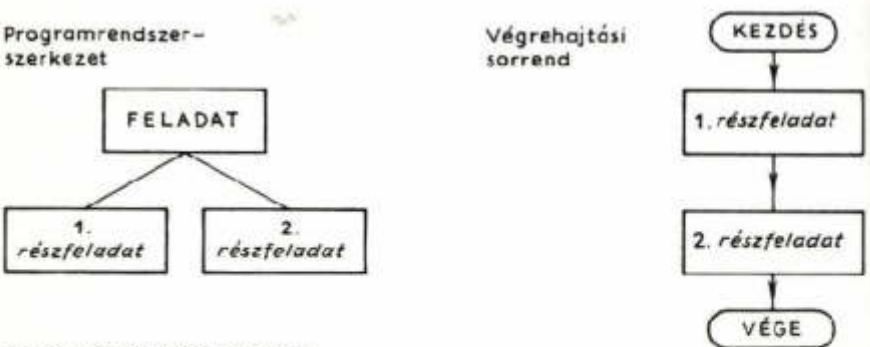
Vigyázzunk! Több programos rendszerben hiba esetén ne felejtsük el megnézni, hogy melyik program futott éppen (pl. listázással)!

A több programos szerkezetek logikai felépítése ugyanolyan, mint más programoké, de a kódolásukban vannak eltérések. Nézzük meg a leggyakrabban előforduló szerkezeteket!

SZEKVENCIÁLIS PROGRAMOK

A legegyszerűbb eset. A programrendszerben lévő programok valamilyen sorrendben hívják egymást (88. ábra).

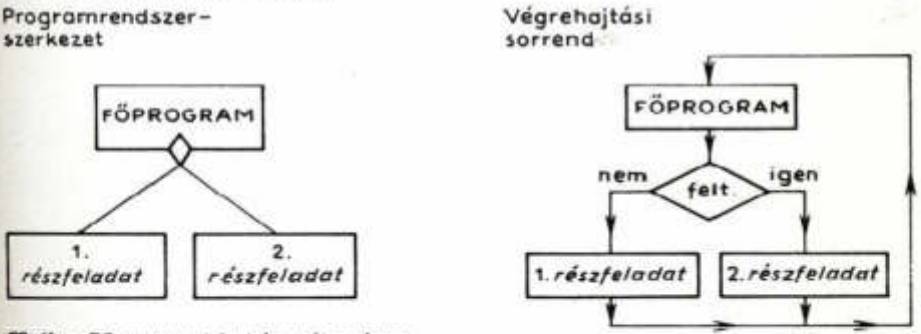
A programrendszer végrehajtása az 1. részfeladatot megvalósító programmal kezdődik, majd ez hívja a 2. részfeladatot megoldó programot. Ennek végrehajtása után a futás befejeződik. Kód szempontjából annyi az újdonság, hogy az 1. részfeladat programjának a végén (az END előtt) egy LOAD utasítást kell elhelyezni.



88. ábra. Szekvenciális programok

FŐPROGRAMOT TARTALMAZÓ SZERKEZETEK

Vannak olyan feladatok, amelyekben több, egymástól többé-kevésbé független funkciót kell ellátni. Az olyan esetekben, amikor a funkciók eléggé összetettek ahhoz, hogy mindegyiket önálló programmal valósítsuk meg, vagy a funkciók hívási sorrendje nem kötött, hanem a felhasználón múlik, akkor szükségünk van egy olyan programra, amely az egyes funkciók programjait hívja. Ez a **főprogram.** A főprogram vezérlési funkciót lát el. Valamilyen algoritmus szerint kiválasztja, hogy melyik programot kell behívni, majd ezt behívja. A program lefut, és ezzel feladatát befejezi. Ekkor a vezérlést vissza kell adni a főprogramnak, hogy az újabb funkciók végrehajtását vezérelhesse. Minden funkció programjának végén ezért el kell helyezni a főprogramot hívó LOAD utasítást (89. ábra).



89. ábra. Főprogramot tartalmazó szerkezet

A főprogram ismét lehetőséget kap, hogy újabb feladat végrehajtását vezérelje. Ez a fajta vezérlési szerkezet ott használatos, ahol a felhasználó dönti el, hogy milyen funkciót kell végrehajtani. Például egy raktári készletnyilvántartó rendszer funkciói az anyagbevételezés, az anyagkiadás, a készletkiírás stb. lehetnek. A részfeladatot a raktáros jelöli ki az igények alapján. Ilyenkor a főprogram egy „menüt" ír ki, hogy a felhasználó láthassa, milyen műveletek közül választhat. Látható, hogy a főprogram a főmodulhoz hasonló szerepet tölt be, de önálló programként.

13. ADATÁLLOMÁNYOK ÉS LÉTREHOZÁSUK

Az adatállományok jellemzői.
A 9. feladat LETRE nevű létrehozó programjának elkészítése

Nagy mennyiségű összetartozó adat tárolására két lehetőség van:

– programon belüli,
– programon kívüli

adattárolás. Az előző módszer szerint néhány száz egyed adatait lehet tárolni tömbben. A programon kívüli tárolás elvileg végtelen mennyiségű adat tárolására ad lehetőséget. Tényleges korlátot a rendelkezésre álló tárolóeszköz adattárolási kapacitása jelent.

Adatokat programtól függetlenül **adatállományokban** lehet tárolni. Az adatállomány valamilyen rendezettségben tárolt adatok összessége.

Az adatállomány egyik legfontosabb jellemzője, hogy a **programtól függetlenül** létezik. Ez azt jelenti, hogy meglévő adatállomány használatára bármikor készíthetünk újabb programokat, vagy már meglévőket törölhetünk. Mindezek a műveletek az adatállományt érintetlenül hagyják. Az adatállomány önálló létéből következik az is, hogy **saját neve** van. Ezt az adatállomány azonosítása teszi szükségessé. Ha ugyanis egy program egy adatállományt használni akar meg kell nevezni, hogy milyen adatokon kíván műveleteket elvégezni. A név azért is szükséges, hogy több adatállomány közül ki lehessen választani a szükségeset. Az adatállományok nevének felépítése gépenként különbözik. A Commodore–64 esetében az adatállománynév 1–16 karakter hosszúságú szöveges változó lehet.

Az adatállományok **rekordokból** épülnek fel. Egy rekord egy egyed tulajdonságainak értékeit tartalmazza egy mátrixsorhoz hasonlóan. A rekordon belüli adatok (egyedek tulajdonságainak értéke) neve itt a *mező*. Ezenkívül olyan mezőket is használunk, amelyek valamilyen adattárolási információt tartalmaznak, az egyedekhez pedig semmi közük. Ilyen adat például a rekord sorszáma. A mátrixos adattároláshoz hasonlóan a mezők sorrendje kötött a rekordon belül, hogy minden rekordot azonos elvek szerint lehessen feldolgozni.

Az adatállományok külső tárolóegységen kapnak helyet, vagyis a táron kívül vannak. Az adatállományok kezelésénél a számítástechnikában az a gyakorlat alakult ki, hogy az adatállományoknak csak egy vagy néhány rekordja van egy-egy alkalommal a gép tárjában, ha a re-

kordok adatai egymástól függetlenül feldolgozhatók. Ez a módszer azt eredményezi, hogy a feldolgozásban részt vevő adatok csak kevés helyet foglalnak el a tárban.

Mielőtt azonban egy adatállomány rekordjaival bármilyen műveletet elvégezhetnénk, a gép és az adatállomány között kapcsolatot kell létrehozni. Ezt a kapcsolatlétrehozást nevezzük az adatállomány **megnyitásának.**

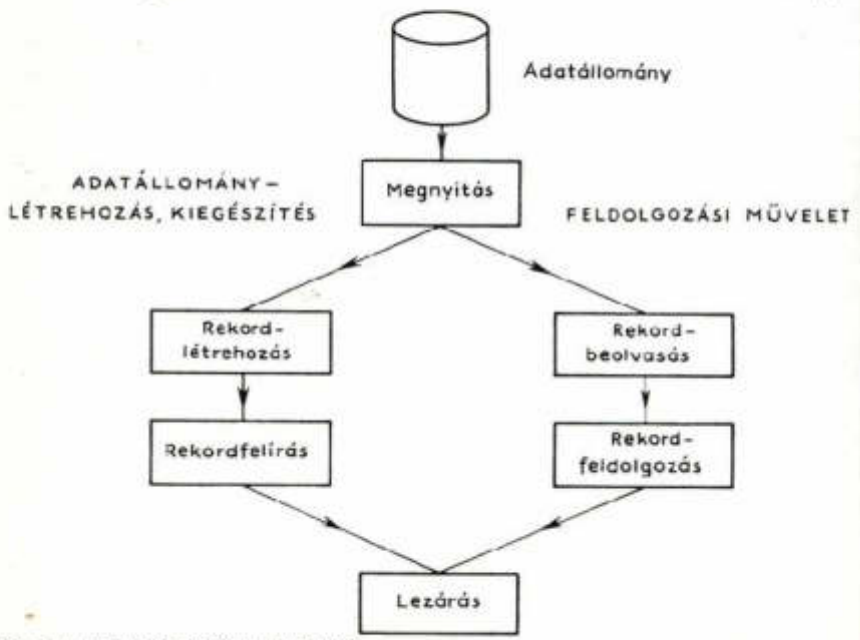
Csak ezután tudjuk a rekordokat a tárba beolvasni és adataival a műveleteket elvégezni. Az adatállomány feldolgozásának végén a gép és az állomány közötti kapcsolatot meg kell szüntetni, vagyis az adatállományt le kell zárni. Az adatállomány feldolgozásának logikai menetét a 90. ábra mutatja.

Az adatállományokkal kapcsolatos műveleteket három csoportra lehet felosztani:

– létrehozás
– feldolgozás
– módosítás

Logikailag az első művelet az adatállomány **létrehozása.** Ennek során a tárban létrehozott rekordokat a gép a külső adattároló berendezésre másolja.

A **feldolgozás** során a rekordokban tárolt adatokat valamilyen



90. ábra. Műveletek adatállományokkal

műveletsorozat (például összesítés) elvégzésére használjuk fel. Ekkor a rekordokat a háttértárolóból be kell olvasni a gép tárjába, és ott kell elvégezni a feldolgozás lépéseit. A feldolgozás nem változtatja meg az adatállomány tartalmát.

A harmadik műveletcsoportba tartoznak az olyan műveletek, amelyek az adatállomány tartalmát megváltoztatják. Ilyen művelet az adatállomány **kiegészítése**, a rekordok **törlése** vagy a rekordok **tartalmának megváltoztatása.**

Az adatállományon belül a rekordok valamilyen szempont szerint sorba rendezhetők a feldolgozás egyszerűsítésére. A rendezett sorrend bizonyos szempontból megkönnyíti a feldolgozást.

Ilyen rendezett esetekkel itt nem foglalkozunk. Az egyszerűség kedvéért csak olyan adatállomány-feldolgozási műveleteket fogunk megismerni, amelyekben a rekordok sorrendje tetszőleges. Ha a rekordok rendezetlenek, és csak a rekordok egy bizonyos csoportján kell valamilyen műveletet elvégezni, akkor is minden rekordot be kell olvasni, hogy megvizsgáljuk, el kell-e végezni a műveletet rajta. Az ilyen adatállományokat **szekvenciális** adatállományoknak nevezzük. Ezek legfőbb jellemzője, hogy feldolgozásuk esetén valamennyi rekordot be kell olvasni. Ez a tulajdonság nem jelent hátrányt akkor, ha valamennyi rekordot fel **kell** dolgozni. Például, ha az adatállomány egy vállalat dolgozóinak bérelszámolási adatait tartalmazza, és a hó végén valamennyi dolgozó adatait fel kell dolgozni a bérük meghatározásához. Ekkor a rekordok tárolási sorrendje közömbös. De ha az a feladat, hogy ugyanebben az adatállományban egyetlen dolgozó béradatát keressük meg, akkor a tárolás rendezettsége komoly segítséget jelenthet a kiválasztott rekord megtalálásában. A rendezetlenség sok felesleges beolvasást okozhat, ami időigényes.

Az adatállománynak még két fontos tulajdonsága: **kezdete** és **vége** van. Szekvenciális adatállományok feldolgozása az állomány elején (az első rekordnál) kezdődik, és az állomány utolsó rekordjánál fejeződik be.

AZ ADATÁLLOMÁNYOK LÉTREHOZÁSA

Az eddig leírtak értelmében adatállományok a következő lépésekben hozhatók létre:

- megnyitás (kapcsolatlétesítés),
- adatállományra írás,
- lezárás.

Létrehozásnál mindig egy **új** adatállományt kell megnyitni írásra. Erre szolgál az OPEN utasítás:

x OPEN *i*, 8, *j*, "0: *név*, S, W"

ahol

i	– az adatállomány logikai hivatkozási száma,
8	– a lemezegység száma,
j	– a csatornaszám (amelyen a kapcsolatot létrehozzuk),
név	– az adatállomány neve,
S	– a szekvenciális jelleg rövidítése,
W	– utalás az írásra való megnyitásra.

Az utasítás hatására a program megnyit egy új adatállományt, amelynek neve a megadott név lesz. Az adatállomány lemezen jön létre, mivel a 8 és 9 egységszám lemezegységet jelent. A program további részében az adatállományra nem a nevével, hanem a logikai hivatkozási számával (*i*) hivatkozunk. A logikai hivatkozási szám a változónévhez hasonlóan egyedi, több állományhoz különböző hivatkozási számokat kell megadni. Ez leegyszerűsíti az írási és a lezárási műveletet.

A csatornaszám tetszőlegesen 2–14 lehet. A mi szempontunkból egyszerű, ha a logikai hivatkozási számmal megegyező csatornaszámot használunk. Az ilyen utasítással létrehozott adatállományra csak írhatunk, más művelet nem végezhető.

A megnyított adatállományra a következő utasítással írhatunk:

x PRINT# *i*, A ; "," ; B ; . . .

Ez a már ismert PRINT# utasítás, amikor is az utasítás tárgyában lévő adatokat nem a képernyőre kell kiírni, hanem a # jel után megadott logikai hivatkozási számú adatállományra. Az utasítás tárgyában a rekord mezőit (a rekordot alkotó adatok) kell felsorolni, és közéjük a vesszőt (mező-elválasztójel) idézőjelbe írva kell az adatállományra kiírni. Egy utasítással egy rekordot lehet az adatállományra felírni. Az állomány létrehozása után az állományt az

x CLOSE *i*

utasítással le kell zárni. Ezután az adatállományon további művelet nem végezhető, csak egy újabb megnyitás esetén. Az utasításban szereplő *i* az adatállomány logikai hivatkozási száma.

9. feladat

Egy raktár anyagnyilvántartását kell számítógépen elvégezni. A számítógépes rendszernek az alábbi szolgáltatásokat kell nyújtania:

- kívánságra ki kell írni a raktárban lévő anyagok készletét (dátum, azonosító, anyagnév, mennyiségi egység, mennyiség, elszámolóár, érték) a nyomtatón;
- anyag be- és kivételezés esetén a változásokat át kell vezetni a tárolt készletadatokon, és anyagonként be- és kivétjegyet kell előállítani (dátum, azonosító, anyagnév, ki- vagy bevett mennyiség).

A raktárban 500-féle anyag van, Az anyagok azonosítója ötjegyű szám, a megnevezés legfeljebb 20 karakterből áll. A mennyiség egész szám, legnagyobb értéke 99999. Az anyagok elszámolóára 1000 Ft alatt van, fillért nem tartalmaz.

A feladat elemzése

A feladat értelmében elkészítendő rendszernek egy raktár anyagnyilvántartásával kapcsolatos szolgáltatásokat kell elvégeznie. A rendszernek huzamosabb ideig kell szolgáltatásait nyújtania a raktári munka rendjének megfelelően. Az egyes szolgáltatások (készletjelentés, bevét, kivét) igénybevételének sorrendje, száma tetszőleges lehet a felhasználó kívánságának megfelelően.

A feladat elvégzéséhez 500-féle anyag adatait kell tárolni. A készletjelentés és a mozgásokat tartalmazó bizonylatok elkészítéséhez a következő adatok szükségesek:

- dátum,
- anyagazonosító,
- anyagnév,
- mennyiségi egység,
- mennyiség,
- elszámolóár,
- érték,
- ki- vagy bevett mennyiség.

Nézzük meg, mit kell tenni azért, hogy ezek az adatok a programrendszer rendelkezésére álljanak!

A dátum aktuális adat, amely minden szolgáltatáshoz szükséges. A kiírások előtt be kell olvasni, mivel kiszámítását automatizálni nem lehet. Tárolása csak az éppen futó programon belül oldható meg.

A többi adatról könnyű felismerni, hogy egy egyedtípus tulajdonságai. Az egyedek a raktárban tárolt anyagféleségek. A rendszerben az egyedek alábbi tulajdonságainak értékeit használjuk:

- anyagazonosító,
- anyagnév,
- mennyiségi egység,
- mennyiség,
- elszámolóár,
- érték,
- ki- vagy bevett mennyiség.

Az egyes anyagokra vonatkozóan az alábbi adatokat kell biztosítani: az ötjegyű szám (ciklusszám), amely az anyagok azonosítását szolgálja. Az anyagnév nem biztonságos azonosító, mivel szöveges adat, és ugyanazt az anyagot többféleképpen lehet megnevezni. A kiíráshoz viszont az anyagnév szükséges, mivel ez „emberibb adat", mint az azonosítószám. Szükség van még a mennyiségi egység (KG, DB, M) tárolására és természetesen az aktuális mennyiségre. Az anyagok készletértékének meghatározásához a mennyiség mellett szükség van az elszámolóárra (egységárra) is. Ebből a két adatból a készletérték bármikor kiszámítható, ezért tárolása szükségtelen. A ki- és bevett mennyiség egy anyagmozgáshoz kapcsolódik, amelyről azonnal bizonylatot kell készíteni. Erre az adatra a továbbiakban nincs szükség, ezért a bizonylat elkészítése után törölhető. Ezzel az adattal kell a készletmennyiséget is módosítani minden mozgás után.

Ebből következik, hogy hosszú távon a következő adatokat kell tárolni minden anyaghoz:

- azonosító (5 bájt),
- név (20 karakter),
- mennyiségi egység (2 karakter),
- mennyiség (5 bájt),
- elszámolóár (5 bájt).

(A numerikus adatok tárolása egységesen 5 bájtot igényel a megadott nagyságrendű számok esetében.) Ez anyagonként 37 bájt, összesen pedig 18 500 bájt. Ez az adatmennyiség programon belül nem tárolható, ezért az adatokat adatállományban kell elhelyezni. Ezt támasztja alá az a jogos igény is, hogy a rendszernek huzamosabb ideig tárolnia kell az adatokat a gép többszöri ki- és bekapcsolása mellett.

A rekordokon belül az adatokat a felsorolás szerint tárolhatjuk. Ezek alkotják a rekord mezőit. Az adatállomány neve legyen ANYAG.

Hogyan kell működnie a rendszernek? A rendszer 3-féle szolgáltatást nyújt:

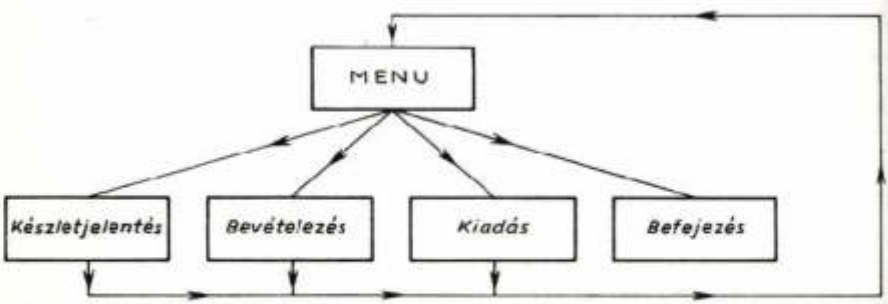
- készletjelentés-készítés,
- bevételezés,
- kiadás.

Az egyes szolgáltatások sorrendje tetszőleges, és egymástól függetlenül bármikor igénybe vehetők. Ez két megoldási lehetőséget sugall. Annyi programot kell készíteni, ahány szolgáltatás van, és azt kell futtatni, amelynek szolgáltatására szükség van. Ez meglehetősen nehézkes. A másik megoldás, hogy egyetlen programot kell elindítani, amely a felhasználónak egy menüben kínálja a szolgáltatásokat. Ennek leggyakoribb formája, hogy a program a képernyőre kiírja az összes szolgáltatást, és a szolgáltatások elé vagy utánuk egy számot is kiír, jelezve, hogy mit kell beírni a szolgáltatás elvégzéséhez.

Például:

(1) KESZLETJELENTES
(2) BEVETELEZES
(3) KIADAS
(4) BEFEJEZES

A menüben a befejezés kiválasztására is lehetőséget kell adni. A menü mindaddig a képernyőn marad, amíg a felhasználó nem igényli valamelyik műveletet. A művelet végrehajtása után ismét ki kell írni a menüt, hogy a felhasználó végrehajthassa a következő kívánt tevékenységet. Ezt a vezérlési technikát láthatjuk a 91. ábrán.



91. ábra. A menü vezérlése

Látható, hogy a rendszernek öt fő funkciót kell megvalósítania:

- menükiírás,
- készletjelentés-készítés,
- bevételezés,
- kiadás,
- befejezés.

A feladat egy programmal is megvalósítható, azonban nyilván túl nagy lenne, ezért minden funkciót érdemes külön programozni. A befejezés nem érdemel önálló programot, ezt a feladatot a menüprogram valósítja meg.

Az egész rendszer működésének alapfeltétele a raktári anyagok adatait tartalmazó adatállomány megléte. Az adatállományt a rendszer üzemszerű működtetése előtt valamilyen időpontbeli állapot szerint létre kell hozni. A létrehozást csak egyszer kell végrehajtani.

Ebben a részben az adatállomány létrehozásához szükséges programot készítjük el, a következő részben folytatjuk az elemzést, és az anyagkiadási funkció programját készítjük el.

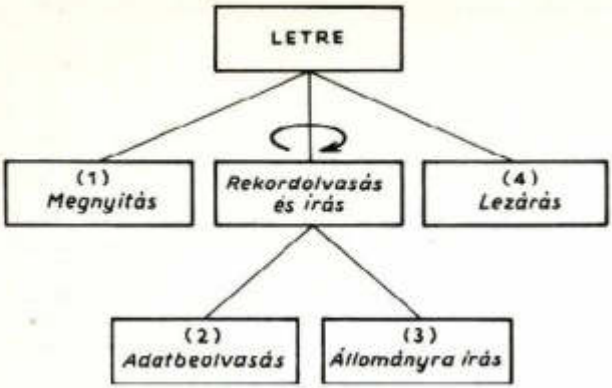
Az **adatállomány-létrehozó** program feladata az, hogy a raktár 500 anyagának adatait (anyagonként 5 adat) a kialakított rendszer szerint egy adatállományba írja be. A legegyszerűbb megoldás, hogy az adatokat a billentyűzeten adjuk be, és a program rekordonként írja be őket az adatállományba. Az adatbeolvasás és az állományra írás ciklust fog alkotni a programon belül. A ciklus befejezésének vezérlésére több lehetőségünk van: előre megkérdezzük a felhasználót, hogy hány rekord adatait kívánja bevinni. Ezután a ciklus végrehajtását számláljuk, és amikor eléri a kívánt számot, a ciklus ismétlése befejeződik.

A másik lehetőség, hogy a program minden rekord bevitele után megkérdezi a felhasználót, folytatja-e még. A kérdésre adott válasz alapján lehet a ciklust folytatni vagy abbahagyni. Válasszuk az utóbbi módszert! Az adható válasz pedig legyen I a folytatásnál, N a befejezésnél. A kivitel előtt az adatállományt meg kell nyitni, a ciklusbefejezés után pedig be kell zárni. A program szerkezetét a 92. ábra mutatja.

A program tervezése

A létrehozó program nyitási és zárási tevékenysége elég egyszerű ahhoz, hogy ne kelljen tovább bontani. A rekordolvasás és -írás — mint nevében is hordozza — két funkciót foglal magában: a ciklikusan végrehajtandó adatbeolvasást és az adatállományra írást. Ezeket célszerű önálló moduloknak tekinteni. Az adatbeolvasási modul beolvassa a rekordok 5 adatát, az állományra író modul pedig az adatokat az ANYAG állományra írja. Ez a modul olvassa be a ciklus vezérlőadatát is. Ezzel a program végleges szerkezete kialakult (93. ábra).

92. ábra. A létrehozó program szerkezete



93. ábra. A program moduljai

A modulok tervezése

(1) *Megnyitás*
A modul **eljárás** típusú, egyetlen művelete az ANYAG állomány megnyitása (94. ábra).

(2) *Adatbeolvasás*
A modul **eljárás** típusú. Bemeneti adatai a felhasználó által begépelt 5 adat, és ezek képezik a kimenetét is. A beolvasást a szokásos módon lehet végrehajtani. Célszerű olyan kiírást készíteni, amellyel a felhasználót segítjük a begépelésben. Hogy a felhasználónak ne kelljen számolgatni az anyagnév 20 karakterét, a beolvasó sor fölé kiírunk egy 20 karakter hosszúságú „mintát". Vagy például a mennyiségi egység beolvasásánál zárójelben kiírjuk, hogy milyen esetek lehetségesek (KG, DB, M). Egy anyag adatainak beolvasását mindig „tiszta" képernyőn kezdjük. Ezért a modul első lépése a képernyőtörlés. Ne felejtsük el, hogy a név (N$) és a mennyiségi egység (M$) szöveges adatok, a többi numerikus (94. ábra).

(3) *Állományra írás*
A modul **eljárás** típusú. Bemenete egy anyag 5 adata, kimenete az öt adatból álló rekord írása az ANYAG állományra.

A modul első művelete az állományra írás. Ezután ki kell írni, hogy a felhasználó folytatja-e. Az N esetében a ciklus befejeződik, ellenkező esetben a (2) modult kell hívni (94. ábra).

(4) *Lezárás*
A modul **eljárás** típusú. Egyetlen művelete az ANYAG állomány lezárzása (94. ábra).

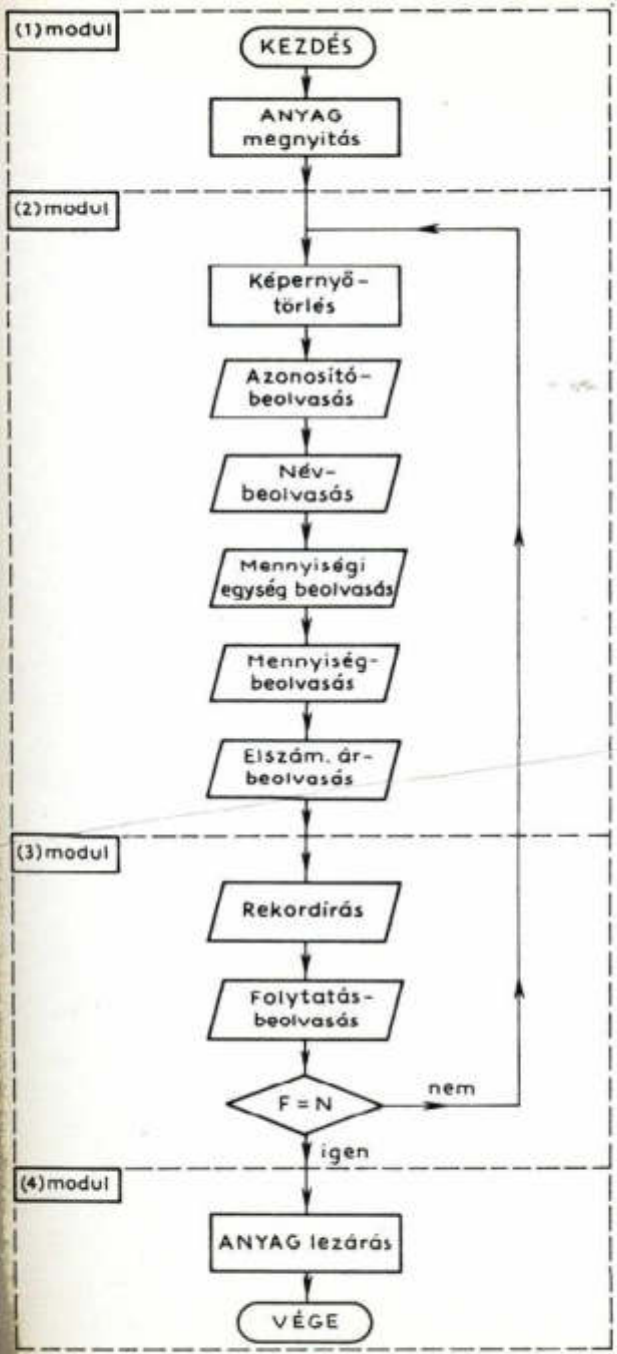
A kódolás

A rendkívül rövid és egyszerű programból néhány újszerű sort bemutatunk. Az állományt „2" logikai hivatkozási és csatornaszámmal nyitjuk meg:

```
70 OPEN 2, 8, 2, "0: ANYAG,S,W"
```

Az adatállományra írásnál ne felejtsük ki a vesszőket:

```
210 PRINT#2, A ; "," ; N$ ; "," ; M$ ; "," ; M ; "," ; AR
```



94. ábra. A program folyamata

Némileg egyszerüsíthető az adatállományra írási utasítás akkor, ha a program elején egy szöveges változónak vesszö értéket adunk, és a mezők közé ezt írjuk be:

```
75 Z$=","
210 PRINT# 2, A ; Z$ ; N$ ; Z$ ; M$ ; Z$ ; M ; Z$ ; AR
```

Adatállományok használatakor több segédfunkció iránt merül fel igény. A program készítője például szeretné tudni, hogy létrejött-e az állomány, és tartalma helyes-e. A létrehozásról legegyszerűbben úgy győződhetünk meg, hogy kiíratjuk a könyvtárunk tartalmát (4. függelék). Az állomány tartalmát pedig csak akkor tudjuk megvizsgálni, ha kiíratjuk. Commodore—64 esetében erre közvetlen lehetőség nincs. Ezért kell írni egy kiíró programot is. Ha CP/M operációs rendszert használunk, akkor a TYPE paranccsal az állomány kilistázható.

14. ADATÁLLOMÁNYOK FELDOLGOZÁSA ÉS MÓDOSÍTÁSA

Az adatállományok feldolgozására szolgáló utasítások.
Az adatállomány módosításának menete.
A 9. feladat két további programjának (MENU és KIVET) elkészítése

Az adatállományokban tárolt adatok **feldolgozása** a létrehozáshoz hasonlóan három lépésben végezhető el. A feldolgozás előtt az adatállományt meg kell nyitni, és egy logikai hivatkozási számot kell neki adni:

x OPEN *i*, 8, *j*, "0 : *név*,S,R"

ahol

i — a logikai hivatkozási szám,
8 (vagy 9) — a lemezegység azonosítója,
j — csatornaszám,
név — az adatállomány neve.

Az S és az R azt jelölik, hogy szekvenciális adatállományt kell olvasásra megnyitni. Felhívjuk a figyelmet, hogy szekvenciális állományt vagy írásra (létrehozásra), vagy olvasásra lehet megnyitni. A többi paraméter értelmezése megegyezik az adatállomány létrehozásánál bemutatottakkal. Az utasítás hatására a gép megkeresi és olvasásra megnyitja a megadott nevű adatállományt. Az adatállomány a megadott logikai hivatkozási számot kapja meg, és a továbbiakban ezzel hivatkozunk az olvasási és lezárási utasításban az adatállományra.

A megnyitott adatállományból az alábbi utasítással lehet egy rekordot beolvasni:

x INPUT# *i*, A, B, . . .

Az utasítás hatására egy rekord beolvasása megy végbe, és a gép fölkészül a következő rekord beolvasására. A beolvasás során a program a rekordban lévő értékeket az utasítás tárgyában felsorolt változóknak adja. A tárolt rekordban lévő mezőhatárokat az adatok közé beírt vesszők jelentik. Szöveges adat olvasásakor szöveges adatnevet kell megadni. Ha kevesebb változónevet (mezőnevet) adunk meg az utasításban, mint amennyi a rekordban van, akkor a gép a maradék részt nem olvassa be. Ellenkező esetben viszont a következő rekord mezőiben talált értéket adja az utasításban szereplő változóknak.

Az állomány feldolgozása is ciklikusan megy végbe. Ez azt jelenti, hogy az adatállomány elejétől a végéig minden rekordot be kell olvasni.

Ha az utolsó rekord (állományvég) után még egyet akarunk olvasni, akkor hiba lép fel. Ezt nyilván el akarjuk kerülni.

A Commodore–64 gépben van egy ST nevű állapotbájt, amelynek tartalma mindaddig nulla, amíg az utolsó rekordot be nem olvastuk. Az utolsó rekord beolvasása után az állomány elérésekor nullától eltérő értéket kap. A feldolgozást az ST értékétől kell függővé tenni. Amíg

$$ST = 0 \ ,$$

addig újabb olvasás végrehajtható. Ha a feltétel nem teljesül, akkor több olvasási műveletet nem szabad végrehajtani, mert elértük az állomány végét. Az olvasási ciklus általában hátul tesztelő.

A rekord feldolgozása során az ST bájt értéke megváltozhat. Ezért tanácsos az ST bájt tartalmát rögtön az olvasás után átmásolni egy végjelző változóba:

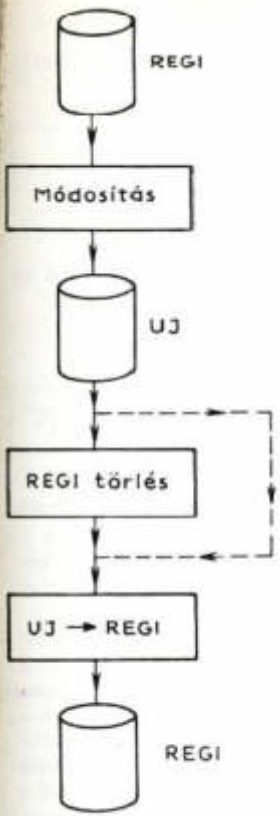
$$VJ = ST \ .$$

A rekord feldolgozása után a VJ tartalmát kell vizsgálni, és ez alapján eldönteni, hogy kell-e folytatni a feldolgozást. Az olvasás végén az állományt a már ismert utasítással kell lezárni:

$$x \ \text{CLOSE} \ i \ .$$

Az **adatállományok tartalmát** a valóságban végbemenő változásoknak megfelelően **módosítani** kell. Hogyan módosíthatunk szekvenciális állományokat? A módosítás során azt akarjuk elérni, hogy egy vagy több rekord mezőinek tartalmát megváltoztassuk, vagy új rekordokat adjunk hozzá az állományhoz, illetve meglévőket töröljünk. Tudjuk viszont, hogy szekvenciális adatállományokat csak olvasásra tudunk megnyitni. Írásra meglévő állományt nem is lehet megnyitni, mert az írásra való megnyitás új állományt hoz létre. Ebből látszik, hogy közvetlenül nem tudjuk a módosítást végrehajtani. A módosítás úgy végezhető el, hogy a módosítandó (REGI) adatállományt olvasásra megnyitjuk, és emellett megnyitunk egy új (UJ) adatállományt is írásra. Ezek után a REGI állomány valamennyi rekordját beolvassuk, amelyeket módosítani szükséges, azt módosítjuk, és valamennyi módosított vagy nem módosított rekordot az UJ állományra írjuk fel.

A művelet végén az új állomány tartalmazza a REGI állomány módosított változatát. Tehát célunkat részben elértük. Az egyedüli kellemetlenség, hogy a művelet végén az állomány más néven szerepel. Ez azt eredményezné, hogy az adatállományt használó programokban az állomány nevét módosítani kellene. De szerencsére ezen is tudunk segíteni. Az UJ állomány nevét REGI-re kell változtatni. Előbb azonban a már feleslegessé vált REGI állományt törölni kell. Ezután a névváltoztatás már végrehajtható, és ezzel a módosítás folyamata befejeződött (95. ábra).



95. ábra. A módosítás folyamata

A létrehozás és feldolgozás műveletein kívül az adatállományokkal egyéb műveleteket is végre kell hajtani: másolás, névváltoztatás, törlés, több állomány összekapcsolása stb. Ezeket a műveleteket az ún. parancscsatornára kiadott utasításokkal lehet végrehajtani. Az állományokkal kapcsolatos műveleteket 3 lépésben kell végrehajtani:

1. A parancscsatorna megnyitása
2. A műveletre vonatkozó utasítás kiadása
3. A parancscsatorna lezárása

A parancscsatornát az

OPEN 15, 8, 15

utasítással lehet megnyitni. A 15 a parancscsatorna száma. Az egyszerűség kedvéért a logikai hivatkozási számot is 15-nek vesszük.

Az adatállományokkal kapcsolatos utasítások a parancscsatornára írt utasítások. A törlés utasítása:

x PRINT# 15, "SCRATCH0: *név*"

ahol

SCRATCH0 — a törlési művelet kulcsszava (S0-val rövidíthető),
név — a törölni kívánt állomány neve.

Állománynevet a következő utasítással lehet megváltoztatni:

x PRINT# 15, "RENAME0: *új név* = *régi név*"

ahol

RENAME0 — a névváltoztatási művelet kulcsszava (R0-val rövidíthető),
új név — az állomány új neve a művelet után,
régi név — az állomány jelenlegi neve (a művelet előtt).

A kívánt művelet elvégzése után a parancscsatornát le kell zárni:

x CLOSE 15

Az állományokkal kapcsolatos további műveletek a 4. függelékben találhatók.

9. feladat (folytatás)

Ebben a részben a MENU programot és az anyagkivételezési funkciót ellátó programot készítjük el.

A feladatok elemzése

A MENU program két feladatot lát el. Mivel ez a programrendszer kezdőprogramja, itt kell a menüt kiírni, majd be kell hívni a kiválasztott funkciót végrehajtó programot (96. ábra).

Az **anyagkivételezést** megvalósító programnak az a feladata, hogy a raktárból kivett anyag mennyiségével csökkentse az illető anyag készletét. Ehhez a programnak be kell olvasnia, hogy melyik anyagból mennyit vettek ki. Ezután az illető anyag készletét módosítani kell. Végül ki kell írni a kivét bizonylatot, és be kell hívni a MENU programot (97. ábra).

A kivét bizonylatot sornyomtatón a következő formában kell elkészíteni:

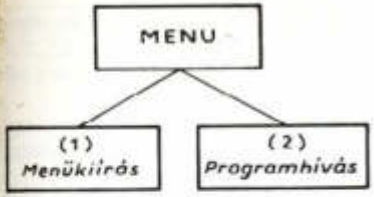
ANYAGKIVETELI JEGY

Kelt: (EV, HO, NA)

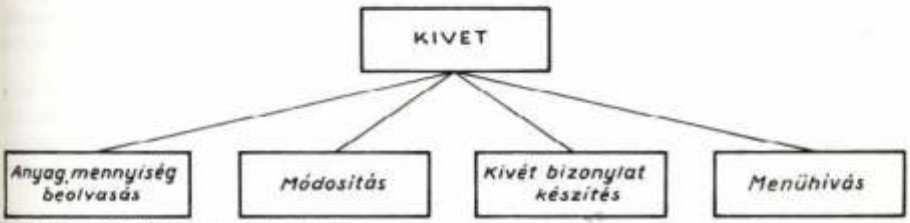
AZ ANYAG AZONOSITOJA: (RA)
AZ ANYAG MEGNEVEZESE: (NS)
MENNYISEGI EGYSEG: (MS)
KIVETT MENNYISEG: (KM)

.....

ALAIRAS



96. ábra. A MENU program szerkezete



97. ábra. A KIVET program szerkezete

A program kezdetén ki kell írni a képernyőre, hogy a kivételezési funkció megy végbe (ANYAGKIVETELEZES). Ezzel tudja ellenőrizni a felhasználó, hogy jó gombot nyomott-e meg.

Egy műveletet külön meg kell vizsgálnunk.

Az **anyagazonosító** beolvasása után ellenőrizni kell, hogy van-e ilyen azonosítójú anyag az ANYAG állományban. Ha nincs, akkor a felhasználó vagy elgépelte az azonosítót, vagy rosszul tudja a cikkszámot. Az első esetben a hiba javítható, a másodikban nem rögtön. Ha azt az eljárást választanánk, hogy hibás azonosító beírása esetén az azonosító beolvasására mennénk vissza, akkor nem javítható hibánál lehet, hogy sohasem tudnánk befejezni a programot. Ez nyilván előnytelen, ezért előbb meg kell kérdezni a felhasználót, hogy ki tudja-e javítani a hibát. Ha igen, akkor visszamegyünk az azonosító beolvasására. Ellenkező esetben a program befejeződik.

A programok tervezése

A MENU program moduljait nem célszerű tovább bontani. Azt jegyezzük meg, hogy a (3) modulban lévő programkiválasztás CASE szerkezetet alkot, mint a 6. részben bemutatott KESZL program menümodulja.

A KIVET program moduljait érdemes szemügyre venni. Az első modulban kap helyet a művelet kiírása, majd az azonosító beolvasása következik. Rögtön ezután – még a mennyiség beolvasása előtt – ellenőrizni kell, hogy a begépelt azonosítójú rekord szerepel-e az ANYAG állományban. Ezt úgy lehet végrehajtani, hogy az adatállomány valamennyi rekordjából beolvassuk az azonosítót, és összehasonlítjuk a begépelttel. Ha a kettő egyezik, akkor a további olvasásra nincs szükség, az állomány lezárható, és a program folytatható. Ha a program az állomány végéig nem talál a begépelttel egyező azonosítót, akkor az állományt le kell zárni, és a hibát ki kell írni. Majd meg kell kérdezni, hogy javítható-e a hiba. Ha igen akkor újabb olvasás következik, ha nem, akkor a vezérlés a MENU hívásra kerül, a program befejeződik. Vegyük észre, hogy a program akár talált, akár nem talált rekordot, a további műveletek előtt az állományt le kell zárni. Hogy tudjuk, melyik tevékenységcsoportot kell végrehajtani a lezárás után, fel kell venni egy T találatjelzőt, amelynek

értéke találat esetén 1, egyébként 0. Ennek értéke dönti el, hogy milyen lépés következik a zárás után. A T-t minden keresés előtt nullázni kell, hogy az előző értéke ne zavarja a következő folyamatot. Ebből látható, hogy az első modult célszerű további modulokra bontani (98. ábra).

A következő művelet a kivett mennyiség beolvasása. Itt azt kell ellenőrizni, hogy van-e a kívánt mennyiség az anyagból. Ha nincs, akkor csak a meglevő mennyiséget lehet kiadni (a készlet nulla lesz), egyébként az igényeltet. Az első esetben egy üzenetet is ki kell írni, hogy a raktáros is tudjon a változásról.

Hogy ne kelljen ismét végigolvasni az ANYAG állományt a mennyiség ellenőrzésére, az azonosító ellenőrzésekor a mennyiséget is beolvassuk. Találat esetén az utolsónak beolvasott rekordban levő mennyiségi adat megmarad, mivel utána nem hajtunk végre újabb olvasást, ami a mennyiséget megváltoztatná.

A következő lépés az ANYAG állomány módosítása. Ezt a módosításnál leírt lépésekben kell elvégezni. Az új állomány neve legyen ATM (átmeneti név). Módosításkor a módosított rekord előtti és utáni rekordokat változatlanul kell átmásolni. A kivett anyag rekordjában a mennyiség értékét a kivétnek megfelelően módosítva kell átírni.

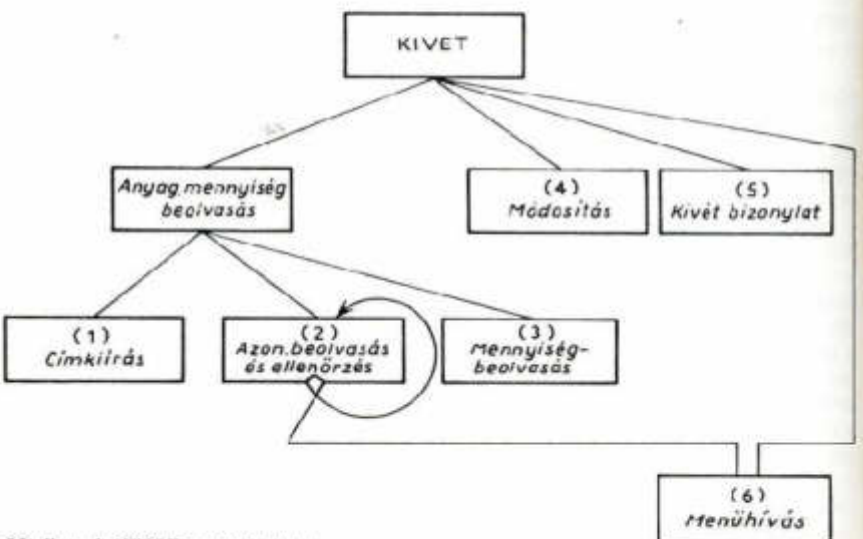
A kivét bizonylat készítése előtt meg kell nyitni a nyomtatót mint állományt, és a kiírást ezen kell elvégezni. A bizonylat kiírásához a talált rekord adatait kell felhasználni, majd ezután ezt az állományt le kell zárni.

A program utolsó modulja a MENU hívás (98. ábra).

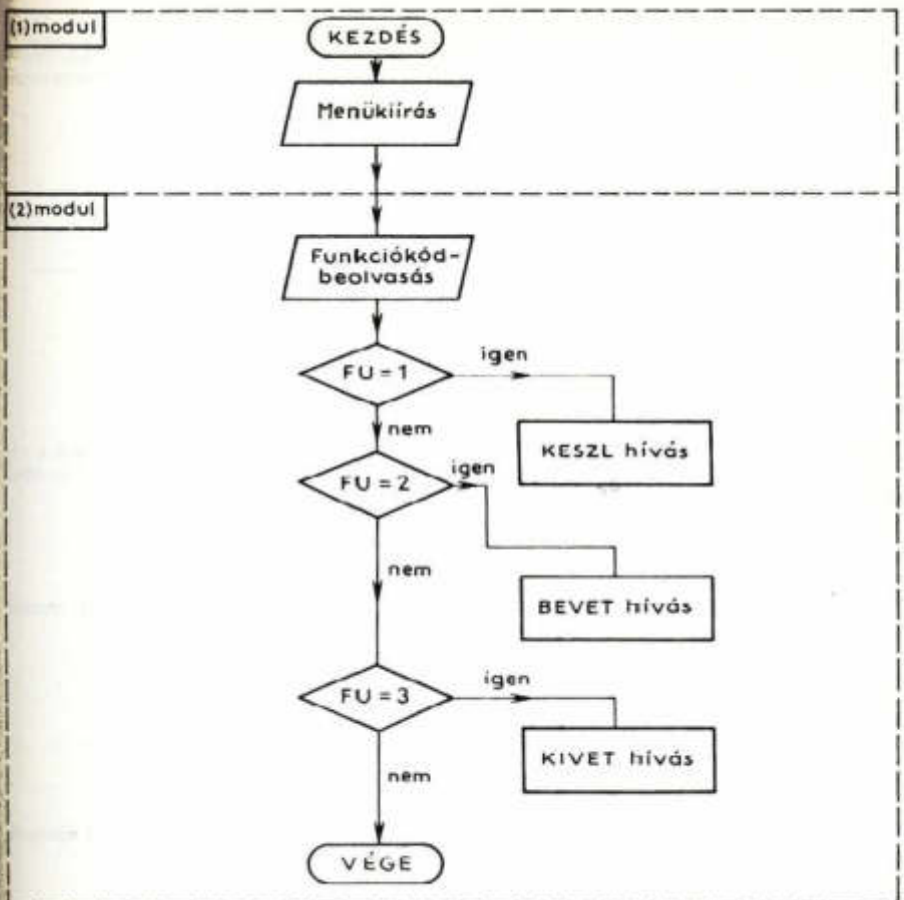
A modulok tervezése

A MENU program moduljai

(1) *MENU kiírás*
Eljárás modul. Kimenete a 13. részben bemutatott formájú menü kiírása (99. ábra).



98. ábra. A KIVET program terve



99. ábra. A MENU program folyamata

(3) *Programhívás*
A modul **eljárás** típusú. Bemeneti adata a kiválasztott funkcióhoz tartozó szám beolvasása. Kimenete a kiválasztott feladatot megvalósító program hívása. A programok hívása előtt a változók tartalmát CLR utasítással nullázni kell (99. ábra).

A KIVET program moduljai

(1) *Címkiírás*
A modul **eljárás** típusú. Kimenete az "ANYAGKIVETELEZES" cím kiírása (100. ábra).

(2) *Azonosító beolvasása és ellenőrzése*
A modul **eljárást** valósít meg. Bemenete a felhasználó által begépelt azonosító és az ANYAG állomány azonosítói. Kimenete egy vezérlés vagy a modul elejére, vagy a (6) modul hívása.

A modul kezdetén be kell olvasni annak az anyagnak az azonosítóját (AA), amelyből kivesznek. Ezután a találatjelzőt nullázni kell, majd meg kell nyitni az ANYAG állományt, és minden rekordból az első négy adatot be kell olvasni:

- azonosító (RA),
- név (N$),
- mennyiségi egység (M$),
- mennyiség (M1).

Minden beolvasás után át kell másolni az ST bájt értékét:

$$VJ = ST .$$

és ellenőrizni kell, hogy elértük-e a keresett rekordot:

$$AA = RA .$$

Ha a feltétel teljesül, akkor előbb a T = 1 értékadást kell végrehajtani, majd a zárásra kell ugrani (a további olvasástól eltekintünk). Ha a feltétel nem teljesül az állomány végéig (VJ ≠ 0), akkor a keresés véget ér, és az állományt le kell zárni.

A további műveleteket a T határozza meg. Ha

$$T = 1 ,$$

akkor a modul befejeződik (megtaláltuk a keresett rekordot), és a (3) modul hajtódik végre.

Ha

$$T = 0 ,$$

akkor a javíthatóságot kell megkérdezni (JA$).

Ha

$$JAS = I ,$$

akkor a modul kezdetére kell visszatérni az újabb beolvasásra. Ellenkező esetben a (6) modul (befejezés) következik (100. ábra).

(3) *Mennyiségbeolvasás*

A modul **eljárás** típusú. Bemeneti adata a kivenni kívánt mennyiség. Kimenete pedig a kiadható mennyiség.

A modulban előbb a kivenni kívánt mennyiséget (KM) kell beolvasni, majd meg kell vizsgálni, hogy a mennyiség kiadható-e:

$$M1 > KM .$$

Ha a feltétel teljesül, akkor az "A MENNYISEG KIADHATO" üzenet kiírása után a maradék mennyiség kiszámítása következik a közös ágban:

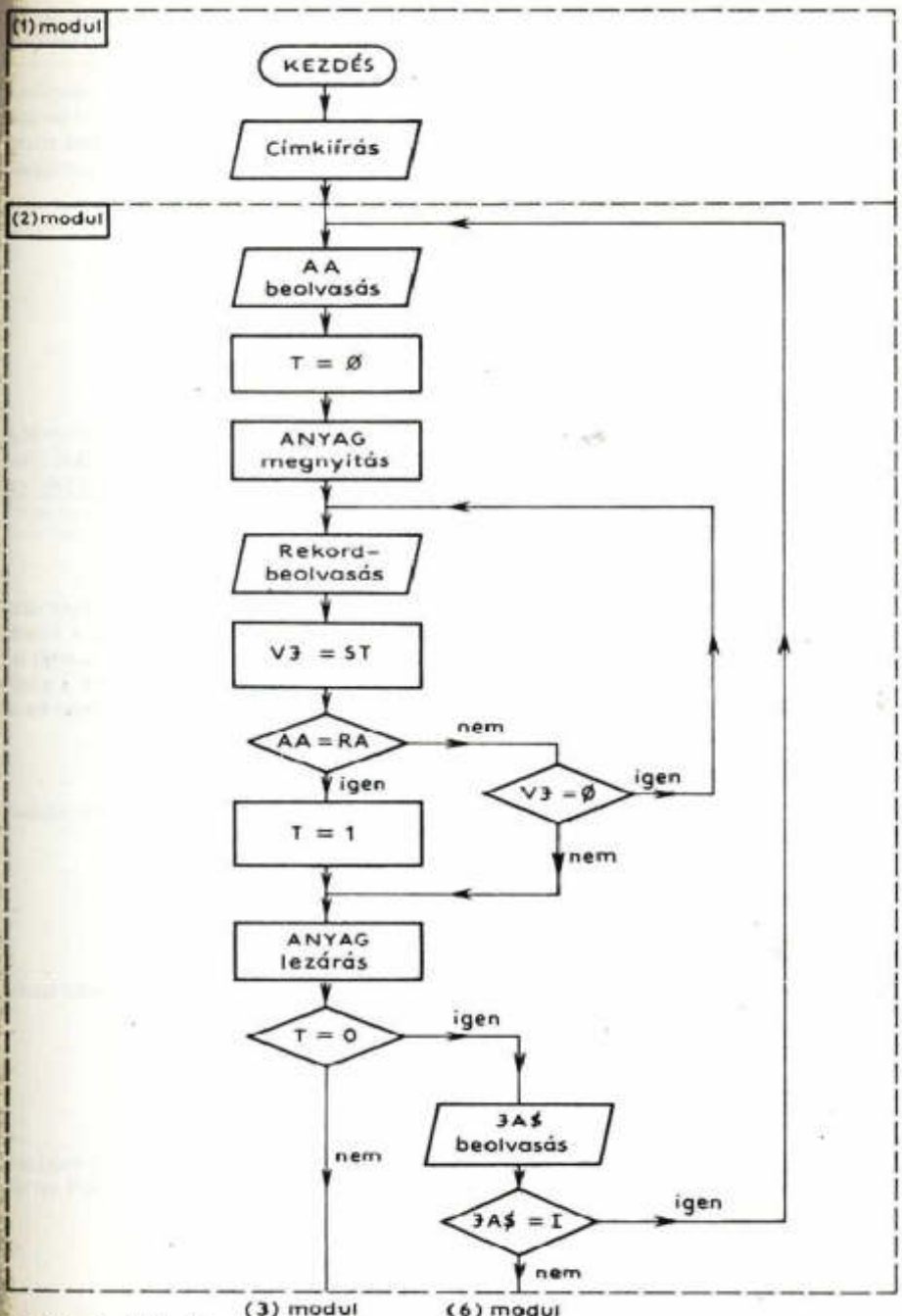
$$MM = M1 - KM .$$

Ellenkező esetben csak M1 mennyiség adható ki, és azt a "CSAK M1 ADHATO KI!" üzenettel tudatni kell a felhasználóval. Ezután a kiadott mennyiséget módosítjuk:

$$KM = M1 .$$

Ezután a közös ágban a maradékot számítjuk ki (101. ábra):

$$MM = M1 - KM .$$



100. ábra. Az (1) és (2) modul folyamata

(4) *Módosítás*
A modul **eljárás** típusú. Bemenete a csökkentett mennyiségű anyag azonosítója, a maradék mennyiség és az ANYAG állomány. Kimenete a módosított ANYAG állomány.

A modulban először meg kell nyitni a módosításhoz szükséges két állományt (ANYAG, ATM), majd elkezdődik a másolási ciklus. Minden beolvasással be kell olvasni a teljes rekordot, de a kereséstől eltérő névvel, hogy a kivételezett anyag adatait ne veszítsük el. Itt ezért az A2, N2$, M2$, M2, AR mezőneveket használjuk. Meg kell vizsgálni, hogy megtaláltuk-e a keresett rekordot. Ha nem, vagyis

$$A2 \neq AA,$$

akkor a rekordot változatlanul az ATM-re írjuk. Ha megtaláltuk, tehát

$$A2 = AA,$$

akkor az M2 értékét MM-re (maradék) módosítjuk:

$$M2 = MM,$$

és ezután hajtjuk végre a kiírást. A művelet ciklusába az ANYAG nevű állomány végének figyelését is be kell építeni. Ha az átmásolás befejeződik, mindkét állományt lezárjuk. Előbb az ANYAG-ot töröljük, majd az ATM nevét ANYAG-ra változtatjuk. Ezzel a módosítás befejeződik (101. ábra).

(5) *Kivét bizonylat*
A modul **eljárás** típusú. Bemeneti adatai a bizonylat elkészítéséhez szükséges adatok: az itt begépelt dátum, valamint az anyagazonosító, az anyagnév és a kivett mennyiség a (2) és a (3) modulokból. Ezek az adatok képezik a modul kimenetét is.

A modulban először meg kell nyitni a nyomtatót, és ezután ki kell írni a kivét bizonylatot. A bizonylat kiírása után a nyomtatót le kell zárni. Ezzel a modul be is fejeződik (102. ábra).

(6) *MENU hívás*
A modul **eljárás** típusú. Egyetlen művelete a MENU program hívása. Nem szabad elfelejteni, hogy a hívás előtt a változók tartalmát nullázni kell (102. ábra).

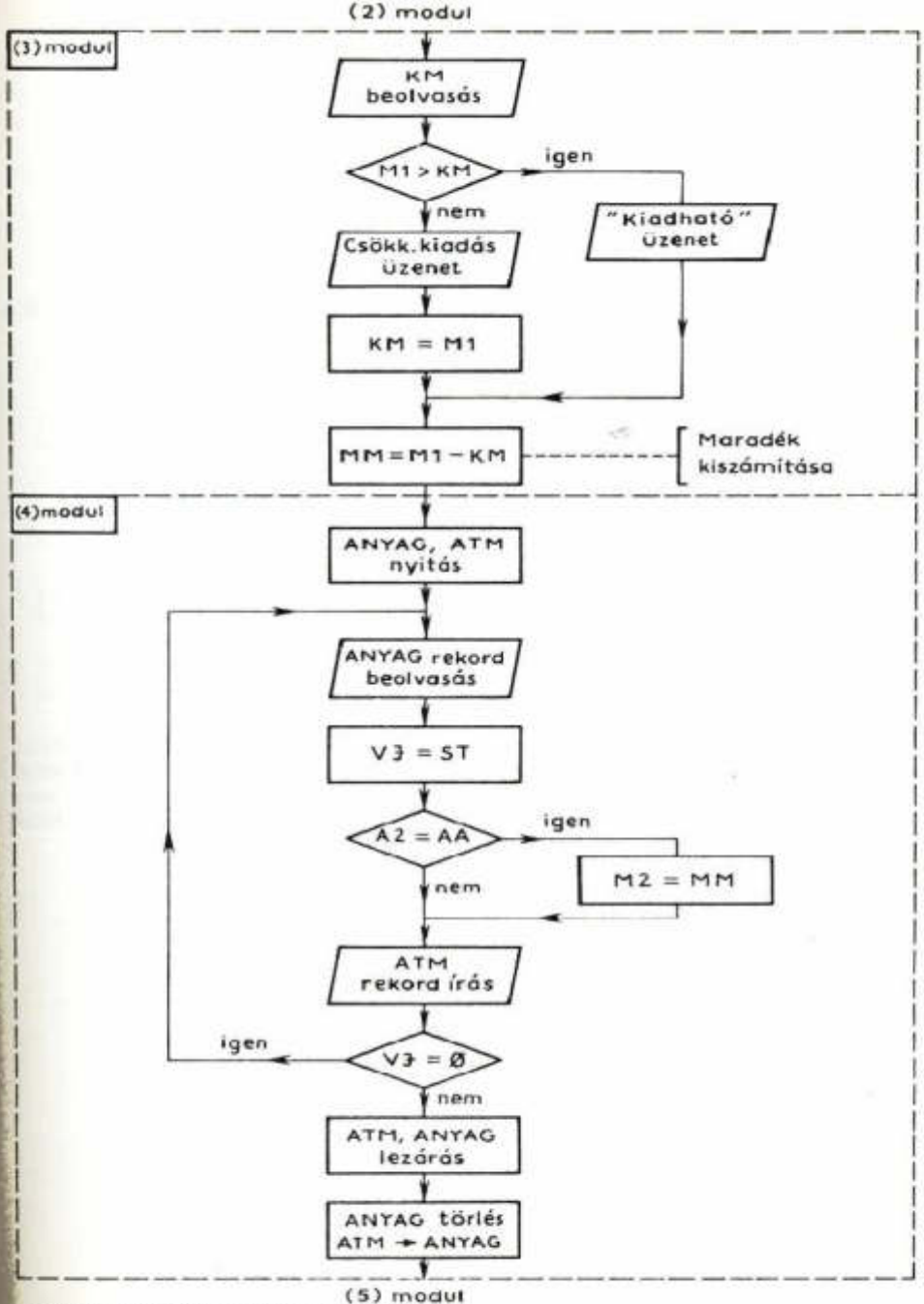
A programok kódolása

A KIVET program kódjából bemutatunk néhány szakaszt. Ha az azonosító hibás volt, akkor ezt ki kell írni, utána kell a javíthatósági kérdést feltenni.

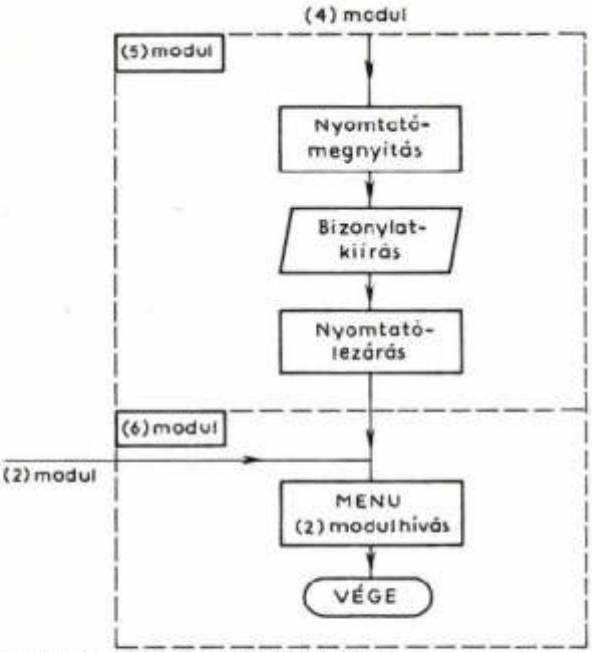
```
250 PRINT"HIBAS AZ AZONOSITO!"
260 PRINT
270 INPUT"JAVITHATO ? (I/N) ";JA$
```

A (4) modul végén a „régi" ANYAG állomány törlése és az ATM nevének módosítása előtt a parancscsatornát meg kell nyitni, a műveletek után pedig le kell zárni.

```
550 OPEN15,8,15
560 PRINT#15,"S0:ANYAG"
570 PRINT#15,"R0:ANYAG=ATM"
580 CLOSE15
```



101. ábra. A (3) és (4) modulok műveletei



102. ábra. Az (5) és (6) modulok műveletei

Értékelés

A bemutatott eljárás pontosan végrehajtja a feladatot. A rendkívül gyakori adatállomány-másolás és -törlés a működést lassúvá teszi, és a lemez helytelen működését idézheti elő. Ezért időszakonként (naponta vagy másnaponként) „rendet" kell tenni a lemezen, az összes hibás állományt ki kell törölni. Ezt a VALIDATE paranccsal végezhetjük el.

A VALIDATE parancsot is a parancscsatornára kell kiadni.

```
OPEN 15, 8, 15 : PRINT# 15, "VALIDATE"
```

Ellenőrző kérdések és feladatok

1. Mi az adatállomány előnye a tömbbel szemben?

2. Ha a 9. feladat 500-féle anyagából valamelyikről meg kell mondani, hogy van-e belőle, akkor hogyan kellene rendszerezni az anyagok rekordjait, hogy gyorsan megtaláljuk a keresett rekordot?

3. Alakítsuk át a 9. feladat programjait úgy, hogy egy vezérlőmodul irányítsa a szubrutinokként kódolt modulokat!

4. Készítsünk programot, amely az anyagok tárolt adatait egy-egy sorba táblázatos formában kiírja!

5. Készítsük el az anyagbevételezést végrehajtó programot, és illesszük be a meglévő rendszerbe!

6. Készítsük el a készletjelentést kiíró programot, és ezt is illesszük be a meglévő rendszerbe!

7. Miért nem lehet adatállományt LIST parancssal kiíratni?

15. GRAFIKAI LEHETŐSÉGEK

A foltábrázolási lehetőség alkalmazása a Commodore–64-en.
A 10. feladat megoldása

Minden számítógéppel lehet ábrákat rajzoltatni akár a terminál képernyőjére, akár a nyomtató felhasználásával. A legegyszerűbb és „legrégibb" módszer az írógépszerű rajzkészítés. A rajzolandó ábrát sorokra bontjuk, és a sorokban olyan karaktereket jelzünk vagy nyomtatunk ki, amit az ábra megkíván. A sötétebb pontokra például M betűt, a világosabbakra pontot (vagy semmit) írunk. Ezekből a karakterekből végül is jól-rosszul kialakul a rajzolni kívánt ábra. Persze elég nagy méretre fel kell nagyítani készítéskor, hogy felismerhető legyen. Mindez elég sok apólékos munkával jár, hiszen pontonként kell az ábrát elkészíteni, és ez alapján kell a programot is megírni. Egyszerű vonalas ábrák viszonylag egyszerűen elkészíthetők, de kontrasztos rajzok csak nehéz munkával.

A legtöbb személyi számítógépnek van valamilyen önálló rajzolási lehetősége, ami a fenti módszernél valamivel egyszerűbb és rajzszerűbb eredményt szolgáltat. Sajnos ezek gépenként különböző alapelvre épülnek, ezért nem tárgyalhatók egységesen, mint a BASIC nyelv. Itt a Commodore—64 folt-(sprite-)készítés technikáját használjuk fel az egyszerű statisztikai diagram megjelenítéséhez.

A Commodore-on van más rajzolási lehetőség is. Ez az ún. **grafikus jelekkel** való rajzolás. A grafikus jelek egy karakternyi területen vízszintes és függőleges vonalakból vagy fehér és fekete foltokból, vagy egyszerű figurákból (pl. szív) álló rajzos „építőkövek", amelyekből táblázatokat, egyszerű figurákat vagy színes területeket lehet összerakni.

Más gépeken más jellegű rajzolási lehetőségek vannak. Elsősorban a vonalas ábrák rajzolása terjedt el. A rajzolás általában miden gép BASIC-jében másképpen működik. Ennek az az oka, hogy a BASIC oktatási célra tudományos számítások és egyszerű adatfeldolgozási feladatok megoldására készült, a grafikus lehetőségek később alakultak ki, s ezt a részt minden gyártó saját maga dolgozta ki, ezért a grafikai részek jelentősen eltérnek egymástól.

A foltrajzolás azért érdekes, mivel ez nemcsak statikus ábrák, hanem mozgó rajzok készítésére is alkalmas. Ebben a részben a foltrajzolásnak csak az alapjait rakjuk le. Az itt elsajátított ismeretek alapján mozgó ábrákat már nem nehéz készíteni.

A foltrajzolás technikájának elsajátításához mélyebben „bele kell bújni" a számítógépbe, mivel ehhez nincsenek külön BASIC utasítások, hanem a meglévőket használjuk fel, amelyek nyilván nem rajzolásra orientáltak, ezért a gépi tulajdonságokat közvetlenebbül kell alkalmazni.

Előbb azonban ismerjünk meg egy új utasítást és egy új függvényt! A számítógép tárjában lévő adatok bájtonként címezhetők, vagyis egy bájt tartalmát egy művelettel lehet kiolvasni vagy új értéket beleírni. 1 bájt 8 bitből áll, egy bit pedig egy kétállapotú áramköri elem. A két állapotot 0-val és 1-gyel jelöljük. Egy bájt tartalmát az határozza meg, hogy az őt alkotó bitek milyen állapotban vannak. Egy bájtnak 256-féle értéke lehet a bitek helyzetétől függően. A tár egy bájtjába íráskor a gép a bájt bitjeit állítja be az adatnak megfelelően. A bájt kiolvasásakor pedig azt nézi meg, hogy milyen állapotban vannak a bitek.

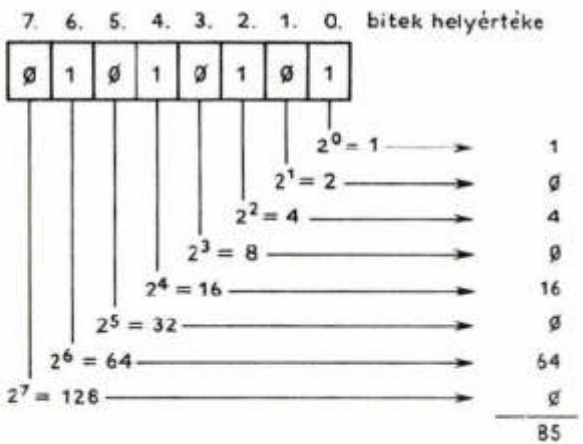
A foltrajzoláshoz nemcsak a bájtok tartalmát kell vizsgálni vagy módosítani, hanem a bitekét is. A bitekhez azonban nem tudunk hozzáférni, csak a bájtokhoz. Amikor tehát biteket kell állítanunk, akkor mindig 8 bitet, azaz 1 bájtot kell állítani, mivel csak ilyen utasításunk van.

Egy bájtba a POKE utasítással lehet valamilyen adatot beírni:

x POKE *cím, tartalom*

A *cím* a tárban címezhető valamelyik bájt címe (0–65535), a *tartalom* pedig egy 0–255 közötti decimális szám. Tegyük fel, hogy a bájt jobb oldali szélső (legalacsonyabb helyértékű) bitjébe 1-et és innen kezdve minden második bitbe ugyancsak 1-et kell beírni (103. ábra).

Ezt így, ilyen formában nem tudjuk a POKE utasításban ábrázolni, mivel ott csak decimális szám adható meg. Ezért a bájtban lévő biteket helyértékes bináris számnak tekintjük, és ennek megfelelően átkódoljuk decimális számmá a 103. ábrán látható rendszer szerint. Ha tehát ezt a bitkombinációt akarjuk beírni, akkor a tartalomhoz 85-öt kell



103. ábra. A bitek jelentése

Oszlopszám	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
bitérték	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1
0. sor																								
1. sor																								
2. sor																								
3. sor																								
4. sor																								
5. sor																								
6. sor																								
7. sor																								
8. sor																								
9. sor																								
10. sor																								
11. sor																								
12. sor																								
13. sor																								
14. sor																								
15. sor																								
16. sor																								
17. sor																								
18. sor																								
19. sor																								
20. sor																								

104. ábra. Feltöltődefiniáló táblázat

beírni. Ha csupa nullát akarunk, akkor 0-t, ha csupa 1-et akarunk, akkor 255-öt kell megadni.

A tár valamelyik bájtjának tartalmát egy függvénnyel tudjuk kiolvasni:

$$x\ A = PEEK\ (cím)$$

A kiolvasott érték mindig egy 0–255 közötti decimális szám.

Ezek után térjünk át a foltrajzolás technikájára! Először is azt kell tudnunk, hogy a Commodore–64 a képernyőt vízszintes irányban 320, függőleges irányban pedig 200 pontra bontja fel. Rajzoláshoz ezt a lehetőséget használjuk fel. A **folt** ezeknek a pontoknak egy együttese. Ezek a pontok, illetve a folt maga lehet egy ábra vagy egy ábrarészlet. A folthoz egy bitekből álló mátrix tartozik a tárban. A folt minden pontjához a mátrix egy bitje tartozik. Hogy ez pontosan mit jelent, a későbbiekben fogjuk látni.

Egy folt a képernyőn vízszintes irányban 24, függőleges irányban 21 pontot foglal magában (összesen 504 pont). A folt a képernyő bármelyik részére elhelyezhető. Egy folt pontjai azonos vagy legfeljebb 3 eltérő színben jeleníthetők meg. A képernyőn egyszerre 8 folt jeleníthető meg 16 különböző színben. Ezek ki- és bekapcsolhatók, azaz megjeleníthetők és eltüntethetők. A foltokat mind vízszintes, mind függőleges irányban kétszeresre lehet nagyítani és vissza is lehet állítani. Továbbá a foltokat mozgatni, ütköztetni stb. lehet, de ezekkel a lehetőségekkel itt most nem foglalkozunk.

A folt alakját a szerint az elv szerint lehet elkészíteni, hogy a megjelenítendő ponthoz tartozó bitnek 1 értéket adunk. A háttér színét felvevő pontok (nem látható pontok) bitjeinek értéke 0 lesz. A foltban lévő ábrát a 104. ábrán látható segédeszközzel készíthetjük el. Az ábra azt is mutatja, hogy 1 sor 24 pontját 3 bájt tartalmazza. Az 504 pontot összesen tehát 63 bájtba lehet elhelyezni. A kezelés megkönnyítésére 64 bájtos egységeket használunk fel a foltok definiálására. A bájtok sorszámozása:

1. sor	0. bájt	1. bájt	2. bájt
2. sor	3. bájt	4. bájt	5. bájt
.	.	.	.
.	.	.	.
.	.	.	.
20. sor	60. bájt	61. bájt	62. bájt

Egy folt tehát 64 bájtot foglal le a gép tárjában. A **foltmutató** bájtok tartalmazzák, hogy ez a 64 bájtos egység hol található a tárban. Ezekből összesen 8 van a 8 megjeleníthető folthoz, a következő rendszerbe foglalva:

A folt sorszáma	A foltmutató bájt címe
0.	2040
1.	2041
2.	2042
3.	2043
4.	2044
5.	2045
6.	2046
7.	2047

A foltmutatóban lévő szám mutatja meg, hogy a tár hányadik 64 bájt hosszúságú egységében található a foltdefiníció. Természetesen nem lehet akárhová elhelyezni egy foltot, mivel a tárban lévő bájtok egy részét a gép saját maga használja. Ezt táblázat alapján ellenőrizhetjük. Jól használható például a BASIC programoknak fenntartott tárterület „hátsó" része, ami még közepes hosszúságú programok esetében is szabad marad. A programoknak fenntartott terület a 2048-as és 40959-es címek között helyezkedik el. Ha 10000 bájt helyet hagyunk a programnak, akkor a 192x64=12288-as címen kezdhetjük a foltok tárolását. Ha például a 0. folt definícióját ettől a címtől kezdve kívánjuk elhelyezni, a 2040-es bájtba 192-t kell beírni:

```
10 POKE 2040, 192
```

Ha az 1. folt definícióját a következő 64 bájtra akarjuk elhelyezni, akkor a

```
20 POKE 2041, 193
```

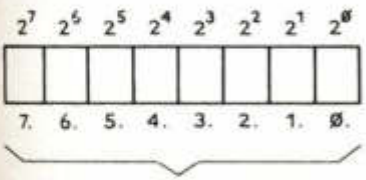
utasítást használjuk. Az 1. folt definíciója tehát a 12352-es (193*64) címen fog kezdődni.

Ahhoz, hogy a folt megjelenjen a képernyőn, **be kell kapcsolni,** illetve ha nem akarjuk mutatni, akkor **ki kell kapcsolni.**

A foltokat az 53269 című bájt megfelelő bitjeinek 1-be állításával lehet megjeleníteni (bekapcsolni). A foltot úgy tudjuk eltüntetni a képernyőről, hogy a hozzá tartozó bitet 0-ra állítjuk (105. ábra).

Ha például csak a 4. foltot akarjuk bekapcsolni, akkor 16-ot kell beírni a bekapcsoló bájtba:

```
100 POKE 53269, 16
```



105. ábra. Foltok be- és kikapcsolása

A foltokat 16 különböző **színben** lehet megjeleníteni. Minden folthoz tartozik egy színbájt, amelynek tartalma határozza meg a folt színét:

A folt sorszáma	A színbájt címe
0.	53287
1.	53288
2.	53289
3.	53290
4.	53291
5.	53292
6.	53293
7.	53294

A színeket a következő értékek beírásával állíthatjuk be:

Szín	Színbájt tartalma
fekete	0
fehér	1
piros	2
ciánkék	3
bíbor	4
zöld	5
kék	6
sárga	7
narancs	8
barna	9
rózsaszín	10
sötétszürke	11
középszürke	12
világoszöld	13
világoskék	14
világosszürke	15

Egy foltot nemcsak egy színben lehet megjeleníteni, hanem több színben is. Egy folton belül összesen három színt lehet használni színezésre. A foltok **mérete** mind függőleges, mind vízszintes és mindkét irányban is megkétszerezhető. A vízszintes irányú kétszerezést az 53277 című bájt megfelelő bitjeinek 1-re állításával érhetjük el. A bitek és a foltok párosítása ugyanúgy történik, mint a bekapcsoló bájtnál. A függőleges irányú kétszerezés pedig az 53271 című bájt megfelelő bitjeinek beállításával érhető el.

Foltot a képernyőn tetszés szerinti helyre állíthatunk. A **foltbeállítás** a legfelső sor és a bal szélső oszlop helyzetének megadásával végezhető el.

Függőleges irányban 250 helyzetbe állítható egy folt (106. ábra). Ebből 200 pozíció (lásd képernyőfelbontás) a képernyőre esik, 50 pedig

fölé. Ebből következik, hogy egy folt akkor jelenik meg teljes egészében a képernyőn, ha az 50. sorra helyezzük, és addig látható mind a 21 sora teljesen, ha a 229. sorra (függőlegesen kétszerezett folt esetében a 208. sorra) helyeztük el a képernyő aljára.

A vízszintes irányú elhelyezés elve ugyanez. A képernyő vízszintes irányú felbontása 320 pozíció, de a folt 343 pontra helyezhető el. A folt akkor látható teljesen a képernyő bal szélén, ha a 24. oszlopra helyezzük, és addig látható teljesen a képernyő jobb szélén, ha a 320. oszlopra (vízszintes irányban megnagyított folt esetén a 296. oszlopra) állítjuk be (106. ábra).

A foltok függőleges irányú pozicionálását egy számértékkel végezhetjük el, amely 0 és 250 közötti érték lehet. Egy bájt 256 különböző értéket képes tárolni, ezért a pozicionáláshoz 1 bájt elegendő.

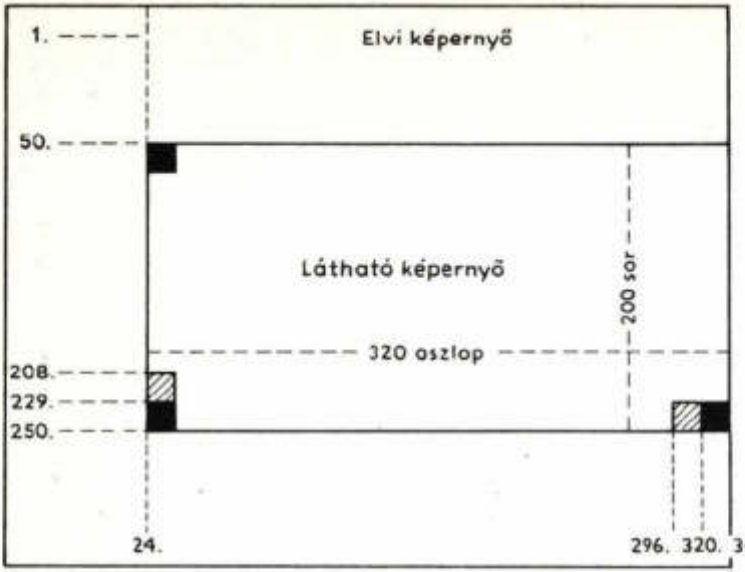
Vízszintes irányban 1 bájt nem elegendő, mivel 343 különböző helyzet lehetséges, ezért 9 bitre van szükség. A 8. folt 9. bitjei egy önálló bájtot alkotnak. A pozicionáló bájtok a következő címeken találhatók:

Folt	Helyzet	Helyzetbájt címe
0.	vízszintes	53248
0.	függőleges	53249
1.	vízszintes	53250
1.	függőleges	53251
2.	vízszintes	53252
2.	függőleges	53253
3.	vízszintes	53254
3.	függőleges	53255
4.	vízszintes	53256
4.	függőleges	53257
5.	vízszintes	53258
5.	függőleges	53259
6.	vízszintes	53260
6.	függőleges	53261
7.	vízszintes	53262
7.	függőleges	53263
	vízszintes 9. bitek	53264

Az 53264 című bájt legalacsonyabb helyértékű bitje a 0., a legmagasabb a 7. folthoz tartozik.

10. feladat

A képernyőn meg kell jeleníteni Magyarország gépkocsi-behozatali adatait 1977-ben, 1980-ban, 1983-ban a 107. ábra szerinti formában. A számadatok mellett a darabszámok hosszával arányos hasábokat is meg kell jeleníteni piros színben.



106. ábra. Foltok elhelyezése a képernyőn

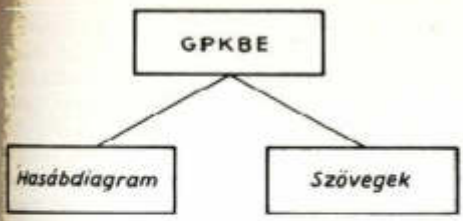
107. ábra. A 10. feladat kiírási képe

A feladat elemzése

A feladat alapvetően két részre bontható: az egyik a hasábdiagram elkészítése, a másik a szöveg kiíratása (108. ábra).

Vizsgáljuk meg először a hasábdiagram elkészítését!

A három hasábnak a behozott gépkocsik számával arányos hosszúságúnak kell lennie. A hasábokat foltokból célszerű felépíteni. Egy folt aránylag alacsony, ezért egy hasábot több foltból kell összerakni. Egyszerre csak 8 folt jeleníthető meg, ezért hasábonként 2 foltot használhatunk. Ez 42 sor magasságú hasábok megjelenítését teszi lehetővé a 200 sorra felbontott képernyőn. Így még mindig elég alacsony, de a kétszerezési lehetőség felhasználásával a magasság legfeljebb 84 sorra növelhető, ami a képernyő magasságának több mint 40%-a.



108. ábra. A 7. feladat szerkezete

Egy folt vízszintes irányú helyfoglalása 24 oszlop. Ha ezt teljesen kihasználnánk, akkor a hasábok nagyon zömökek lennének, és a magassági eltérések nem lennének eléggé feltűnőek. Ezért csökkentsük a vízszintes kiterjedést egyharmadára (8 oszlop).

A foltok piros színű megjelenítése megoldható. Hová helyezzük el a hasábokat? A hasábok alatt és felett kiírás is van. Ezeket a PRINT utasítás segítségével tudjuk kiíratni, legegyszerűbben úgy, hogy közéjük vesszőt teszünk. A kiírást nem a képernyő bal szélső oszlopán kezdjük, hogy az ábra középre kerüljön. A szövegelemek vesszővel való elválasztása miatt a szövegelemek a tíz karakter hosszúságú oszlopok kezdetére kerülnek, ami 80 grafikus oszlopnak felel meg. Az első szövegelem a 11. karakterpozíción fog kezdődni, érdemes tehát a hasábokat a szövegekhez igazítani. A bal oldali hasáb vízszintes beállításánál figyelembe kell venni, hogy az alá és fölé kerülő számértékek csak a 12. karakterpozíción kezdődnek (egy pozíció az előjelnek van fenntartva). Mivel a számok 4 vagy 5 karakter hosszúságúak, akkor lesz tetszetős az ábránk, ha még egy pozícióval jobbra helyezzük a hasábot. Végül is a 13. karakterpozícióban kell az első hasábot megjeleníteni (azaz előtte 12 pozíciót kell üresen hagyni). Ez azt jelenti, hogy a képernyő bal szélétől (ami a 24. oszlop) a $12 \times 9 = 96$. grafikus oszlopra helyezzük a bal oldali hasábot. Ez összesen 120 grafikus oszlopnyi távolság. A következőt 10 karakterpozícióval jobbra, vagyis a $120 + 80 = 200$. grafikus pontra. A harmadik pedig a 280. grafikus pozíción fog kezdődni.

A függőleges irányú elhelyezésnél a szöveg helyfoglalása a meghatározó. Az 59. ábra szerint a cím 3 sort foglal el. Ezután legalább 2 üres sort érdemes kihagyni, majd a darabszámok következnek. Ezután is tanácsos legalább egy üres sort hagyni, hogy a legnagyobb hasáb ne érje el a számokat. Mindez azt jelenti, hogy 7 szövegsornak kell helyet hagyni. Egy szövegsor 8 grafikus sornak felel meg. Ebből következik, hogy a hasábok felső sorát a kép felső határától – ami az 50. sor – $7 \times 8 = 56$. grafikus sorra kell beállítani (106. sor). Mivel egy hasáb két foltból áll, az alsó foltot 42 grafikus sorral lejjebb kell elhelyezni (a nagyítás miatt 42), azaz a 148. grafikus sorban. A hasábok hossza a gépkocsik darabszámával arányos. Egy hasábot $2 \times 21 = 42$ grafikus oszlopra bonthatunk (a nagyítás miatt a sorok száma nem növekszik!). Mondjuk azt, hogy ez feleljen meg a legnagyobb értéknek (110.5). Ennek megfelelően a bal szélső hasáb magassága:

$$\frac{\frac{94.3}{110.5}}{42} = 35{,}8 \sim 36$$

sor lesz. Vagyis az alsó folt teljes magasságú lesz, a felső viszont csak 15 grafikus sort foglal el. A jobb szélső magassága pedig

$$\frac{\frac{89.1}{110.5}}{42} = 33{,}8 \sim 34$$

sor lesz.

A hasábdiagram megtervezésekor részben a szövegeket is megterveztük. Eszerint a címet 3 sorba írjuk ki, utána két üres sort hagyunk. A következő sorba írjuk a darabszámokat a 11. oszloptól kezdve 10 oszloponként. Ezután következik egy üres sor, majd a legfeljebb 84 grafikus sor magasságú hasábok. Egy hasáb több mint 10 szövegsornak felel meg. Ezért célszerű összesen 11 üres szövegsort kihagyni a darabszámok és az évszámok között. Az évszámokat a darabszámokhoz hasonlóan lehet kiíratni. A programban előbb a hasábokat ábrázoló foltokat kell megjeleníteni, utána pedig a szövegeket, mivel a két dolog megjelenítése független egymástól.

A program tervezése

Vizsgáljuk meg előbb a hasábdiagram-készítő funkciót! Ezen belül a következő részfunkciókat kell ellátni ahhoz, hogy a hasábok a képernyőn megjelenjenek:

- foltterületek kijelölése a tárban,
- definiálás (forma),
- a színbájtok beállítása,
- kétszerezés beállítása,
- függőleges elhelyezés,
- vízszintes elhelyezés,
- a bekapcsoló bájt beállítása (bekapcsolás).

Ezek a műveletek különböző bonyolultságúak, de mivel logikailag jól elkülöníthetőek, tárgyaljuk őket külön modulonként. Az egyes foltokat a megismert szabály szerint meg kell számozni, hogy a programban egyedileg tudjuk őket kezelni. A jelkiosztást a 109. ábra mutatja.
A szövegkészítési funkció sokkal egyszerűbb, azt egyetlen modulként is kezelhetjük. A program tehát a 110. ábrán látható modulokból fog állni.

A modulok tervezése

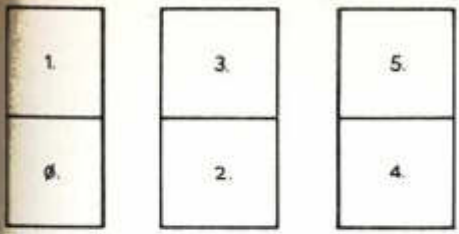
(1) *Területkijelölés*
A modul **eljárás** típusú. Bemeneti adatai a konstansként megadott kezdőcímek, kimenete ezeknek az értékeknek a beírása a foltmutató bájtokba.

Helyezzük el a foltokat a tárprogramok számára fenntartott terület hátsó részén az alábbiak szerint:

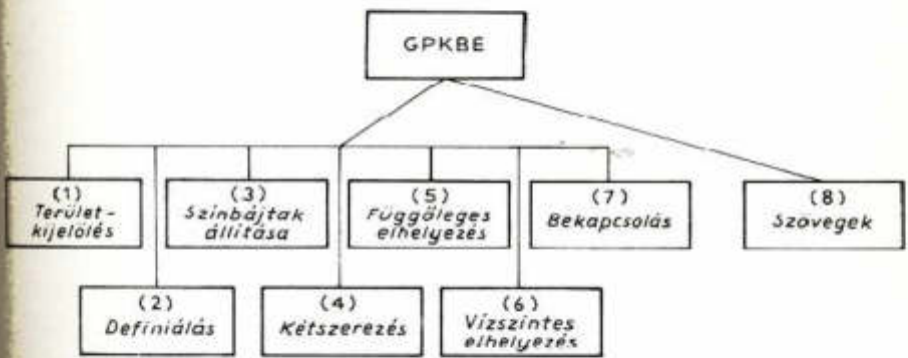
Foltsorszám	Kezdőcím
0	200·64 = 12800
1	201·64 = 12864
2	202·64 = 12928
3	203·64 = 12992
4	204·64 = 13056
5	205·64 = 13120

Ezek szerint a foltmutató bájtokba 200, 201, . . . 205-öt kell beírni. A foltok kezdőcímére a definiálásnál szükségünk lesz, ezért ezeket a K0, . . . , K5 változókban őrizzük meg. A modul műveleteit a 111. ábra mutatja.

(2) *Definiálás*
A modul **eljárás** típusú. Külső bemeneti adata nincs, kimenete a 6 folt formájának meghatározása.



109. ábra. A foltok jelölése



110. ábra. A program szerkezete

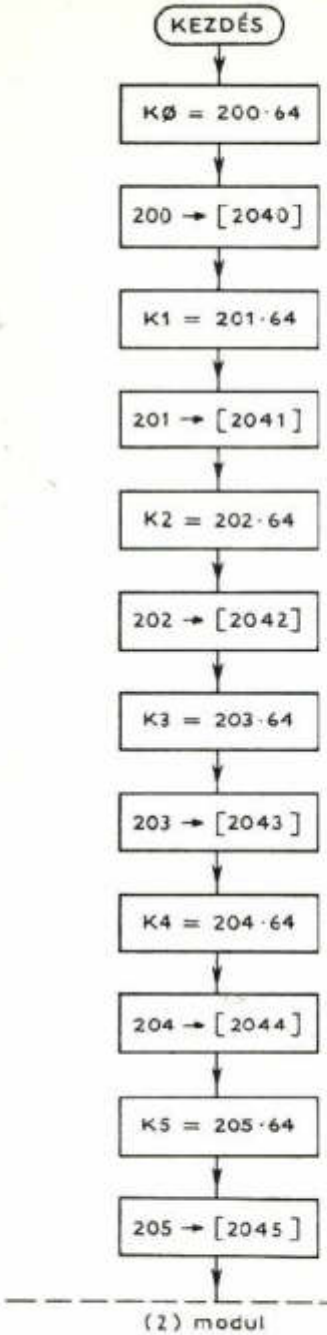
Először nézzük meg, hogy mi az egyes foltok tartalma! Az elemzés szerint a foltok bal oldali egyharmada jelenik meg. Ez azt jelenti, hogy a foltok bal oldali 8 oszlopában minden pont 1 lesz, a többi nulla. A bal oldali hasábot megjelenítő 0 jelű folt a leírtak szerint fog kinézni, de az 1 jelű folt már nem, mivel ennek csak 36–21=15 sorát – méghozzá alsó sorát – kell megjeleníteni. Készítsünk foltdefiniáló táblázatot (pl. kockás papírból), és jelöljük be, hogy a bal oldali hasáb két foltjának mely részeit kell megjeleníteni (vagyis hová kell 1-et vagy 0-t írni)! Ennek eredményét a 112. ábra mutatja. Az ábrán csak a megjelenítendő pontokat jelöltük be, a többi ponthoz a 0-kat nem írtuk be.

A középső hasáb teljes hosszúságú lesz, ezért mind a 2, mind a 3 jelű foltot teljes magasságban ki kell tölteni (113. ábra).

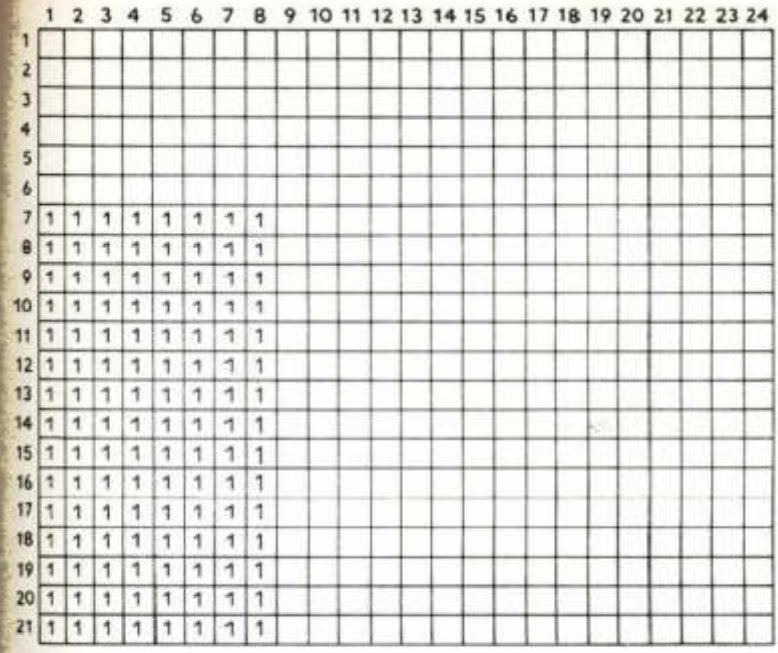
A jobb szélső hasáb alsó foltja szintén teljes hosszúságú lesz. A felső foltban viszont csak 13 sor magasságú lesz a megjelenítendő rész, mivel összesen 34 sor magasságú hasábot kell ábrázolni (114. ábra).

A foltok kitöltésében könnyű észrevenni a szabályt. A 0, 2, 3 és 4 jelű foltok 1. bájtjaiba (bal felső bájtok) csupa 1-et kell írni (decimális 255), majd a következő kettőbe 0-t. A 4. bájtba ismét 255-öt, a következő kettőbe pedig 0-t kell írni. Jól látható, hogy ezt a műveletet ciklusban érdemes elvégezni. Ezeknek a foltoknak a kitöltése azonos, az egyedüli eltérés az, hogy a foltok kezdőcíme más lesz. Tehát minden foltnál más kezdőcímtől kezdve kell a kitöltést elvégezni. Ez teszi lehetővé, hogy a műveletet szubrutinként kódoljuk. A szubrutin hívása előtt a programban meg kell adni a folt aktuális kezdőcímét.

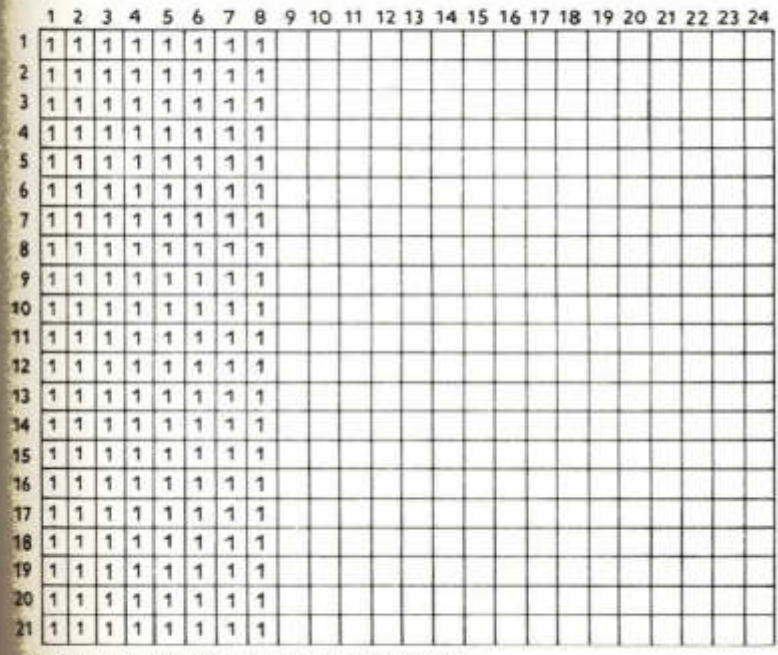
A fenti elv nem alkalmazható az 1 és 5 jelű foltokra, mivel ezek felső része változó mértékben nullákkal van kitöltve. Az alsó rész kitöltési szabálya már megegyezik a 0, 2, 3, 4 jelű foltok kitöltési szabályával. Az 1 és 5 jelű foltok kitöltésében



111. ábra. Az (1) modul műveletei

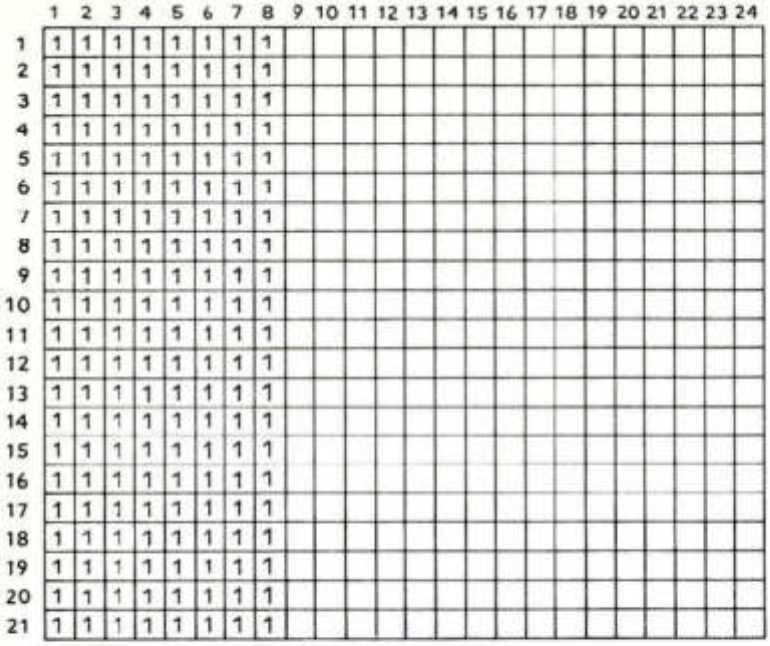


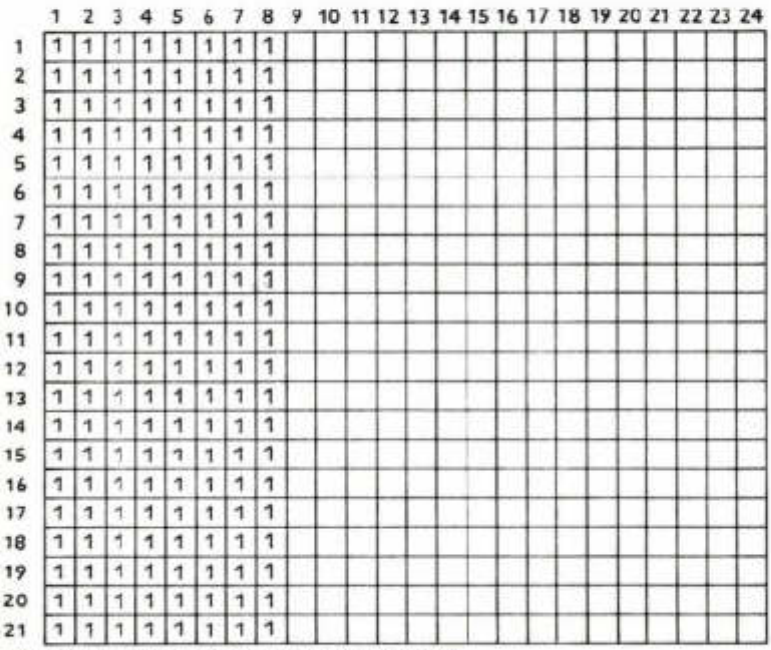
1 folt



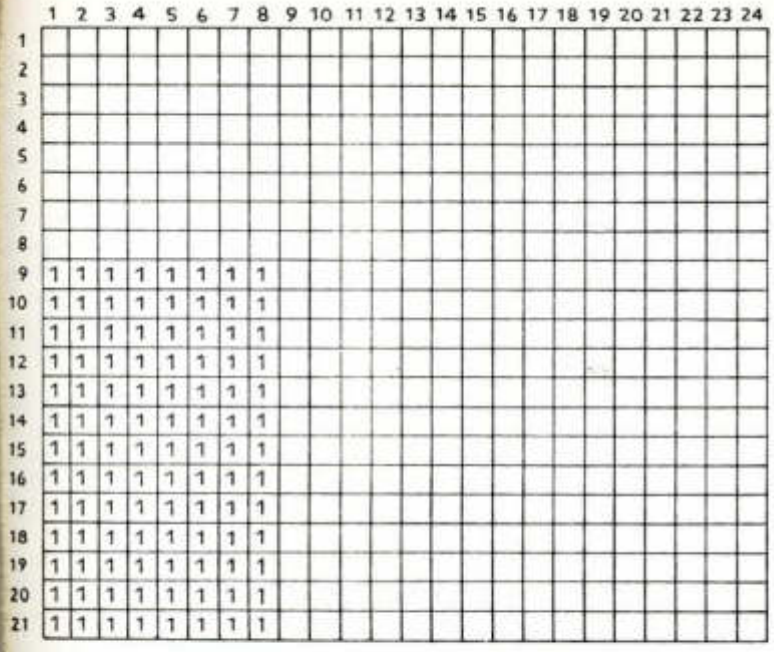
ø folt

112. ábra. A bal oldali hasáb foltjainak definiálása

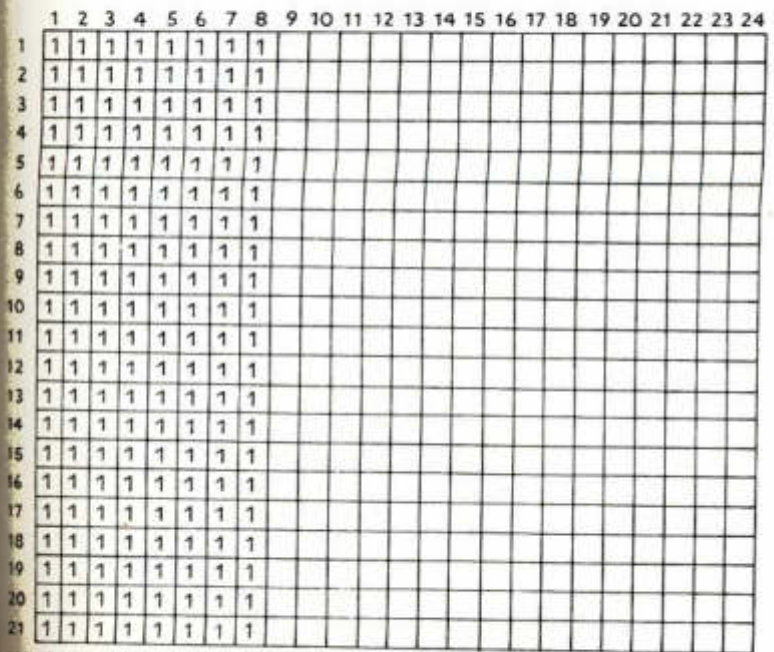




113. ábra. A középső hasáb foltjainak definiálása

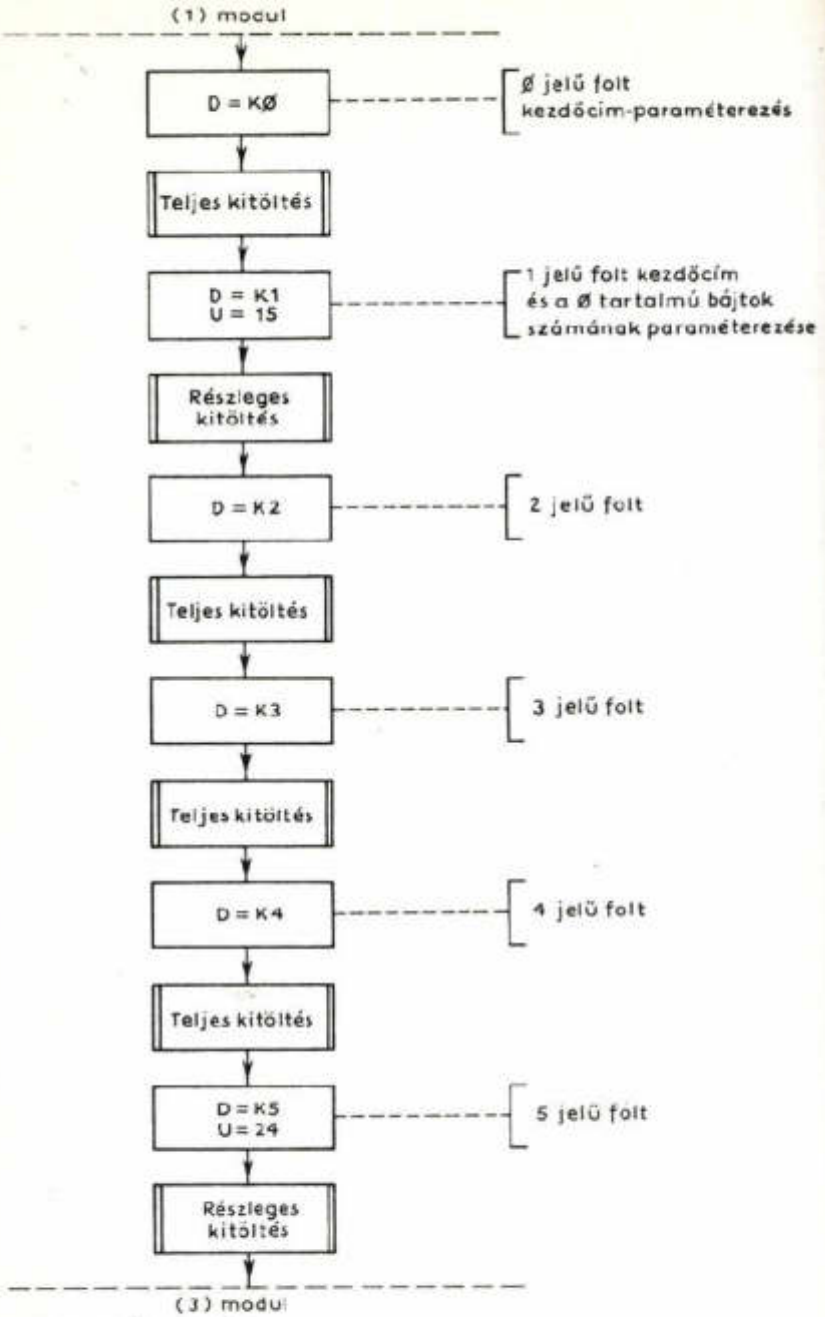


5 folt



4 folt

114. ábra. A jobb szélső hasáb foltjainak definiálása



115. ábra. A (2) modul folyamata

az a szabály fedezhető fel, hogy az első valahány bájtot (az 1 jelűnél 15, az 5 jelűnél 24) 0-val kell kitölteni, majd minden harmadik bájtba 255-öt, a közbensőkbe 0-t kell írni. A két folt kitöltése nemcsak a folt kezdőcímében tér el, hanem abban is, hogy hány bájtba kell 0-t írni az elején. Ez még mindig lehetővé teszi, hogy a definiálást szubrutinként kódoljuk, de itt most a szubrutin hívása előtt két paramétert kell megadni: a kezdőcímet és a nullázandó bájtok számát.

A modul műveleteinek folyamata ezek alapján összeállítható (lásd 115. ábra). A szubrutinok műveletei a 116. és a 117. ábrán láthatók.

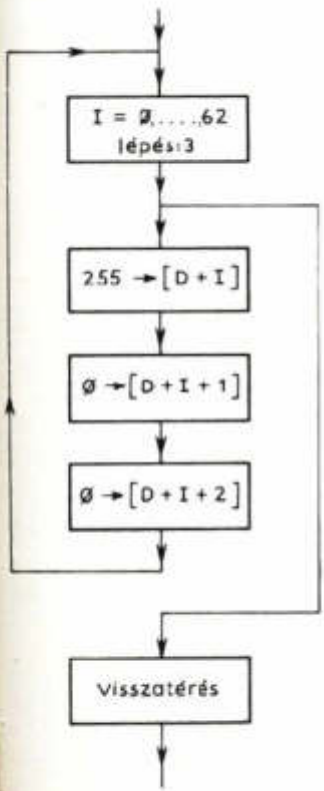
(3) *Színbájtok beállítása*

A modul **eljárás** típusú. Bemenete a foltok piros színét beállító 2 szám, kimenete pedig a színbájtok tartalma (2).

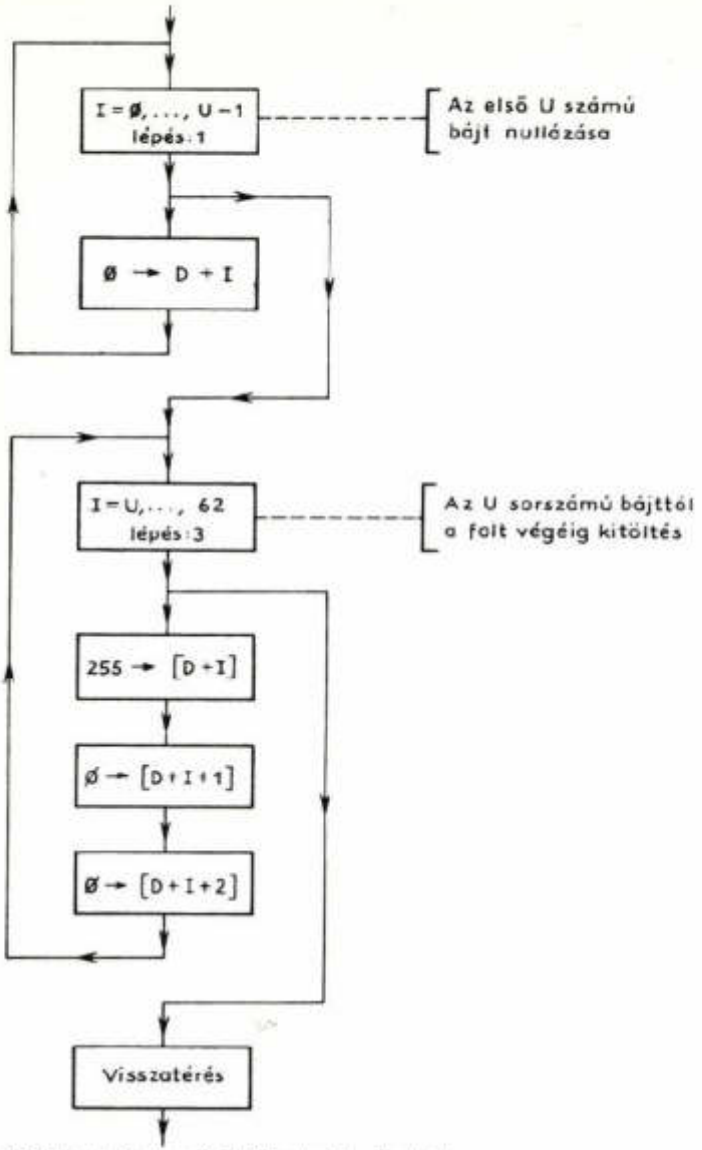
A modul folyamán a 6 folthoz tartozó színbájtba 2-t kell beírni. A színbájtok egymás mellett helyezkednek el a tárban, ezért feltöltésük ciklusban elvégezhető (118. ábra).

(4) *Kétszerezés*

A modul **adat** típusú. Bemenete a függőleges irányban kétszerezendő foltokat meghatározó szám, amit az 53271 című bájtba be kell írni. A kétszerező bájt jobb oldali 6 bájtjába egyet kell írni (119. ábra). Ez a 32 + 16 + 8 + 4 + 2 + 1 = 63 decimális számnak felel meg. A modul egyetlen műveletét a 120. ábra mutatja.



116. ábra. A teljes kitöltési szubrutin műveletei

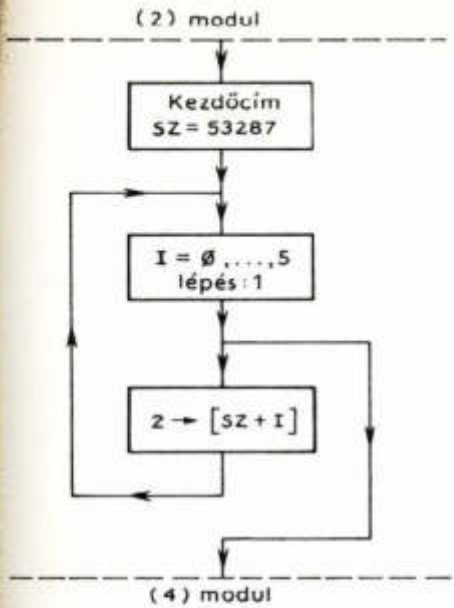


117. ábra. A részleges kitöltési szubrutin műveletei

(5) *Függőleges elhelyezés*

A modul **eljárás** típusú. Bemeneti adata a foltok függőleges helyzetét meghatározó két adat (106 és 148), kimenete a függőleges beállítást tartalmazó 6 bájtba írt érték.

Az elemzésnél megállapítottuk, hogy az 1, 3 és 5 jelű foltokat a 148. grafikus sortól kezdve kell elhelyezni, a másik hármat pedig 42 sorral lejjebb. Az ezeknek megfelelő értéket kell a függőleges pozíciót meghatározó 6 bájtba beírni (120. ábra).



118. ábra. A (3) modul folyamata

	128	64	32	16	8	4	2	1
53271	0	0	1	1	1	1	1	1
	7.	6.	5.	4.	3.	2.	1.	0.

119. ábra. A kétszerező bájt tartalma

(6) *Vízszintes elhelyezés*

A modul **eljárás** típusú. Bemenete a foltok vízszintes helyzetét meghatározó három adat: 120, 200, 280, kimenete a vízszintes beállítást tartalmazó 7 bájtba írt érték.

Az elemzés alapján a 0 és 1 jelű foltokat a 120., a 2 és 3 jelű foltokat a 200., a 4 és 5 jelű foltokat a 280. grafikus oszlopba kell állítani. Ezeket az értékeket kell a foltok vízszintes pozicionáló bájtjaiba beírni.

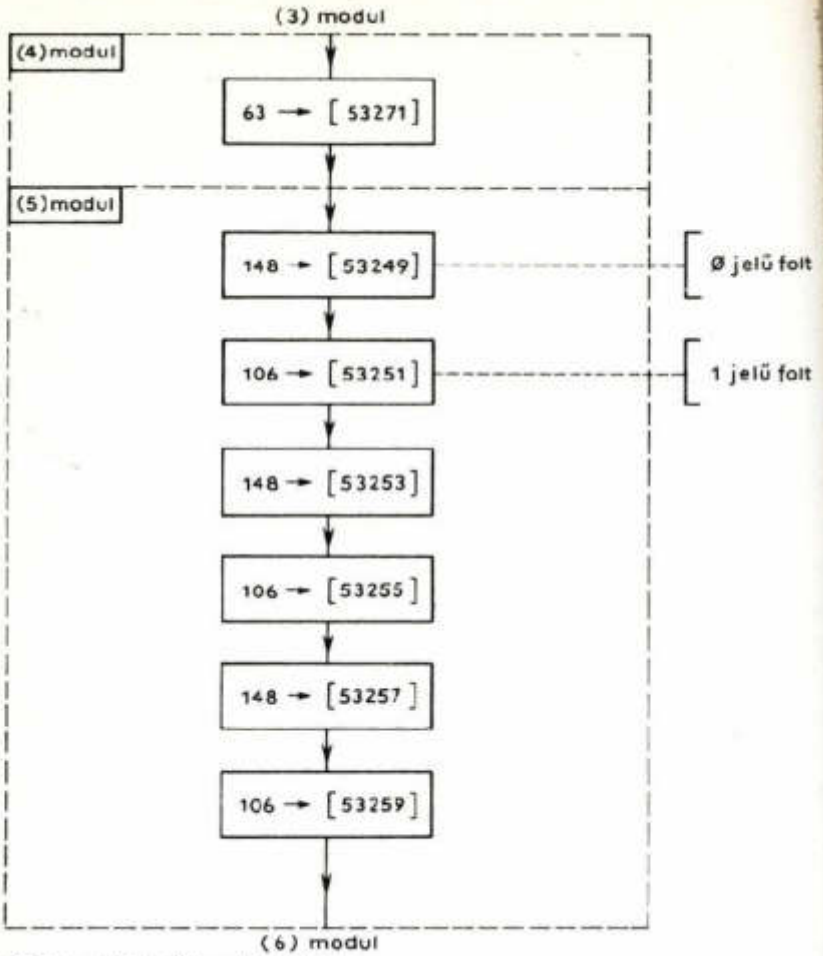
A 4 és 5 jelű foltoknál a pozíció 256-nál magasabb sorszámú oszlopra esik, ezért az ezekhez a foltokhoz tartozó 9. biteket 1-be kell állítani a 9. bitek bájtjában (121. ábra).

A beállított érték 32 + 16 = 48. A két folt vízszintes pozicionáló bájtjába már csak 280 – 256 = 24-et kell beírni, mivel a 9. bit 256-nak felel meg. A modul műveleteit a 122. ábra mutatja.

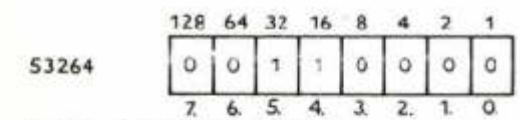
(7) *Bekapcsolás*

A modul **adat** típusú. Bemenete a bekapcsolni kívánt 6 foltot meghatározó szám, kimenete pedig ennek az értéknek a beírása a bekapcsoló bájtba.

A bekapcsoló bájt jobb oldali 6 bitjébe 1-et kell írni a kétszerező bájthoz hasonlóan. Ez azt jelenti, hogy 63-at kell beírni az 53269 című bájtba (122. ábra).



120. ábra. A (4) és (5) modul műveletei

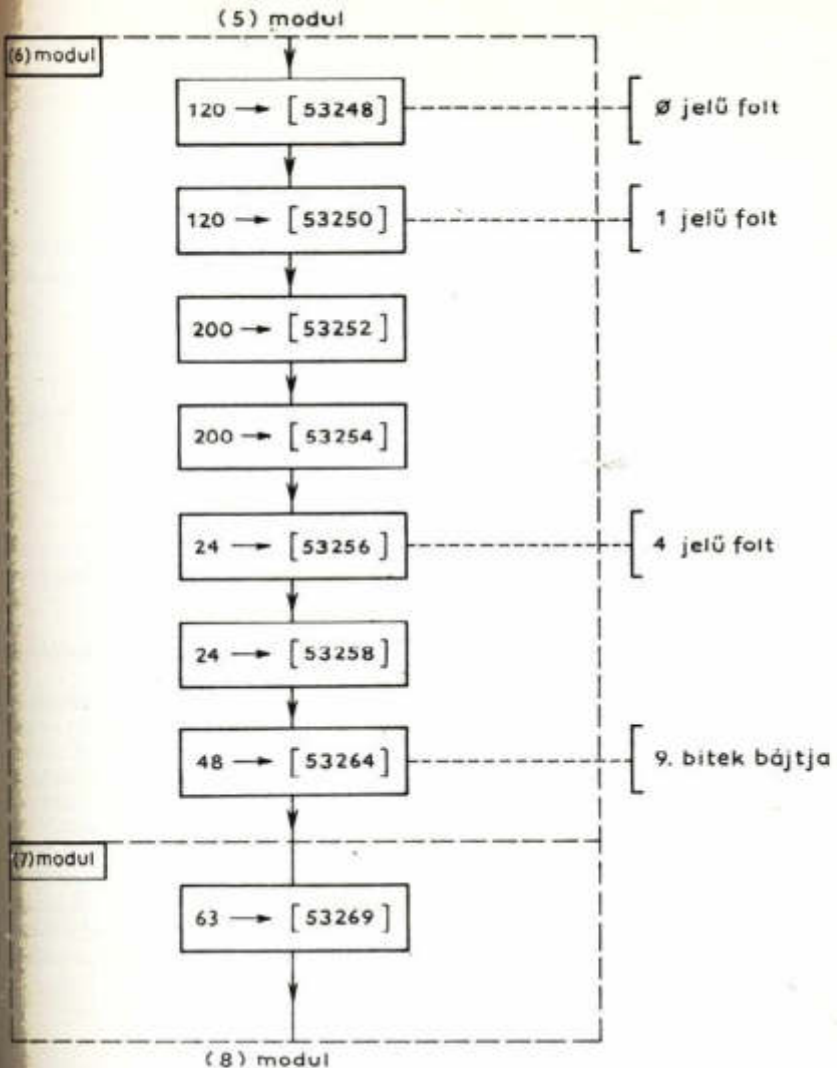


121. ábra. A 9. bitek bájt tartalma

(8) *Szövegek*

A modul **eljárás** típusú. Bemenetét a kiírni kívánt szövegek képezik, kimenete pedig a képernyőn megjelenő szöveg.

A modulban a specifikációban megadott szövegelemeket kell az elemzésnek megfelelő beosztásban kiírni (123. ábra).



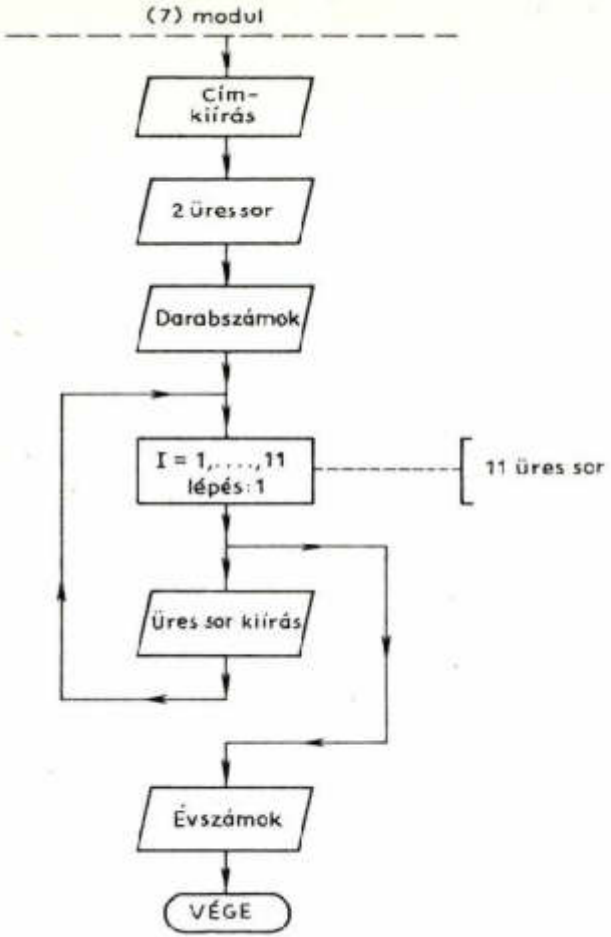
122. ábra. A (6) és (7) modul műveletei

Kódolás

A kódolásból néhány kritikus és újszerű részt mutatunk be. Az (1) modulban a területkijelölések közül az elsőt bemutatjuk.

```
70 K0=200*64
80 POKE 2040,200
```

Látható, hogy K0-nak nem a kiszámított értéket adtuk, hanem a gépre bíztuk a számolást. Így kisebb a veszélye, hogy hibás számot adunk meg.



123. ábra. A (8) modul müveletei

A (2) modulban az első két folt definiálása a következő:

```
210 D=K0
220 GOSUB  1000
230 D=K1
240 U=15
250 GOSUB  1200
```

Jól látható, hogy a szubrutinok hívása előtt elvégezzük a paraméterezést. A teljes kitöltési szubrutinban a ciklusváltozót 0-ról indítjuk el, hogy ha hozzáadjuk a folt-terület címét, akkor az első lépésben valóban a terület első bájtjának értékét állítsuk be. A következő lépésben I=1, így az eggyel magasabb című (második) bájtot kapjuk:

```
1020 FOR I=0 TO 62 STEP 3
1030 POKE D+I,255
1040 POKE D+I+1,0
1050 POKE D+I+2,0
1060 NEXT I
1070 REM* CIKLUS VEG *
1080 RETURN
```

A részleges kitöltési szubrutin ehhez hasonlóan működik, azzal a különbséggel, hogy a szubrutin elején egy nullázási ciklus is van:

```
1220 FOR I=0 TO U-1
1230 POKE D+I,0
1240 NEXT I
```

A szövegkiírásból érdemes megtekinteni a 11 üres sort kiíró ciklust:

```
750 FOR I=1 TO 11
760 PRINT
770 NEXT I
```

Ez a ciklus 11 PRINT utasítás leírásától kímél meg minket. 2–3 soremelésre még nem célszerű alkalmazni.

A megoldás értékelése

A feladatot általánosabb módszerek felhasználásával is megoldhattuk volna. Ezek ugyan nagyobb fáradságba kerültek volna, de több feladatnál felhasználható eredményt szolgáltattak volna. Rugalmasabb lett volna a megoldás akkor, ha például a foltok függőleges méretét a program számolta volna ki egy begépelt adat alapján. (Ilyenkor az adatból kellene meghatározni, hogy melyik biteket kell 1-be állítani.) Ez nyilván nem egy egyszerű műveletsorozattal oldható meg, de a kapott megoldás jól hasznosítható lenne ehhez hasonló feladatokban is.

A feladatot úgy is meg lehetne oldani, hogy mind a 6 foltot függőlegesen azonos, maximális méretre definiáljuk, és a hasábokat az 1, 3 és 5 jelű foltok függőleges pozíciójának meghatározásával alakítjuk ki. Ez az előzőnél egyszerűbb lehetőséget nyújtana. Gyakorlásképpen ezt a megoldást javasoljuk.

Ellenőrző kérdések és feladatok

1. Készítsük el a 10. feladat oszlopdiagramjait úgy, hogy hat teljes magasságú foltot rajzoljunk, és ezek függőleges elhelyezésével alakítsuk ki az oszlopokat!

2. Módosítsuk úgy a programot, hogy az oszlopok magasságát a billentyűzetről beolvasott adat értékétől függően állítsuk be!

3. Egy foltban rajzoljunk egy helikoptert, amelyet a billentyűzetről beolvasott adatok alapján a képernyőn bárhol meg lehet jeleníteni!

4. Az 5. feladat kiírásait vonalakkal megrajzolt táblázatban készítsük el!

5. Próbálja meg a 8. feladatot úgy megoldani, hogy a csatatér széleit a képernyőn jelenítse meg! (Nem minden géppel oldható meg!)

IRODALOM

ASZALÓS J. – ERKI I.: Bevezetés a strukturált programozásba. SZÁMOK, Budapest, 1980.

DR. KOCSIS ANDRÁS: Programozás BASIC nyelven. SZÁMALK, Budapest, 1983.

DR. KOCSIS ANDRÁS: Programozási kézikönyv (BASIC). SZÁMALK, Budapest, 1983.

LOTT, R. W.: BASIC with Business Applications. John Wiley & Sons, USA, 1982.

NAGY KÁLMÁN: Strukturált programozás COBOL nyelven. SZÁMOK, Budapest, 1980.

VICKERS, S.: Sinclair ZX SPECTRUM BASIC-Programmierung. Verlag Cooperation, München, 1983.

VICKERS, S. – BRADBEER, R.: Sinclair ZX SPECTRUM Einführung. Verlag Cooperation, München, 1983.

ZAKS, R.: The CP/M Handbook with MPM. SYBEX Inc, 1980.

Commodore 64 Programmer's Reference Guide, Commodore Business Machines, Inc. and Howard W. Sams & Co. Inc. Slough (England).

HT Iskolaszámítógép; HT–2080Z Számítógép BASIC kézikönyv. Híradástechnika Szövetkezet, Budapest, 1983.

HT–1080Z Iskolaszámítógép; HT–2080Z Számítógép-használati útmutató. Híradástechnika Szövetkezet, Budapest, 1984.

PRIMO felhasználói kézikönyv. Ifjúsági Lap- és Könyvkiadó Vállalat, Budapest, 1984.

VIC–1541 User's Manual, Commodore Business Machines, Inc, 1981.

FÜGGELÉK

A KÖNYVBEN SZEREPLŐ FONTOSABB FOGALMAK ÉS MAGYARÁZATUK

Adat: Tények és elképzelések nem értelmezett, de értelmezhető formában való közlése. Az adat a feldolgozás tárgya és eredménye.

Adatállomány: Sok egyed tulajdonságainak értékeiből álló, valamilyen szempontból összetartozó adathalmaz.

Adatállomány-feldolgozás: A tárba beolvasott rekordokon valamilyen művelet végrehajtása a feldolgozás céljának megfelelően.

Adatállomány-létrehozás: A tárban létrehozott rekordok kimásolása a háttértárolón megnyitott adatállományba.

Adatállományok módosítása: Egy adatállományhoz új rekordok hozzáadása, meglevő rekordok törlése és a meglevő rekord adatainak módosítása.

Adatállomány-nyitás: Egy megadott nevű adatállomány és a gép közötti kapcsolat létrehozása.

Adatállomány-zárás: Egy megadott nevű adatállomány és a gép közötti kapcsolat megszüntetése.

Adatkezelés: Az adatokkal kapcsolatos műveletek összessége (beolvasás, ellenőrzés, rendszerezés, azonosítás, tárolás stb.).

Adatmodul: Adatok meghatározására (definiálás, beolvasás, kezdőértékadás) szolgál.

Algoritmus: Valamilyen feladat megoldását célzó műveletsorozat.

Állapotbájt: Adatállomány-olvasáskor az állomány állapotáról tartalmaz információt (az állományvég elérésekor értéke 1, egyébként 0).

Argumentum: A függvény bemeneti adata (vagy független változója), amelyből a függvény egy eredményadatot állít elő.

ASCII kód: A számítógépen használt jelekhez és a BASIC karaktereihez hozzárendelt bájttartalmak együttese.

Bájt: 8 bitből álló adattároló egység, melynek 256 különböző állapota van. Az állapotok tetszőleges karaktereknek feleltethetők meg valamilyen rendszer szerint (pl. ASCII kódrendszer).

Bemeneti adat: A feldolgozás kiinduló adatai, amelyekből az eredményadatok készülnek.

Betöltés: A külső tárolóeszközről a gép tárjába való programmásolás.

Billentyűzet: Egy számítógéphez tartozó billentyűk és nyomógombok.

Bit: Kétállapotú adattároló egység.

C helyőr *(Sinclair-gépek):* Az L helyőr alesete, amelyben csak nagybetűk írása lehetséges.

Ciklus: Különböző adatokkal ismételten végrehajtott műveletsorozat.

Ciklusfeltétel: A ciklus befejezését vagy folytatását meghatározó feltétel.

Ciklusmag: A ciklikusan végrehajtott műveletsor a ciklusban.

Ciklusváltozó: Az az adat, amelynek értéke meghatározza a ciklus folytatását vagy befejezését.

Cím: A tár egy bájtjához tartozó szám, amelynek megadásával tartalmát ki lehet olvasni, illetve adatot lehet beleírni.

CP/M: Mikroszámítógép operációs rendszere.

Csatorna: A számítógép és a perifériák között fizikai kapcsolatot valósít meg.

Egyed: Egy rendszernek olyan eleme, amelyet adatokkal kívánunk leírni.

Egykijáratúság: A modul végrehajtása mindig ugyanazon az utasításszámon kezdődik, és mindig ugyanazon a soron fejeződik be.

Egymásba ágyazott ciklus: Egy ciklus magjában egy másik ciklus helyezkedik el.

Egymásba ágyazott szubrutin: A program egy szubrutinból egy másikat is hív.

E helyőr *(Sinclair-gépek):* A CAPS SHIFT és SYMBOL SHIFT billentyű együttes lenyomásakor jelenik meg. Ekkor a billentyűk feletti zöld feliratok írhatók ki.

Elágazás: Valamilyen feltételtől függően egy vagy több művelet közül az egyik kiválasztása és végrehajtása.

Eljárásmodul: Tényleges műveletet végző modul.

Eredményadat: A feldolgozási folyamat terméke, valamilyen számérték vagy szöveg.

Érték: Egy egyed tulajdonságának konkrét megjelenése.

Feldolgozás: Valamilyen feladat megoldása végett adatokon végzett műveletek összessége.

Felhasználói függvény: A felhasználó által definiált függvény, amely egyetlen programban érvényes.

Feltételváltozó: Az a változó, amelynek értéke eldönti, hogy az elágazás melyik ága hajtódik végre.

Felülről lefelé haladás (top-down): A feladat lépésenkénti, egyre részletesebb felbontása részfeladatokra.

Folt *(C64):* Vízszintes irányban 24, függőleges irányban 21 pontból álló, a felhasználó által meghatározható formájú jel, amely a képernyő bármely részén megjeleníthető.

Folyamatábra: A program (modul) műveleteinek ábrázolása szabványos jelekkel.

Főág: A programnak az a része, amely nem tartalmaz szubrutint, és amely a szubrutinok végrehajtását vezérli.

Főprogram: Több programból álló szerkezet vezérlőprogramja.

Funkcionális billentyű: Lenyomásakor a számítógép valamilyen műveletet végez el.

Függvény: Egy független változó értékből (argumentumból) valamilyen szabály szerint egy eredményértéket számít ki.

G helyőr *(Sinclair-gépek):* A CAPS SHIFT és a 9 billentyű együttes lenyomása után jelenik meg. Ekkor a billentyűk grafikus jelei írhatók ki.

Gyűjtő: Hasonló feladatot betöltő változók összegét tartalmazó változó, amelynek tartalma (összeg) ciklusban alakul ki.

Helyőr: Egy (villogó) karakter, mely a következő kiírandó karakter helyét mutatja.

Kalkulátormód: Lásd közvetlen mód.

Karakter: Egy jel (betű, szám vagy speciális jel), amely a programozásban valamilyen funkciót lát el.

Kazettás tároló: A magnetofon elve szerint működő adatot tároló berendezés.

Képernyősor: A képernyő bal széle és jobb széle között megjelenített karaktersor.

Kétszeres pontosságú változó: A normál numerikus változónál nagyobb pontossággal tárolt változó. A kétszeres pontosságú változó tárolásához a gép több bájtot vesz igénybe, és több számjeggyel képes kiírni, mint a normál numerikus változókat.

K helyőr *(Sinclair-gépek):* Utasítássorszám vagy kulcsszó begépelését teszi lehetővé.

Kiíró: Számítógép által vezérelt elektronikus íróberendezés.

Kimentés: A program kimásolása a gép tárjából külső tárolóberendezésre.

Kódlap: Programkód írását segítő űrlap.

Kódolás: Az algoritmusban meghatározott műveletek megfogalmazása egy programozási nyelven.

Közvetlen mód: Egy utasítássorba írt sorszám nélküli utasítás(oka)t a gép azonnal végrehajt a sorzáró billentyű lenyomása után.

Kulcsszó: Valamilyen programozási nyelv által lefoglalt szó, amelyet csak a nyelv által előírt célra és módon lehet használni (pl. nem lehet adatnév).

Külső tároló: A számítógéphez hozzákapcsolt adattároló berendezés, amely adatok hosszú távú tárolására alkalmas.

L helyőr *((Sinclair-gépek):* A billentyűkön levő főjelek (betűknél kisbetűk) begépelését teszi lehetővé.

Logikai ÉS reláció: A reláció két feltétel egyidejű teljesülése esetén áll fenn,.

Logikai hivatkozási szám: Egy adatállomány programon belüli azonosítója.

Logikai VAGY reláció: A reláció egyik vagy másik feltétel teljesülése esetén fennáll.

Mágneslemez: Lemez alakú, mágneses elven működő adattároló.

Menü: A programmal elvégezhető részfeladatok kiírása, amelyből a felhasználó valamelyiket kiválaszthatja.

Mező: A rekord egy önállóan értelmezett adata (egy egyed egy tulajdonságának értéke).

Modul: Az egész feladaton belül elkülönített részfeladat, amely önállóan is elkészíthető.

Moduláris programozás: A feladatot előbb modulokra kell bontani, ezeket önállóan kell elkészíteni, és belőlük kell összeállítani a programot.

Modulok közötti kapcsolat: A modulok végrehajtási sorrendjét meghatározó viszony.

Operációs rendszer: Programok összessége, amelyek a számítógéprendszer egészét irányítják, az elemek működését összehangolják, és a programozónak szolgáltatásokat nyújtanak.

Paraméterezés: Szubrutinok bemeneti adatainak beállítása a hívási rész követelményei szerint.

Parancs: Sorszám nélküli utasítás(ok), amelye(ke)t a gép a sorzáró gomb lenyomása után azonnal végrehajt. Vannak olyan kulcsszavak, amelyek parancsban nem használhatók (pl. INPUT), illetve vannak olyan kulcsszavak, amelyek csak parancsként használhatók (pl. AUTO).

Perifériák: A számítógéphez kapcsolt kiegészítő berendezések (nyomtató, háttértár, kijelző stb.).

Program: Utasítások sorozata, melyek megmondják a számítógépnek, hogy milyen műveleteket kell végrehajtani a feladat megoldása érdekében.

Programdokumentáció: A programról rendelkezésre álló írásos információ.

Programkódlista: A program egyes utasítássorai alkotják.

Programkönyvtár: Egy mágneses tárolón tárolt programok összessége.

Rekord: Az adatállomány része, amely egy egyed tulajdonságainak értékét tartalmazó adathalmaz.

Rendezés: Egy tömb vagy egy adatállomány adatcsoportjainak (pl. egy tömbsor vagy egy rekord) valamilyen szempont szerint csökkenő vagy növekvő rendben való összeállítása, ami lehetővé teszi az adatcsoportok megadott szempont szerinti rendezett kiolvasását.

Sorszám: Az utasítássor programon belüli végrehajtását meghatározó szám.

Sorzáró billentyű: A begépelés során egy sort fejez be.

Szekvencia: Két vagy több utasításból álló műveletsorozat, amelynek minden elemét végre kell hajtani.

Szekvenciális adatállomány: Bármilyen (teljes körű vagy csak egy rekordra vonatkozó) feldolgozás esetén a rekordokat az állomány elejétől a végéig be kell olvasni.

Szintaktikai hiba: A programozási nyelv szabályainak be nem tartása miatt fellépő formai hiba.

Szubrutin: A program tetszés szerinti helyéről hívható részfeladat, amely lehet egy modulrészlet, egy modul vagy több modul.

Tár: A számítógép adattároló egysége.

Tesztelés: A program kipróbálása.

Többszörös utasítás: Egy utasítássorba (egy sorszám után) írt és elválasztójellel (:) elválasztott utasítások.

Tömb: Egy *n* dimenziós mátrix, amelynek elemei valamilyen szempontból összetartozó adatok.

Tulajdonság: Az egyedek jellemzői, amelyekkel ezek leírhatók.

Utasítássor: Sorkezdettől a sorzáró karakter lenyomásáig begépelt jelek sorozata, amely több képernyősort is elfoglalhat.

Utasítás tárgya: Legtöbbször valamilyen adat vagy utasítássorszám. Bizonyos esetekben hiányzik.

Üzenet: Valamilyen művelet végrehajtása után vagy hiba esetén a gép által kiírt szöveg a képernyőre, a felhasználó tájékoztatására.

Változó: Szimbolikus névvel ellátott adat.

Vektor: Egy egydimenziós tömb.

Vezérlőmodul: Egy vagy több modul végrehajtását irányítja.

Véletlenszám-generálás: Egy adott tartományba eső szám vagy számsorozat előállítása, amelynek tagjai szabálytalanul követik egymást.

BASIC UTASÍTÁSOK, PARANCSOK ÉS KULCSSZAVAK

Jelmagyarázat

F = függvény
K = kiegészítő kulcsszó
P = parancs
U = utasítás

Kulcsszó	Funkció	Típus	Leírás old.	C–64	HT 1080Z	PRIMO	Spectrum
ABS(A) *	A abszolút érték számítása	F	155	X	X	X	X
ACS A	A arkusz koszinusza	F	–				X
AND	Logikai ÉS kapcsolat szava	K	–	X	X	X	
ASC(A$)	A$ első karakterének ASCII kódja	F	–	X	X	X	
ASN A	A arkusz szinusza	F	–				X
ATN (A) *	A arkusz tangense	F	–	X	X		X
ATTR X,Y	Az X sorban és Y oszlopban kiadott jel megjelenési módja	F	–				X
AUTO (A,B)	Automatikus sorszámozás A-tól B lépésben	P	75		X	X	
BEEP X,Y	Y magasságú hangkiadás X s-ig	F	–				X
BIN	Az utána írt számot binárisan olvassa	F					X
BORDER A	Az A-nak megfelelő színre állítja a keretet	U	–				X
BRIGHT A	Az A-nak megfelelő fényességet állít be	U	–				X
CALL (X)	A X címen levő gépi kódú program hívása	F	–			X	
CDBL (X)	X szám kétszeres pontosságú változó	F	–		X	X	
CHR$ (A) *	Az A számnak megfelelő jelet adja vissza	F	–	X	X	X	X
CINT (A)	A-t egész típusúra alakítja	F	–		X	X	
CIRCLE (X,Y,Z)	X, Y középpontú, Z sugarú kört rajzol	F	–				X
CLEAR (A)	Az A számú bájtot és a változókat törli	P	–		X	X	X

Kulcsszó	Funkció	Típus	Leírás old.	C–64	HT–1080Z	PRIMO	Spectrum
CLOAD#–1,A	Az A nevű programot betölti	P	43		X		
CLOAD? A	Az A nevű állományt összehasonlítja a tárban levővel	P	42		X		
CLOSE A	Lezárja az A logikai hivatkozási számú állományt	U	194	X		X	X
CLR	A változókat törli	U	224	X			
CLS	Képernyőtörlés	U			X	X	X
CMD A	A kiírást az A logikai hivatkozási számú állományra irányítja	U	78	X			
CODE A$	A szöveg első jelének kódját adja	F	–				X
CONT	A programot a leállás helyéről folytatja	P	81	X	X	X	
CONTINUE	Ua., mint CONT	P	–				X
COPY	Az első 22 sort kinyomtatja	U	–				X
COS (A) *	A koszinusza	F	–	X	X	X	X
CSAVE#–1,A	A tárban levő programot A névvel kimásolja	P	42		X		
CSNG (A)	A-t valósra konvertálja	F	–			X	
CREATE	Állomány létrehozása	U	–				X
DATA	Adatokat helyez el	U	119	X	X	X	X
DEFDBL	Az utána írt változók kétszeres pontosságúak	U	–		X	X	
DEF FN	Felhasználói függvény definiálása	U	156	X			X
DEFINT	Az utána írt változok egész típusúak	U	–		X	X	
DEFSNG	Az utána írt változók numerikus változók	U	–		X	X	
DEFSTR	Az utána írt változók szövegesek	U	–		X	X	
DELETE	Az utána álló sorszámú utasításloka)t törli	P	–		X	X	
DIM	Tömböt definiál	U	173	X	X	X	X
DRAW X,Y	A jelenlegi pont és X,Y között egyenest rajzol	U	–				X
DRAW X,Y,Z	A jelenlegi pont és X,Y között egyenest húz Z-vel elforgatja	U	–				X

Kulcsszó	Funkció	Típus	Leírás old.	C–64	HT–1080Z	PRIMO	Spectrum
EDIT A	A sorszámú utasítás módosítása	P	301 303		X	X	X
ELSE	Lásd az IF THEN ELSE-nél	K	–		X	X	
END	Program vége	U	87	X	X	X	X
ERL	A hibás sorszámot adja vissza	F	–		X	X	
ERR	A hibakód számát adja vissza	F	–		X	X	
ERROR A	Az A hibakódot állítja elő	U	–		X		
EXP (A) *	Az *e* A-adik hatványát adja vissza	F		X	X	X	X
FLASH A	A-nak megfelelően villogást ad	U	–				X
FIX (X)	X-ről leválasztja a törtrészt	F	–		X	X	
FN *név* (A)	Az FN *név* függvény A pontbeli értékét adja	F	–	X			X
FOR TO STEP	Ciklust hajt végre	U	154	X	X	X	X
FRE (0)	A tárban a szabad hely mérete bájtban	F	–	X	X	X	
FRE (0$)	A szöveges változók szabad tárolóhelye	F	–			X	
GET A$	A lenyomott billentyű kódja A$-ba	U	203	X			
GET # A	Az A számú eszközről egy bájtot olvas	U	–	X			
GOSUB A	Szubrutint hív az A sorszámon	U	116	X	X	X	X
GO TO A	A végrehajtás A sorszámon folytatódik	U	96	X	X	X	X
IF THEN	Elágazási utasítás	U	94	X	X	X	X
IF THEN ELSE	Elágazási utasítás	U	96		X		X
INK A	A képernyőre írás színét állítja be	U	–				X
INKEY$	A lenyomott billentyű kódját adja vissza	U	203		X	X	X
INP (A)	Az A kapuról egy bájtot beolvas	F	–		X	X	
INPUT	A billentyűzetről adatot olvas be	U	105	X	X	X	X

Kulcsszó	Funkció	Típus	Leírás old.	C–64	HT–1080Z	PRIMO	Spectrum
INPUT # A	Az A logikai hivatkozási számú állományról egy rekordot olvas	U	241	X		X	
INVERSE A	A-nak megfelelően negatív írást ad ki	U	–				X
INT (A) *	A-nál kisebb egész értéket ad	F	156	X	X	X	X
LEFT$(A$,B)	B sorszámú karakter az A$ bal oldaláról	F		X	X	X	
LEN(A$) *	A$ karakterszámát adja vissza	F	–	X	X	X	X
LET	Értékadás	U	84	X	X	X	X
LINE	Szöveges változó beolvasása INPUT-tal	U	–				X
LIST	Programlistázás	P	77	X	X	X	X
LLIST	Programlistázás nyomtatóra	P	79		X	X	X
LOAD	Programbetöltés	P	40	X	X	X	X
LOG (X)	X algoritmusát adja vissza	U	–	X		X	
LPRINT	Kiírás nyomtatóra	U	195		X	X	X
MEM	A tárban nem használt bájtok száma	F	–		X		
MERGE	A betöltött és a bent levő programból egy újat hoz létre	U	–				X
MID$ (A$,B,C)	Az A$ szöveg B-edik és C-edik helyértéke közötti szöveget adja	F	–	X	X	X	
NEW	A bent levő programot törli	P	21	X	X	X	X
NEXT	Cikluszárás	U	154	X	X	X	X
NOT	Logikai tagadás	–	–	X	X	X	X
ON ERROR GOTO	Hiba esetén elágazás	U	–		X	X	
ON A GOTO	A-tól függően ágazik el	U	–	X	X	X	
ON A GOSUB	A-tól függően egy szubrutint hív	U	–	X	X	X	
OPEN	Megnyit egy perifériát	U	192	X		X	
OR	Logikai VAGY kapcsolatot jelöl	U	217	X	X	X	X
OUT(A,B) *	Az A kapun B számot ír ki	U	–		X	X	X
OVER A	A-nak megfelelő módon egy jelet átír	U	–				X

Kulcsszó	Funkció	Típus	Leírás old.	C–64	HT–1080Z	PRIMO	Spectrum
PAPER A	A-nak megfelelően alapszínt állít	F	–				X
PAUSE A	A végrehajtást leállítja és A képet mutat	F	–				X
PEEK (A) *	Az A címen levő bájt tartalma	F	260	X	X	X	X
PI	π értékét adja	F	–				X
PLOT A,X,Y	A-nak megfelelő színű pontot rajzol az X,Y koordinátákba	F	–				X
POINT (X,Y)	Az X,Y koordinátájú pont állapotát vizsgálja	F			X	X	
POKE A,B	Az A című bájtba B értékét írja	U	258	X	X	X	X
POS (0)	A helyőr jelenlegi helye egy sorban	F	–	X	X		
PRINT	Adatkiírás	U	86	X	X	X	X
PRINT # A	Az A logikai hivatkozási számú állományra írás	U	192	X			
RANDOM	Véletlenszámsort véletlenszerűvé tesz	U	210		X	X	
RANDOMIZE	Véletlenszámsort véletlenné tesz	U	210				X
RE (A,B)	A-tól B lépésben újraszámozza az utasításokat	P	–		X		
READ	Adatolvasás a DATA-ból	U	119	X	X	X	X
REM	Megjegyzés, magyarázat, tárolás	U	76	X	X	X	X
RESET (X,Y)	A X,Y koordinátájú pontot kikapcsolja	F	–		X	X	
RESTORE	A DATA olvasást visszaállítja az első adatra	U	–	X	X	X	X
RESUME	A hibakezelést befejezi, és visszatér a hibás utasításra	U	–			X	
RESUME A	A hibakezelés után az A sorszámra adja a vezérlést	U	–		X		
RESUME NEXT	A hibakezelést befejezi, és visszatér a hívás utáni utasításra	U	–			X	
RETURN	Szubrutin-befejezés	U	116	X	X	X	X
RIGHT$(A$,B)	Az A$ jobb oldaláról B karaktert vesz	F	–	X	X	X	

Kulcsszó	Funkció	Típus	Leírás old.	C–64	HT–1080Z	PRIMO	Spectrum
RND (A) *	A-tól függő véletlenszámot állít elő	F	209	X	X	X	X
RUN	Programvégrehajtás	P	81	X	X	X	X
SET (X,Y)	Az X,Y koordinátájú pontot kapcsolja	F	–		X	X	
SAVE	A programot háttértárolóra másolja	P	38	X	X	X	X
SAVE-SCREEN A	Képernyőmásolás az A nevű állományra	F	–				X
SCREEN$ (X,Y)	A képernyő X,Y koordinátájú pontját olvassa	F	–				X
SGN (A) *	Az A előjelétől függően −1, 0, +1	F	–	X	X	X	X
SIN (A) *	A szinusza	F	155	X	X	X	X
SPC (A)	A számú szóközt nyomtat	F	–	X			
SQR (A)	A négyzetgyökét adja	F	–	X	X	X	X
STATUS	Az állomány helyzetét tartalmazza olvasásnál	F	–	X			
STEP	Lásd a FOR TO STEP utasításnál	K	–	X	X	X	X
STOP	A végrehajtást leállítja	U	80	X	X	X	X
STRING$(A,B)	B-t A jelből álló szöveges adattá alakítja	F	–		X	X	
STR$(A) *	A-t szöveges formájú adattá alakítja	F	–	X	X	X	X
SYS(A)	Az A tárcímtől gépi kódú programot hív	U	–	X			
SYSTEM	Gépi kódú üzemmódba kapcsol	P	–		X		
TAB (A) *	PRINT utasításban A-adik pozícióra áll be	F	164	X	X	X	X
TAN (A) *	A tangense	F	–	X	X	X	X
TEST	A tárban levő és a kazettán levő programot összehasonlítja	P	44			X	
THEN	Lásd az IF THEN és IF THEN ELSE utasításoknál	K	–	X	X	X	X
TI	A belső órát olvassa	F	–	X			
TI$	A belső órát szöveges formában olvassa	F	–	X			
TO	Lásd a FOR TO STEP utasításnál	K	–	X	X	X	X

Kulcsszó	Funkció	Típus	Leírás old.	C–64	HT–1080Z	PRIMO	Spectrum
TROFF	Kikapcsolja a nyomkövetést	P	–		X	X	
TRON	A nyomkövetést bekapcsolja	P	–		X	X	
USING	Formátumkiírás kulcsszava PRINT utasításban	K	–			X	
USR (A) *	Az A tárcímtől gépi kódú szubrutint hív	F	–	X	X		X
USR A$	Az A$ nevű felhasználói grafika kezdőcíme	F	–				X
VAL (A$) *	A$ numerikus értékét adja	F	–	X	X	X	X
VARPTR A	Az A címét adja meg	F	–		X	X	
VARPTR (A$)	Az A$ címét adja meg	F	–			X	
VERIFY	A tárban és a háttértáron levő programot hasonlítja össze	P	39	X			
WAIT	Programleállítás egy eseményig	U	–	X			

A csillaggal jelölt függvények argumentumát a Sinclair-gépeken nem kell zárójelbe tenni.

GYAKRABBAN ELŐFORDULÓ PROGRAMHIBÁK ÉS JAVÍTÁSUK

Bemutatunk néhány gyakrabban előforduló hibát és az ekkor megjelenő hibaüzenetet, valamint azt, hogy hogyan lehet a hibát kijavítani.

Leírásunk nem teljes körű, ezért azt ajánljuk az olvasónak, hogy ha olyan hibaüzenetet ír ki a gépe, amit itt nem talál meg, azt a gép kézikönyvében keresse meg, és így derítse ki a hiba okát, vagy kérjen tanácsot egy szakértőtől.

Jelenség/Hibaüzenet	Magyarázat és javítás
A ciklust nem a kívánt számban hajtja végre a program	A ciklus végfeltételét helytelenül állapítottuk meg elöl tesztelő vagy hátul tesztelő ciklus esetén. A feltételeket ki kell javítani
DIVISION BY ZERO (C64) 10 (■ HT és ▲ PRIMO)	Nullával való osztás. Nem vettük figyelembe, hogy a nevező 0 is lehet, vagy a nevező helytelen számítás eredményeként 0
NEXT WITHOUT FOR (C64) NF (■ HT és ▲ PRIMO)	Nincs FOR utasítás a FOR,NEXT utasításpárral készített ciklusban
1 NEXT without FOR (● Sinclair)	Kifelejtettük a FOR utasítás leírását a programból. A hiányzó utasítást be kell írni
OUT OF DATA (C64) OD (■ HT és ▲ PRIMO) E OUT of DATA (● Sinclair)	READ utasítással adatot akarunk kiolvasni DATA utasításból. Kevés adatot írtunk be a DATA utasításba, vagy felesleges READ utasítást írtunk be. A hibát a feladat helyes megoldása szempontjából kell értékelni és kijavítani.

Jelenség/Hibaüzenet	Magyarázat és javítás
REDO FROM START (C64) TM (■ HT és ▲ PRIMO)	A gép numerikus értéket vár egy INPUT utasításban, de szövegeset írtunk be. A program nem fogadja el az adatot, újra be kell írni helyesen
RETURN WITHOUT GOSUB (C64) RG (■ HT és ▲ PRIMO) 7 RETURN without GO SUB (● Sinclair)	A program egy RETURN sort talált anélkül, hogy előtte GOSUB utasítást hajtott volna végre. A főág végén nem kerültük ki a szubrutinokat, vagy a főágba szubrutint kevertünk bele. A szubrutinokat a felesleges végrehajtás elkerülése végett ki kell kerülni
?SYNTAX ERROR (C64) SN (■ HT és ▲ PRIMO)	Formai hibát követtünk el. Egy utasítás kulcsszavát helytelenül írtuk be, a kulcsszóban vagy az utasításssorszámban szóközt írtunk, a nyitó- és zárójelek száma eltér, valamit kifelejtettünk az utasításból, és így tovább. A hibás sort meg kell vizsgálni és ki kell javítani. Ilyen hiba a Sinclair-gépeken nem fordulhat elő, mert a gép nem fogadja el a formailag hibás utasítást
TYPE MISMATCH (C64)	Numerikus változó helyett szövegeset adtunk meg vagy fordítva, vagy kifelejtettük a $ jelet a szöveges változó végéről, vagy nem jól vettük figyelmbe a változók típusát. Újra át kell gondolni a változók típusát, a gépelési hibákat ki kell javítani stb.
UNDEF'D STATEMENT (C64)	Egy GOTO vagy GOSUB, vagy RUN *sorszám* utasításban olyan sorszám szerepel, amilyen nincs a programban. Vagy a hiányzó utasítást kell pótolni, vagy a hibás sorszámot ki kell javítani

Jelenség/Hibaüzenet	Magyarázat és javítás
BAD SUBSCRIPT (C64) BS (■ HT és ▲ PRIMO) 3 Subscript wrong (● Sinclair)	Helytelen indexet használtunk a programban. A tömböt nem definiáltuk a használat előtt, vagy a méretét nem elég nagyra vettük. A hiányzó DIM utasítást pótolni kell, vagy ki kell javítani

PROGRAMSOROK JAVÍTÁSA

1. A COMMODORE-GÉP HIBAJAVÍTÁSI LEHETŐSÉGE

A tárban levő program sorait a képernyőn lehet javítani. Ezért először a hibás sort a LIST paranccsal ki kell íratni a képernyőre. Ezután a helyőrt a hibás sor elejére állítjuk a helyőrmozgató gombok segítségével:

felfelé mozgatás	SHIFT	és	CRSR ⇑
lefelé mozgatás	CRSR ⇓		
balra mozgatás	SHIFT	és	CRSR ⇐
jobbra mozgatás	CRSR ⇒		

Ezután tudunk hibát javítani.

KARAKTEREK TÖRLÉSE

A helyőrt a törölni kívánt karaktersorozatból egy helyértékkel jobbra állítjuk, és annyiszor nyomjuk le a törlőgombot (INST/DEL), ahány jelből áll a törölni kívánt szövegrész. Törlés közben a helyőrtől jobbra eső szövegrész a helyőrrel balra mozog. Tegyük fel, hogy az alábbi utasításból ki akarjuk törölni a GOTO-t:

```
100 IF A=B THEN GO TO 500
```

A helyőrt az O és az 5 közé állítjuk, majd annyiszor lenyomjuk az INST/DEL gombot, hogy a GO TO eltűnjön. Ha a sor javítását befejeztük, nyomjuk le a RETURN gombot, mert csak így marad meg a módosítás.

KARAKTEREK BESZÚRÁSA

A helyőrt arra a helyre állítjuk a programon belül, amely után valamilyen szövegrészt be kell szúrni. Ezután a SHIFT és az INST/DEL gomb együttes lenyomásával a helyőrtől jobbra üres helyek keletkeznek, ahová a hiányzó szövegrészt begépelhetjük. Az üres helyek kitöltése után a javítást a RETURN gomb lenyomásával lehet befejezni.

Ha egy PRINT utasításban páratlan sorszámú idézőjelet írunk, akkor a helyőrmozgató gombok hatástalanná válnak. Ha ilyenkor egy helyőrmozgató gombot nyomunk le, akkor a gombnak megfelelő fordított karakter jelenik meg a képernyőn. Ez egyébként helyőrmozgatást fog végrehajtani a program futásakor. Ezért idézőjelbe tett szövegrészeken belül csak az INST/DEL gombbal lehet balra haladva törölni.

SZÖVEGES ÁTÍRÁS

A programsor szövegét más szöveg begépelésével módosíthatjuk. A módosítás után ne felejtsük el a RETURN gombot lenyomni, mert csak ezzel válik véglegessé a javítás. Ha több programsor van a képernyőn, és a javítás után a helyőr a következő sor elejére áll, de ezt nem kell javítani, akkor a RETURN gomb lenyomásával haladhatunk lefelé.

2. ■ A HT GÉP HIBAJAVÍTÁSI LEHETŐSÉGE

A tárban levő program sorait az EDIT szövegszerkesztővel lehet módosítani. Az EDIT nagyon sok hibajavítási lehetőséget tartalmaz. Ezek közül a fontosabbakat mutatjuk be.

A javítani kívánt sort az EDIT *sorszám* paranccsal hívhatjuk be, például:

EDIT 200

Ezután el lehet kezdeni a javítást. Előtte azonban érdemes megjeleníteni a keresett sort az

L

(listázás) paranccsal.

KARAKTEREK TÖRLÉSE

A helyőrt a szóközbillentyű lenyomásával (jobbra mozgatás) és a ← gombbal (balra mozgatás) tudjuk a soron belül (pontosabban a sor alatt) mozgatni.

A törlést az *n*D paranccsal lehet végrehajtani. Hatására a helyőrtől jobbra álló karakter törlődik. A kiírásban a törölt szövegrész nem tűnik el, hanem két felkiáltójel között jelenik meg. A NEW LINE lenyomása után a törlés véglegesen megtörténik. Például a következő sorból töröljük ki a GOTO-t:

200 IF A>B THEN GOTO 500

A helyőrt állítsuk a G betűre, és adjuk ki az

5D

parancsot, melynek hatására az alábbi kiírás jelenik meg:

200 IF A>B THEN ! GOTO ! 500

A 2 felkiáltójel határolja a törölni kívánt szöveget. Ha meggondoltuk magunkat, vagy rosszul írtuk be a javítást, akkor

A

parancsot kell kiadni, amely visszaállítja a programsort az EDIT-be való belépéskor meglevő állapotába, és a módosítást elölről lehet kezdeni. A javítást a NEW LINE gombbal lehet véglegesíteni.

KARAKTER BESZÚRÁSA

A helyőrt arra a pozícióra kell állítani, ahol a beszúrást el akarjuk kezdeni. Ekkor

I

parancsot adunk ki, és beírjuk a beszúrandó szöveget. A begépelés végén a SHIFT és a ↑ billentyű együttes lenyomásával befejezzük a beszúrást. Az L paranccsal kiírható a módosított utasítássor.

KARAKTER CSERÉJE

A helyőrt állítsuk közvetlenül a kicserélni kívánt szövegrész elé. Ekkor itt egy

*n*C

parancsot adunk ki, amelyben *n* a kicserélendő karakterek számát jelöli. Ezután beírjuk az új szöveget. Például az alábbi sorban az AB változónevet B2-re akarjuk javítani:

200 PRINT AB

A helyőrt az A-ra állítjuk, és kiadjuk a cserére vonatkozó parancsot:

2C

majd beírjuk az új szöveget:

B2

Ezzel a módosítás megtörtént. A módosított sort az L paranccsal kilistázzuk:

200 PRINT B2

Az EDIT szerkesztési műveletet a NEW LINE gomb lenyomásával fejezhetjük be.

Az EDIT nem teszi lehetővé a sorszám módosítását!

3. ▲ A PRIMO GÉP HIBAJAVÍTÁSI LEHETŐSÉGE

A tárban levő programot az

EDIT *sorszám*

paranccsal lehet módosítani. A *sorszám* a programsor sorszáma. A módosítandó sort a

→

gomb ismételt lenyomásával lehet kiíratni. A sorból a

←

gombbal lehet **törölni.** Ha a törölt szövegrész helyébe újat akarunk beírni, akkor azt ilyenkor be lehet gépelni. A

RETURN

gomb lenyomásával a helyőrtől jobbra levő szövegrész **törlődik,** és a módosítás befejeződik.

Ha több részt kell módosítani, akkor az egyes részek módosítása után a SHIFT és ← gomb együttes lenyomásával lehet a sor elejére állítani a helyőrt és folytatni a módosítást.

Ha nem akarunk javítani, vagy a javítást nem akarjuk átvezetni, akkor a ↓ gombot nyomjuk le, és a sor visszaáll az EDIT-be lépés előtti formájára.

A módosítást a SHIFT és a → gomb együttes lenyomásával zárhatjuk le.

Az EDIT-tel a sorszám is módosítható. Ilyenkor mind az eredeti, mind a módosított sor megmarad.

4. ● A SINCLAIR-GÉPEK HIBAJAVÍTÁSI LEHETŐSÉGE

A Sinclair-gépeken az utasítássorok beírásakor is van lehetőség olyan hibajavításra, amely nem a törlésen és újraíráson alapul. A ← gomb se-

gítségével a begépelt szövegben visszaléptethetjük a helyőrt a javítás helyére. Ekkor a helyőrtől jobbra levő szövegrész helyett **újat írhatunk** be, vagy a helyőrtől balra levő szövegrészt a DELETE gomb ismételt lenyomásával törölhetjük.

A tárban levő **programsorokat** az EDIT paranccsal lehet javítani. Előtte azonban a javítandó sort ki kell listázni. Ha több sor közül kell a javítandót kiválasztani, akkor a ↑ és ↓ gombbal tudjuk mozgatni a > sormutatójelet, és ezzel választhatjuk ki azt a sort, amelyet javítani akarunk.

A kívánt sorra ráállunk, és lenyomjuk az EDIT gombot. Ekkor a sor megjelenik a képernyő alsó részén. A javítást ugyanúgy végezhetjük el, mint egy most begépelt sorban. A javítás befejeztével lenyomjuk az ENTER gombot, ekkor a javított sor kerül a képernyő felső részén levő régi sor helyébe.

ADATÁLLOMÁNY ÉS PROGRAMKÓDOT TARTALMAZÓ ÁLLOMÁNYMŰVELETEK A COMMODORE–64 GÉPEN

1. A LEMEZ HASZNÁLATA

A lemezegység hálózati csatlakozóját bedugjuk az aljzatba. A számítógéphez csatlakozó kábel egyik végét a számítógép hátán levő egyik dugóba, a másik végét pedig a lemezegység „SERIAL BUS" feliratú dugójába dugjuk. Ezután bekapcsoljuk a tápfeszültséget. A számítógépet mindig a lemezegység **után** kapcsoljuk be!

A lemezt címkéjével felfelé fordítjuk úgy, hogy az írásvédelmi bevágás a bal oldalon legyen (nem kell leragasztani!), és behelyezzük a lemezegységbe. A becsúsztatás végén már csak egy ujjal tudjuk nyomni, és ütközésig nyomjuk be finoman. Ekkor az ajtót lehúzzuk.

A lemez kivevésénél az ajtó fogóját kissé befelé nyomjuk, ekkor az magától felugrik, a lemez pedig egy kicsit kiugrik, hogy könnyen meg tudjuk fogni kézzel.

A lemezt mindig a bekapcsolt berendezésbe tegyük be! Ha a zöld lámpa ég, ne vegyük ki a lemezt a berendezésből, mert a tartalma megsérülhet! Kikapcsolás előtt is mindig vegyük ki a lemezt az egységből!

2. AZ ÁLLOMÁNYMŰVELETEK PARANCSFORMÁJA

Adatállományokkal és programkódot tartalmazó állományokkal kapcsolatos műveleteket parancscsatornára írás formájában adunk ki. Ezért minden művelet elején meg kell nyitni a parancscsatornát az alábbi paranccsal:

OPEN 15, 8, 15

ahol

az első 15	— az állomány logikai hivatkozási száma,
8	— a lemezegység száma (lehet 9 is),
a második 15	— a csatorna száma, amelyen keresztül a lemez és a számítógép összekapcsolása létrejön. A 15 a parancscsatorna száma.

A parancsot PRINT utasítással adjuk ki. A parancs után a számítógép és a lemezegység közötti kapcsolatot meg kell szüntetni a

CLOSE15

paranccsal.

3. AZ ÁLLOMÁNYMŰVELETEK PARANCSAI

3.1 A LEMEZ FORMÁLÁSA

Mielőtt egy lemezt használatba veszünk vásárlás után, vagy a lemez tartalmára nincs szükségünk, és fel akarjuk „újítani" magunknak, akkor kell ezt a műveletet elvégezni:

PRINT#15, "NEW0: *név, azonosító*"

ahol

NEW0:	—	a formálás kulcsszava (N0:-val rövidíthető),
név	—	a lemez neve (max. 16 karakter),
azonosító	—	tetszőleges azonosító kód (2 karakter).

Példa:

```
OPEN 15,8,15:PRINT#15,"NEW0:TVBASIC,PL":CLOSE15
```

A lemezt formálja a TVBASIC névvel és PL azonosítóval.

3.2 ÁLLOMÁNYOK MÁSOLÁSA

Ha egy meglevőhöz hasonló programot kívánunk készíteni, akkor a kész program módosításával célunkat elérhetjük. Ehhez viszont előbb egy másolatot kell készíteni a programról, és a másolatot kell módosítanunk. A másolási parancs formája:

PRINT#15,"COPY0:*új név* = 0:*régi név*"

ahol

COPY 0:	—	a másolás parancsszava (C0:-val rövidíthető),
új név	—	a másolat neve (max. 16 karakter),
régi név	—	a meglevő program neve.

Példa:

```
OPEN15,8,15:PRINT#15,"COPY0:AMOR1=0:AMOR":CLOSE15
```

Az AMOR nevű programot AMOR1 névvel lemásolja.

Megjegyzés: programokat úgy is másolhatunk, hogy a másolni kívánt programot a LOAD paranccsal betöltjük a tárba, és — módosítás után vagy anélkül — új névvel kimásojuk a SAVE parancs segítségével.

3.3 ÁLLOMÁNYOK ÖSSZEKAPCSOLÁSA

Meglevő programmodulokból úgy tudunk létrehozni egy újabb programot, hogy a modulokat a feladat megoldásának megfelelő sorrendben egymás után összekapcsoljuk. Az összekapcsolást a COPY0: paranccsal lehet végrehajtani:

PRINT#15,"C0: *új név* = 0:*régi név 1*, 0:*régi név 2*, . . ."

A régi nevű állományokat az összekapcsolás sorrendjében kell felsorolni (a *régi név 1* lesz elöl, utána a *régi név 2*, . . .). Egy paranccsal legfeljebb négy állományt lehet összekapcsolni.

3.4 ÁLLOMÁNYOK NEVÉNEK VÁLTOZTATÁSA

Egy állomány nevét a következő paranccsal lehet módosítani:

PRINT#15, "RENAME0:*új név* = *régi név*"

ahol

RENAME0:	—	a névváltoztatás parancsszava (R0-val rövidíthető),
új név	—	az állomány új neve,
régi név	—	az állomány jelenlegi neve.

Példa:

```
OPEN15,8,15:PRINT#15,"R0 :PELDA = MINTA":CLOSE15
```

A MINTA nevű állomány nevét PELDA-ra módosítja.

3.5 ÁLLOMÁNY TÖRLÉSE

Ha valamelyik állományra (adat vagy program) már nincs szükségünk, akkor a tárolóhely felszabadítása végett a felesleges állományt kell törölni az alábbi paranccsal:

PRINT#15, "SCRATCH0:*név*"

ahol

SCRATCH0:	—	a törlés parancsszava (S0:-val rövidíthető),
név	—	a törölni kívánt állomány neve.

Példa:

```
OPEN15,8,15:PRINT#15,"S0:GEPKI":CLOSE15
```

A parancs hatására a GEPKI nevű állomány törlődik a lemezről.

3.6 A LEMEZ „RENDBE RAKÁSA"

Ha egy lemezen sokszor adunk ki SCRATCH parancsot, vagy sok program másolunk át a lemezre, akkor a lemez tartalomjegyzéke megsérülhet. Ezt elkerülhetjük, ha a lemezt időnként rendbe rakjuk a következő paranccsal:

```
PRINT#15,"VALIDATE"
```

ahol

VALIDATE – a rendbe rakás parancsszava (V-vel rövidíthető).

4. PROGRAMHIBA

Ha egy programban adatállományt használunk, és a program valamilyen hiba miatt leáll, akkor a program ugyan lezárja az állományokat, de nem a lemezen, ezért az állományok megsemmisülhetnek. Ennek elkerülésére ilyenkor mindig adjuk ki az alábbi parancsot:

```
CLOSE15:OPEN15,8,15:CLOSE15
```

5. ÁLLOMÁNYMŰVELETEK A CP/M OPERÁCIÓS RENDSZERBEN

Az előbbi műveleteket a CP/M operációs rendszerben is el lehet végezni. Felhívjuk a figyelmet arra, hogy a CP/M lényegesen több szolgáltatást tud nyújtani, mint a Commodore–64 operációs rendszere. A következőkben röviden összefoglaljuk a CP/M néhány, állományokkal kapcsolatos parancsát:

– állománymásolás

```
>PIP új név.BAS = régi név.BAS
```

ahol

PIP – néhány állományművelet általános parancsszava,
új név – a másolat neve,
régi név – a jelenlegi állomány neve,
BAS – BASIC programállományt jelöl.

– állományok összekapcsolása

```
>PIP új név.BAS = régi név1.BAS, régi név2.BAS
```

ahol

új név – a létrehozott új program neve,

régi név1 és
régi név2 — az összekapcsolandó állományok neve az összekapcsolás sorrendjében.

– állomány nevének változtatása

>REN *új név*.BAS = *régi név*.BAS

ahol

REN — a névváltoztatás parancsszava.

– állomány törlése

>ERA *név*.BAS

ahol

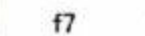
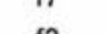
ERA — a törlés parancsszava,
név — a törölni kívánt állomány neve.

A COMMODORE-BAN HASZNÁLT ASCII KÓD KARAKTEREI

PRINTS*	CHRS**	PRINTS*	CHRS**
	0	CRSR	17
	1	RVS ON	18
	2	CLR HOME	19
	3	INST DEL	20
	4		21
WHI	5		22
	6		23
	7		24
DISABLES SHIFT C=	8		25
ENABLES SHIFT C=	9		26
	10		27
	11	RED	28
	12	CRSR	29
RETURN	13	GRN	30
SWITCH TO LOWER CASE	14	BLU	31
	15	SPACE	32
	16	!	33

*Megjelenített karakter
**Bájt értéke

PRINTS	CHRS	PRINTS	CHRS
"	34	B	66
#	35	C	67
$	36	D	68
%	37	E	69
&	38	F	70
'	39	G	71
(	40	H	72
)	41	I	73
*	42	J	74
+	43	K	75
,	44	L	76
–	45	M	77
.	46	N	78
/	47	O	79
0	48	P	80
1	49	Q	81
2	50	R	82
3	51	S	83
4	52	T	84
5	53	U	85
6	54	V	86
7	55	W	87
8	56	X	88
9	57	Y	89
:	58	Z	90
;	59	[	91
	60	£	92
=	61	]	93
	62	↑	94
?	63	←	95
@	64		96
A	65		

PRINTS	CHRS	PRINTS	CHRS
	97		126
	98		127
	99		128
	100	Orange	129
	101		130
	102		131
	103		132
	104	f1	133
	105	f3	134
	106	f5	135
	107	f7	136
	108	f2	137
	109	f4	138
	110	f6	139
	111	f8	140
	112		141
	113		142
	114		143
	115		144
	116		145
	117		146
	118		147
	119		148
	120	Brown	149
	121	Lt. Red	150
	122	Grey 1	151
	123	Grey 2	152
	124	Lt. Green	153
	125	Lt. Blue	154

PRINTS	CHRS	PRINTS	CHRS
Grey 3	155		174
	156		175
	157		176
	158		177
	159		178
	160		179
	161		180
	162		181
	163		182
	164		183
	165		184
	166		185
	167		186
	168		187
	169		188
	170		189
	171		190
	172		191
	173		

COMMODORE-64 LISTÁK

1. FELADAT

```
10 REM**************
20 REM*              *
30 REM*     EREL     *
40 REM*              *
50 REM**************
60 REM***** BEMENO ADATOK TAROLASA *****
70 LET R1=1000
80 LET R2=2000
90 LET R3=500
100 LET R4=200
110 REM***** ELLENALLAS SZAMITAS *****
120 LET K0=1/R1+1/R2+1/R3
130 LET R0=1/K0
140 LET R9=R0+R4
150 REM***** EREDMENYKIIRAS *****
160 PRINT"AZ EREDO ELLENALLAS (OHM): ";R9
170 END
```

2. FELADAT

```
10 REM****************
20 REM*                *
30 REM*    AMOR        *
40 REM*                *
50 REM****************
60 REM***** BEMENETI ADATOK *****
70 J=1984
80 PRINT:PRINT
90 INPUT"BRUTTOERTEK: ";B
100 PRINT
110 INPUT"BESZERZES EVE: ";E
120 REM***** AMORTIZACIO SZ. *****
130 K=J-E
140 IF K<5THEN 160
150 GO TO 190
160 REM* THEN AG *
170 N=(B*(5-K))/5
180 GO TO 210
190 REM* ELSE AG *
200 N=0.1
210 REM* IF VEG *
220 REM***** KIIRAS *****
230 PRINT"           NETTO ERTEK SZAMITAS"
240 PRINT"           ----- ----- --------"
250 PRINT:PRINT
260 PRINT"BRUTTO ERTEK      ";B;"EFT"
270 PRINT"A BESZERZES EVE   ";E;"EV"
280 PRINT"ELETKOR           ";K;"EV"
290 IF K<5 THEN 310
300 GO TO 340
310 REM* THEN AG *
320 PRINT"NETTO ERTEK       ";N;"EFT"
330 GO TO 360
340 REM* ELSE AG *
350 PRINT"NETTO ERTEK       ";N;"EFT (ESZMEI ERTEK)"
360 REM* IF VEG *
370 END
```

2.1 FELADAT

```
10 REM***************
20 REM*              *
30 REM*    AMOR1     *
40 REM*              *
50 REM***************
60 REM***** VEZERLES *****
70 GOSUB 200
80 GOSUB 300
90 GOSUB 500
100 INPUT"TOVABB ? (IGEN=1, NEM=0) ";T
110 IF T=1 THEN 80
120 GO TO 860
200 REM***** EVSZAM BEOLVASAS *****
210 INPUT"JELENLEGI EVSZAM: ";J
220 RETURN
300 REM***** AMORTIZACIO SZAMITAS *****
310 READ B,E
320 K=J-E
330 IF K<5 THEN 350
340 GO TO 380
350 REM* THEN AG *
360 N=(B*(5-K))/5
370 GO TO 400
380 REM* ELSE AG *
390 N=0.1
400 REM* IF VEG *
410 RETURN
500 REM***** KIIRAS *****
510 PRINT"□"
520 PRINT"          NETTO ERTEK SZAMITAS"
530 PRINT"          ----- ----- --------"
540 PRINT:PRINT
550 PRINT"BRUTTO ERTEK      ";B;"EFT"
560 PRINT"A BESZERZES EVE   ";E;"EV"
570 PRINT"ELETKOR           ";K;"EV"
580 IF K<5 THEN 600
590 GO TO 630
600 REM* THEN AG *
610 PRINT"NETTO ERTEK       ";N;"EFT"
620 GO TO 650
630 REM* ELSE AG *
640 PRINT"NETTO ERTEK       ";N;"EFT (ESZMEI ERTEK)"
650 REM* IF VEG *
660 PRINT: PRINT
670 RETURN
```

```
800 REM***** ADATOK *****
810 DATA 2500,1982,500,1984
820 DATA 3050,1984,162,1984
830 DATA 520,1980,1610,1978
840 DATA 3960,1979,865,1980
850 DATA 32,1976,785,1981
860 END
```

3. FELADAT

```
10 REM*****************
20 REM*                 *
30 REM*      JOVSZ      *
40 REM*                 *
50 REM*****************
60 REM***** ADATBEOLVASAS *****
70 INPUT"BRUTTO JOVEDELEM: ";BJ
80 PRINT
90 INPUT"TARSADALOM BIZTOSITAS: ";TB
100 PRINT
120 REM***** ADOALAP SZAMITAS *****
130 AA=0.95*BJ
140 AA=AA-TB
150 REM***** JOV. ADO SZAMITAS *****
160 REM* CASE KEZDET *
170 IF AA<=20000 THEN 250
180 IF AA<=40000 THEN 280
190 IF AA<=60000 THEN 310
200 IF AA<=100000 THEN 340
210 IF AA<=200000 THEN 370
220 IF AA<=400000 THEN 400
230 IF AA<=600000 THEN 430
240 GO TO 460
250 REM* 1. ESET *
260 JA=AA*0.02
270 GO TO 480
280 REM* 2. ESET *
290 JA=400+(AA-20000)*0.06
300 GO TO 480
310 REM* 3. ESET *
320 JA=1600+(AA-40000)*0.1
330 GO TO 480
340 REM* 4. ESET *
350 JA=3600+(AA-60000)*0.2
360 GO TO 480
370 REM* 5. ESET *
380 JA=11600+(AA-100000)*0.38
390 GO TO 480
400 REM* 6. ESET *
410 JA=49600+(AA-200000)*0.5
420 GO TO 480
430 REM* 7. ESET *
440 JA=149600+(AA-400000)*0.6
450 GO TO 480
```

```
460 REM* 8. ESET *
470 JA=269600+(AA-600000)*0.65
480 REM* CASE VEG *
500 REM***** NETTO JOV. SZAMITAS *****
510 KF=0.1*JA
520 NJ=BJ-(TB+JA+KF)
530 REM***** NETTO JOV. KIIRAS *****
540 F$=" FT"
550 PRINT"BRUTTO JOVEDELEM    ";BJ;F$
560 PRINT"JOVEDELEM ADO       ";JA;F$
570 PRINT"KOFA                ";KF;F$
580 PRINT"NETTO JOVEDELEM     ";NJ;F$
590 END
```

4. FELADAT

```
10 REM*****************
20 REM*                 *
30 REM*     KESZL       *
40 REM*                 *
50 REM*****************
60 REM***** KESZLET MASOLAS *****
70 READ K,AR
90 REM***** MENU *****
110 PRINT"□"
120 PRINT: PRINT: PRINT: PRINT
130 PRINT"          KESZLETNYILVANTARTAS"
140 PRINT: PRINT
150 PRINT"(1) BEVETELEZES"
160 PRINT"(2) KIADAS"
170 PRINT"(3) BEFEJEZES"
180 PRINT
190 INPUT"MELYIKET VALASZTJA ";D
200 IF D=1 THEN GOSUB 250
210 IF D=2 THEN GOSUB 440
220 IF D=3 THEN GOSUB 750
230 IF D=3 THEN 960
240 GO TO 110
250 REM***** BEVETELEZETT M. *****
260 PRINT"□"
270 PRINT: PRINT
280 INPUT"A BEVETELEZETT MENNYISEG: ";BM
290 K=K+BM
300 REM***** ERTEK SZAM. *****
310 MA=BM
320 GOSUB 900
330 REM***** ADATKIIRAS (B) *****
340 PRINT"□"
350 PRINT: PRINT
360 PRINT"         ANYAGBEVETELEZES"
370 PRINT: PRINT
380 PRINT"A BEVETELEZETT MENNYISEG: ";BM
390 PRINT
400 PRINT"A BEVETELEZETT ERTEK: ";ER
410 PRINT: PRINT: PRINT
420 INPUT"HA BEFEJEZTE, NYOMJON LE EGY GOMBOT !";F$
430 RETURN
```

```
440 REM***** KIADOTT M. *****
460 PRINT"␣"
470 PRINT: PRINT
480 INPUT"A KIADOTT MENNYISEG: ";KM
490 IF KM>K THEN 510
500 GO TO 550
510 REM* THEN AG *
520 PRINT: PRINT
530 PRINT"ENNYI NINCS !"
540 GO TO 480
550 REM* ELSE AG *
560 K=K-KM
570 REM* IF VEG *
580 REM****** ERTEK SZAMITAS *****
590 MA=KM
600 GOSUB 900
620 REM***** ADATKIIRAS (K) *****
630 PRINT"␣"
640 PRINT: PRINT
650 PRINT"          ANYAGKIADAS"
660 PRINT: PRINT
670 PRINT"A KIADOTT MENNYISEG: ";KM
680 PRINT
690 PRINT"A KIADOTT ERTEK: ";ER
710 PRINT: PRINT: PRINT
720 INPUT"HA BEFEJEZTE, NYOMJON LE EGY GOMBOT !";Fs
730 RETURN
750 REM***** KESZLET ATIRAS *****
760 PRINT"␣"
770 PRINT"NE FELEJTSE ATIRNI A KESZLETET !"
780 PRINT
790 PRINT"940 DATA";K
795 PRINT: PRINT
800 REM***** BEFEJEZES *****
810 PRINT"             VEGE"
820 RETURN
900 REM***** ERTEKSZAM. SZUBRUTIN *****
910 ER=MA*AR
920 RETURN
930 REM***** ADATOK *****
940 DATA 100
950 DATA 2.3
960 END
```

5. FELADAT

```
10 REM***************
20 REM*              *
30 REM*    KAMAT     *
40 REM*              *
50 REM***************
60 REM***** ADATBEOLV. *****
70 INPUT"A KOLCSON OSSZEGE: ";F
80 IF F<=0 THEN 100
90 GO TO 130
100 REM* THEN AG *
110 PRINT: PRINT"CSAK POZITIV LEHET !"
120 GO TO 70
130 REM* IF VEG *
140 INPUT"KAMATLAB (%): ";K
150 IF K<=0 THEN 170
160 GO TO 200
170 REM* THEN AG *
180 PRINT: PRINT"CSAK POZITIV LEHET !"
190 GO TO 140
200 REM* IF VEG *
210 K=K/100
230 REM***** RESZLET SZAMITAS *****
240 K1=K/12
250 R=F*K1/(1-(1/(1+K1)↑12))
260 REM***** HK, CK, FJ, SZAM. *****
270 PRINT: PRINT
280 PRINT"RESZLET","KAMAT","VISSZA-","KOLCSON"
290 PRINT"OSSZEG","OSSZEG","FIZ. O.","OSSZEG"
300 PRINT
310 REM* CIKLUS KEZDET *
350 FOR H=1 TO 12
360 HK=F*K1
370 CK=R-HK
380 F=F-CK
390 REM***** OSSZEGZES *****
400 SR=SR+R
410 SK=SK+HK
420 SC=SC+CK
440 REM***** KIIRAS *****
450 R=INT(R+0.5)
460 HK=INT(HK+0.5)
470 CK=INT(CK+0.5)
480 F=INT(F+0.5)
```

```
490 PRINT R,HK,CK,F
500 NEXT H
510 REM* CIKLUS VEG *
530 REM***** OSSZEG KIIRAS *****
540 PRINT
550 SR=INT(SR+0.5)
560 SK=INT(SK+0.5)
570 SC=INT(SC+0.5)
580 PRINT SR,SK,SC
590 END
```

6. FELADAT

```
10 REM***************
20 REM*              *
30 REM*    MERES     *
40 REM*              *
50 REM***************
60 REM***** ADATBEVITEL *****
70 DIM T(100)
80 PRINT"␣"
90 PRINT"ADATBEVITEL"
100 PRINT
110 INPUT"A MERESEK SZAMA: ";M
120 PRINT: PRINT: PRINT
130 REM* CIKLUS KEZDET *
140 FOR I=1 TO M
150 PRINT"A(Z)";I;". ADAT KOVETKEZIK :";
160 INPUT T(I)
170 PRINT: PRINT
180 NEXT I
190 REM* CIKLUS VEG *
200 REM***** ATLAGSZAMITAS *****
210 S=0
220 REM* CIKLUS KEZDET *
230 FOR I=1 TO M
240 S=S+T(I)
250 NEXT I
260 REM* CIKLUS VEG *
270 A=S/M
300 REM***** ATLAGKIIRAS *****
310 PRINT"␣"
320 PRINT,"A MERES EREDMENYEI"
330 PRINT,"- ----- ----------"
340 PRINT: PRINT
350 PRINT"A MERESEK SZAMA: ";M
360 PRINT
370 PRINT"AZ ATLAGOS HOMERSEKLET: ";A;" CELSIUS FOK"
400 REM***** SZORASSZAMITAS *****
410 REM* CIKLUS KEZDET *
420 FOR I=1 TO M
430 K=K+(A-T(I))↑2
440 NEXT I
450 REM* CILUS VEG *
460 D2=SQR(K/M)
```

```
500 REM***** SZORAS KIIRAS *****
510 PRINT
520 PRINT"AZ ERTEKEK SZORASA: ";D2
530 END
```

7. FELADAT

```
10 REM****************
20 REM*                *
30 REM*     MSZK       *
40 REM*                *
50 REM****************
60 REM***** FELADAT VEZ. *****
70 DIM K(100,3)
80 GOSUB 200
90 GOSUB 300
100 GO TO 1050
200 REM***** DATUM BEOLV. *****
210 INPUT"EV: ";EV
220 PRINT
230 INPUT"HO: ";HO
240 PRINT
250 INPUT"NAP: ";NAP
260 RETURN
300 REM***** MENU *****
310 PRINT"␣"
320 PRINT: PRINT
330 PRINT"           KIFIZETESI MUVELETEK"
340 PRINT: PRINT
350 PRINT"  (1) KIFIZETESI ADATOK BEIRASA"
360 PRINT
370 PRINT"  (2) KIMUTATAS KESZITES"
380 PRINT: PRINT
390 INPUT"MELYIKET VALASZTJA";D
400 IF D=1 THEN GOSUB 500
410 IF D=2 THEN GOSUB 600
420 IF D=2 THEN 440
430 GO TO 300
440 RETURN
500 REM***** KIFIZ. ADATOK *****
510 PRINT"␣"
520 N=N+1
530 INPUT"MUNKASZAM: ";K(N,1)
540 PRINT
550 INPUT"OSSZEG: ";K(N,2)
560 PRINT
570 INPUT"BIZONYLAT SORSZAM: ";K(N,3)
590 RETURN
```

```
600 REM***** KIMUTATAS VEZ. *****
610 GOSUB 700
620 GOSUB 900
630 RETURN
700 REM***** RENDEZES *****
710 F=0
720 FOR I=1 TO N-1
730 IF K(I,1)>K(I+1,1) THEN 750
740 GO TO 880
750 REM* THEN AG *
760 SM=K(I,1)
770 K(I,1)=K(I+1,1)
780 K(I+1,1)=SM
790 SF=K(I,2)
800 K(I,2)=K(I+1,2)
810 K(I+1,2)=SF
820 SB=K(I,3)
830 K(I,3)=K(I+1,3)
840 K(I+1,3)=SB
850 REM* IF VEG *
860 F=1
870 NEXT I
880 IF F=1 THEN 710
890 RETURN
900 REM***** KIIRAS *****
910 OPEN2,4
920 PRINT#2,"            NAPI KIFIZETESEK"
930 PRINT#2,"            ---- -----------"
940 PRINT#2: PRINT#2
950 PRINT#2,"KELT,";EV;".EV";HO;". HO";NA;". NAP"
960 PRINT#2
970 PRINT#2,"MUNKA-";TAB(10);"KIFIZETETT";TAB(10);"BI
980 PRINT#2,"SZAM";TAB(12);"OSSZEG";TAB(13);"SORSZ."
990 PRINT#2
1000 FOR I=1 TO N
1010 PRINT#2,K(I,1);TAB(12);K(I,2);TAB(14);K(I,3)
1020 NEXT I
1030 CLOSE2
1040 RETURN
1050 END
```

8. FELADAT

```
10 REM*****************
20 REM*                 *
30 REM*      CSATA      *
40 REM*                 *
50 REM*****************
60 REM***** CSATA VEZ. *****
70 GOSUB 130
80 GOSUB 180
90 GOSUB 500
100 GOSUB 600
110 GO TO 1700
130 REM***** KEZDO MUV. *****
140 GOSUB 700
150 GOSUB 800
160 RETURN
180 REM***** KEZDO LOVES *****
190 PRINT:PRINT
200 INPUT"AZ 1. LOVES KOORDINATAI (X,Y):";X2,Y2
210 IF X2<1 THEN 230
220 GO TO 260
230 REM* THEN AG *
240 PRINT"AZ X<1 NEM LEHET!"
250 GO TO 200
260 REM* IF VEG *
270 IF X2>20 THEN 290
280 GO TO 320
290 REM* THEN AG *
300 PRINT"AZ X>20 NEM LEHET!"
310 GO TO 200
320 REM* IF VEG *
330 IF Y2<1 THEN 350
340 GO TO 380
350 REM* THEN AG *
360 PRINT"AZ Y<1 NEM LEHET!"
370 GO TO 200
380 REM* IF VEG *
390 IF Y2>20 THEN 410
400 GO TO 440
410 REM* IF VEG *
420 PRINT"AZ Y>20 NEM LEHET!"
430 GO TO 200
```

```
440 REM* IF VEG *
450 X2=INT(X2)
460 Y2=INT(Y2)
470 L=1
480 RETURN
500 REM***** TOVABBI LOVES *****
510 GOSUB 850
520 GOSUB 900
530 GOSUB 1100
540 IF (L>15) OR T=1 THEN 560
550 GO TO 510
560 RETURN
600 REM***** BEFEJEZES *****
610 IF T=1 THEN 630
620 GO TO 660
630 REM* THEN AG *
640 GOSUB 1200
650 GO TO 680
660 REM* ELSE AG *
670 GOSUB 1300
680 REM* IF VEG *
690 RETURN
700 REM***** JATEKSZABALYOK *****
710 PRINT "■"
720 PRINT"EGY 20X20-AS CSATAMEZOBEN KELL EGY"
730 PRINT"ISMERETLEN HELYEN LEVO HAJOT ELTALALNI"
740 PRINT"A CELPONT KOORDINATAINAK MEGADASAVAL."
750 PRINT"A PROGRAM A BECSAPODAS ES A HAJO"
760 PRINT" TAVOLSAGAT KOZLI."
770 PRINT"A HAJOKRA LEGFELJEBB 15 LOVEDEKET LOHET K
780 RETURN
800 REM***** HAJO HELYZETE *****
810 X1=INT(RND(0)*20)+1
820 Y1=INT(RND(0)*20)+1
830 RETURN
850 REM***** LOVESSZAM VIZSGALAT *****
860 IF L>1 THEN GOSUB 1400
870 RETURN
900 REM***** CSILLAG KIIRAS *****
910 PRINT"■"
920 IF 20-Y2=0 THEN 960
930 FOR I=1 TO 20-Y2
940 PRINT"■";
950 NEXT I
960 IF X2=1 THEN 1000
```

```
370 FOR J=1 TO X2-1
380 PRINT"■";
390 NEXT J
1000 PRINT "*"
1010 RETURN
1100 REM***** ERTEKELES *****
1110 R=INT(SQR((X2-X1)↑2+(Y2-Y1)↑2))
1120 IF R=0 THEN T=1
1130 FOR I=1 TO Y2
1140 PRINT
1150 NEXT I
1160 L=L+1
1170 PRINT"A BECSAPODAS TAVOLSAGA: ";R
1180 PRINT"A";L;". LOVES KOVETKEZIK."
1190 RETURN
1200 REM***** TALALAT *****
1210 PRINT"ON ELSULLYESZTETTE AZ ELLENSEG HAJOJAT!"
1220 RETURN
1300 REM***** NEM TALALT *****
1310 PRINT"EZ NEM SIKERULT!"
1320 PRINT"A HAJO A ";X1;",";Y1;"KOORDINATAKON ALLT!"
1330 RETURN
1400 REM***** LOVES *****
1410 GET A$
1415 IF A$="" THEN 1410
1420 IF A$="P" THEN 1470
1430 IF A$="L" THEN 1510
1440 IF A$=":" THEN 1550
1450 IF A$="." THEN 1590
1460 GOTO 1410
1470 REM* P GOMB *
1480 IF Y2=20 THEN Y2=1: GOTO 1620
1490 Y2=Y2+1
1500 GOTO 1620
1510 REM* L GOMB*
1520 IF X2=1 THEN X2=20:GOTO 1620
1530 X2=X2-1
1540 GO TO 1620
1550 REM* : GOMB *
1560 IF X2=20 THEN X2=1: GO TO 1620
1570 X2=X2+1
1580 GO TO 1620
1590 REM* . GOMB *
1600 IF Y2=1 THEN Y2=20:GO TO 1620
1610 Y2=Y2-1
1620 RETURN
1700 PRINT"VEGE"
1710 END
```

9/1. FELADAT

```
10 REM****************
20 REM*                *
30 REM*     LETRE      *
40 REM*                *
50 REM****************
60 REM***** MEGNYITAS *****
70 OPEN2,8,2,"0:ANYAG,S,W"
80 REM***** ADATBEOLVASAS *****
90 PRINT"□"
100 INPUT"AZONOSITO: ";A
110 PRINT"        123456789012345678901234567890"
120 INPUT"NEV: ";N$
130 INPUT"MENNYISEGI EGYSEG (KG, DB, M): ";M$
140 INPUT"MENNYISEG: ";M
150 INPUT "ELSZAMOLO AR: ";AR
200 REM***** ALLOMANYRA IRAS *****
210 PRINT#2,A;",";N$;",";M$;",";M;",";AR
220 PRINT
230 INPUT"FOLYTATJA ? (I/N):";F$
240 IF F$="N" THEN 260
250 GO TO 80
260 REM***** LEZARAS *****
270 CLOSE2
280 END
```

9/2. FELADAT

```
10 REM*****************
20 REM*                 *
30 REM*     MENU        *
40 REM*                 *
50 REM*****************
100 REM***** MENU KIIRAS *****
110 PRINT"■"
120 PRINT: PRINT: PRINT
130 PRINT"A VALASZTHATO MUVELETEK:"
140 PRINT"- ---------- ----------"
150 PRINT
160 PRINT,"(1) KESZLETJELENTES"
170 PRINT
180 PRINT,"(2) BEVETELEZES"
190 PRINT
200 PRINT,"(3) KIADAS"
210 PRINT
220 PRINT,"(4) BEFEJEZES"
230 PRINT: PRINT
240 INPUT"MELYIKET VALASZTJA ?";FU
250 REM***** PROGRAM HIVAS *****
260 REM* CASE KEZDET *
270 IF FU=1 THEN CLR:LOAD"KESZL",8
280 IF FU=2 THEN CLR:LOAD"BEVET",8
290 IF FU=3 THEN CLR:LOAD"KIVET",8
300 REM* CASE VEG *
310 END
```

9/3. FELADAT

```
10 REM***************
20 REM*              *
30 REM*   KIVET      *
40 REM*              *
50 REM***************
60 REM***** CIM KIIRAS *****
70 PRINT"■"
80 PRINT,"ANYAG KIVETELEZES"
90 PRINT: PRINT: PRINT
100 REM***** AZON. BEOLV. ES ELL. *****
110 INPUT"AZ ANYAG AZONOSITOJA: ";AA
120 T=0
130 OPEN2,8,2,"0:ANYAG,S,R"
140 INPUT#2,RA,N$,M$,M1
150 VJ=ST
160 IF AA=RA THEN 180
170 GO TO 210
180 REM* THEN AG *
190 T=1
200 GO TO 230
210 REM* IF VEG *
220 IF VJ=0 THEN 140
230 CLOSE2
240 IF T=1 THEN 320
245 PRINT
250 PRINT"HIBAS AZ AZONOSITO!"
260 PRINT
270 INPUT"JAVITHATO ? (I/N) ";JA$
290 IF JA$="I" THEN 110
300 GO TO 800
320 REM***** MENNY. BEOLV. *****
330 INPUT"A KIVETT MENNYISEG: ";KM
340 IF M1>KM THEN 360
350 GO TO 390
360 REM* THEN AG *
370 PRINT: PRINT"A MENNYISEG KIADHATO."
380 GO TO 420
390 REM* ELSE AG *
400 PRINT"CSAK";M1;"ADHATO KI !!"
410 KM=M1
420 REM* IF VEG *
430 MM=M1-KM
```

```
450 REM***** MODOSITAS *****
460 OPEN3,8,3,"0:ANYAG,S,R"
470 OPEN4,8,4,"0:ATM,S,W"
480 INPUT#3,A2,N2$,M2$,M2,AR
490 VJ=ST
500 IF A2=AA THEN M2=MM
510 PRINT#4,A2;",";N2$;",";M2$;",";M2;",";AR
520 IF VJ=0 THEN 480
530 CLOSE3
540 CLOSE4
550 OPEN15,8,15
560 PRINT#15,"S0:ANYAG"
570 PRINT#15,"R0:ANYAG=ATM"
580 CLOSE15
590 REM***** KIVET BIZ. *****
600 INPUT"A MAI DATUM (EV, HO, NAP): ";EV,HO,NA
610 OPEN5,4
630 PRINT#5,,"ANYAGKIVETELI JEGY"
640 PRINT#5,,"------------- ----"
650 FOR I=1 TO 4
660 PRINT#5
670 NEXT I
680 PRINT#5,"KELT,";EV;". EV";HO;". HO";NA;". NAP"
690 PRINT#5
700 PRINT#5,"AZ ANYAG AZONOSITOJA: ";RA
710 PRINT#5,"AZ ANYAG MEGNEVEZESE: ";N$
720 PRINT#5,"MENNYISEGI EGYSEG: ";M$
730 PRINT#5,"KIVETT MENNYISEG: ";KM
740 PRINT#5: PRINT#5: PRINT#5
750 PRINT#5,,,,"--------------------"
760 PRINT#5,,,,"      ALAIRAS"
770 CLOSE5
800 REM***** MENU HIVAS *****
805 CLR
810 LOAD"MENU",8
820 END
```

10. FELADAT

```
10 REM*****************
20 REM*                 *
30 REM*     GPKBE       *
40 REM*                 *
50 REM*****************
60 REM***** TERULET KIJELOLES*****
70 K0=200*64
80 POKE 2040,200
90 K1=201*64
100 POKE 2041,201
110 K2=202*64
120 POKE 2042,202
130 K3=203*64
140 POKE 2043,203
150 K4=204*64
160 POKE 2044,204
170 K5=205*64
180 POKE 2045,205
200 REM***** DEFINIALAS *****
210 D=K0
220 GOSUB 1000
230 D=K1
240 U=15
250 GOSUB 1200
260 D=K2
270 GOSUB 1000
280 D=K3
290 GOSUB 1000
300 D=K4
310 GOSUB 1000
320 D=K5
330 U=24
340 GOSUB1200
350 REM***** SZINBAJTOK ALLITASA *****
360 SZ=53287
370 REM* CIKLUS KEZDET*
380 FOR I=0 TO 5
390 POKE SZ+I,2
400 NEXTI
```

```
410 REM* CIKLUS VEG*
430 REM***** KETSZEREZES *****
440 POKE 53271,63
460 REM***** FUGGOLEGES ELHELYEZES *****
470 POKE 53249,148
480 POKE 53251,106
490 POKE 53253,148
500 POKE 53255,106
510 POKE 53257,148
520 POKE 53259,106
550 REM***** VIZSZINTES ELHELYEZES *****
560 POKE 53248,120
570 POKE 53250,120
580 POKE 53252,200
590 POKE 53254,200
600 POKE 53256,24
610 POKE 53258,24
620 POKE 53264,48
650 REM***** BEKAPCSOLAS ******
660 PRINT"■"
670 POKE 53269,63
680 REM***** SZOVEGEK *****
690 PRINT"  MAGYARORSZAG GEPKOCSI BEHOZATALA"
700 PRINT"        1977, 1980, 1983EVEKBEN"
710 PRINT"                (EZERDB)"
720 PRINT:PRINT
730 PRINT,94.3,110.5,69.1
740 REM*URES SOR CIKL. KEZDET*
750 FOR I=1 TO 11
760 PRINT
770 NEXT I
780 REM* CIKLUS VEG *
790 PRINT,1977,1980,1983
800 GOTO 1350
1000 REM*** TELJES KITOLTES ***
1010 REM*CIKLUS KEZDET *
1020 FOR I=0 TO 62 STEP 3
1030 POKE D+I,255
1040 POKE D+I+1,0
1050 POKE D+I+2,0
1060 NEXT I
1070 REM* CIKLUS VEG *
1080 RETURN
```

```
1200 REM*** RESZLEGESKITOLTES ***
1210 REM*NULLAZOCIKL. KEZDET*
1220 FOR I=0 TO U-1
1230 POKE D+I,0
1240 NEXT I
1250 REM* CIKLUS VEG*
1260 REM* KITOLTESICIKL. KEZDET*
1270 FOR I=U TO 62 STEP 3
1280 POKE D+I,255
1290 POKE D+I+1,0
1300 POKE D+I+2,0
1310 NEXT I
1320 REM* CIKLUS VEG *
1330 RETURN
1350 END
```

HT-1080Z LISTÁK

2.1 FELADAT

```
10 REM*****************
20 REM*                 *
30 REM*     AMOR1       *
40 REM*                 *
50 REM*****************
60 REM***** VEZERLES *****
70 GOSUB 200
80 GOSUB 300
90 GOSUB 500
100 INPUT"TOVABB ? (IGEN=1, NEM=0) ";T
110 IF T=1 THEN 80
120 GOTO 860
200 REM****** EVSZAM BEOLVASAS *****
210 INPUT"JELENLEGI EVSZAM: ";J
220 RETURN
300 REM***** EVSZAM BEOLVASAS *****
310 READ B,E
320 K=J-E
330 IF K<5 THEN 350
340 GOTO 380
350 REM* THEN AG *
360 N=(B*(5-K))/5
370 GOTO 400
380 REM* ELSE AG *
390 N=0.1
400 REM* IF VEG *
410 RETURN
500 REM***** KIIRAS *****
510 CLS
520 PRINT"          NETTO ERTEK SZAMITAS"
530 PRINT"          ----- ----- --------"
540 PRINT: PRINT
550 PRINT"BRUTTO ERTEK       ";B;"EFT"
560 PRINT"A BESZERZES EVE    ";E;"EV"
570 PRINT"ELETKOR            ";K;"EV"
```

```
580 IF K<5 THEN 600
590 GOTO 630
600 REM* THEN AG *
610 PRINT"NETTO ERTEK          ";N;"EFT"
620 GOTO 650
630 REM* ELSE AG *
640 PRINT"NETTO ERTEK          ";N;"EFT (ESZMEI ERTEK)
650 REM* IF VEG *
660 PRINT: PRINT
670 RETURN
800 REM***** ADATOK *****
810 DATA 2500,1982,500,1984
820 DATA 3050,1984,162,1984
830 DATA 520,1980,1610,1978
840 DATA 3960,1979,865,1980
850 DATA 32,1976,785,1981
860 END
```

5. FELADAT

```
10 REM*****************
20 REM*                 *
30 REM*     KAMAT       *
40 REM*                 *
50 REM*****************
60 REM***** ADATBEOLV. *****
70 INPUT" A KOLCSON OSSSZEGE: ";F
80 IF F<=0 THEN 100
90 GOTO 130
100 REM* THEN AG *
110 PRINT:PRINT"CSAK POZITIV LEHET !"
120 GOTO 70
130 REM* IF VEG *
140 INPUT"KAMATLAB (%): ";K
150 IF K<=0 THEN 170
160 GOTO 200
170 REM* THEN AG *
180 PRINT: PRINT"CSAK POZITIV LEHET !"
190 GOTO 140
200 REM* IF VEG *
210 K=K/100
230 REM***** RESZLET SZAMITASA *****
240 K1=K/12
250 R=F*K1/(1-(1/(1+K1)^12))
260 REM***** HK, CK, FJ, SZAM *****
270 PRINT: PRINT
280 PRINT"RESZLET","KAMAT","VISSZA-","KOLCSON"
290 PRINT"OSSZEG","OSSZEG","FIZ. O.","OSSZEG"
300 PRINT
310 REM* CIKLUS KEZDET *
350 FOR H=1 TO 12
360 HK=F*K1
370 CK=R-HK
380 F=F-CK
390 REM***** OSSZEGZES *****
400 SR=SR+R
410 SK=SK+HK
420 SC=SC+CK
440 REM***** KIIRAS *****
450 R=INT(R+0.5)
460 HK=INT(HK+0.5)
470 CK=INT(CK+0.5)
480 F=INT(F+0.5)
```

```
490 PRINT R,HK,CK,F
500 NEXT H
510 REM* CIKLUS VEG *
530 REM***** OSSZEG KIIRAS *****
540 PRINT
550 SR=INT(SR+0.5)
560 SK=INT(SK+0.5)
570 SC=INT(SC+0.5)
580 PRINT SR,SK,SC
590 END
```

7. FELADAT

```
10 REM*****************
20 REM*                 *
30 REM*     MSZK        *
40 REM*                 *
50 REM*****************
60 REM***** FELADAT VEZ. *****
70 DIM K(100,3)
80 GOSUB 200
90 GOSUB 300
100 GOTO 1050
200 REM***** DATUM BEOLV.*****
210 INPUT"EV: ";EV
220 PRINT
230 INPUT"HO: ";HO
240 PRINT
250 INPUT"NAP: ";NA
260 RETURN
300 REM***** MENU *****
310 CLS
320 PRINT: PRINT
330 PRINT"        KIFIZETESI MUVELETEK"
340 PRINT:PRINT
350 PRINT"  (1) KIFIZETESI ADATOK BEIRASA"
360 PRINT
370 PRINT"  (2) KIMUTATAS KESZITES"
380 PRINT: PRINT
390 INPUT"MELYIKET VALASZTJA ";D
400 IF D=1 THEN GOSUB 500
410 IF D=2 THEN GOSUB 600
420 IF D=2 THEN 440
430 GOTO 300
440 RETURN
500 REM***** KIFIZ. ADATOK *****
510 CLS
520 N=N+1
530 INPUT"MUNKASZAM: ";K(I,1)
540 PRINT
550 INPUT"OSSZEG: ";K(I,2)
560 PRINT
570 INPUT"BIZONYLAT SORSZAM: ";K(I,3)
590 RETURN
```

```
600 REM***** KIMUTATAS VEZ. *****
610 GOSUB 700
620 GOSUB 900
630 RETURN
700 REM***** RENDEZES *****
710 F=0
720 FOR I=1 TO N-1
730 IF K(I,1)>K(I+1,1) THEN 750
740 GOTO 880
750 REM* THEN AG *
760 SM=K(I,1)
770 K(I,1)=K(I+1,1)
780 K(I+1,1)=SM
790 SF=K(I,2)
800 K(I,2)=K(I+1,2)
810 K(I+1,2)=SF
820 SB=K(I,3)
830 K(I,3)=K(I+1,3)
840 K(I+1,3)=SB
850 REM* IF VEG *
860 F=1
870 NEXT I
880 IF F=1 THEN 710
890 RETURN
900 REM***** KIIRAS *****
910 LPRINT"        NAPI KIIFIZETESEK"
920 LPRINT"        ---- ------------"
930 LPRINT: LPRINT
940 LPRINT"KELT, ";EV;". EV";HO;". HO";NA;". NAP"
950 LPRINT
960 LPRINT"MUNKA-","KIFIZETETT","BIZ."
970 LPRINT"SZAM","OSSZEG","SORSZ."
980 LPRINT
990 FOR I=1 TO N
1000 LPRINT K(I,1),K(I,2),K(I,3)
1010 NEXT I
1020 RETURN
1050 END
```

8. FELADAT

```
10 REM*****************
20 REM*                 *
30 REM*     CSATA       *
40 REM*                 *
50 REM*****************
60 REM***** CSATA VEZ. *****
70 GOSUB 130
80 GOSUB 180
90 GOSUB 500
100 GOSUB 600
110 GOTO 1700
130 REM***** KEZDO MUV. *****
140 GOSUB 700
150 GOSUB 800
160 RETURN
180 REM***** KEZDO LOVES *****
190 PRINT: PRINT
200 INPUT"AZ 1. LOVES KOORDINATAI (X, Y) ";X2,Y2
210 IF X2<1 THEN 230
220 GOTO 260
230 REM* THEN AG *
240 PRINT"AZ X<1 NEM LEHET!"
250 GOTO 200
260 REM* IF VEG *
270 IF X2>12 THEN 290
280 GOTO 320
290 REM* THEN AG *
300 PRINT"AZ X>12 NEM LEHET!"
310 GOTO 200
320 REM* IF VEG *
330 IF Y2<1 THEN 350
340 GOTO 380
350 REM* THENN AG *
360 PRINT"AZ Y<1 NEM LEHET!"
370 GOTO 200
380 REM* IF VEG *
390 IF Y2>12 THEN 410
400 GOTO 440
410 REM* IF VEG *
420 PRINT"AZ Y>12 NEM LEHET!"
430 GOTO 200
```

```
600 REM***** KIMUTATAS VEZ. *****
610 GOSUB 700
620 GOSUB 900
630 RETURN
700 REM***** RENDEZES *****
710 F=0
720 FOR I=1 TO N-1
730 IF K(I,1)>K(I+1,1) THEN 750
740 GOTO 880
750 REM* THEN AG *
760 SM=K(I,1)
770 K(I,1)=K(I+1,1)
780 K(I+1,1)=SM
790 SF=K(I,2)
800 K(I,2)=K(I+1,2)
810 K(I+1,2)=SF
820 SB=K(I,3)
830 K(I,3)=K(I+1,3)
840 K(I+1,3)=SB
850 REM* IF VEG *
860 F=1
870 NEXT I
880 IF F=1 THEN 710
890 RETURN
900 REM***** KIIRAS *****
910 LPRINT"        NAPI KIIFIZETESEK"
920 LPRINT"        ---- ------------"
930 LPRINT: LPRINT
940 LPRINT"KELT, ";EV;". EV";HO;". HO";NA;". NAP"
950 LPRINT
960 LPRINT"MUNKA-","KIFIZETETT","BIZ."
970 LPRINT"SZAM","OSSZEG","SORSZ."
980 LPRINT
990 FOR I=1 TO N
1000 LPRINT K(I,1),K(I,2),K(I,3)
1010 NEXT I
1020 RETURN
1050 END
```

8. FELADAT

```
10 REM*****************
20 REM*                 *
30 REM*     CSATA       *
40 REM*                 *
50 REM*****************
60 REM***** CSATA VEZ. *****
70 GOSUB 130
80 GOSUB 180
90 GOSUB 500
100 GOSUB 600
110 GOTO 1700
130 REM***** KEZDO MUV. *****
140 GOSUB 700
150 GOSUB 800
160 RETURN
180 REM***** KEZDO LOVES *****
190 PRINT: PRINT
200 INPUT"AZ 1. LOVES KOORDINATAI (X, Y) ";X2,Y2
210 IF X2<1 THEN 230
220 GOTO 260
230 REM* THEN AG *
240 PRINT"AZ X<1 NEM LEHET!"
250 GOTO 200
260 REM* IF VEG *
270 IF X2>12 THEN 290
280 GOTO 320
290 REM* THEN AG *
300 PRINT"AZ X>12 NEM LEHET!"
310 GOTO 200
320 REM* IF VEG *
330 IF Y2<1 THEN 350
340 GOTO 380
350 REM* THENN AG *
360 PRINT"AZ Y<1 NEM LEHET!"
370 GOTO 200
380 REM* IF VEG *
390 IF Y2>12 THEN 410
400 GOTO 440
410 REM* IF VEG *
420 PRINT"AZ Y>12 NEM LEHET!"
430 GOTO 200
```

```
440 REM* IF VEG *
450 X2=INT(X2)
460 Y2=INT(Y2)
470 L=1
480 RETURN
500 REM***** TOVABBI LOVES *****
510 GOSUB 850
520 GOSUB 900
530 GOSUB 1100
540 IF (L>10) OR T=1 THEN 560
550 GOTO 510
560 RETURN
600 REM***** BEFEJEZES *****
610 IF T=1 THEN 620 ELSE 650
620 REM* THEN AG *
630 GOSUB 1200
640 GOTO 670
650 REM* ELSE AG *
660 GOSUB 1300
670 REM* IF VEG *
680 RETURN
700 REM***** JATEKSZABALYOK *****
710 CLS
720 PRINT"EGY 12X12-ES CSATAMEZOBEN KELL"
730 PRINT"EGY ISMERETLEN HELYEN LEVO"
740 PRINT"HAJOT ELTALALNI A CELPONT"
750 PRINT"KOORDINATAINAK MEGADASAVAL."
760 PRINT"A PROGRAM A BECSAPODAS ES A"
770 PRINT"HAJO TAVOLSAGAT KOZLI."
780 PRINT"A HAJORA LEGFELJEBB T I Z"
790 PRINT"      LOVEDEKET LOHET  KI."
795 RETURN
800 REM***** A HAJO HELYZETE *****
810 X1=RND(12)
820 Y1=RND(12)
830 RETURN
850 REM***** LOVESSSZAM VIZSGALAT *****
860 IF L>1 THEN GOSUB 1400
870 RETURN
900 REM***** CSILLAG KIIRAS *****
910 CLS
920 M=64*(12-Y2)+X2-1
930 PRINT é M,"*"
940 RETURN
```

```
1100 REM***** ERTEKELES *****
1110 R=INT(SQR((X2-X1)A2+(Y2-Y1)A2))
1120 IF R=0 THEN T=1
1130 FOR I=1 TO Y2
1140 PRINT
1150 NEXT I
1160 L=L+1
1170 PRINT"A BECSAPODAS TAVOLSAGA: ";R
1180 PRINT"A";L;". LOVES KOVETKEZIK."
1190 RETURN
1200 REM***** TALALAT *****
1210 PRINT"ON ELSULLYESZTETTE AZ ELLENSEG HAJOJAT!"
1220 RETURN
1300 REM***** NEM TALALT *****
1310 PRINT"EZ NEM SIKERULT!"
1320 PRINT"A HAJO A";X1;",";Y1;"KOORDINATAKON ALLT!"
1330 RETURN
1400 REM***** LOVES *****
1410 A$=INKEY$
1415 IF A$="" THEN 1410
1420 IF A$="P" THEN 1470
1430 IF A$="L" THEN 1510
1440 IF A$=";" THEN 1550
1450 IF A$="." THEN 1590
1460 GOTO 1410
1470 REM* P GOMB *
1480 IF Y2=12 THEN Y2=1: GOTO 1620
1490 Y2=Y2+1
1500 GOTO 1620
1510 REM* L GOMB *
1520 IF X2=1 THEN X2=12: GOTO 1620
1530 X2=X2-1
1540 GOTO 1620
1550 REM* ; GOMB *
1560 IF X2=12 THEN X2=1: GOTO 1620
1570 X2=X2+1
1580 GOTO 1620
1590 REM* . GOMB *
1600 IF Y2=1 THEN Y2=12: GOTO 1620
1610 Y2=Y2-1
1620 RETURN
1630 P
1700 PRINT"VEGE"
1710 END
```

PRIMO LISTÁK

2.1 FELADAT

```
10 REM*****************
20 REM*                 *
30 REM*      AMOR1      *
40 REM*                 *
50 REM*****************
60 REM***** Vezérlés *****
70 GOSUB 200
80 GOSUB 300
90 GOSUB 500
100 INPUT"Tovább ? (Igen=1, Nem=0) ";T
110 IF T=1 THEN 80
120 GOTO 860
200 REM***** Évszám beolvasás *****
210 INPUT"Jelenlegi évszám: ";J
220 RETURN
300 REM***** Amortizáció számitás *****
310 READ B,E
320 K=J-E
325 PRINT K
330 IF K<5 THEN 350
340 GOTO 380
350 REM* Then ág *
360 N=(B*(5-K))/5
370 GOTO 400
380 REM* Else ág *
390 N=0.1
400 REM* If vég *
410 RETURN
500 REM***** Kiirás *****
510 CLS
520 PRINT"       Nettó érték számitás"
530 PRINT"       ----- ----- --------"
540 PRINT: PRINT
550 PRINT"Bruttó érték          ";B;"eFt"
560 PRINT"A beszerzés éve       ";E;"év"
570 PRINT"Életkor               ";K;"év"
```

```
580 IF K<5 THEN 600
590 GOTO 630
600 REM* Then ág *
610 PRINT"Nettó érték          ";N;"eFT"
620 GOTO 650
630 REM* Else ág *
640 PRINT"Nettó érték          ";N;"eFt (eszmei érték)"
650 REM* If vég *
660 PRINT: PRINT
670 RETURN
800 REM* Adatok *
810 DATA 2500,1982,500,1984
820 DATA 3050,1984,162,1984
830 DATA 520,1980,1610,1978
840 DATA 3960,1979,865,1980
850 DATA 32,1976,785,1981
860 END
```

```
10 REM***************
20 REM*             *
30 REM*    KAMAT    *
40 REM*             *
50 REM***************
60 REM* Adatbeolvasás *
70 INPUT"A kölcsön összege: ";F
80 IF F<=0 THEN 100
90 GOTO 130
100 REM* Then ág *
110 PRINT: PRINT"Csak pozitiv lehet!"
120 GOTO 70
130 REM* If vég *
140 INPUT"Kamatláb (%): ";K
150 IF K<=0 THEN 170
160 GOTO 200
170 REM* Then ág *
180 PRINT: PRINT"Csak pozitiv lehet! "
190 GOTO 140
200 REM* If vég *
210 K=K/100
230 REM***** Részlet számítás *****
240 K1=K/12
250 R=F*K1/(1-(1/(1+K1)^12))
260 REM***** HK, CK,FJ, Szám. *****
270 PRINT: PRINT
280 PRINT"Részlet";TAB(10);"Kamat";TAB(20);"Vissza-";TAB(30);"Kölcsön"
290 PRINT"összeg";TAB(10);"összeg";TAB(20);"fiz. ö.";TAB(30);"összeg"
350 FOR H=1 TO 12
```

```
360 HK=F*K1
370 CK=R-HK
380 F=F-CK
390 REM***** Összegzés *****
400 SR=SR+R
410 SK=SK+HK
420 SC=SC+CK
440 REM***** Kiírás *****
450 R=INT(R+0.5)
460 HK=INT(HK+0.5)
470 CK=INT(CK+0.5)
480 F=INT(F+0.5)
490 PRINT R;TAB(10);HK;TAB(20);CK;TAB(30);F
500 NEXT H
510 REM* Ciklus vég *
530 REM***** Összeg kiírás *****
550 SR=INT(SR+0.5)
560 SK=INT(SK+0.5)
570 SC=INT(SC+0.5)
580 PRINT SR;TAB(10);SK;TAB(20);SC
585 INPUT A
590 END
```

7. FELADAT

```
10 REM*****************
20 REM*                 *
30 REM*     MSZK        *
40 REM*                 *
50 REM*****************
60 REM***** Feladat vez. *****
70 DIM K(100,3)
80 GOSUB 200
90 GOSUB 300
100 GOTO 1050
200 REM***** Dátum beolv. *****
210 INPUT "Év: ";EV
220 PRINT
230 INPUT "Hó: ";HO
240 PRINT
250 INPUT "Nap: ";NA
260 RETURN
300 REM***** Menü *****
310 CLS
320 PRINT: PRINT
330 PRINT"          KIFIZETÉSI MÜVELETEK"
340 PRINT: PRINT
350 PRINT"  (1) Kifizetési adatok beirása"
360 PRINT
370 PRINT"  (2) Kimutatás készités"
380 PRINT: PRINT
390 INPUT"Melyiket választja ";D
400 IF D=1 THEN GOSUB 500
410 IF D=2 THEN GOSUB 600
420 IF D=2 THEN 440
430 GOTO 300
440 RETURN
500 REM***** Kifiz. adatok *****
510 CLS
520 N=N+1
530 INPUT"Munkaszám: ";K(N,1)
540 PRINT
550 INPUT"Összeg: ";K(N,2)
560 PRINT
570 INPUT"Bizonylat sorszám: ";K(N,3)
590 RETURN
```

```
600 REM***** Kimutatás vezérlés *****
610 GOSUB 700
620 GOSUB 900
630 RETURN
700 REM***** Rendezés *****
710 F=0
720 FOR I=1 TO N-1
730 IF K(I,1)>K(I+1,1) THEN 750
740 GOTO 880
750 REM* Then ág *
760 SM=K(I,1)
770 K(I,1)=K(I+1,1)
780 K(I+1,1)=SM
790 SF=K(I,2)
800 K(I,2)=K(I+1,2)
810 K(I+1,2)=SF
820 SB=K(I,3)
830 K(I,3)=K(I+1,3)
840 K(I+1,3)=SB
850 REM* If vég *
860 F=1
870 NEXT I
880 IF F=1 THEN 710
890 RETURN
900 REM***** Kiírás *****
910 LPRINT"           NAPI KIFIZETÉSEK"
920 LPRINT"           ---- -----------"
930 LPRINT: LPRINT
940 LPRINT"Kelt, ";EV;". év ";HO;". hó ";NA;". nap"
950 LPRINT
960 LPRINT"Munka-";TAB(10);"Kifizetett";TAB(12);"Biz.
970 LPRINT"szám";TAB(14);"összeg";TAB(12);"sorsz."
980 LPRINT
1000 FOR I=1 TO N
1010 LPRINT K(I,1);TAB(14);K(I,2);TAB(12);K(I,3)
1020 NEXT I
1030 RETURN
1050 END
```

8. FELADAT

```
10 REM*****************
20 REM*                 *
30 REM*     CSATA       *
40 REM*                 *
50 REM*****************
60 REM***** Csata vez. *****
70 GOSUB 130
80 GOSUB 180
90 GOSUB 500
100 GOSUB 600
110 GOTO 1700
130 REM***** Kezdö müveletek *****
140 GOSUB 700
150 GOSUB 800
160 RETURN
180 REM***** Kezdö lövés *****
190 PRINT: PRINT
200 INPUT"Az 1. lövés koordinátái (X,Y,):";X2,Y2
210 IF X2<1 THEN 230
220 GOTO 260
230 REM* Then ág *
240 PRINT"Az X<1 nem lehet!"
250 GOTO 200
260 REM* If vég *
270 IF X2>12 THEN 290
280 GOTO 320
290 REM* Then ág *
300 PRINT"Az X>12 nem lehet!"
310 GOTO 200
320 REM* Then ág *
330 IF Y2<1 THEN 350
340 GOTO 380
350 REM* Then ág *
360 PRINT"Az Y<1 nem lehet!"
370 GOTO 200
380 REM* If vég *
390 IF Y2>12 THEN 410
400 GOTO 440
410 REM* If vég *
420 PRINT"Az Y>12 nem lehet!"
430 GOTO 200
```

```
440 REM* If vég *
450 X2=INT(X2)
460 Y2=INT(Y2)
470 L=1
480 RETURN
500 REM***** További lövés *****
510 GOSUB 850
520 GOSUB 900
530 GOSUB 1100
540 IF (L>10) OR T=1 THEN 560
550 GOTO 510
560 RETURN
600 REM***** Befejezés *****
610 IF T=1 THEN 620 ELSE 650
620 REM* Then ág *
630 GOSUB 1200
640 GOTO 670
650 REM* Else ág *
660 GOSUB 1300
670 REM* If vég *
680 RETURN
700 REM***** Játékszabályok *****
710 CLS
720 PRINT"Egy 12X12-es csatamezöben kell egy"
730 PRINT"ismeretlen helyen levö hajót eltalálni"
740 PRINT"a célpont koordinátáinak megadásával."
750 PRINT"A program a becsapódás és a hajó"
760 PRINT"távolságát közli."
770 PRINT"A hajókra 10 lövedéket löhet ki."
780 RETURN
800 REM***** Hajó helyzete *****
810 RANDOM
820 X1=RND(11)+1
830 Y1=RND(11)+1
840 RETURN
850 REM***** Lövésszám vizsgálat *****
860 IF L>1 THEN GOSUB 1400
870 RETURN
900 REM***** Csillag kiírás *****
910 CLS
920 PRINT@ (12-Y2),(X2-1),"*"
930 RETURN
```

```
1100 REM***** Értékelés *****
1110 R=INT(SQR((X2-X1)^2+(Y2-Y1)^2))
1120 IF R=0 THEN T=1
1130 FOR I=1 TO Y2
1140 PRINT
1150 NEXT I
1160 L=L+1
1170 PRINT"A becsapódás távolsága: ";R
1180 PRINT"A";L;". lövés következik."
1190 RETURN
1200 REM***** Találat *****
1210 PRINT"Ön elsüllyesztette az ellenség hajóját!"
1220 RETURN
1300 REM***** Nem talált *****
1310 PRINT"Ez nem sikerült!"
1320 PRINT"A hajó a";X1;",";Y1;"koordinátákon állt!"
1330 RETURN
1400 REM***** Lövés *****
1410 A$=INKEY$
1415 IF A$="" THEN 1410
1420 IF A$="Ü" THEN 1470
1430 IF A$="Á" THEN 1510
1440 IF A$="*" THEN 1550
1450 IF A$="ó" THEN 1590
1460 GOTO 1410
1470 REM* Felfelé gomb *
1480 IF Y2=12 THEN Y2=1: GOTO 1620
1490 Y2=Y2+1
1500 GOTO 1620
1510 REM* Balra gomb *
1520 IF X2=1 THEN X2=12: GOTO 1620
1530 X2=X2-1
1540 GOTO 1620
1550 REM* Jobbra gomb *****
1560 IF X2=12 THEN X2=1: GOTO 1620
1570 X2=X2+1
1580 GOTO 1620
1590 REM* Lefelé gomb *
1600 IF Y2=1 THEN Y2=12: GOTO 1620
1610 Y2=Y2-1
1620 RETURN
1700 PRINT"          V É G E"
1710 END
```

SINCLAIR LISTÁK

2.1 FELADAT

```
 10 REM ****************
 20 REM *               *
 30 REM *     AMOR1     *
 40 REM *               *
 50 REM ****************
 60 REM ***** VEZERLES *****
 70 GO SUB 200
 80 GO SUB 300
 90 GO SUB 500
100 INPUT "TOVABB? (IGEN=1, NEM=0) ";T
110 IF T=1 THEN  GO TO 80
120 GO TO 860
200 REM ***** EVSZAM BEOLVASAS *****
210 INPUT "JELENLEGI EVSZAM: ";J
220 RETURN
300 REM ***** AMORTIZACIO SZAMITAS *****
310 READ B,E
320 LET K=J-E
330 IF K<5 THEN  GO TO 350
340 GO TO 380
350 REM * THEN AG *
360 LET N=(B*(5-K))/5
370 GO TO 400
380 REM * ELSE AG *
390 LET N=0.1
400 REM * IF VEG *
410 RETURN
500 REM ***** KIIRAS *****
510 CLS
520 PRINT "   NETTO ERTEK SZAMITAS"
530 PRINT "   _____ _____ ________"
540 PRINT : PRINT
550 PRINT "BRUTTO ERTEK      ";B;" EFT"
560 PRINT "A BESZERZES EVE   ";E;" EV"
570 PRINT "ELETKOR           ";K;" EV"
```

```
580 IF K<5 THEN  GO TO 600
590 GO TO 630
600 REM * THEN AG *
610 PRINT "NETTO ERTEK       ";N;" EFT"
620 GO TO 660
630 REM * ELSE AG B
640 PRINT "NETTO ERTEK       ";N;" EFT (ESZMEI"
650 PRINT "                          ERTEK)"
660 REM * IF VEG *
670 PRINT : PRINT
680 RETURN
800 REM ***** ADATOK *****
810 DATA 2500,1982,500,1984
820 DATA 3050,1984,162,1984
830 DATA 520,1980,1610,1978
840 DATA 3960,1979,865,1980
850 DATA 32,1976,785,1981
860 REM
```

5. FELADAT

```
10 REM ****************
20 REM *               *
30 REM *   KAMAT       *
40 REM *               *
50 REM ****************
60 REM ***** ADATBEOLV. *****
70 INPUT "A KOLCSON OSSZEGE: ";F
80 IF F<=0 THEN  GO TO 100
90 GO TO 130
100 REM * THEN AG *
110 PRINT : PRINT "CSAK POZITIV LEHET!"
120 GO TO 70
130 REM * IF VEG *
140 INPUT "KAMATLAB (%): ";K
150 IF K<=0 THEN  GO TO 170
160 GO TO 200
170 REM * THEN AG *
180 PRINT : PRINT "CSAK POZITIV LEHET!"
190 GO TO 140
200 REM * IF VEG *
210 LET K=K/100
230 REM ***** RESZLET SZAMITAS *****
240 LET K1=K/12
250 LET R=F*K1/(1-(1/(1+K1)^12))
260 REM ***** HK, CK, FJ SZAM. *****
270 PRINT : PRINT
280 PRINT "RESZLET";TAB 8;"KAMAT";TAB 16;"VISSZA";TAB 24;"KOLCSON"
290 PRINT "OSSZEG";TAB 8;"OSSZEG";TAB 16;"FIZ. O.";TAB 24;"OSSZEG"
300 PRINT
310 LET SR=0: LET SK=0: LET SC=0
320 REM * CIKLUS KEZDET *
350 FOR H=1 TO 12
360 LET HK=F*K1
370 LET CK=R-HK
380 LET F=F-CK
390 REM ***** OSSZEGZES *****
400 LET SR=SR+R
410 LET SK=SK+HK
420 LET SC=SC+CK
440 REM ***** KIIRAS *****
450 LET R=INT R+0.5
460 LET HK=INT HK+0.5
470 LET CK=INT CK+0.5
```

```
480 LET F=INT F+0.5
490 PRINT R;TAB 8;HK;TAB 16;CK;TAB 24;F
510 NEXT H
520 REM * CIKLUS VEG *
530 REM ***** OSSZEG KIIRAS *****
540 PRINT
550 LET SR=INT SR+0.5
560 LET SK=INT SK+0.5
570 LET SC=INT SC+0.5
580 PRINT SR;TAB 8;SK;TAB 16;SC
590 REM
```

7. FELADAT

```
 10 REM ****************
 20 REM *               *
 30 REM *     MSZK      *
 40 REM *               *
 50 REM ****************
 60 REM ***** FELADAT VEZ. *****
 65 LET N=0
 70 DIM K(100,3)
 80 GO SUB 200
 90 GO SUB 300
100 GO TO 1050
200 REM ***** DATUM BEOLV. *****
210 INPUT "EV: ";EV
220 PRINT
230 INPUT "HO: ";HO
240 PRINT
250 INPUT "NAP: ";NA
260 RETURN
300 REM ***** MENU *****
310 CLS
320 PRINT : PRINT
330 PRINT "    KIFIZETESI MUVELETEK"
340 PRINT : PRINT
350 PRINT "  (1) KIFIZETESI ADATOK BEIRASA"
360 PRINT
370 PRINT "  (2) KIMUTATAS KESZITES"
380 PRINT : PRINT
390 INPUT "MELYIKET VALASZTJA ? ";D
400 IF D=1 THEN  GO SUB 500
410 IF D=2 THEN  GO SUB 600
420 IF D=2 THEN  GO TO 440
430 GO TO 300
440 RETURN
500 REM ***** KIFIZ. ADATOK *****
510 CLS
520 LET N=N+1
530 INPUT "MUNKASZAM: ";K(N,1)
540 PRINT
550 INPUT "OSSZEG:  ";K(N,2)
560 PRINT
570 INPUT "BIZONYLAT SORSZAM: ";K(N,3)
590 RETURN
```

```
600 REM ***** KIMUTATAS VEZ. *****
610 GO SUB 700
620 GO SUB 900
630 RETURN
700>REM ***** RENDEZES *****
705 LET SM=0: LET SF=0: LET SB=0
710 LET F=0
720 FOR I=1 TO N-1
730 IF K(I,1)>K(I+1,1) THEN  GO TO 750
740 GO TO 880
750 REM * THEN AG *
760 LET SM=K(I,1)
770 LET K(I,1)=K(I+1,1)
780 LET K(I+1,1)=SM
790 LET SF=K(I,2)
800 LET K(I,2)=K(I+1,2)
810 LET K(I+1,2)=SF
820 LET SB=K(I,3)
830 LET K(I,3)=K(I+1,3)
840 LET K(I+1,3)=SB
850 REM * IF VEG *
860 LET F=1
870 NEXT I
880 IF F=1 THEN  GO TO 710
890 RETURN
900 REM ***** KIIRAS *****
910 LPRINT "    NAPI KIFIZETESEK"
920 LPRINT "    ---- -----------"
930 LPRINT
940 LPRINT
950 LPRINT "KELT. ";EV;". EV ";HO;". HO ";NR;" . NAP"
960 LPRINT
970 LPRINT "MUNKA-";TAB 10;"KIFIZETETT";TAB 25;" BIZ."
980 LPRINT "SZAM",TAB 10;" OSSZEG";TAB 25;"SORSZ."
990 LPRINT
1000 FOR I=1 TO N
1010 LPRINT K(I,1);TAB 10;K(I,2);TAB 25;K(I,3)
1020 NEXT I
1040 RETURN
1050 REM
```

8. FELADAT

```
 10 REM ****************
 20 REM *               *
 30 REM *  CSATA        *
 40 REM *               *
 50 REM ****************
 60 REM * CSATA VEZ. *****
 65 LET T=0
 70 GO SUB 130
 80 GO SUB 180
 90 GO SUB 500
100 GO SUB 600
110 GO TO 1700
130 REM ***** KEZDO MUV. *****
140 GO SUB 700
150 GO SUB 800
160 RETURN
180 REM ***** KEZDO LOVES *****
190 PRINT : PRINT
200 INPUT "Az elso loves koordinatai (X,Y) ";X2,Y2
210 IF X2<1 THEN  GO TO 230
220 GO TO 260
230 REM * THEN AG *
240 PRINT "AZ X<1 NEM LEHET!"
250 GO TO 200
260 REM * IF VEG *
270 IF X2>18 THEN  GO TO 290
280 GO TO 320
290 REM * THEN AG *
300 PRINT "AZ X>18 NEM LEHET!"
310 GO TO 200
320 REM * IF VEG *
330 IF Y2<1 THEN  GO TO 350
340 GO TO 380
350 REM * THEN AG *
360 PRINT "AZ Y<1 NEM LEHET!"
370 GO TO 200
380 REM * IF VEG *
390 IF Y2>18 THEN  GO TO 410
400 GO TO 440
410 REM * IF VEG *
420 PRINT "AZ Y>18 NEM LEHET!"
430 GO TO 200
```

```
440 REM * IF VEG *
450 LET X2=INT X2
460 LET Y2=INT Y2
470 LET L=1
480 RETURN
500)REM ***** TOVABBI LOVES *****
510 GO SUB 850
520 GO SUB 900
530 GO SUB 1100
540 IF (L)12) OR T=1 THEN  GO TO 560
550 GO TO 510
560 RETURN
600 REM ***** BEFEJEZES *****
610 IF T=1 THEN  GO TO 630
620 GO TO 660
630 REM * THEN AG *
640 GO SUB 1200
650 GO TO 680
660 REM * ELSE AG *
670 GO SUB 1300
680 REM * IF VEG *
690 RETURN
700 REM *****JATEKSZABALYOK *****
710 CLS
720 PRINT "Egy 18X18-as mezoben kell egy"
730 PRINT "ismeretlen helyen levo hajot"
740 PRINT "eltalalni a celPont koordina-"
745 PRINT "tainak megadasaval."
750 PRINT
755 PRINT "A program a becsaPodas es a hajo"
760 PRINT "tavolsagat kozli."
765 PRINT
770 PRINT "A hajokra legfeljebb 12 lovede-"
775 PRINT "ket lohet ki."
780 RETURN
800 REM ***** HAJO HELYZETE *****
810 RANDOMIZE
820 LET X1=1+INT (RND*18)
830 LET Y1=1+INT (RND*18)
840 RETURN
```

```
850 REM ***** LOVESSZAM VIZSGALAT *****
860 IF L>1 THEN  GO SUB 1400
870 RETURN
900 REM ***** CSILLAG KIIRAS *****
910 CLS
920 PRINT AT 18-Y2,X2-1;"*"
930 RETURN
1100>REM ***** ERTEKELES *****
1110 LET R=INT (SQR (ABS (X2-X1)^2+ABS (Y2-Y1)^2))
1120 IF R=0 THEN  LET T=1
1130 FOR I=1 TO Y2
1140 PRINT
1150 NEXT I
1160 LET L=L+1
1170 PRINT "A becsapodas tavolsaga: ";R
1180 PRINT "A ";L;". loves kovetkezik."
1190 RETURN
1200 REM ***** TALALAT *****
1210 PRINT "On elsullyesztette az ellenseg"
1215 PRINT "hajojat!"
1220 RETURN
1300 REM ***** NEM TALALT *****
1310 PRINT "Ez nem sikerult!"
1320 PRINT "A hajo a ";x1;",";y1;" koordinatakon allt!"
1330 RETURN
1400 REM ***** LOVES *****
1410 IF INKEY$="" THEN  GO TO 1410
1420 IF INKEY$="I" THEN  GO TO 1470
1430 IF INKEY$="J" THEN  GO TO 1510
1440 IF INKEY$="K" THEN  GO TO 1550
1450 IF INKEY$="M" THEN  GO TO 1590
1460 GO TO 1410
1470 REM * I GOMB *
1480 IF Y2=18 THEN  LET Y2=1: GO TO 1620
1490 LET Y2=Y2+1
1500 GO TO 1620
1510 REM * J GOMB *
1520 IF X2=1 THEN  LET X2=18: GO TO 1620
1530 LET X2=X2-1
1540 GO TO 1620
1550 REM * K GOMB *
1560 IF X2=18 THEN  LET X2=1: GO TO 1620
       X2=X2+1
      TO 1620
```

```
1590 REM * M GOMB *
1600 IF Y2=1 THEN  LET Y2=18: GO TO 1620
1610 LET Y2=Y2-1
1620 RETURN
1700 PRINT "     V E G E"
1710 REM
```

TÁRGYMUTATÓ

A

B

C

D

E

J

K

L

R

S

SZ

T

U

V

Kiadja: Számítástechnika-alkalmazási Vállalat
Felelős kiadó: Juhász János vezérigazgató

Felelős szerkesztő: Huba Zoltánné
Műszaki vezető: Molnár Zoltán
Műszaki szerkesztő: G. Müller Zsuzsa
A fedelet Székely Edith tervezte
Az ábrákat Zeikfalvy Lenke rajzolta
Megjelent: 27,5 (A/5) ív terjedelemben

Készült a Dabasi Nyomdában
Felelős vezető: Bálint Csaba igazgató
A nyomdai megrendelés törzsszáma: 84–2054
Budapest–Dabas, 1984.

Ára: 120,– Ft