Sasse

COMPILER

DATA BECKER – NOVOTRADE

Sasse

COMPILER

DATA BECKER – NOVOTRADE

A mű eredeti címe: COMPILER • verstehen • anwenden • entwickeln

Fordította: DR. SZÉCHYNÉ DR. BÁLINT ÁGNES

Lektorálta: DR. BENDE SÁNDOR és BARTHA IMRE

A kiadásért felel: RÉNYI GÁBOR, a NOVOTRADE RT. igazgatója

Kiadványmenedzser: BÉKÉS TAMÁS
Felelős szerkesztő: SZŐLLŐSI ERZSÉBET
Műszaki szerkesztő: DÉVÉNYI ERIKA

Szedte a Nyomdaipari Fényszedő Üzem, Budapest (86.8507/09)
Készült a Somogy Megyei Nyomdaipari Vállalatnál, Kaposváron (15 A/5 ív)

ISBN 963 02 4536 1
Hungarian translation © Dr. Széchyné dr. Bálint Ágnes

Copyright © 1984 DATA BECKER GmbH – Merowingerstr. 30. 4000 Düsseldorf

Minden jog fenntartva. A DATA BECKER cég írásbeli hozzájárulása nélkül tilos a könyvet vagy annak részeit bármilyen eljárással (nyomtatás, fotókópia vagy egyéb technika), elektronikus rendszerek felhasználásával másolni, sokszorosítani, terjeszteni.

FONTOS TUDNIVALÓ

A jelen könyv keretén belül ismertetett kapcsolások, eljárások és programok nem tekinthetők szabadalmi oltalom alá eső ipari termékeknek. Ezek elsősorban amatőr és oktatási célokat szolgálnak. A szerzők rendkívül nagy gondot fordítottak a kapcsolások, műszaki adatok és programok helyességére, a részletek kidolgozása során többszöri ellenőrzést végeztek. Mindez azonban nem zárja ki az esetleges hibalehetőségeket.
Az előforduló hibákért és az ebből adódó következményekért a DATA BECKER cég sem szavatosságot, sem jogi felelősséget nem vállal. Az esetlegesen előforduló hibák közlését a szerzők hálásan fogadják.

TARTALOMJEGYZÉK

BEVEZETÉS

Bizonyára sokan szeretnék tudni, hogyan működik egy programnyelveket fordító rendszer. Talán olyanok is vannak, akik maguk által szerkesztett nyelvekhez szeretnének fordítóprogramot írni.

Nagyszerű érzés, ha valaki olyan programot tud készíteni, amelyik egyik programnyelvről egy másik programnyelvre fordít. A könyv tanulmányozása után az Olvasó számára ez érthetővé válik, sőt maga is tud majd fordítóprogramot írni.

Először a MINIATUR programnyelvet ismertetjük, amit a fordítók bemutatására fejlesztettünk ki.

Ismertetjük a fordításhoz szükséges eljárásokat és végrehajtjuk a MINIATUR nyelvű program BASIC programmá alakítását. Ezzel az Olvasó a könyvben leírt nyelvre vonatkozóan a teljes fordító birtokába jut, amelyet tanulmányozhat és tetszés szerint módosíthat, pl. bővítheti a leírt nyelvet.

Ezt mintaként is használhatjuk arra, hogy egy tetszés szerinti nyelvhez fordítót írjunk. Előfordulhat, hogy a számítógépet valamilyen meghatározott célra kívánjuk használni, de erre a célra még nincs megfelelő programnyelv. Ebben az esetben ki lehet fejleszteni egy olyan nyelvet, amely jól illeszkedik a problémáinkhoz, és meg lehet írni hozzá a megfelelő fordítót.

Ezt a könyvet azonban nemcsak azoknak szánjuk, akik a compiler működését meg akarják érteni, vagy fordítóprogramot akarnak írni, hanem azoknak is, akik többet kívánnak tudni a számítógépek működéséről.

Bekapcsolódhat az Olvasó a gépi kódban történő programozásba is, könyvünkben ugyanis egy teljes ASSEMBLER- és egy DISASSEMBLER programot ismertetünk. A fordításhoz használt 6510-es gépi utasításai is megtalálhatóak a könyvben.

Az olvasó így alaposabban megismeri a programnyelvek és a számítógépek működését.

1. MI A COMPILER?

1.1 Miért van szükség compilerre?

A mai számítógépek szíve a mikroprocesszor. A mikroprocesszorok azonban csak olyan programokat tudnak végrehajtani, amelyeket az ő gépi kódjukban írtak. Most azt gondolhatjuk, hogy a mi számítógépünk a BASIC programnyelvet megérti. Ez az ellentmondás csupán látszólagos! Számítógépünk azért érti meg a BASIC programnyelvet, mert egy olyan gépi kódban írt programot tárol, amely lehetővé teszi, hogy a BASIC utasításokat felismerje és végrehajtsa. Ha tehát egy BASIC programot futtatunk, a számítógép valójában egy olyan gépi kódban meglévő programot hajt végre, amely a BASIC programok utasításait tartalmazza, vagyis felismeri, értelmezi és az ezeknek megfelelő gépi műveleteket elvégzi. Ez meglehetősen sok munkát igénylő és mindenekelőtt időt rabló módja a program végrehajtásának. Felmerül tehát a kérdés, hogy akkor miért nem programozunk egyszerűen gépi kódban?

A gépi kódú programozás nagyon távol áll az ember gondolkodásától. A gépi kódú program a kettes, a tizenhatos vagy a tízes számrendszerbeli számjegyek sorozata. Ez a sorozat a program valódi, hű képe, ahogyan azt a mikroprocesszor a számítógép tárjában megtalálja.

Milyen utasítást hajt végre a COMMODORE számítógép, ha a következő gépi kódú programot találja?

Tízes számrendszerben:

169 65 32 210 255

Tizenhatos számrendszerben:

A2 41 20 02 FF

Kettes számrendszerben:

10101001 01000001 00100000 11010010 11111111

Ezekből a számsorozatokból nehéz kitalálni, hogy a fenti utasítás hatására az A betű jelenik meg a képernyőn.

A PRINTA BASIC utasítás viszont már sokkal közelebb áll az emberi gondolkodásmódhoz. A PRINTA BASIC utasítás gépi végrehajtásának a megindításához a BASIC interpreternek (értelmezőnek) mintegy százszor hosszabb gépi kódú programra van szüksége. A mikroprocesszor a programnyelvű utasítás végrehajtása előtt viszonylag hosszú időt tölt azzal, hogy felismerje, mit is kell tennie. Sokszor egy utasítás felismerése tovább tart, mint a végrehajtása. A fordítás sokkal bonyolultabb lesz, ha pl. a PRINTA BASIC utasítást egymás után ezerszer kell végrehajtani. Az utasítást ekkor egymás után ezerszer kell felismerni, de az interpreterrel ez nem megy másként.

Gondoljuk végig a fordítás folyamatát!

Vegyünk egy olyan programot, amelyet valamilyen magasabb szintű nyelven (pl. BASIC-ben) írtunk, és ezt alakítsuk át gépi kódú programmá. Így olyan programmal rendelkezünk, amelyet kényelmesen megírhatunk és amelyet a gép mégis gyorsan fel tud dolgozni, mert az utasítások hosszadalmas felismerése elmarad.

Nos, ezt a munkát végzi a compiler!

A compiler tehát nem más, mint egy olyan program, amelynek az a feladata, hogy valamilyen magasabb szintű nyelven megírt programot gépi kódú programmá alakítson át.

Szükségszerűen felmerül a kérdés: milyen nyelven írjuk meg a compilert?

Ha egy elhagyott szigeten lennénk olyan számítógéppel, amelynek valamilyen okból az az építőeleme, amely a BASIC interpretert tartalmazza, működésképtelen, akkor bizony nem lenne más választásunk, mint a compilert gépi kódban megírni.

Egy idő után biztosan azon gondolkodnánk, hogy a gépi kódot egy kissé olvashatóbbá tegyük. Azokat a műveleteket, amelyeket a mikroprocesszor egyáltalán el tud végezni, a gépi kódoláson kívül másként is kódolhatjuk.

Nézzük az előbbiekben említett példánkat (A nyomtatása). Ebben az esetben felírhatjuk:

169+65	azt jelenti, közvetlenül töltsd be az akkumulátorba az A-t;
32+210+255	azt jelenti, ugorj ahhoz az alprogramhoz, amelyik a 65490-es tárolóhelynél kezdődik.

Vagy röviden:

```
LDA # "A"
JSR 65490
```

Ezután egy olyan program írásához fognánk hozzá, amely ezeket a rövidítéseket gépi kódra fordítaná. Ezt a munkát érdemes elvégezni ahhoz, hogy a gépi kódú karaktersorozatokkal ne legyen több gondunk.

Ezeket a rövidítéseket mnemonikus kódoknak nevezik, és az ezekből felépített programot hívják assembler nyelvű programnak. Annak a programnak pedig, amely egy assembler nyelvű programot gépi kódú programmá tud alakítani, ASSEMBLER a neve. Erre később majd még visszatérünk.

Az assembler nyelvű fordítóprogram azonban még mindig eléggé áttekinthetetlen. Minthogy azonban azt akarjuk megtanulni, hogyan működik a compiler, és ráadásul a számítógépünk egyik BASIC alkotórésze sem működésképtelen, írjuk tehát fordítóprogramunkat a rendelkezésünkre álló legkényelmesebb nyelven, vagyis BASIC-ben!

1.2 A compilerek felépítése

Amint azt láttuk, a compilernek az a feladata, hogy egy programot valamilyen meghatározott nyelvről gépi kódba ültessen át.

Vizsgáljuk meg, milyen feladatokat kell itt elvégezni. A feladatok négy területre oszthatók:

1) lexikális elemzés;
2) szintaktikai elemzés;
3) szemantikai elemzés;
4) kódgenerálás.

A compilerek felépítésénél ugyanúgy kell eljárnunk, mint a programozás során mindenütt, ha egy komplex feladatot kell megoldanunk. A feladatot sok kis részfeladatra osztjuk fel, amelyeket könnyebb áttekinteni és programozni. A feladatok felsorolásánál az elemzés szó háromszor is előfordult, míg a generálás csak egyszer (generáláson a gépi kódú, ill. ASSEMBLER program létrehozását értjük). Az első három rész feladata az, hogy a programban hibakeresést végezzen és megszerezze azokat az információkat, amelyek szükségesek ahhoz, hogy a gépi kódú, ill. ASSEMBLER programot létre lehessen hozni.

1) *Lexikális elemzés*

A lexikális elemzés során a lefordítandó programot a legkisebb értelemhordozó egységekre, szavakra bontjuk, és ellenőrizzük, hogy az előforduló szavak az adott nyelv megengedett alapkarakter-készletébe tartoznak-e. Ezt a következő mondattal mutatjuk be:

Az elefánt a zsákbanfutást gyakorolja.

A lexikális elemzés során a mondatot felbontjuk:

Az / elefánt / a / zsákbanfutást / gyakorolja.

Minthogy minden egyes szó a magyar nyelvben használatos, a mondat lexikális szempontból helyes.

2) *Szintaktikai elemzés*

A szintaktikai elemzés azt vizsgálja, hogy mondatunk a szintaktikai szabályok szerint helyes-e. Mondatunk kétségtelenül helyes. A programnyelveknél ez azt jelenti, hogy a programunk elemei a nyelv jelkészleteiből a szintaktikai szabályoknak megfelelően vannak-e képezve.

Példa egy szintaktikailag hibátlan BASIC programra:

```
10 for j=1 to 10
20 for i=1 to 10
30 print j, i
40 next j
50 next i
60 end
```

Ez a program szintaktikailag hibátlan ugyan, de van-e értelme?

3) *Szemantikai elemzés*

A BASIC program szemantikai elemzésénél meg kellene állapítanunk, hogy a ciklusok rosszul vannak összekapcsolva, a 40-es és az 50-es sort fel kellene cserélnünk ahhoz, hogy futtatható programot kapjunk.

Visszatérve mondatunkhoz: nincs értelme, mivel az elefántok nem tudnak zsákbanfutni.

4) *Kódgenerálás*

Feltéve, hogy programunk az eddigi ellenőrzések alapján helyes, gépi kódra fordítandó. Gyakran azonban a programot egy ún. köztes kódba fordítják le. Ezt a köztes kódot a mikroprocesszor ugyan nem érti meg, de lehet egy gyors interpretert szerkeszteni, amely azután ezt a köztes kódot értelmezi.

Programunkból először egy assembler nyelvű programot készítünk, amelyet azután az ASSEMBLER gépi kódra le tud fordítani. Ennek megvan ugyan az a hátránya, hogy a fordítási idő hosszabb lesz, de a compiler készítést tanuló számára különösen előnyös, mivel:

a) kényelmesen nyomon követhetjük, hogy a programunk milyen gépi kódú utasításokká fordítódik le;

b) a kódgenerálási eljárások sokkal áttekinthetőbbek lesznek.

Meg kell még jegyezni azt is, hogy számunkra egyetlen felismerés sem vész el, mert ha értjük, hogy a mellérendelt ASSEMBLER hogyan működik, akkor biztosan nem okoz problémát az sem, hogy a kódgenerálást úgy alakítsuk át, hogy az közvetlenül gépi kódú program legyen. Ízelítőül egyenlőre elégedjünk meg ennyivel.

Végezetül még megjegyezzük, hogy egy compilert természetesen sokféleképpen meg lehet írni, de az alapfeladatok mindig ugyanazok. Itt is érvényes az a szabály, hogy ha az alapfeladatot megértettük, akkor azok különböző megjelenési formáit is fel tudjuk majd ismerni.

2. A PROGRAMNYELVÜNK

2.1 Bevezetés

Ebben a fejezetben áttekintést szeretnénk adni arról a nyelvről, amelyhez a compilert készítjük. Minthogy ezt a nyelvet szabadon választhatjuk, ezért olyant kell készítenünk, amely céljainknak legjobban megfelel.

Olyan nyelvre van szükségünk, amely jól áttekinthető, hogy így a compiler könnyen kezelhető legyen. Szeretnénk egyúttal a programnyelvek fejlődésében megfigyelhető tendenciákat is követni, ezért egy blokkszerkezetűt választottunk. Az alapvető lehetőségeket úgy válogattuk össze, ahogyan azok minden modern programnyelvben megtalálhatóak.

A nyelvet MINIATUR-nak neveztük el. Ez a nyelv tartalmazza a programnyelvek alapjait és könnyen bővíthető.

Nos, mit tud a MINIATUR?

A MINIATUR program egy főprogramból, és szükség szerint alprogramokból áll. Ahhoz, hogy a számítógéppel kapcsolatot tudjunk kialakítani, az adatok ki- és bevitelére, valamint a szövegek és a vezérlőjelek képernyőn és nyomtatón való megjelenítésére van szükségünk.

A számításokat a MINIATUR lebegőpontos aritmetikával végzi, és lehetőségünk van az elemi függvények, pl. a szinusz, a koszinusz és a logaritmus kiszámítására is.

A program futását ciklusokkal, feltételes utasításokkal és ugrásokkal vezérelhetjük.

Ezek elegendőek ahhoz, hogy egy compiler működését megvilágítsák. A nyelvet úgy építettük fel, hogy a felhasználók minden nehézség nélkül továbbfejleszthessék. Aki pl. a lebegőpontos aritmetikát megértette, annak nem lesz nehéz az egészszám-aritmetikát is kidolgozni, mivel ez elvileg már semmiféle új ismeretet nem tételez fel.

A következő fejezetek elolvasása után láthatjuk majd, hogy a MINIATUR nyelven egészen jó programok írhatók.

2.2 A főprogram

A MINIATUR-ban írt programok felépítésére nézzük az első mintapéldánkat!

Mintapélda: **kezdés**

```
100 programm kezdés ist
110 – –
120 – – Itt deklaráljuk később azokat a változókat,
130 – – amelyeket programjaink használni
140 – – fognak
150 beginne
160 – –
170 – – Ide kerülnek a végrehajtandó
180 – – utasítások
190 – –
200 leer.
210 pende kezdés.
```

A MINIATUR-ban ez a legkisebb program. Neve kezdés, ez megjelenik a program elején és a végén. Programunk a *programm* kulcsszóval kezdődik.

Mik a kulcsszavak?

A kulcsszavak olyan szavak, amelyek egy nyelvben különleges szerepet töltenek be, és meghatározott jelentésük van. A felhasználó ezt a jelentést nem tudja megváltoztatni és ezeket a szavakat csak az előre meghatározott jelentésüknek megfelelően szabad használnia. A kulcsszavakkal építjük fel programunk vázát, és ezt közöljük a fordítóval. A *programm* szó itt azt jelenti, hogy egy főprogram kezdődik. A *programm* után következik a program neve, amelyet szabadon választhatunk. A programok nevét programkarakternek hívjuk.

Az alábbiakban felsoroljuk a MINIATUR programnyelv 49 kulcsszavát, amelyeket természetesen a programokban nem fordítottuk le, de a következő táblázatban a kulcsszavak magyar jelentését tájékoztatásul megadtuk.

(0) =	addiere	összeadás
(1) =	ausgabegeraet	kivitel
(2) =	ausgang	kilépés
(3) =	beginne	kezdés

(4) =	bilde	képzés
(5) =	cuspalte	kurzor a megfelelő oszlopba (kurzoroszlop)
(6) =	cuzeile	kurzor a megfelelő sorba (kurzorsor)
(7) =	dann	akkor
(8) =	durch	osztva
(9) =	dividiere	osztás
(10) =	fliess	lebegőpontos változó
(11) =	hintergrund	háttér
(12) =	hole	bevitel
(13) =	ist	van
(14) =	leer	üres
(15) =	mit	szorozva
(16) =	multipliziere	szorzás
(17) =	nach	(-ba, -be) eredménye
(18) =	pende	a program vége
(19) =	potenziere	hatványozás
(20) =	programm	program
(21) =	rahmen	keret
(22) =	rufe	hívás
(23) =	schrift	írás
(24) =	schirmfrei	képernyőtörlés
(25) =	schleife	ciklus
(26) =	sende	ciklus vége
(27) =	sonst	különben
(28) =	*springe*	*ugrás*
(29) =	sprungmarke	ugrási cím
(30) =	subtrahiere	kivonás
(31) =	ueber	az alprogram kezdete
(32) =	uebertrage	értékadás; az értékek átvitele
(33) =	uende	az alprogram vége
(34) =	unterprogramm	alprogram
(35) =	von	-ból, -ből
(36) =	vorschub	eltolás
(37) =	wende	a feltételes ciklus vége
(38) =	wenn	ha

(39) =	zeige	kiírás (szöveg és változó)
(40) =	zeigez	kiírás soremeléssel
(41) =	zeigeas	kiírás ASCII kódban
(42) =	zu	plusz
(43) =	fuer	a paraméteres ciklusutasítás értékadásának kezdete
(44) =	bis	-ig (a paraméteres ciklusutasítás értékadásának vége)
(45) =	wiederhole	a paraméteres ciklusutasítás kezdete
(46) = / =		
(47) = < =		
(48) = > =		

MELYEK A MINIATUR AZONOSÍTÓI?

A felhasználó által szabadon választható azonosítók, pl. a programok, az alprogramok, a ciklusok és a változók stb. nevei.

Az azonosítók maximum 80 karakter hosszúságúak lehetnek, és csak betűket tartalmazhatnak.

A név után az *ist* kulcsszó következik. Az *ist* és a *beginne* kulcsszavak közé helyezzük el a programunkban szereplő változók azonosítóit.

A MINIATUR-ban a megjegyzés sorokat két egymást követő mínuszjellel kezdjük. A megjegyzések egy egész sort foglalnak el. A 110-estől a 140-esig, a 160-astól a 190-esig és a 210-es sorok tehát megjegyzés sorok.

A MINIATUR programokban a sorszámok csak arra valók, hogy a felhasználó jobban eligazodjon, és hogy a programokat a C 64-esbe beépített editorral (szerkesztő) hozhassa létre. Később a program fordításánál a hibás sorok sorszáma megjelenik, és így az jobban követhető.

A MINIATUR programok írásához kis- és nagybetűket egyaránt használhatunk. A kulcsszavakat és az azonosítókat mindig kisbetűkkel írjuk.

A *beginne* és a *pende* kulcsszavak közé a végrehajtandó utasításokat fogjuk majd írni. Egy logikailag összetartozó utasítássort blokknak nevezünk. A blokknak legalább egy utasítást tartalmaznia kell. Ha még nem tudjuk, hogy egy adott blokk milyen utasításokból fog állni, akkor üres utasítást iktatunk be. Az üres utasítást a MINIATUR-ban *leer*-nek nevezzük.

A MINIATUR-ban az utasításokat, a programokat és az alprogramokat ponttal zárjuk le.

A *pende* után még egyszer ki kell írni a program nevét és utána pontot kell tenni.

A kezdés nevű program megfelel a MINIATUR programok feltételeinek. Ha lefordítanánk gépi nyelvre, hatására nem történne semmi.

Még egyszer felhívjuk a figyelmet arra, hogy a MINIATUR programokban a sorszámoknak nincs jelentősége. Az előző programot például a következőképpen is írhattuk volna:

```
1 kezdés programm ist beginne leer pende kezdés
```

Ez természetesen nem szerencsés felírás, mivel így egyáltalán nem látható a program szerkezete.

2.3 A szövegkiírás

Szövegek kiírásához a programok eredményeinek megfelelő formában való megjelenítésére van szükség.

A MINIATUR-ban erre két lehetőség van:

zeige "karaktersor".
zeigez "karaktersor".

A karaktersort a megfelelő billentyűk lenyomásával állítjuk elő. A karaktersor előtt és után idézőjel áll.

Ezeket a parancsokat ponttal zárjuk le. Az utasításokat a *zeige* vagy a *zeigez* kulcsszavakkal kezdjük. A karaktersor mindkét parancs esetén a pillanatnyi kurzor állásnál jelenik meg. A *zeige* parancs alkalmazása után a kurzor a következő mező elejére, a *zeigez* parancs hatására pedig a következő sor elejére kerül; azaz egy sort is emel. Alapesetben a kivitel a képernyőre történik. A nyomtatóra való kivitelt egy későbbi fejezetben ismertetjük.

Nézzünk egy példát a szövegek kiírására!

Mintapélda: **kiírás**

```
100 programm kiírás ist
110 – –
120 – –
130 beginne
140 – –
150 zeigez "mintapélda".
160 zeige "szövegek".
170 zeige "kiírására".
180 – –
190 pende kiírás.
```

A program hatására megjelenik a mintapélda, a következő sorban pedig közvetlenül egymás után a szövegek és a kiírására karaktersorozat.

2.4 A kiírás vezérlése

A kiírást szeretnénk érzékenyebben vezérelni, mint ahogy azt a *zeige* és a *zeigez* utasítás lehetővé teszi. A kurzort a képernyő tetszőleges helyére akarjuk állítani, a képernyőt törölni szeretnénk, vagy tetszőleges sortávolságot akarunk beállítani.

A
vorschub

paranccsal végezhetjük el egy sor eltolását. Ha az utolsó sorban *zeige* paranccsal vittünk ki, akkor a *vorschub* hatására a kivitel egy új, következő sorban megy végbe. Ha előzőleg *zeigez* paranccsal vittünk ki, akkor a *vorschub* hatására üres sor jön létre.

A képernyőt

schirmfrei

paranccsal törölhetjük.

A kurzort
cuspalte kurzoroszlop.

paranccsal vihetjük tetszőleges helyre. Az oszlop szó helyére az 1-es és a 40-es közötti egész számot kell írni. A kurzort az alábbi utasítással helyezhetjük a 25. oszlopba:

cuspalte 25.
A kurzort
cuzeile kurzorsor.

paranccsal vihetjük adott sorba. A sor értéke az 1-es és a 24-es közé eső egész szám. A *cuzeile* 15. parancs hatására tehát a kurzor a 15. sorba kerül.

Vezérlőkódok segítségével a képernyőn a kiírást tetszőlegesen szabályozhatjuk. A kurzort pl. eggyel jobbra vihetjük, ha a 29-esnek megfelelő ASCII kódot használjuk. Ez a

zeigeas ASCII kód.

parancs hatására megy végbe. Az ASCII kódnak a 0 és a 255-ös közé kell esnie. Az egyes kódok jelentését a számítógép kézikönyvében megtalálhatjuk. Néhány fontosabbat felsorolunk közülük.

3 stop
10 soremelés
17 kurzor le
19 kurzor a bal felső sarokba
20 sortörlés
29 kurzor jobbra
32 szóköz
145 kurzor fel
147 képernyőtörlés
148 egy karakter beszúrása
157 kurzor balra

Mintapélda: **kiírásb**

```
1000 programm kiírásb ist
1010 – –
1020 – –
1030 – – Ez a program törli a képernyőt, a
1040 – – kurzort az 5. sor 5. oszlopára
1050 – – állítja,
1060 – – kiírja az "írás a képernyőre" mondatot,
1070 – – két üres sort hagy
1080 – – és még egyszer kiírja az előbbi mondatot.
1090 – –
1100 – –
1110 beginne
1120 – –
1130 schirmfrei.
1140 – –
1150 cuzeile 5. cuspalte 5.
1160 – –
1170 zeigeas 18.
1180 – –
1190 zeige "írás a képernyőre".
1200 – –
1210 zeigeas 146.
1220 – –
1230 vorschub. vorschub. vorschub.
1240 – –
1250 zeigez "írás a képernyőre".
1260 – –
1270 – –
1280 pende kiírásb.
```

2.5 A kivitel a nyomtatóra

Ha egy program listáját papíron akarjuk megőrizni, akkor lehetőségünk van arra, hogy a képernyő helyett a nyomtatót használjuk. Ilyenkor az

ausgabegeraet nyomtató.

utasítást kell adni.
Ha a kivitelt ismét a képernyőre akarjuk irányítani, akkor a

ausgabegeraet képernyő.

utasítást adjuk.
Arra kell azonban vigyáznunk a kivitel vezérlésénél, hogy kétszer egymás után ne nevezzük meg ugyanazt a perifériát, mert egy adatátviteli csatornát ebben a formában csak egyszer nyithatunk meg.

Mintapélda: **nyomtató**

```
1000 programm nyomtató ist
1010 - -
1020 - -
1030 - - Ez a program a nyomtatóra írja a követ-
1040 - - kező mondatot "most nyomtatunk",
1050 - - ezután a képernyőre írja az
1060 - - "ismét a képernyőre írunk" mondatot.
1070 - -
1080 - -
1090 beginne
1100 - -
1110 ausgabegeraet nyomtató.
1120 - -
1130 zeigez "most nyomtatunk".
1140 - -
1150 ausgabegeraet képernyő.
1160 - -
1170 - -
1180 zeigez "ismét a képernyőre írunk".
1190 - -
1200 - -
1210 pende nyomtató.
```

2.6 A képernyő színének megválasztása

A képernyő keretét, a hátteret és az írás színét a lehetőségeken belül tetszés szerint választhatjuk.

A keret színének meghatározására a következő parancsokat használhatjuk:

rahmen fekete.
rahmen fehér.
rahmen türkiz.
rahmen piros.
rahmen lila.
rahmen kék.
rahmen sárga.
rahmen narancs.
rahmen barna.
stb.

A háttér színét a következő parancsokkal választhatjuk meg:

hintergrund fekete.
hintergrund fehér.
hintergrund türkiz.
hintergrund piros.
hintergrund lila.
hintergrund kék.
hintergrund sárga.
hintergrund narancs.
hintergrund barna.
stb.

Az írás színének megválasztása pedig az alábbi parancsokkal hajtható végre:

schrift fekete.
schrift fehér.
schrift piros.
schrift zöld.
schrift kék.
schrift sárga.
stb.

Mintapélda: **szín**

```
100 programm szín ist
110 – –
120 – –
130 – – Ez a program az alábbi színeket jelöli ki:
140 – – a háttér fekete,
150 – – a keret fehér,
160 – – az írás fehér.
170 – –
180 – – A 12-es sor 17. oszlopába
190 – – kiírjuk a "szín" szót,
200 – – a háttér színét fehérre,
210 – – a keret színét feketére változtatjuk.
220 – –
230 – –
240 beginne
250 – –
260 schirmfrei.
270 – –
280 hintergrund fekete.
290 rahmen fehér.
300 schrift fehér.
310 – –
320 cuzeile 12. cuspalte 17.
330 zeigeas 18. zeigez "szín". zeigeas 146.
340 – –
350 hintergrund fehér.
370 rahmen fekete.
380 – –
390 pende szín.
```

2.7 A változók meghatározása

A programban szereplő változókat a program elején meg kell határozni. Fordítóprogramunkban lebegőpontos aritmetikát használunk. Ezért a változókat csak ennek megfelelően határozhatjuk meg.

A lebegőpontos változók + −1.70141183E+38 és + −2.93873588E−39 közötti értékeket vehetnek fel.

A meghatározás kétféleképpen lehetséges.

1) Egy lebegőpontos változó meghatározásánál:

fliess változónév.

2) Több lebegőpontos változó meghatározása esetén:

fliess változónév 1, változónév 2, ...

Mintapélda: **definíció**

```
100 programm definíció ist
110 – –
120 – –
130 – – otto, enno, benno, változókat
140 – – lebegőpontos változóként határozzuk meg.
150 – –
160 – –
170 fliess otto.
180 fliess benno,enno.
190 – –
200 – –
210 beginne
220 – –
230 leer.
240 – –
250 pende definíció.
```

2.8 Az adatok be- és kivitele

A változók meghatározása után ismerkedjünk meg a változók értékének be- és kivitelével.

Kivitelre a már ismert két parancsot: a *zeige* és a *zeigez* parancsokat használjuk, azzal a különbséggel, hogy a változónevet nem kell idézőjelbe tenni.

zeige változónév.
zeigez változónév.

A fenti két parancs között a különbség az, hogy a *zeigez* alkalmazása esetén a kivitel egy új, a következő sor elejére kerül.

A bevitelt a

hole változónév.

paranccsal végezzük.

Az adatokat billentyűzetről visszük be. A *hole* parancs hatására egy kérdőjel jelenik meg a képernyőn, és beírhatunk egy lebegőpontos számot.
A bevitelt a RETURN billentyű lenyomásával fejezzük be.

Mintapélda: **adatbevitel**

```
100 programm adatbevitel ist
110 - -
120 - -
130 - - Ez a program megkérdezi az ön életkorát
140 - - és kiírja azt.
150 - -
160 - -
170 fliess kora.
180 - -
200 beginne
210 - -
220 schirmfrei.
230 - -
240 zeige "kora:".
250 hole kora. vorschub.
260 zeige "Ön".
270 zeige "kora".
280 zeige "éves".
290 - -
300 - -
310 pende be.
```

2.9 Az értékek megváltoztatása

Most már meg tudjuk határozni a változókat, amelyeknek a billentyűzetről értékeket adhatunk. Ezeket az értékeket ki is vihetjük. Már csak az a lehetőségünk hiányzik, hogy az értékeket egy programon belül a beviteltől függetlenül megváltoztathassuk.

Nagyon egyszerű szerkezeteket fogunk használni, amelyek a compiler program során jól követhetőek lesznek. A bővítésnek ezek után már semmi akadálya. Először csak pozitív értékek adását (hozzárendelését) engedjük meg.

Adjunk a változónak lebegőpontos értéket:

uebertrage lebegőpontos szám *nach* változónév.

uebertrage 3.4e+17 *nach* otto.

Az utasítás végrehajtása után az otto változó értéke 3.4e+17 lesz.

Vigyük át az egyik változó értékét egy másik változóba:

uebertrage változónév *nach* változónév.

uebertrage otto *nach* enno.

Fenti utasítás után az enno változó értéke azonos lesz az otto változó értékével.

Két változó értékeinek összeadása:

addiere változónév *zu* változónév *nach* változónév.

addiere otto *zu* enno *nach* otto.

Eszerint otto új értéke otto és enno értékének összegével lesz egyenlő.

Egyik változó kivonása a másikból:

subtrahiere változónév *von* változónév *nach* változónév.

subtrahiere otto *von* benno *nach* enno.

Enno értékét tehát úgy kapjuk meg, hogy a benno értékéből az otto értékét kivonjuk.

Két változó szorzása:

multipliziere változónév *mit* változónév *nach* változónév.

multipliziere benno *mit* enno *nach* enno.

Benno értékét enno értékével szorozzuk és ezt az értéket az enno-ban helyezzük el.

Változók osztása:

dividiere változónév *durch* változónév *nach* változónév.

dividiere otto *durch* benno *nach* enno.

Tehát enno értékét otto és benno értékének hányadosa adja meg.

Két változó hatványozása:

potenziere változónév *mit* változónév *nach* változónév.

potenziere benno *mit* enno *nach* otto.

Eszerint otto értéke nem lesz más, mint benno értékének az enno-dik hatványa.

Mintapélda: **aritmetika**

```
1000 programm aritmetika ist
1010 - -
1020 - -
1030 - - Ez a program a lebegőpontos aritmeti-
1040 - - kát mutatja be.
1050 - -
1060 - -
1070 - - fliess monika, tamás, cecília.
1080 - - fliess benno, enno, otto.
1090 - -
1100 - -
1110 beginne
1120 - -
1130 - -
1140 schirmfrei. vorschub.
```

```
1150 zeigez "a lebegőpontos aritmetika bemutatása".
1160 vorschub.
1170 - -
1180 uebertrage kettő nach monika.
1190 uebertrage öt nach tamás.
1200 uebertrage tamás nach cecília.
1210 - -
1220 - -
1230 zeige "monika =". zeigez monika.
1240 zeige "tamás =". zeigez tamás.
1250 zeige "cecilia =". zeigez cecília.
1260 - -
1270 - -
1280 addiere monika zu tamás nach benno.
1290 vorschub.
1300 zeige "2 + 5 =". zeigez benno.
1310 - -
1320 subtrahiere cecilia von benno nach enno.
1330 zeige "7 - 2 =". zeigez enno.
1340 - -
1350 multipliziere tamás mit monika nach otto.
1360 zeige "5 * 2 =". zeigez otto.
1370 - -
1380 dividiere cecilia durch monika nach enno.
1390 zeige "5 / 2 =". zeigez enno.
1400 - -
1410 potenziere monika mit tamás nach benno.
1420 zeige "2 ↑ 5 =". zeigez benno.
1430 - -
1440 - -
1450 pende aritmetika.
```

2.10 A függvények

Ebben a fejezetben azokat a numerikus függvényeket ismertetjük, amelyeket a compilerbe be akarunk építeni.

Az utasítás általános alakja:

bilde függvényazonosító *von* változónév *nach* változónév.

vagy:

bilde függvényazonosító *von* változónév.

A két utasítás között az a különbség, hogy az első esetben a kiszámított értéket ahhoz a változóhoz rendeljük hozzá, amelynek neve a *nach* kulcsszó mögött áll, míg a második esetben a kiszámított értéket az eredeti változóhoz, vagyis ahhoz rendeljük hozzá, amelyből a függvény értékét kiszámítottuk.

A következőkben a függvény szó után megadjuk a függvények azonosítóit, majd pedig röviden megmagyarázzuk az utasítást. Javasoljuk az olvasónak, hogy lapozzon a függvények mintapéldához, és olvassa a következőket azzal párhuzamosan, ott minden függvényre példát is talál.

Függvény: *absolut* (abszolút érték)

A mennyiség abszolút értékét képezi.

Függvény: *actangens* (árkusz tangens)

Az érték arkusz tangensét számítja ki. Az értéket ívmértékben (radiánban) kell megadni.

Függvény: *cosinus* (koszinusz)

Az ívmértékben megadott érték koszinuszát számítja.

Függvény: *exponent* (exponenciális)

Az utasítás az *e* ($e = 2.718\,271\,83$) értéknek a változó értékére emelt hatványát adja meg.

Függvény: *integer* (egészre kerekítés)

Ezt a függvényutasítást akkor használjuk, ha egy lebegőpontos számot kerekített egész számmá akarunk átalakítani. Így pl. *integer* 3.45 egyenlő 3-mal és *integer* 4.6 egyenlő 5-tel.

Függvény: *logarithmus* (logaritmus)

Az utasítás a változó értékének természetes (*e* alapú) logaritmusát képezi.

Függvény: *speicherwert* (tárérték)

Ebben a függvényutasításban a változó egy tárhely címét adja meg, és az ezen helyen tárolt értéket a program kiolvassa.

Függvény: *zufall* (véletlenszám-sorozat)

Ez a függvényutasítás 0 és 1 közötti véletlen számokat generál. A véletlen számok a változók értékétől függően képződnek. Ha a változó értéke negatív, akkor azonos negatív érték esetén mindig ugyanazok a véletlenszám-sorozatok képződnek. Ha a változó értéke nulla vagy annál nagyobb, akkor mindig új számok képződnek.

Függvény: *vorzeichen* (előjel)

A függvény a következő értékeket adja:

−1, ha a változó értéke nullánál kisebb;
0, ha a változó értéke nullával egyenlő;
+1, ha a változó értéke nullánál nagyobb.

Függvény: *sinus* (szinusz)

A függvényutasítás az ívmértékben megadott argumentum szinuszát számolja.

Függvény: *quadratwurzel* (négyzetgyök)

Az argumentum négyzetgyökét számítja ki. A beadott értéknek pozitívnak kell lennie.

Függvény: *tangens* (tangens)

Az ívmértékben megadott érték tangensét számítja ki.

Mintapélda: **függvények**

```
1000 programm függvények ist
1010 - -
1020 - -
1030 - - Ez a program a MINIATUR programnyelv
1040 - - függvényeinek
1050 - - használatát mutatja be.
1060 - -
1070 - -
1080 fliess a, b, c, d, e, f, g.
1090 fliess pinegyed.
1100 - -
1110 beginne
1120 - -
1130 uebertrage 4 nach a.
1140 uebertrage 4 nach b.
1150 uebertrage 3.1415 nach c.
1160 dividiere c durch b nach pinegyed.
1170 - -
1180 schirmfrei.
1190 zeigez "függvények".
1200 - -
1210 vorschub.
1220 bilde integer von c nach d.
1230 - -
1240 - - c kerekített egész szám értékét kiszámí-
1250 - - tottuk és d-ben tároltuk.
1260 - - a nem változott.
1270 - -
1280 zeige "c=". zeigez c.
1290 zeige "d=". zeigez d.
1300 - -
1310 bilde absolut von a.
1320 - -
1330 - - E formánál "a" tartalma
1340 - - megváltozott.
1350 - -
1360 zeige "abszolút érték 4 =". zeige a.
1370 - -
1380 - -
1390 bilde actangens von pinegyed nach e.
1400 zeige "árkusz tangens pinegyed =", zeigez e.
1410 - -
1420 bilde cosinus von pinegyed nach e.
```

```
1430 zeige "koszinusz pinegyed =". zeigez e.
1440 --
1450 bilde exponent von b nach e.
1460 bilde "exponenciális 4 =". zeigez e.
1470 --
1480 bilde integer pinegyed nach e.
1490 zeige "egész részre kerekítés pinegyed =". zeigez e.
1500 --
1510 bilde logarithmus von b nach e.
1520 zeige "logaritmus 4 =". zeigez e.
1530 --
1540 bilde speicherwert von b nach f.
1550 zeige "4. tárhely tartalma =". zeigez f.
1560 --
1570 bilde zufall von a nach g.
1580 zeige "véletlen szám a =". zeigez g.
1590 --
1600 bilde vorzeichen von b nach g.
1610 zeige "előjel 4 =". zeigez g.
1620 --
1630 bilde sinus von pinegyed nach f.
1640 zeige "szinusz pinegyed =". zeigez f.
1650 --
1660 bilde quadratwurzel von b nach g.
1670 zeige "négyzetgyök 4 =". zeigez g.
1680 --
1690 bilde tangens von pinegyed nach f.
1700 zeige "tangens pinegyed =". zeigez f.
1710 --
1720 zeigez "jó?".
1730 --
1740 --
1750 pende függvények.
```

2.11 A feltételes szerkezetek

Eddig nem volt szó olyan lehetőségekről, hogy miképp lehet programunkban az egyes utasításokat átugrani. Most azt szeretnénk ismertetni, hogyan dönthetjük el az egyes változók értékének függvényében, hogy mely utasításokat kell végrehajtani. Ehhez egy programrészletet két ágra bontunk, és a két ág egyikét vagy másikát hajtjuk végre aszerint, hogy a változók aktuális értékei valamely meghatározott összehasonlító feltételnek eleget tesznek-e vagy sem.

Formálisan a következőkről van szó:

```
wenn feltétel leírása
dann
- -
- -
- - utasításblokk 1
- -
- -
sonst
- -
- -
- - utasításblokk 2
- -
- -
wende.
```

A feltétel felépítése a következő:

változónév operátor változónév

Operátorként a következőket használhatjuk:

=	egyenlő
/=	nem egyenlő
<	kisebb
<=	kisebb egyenlő
>	nagyobb
>=	nagyobb egyenlő

Lássunk mindjárt egy példát:

```
wenn otto = enno
dann
zeigez "otto egyenlő enno".
- -
sonst
- -
zeigez "otto nem egyenlő enno".
- -
wende.
```

Ebben a példában otto és enno értékét hasonlítjuk össze. Ha a két érték egyenlő, akkor a képernyőn az otto egyenlő enno, ha a két érték nem egyenlő, akkor az otto nem egyenlő enno szöveg jelenik meg. A program ezután a *wende* kulcsszó utáni utasításnál folytatódik.

A *wenn dann sonst wende* utasítássorozat így két utasításblokkot tartalmaz, ill. köt össze. Mindkét blokk utasítások lezárt sora. A MINIATUR-ban minden blokknak legalább egy utasítást tartalmaznia kell. A blokkokat összekötő utasításnak megvan a maga bemenete és kimenete. A *wenn dann sonst wende* utasítássorozat bemenete, azaz nyitása a *wenn*, és *kimenete*, azaz lezárása a *wende*.

A blokkokba szervezett nyelvek egyik előnye az, hogy a logikailag összetartozó utasításokat gyorsan fel lehet ismerni.

Minthogy célunk az, hogy egy compiler programot megértsünk, ill. megírjunk, nagyon fontos, hogy kellő számú mintapéldánk legyen. Ezek a fordító lehetőségeit bemutatják és így megérthetjük azt is, hogy mit csinál a compiler egy utasítás hatására. A könyvben közölt mintapéldák tehát egyúttal tesztprogramok is, amelyeket elsőként kell lefordítani, miután compilerünket elkészítettük. Elvben végtelen sok program írható a MINIATUR programnyelven, így érthető, hogy csak a legfontosabb lehetőségeket tesztelhetjük. Jogos az a remény, hogy a compiler akkor is hibátlanul működik, ha pl. két utasítást felcserélünk. Amint az eddigi mintapéldákon észrevehettük, a tesztprogramokat is úgy kell megírni, hogy a hibák a program végrehajtásakor előjöjjenek.

Mutassunk be erre egy példát!

A programnak azt kell eldöntenie, hogy a beadott szám páros vagy páratlan-e. Ennek megfelelően vagy a szám páros, vagy a szám páratlan kiírásnak kell megjelennie a képernyőn.

Annak eldöntésére, hogy a program fordítása helyes és az hibátlanul is működik, jó lehetőség pl. a számnak és a következő mondatnak a kiírása:

zeige "szám". *zeigez* "a szám páros.".

vagy

zeige "szám". *zeigez* "a szám páratlan.".

Mintapélda: **páros vagy páratlan**

```
1000 programm páros vagy páratlan ist
1010 - -
1020 - -
1030 - - Ez a program a billentyűzetről
1040 - - egy számot kér és
1050 - - eldönti, hogy a szám páros
1060 - - vagy páratlan-e.
1070 - -
1080 - -
1090 fliess szám, pót, kettő, nulla, pótv.
1100 - -
1110 beginne
1120 - -
1130 schirmfrei. vorschub. uebertrage 2 nach 2.
1140 - -
1150 zeige "kérek egy egész számot.".
1160 hole szám. vorschub.
1170 - -
1180 - -
1190 uebertrage szám nach pót.
1200 dividiere pót durch kettő nach pót.
1210 bilde integer von pót.
1220 dividiere szám durch kettő nach pótv.
1230 subtrahiere pót von pótv. nach pót.
1240 - -
1250 uebertrage 0.0 nach nulla.
1260 - -
1270 wenn pót = nulla dann
1280 - -
1290 zeige "szám". zeigez "páros.".
1300 - -
1310 sonst
1320 - -
1330 zeige "szám". zeigez "páratlan.".
1340 - -
1350 wende.
1360 - -
1370 pende páros vagy páratlan.
```

Mintapélda: **választás**

```
1000 programm választás ist
1010 – –
1020 – –
1030 – – Ez a program adott feltételtől
1040 – – függően a képernyőre vagy a nyomtatóra
1050 – – egy mondatot ír ki.
1060 – –
1070 – –
1080 fliess teszt, egy.
1090 – –
1100 beginne
1120 schirmfrei. vorschub.
1130 – –
1140 zeige "nyomtató (1) / képernyő (2)?".
1150 hole teszt.
1160 vorschub.
1170 uebertrage 1.0 nach egy.
1180 – –
1190 wenn teszt = egy dann
1200 – –
1210 ausgabegeraet nyomtató.
1220 zeigez "nyomtató.".
1230 ausgabegeraet képernyő.
1240 – –
1250 sonst
1260 – –
1270 zeigez "képernyő.".
1280 – –
1290 wende.
1300 – –
1310 pende választás.
```

Mintapélda: **döntésteszt**

```
1000 programm döntésteszt ist
1010 – –
1020 – –
1030 – – Ez a program
1040 – – a különféle döntési lehetőségeket
1050 – – teszteli.
1060 – –
1070 – –
1080 fliess három, négy.
```

```
1090 - -
1100 beginne
1110 - -
1120 schirmfrei. vorschub.
1130 - -
1140 uebertrage 3.0 nach három.
1150 uebertrage 4.0 nach négy.
1160 - -
1170 - -
1180 zeigez "döntések:".
1190 - -
1200 - -
1210 wenn három = négy dann
1220 - -
1230 zeigez "=hiba!".
1240 - -
1250 sonst
1260 - -
1270 zeigez "=nincs hiba!".
1280 - -
1290 wenn három > négy dann
1300 - -
1310 zeigez "hiba >-nél!".
1320 - -
1330 sonst
1340 - -
1350 zeigez "nincs hiba!".
1360 - -
1370 wenn három >= négy dann
1380 - -
1390 zeigez ">= hiba!".
1400 - -
1410 sonst
1420 - -
1430 zeigez ">= nincs hiba!".
1440 - -
1450 wenn három /= négy dann
1460 - -
1470 zeigez "/= nincs hiba!".
1480 - -
1490 wenn három < négy dann
1500 - -
1510 zeigez "nincs hiba <-nél!".
1520 - -
1530 wenn három <= négy dann
```

```
1540 – –
1550 zeigez "< = nincs hiba!".
1560 – –
1570 sonst
1580 – –
1590 zeigez "= hiba!".
1600 – –
1610 wende.
1620 – –
1630 sonst
1640 – –
1650 zeigez "< hiba!".
1660 – –
1670 wende.
1680 – –
1682 sonst
1684 – –
1690 zeigez "/= hiba!".
1700 – –
1710 wende.
1720 wende.
1730 wende.
1740 wende.
1750 – –
1760 pende döntésteszt.
```

2.12 A ciklusok

Ahhoz, hogy egy utasításblokkon többször végig tudjunk futni, ciklusok szervezésére van szükségünk.

A MINIATUR-ban ehhez két utasítástípus van:

1) végtelen ciklus kilépési utasítással;
2) paraméteres ciklusutasítás.

1) A végtelen ciklus

schleife ciklusváltozó ueber
– –
– –
sende ciklusváltozó.

A végtelen ciklus egyik változata a következő:

schleife ciklusváltozó
– –
– –
– – utasításblokk
– –
– –
– – sende ciklusváltozó.

Ez a legegyszerűbb ciklus, mégis ezt használjuk legritkábban.

Ha egyszer bekerülünk ebbe a ciklusba, akkor a megadott utasítássort – a ciklus magját – addig hajtja végig a program, amíg csak a számítógépet ki nem kapcsoljuk. Ennek alkalmazása lehetne pl. az, hogy a számítógép bekapcsolásakor elkezdődik a végtelen ciklus, a gép parancsot vár a kezelőjétől, majd a parancs végrehajtása után a gép ismét visszatér kiindulási pontjához. Az ilyen ciklus a számítógépben található legmagasabbrendű műveleti sor és minden egyéb műveleti sor ennek a ciklusnak rendelődik alá.

Ebből a ciklusból azonban nem tudunk kijutni.

Másik példa a végtelen ciklus alkalmazására.

A gépnek az a feladata, hogy egymás után minden pozitív egész számot írjon ki.

Mintapélda: **végtelen**

```
100 programm végtelen ist
110 – –
120 – –
130 – – Ez a program
140 – – minden pozitív egész számot
150 – – kiír.
160 – –
170 – –
180 fliess szám, egy.
190 – –
200 beginne
210 – –
220 uebertrage 1.0 nach egy.
230 – –
240 uebertrage 0.0 nach szám.
250 – –
260 schleife egész ueber
270 – –
280 zeigez "szám".
290 – –
300 addiere egy zu szám nach szám.
310 – –
320 sende egész.
330 – –
340 pende végtelen.
```

Ahhoz, hogy a program csak az első 100 pozitív egész számot írja ki, ki kell léptetnünk a végtelen ciklusból, mégpedig akkor, amikor a kiírást abba kell hagynia. A kilépés utasítás felépítése a következő:

ausgang ciklusváltozó *wenn* feltétel.

A feltétel felépítését az előző fejezetben már ismertettük, tehát:

változónév operátor változónév

Operátorként *A feltételes szerkezetek* c. fejezetben leírtak használhatóak.

Ha a feltétel teljesül, akkor a program a megadott nevű ciklus végére ugrik, vagyis a program ezután azt az utasítást hajtja végre – ha van ilyen –, amelyik a ciklus után következik. Nézzük meg az előbbi példát ezzel a kiegészítéssel!

Mintapélda: **száz**

```
100 programm száz ist
110 - -
120 - -
130 - - Ez a program minden
140 - - pozitív egész számot
150 - - kiír százig.
160 - -
170 - -
180 fliess szám, egy, száz.
190 - -
200 beginne
210 - -
220 uebertrage 1.0 nach egy.
230 - -
240 uebertrage 0.0 nach szám.
250 - -
260 uebertrage 100.0 nach száz.
280 schleife egész ueber
290 - -
300 zeigez "szám".
310 - -
320 addiere egy zu szám nach szám.
330 - -
340 ausgang egész wenn száz=szám.
350 - -
360 sende egész.
370 - -
380 pende száz.
```

Minden ciklusnak külön nevet kell adni. Ennek előnyei a következők:

- a program szerkezete világosabbá válik;
- egymásba szerkesztett ciklusok áttekinthetőbbek lesznek;
- a kilépési utasítással az egymásba szerkesztett ciklusokból ki is lehet lépni.

Mintapélda: **kilépés**

```
100 programm kilépés ist
110 - -
120 - -
130 - - Ez a program az egymásba
```

```
140 – – szerkesztett ciklusból való
150 – – kilépést teszteli.
160 – –
170 – –
180 fliess gudrun, enno, egy.
190 – –
200 beginne
210 – –
220 uebertrage 1.0 nach egy.
230 uebertrage 0.0 nach enno.
240 uebertrage 100.0 nach gudrun.
250 – –
260 – –
270 schleife kívül ueber
280 – –
290 – –
300 schleife belül ueber
310 – –
320 – –
330 zeigez "enno".
340 – –
350 ausgang kívül wenn enno = gudrun.
360 – –
370 addiere egy zu enno nach enno.
380 – –
390 – –
400 sende belül.
410 – –
420 sende kívül.
430 – –
440 sende kilépés.
```

Itt tehát mind a belső, mind a külső ciklusból kilépünk, ha az enno=gudrun feltétel teljesül.

Egy rövid programot közlünk még, a kilépési utasítás vizsgálatára.

Mintapélda: **kilépésteszt**

```
1000 programm kilépésteszt ist
1010 – –
1020 – –
1030 – – E program a
1040 – – kilépési utasítást teszteli.
```

```
1050 – –
1060 – –
1070 fliess gudrun, enno, monika.
1080 – –
1090 beginne
1100 – –
1105 schirmfrei. vorschub.
1110 zeigez "teszt".
1120 vorschub.
1130 – –
1140 uebertrage 2.0 nach gudrun.
1150 uebertrage 2.0 nach monika.
1160 uebertrage 7.0 nach enno.
1170 – –
1180 – –
1190 schleife ciklusa ueber
1200 – –
1210 schleife ciklusb ueber
1220 – –
1230 schleife ciklusc ueber
1240 – –
1250 schleife ciklusd ueber
1260 – –
1270 schleife ciklυse ueber
1280 – –
1290 schleife ciklusf ueber
1300 – –
1310 schleife ciklusg ueber
1320 – –
1330 schleife ciklush ueber
1340 – –
1350 – –
1360 ausgang ciklush wenn gudrun /= enno.
1370 – –
1380 zeigez "kilépés ciklush hibás.".
1390 – –
1400 sende ciklush.
1410 – –
1420 zeigez "kilépés ciklush rendben.".
1430 – –
1440 ausgang ciklusg wenn gudrun < enno.
1450 – –
1460 zeigez "kilépés ciklusg hibás.".
1470 – –
1480 sende ciklusg.
```

```
1490 - -
1500 zeigez "kilépés ciklusg rendben.".
1510 - -
1520 ausgang ciklusf ha gudrun < = enno.
1530 - -
1540 zeigez "kilépés ciklusf hibás.".
1550 - -
1560 sende ciklusf.
1570 - -
1580 zeigez "kilépés ciklusf rendben.".
1590 - -
1600 ausgang cikluse ha gudrun < = monika.
1610 - -
1620 zeigez "kilépés cikluse hibás.".
1630 - -
1640 sende cikluse.
1650 - -
1660 zeigez "kilépés cikluse rendben.".
1670 - -
1680 ausgang ciklusd ha enno > monika.
1690 - -
1700 zeigez "kilépés ciklusd hibás.".
1710 - -
1720 sende ciklusd.
1730 - -
1740 zeigez "kilépés ciklusd rendben.".
1750 - -
1760 ausgang ciklusc ha enno > = monika..
1770 - -
1780 zeigez "kilépés ciklusc hibás.".
1790 - -
1800 sende ciklusc.
1810 - -
1820 zeigez "kilépés ciklusc rendben.".
1830 - -
1840 ausgang ciklusb ha gudrun > = monika.
1850 - -
1860 zeigez "kilépés ciklusb hibás.".
1870 - -
1880 sende ciklusb.
1890 - -
1900 zeigez "kilépés ciklusb rendben.".
1910 - -
1920 ausgang ciklusa ha gudrun = monika.
1930 - -
```

```
1940 zeigez "kilépés ciklusa hibás.".
1950 - -
1960 sende ciklusa.
1970 - -
1980 zeigez "kilépés ciklusa rendben.".
1990 - -
2000 zeigez "vége.".
2010 - -
2020 pende kilépésteszt.
```

A tesztprogramok nagyon könnyen hosszúvá válhatnak; valójában ez a program túlságosan is rövid. Nem próbáltuk ki pl. még azt sem, hogy a ciklus összefér-e bizonyos feltételes szerkezetekkel.

2) A paraméteres ciklusutasítás

Ha tudjuk, hogy egy bizonyos utasítássorozatnak hányszor kell lefutnia, akkor ezt a ciklusutasítást kell választanunk.

Az utasítás általános formája a következő:

```
fuer változónév von változónév bis változónév
wiederhole
- -
- -
- -
sende.
```

Mintapélda: **számlálás**

```
100 programm számlálás ist
110 - -
120 - -
130 - - Ez a program 1-től 10-ig
140 - - kiírja a számokat.
150 - -
160 - -
170 fliess index, alsó határ, felső határ.
180 - -
190 beginne
200 - -
210 uebertrage 1.0 nach alsó határ.
220 uebertrage 10.0 nach felső határ.
```

```
230 – –
240 fuer index von alsó határ bis felső határ wiederhole
250 – –
260 zeigez „index".
270 – –
280 sende.
290 – –
300 pende számlálás.
```

Mintapélda: **partest**
(Tesztprogram egymásba szerkesztett ciklusokra.)

```
1000 programm partest ist
1010 – –
1020 – –
1030 – – Ez a program paraméteres egymásba szerkesztett ciklusokat tesztel.
1040 – –
1050 – – Az eredmény 1000 legyen.
1060 – –
1070 – –
1080 fliess indexa, indexb, indexc.
1090 fliess alsó határ, felső határ, szám, egy.
1100 – –
1110 beginne
1120 – –
1130 uebertrage 1.0 nach alsó határ.
1140 uebertrage 10.0 nach felső határ.
1150 uebertrage 0.0 nach szám.
1155 uebertrage 1.0 nach egy.
1160 – –
1170 – –
1180 fuer indexa von alsó határ bis felső határ wiederhole
1190 – –
1200 fuer indexb von alsó határ bis felső határ wiederhole
1210 – –
1220 fuer indexc von alsó határ bis felső határ wiederhole
1230 – –
1240 addiere egy zu szám nach szám.
1250 – –
1260 sende.
1270 sende.
1280 sende.
1290 – –
1300 zeigez „eredmény:". zeigez „szám."
1310 – –
1320 pende partest.
```

2.13 Az átirányító (ugró) utasítások

Egy programon belül szükségünk lehet arra is, hogy a program meghatározott helyen folytatódjék. Ezt ugró utasítással érhetjük el, pl. a BASIC-ben ilyen a GOTO utasítás. A GOTO 100 azt jelenti, hogy a programot a 100-as sorszámú utasítástól kell folytatni. A MINIATUR-ban azonban a sorszámozásnak nincs jelentősége, így valami mást kell kitalálnunk. Ez pedig:

sprungmarke név.

A következő utasítással érhetjük el, hogy a program ezen a helyen folytatódjék:

springe név.

Itt a név olyan azonosító, amely a programban egyébként nem fordul elő.

Mintapélda: **ugrás**

```
100 programm ugrás ist
110 - -
120 - -
130 - - Ezt a programot egy 1-es
140 - - bevitele leállítja.
150 - -
160 - -
170 fliess egy, teszt
180 - -
190 beginne
200 - -
210 uebertrage 1.0 nach egy.
220 - -
230 sprungmarke kezdet.
240 - -
250 hole teszt. vorschub.
260 - -
270 wenn teszt = egy dann
280 - -
290 leer.
300 - -
310 sonst
320 - -
330 springe anfang.
340 - -
350 wende
360 - -
370 pende ugrás.
```

2.14 Az alprogramok

Alprogramokat akkor használunk, ha az utasítások egy sorozatát csak egyszer kívánjuk tárolni, de a programban viszont többször is felhasználjuk azokat. A MINIATUR-ban írt alprogramok nem tartalmazhatnak lokális változókat, és csak végrehajtható utasításokból állhatnak.

Az alprogramokat a főprogramhoz csatlakoztatjuk és a következőképpen definiáljuk:

unterprogramm alprogramnév ist
– –
– –
– – utasításblokk
– –
– –
uende alprogramnév.

Az alprogramokat a következő utasítással hívjuk:

rufe alprogramnév.

A következő mintapélda az alprogram és a hívó utasítás működését mutatja be.

Mintapélda: **aprogteszt**

100 programm aprogteszt ist
110 – –
120 – –
130 – – Ez a program vagy az A vagy a B
140 – –alprogramot hívja a választás szerint.
150 – –
160 – –
170 fliess teszt, egy.
180 – –
190 beginne
200 – –
210 uebertrage 1.0 nach egy.
220 – –
230 sprungmarke kezdés.

```
240 - -
250 zeigez „alprogram 1/2?"
260 - -
270 hole teszt. vorschub.
280 - -
290 wenn teszt /= egy dann
300 - -
310 rufe a.
320 - -
330 sonst
340 - -
350 rufe b.
360 wende.
380 - -
390 springe kezdés.
400 - -
410 pende aprogteszt.
420 - -
430 - - - - - - - - - - - - - - - - -
440 unterprogramm a ist
450 - -
460 zeigez „alprogram a.".
470 - -
480 uende a.
490 - -
500 - - - - - - - - - - - - - - -
510 unterprogramm b ist
520 - -
530 zeigez „alprogram b.".
540 - -
550 uende b.
```

3. A LEXIKÁLIS ELEMZÉS

3.1 Miért szükséges a lexikális elemzés?

A programellenőrzés első lépése a lexikális elemzés. Feladata a programok olyan átalakítása, amely lehetővé teszi a további lépések egyszerűsítését.

A programokat ún. normálalakra kell hozni, amely a program bizonyos, lényegtelen részeit már nem tartalmazza. MINIATUR programot sokféleképpen leírhatunk anélkül, hogy magát a program logikáját megváltoztatnánk. Választhatunk nagy- vagy kisbetűs írásmódot, az egyes szavak között annyi üres helyet hagyhatunk, amennyit akarunk, megjegyzéseket tehetünk, a sorok számozását önkényesen választhatjuk meg stb.

A lexikális elemzésnek a tulajdonképpeni feladata a lényegesnek a lényegtelentől való elválasztása. A lényeges rész a compiler szempontjából értendő, hiszen egy megjegyzések nélküli program a felhasználó számára inkább csak sejtés, mint érthető rendszer. A programot ún. tokenekre (lexikális egységekre) bontjuk, amelyek a program legkisebb, önálló értelemmel rendelkező szimbólumai. A MINIATUR programokban a tokenek az olyan kulcsszavak, mint pl. a *programm*, az *ist* =, a., a), a *pende* stb., valamint a felhasználó által választott szavak, mint pl. a programnév, a változónév stb., továbbá bizonyos karaktersorozatok és a számok.
Azt a programot, amely kikeresi a MINIATUR programunkból a tokeneket, SCANNER-nek (keresőprogramnak) nevezzük.
Ezzel röviden megmagyaráztuk, mire való, ill. miért szükséges a lexikális elemzés. Részletesen a SCANNER-ről a következő fejezetben lesz szó. Azt, hogy a SCANNER feladata miért olyan fontos, a szintaktikai elemzésről szóló fejezetben tárgyaljuk.

3.2 Hogyan működik a SCANNER?

Vegyük újra szemügyre az előbbi fejezetben bemutatott program következő sorát!

340 ausgang egész wenn száz = szám.

Nos, mit csinál a SCANNER, amikor ezt a sort feldolgozza? A sor elején a sor száma található. Minthogy a program szempontjából ez lényegtelen, ezen átugrik. Ezután az *ausgang* betűkombináció, majd üres hely következik. Az üres helyről ismeri fel a SCANNER, hogy a szónak vége. Azt is szeretnénk azonban tudni, hogy az *ausgang* kulcsszó-e, vagy pedig a felhasználó által választott szó. Ennek eldöntésére a SCANNER átnézi a kulcsszavak szótárát, és meg is találja ezt a szót. A további programelemzési lépésekhez nem kell az egész szót megjegyezni, mivel a kulcsszavakhoz számokat rendelünk, amelyeket azok helyén jegyzünk meg. Ez megkönnyíti a kulcsszavakkal való foglalkozást, nem kell mindig a szótárban utánanézni, melyik kulcsszóról is van szó.

Az *ausgang* szó után az egész szó következik, amelynek végét a SCANNER ismét az üres helyről ismeri fel. Ez a szó nem kulcsszó, de megjegyezzük. A következő szó a *wenn,* ami kulcsszó, és azt már tudjuk, hogy a SCANNER-nek mi a teendője.
Minden programnyelvben vannak olyan jelek, amelyek egyértelműen jelzik, ha egy szónak vége van. E jeleket elhatároló jeleknek (angolul DELIMITER-nek) nevezik. A MINIATUR-ban ilyen többek között az egyenlőségjel és a pont. Ennek alapján most már tudjuk, hogyan megy végbe a megadott sor további részének feldolgozása.
A kicsit pontatlan leírás után, amelynek fő célja csak a szemléltetés volt, már nem is olyan sok választ el minket attól, hogy nekifogjunk a SCANNER program megírásának. Előbb azonban a SCANNER részfeladatait pontosan meg kell fogalmaznunk.

A részfeladatok a következők:

- a program olvasása és a listasorok kiírása;
- az elhatárolók felismerése;
- a kulcsszavak felismerése;
- az üres helyek megszüntetése, kivéve az olyan idézőjelbe tett szövegrészeket, mint pl. „enno a bolondos";
- a megjegyzéssorok eltávolítása;
- a tokenek kiadása.

3.3 A program olvasása és a listasorok kiírása

Ahhoz, hogy a számítógép a programot elemezni tudja, természetesen először is el kell tudnia olvasni. Ezt sokszor könnyebb kimondani, mint megtenni.

Eddig még nem beszéltünk arról sem, hogy milyen segédeszközökkel kell a lefordítandó programokat megírni. Az olyan programot, amellyel valamilyen nyelven programokat lehet szerkeszteni, editornak hívják.

Editorok pl. a különböző szövegszerkesztő programok, de editor pl. az olyan saját magunk által írt program is, amit arra a célra készítettünk, hogy adott nyelven írt programok bevitelét végezze. Számunkra mindenekelőtt a következő eset a legegyszerűbb.

Vesszük a gép operációs rendszerébe beépített editort, amellyel normális körülmények között BASIC programokat írunk. A BASIC programok a szokásos módon feldolgozhatók, változtathatók, tárolhatók és kinyomtathatók. A továbbiakban abból indulunk ki, hogy a lefordítandó program lemezen van. A lemezen lévő program csupán egy szöveghalmaz, amelyet számunkra értelmes szövegként szeretnénk olvasni. Értelmes szövegen ebben az esetben azt a betűsorozatot értjük, amelyet begépeltünk és amely ugyanilyen alakban listázható. Az értelmes szöveg tulajdonképpen a forrásprogram.

A BASIC editor minden BASIC kulcsszót számként kódolva tárol, így pl. a FOR szót 129-ként kódolja. A BASIC editor minden újonnan bevitt soron lexikális analízist végez, amely természetesen a BASIC nyelvű szövegre és nem a MINIATUR nyelvű szövegre vonatkozik. A sorok is meghatározott alakban kerülnek be a tárba és így kerülnek rá a lemezre is.

Nézzük meg tehát, hogy milyen alakban találjuk a BASIC editorral előállított szöveget! A szöveg elején két karaktert olvasunk el. Ezek azt az információt tartalmazzák, hogy a programszöveg melyik tárolóhelynél kezdődött, amikor a programot bevitték. E két karakter bennünket most nem érdekel. Ezután következnek a programsorok a megadott formában.

Az első két karakter a kapcsolócímke, amely a BASIC editornak azt jelezte, hogy a következő programsor melyik tárhelynél kezdődik. Ez a két karakter sem érdekel minket.

Ezután két olyan karakter következik, amelyek a dekódolás után a programsor számát adják meg.

Bár a MINIATUR programokhoz nem kell a sorok számozása, mégis, ha később el akarjuk készíteni programunk listáját, vonatkoztatási helyként szükség lesz rájuk.

A sorok számozását a következőképpen lehet dekódolni: az első karakter ASCII kódjához a második karakter ASCII kódjának 256-szorosát hozzáadjuk.

Az A és I betűt olvastuk ki. Az A-hoz és az I-hez tartozó ASCII értékek: A = 65 és I = 73. Így a sor számát a következő módon számítjuk ki:

65 + 73*256 = 18 751.

Ezután kiolvassuk a programsorunk további karaktereit. Itt a BASIC kulcsszavak kódoltan találhatóak. A BASIC szavakat 128 és 203 közötti ASCII értékekre kódolták. A normál karakterek ASCII értéke csak 127-ig terjed, így pontosan tudhatjuk, mikor találkozunk egy BASIC szóval, és mikor olvasunk normál karaktert. A kódolt BASIC szót dekódoljuk és így megkapjuk az általunk keresett értelmes szöveget.

A következő ASCII értékeket olvastuk ki:

80, 128, 69

Mivel a mellékelt táblázat szerint:

80 = p, 128 = end, 69 = e,

ebből a *pende* szó adódik.

A sor végét a BASIC editor a nulla ASCII értékkel jelzi.

A program végét a kapcsolócím jelzi, amely nulla nulla ASCII értékekből áll.

Ha olyan programot akarunk írni, amelyik a szöveget elolvassa, értelmes szöveggé alakítja és kiírja a képernyőre, akkor ehhez szükségünk van egy olyan táblázatra, amely (kölcsönösen és egyértelműen) megfeleltei a betűket és az egyéb jeleket (együttesen karakterek) azok ASCII-értékeinek. Ismernünk kell még a BASIC szavakhoz egyértelműen hozzárendelt ASCII értékeket is.

Az ASCII értékeken a karakterek és a kulcsszavak azon decimális értékeit értjük, amelyet a C64-es a belső rendszerében használ. Egy más típusú számítógép esetén ezek a listák valószínűleg másként néznek ki (mások a számértékek), de ezeket az értékeket a számítógép kézikönyvéből ki lehet keresni.

A következő táblázatban a karakterek ASCII értékeit adjuk meg.

Érték	Karakter	Érték	Karakter
32	üres hely	33	!
34	üres hely	35	#
36	$	37	%
38	&	39	'
40	(	41	)
42	*	43	+
44	,	45	-
46	.	47	/
48	0	49	1
50	2	51	3
52	4	53	5
54	6	55	7
56	8	57	9
58	:	59	;
60	<	61	=
62	>	63	?
64	§	65	a
66	b	67	c
68	d	69	e
70	f	71	g
72	h	73	i
74	j	75	k
76	l	77	m
78	n	79	o
80	p	81	q
82	r	83	s
84	t	85	u
86	v	87	w
88	x	89	y
90	z	91	[
92	£	93	]
94	.	95	→
96	emelt üres hely		

97-től a grafikai szimbólumok következnek a 128-as ASCII értékig, majd a BASIC szimbólumok ASCII értékei:

128 END	129 FOR
130 NEXT	131 DATA
132 INPUT #	133 INPUT
134 DIM	135 READ
136 LET	137 GOTO
138 RUN	139 IF

140 RESTORE	141 GOSUB
142 RETURN	143 REM
144 STOP	145 ON
146 WAIT	147 LOAD
148 SAVE	149 VERIFY
150 DEF	151 POKE
152 PRINT#	153 PRINT
154 CONT	155 LIST
156 CLR	157 CMD
158 SYS	159 OPEN
160 CLOSE	161 GET
162 NEW	163 TAB(
164 TO	165 FN
166 SPC(	167 THEN
168 NOT	169 STEP
170 +	171 −
172 *	173 /
174 ↑	175 AND
176 OR	177 <
178 =	179 >
180 SGN	181 INT
182 ABS	183 USR
184 FRE	185 POS
186 SQR	187 RND
188 LOG	189 EXP
190 COS	191 SIN
192 TAN	193 ATN
194 PEEK	195 LEN
196 STR$	197 VAL
198 ASC	199 CHR$
200 LEFT$	201 RIGHT$
202 MiD$	203 GO

Most pedig kezdjünk hozzá ahhoz a programhoz, amely a lemezen lévő szöveghalmazt értelmes szöveggé dolgozza fel, és sorról sorra listázza. Ezt a programot fogjuk azután a fennmaradó részfeladatoknak megfelelően bővíteni. A programot úgy írjuk meg, hogy alprogramonként a szintaktikai elemzésre is felhasználhassuk.

A szintaktikai elemzést végző program mindannyiszor be fogja hívni a lexikális elemzést végző programot, valahányszor új tokent talál.

A program felosztása

Az 1000-esig terjedő sorokat a felhasznált változók meghatározására és magyarázatára használjuk fel. Az 1000-estől a 2000-esig a sorokat az előkészítendő feladatok számára tartjuk fenn.

A 2000-estől az 50 000-esig a sorokba a szintaktikai elemzést végző program kerül. Az 50 000-es sortól kezdődik majd a lexikális elemzést végző program.

Az egyes sorokat nem feltétlenül abban a sorrendben fogjuk tanulmányozni, ahogyan majd a kész programban megjelennek, hanem megpróbáljuk a programot logikai felépítése szerint megmagyarázni.

```
10 rem Program a MINIATUR programok
20 rem lexikális és szintaktikai
30 rem elemzésére
40 rem – –
50 rem Deklarációk
60 rem – –
70 :
```

Abból indulunk ki, hogy a fordítandó program lemezen van. A szintaktikai elemzés eredményeit lemezen tároljuk, a programlistát pedig a képernyőn vagy a nyomtatón jelenítjük meg.

```
80 fl = 8:                    rem  A lemez száma
90 be = 15:                   rem  A lemezhez rendelt logikai file száma
95:                           rem  és másodlagos címe
100 in = 8:                   rem  Az adatbeviteli csatorna logikai
105:                          rem  file száma és másodlagos címe
110 ou = 7:                   rem  Az adatkiviteli csatorna logikai
115:                          rem  file száma és másodlagos címe
120 pr = 3:                   rem  A programlista kiviteli csatornája
125:                          rem  Képernyőbeállítás
130 i1$="":i2$="":i3$="":     rem  Beviteli változók
140 i1= 0:i2= 0:i3|0:         rem  A megfelelő ASCII értékek
```

A programot bevihetjük úgy, ahogy ezt itt leírtuk, és a könyv végén megtalálhatjuk a program listáját is.

Kezdjük tehát azokkal az előkészületekkel, amelyet a programnak mindjárt az elején el kell végeznie:

```
1000 rem – –
1010 rem Előkészületek
1020 rem – –
```

```
1030 :
1040 print "clr billentyű" : rem Képernyőtörlés
```

Az 1040-es sornál a képernyő törlése következne, ezért a CLR billentyű kiírása helyett egyszerűen meg kell nyomni ezt a billentyűt.

```
1050 print : print "a MINIATUR fordító": print
1060 print:"lista a képernyőre vagy a nyomtatóra (k/ny)?"
1070 get i1$ : if i1$ = "k" or i1$ = "ny" then 1090
1080 goto 1070
1090 if i1$ = "ny" then pr = 4
1100 open pr,pr,7 : rem A kiviteli csatorna megnyitása
```

A programlista kivitelére a megfelelő csatorna megnyitása.

```
képernyő esetén open 3,3,7
nyomtató esetén open 4,4,7.
```

A nyomtató tehát a 4-es egységszámot kapja.

```
1110 print : print "tessék a lemezt betenni."
1120 print "tovább returnnal!" : print
1130 get i1$ : if i1$ <> chr $(13) then 1130
1140 print "melyik programot kell lefordítani?"
1150 input "név:",i2$
```

Most már minden információnk megvan ahhoz, hogy a lemezen levő megfelelő filet megnyissuk. Először azonban még megnyitjuk a lemezegységhez tartozó csatornát és a lemezegységet olyan állapotba hozzuk, amilyenben közvetlenül a bekapcsolás után volt.

Lehet, hogy az olvasó a lemezegységgel kapcsolatos eljárást túl körülményesnek érzi, de a parancsnak és az előírásnak, ami első pillanatra feleslegesnek tűnik, megvan a maga értelme.

A szintaktikai elemzés eredményeit szekvenciális fileba, miniatur syn néven szeretnénk megőrizni. Ha a lemezzel már végeztünk fordítást, úgy ilyen nevű file már létezik, ezért ezt törölnünk kell. Az is megtörténhet, hogy a program felhasználója a lemezen lévő program behívásakor megnyomja ugyan a RETURN billentyűt, de előzőleg nem helyezett lemezt az egységbe. Ezért most írunk egy olyan alprogramot, amelyet akkor hívunk meg, ha tudni akarjuk, hogy a lemezegységgel kapcsolatban elkövettünk-e valamilyen hibát.

A csatorna megnyitása a parancsok számára:

```
1160 open be,fl,be
1170 print# be,"i"
```

A hibakezelést végző alprogram:

```
60000 rem A lemezegység hibacsatornájának lekérdezése
60010 :
60020 input#be,en,em$,et,es : if en=0 then return
60030 print
60040 print "* * * * hiba van a lemezen!"
60050 print "* * * * hiba száma: ";en
60060 print "* * * * hibajel:"
60070 print "* * * *"; em$
60080 print "* * * * hibás szektor : ";et
60090 print "* * * * hibás nyom :";es
60100 print#be,"i"
60110 close be : close pr
60120 print : print "* * * * program leállt!"
60130 end
```

Itt az en, az em$, az et és az es új változókat használtuk. Ezért a 150-es számú sorra is szükségünk van (az előzőekben csak a 140-esig jutottunk), amely a következőképpen alakul:

```
150 en=0:em$="":et=0:es=0:rem A hibacsatorna lekérdezése
```

A miniatur syn törlése:

```
1180 print#be,"S:miniatur syn"
1190 Input#be,en,em$,et,es
1200 if en<> 1 then 60030
```

Egy adat törlése után az en értéke 1, ezért közvetlenül nem tudjuk az alprogramot működtetni, le kell kérdezni a hibacsatornát.

Megnyitjuk a be- és kiviteli adatfile-okat:

```
1210 open in,fl,in,i2$+"p,r" : gosub 60000
1220 get#in,i1$,i1$
1230 open ou,fl,ou,"miniatur syn,s,n" : gosub 60000
```

Az 1220-as sorban az adatfile első két karakterét átugorjuk, mert nincsen számunkra jelentőségük.

```
1240 gosub 59000
```

Az 59000-es sorban kezdődő alprogramban a kódolt BASIC szavakat értelmes szövegként a ba$-ban tároljuk. Mivel összesen 76 BASIC szó van, a 160-as sor a következő lesz:

```
160 dim ba$(75) : rem BASIC szavak az értelmes szövegben

59000 rem BASIC szavak az értelmes szövegben
59005 :
59008 ba$(0) = "END"
59010 ba$(1) = "FOR"
59020 ba$(2) = "NEXT"
59030 ba$(3) = "DATA"
59040 ba$(4) = "INPUT#"
59050 ba$(5) = "INPUT"
59060 ba$(6) = "DIM"
59070 ba$(7) = "READ"
59080 ba$(8) = "LET"
59090 ba$(9) = "GOTO"
59100 ba$(10) = "RUN"
59110 ba$(11) = "IF"
59120 ba$(12) = "RESTORE"
59130 ba$(13) = "GOSUB"
59140 ba$(14) = "RETURN"
59150 ba$(15) = "REM"
59160 ba$(16) = "STOP"
59170 ba$(17) = "ON"
59180 ba$(18) = "WAIT"
59190 ba$(19) = "LOAD"
59200 ba$(20) = "SAVE"
59210 ba$(21) = "VERIFY"
59220 ba$(22) = "DEF"
59230 ba$(23) = "POKE"
59240 ba$(24) = "PRINT#"
59260 ba$(26) = "CONT"
59270 ba$(27) = "LIST"
59280 ba$(28) = "CLR"
59290 ba$(29) = "CMD"
59300 ba$(30) = "SYS"
59310 ba$(31) = "OPEN"
59320 ba$(32) = "CLOSE"
59330 ba$(33) = "GET"
59340 ba$(34) = "NEW"
59350 ba$(35) = "TAB("
59360 ba$(36) = "TO"
59370 ba$(37) = "FN"
59380 ba$(38) = "SPC("
59390 ba$(39) = "THEN"
59400 ba$(40) = "NOT"
59410 ba$(41) = "STEP"
```

```
59420 ba$(42) = " + "
59430 ba$(43) = " - "
59440 ba$(44) = "*"
59450 ba$(45) = "/"
59460 ba$(46) = "↑"
59470 ba$(47) = "AND"
59480 ba$(48) = "OR"
59490 ba$(49) = " < "
59500 ba$(50) = " = "
59510 ba$(51) = " > "
59520 ba$(52) = "SGN"
59530 ba$(53) = "INT"
59540 ba$(54) = "ABS"
59550 ba$(55) = "USR"
59560 ba$(56) = "FRE"
59570 ba$(57) = "POS"
59580 ba$(58) = "SQR"
59590 ba$(59) = "RND"
59600 ba$(60) = "LOG"
59610 ba$(61) = "EXP"
59620 ba$(62) = "COS"
59630 ba$(63) = "SIN"
59640 ba$(64) = "TAN"
59650 ba$(65) = "ATN"
59660 ba$(66) = "PEEK"
59670 ba$(67) = "LEN"
59680 ba$(68) = "STR$"
59690 ba$(69) = "VAL"
59700 ba$(70) = "ASC"
59710 ba$(71) = "CHR$"
59720 ba$(72) = "LEFT$"
59730 ba$(73) = "RIGHT$"
59740 ba$(74) = "RND$"
59750 ba$(75) = "GO"
59760 RETURN
59770 :
```

Ezt az alprogramot egyszer lefuttatjuk a lexikális elemzés kezdetekor.

Most már csaknem eljutottunk odáig, hogy megírhatjuk azt a programot, amely a szintaktikai elemzésnél először belép. Ez a program nem tesz mást, mint a lexikális elemzést végző alprogramtól mindaddig új tokent követel, amíg csak a szöveg olvasásának végére nem jutott.

A lexikális elemzést végző programnak azonban még az induláshoz segítséget is kell adnunk, nevezetesen el kell olvasnunk számára, az első kapcsolócímet

és az első sorszámot. Ehhez írunk egy alprogramot, (az 51000-es sorban kezdődik), amelyet a későbbiekben mindig akkor hívunk be, ha egy sor végére érünk, és az említett feladatok előttünk állnak.

A lexikális elemzés során az i1$-ban annak a karakternek kell lennie, amelyet éppen feldolgozunk, és az i2$-ban, valamint az i3$-ban pedig a soron következő két karakternek, ezáltal lehetőségünk nyílik arra, hogy programunkban kissé előre tekinthessünk. Amint majd látni fogjuk, gyakorlati szempontból ez előnyösnek bizonyul. Az i1, i2, i3-ban a megfelelő füzérváltozók ASCII értékeit fogjuk tárolni.

Az az alprogram, amely egy karaktert beolvas és az i1$, az i2$, az i3$ értékét megfelelően lépteti, az 51500 sornál kezdődik:

```
51500 rem karaktert beolvasni
51505:
51510 i1$=i2$ : i2$=i3$ : get#in,i3$
51520 i1=i2 : i2=3 : if i3$=""then i3=0: return
51530 i3=asc(i3$): return
51540
```

Ezt az alprogramot kétszer kell hívnunk, mielőtt az 51000-es sorban kezdődő alprogramhoz ugrunk:

```
1250 gosub 51500 : gosub 51500 : gosub 51000
51000 rem sorkezdet
```

Az i2$-ban és az i3$-ban a következő sorra utaló kapcsolócímet találjuk, ezt átugorjuk.

Ezután az a két karakter következik, amelyek az utasítás sorszámát tartalmazzák. A sorszám kiíratásához először egy, a 254-es ASCII értékű karaktert, majd a sor számának két karakterét visszük ki a lemezegységre. A 254-es karakter más összefüggésben nem kerülhet elő, és így később a 254-es karakter olvasásakor az utána következő sorszámot fel tudjuk ismerni.

A továbbiakban a képernyőre, ill. a nyomtatóra való kivitelt a következő sorra kell állítanunk. Ehhez egy kocsi vissza jelet küldünk a PR csatornára, majd kiírjuk a megfelelő utasítás sorszámát. Ezután hívjuk az i1$-t, az i2$-t és az i3$-t. Az i1$-ban az új sor első karakterének kell állnia.

```
51005 :
51010 gosub 51500 : gosub 51500 : gosub 51500
51020 print#uo chr$(254); i1$;i2$;
51030 print#pr,chr$(13);str$(i1+i2*256);
51040 gosub 51500 : gosub 51500 : return
```

Annak érdekében, hogy a program áttekinthetőbb legyen, a gyakran futtatott alprogramok egy részét a teljes program végére helyezzük! Azoknál a programoknál, amelyek interpreterrel futnak, ez nem jó, mert a program futtatása így lassúbb.

A gépi kódra fordított programoknál ez nem fordulhat elő. A programrész, amely a szintaktikai elemzést vezérli:

```
2000 gosub 50000 : if t$<>chr$(255) then 2000
2010 print#pr, chr$(13) "lexikális elemzés lezárva!"
2020 close pr
2030 close in
2040 close ou
2050 close be
2060 end
```

Ez a programrész mindaddig egy új tokent hív, amíg a t$=chr$(255) feltétel nem teljesül. Ez a karakter a program végét fogja jelenteni. A megnyitott file-okat lezárjuk, hogy a program befejezése teljes legyen.

Térjünk vissza a program olvasása és a listasorok kiadása című feladathoz:

```
50000 rem Lexikális elemzés
50010 if i1=0 then if i2=0 then if i3=0 then t$=chr$(255): return
```

Ha az i1, az i2 és az i3 egyenlő nullával, akkor ez azt jelenti, hogy a program végére jutottunk. A kapcsolócím nulla.

```
50020 if i1=0 then gosub 51000
```

A sor végét felismerjük, és a megfelelő alprogramra ugrunk.

```
50030 if i1>127 and i1<204 then i1$=ba$(i1-128)
```

Ha i1 nagyobb mint 127 és kisebb mint 204, akkor az i1$-ban egy kódolt BASIC szó van, amelyet dekódolnunk kell, vagyis az i1$ helyébe az az értelmes szöveg kerül, amelyet ba$-ban tárolunk.

```
50200 print#pr, i1$; gosub 51500
50210 return
```

Most néhány sorszámot kihagyunk, hogy a SCANNER további feladatainak elvégzését majd innen vezéreljük.

Az így elkészített program most már futtatható. A futtatás eredményeképpen a MINIATUR programunk egy listáját kapjuk.

3.4 A megjegyzéssorok eltávolítása

A következő részfeladat a megjegyzéssorok eltávolítása.

A megjegyzéseket két egymás után következő vonal (mínuszjel) vezeti be. A sor ezután következő részét tehát el lehet hagyni.

A program a megjegyzéseket a 50040-es sor segítségével ismeri fel:

```
50040 if i1=171 then if i2=171 then gosub 51700 : goto 50000
```

A következő utasítások a programsort végigolvassák, és kiviszik a képernyőre:

```
51700 rem A sort végéig elolvasni és elhagyni
51705 :
51710 print#pr, i1$; : gosub 51500 : if i1=0 then return
51720 if i1>127 and i1<204 then i1$=ba$(i1-128)
51730 goto 51710
51740 :
```

3.5 Az üres helyek megszüntetése

A SCANNER fontos részfeladata a felesleges üres helyek felismerése és eltávolítása. Most ezzel foglalkozunk.

A MINIATUR-programban egynél több üres hely felesleges. Ezeket a következő programsorral távolítjuk el:

```
50050 if i1$=" " then ifi2$=" " then print # pr, i1$; : gosub
51500 : goto 50000
```

A tokenek azonban legtöbbször több karakterből állnak, és a végüket az elhatároló jelek alapján ismerjük fel. Eszerint az i$-t mindaddig T$-al bővítjük, amíg az i2$-ban elhatároló jelet nem találunk:

```
50190 t$=t$+i1$
```

A 2000-es sort a következőképpen bővítjük:

```
2000 t$=" " : gosub 50000...
```

A továbbiakban még két alprogramra van szükségünk: az egyikre annak eldöntéséhez, hogy egy token MINIATUR kulcsszó-e; a másikra pedig a karaktersorozatok összeállításához. Karaktersorozatokon belül ugyanis sem a nagybetűs írást kisbetűssé alakítani, sem pedig az üres helyeket törölni nem lehet.

Az első alprogramot az 52000-es sortól, a másodikat pedig az 53000-es sortól kezdődően írjuk meg.

```
50070 if i1=34 then gosub 53000 : goto 50000
```

Ha az aktuális karakter idézőjel, akkor egy karaktersor elolvasásáról és megőrzéséről van szó:

```
53000 rem A karaktersor elolvasása
53010 t$=t$+i1$: gosub 51500 : if i1<>34 then goto 53010
53020t$=t$+i1$: return
53030 :
```

Az elhatároló jelek a mi esetünkben a következők:

=, /, <, >, ., /, /, sorvége, ", üres hely.

Négy eset lehetséges.
1) Olyan karaktersorozatot olvastunk el, amelyet pont követ:

```
50080 if i1=34 then if i2$="." then 50200
50085 if i1=32 then if i2=34 then goto 50130
```

2) Az i2$-ban elhatároló jel van:

```
50090 if i2$="=" or i2$="/" or i2$="<" or i2$=">" then gosub 52000 :
      goto 50200
50100 if i2$="." or i2$="  " or i2=0 or i2=34 then gosub 52000 : goto 50200
50110 if i2$=',' then gosub 52000 : goto 50200
```

3) Az i1$-ban elhatároló jel van:

```
50126 if i1$="." or i1$="," or i1$="=" then t$=i1$: goto 50200
50128 if i1$="/" or i1$="<" or i1$=">" then t$=i1" : goto 50200
```

4) Az i1$ üres hely:

```
50130 if i1$="  " then print#pr, "  "; : gosub 51500 : goto 50000
```

Most már csak az az 52000-es sorral kezdődő alprogram hiányzik, amely felismeri, hogy egy token MINIATUR kulcsszó-e vagy nem? Ha a token MINIATUR kulcsszó, akkor t$ értéke a kulcsszó kódja lesz.

A MINIATUR kulcsszavakhoz a 128-astól kezdődő számokat rendeljük hozzá.

A következőkben közöljük a MINIATUR kulcsszavakat, és a hozzájuk tartozó számokat.

Érték	Kulcsszó
128	addiere
129	ausgabegeraet
130	ausgang
131	beginne
132	bilde
133	cuspalte
134	cuzeile
135	donn
136	durch
137	dividiere
138	fliess
139	hintergrund
140	hole

141	ist
142	leer
143	mit
144	multipliziere
145	noch
146	pende
147	potenziere
148	programm
149	rahmen
150	rufe
151	schrift
152	schirmfrei
153	schleife
154	sende
155	sonst
156	springe
157	sprungmarke
158	subtrahiere
159	ueber
160	uebertrage
161	uende
162	unterprogramm
163	von
164	vorschub
165	wende
166	wenn
167	zeige
168	zeigez
169	zeigeas
170	zu

Eljárhattunk volna úgy is, hogy a t$-t minden kulcsszóval összehasonlítjuk. Ez azonban nagyon időigényes lenne, ezért inkább megnézzük a t$ kezdőbetűjét, és akkor már tudjuk, hogy egyáltalában milyen kulcsszavak jöhetnek szóba. Az ezzel a betűvel kezdődő szavakat már hamar át lehet nézni.

A MINIATUR kulcsszavakat szintén egy megadott programtartományban helyezzük el. Ez ugyan pillanatnyilag még nem volna feltétlenül szükséges, de mivel majd a szintaktikai elemzés számára a hibakezelésnél újra szükségünk lesz ezekre a kulcsszavakra, ily módon végül is tárhelyet takarítunk meg.

Összesen 49 MINIATUR kulcsszó van.

```
  170 dim mi$(48) : rem Az értelmes szöveg miniatur szavai
 1245 gosub 58000
58000 rem miniatur kulcsszavak
58005 :
58008 mi$(0) = "addiere"
58010 mi$(1) = "ausgabegeraet"
58020 mi$(2) = "ausgang"
58030 mi$(3) = "beginne"
58040 mi$(4) = "bilde"
58050 mi$(5) = "cuspalte"
58060 mi$(6) = "cuzeile"
58070 mi$(7) = "dann"
58080 mi$(8) = "durch"
58090 mi$(9) = "dividiere"
58100 mi$(10) = "fliess"
58110 mi$(11) = "hintergrund"
58120 mi$(12) = "hole"
58130 mi$(13) = "ist"
58140 mi$(14) = "leer"
58150 mi$(15) = "mit"
58160 mi$(16) = "multipliziere"
58170 mi$(17) = "nach"
58180 mi$(18) = "pende"
58190 mi$(19) = "potenziere"
58200 mi$(20) = "programm"
58210 mi$(21) = "rahmen"
58220 mi$(22) = "rufe"
58230 mi$(23) = "schrift"
58240 mi$(24) = "schirmfrei"
58250 mi$(25) = "schleife"
58260 mi$(26) = "sende"
58270 mi$(27) = "sonst"
58280 mi$(28) = "springe"
58290 mi$(29) = "sprungmarke"
58300 mi$(30) = "subtrahiere"
58310 mi$(31) = "ueber"
58320 mi$(32) = "uebertrage"
58330 mi$(33) = "uende"
58340 mi$(34) = "unterprogramm"
58350 mi$(35) = "von"
58360 mi$(36) = "vorschub"
58370 mi$(37) = "wende"
58380 mi$(38) = "wenn"
58390 mi$(39) = "zeige"
58400 mi$(40) = "zeigez"
```

```
58410 mi$(41)="zeigeas"
58420 mi$(42)="zu"
58430 mi$(43)="fuer"
58440 mi$(44)="bis"
58450 mi$(45)="wiederhole"
58460 mi$(46)="/="
58470 mi$(47)="<="
58480 mi$(48)=">="
58490 return
58500 :

52000 rem a t$ token?
52010
52020 t$=t$+i1$ : t=0 : a=asc(left$(t$,1))
```

A t értékbe az alprogram futása után a MINIATUR kulcsszó ASCII kódjának kell kerülnie. Ha a t értéke az alprogram futása során nem változik meg, akkor a t$ nem kulcsszót tartalmaz. Az a-ba a t$ első jelének ASCII értéke kerül.

```
180 t$ ="   ":t=0     : rem Token változók
190 a=0               : rem A segédváltozó a lexikális elemzéshez
```

Felmerül a kérdés, hogy a t$ egyáltalában betűkombinációt tárol-e.

```
52030 if a < 65 vagy a > 90 then return
```

Most egy kiszámított ugrás következik az egyes betűkhöz tartozó alprogramokhoz. BASIC nyelvben ez kissé komplikáltnak látszik, de valójában nem az.

Először is az ábécét felosztjuk 5 olyan szakaszra, amely egyenként 5–5 betűt tartalmaz, és egy olyan szakaszra, amely egy betűt tartalmaz.

```
52040 on int ((a-65)/5) goto 52060, 52070, 52080, 52090, 52100
```

Most már az egyes betűknek megfelelően ágaztathatunk el. Ha a értéke 65 és 69 között van, akkor int ((a−65/5) értéke nulla, és a program a következő sorra tér rá.

Ha egy betűhöz nem tartozik kulcsszó, akkor az 52900-as sorra ugrunk, amelyben egy RETURN található:

```
52900 return

52050 on (a-64) goto 52200, 52220, 52240, 52260, 52900
52060 on (a-69) goto 52300, 52900, 52340, 52360, 52900
```

```
52070 on (a-74) goto 52900, 52380, 52400, 52420, 52900
52080 on (a-79) goto 52440, 52900, 52460, 52480, 52900
52090 on (a-84) goto 52500, 52520, 52540, 52900, 52900
52100 goto 52560
```

Miután egy betűre befejeztük ezt az eljárást, az 52800-as sorra térünk.

```
52800 if t=0 then return
52810 t$=chr$(t) : return
```

Ha t=0, akkor t$ nem kulcsszót tartalmaz, minden más esetben a kódolt kulcsszó a t$-ban található.

```
52200 b=0 : c=2 : gosub 52700 : return : rem a
52220 b=3 : c=4 : gosub 52700 : rem b
52230 b=44 : c=44 : gosub 52700 : return : rem bis
52240 b=5 : c=6 : gosub 52700 : return : rem c
52260 b=7 : c=9 : gosub 52700 : return : rem d
52300 b=10 : c=10 : gosub 52700 : rem f
52310 b=43 : c=43 : gosub 52700 : return : rem fuer
52340 b=11 : c=12 : gosub 52700 : return : rem h
52360 b=13 : c=13 : gosub 52700 : return : rem i
52380 b=14 : c=14 : gosub 52700 : return : rem l
52400 b=15 : c=16 : gosub 52700 : return : rem m
52420 b=17 : c=17 : gosub 52700 : return : rem n
52440 b=18 : c=20 : gosub 52700 : return : rem p
52460 b=21 : c=22 : gosub 52700 : return : rem r
52480 b=23 : c=30 : gosub 52700 : return : rem s
52500 b=31 : c=34 : gosub 52700 : return : rem u
52520 b=35 : c=36 : gosub 52700 : return : rem v
52540 b=37 : c=38 : gosub 52700 : rem w
53550 b=45 : c=45 : gosub 52700 : return : rem wiederhole
52560 b=39 : c=42 : gosub 52700 : return : rem z

52700 for d=b to c
52710 if t$=mi$(d) then t=128+d
52720 next d
```

Az mi$ tömbben a MINIATUR szavakat ABC sorrendben tároljuk. A fenti alprogrammal egy adott betűnél kezdődő kulcsszavak tömbbeli helyét határozzuk meg. A kezdő index értékét b-ben, az utolsó index értékét c-ben tároljuk. Ne felejtsük el az a-t, a b-t, a c-t és a d-t kinullázni!

```
190 a=0:b=0:c=0:d=0:rem A segédváltozók a lexikális elemzéshez
```

Ezzel a lexikális elemzés végéhez értünk. Ebben a könyvben a BASIC programozás speciális lehetőségeit ismertetjük, amelyeket könnyű megérteni, változtatni és bővíteni. Ezek a lehetőségek sokszor bonyolultnak tűnhetnek, de a programban már egészen egyszerűek.

Ha a programot az olvasó már bevitte a számítógépbe, akkor kérjük, hasonlítsa össze a könyvben hátrább található listával, amelynek címe: A Parser listája. Ez a könyv egyik programja alapján készült. A fenti programsorokat kézzel gépelték, ezért kétségek, ill. eltérések esetén a gépi listázás a mértékadó.

4. A SZINTAKTIKAI ELEMZÉS

A szintaktikai elemzésnél a következő kérdés merül fel: megfelel-e MINIATUR programunk a MINIATUR nyelvtan szabályainak? A MINIATUR programot egy – ezen a nyelven írt – forrásszövegként fogjuk fel, és megvizsgáljuk, hogy a szöveg mondatai kielégítik-e a nyelv szabályait. A feladat hasonló ahhoz, mint amikor megvizsgáljuk, hogy pl. egy magyar mondat megfelel-e a magyar nyelv szabályainak. Induljunk tehát ki egy ilyen példából.

A kutya ugat.

Rövidítsük a mondatra vonatkozó szabályt a következőképpen:

mondat: : = alany állítmány.

Ezt így is olvashatjuk: a mondat alanyból, állítmányból és egy pontból áll. Most már tudjuk, miből áll a mondat; de mi az az alany, és mi az az állítmány? Azt, hogy mi az a pont, már tudjuk, ott is van a mondat végén. A mondatban előforduló olyan szimbólumokat – ilyen a pont is – amelyek további elemzést nem igényelnek, terminális szimbólumoknak fogjuk nevezni. A további elemzésre szoruló szimbólumokat pedig ennek megfelelően nem terminális szimbólumoknak nevezzük.

```
Alany:   : = névelő főnév
Névelő:  : = a
         : = az
Főnév:   : = autó
         :   fémdoboz
         :   kutya
```

Az alany a névelőből és a főnévből tevődik össze. A névelő lehet a vagy az, tehát több lehetőség áll fenn; ezt a kettőspont használatával jelezzük. Főnévként az autó a fémdoboz és a kutya a lehetséges választék.

```
Állítmány : : = ige
Ige       : : = ugat
            :   csillog
            :   gurul
```

Az állítmány csak igéből áll, és a fenti három ige közül választhatunk.

Elemezzük most mondatunkat:

```
   A  kutya     ugat
   alany        állítmány
névelő főnév     ige
   A  kutya     ugat.
```

Megállapíthatjuk, hogy ez a mondat szabályos magyar mondat. A nyelvtan, amelyet kialakítottunk nagyon egyszerű, és az elemzésnél úgy járhattunk el, hogy az egyes lehetőségeket végigpróbáltuk. Hosszabb, bonyolultabb nyelvtani szerkezetek esetében azonban ez nagyon soká tarthat, még akkor is, ha számítógéppel végezzük. Itt majd még egy előírást vagy algoritmust kell tanulmányoznunk, de erről majd később beszélünk. Először lássunk hozzá ahhoz, hogy a MINIATUR nyelvtanát összeállítsuk. Ehhez még egyszer pontosan szemügyre kell vennünk a MINIATUR utasításait.

A nyelvtan

Minden MINIATUR program alapvető felépítése a következő:

```
programm programnév ist
– –
– – definíciós blokk
– –
beginne
– –
– – utasításblokk
– –
pende programnév
– –
– –
alprogramok
```

Írjuk fel szabályként:

```
program: := 'programm' programnév 'ist'
            definíciós blokk 'beginne'
            utasításblokk 'pende'
            programnév '.' alprogram EOF
```

Ezzel a Program nyelvtani fogalmát határoztuk meg. Ebben még a következő eddig nem meghatározott fogalmak fordulnak elő:

programnév,
definíciós blokk,
utasításblokk,
alprogram.

Ha az ismeretlen fogalmakat és azok szabályait meghatározzuk, akkor nyelvtanunk összeállításával elkészültünk.

Még néhány megjegyzés az írásmódról. A ' ' közötti szavak terminális szimbólumok. Minden nem terminális szimbólumot, amely névre végződik, együttesen határozzunk meg. A szó előtagja (pl. program-, ciklus-, alprogram-) alapján szintaktikai elemzést nem végzünk. Minden névnek ugyanolyan szintaktikai felépítésűnek kell lennie, míg a program-, alprogram- előtag a név fajtáját adja meg. Erre majd a szemantikai elemzésnél lesz szükségünk. Pl: egy programnevet nem szabad változónévként használnunk.

Az EOF nem terminális szimbólum jelentése: END OF FILE a programban szereplő adatállomány (röviden a program) fizikai vége.

Az EOF bevezetése kissé mesterkéltnek tűnhet, de erre a szimbólumra a szintaktikai elemzésnél szükségünk lesz.

Alprogram:: = *'unterprogamm'* alprogramnév
'ist' utasításblokk *'uende'*
alprogramnév '.'
Alprogram
:L

E szabály alkalmazásakor két lehetőségünk van. Az L azt jelenti, hogy az alprogram üres is lehet (*Leer*-üres), ugyanis egy MINIATUR program alprogramokat nem feltétlenül tartalmaz. Minthogy a Program nyelvtani szabálya minden MINIATUR programra érvényes, és ebben a szabályban az Alprogram-szabály is előfordul, az Alprogram-szabályt mindig alkalmaznunk kell.

Ha MINIATUR programunkban nincsenek alprogramok, akkor az L anternatívát alkalmazzuk.

Az első alternatíva azt írja elő, hogy egy alprogramot kell felépíteni, és ez ismét az Alprogram hívásával végződik. Ez a szabály a végén újra önmagát hívja, mert lehetséges, hogy több alprogram is van a programban. Ha csak egy alprogram van, akkor a szabály második alkalmazásakor az L alternatívát kell venni. Újabb nem terminális szimbólumok az Alprogram-szabály során nem lépnek fel.

Programnév, változónév

Név :: = Betű Név-1
Név-1 :: = Betű Név-1
: L

```
Betű: : = 'a'
      :   'b'
      :   'c'
      :   .
      :   .
      :   'x'
      :   'y'
      :   'z'
```

Eszerint egy név tetszőleges számú betűből állhat. Korlátként a 80 betűt állítjuk fel; ugyanis ennél több betűből álló nevet nem is tudunk bevinni.

Definíciós blokk

```
Definíciós blokk          : : = Változódefiniálás
                            : L
Változódefiniálás         : : = 'fliess' változónév
                                  Változódefiniálás-2'.'
                                  Definíciós blokk
Változódefiniálás-2       : : = ',' változónév
                                  Változódefiniálás-2
                            : L
```

A definíciós blokk üres is lehet. Ha nem az, akkor a változók meghatározásából áll. A Változódefiniálás című szabály a végén a definíciós blokkot hívja, mert tetszőleges számú változódefiniálás lehet a programban. Egy változódefiniáláson belül több változót is lehet meghatározni (ezeket vesszővel kell elválasztani). Ezért szerepel a Változódefiniálás-2.

Az utasításblokk

Az utasításblokkra vonatkozó szabály eléggé terjedelmes, mert az összes utasításképzési lehetőséget figyelembe kell venni.

```
Utasításblokk             : : = Utasítás Utasítássorozat
Utasítássorozat           : : = Utasítás Utasítássorozat
                            : L
Utasítás                  : : = 'leer' '.'
                              : 'zeige' kiírás–1 '.'
                              : 'zeigez' kiírás soremeléssel '.'
                              : 'vorschub' '.'
                              : 'schirmfrei' '.'
                              : 'cuspalte' egészszám '.'
                              : 'cuzeile' egészszám '.'
```

: *'zeigeas'* egészszám '.'
: *'ausgabegaraet'* készülékmegjelölés '.'
: *'rahmen'* színmegjelölés '.'
: *'hintergrund'* színmegjelölés '.'
: *'schrift'* színmegjelölés '.'
: *'hole'* változónév '.'
: *'uebertrage'* átvitel–1 '.'
: *'addiere'* változónév *'zu'* változónév *'nach'* változónév '.'
: *'subtrahiere'* változónév *'von'* változónév *'nach'* változónév '.'
: *'multipliziere'* változónév *'mit'* változónév *'nach'* változónév '.'
: *'dividiere'* változónév *'durch'* változónév *'nach'* változónév '.'
: *'potenziere'* változónév *'mit'* változónév *'nach'* változónév '.'
: *'bilde'* függvénynév *'von'* változónév Képzés–1 '.'
: *'wenn'* feltétel *'dann'*
utasításblokk *'sonst'*
utasításblokk *'wende'* '.'
: *schleife'* ciklusnév *'ueber'*
utasításblokk '.'
: *'ausgang'* ciklusnév *'wenn'*
feltétel '.'
: *'fuer'* változónév *'von'* változónév *'bis'* változónév *'wiederhole'*
: *'sprungmarke'* cím '.'
: *'springe'* cím '.'
: *'rufe'* alprogramnév '.'

Ez tehát az Utasítás-szabály. Eddig még nem ismertettük az átvitel–1, a képzés–1, a feltétel, a karakterlánc, az egészszám és a kiírás–1 fogalmakat.

Átvitel-1 : : = Lebegőpontos szám *'nach'* változónév
: változónév *'nach'*
változónév

Képzés–1 : : = *'nach'* változónév
: L

Feltétel : : = változónév operátor
változónév

Operátor : : = '='
: '/='
: '<'

```
                    :   '<='
                    :   '>'
                    :   '>='
Karakterlánc        ::= Billentyűkarakter Karakterlánc
                    :L
```

A billentyűkaraktert már nem kell további fogalmakkal meghatározni, ezen egyszerűen a billentyűzetről bevihető összes karaktert értjük, kivéve az idézőjelet.

```
Egészszám           ::= Számjegy Egészszám–1
Egészszám–1         ::= Számjegy Egészszám–1
                    :L
Számjegy            ::= '0'
                    :   '1'
                    :   '2'
                    :   .
                    :   '9'
Lebegőpontos szám   ::= Egészszám Lebegőpontos szám–1
Lebegőpontos szám–1 ::= '.' Egészszám Lebegőpontos szám–2
                    : Exponenciális
                    ::L
Lebegőpontos szám–2 ::= Exponenciális
                    :L
Exponenciális       ::= 'e' Exponenciális–1
Exponenciális–1     ::= '+' Egészszám
                    :   '–' Egészszám
                    :   Egészszám
Kiírás–1 (itt szöveg) ::= '"' Karakterlánc '"'
                    : Változónév
```

Ezzel a MINIATUR programnyelv nyelvtanát leírtuk.

Ahhoz, hogy a nyelvtannal dolgozhassunk, először is sorszámokkal ellátva összefoglaljuk a nyelvtan szabályait, majd egy tárgymutatót mellékelünk.

4.1 A nyelvtan szabályai

```
01   Program                    :: = 'programm' programnév 'ist'
                                   Definíciós blokk 'beginne'
                                   Utasításblokk 'pende' programnév
                                   '.' alprogram EOF
02   Alprogram                  :: = 'unterprogramm' alprogramnév
                                   'ist' Utasításblokk 'uende' alprog-
                                   ramnév '.' alprogram
03                              : L
04   Név                        :: = Betű Név–1
04a  Név–1                      :: = Betű Név–1
05                              :    L
06   Betű                       :: = 'a'
                                :    'b'
                                :    'c'
                                :    .
                                :    .
                                :    'x'
                                :    'y'
                                :    'z'
07   Definíciós blokk           :: = Változódefiniálás
08                              :    L
09   Változódefiniálás          :: = 'fliess' változónév
                                   Változódefiniálás–2
                                   '.' Definíciós blokk.
10   Változódefiniálás–2        :: = ',' változónév
                                   Változódefiniálás–2
11                              :    L
12   Utasításblokk              :: = Utasítás Utasítássorozat
13   Utasítássorozat            :: = Utasítás Utasítássorozat
14                              :    L
15   Utasítás                   :: = 'leer' '.'
16                              : 'zeige' kiírás–1 '.'
17                              : 'zeigez' kiírás–1 '.'
18                              : 'vorschub' '.'
19                              : 'schirmfrei' '.'
20                              : 'cuspalte' egészszám '.'
21                              : 'cuzeile' egészszám '.'
22                              : 'zeigeas' egészszám'.'
```

```
23                                  : 'ausgabegeraet'
                                      készülékmegjelölés '.'
24                                  : 'rahmen' színmegjelölés '.'
25                                  : 'hintergrund' színmegjelölés '.'
26                                  : 'schrift' színmegjelölés '.'
27                                  : 'hole' változónév '.'
28                                  : 'uebertrage' átvitel–1 '.'
29                                  : 'addiere' változónév 'zu' változónév
                                      'nach' változónév '.'
30                                  : 'subtrahiere' változónév 'von' változó-
                                      név 'nach' változónév '.'
31                                  : 'multipliziere' változónév 'mit' válto-
                                      zónév 'nach' változónév '.'
32                                  : 'dividiere' változónév 'durch' változó-
                                      név 'nach' változónév '.'
33                                  : 'potenziere' változónév 'mit' változó-
                                      név 'nach' változónév '.'
34                                  : 'bilde' függvénynév 'von' változónév
                                      képzés–1 '.'
35                                  : 'wenn' feltétel 'dann' utasításblokk
                                      'sonst' utasításblokk 'wende' '.'
36                                  : 'schleife' ciklusnév 'ueber' utasítás-
                                      blokk
                                      'sende' ciklusnév '.'
37                                  : 'ausgang' ciklusnév
                                      'wenn' feltétel '.'
38                                  : 'fuer' változónév 'von' változónév
                                      'bis' változónév 'wiederhole'
                                      utasításblokk 'sende' '.'
39                                  : 'sprungmarke' cím'.'
40                                  : 'springe' cím'.'
41                                  : 'rufe' alprogramnév '.'
42  Átvitel–1                       :: = lebegőpontos szám 'nach' változó-
                                      név
43                                  : változónév 'nach' változónév '.'
44  Képzés–1                        :: = 'nach' változónév
45                                  : L
46  Feltétel                        :: = változónév operátor változónév
47  Operátor                        :: = '='
48                                  : '/='
49                                  : '<'
50                                  : '<='
51                                  : '>'
52                                  : '>='
```

```
53    Karakterlánc                  :: = karakter karekterlánc
54                                  : L
55    Egészszám                     :: = Számjegy Egészszám–1
55a   Egészszám–1                   :: = Számjegy Egészszám–1
56                                  : L
57    Számjegy                      :: = '0'
                                    : '1'
                                    : '2'
                                    : .
                                    : '9'
58    Lebegőpontos szám             :: = Egészszám Lebegőpontos szám–1
59    Lebegőpontos szám–1           :: = '.' Egészszám Lebegőpontos
                                       szám–2
60                                  : Exponenciális
61                                  : L
62    Lebegőpontos szám–2           :: = Exponenciális
63                                  : L
64    Exponenciális                 :: = 'e' Exponenciális–1
65    Exponenciális–1               :: = '+' Egészszám
66                                  : '–' Egészszám
67                                  : Egészszám
68    Kiírás–1 (itt szöveg)         :: ='"' Karakterlánc '"'
69                                  : változónév
```

4.2 A nyelvtan tárgymutatója

A sor elején a szabály kulcsszáma áll. Ez akkor jelenik meg, amikor a szintaktikai elemzés során a veremből (STACK) előkerül. De erről a későbbiekben bővebben lesz szó!

Az ezután következő szám a szabály sorszáma, ez megegyezik az előző fejezetben található sorszámokkal. A perjel után következő számok pedig azt közlik, hogy a nyelvtan mely szabályaiban szerepel az adott szabály.

01 Utasítás	15/12,3
02 Utasításblokk	12/1, 2, 35, 36, 38
03 Utasítássorozat	13/12, 13
04 Feltétel	46/35, 37
05 Név	4/1, 2, 4, 9, 10, 23, 24, 25, 26, 27, 29, 30, 31, 32, 33, 34, 36, 37, 38, 39, 40, 41, 42, 43, 44, 46
24 Név–1	4a/4, 4a
06 Képzés–1	44/34
07 Betű	6/4, 4a
08 Definíciós blokk	7/1
09 Exponenciális	64/60, 62
10 Exponenciális–1	65/64
11 Lebegőpontos szám	58/42
12 Lebegőpontos szám–1	59/58
13 Lebegőpontos szám–2	62/59
14 Egészszám	55/20, 21, 22, 58, 59, 65, 66, 67
23 Egészszám–1	55a/55, 55a
15 Operátor	47/46
16 Program	1/
17 Átvitel–1	42/28
18 Alprogram	2/1, 2
19 Változódefiniálás	9/7, 9
20 Változódefiniálás–2	10/9, 10
21 Karakterlánc	53/53, 68
25 Kiírás–1	68/16, 17
22 Számjegy	57/55
26 Billentyűkarakter	/53

4.3 Hogyan használja a fordító a nyelvtant?

Ebben és a következő két fejezetben azokat az előmunkálatokat szeretnénk bemutatni, amelyek ahhoz szükségesek, hogy a szintaktikai elemzést végző programot megírjuk. A szintaktikai elemzést végző programot PARSER-nek hívják. Egyelőre abból indulunk ki, hogy vizsgálandó MINIATUR programunk szintaktikailag hibátlan, és így csak az elemzés technikájával foglalkozunk. Majd ezután térünk rá a hibák kezelésére.

Amint már említettük, a PARSER a nyelvtan és a tárgymutató alapján működik. Ahhoz, hogy az elemzés módszerét megismerjük, végezzük el a következő kis program szintaktikai vizsgálatát:

```
10 programm a ist
20 beginne
30 leer
40 pende a.
```

Tudjuk, hogy minden MINIATUR programnak ki kell elégítenie a Program-szabályt, amelynek sorszáma a nyelvtanban 01-es. Ez a szabály a MINIATUR programot teljesen átfogja, legyen az akármilyen bonyolult is. A PARSER tehát először ezt a szabályt veszi elő, és megjegyzi. Felmerül természetesen az a kérdés, hogyan jegyzi meg a PARSER ezt a szabályt?

A PARSER a szabályt az ún. veremben tárolja. Következő kérdés: mi ez a verem? A verem működését talán a következő furcsa, és látszólag a verem szóval nagyon is nem összeillő példán érthetjük meg a legkönnyebben. Képzeljük el egy felhőkarcoló liftjét, amely a következő elven működik: a lift kabinjában csak két gomb van, az 1-es gomb: egy emeletnyit fel, a 2-es gomb: egy emeletnyit le.

A liftezés kiindulópontja a 0 szint. Innen mindig egy-egy emelettel feljebb tudunk menni, és minden emeleten információt helyezhetünk el. Ezt az otthagyott információt elfelejtjük, amint azt az emeletet, ahol éppen vagyunk, elhagyjuk. Amint magasabbra megyünk, ez az információ az előbb elhagyott emeleten megvan, de csak akkor tudunk hozzáférni, ha visszamegyünk erre az emeletre. A magasabb emeleteken elhelyezett információk számunkra nem léteznek. Egy emelettel feljebb csak akkor mehetünk, ha azon az emeleten, ahol éppen vagyunk, információt helyeztünk el. Így minden alattunk lévő emeleten információk vannak.

Ez a szerkezet azt a célt szolgálja, hogy megtaláljunk olyan információkat, amelyekre egy későbbi időpontban szükségünk lesz.

Gondoljuk most meg a 01-es Program-szabályt! Talán meglepően hat, de a PARSER a szabályokat jobbról balra haladva jegyzi meg.

Emelet	Információ
0 szint	EOF
1 szint	alprogram
2 szint	'.'
3 szint	programnév
4 szint	*'pende'*
5 szint	utasításblokk
6 szint	*'beginne'*
7 szint	definíciós blokk
8 szint	*'ist'*
9 szint	programnév
10 szint	*'programm'*

Ha a 10. szinten vagyunk, akkor most a *'programm'* információval rendelkezünk. Mihez kezdünk ezzel az információval?

Tudjuk, hogy a *'programm'* terminális szimbólum. Tudjuk továbbá azt is, hogy minden MINIATUR programnak a *programm* kulcsszóval kell kezdődnie. Ezt éppen onnan tudjuk, hogy a legfelső szinten lévő információ terminális szimbólum.

Gondolkodjunk el ezeken a mondatokon! E mondatok tartalmazzák ugyanis a szintaktikai elemzés lényegét.

4.4 A szintaktikai elemzés stratégiája

Az elemzés előkészületei során a PARSER megjegyezte az előbbi szabályt, és a SCANNER utasítást kapott arra, hogy a MINIATUR program első tokenjét előkeresse. Programunk első tokenje a *programm* szó. A PARSER-nak most semmi mást nem kell tennie, mint a *'programm'* terminális szimbólumot a *programm* tokennel össze kell hasonlítania. Minthogy a két információ azonos, a PARSER most már tudja, hogy a MINIATUR programunk a *programm* szóig megfelel a MINIATUR szintaktikai szabályainak. Ezért a *programm* tokent elfelejtheti és a következő tokenért fordul a SCANNER-hez. Megkapja a program nevét, az a-t. Ekkor egy szinttel lejjebb megy, mivel a *programm* információ feleslegessé vált, és a program további alakulására vonatkozó információra van szüksége. Elolvassa tehát a program nevét, a Név szintaktikai értelmének megfelelően. A Név azonban nem terminális szimbólum, így azt nem tudja a-val összehasonlítani. A Név egy szintaktikai szabályt jelent.

Milyen stratégiát alkalmaz most a PARSER? A Név szimbólumot rendre a megfelelő szabályokkal helyettesíti mindaddig, amíg a verem legfelső szintjén ismét terminális szimbólum nem jelenik meg.

A Név szabálya:

```
04 Név: : = Betű Név
05      : L
```

Így tehát a PARSER-nak két lehetősége van: vagy a Betű Név, vagy az L alternatívát választja. Milyen alapon dönt? Minden alternatívához tartozik egy lista, amelyben megtalálható, hogy az aktuális tokenhez melyik alternatívát kell választani. Hogy miképp jut a PARSER ezekhez a listákhoz, azt majd később ismertetjük. Egyelőre azonban higgyük el, hogy a PARSER helyesen választ.

A PARSER tehát a 04-es sorszámú Betű Név szabályt választja a nyelvtanból. Törli a 9-es szintet, ahol eddig a programnév volt, és itt, majd a 10-es szinten az alábbi kombinációt helyezi el:

9-es szint	Név

majd következik a:

10-es szint	Betű

A verem legfelső szintjén tehát most Betű van. Betűn az összes kisbetűt értjük, és mivel az a kisbetű, a PARSER a betű helyére az a-t teheti és összehasonlít-

hatja az a tokennel. Minthogy a kettő megegyezik, kéri a következő tokent és egy szinttel lejjebb megy. A következő token az *'ist'*. A 9-es szinten a Név információ található. A PARSER-nak ekkor ismét választania kell a Betű Név és az L alternatíva között. Ezúttal az L alternatívát választja. Ez azonban törlést, ill. az egy szinttel lejjebb való keresést jelenti. A 8-as szinten a PARSER az *'ist'* információt találja, ami terminális szimbólum. Következik tehát a tokennel való összehasonlítás. Minthogy a kettő megegyezik, kéri a következő tokent és így tovább.

Az itt leírtak mindaddig folytatódnak, amíg a SCANNER az END OF FILE (vége a MINIATUR programnak) információt nem jelenti, és a PARSER a 0-ás szinten az egyezést meg nem állapítja.

Ekkor elmondható, hogy a MINIATUR program szintaktikailag helyes, és ezzel a PARSER a feladatát befejezte.

Reméljük, hogy ezzel érthetővé vált, hogyan működik a PARSER. Javasoljuk, hogy az olvasó próbálja meg a program hátralévő részén a szintaktikai elemzést elvégezni (tehát ahol mi az – és így tovább – -bal befejeztük, ott tessék folytatni). Ha egy szabálynál nem tudjuk, hogy melyik alternatívát válasszuk, akkor próbáljuk ki a lehetőségeket és döntsük el magunk.

Az olvasó biztosan észrevette, hogy az előbb a Név-szabályt nem úgy bontottuk ki, ahogyan korábban a nyelvtan összeállításakor azt meghatároztuk. Javasoljuk, hogy az olvasó gyakorlásul vegye elő az eredeti, tehát a nyelvtan összeállításakor meghatározott szabályt, és vezesse végig az előbbi esetet ilyen módon is

Három kérdés maradt még nyitva.

- Hogyan jutunk ahhoz a listához, amelyik az alternatívák kiválasztását vezérli?
- Hogyan kezeli a PARSER a hibás programokat?
- Mit kell a PARSER-nak tennie ahhoz, hogy a programokat gépi nyelvre lehessen lefordítani?

4.5 Az alternatívák kiválasztása

A PARSER az elemzésnél a következőképpen jár el: a verem legmagasabb szintjén lévő információ alapján kikeresi azt a szabályt, amelyet alkalmaznia kell. Ezután megnézi magának az aktuális tokent és egy lista alapján kiválasztja a megfelelő alternatívát.

Hogyan jutunk ilyen listához?

Programjaink tokenjei mindig terminális szimbólumok. Minden alternatívához ki kell keresnünk azokat a terminális szimbólumokat, amelyek az adott alternatíva alkalmazásánál elsőként fordulhatnak elő. Ez nagyon egyszerűnek látszik, de e feladat elvégzése hosszadalmas is lehet, különösen, ha egy bonyolultabb nyelvtant kell feldolgoznunk, ahol csak nagyon kevés alternatíva kezdődik terminális szimbólummal. Egyes könyvekben találhatunk ugyan olyan előírásokat, amelyek sikerrel alkalmazhatóak, de ezek lényegében csupán ötletek egy olyan program megírásához, amely a megfelelő terminális szimbólumokat megtalálja. Az ilyen előírások alapján legtöbbször nem lehet felismerni, hogy mi a cél, és így sok gépies és fölösleges munkát végez az ember. Ennek elkerülésére minden egyes alternatívánál megvizsgáljuk, hogy milyen terminális szimbólumok fordulhatnak elő.

Kezdjük el, és a teljesség kedvéért nézzük meg a 01 Program-szabályt is, bár ezt a szabályt sohasem kell alternatívaként választanunk, mert a szintaktikai elemzés kezdetén már megjelenik.

Alternatíva	*Terminális szimbólum*
01	program
02	alprogram
03	EOF

Ha egy alternatíva üres, akkor meg kell nézzük, hogy milyen szabályokban található az adott szabály, és melyek lehetnek azok az első terminális szimbólumok, amelyek azt esetlegesen követik. Pl.: az Alprogram-szabály csak a Program-szabályban található. Az egyetlen terminális szimbólum, ami ezután jöhet, az az EOF cím. Ebből az következik, hogyha a legmagasabb szintű verem információ az Alprogram és az aktuális token az *unterprogramm*, akkor a 02-es alternatívát választjuk.

Ha a legmagasabb szintű verem információ az Alprogram és az aktuális token az EOF, akkor a 03-as szabályt választjuk. Tovább:

04 Betű
04a Betű
05 ist, '.' '.', *zu, nach, von, mit, durch, ueber, wenn, bis, wiederhole* '=', '/=', '<', '<=', '>', '>=', *dann.*

Kérjük az Olvasót, próbálja ki, hogyan jutottunk el a 05-ös szabályhoz tartozó terminális szimbólumokhoz! Használja fel ehhez a már előbb ismertetett tárgymutatót.

06 betű
07 *fliess*
08 *beginne*
09 *fliess*
10 ','
11 '.'
12 *leer, zeige, zeigez, vorschub, schirmfrei, cuspalte, cuzeile, zeigeas, ausgabegeraet, rahmen, hintergrund, schrift, hole, uebertrage, addiere, subtrahiere, multipliziere, dividiere, potenziere, bilde, wenn, schleife, ausgang, fuer, sprungmarke, springe, rufe*
13 lásd 12
14 *pende, uende, sonst, wende, '.', sende*
15 *leer*
16 *zeige*
17 *zeigez*
18 *vorschub*
19 *schirmfrei*
20 *cuspalte*
21 *cuzeile*
22 *zeigeas*
23 *ausgabegeraet*
24 *rahmen*
25 *hintergrund*
26 *schrift*
27 *hole*
28 *uebertrage*
29 *addiere*
30 *subtrahiere*
31 *multipliziere*
32 *dividiere*
33 *potenziere*
34 *bilde*
35 *wenn*
36 *schleife*
37 *ausgang*
38 *fuer*

39 *sprungmarke*
40 *springe*
41 *rufe*
42 számjegy
43 betű
44 *nach*
45 '.'
46 betű
47 ·=·
48 ·/=·
49 ·<·
53 billentyűkarakter, kivéve az idézőjel
54 ·,,·
55 számjegy
55a számjegy
56 ·.·'e', *nach*
57 számjegy
58 számjegy
59 ·.·
60 'e'
61 *nach*
62 'e'
63 *nach*
64 'e'
65 '+'
66 '–'
67 számjegy
68 '"'
69 betű

Az ilyen listák összeállítása nehéznek, bonyolultnak tűnik, pedig valójában egészen egyszerű. Szeretnénk egy példával még egyszer ezt bemutatni.

Vegyük az 56-os alternatívát (lásd a nyelvtan szabályait!)

Ez az alternatíva üres, így felmerül a kérdés: mi következhet az Egészszám–1-szabály után (55a)? Az Egészszám–1-szabály csak az Egészszám-szabályban fordul elő, éspedig utolsó elemként. Mi következhet az Egészszám-szabály után? A tárgymutató szerint az Egészszám-szabály a 20, 21, 58, 59, 65, 66, 67-es alternatívákban fordul elő. A 20, 21 és a 22-es alternatívákban az Egészszám-szabály után pont következik.

Az 58-as szabály az Egészszám-szabály a Lebegőpontos szám–1 szabály előtt áll. Az Egészszám-szabály után következők tehát ebből az utóbbi szabályból adódnak, mint az ott előforduló első terminális szimbólumok. A Lebegőpontos szám–1-szabály a 61-es üres alternatívát és a Lebegőpontos szám–1-szabály további terminális szimbólumait is magában foglalja.

A Lebegőpontos szám–1-szabály első terminális szimbólumai a '.' és az Exponenciális-szabály első szimbóluma az 'e'. A Lebegőpontos szám–1-szabály további terminális szimbólumai a Lebegőpontos szám-szabály terminális szimbólumaiból adódnak, mert a Lebegőpontos szám–1-szabály csak a Lebegőpontos szám-szabály utolsó elemeként fordul elő. A Lebegőpontos szám-szabály azonban csak a 42-es alternatívában fordul elő, ahol is az utánakövetkező terminális szimbólum a *'nach'*.

Eddig tehát a '.' az 'e' és a *'nach'* terminális szimbólumokhoz jutottunk.

Meg kell még vizsgálni a 65-ös, a 66-os és a 67-es alternatívákat is. Mindhárom Egészszám-szabállyal végződik és az Exponenciális–1 szabályhoz tartozik.

Kérdés: Melyek az Exponenciális–1-szabály utánakövetkező terminális szimbólumai?
Válasz: az Exponenciális-szabály utánakövetkező terminális szimbólumai!
Kérdés: Melyek az Exponenciális-szabály utánakövetkező terminális szimbólumai?
Válasz: a Lebegőpontos szám–2 és a Lebegőpontos szám–1 szabályok utánakövetkező terminális szimbólumai!
Kérdés: Melyek a Lebegőpontos szám–2-szabály utánakövetkező terminális szimbólumai?
Válasz: a Lebegőpontos szám–1-szabály utánakövetkező terminális szimbólumai!

A Lebegőpontos szám–1-szabály utánakövetkező terminális szimbólumaként már az előbb meghatároztuk a *'nach'*-ot.

Így állítjuk tehát össze az alternatívák listáját. Az összeállítás közben azonban nyilvánvalóvá válik, hogy bonyolultabb és hosszabb szerkezetű nyelvtanok esetében gondolkodni kell olyan technikák kialakításán, amelyekkel hiba nélkül célhoz tudunk érni. Az eddigi fejtegetések során eddig nem említettük, hogy a PARSER csak akkor tudja pontosan, hogy melyik alternatívát kell választania, ha a listák egy szabály alternatíváihoz tartozó összes különböző tokeneket tartalmazzák.

Ez egyébként a nyelvtan egyik legfontosabb próbája. Ha egy szabály alternatíváinak a listáiban ugyanazok a tokenek fordulnak elő, akkor vagy a nyelvtant kell megváltoztatnunk, vagy más szintaktikai elemzési módszert kell keresnünk.

4.6 A hibás programok vizsgálata

Eddig abból indultunk ki, hogy MINIATUR programunk szintaktikailag hibátlan. Ha a programban szintaktikai hibák fordulnak elő, akkor ezek kezelésére két lehetőségünk adódik:

1. kiadjuk a megtalált hibát, és megszakítjuk a szintaktikai elemzést;
2. megkíséreljük a szintaktikai elemzés folytatását; hogy a program további részeit is megvizsgálhassuk.

Az első lehetőséget elfogadhatatlannak tartjuk. Képzeljük el, hogy egy programban pl. 20 hiba fordul elő, ekkor legalább 21 próbálkozásra van szükségünk ahhoz, hogy szintaktikailag hibátlan programot nyerjünk.

Foglalkozzunk tehát a második lehetőséggel. Hogyan veszi észre a PARSER, hogy egy program hibás? Ez is kétféleképpen lehetséges:

a) a legfelső (legmagasabb szintű) vereminformációnak terminális szimbólumnak kell lennie, az aktuális token pedig nem egyezik meg a terminális szimbólummal, pl.: a gépelésnél hibáztunk és programot írtunk a *programm* kulcsszó helyett,
b) a PARSER egy szabálynál keresi a megfelelő alternatívát és egyik listában sem találja az aktuális tokent. Pl.: elfelejtettünk a programnak nevet adni; *programm ist* ––––––, az aktuális token ekkor az *ist*, a PARSER azonban nevet, azaz egy betűt vár.

Nézzük meg, milyen stratégiát követhetünk az *a)* és a *b)* esetben!

A vizsgálandó program egy részét nem tudjuk szintaktikailag elemezni, mert a vereminformáció a program szerkezetével nem egyezik meg. Meg kell próbálnunk a vermet és a programot újra úgy megfeleltetni egymásnak, hogy az elemzést folytatni lehessen.

Az első esetben pontosan tudjuk, hogy a PARSER milyen karaktert, ill. szimbólumot várt, és mit talált. Így nagyon pontos hibajelzést adhatunk. Emellett megpróbálhatjuk a várt szimbólumot a programban megkeresni, amennyiben a SCANNER-től új tokent kérünk. Gépelési hibánál azonban ez nem mindig sikerül, és esetleg nem találjuk meg a keresett szimbólumot. Ennek elkerülésére legfeljebb csak a legközelebbi '.'-ig keressünk, mivel a pont a MINIATUR-ban utasításokat zár le, amelyeket kis szintaktikai egységeknek tekinthetünk. Ezután már csak a vermet kell megfelelően egyeztetni, úgy, hogy azt felülről kezdve töröljük mindaddig, amíg újra '.'-hoz érünk. Ezzel a verem és a program szinkronizálása megtörtént, az elemzés folytatódhat.

A második esetben megadhatjuk azon tokenek listáját, amelyek a szintaktikai elemzés továbbvitelét segíthetik. Ekkor a következőképpen járunk el: a veremben lefelé haladunk, és keressük a legközelebbi terminális szimbólumot, majd továbbolvassuk programunkat ahhoz, hogy megállapíthassuk, előfordul-e ez a terminális szimbólum. A programot azonban csupán a legközelebbi pontig olvassuk. A veremben is addig haladunk lefelé, amíg ponthoz nem érünk. Így újra egyezést értünk el a verem és a program között.

Lehet, hogy az olvasónak az a véleménye, hogy ezek nagyon durva stratégiák, és ráadásul nem mindig vezetnek sikerekhez. Ezenkívül azt a lehetőséget is figyelmen kívül hagytuk, hogy a számítógép esetleg önálló hibajavítást is végezhetne. Tulajdonképpen az Olvasónak igaza van, de a hibakezelést majd úgy helyezzük el a programban, hogy mindenki saját ötletei alapján is tudjon hibakezelést bővíteni, ha van elég tárkapacitása. Vannak olyan fordítóprogramok, amelyekben a hibajavítás kb. ugyanakkora terjedelmű, mint maga az elemzés. Egy kisszámítógépnél azonban kompromisszumokat kell tennünk.

Most már minden MINIATUR programról megállapíthatjuk, hogy szintaktikailag hibátlan-e. A szintaktikai elemzés után a szemantikai vizsgálat és a kódgenerálás következik. Ahhoz, hogy a szemantikai vizsgálat egyáltalán végrehajtható legyen, a szintaktikai elemzés során információkat kell a szemantikai vizsgálatra továbbítanunk. Minthogy a szintaktikai elemzés során a program szerkezetét megismertük, használjuk ki ezt az információt is! A program szerkezetét a nyelvtan határozza meg, és a PARSER ellenőrzi. Használjuk tehát a nyelvtant információs forrásul. Ehhez a nyelvtant olyan szimbólumokkal egészítjük ki, amelyek abba beilleszthetők, de a szintaktikai elemzést nem befolyásolják. Ezek a szemantikai szimbólumok külön jelölést kapnak. Ha egy szemantikai szimbólum legfelső (legmagasabb szintű) vereminformációként jelenik meg, akkor a PARSER ezt felismeri, és egy olyan alprogramot hív, amely azt fel is tudja dolgozni. Ezt követően ezt a szimbólumot kivisszük a veremből és a PARSER folytatja a munkáját. A szemantikai szimbólum alprogramjai információkat gyűjtenek és adnak ki. Az információk ugyanarra az adathordozóra mennek ki, amelyre a sorok számát is írtuk. Ezen az adathordozón tároljuk mindazokat az információkat, amelyekre szükség van ahhoz, hogy MINIATUR programunkat gépi kódú programmá alakítsuk. Pontosan milyen információkat is akarunk kiadni? Ehhez még egyszer végig kell mennünk a nyelvtanon, és minden alternatíva esetére végig kell gondolnunk, hogy mit is fogunk csinálni.

A szemantikai szimbólumokat kerek zárójelbe tesszük.

01 Programm :: = *'programm'* (programnév definíció) programnév *'ist'* (a definíciós rész kezdete)
definíciós blokk (a definíciós rész vége)
'beginne' utasításblokk *'pende'* (programnév elemzés)
programnév'.' alprogram (program vége) EOF

A fenti szabály feldolgozásával meghatározzuk a programnevet, és az információkat, valamint a megfelelő nevet kiadjuk (lásd 04, 05!). A szemantikai vizsgálat során azután ezt a nevet felvesszük egy listára és programnévként tartjuk számon. A definíciós rész kezdete szimbólumnak az a célja, hogy minden következő név rákerüljön a listára. A definíciós rész vége szimbólum után már nem szabad további változókat meghatározni, és minden eddig előforduló változónak már szerepelnie kell a listán.

A programnév vizsgálat után a következő névnek a listán programnévként kell megjelennie.

A szemantikai vizsgálatot és a kódgenerálást a következő fejezetben fogjuk részletesen tárgyalni.

Bővítsük ki a nyelvtant ahhoz, hogy a PARSER programját megírhassuk.

```
02 Alprogram                    :: = 'unterprogramm' (alprogramnév definíció
                                     alprogramnév
                                     'ist' utasításblokk 'uende' (alprogramnév vizsgálat)
                                     alprogramnév '.' alprogram
03                               : L
04 Név                          :: = (Betűt megjegyezni) Betű Név–1
04a Név–1                       :: = (Betűt megjegyezni) Betű Név–1
05                               : (nevet kiadni) L
06 Betű                         :: = 'a', 'b',....., 'z'
07 Definíciós blokk             :: = Változódefiniálás
08                               : L
09 Változódefiniálás            :: = 'fliess' lebegőpontos változónév
                                     Változódefiniálás–2
                                     '.' Definíciós blokk
10 Változódefiniálás–2          :: = ',' változónév
                                     Változódefiniálás–2
11                               : L
12 Utasításblokk                :: = Utasítás Utasítássorozat
13 Utasítássorozat              :: = Utasítás Utasítássorozat
14                               : L
15 Utasítás                     :: = 'leer' '.'
16                               : (kivitel) 'zeige' kiírás–1 '.'
17                               : (kiírás soremeléssel) 'zeigez' kiírás–1 '.'
18                               : 'vorschub' '.'
19                               : (képernyőtörlés) 'schirmfrei' '.'
20                               : (kurzoroszlop) 'cuspalte' '.'
21                               : (kurzorsor) 'cuzeile' Egészszám '.'
```

```
22                  : (kiírás ASCII kódban) 'zeigeas'
                      egészszám '.'
23                  : (kiviteli készülék) 'ausgabegeraet'
                      készüléknév '.'
24                  : (keret) 'rahmen' színnév '.'
25                  : (háttér) 'hintergrund' színnév '.'
26                  : (írás) 'schrift' színnév '.'
27                  : (bevitel) 'hole' változónév '.'
28                  : (átvitel) 'uebertrage' átvitel–1 '.'
29                  : (összeadás) 'addiere' változónév
                      'zu' változónév 'nach' változó-
                      név '.'
30                  : (kivonás) 'subtrahiere' változónév
                      'von' változónév '.'
31                  : (szorzás) 'multipliziere' változónév
                      'mit' változónév 'nach' változó-
                      név '.'
32                  : (osztás) 'dividiere' változónév
                      'durch' változónév '.'
33                  : (hatványozás) 'potenziere' változó-
                      név 'mit' változónév 'nach' válto-
                      zónév '.'
34                  : (képzés) 'bilde' (függvénynév 'von'
                      függvénynév
                      képzés–1 '.'
35                  : (döntés) 'wenn' feltétel 'dann' uta-
                      sításblokk 'sonst' utasításblokk
                      'wende' (feltételes ciklus vége)
                      '.'
36                  : 'schleife' (ciklus)
                      'ueber' utasításblokk 'sende'
                      (ciklus vége)
                      ciklusnév '.'
37                  : (kilépés) 'ausgang' ciklusnév
                      'wenn' feltétel '.'
38                  : (paraméteres ciklusutasítás érték-
                      adásának kezdete) 'fuer' válto-
                      zónév 'von' változónév 'bis' vál-
                      tozónév 'wiederhole' (paramété-
                      res ciklusutasítás kezdete)
39                  : (ugrás) 'sprungmarke' cím '.'
40                  : (ugrás) 'springe' cím '.'
41                  : (hívás) 'rufe' alprogramnév '.'
42 átvitel–1        :: = lebegőpontos szám 'nach' válto-
                      zónév
```

```
43                                  : (változó) változónév 'nach' változó-
                                      név
44 képzés–1                         :: = (képzés–1) 'nach' változónév
45                                  : L
46 feltétel                         :: = változónév operátor változónév
47 operátor                         :: = (operátor=) '='
48                                  : (operátor/=) '/='
49                                  : (operátor) '<'
50                                  : (operátor=) '<='
51                                  : (operátor) '>'
52                                  : (operátor=) '>='
53 karakterlánc                     :: = (karaktert megjegyezni) billen-
                                      tyűkarakter karakterlánc
54                                  : (karakterláncot kivinni) L
55 egészszám                        :: = (számjegyet megjegyezni)
                                      számjegy egészszám–1
55a egészszám–1                     :: = (számjegyet megjegyezni)
                                      számjegy egészszám–1
56                                  : (egészszámot kivinni) L
57 számjegy                         :: = '0', '1', '2',·····'9'
58 lebegőpontos szám                :: = egészszám lebegőpontos
                                      szám–1
59 lebegőpontos szám–1              :: = (tizedes helyek) '.' egészszám
                                      lebegőpontos szám–2
60                                  : exponenciális
61                                  : (nincs exponenciális) L
62 lebegőpontos szám–2              :: = exponenciális
63                                  : (nincs exponenciális) L
64 exponenciális                    :: = (exponenciális) 'e' exponenciá-
                                      lis–1
65 exponenciális–1                  :: = (plusz) '+' egészszám
66                                  : (mínusz) '–' egészszám
67                                  : egészszám
68 kiírás–1 (itt szöveg)            :: = '"' karakterlánc '"'
69                                  : változónév
```

A PARSER-rel ilyen formában nem közölhetjük a nyelvtant. A közléshez a terminális, a nem terminális és a szemantikai szimbólumokat kódolni fogjuk.

A terminális szimbólumok számára a számokat *A lexikális elemzés* c. fejezetből vesszük, a nem terminálisokhoz a nyelvtan tárgymutatójából és a szemantikai szimbólumokat pedig az alábbiakban kódoljuk.

Kód	*Szimbólum*
1	programnév definíció
2	definíciós rész kezdete
3	definíciós rész vége
4	programnév vizsgálat
5	program vége
6	alprogramnév definíció
7	alprogramnév vizsgálat
8	kivitel
9	kivitel soremeléssel
10	soremelés
11	képernyőtörlés
12	kurzoroszlop
13	kurzorsor
14	kivitel ASCII kódban
15	kiviteli készülék
16	keret
17	háttér
18	írás
19	bevitel
20	átvitel
21	összeadás
22	kivonás
23	szorzás
24	osztás
25	hatványozás
26	képzés
27	pont
28	feltétel
29	különben
30	feltételes ciklus vége
31	ciklus
32	ciklus vége
33	kilépés
34	paraméteres ciklus
35	cikluskezdet
36	paraméteres ciklus vége
37	cím
38	ugrás
39	hívás
40	lebegőpontos szám
41	változó
42	képzés–1
43	operátor =
44	operátor /=

45		operátor <
46		operátor < =
47		operátor >
48		operátor > =
49		tizedeshelyek
50		nincs exponenciális
51		exponenciális
52		plusz
53	-	mínusz
54		betűt megjegyezni
55		nevet kiadni
56		karaktert megjegyezni
57		karakterláncot kiadni
58		számjegyet megjegyezni
59		egészszámot kiadni

Ahhoz, hogy a PARSER mindig el tudja dönteni, hogy éppen milyen szimbólum, ill. információ található a verem legfelső szintjén, a következőképpen járunk el.

A terminális szimbólumok számok között kerülnek tárolásra.

A nem terminális szimbólumokat két szinten tároljuk:

az alsó szint a szimbólum számát tárolja,
a felső szint a 253-as kulcsszámot tartalmazza.

A szemantikai szimbólumok az 53-as számmal bezárólag eltolásként a 252-es kulcsszámot kapják, az 54-es számtól kezdve pedig a 251-es kulcsszámot. 53-ig terjed a szám kiadása és 54-től bizonyos járulékos tevékenységek szimbólumai tárolódnak.

Ezzel a kódolással a 01-es szabály tehát a következőképpen alakul:

148, 252, 1, 253, 5, 141, 252, 2, 253, 8, 252, 3, 131, 253, 2, 146, 252, 4, 253, 5, 46, 253, 18, 252, 5, 254.

4.7 A Parser

Most jutottunk el odáig, hogy összeállíthatjuk a PARSER programot. Minthogy a lexikális elemzésnél minden lépést elmagyaráztunk, a következő programoknál kevesebb megjegyzést fogunk tenni.

A nyelvtan minden szabályát egy karakterlánc tartományba helyeztük el úgy, hogy egy szabály alkalmazásakor csupán a megfelelő változótartalmat kell a verembe átvinnünk. Mivel az egyes füzérváltozók értékadását nem tudjuk közvetlenül elvégezni, az alternatívákat adatsorokba helyezzük el, és ezután visszük át ezek tartalmát a változókba.

A nyelvtan a 49 000-estől az 50 000-esig terjedő sorokban található. A 10 040-estől a 10 670-es sorig valósul meg a hibakeresés.

A PARSER hibát talált, és a korábbiakban leírt stratégiának megfelelően működik. A verem legfelső szintjén található szimbólum tartalmazza azt az információt, mely megmutatja, hogy mely szimbólumok jöhetnek szóba.

A 11 000-estől a 14 630-as sorokban a PARSER azt vizsgálja, hogy melyik alternatívát kell választania.

Kérjük az Olvasót, kövesse a PARSER működését a legkisebb MINIATUR program esetében, és kiderül, hogy a programelemzés milyen egyszerűvé vált.

P.1

```
a Parser listaja

10 rem miniatur-programok
20 rem lexikai es szintaktikai
30 rem elemzo programja
40 rem --
50 rem definiciok
60 rem --
70 :
80 fl=0
90 be=15
100 in=8
110 ou=7
120 pr=3
130 i1$="" : i2$="" : i3$=""
140 i1=0   : i2=0   : i3=0
150 en=0 :em$="" : et=0 : es=0
160 dim ba$(75) : rem basic alapszavak
```

```
170 dim mi$(48) : rem miniatur alaPszavak
180 t$="" : t=0
190 a=0 : b=0 : c=0 : d=0
200 dim al$(69) : rem nyelvtani szabaly
210 z=0 : z%=0 : rem lokalis valtozok
220 s9=1000 : rem verem meret
230 dim s%(s9) : rem verem
240 s=0 : rem verem mutato
250 al=1 : rem ****** Pillanatnyi alternativa mutatoja
260 al%=0 : rem az a1 hossza
270 t9=0 : rem lokalis ciklusindex
280 tl=1 : 9$="" : 9=0
290 o=0 : rem le9felso verem szimbolum
300 bu$="" : rem munkavaltozo
310 fe=0 : rem hibavaltozo
320 zn=0 : rem sorkezdet jelzo
1000 rem --
1010 rem elokeszuletek
1020 rem --
1030 :
1040 Print"■"
1050 Print:Print"MINIATUR-fordito :":Print
1060 Print"A Protokoll nyomtatora va9y kePernyore  kerul
ljon (n/k)?"
1070 9et i1$ : if i1$="n" or i1$="k" then 1090
1080 9oto 1070
1090 if i1$="n" then Pr=4
1100 oPen Pr,Pr,7 : rem csatorna me9nyitasa
1110 Print:Print"Kerem az adatlemezt!"
1120 Print"<RETURN>":Print
1130 9et i1$ : if i1$<>chr$(13) then 1130
1140 Print"Melyik Pror9ramot akarja forditani?"
1150 inPut"nev ";i2$
1160 oPen be,fl,be
1170 Print#be,"i"
1180 Print#be,"s:miniatur-syn"
1190 inPut#be,en,em$,et,es
1210 oPen in,fl,in,"0:"+i2$+",P,r" : 9osub 60000
1220 9et#in,i1$,i1$
1230 oPen ou,fl,ou,"0:miniatur-syn,s,w" : 9osub 60000
1240 9osub 59000
1245 9osub 58000
1250 9osub 51500 : 9osub 51500 : 9osub 51000
1260 9osub 49000 : rem nyeltan hivasa
1270 9osub 42000 : rem elso karakter allitasa
1280 9osub 41000 : rem elso lehetose9 hivasa
2000 rem Parser hurok
2010 o=s%(s)
```

```
2020 al=255
2030 if o=251 then goto 38000
2040 if o=252 then goto 39000
2050 if o=253 then goto 10000
2060 if s%(s)=255 and g=255 then goto 35000
2070 if s%(s)=g then gosub 40000 : gosub 42000 : goto 2
000
2080 Print#Pr,chr$(13)"Hiba!"
2085 if s%(s)=255 then Print#Pr,"EOF-ra varok."
2086 if s%(s)=255 then Print#Pr,"Befejeztem a Prog ramot
." : goto 35000
2090 if s%(s)>127 then Print#Pr,"A kovetkezo alaPszot v
arom : ";mi$(s%(s)-128)
2100 if s%(s)<128 then Print#Pr,"A kovetkezo kodot varo
m : ";s
2110 goto 10400
10000 rem alternativak keresese
10010 o=s%(s-1)
10020 gosub 11000
10025 if al=255 then goto 10040
10030 if len(al$(al))<>0 then gosub 40000 : gosub 41000
 : goto 2000
10035 gosub 41000 : goto 2000
10040 Print#Pr,chr$(13)
10050 Print#Pr,"Hiba !"
10060 if o=1oro=2oro=3 then Print#Pr,"Egyik alternativa
 sem lehetseges !"
10070 if o=1oro=2oro=3 then Print#Pr,"A kovetkezo szimb
olumok egyiket varom :"
10080 if o=1 oro=2 or o=3then Print#Pr,"leer zeige zeig
ez vorschub schirmfrei ";
10090 if o=1oro=2 or o=3then Print#Pr,"cusPalte cuzeile
 zeigeas ausgabegeraet ";
10100 if o=1 or o=2 or o=3 then Print#Pr,"rahmen hinter
grund schrift hole ";
10110 if o=1 or o=2 or o=3 then Print#Pr,"uebertrage ad
diere subtrahiere ";
10120 if o=1 or o=2 or o=3 then Print#Pr,"multiPliziere
 dividiere Potenziere ";
10130 if o=1 or o=2 or o=3 then Print#Pr,"bilde wenn sc
hleife ausgang fuer ";
10140 if o=1 or o=2 or o=3 then Print#Pr,"sPrungmarke s
pringe rufe ";
10150 if o=4 or o=5 or o=6 or o=17 or o=24 or o=25 then
 Print#Pr,"betu ";
10160 if o=24 or o=15 then Print#Pr,"= /= < <= > >= ";
10170 if o=24 then Print#Pr,"ist zu von mit durch ueber
 wenn bis ";
```

```
10180 if o=24 then Print#Pr,"wiederhole dann ";
10190 if o=24 or o=20 then Print#Pr,", ";
10200 if o=24 or o=20 or o=3 or o=6 or o=23 or o=12 the
n Print#Pr,". "
10210 if o=16 then Print#Pr,"Pro9ramm ";
10220 if o=18 then Print#Pr,"unterPro9ramm eof ";
10230 if o=7 or o=19 then Print#Pr,"fliess ";
10240 if o=8 then Print#Pr,"be9inne ";
10250 if o=17 or o=14 or o=23 or o=22 or o=11 or o=10 t
hen Print#Pr,"szamje9y ";
10260 if o=6 or o=23 or o=12 or o=13 then Print#Pr,"nac
h ";
10270 if o=3 then Print#Pr," Pende uende sonst uende se
nde ";
10280 if o=21 then Print#Pr,"billentyukarakter idezojel
 nelkul ";
10290 if o=21 or o=25 then Print#Pr,"idezojel ";
10300 if o=23 or o=12 or o=13 or o=9 then Print#Pr,"e "
;
10310 if o=10 then Print#Pr,"+ - ";
10400 fe=fe+1
10405 Print#Pr,
10410 Print#Pr,"A tovabbiakban a kovetkezo alapszot
 varom : ";
10420 if s%(s)>250 and s>=2 then s=s-2 : 9oto 10420
10430 if s<=0 then 35000
10440 if s%(s)>127 then Print#Pr,"  ";mi$(s%(s)-128)
10450 if s%(s)<128 then Print#Pr,"  ";s
10460 Print#Pr,"<RETURN>"
10470 9et em$ : if em$<>chr$(13) then 10470
10480 Print#Pr,"Me9nezem, van-e a Pro9ramban ";
10490 if s%(s)>127 then Print#Pr,mi$(s%(s)-128);
10500 if s%(s)<128 then Print#Pr,s;
10510 Print#Pr,"."
10520 Print#Pr,"Le9feljebb a kovetkezo Ponti9 vag9 "
10530 Print#Pr,"sorkezdeti9 keresem."
10540 if nz<>1 then if9<>46 then if 9<>s%(s) then 9osub
 42000 : 9oto 10540
10545 Print#Pr,
10550 if nz=1 then Print#Pr,"A kovetkezo sori9 olvestam
 !"
10560 if 9=s%(s) and s%(s)>127 then Print#Pr,mi$(s%(s)-
128);" alapszot talaltam ."
10570 if 9=s%(s) and s%(s)<128 then Print#Pr,s;" kodot
talaltam."
10580 if 9=s%(s) then Print#Pr,"Folytatom a szintaktiku
s elemzest."
10590 if 9=s%(s) then 9osub40000 : 9osub 42000 : 9oto 2
000
```

```
10600 if g=46 then Print#Pr,"A pontig olvastam !"
10610 Print#Pr,"Megprobalom ujra kezdeni es "
10620 Print#Pr,"a szintaktikai elemzest folytatni."
10640 if s%(s)>250 and s>=2 then s=s-2 : goto 10640
10650 if s<=0 then 35000
10660 if s%(s)<>46 then s=s-1 : goto 10640
10670 gosub 42000 : gosub 40000 : goto 2000
11000 rem **** szetagazas az alternativakhoz
11010 on int(o/10) goto 11030,11040
11020 on o goto 12100,12200,12300,12400,12500,12600,127
00,12800,12900
11030 on o-9 goto 13000,13100,13200,13300,13400,13500,1
3600,13700,13800,13900
11040 on o-19 goto 14000,14100,14200,14300,14400,14500,
14600
11050 Print#Pr,chr$(13)"Nincs ilyen szabaly!"
11060 stop
12000 rem alternativak kivalasztasa
12100 rem /01/ utasitas
12105 if g=142 then al=15 : return
12110 if g=167 then al=16 : return
12115 if g=168 then al=17 : return
12120 if g=164 then al=18 : return
12125 if g=152 then al=19 : return
12130 if g=133 then al=20 : return
12135 if g=134 then al=21 : return
12140 if g=169 then al=22 : return
12145 if g=129 then al=23 : return
12150 if g=149 then al=24 : return
12155 if g=139 then al=25 : return
12157 if g=151 then al=26 : return
12160 if g=140 then al=27 : return
12162 if g=160 then al=28 : return
12164 if g=128 then al=29 : return
12166 if g=158 then al=30 : return
12168 if g=144 then al=31 : return
12170 if g=137 then al=32 : return
12172 if g=147 then al=33 : return
12174 if g=132 then al=34 : return
12176 if g=166 then al=35 : return
12178 if g=153 then al=36 : return
12180 if g=130 then al=37 : return
12182 if g=171 then al=38 : return
12184 if g=157 then al=39 : return
12186 if g=156 then al=40 : return
12188 if g=150 then al=41 : return
12190 return
12200 rem /02/ utasitasblokk
```

```
12210 gosub 12100
12220 if a1<>255 then a1=12
12230 return
12300 rem /03/ utasitassorozat
12310 gosub 12100
12320 if g=146 then a1=14 : return
12330 if g=161 then a1=14 : return
12340 if g=155 then a1=14 : return
12350 if g=165 then a1=14 : return
12360 if g=46 then a1=14 : return
12370 if g=154 then a1=14 : return
12380 if a1<>255 then a1=13 : return
12390 return
12400 rem /04/ feltetel
12410 if g>64 and g<91 then a1=46
12420 return
12500 rem /05/ jeloles
12510 if g>64 and g<91 then a1=4
12520 return
12600 rem /06/ kepzes-1
12610 if g=145 then a1=44 : return
12620 if g=46  then a1=45 : return
12630 return
12700 rem /07/ betuk
12710 a1=6
12720 return
12800 rem /08/ definicios blokk
12810 if g=138 then a1=7
12820 if g=131 then a1=8
12830 return
12900 rem /09/ exponens
12910 if g=69 then a1=64
12920 return
13000 rem /10/ exponens-1
13010 if g=43 then a1=65
13020 if g=45 then a1=66
13030 if g>47 and g<58 then a1=67
13040 return
13100 rem /11/ lebegopontos szam
13110 if g>47 and g<58 then a1=58
13120 return
13200 rem /12/ lebegopontos szam-1
13210 if g=46 then a1=59
13220 if g=69 then a1=60
13230 if g=145 then a1=61
13240 return
13300 rem /13/ lebegopontos szam-2
13310 if g=69 then a1=62
```

```
13320 if g=145 then al=63
13330 return
13400 rem /14/ egesz szam
13410 if g>47 and g<58 then al=55
13420 return
13500 rem /15/ operator
13510 if g=61 then al=47 : return
13520 if g=174 then al=48 : return
13530 if g=60  then al=49 : return
13540 if g=175 then al=50 : return
13550 if g=62  then al=51 : return
13560 if g=176 then al=52 : return
13570 return
13600 rem /16/ program
13610 if g=148 then al=1
13620 return
13700 rem /17/ atvitel-1
13710 if g>47 and g<58 then al=42 : return
13720 if g>64 and g<91 then al=43
13730 return
13800 rem /18/ alprogram
13810 if g=162 then al=2
13820 if g=255 then al=3
13830 return
13900 rem /19/ valtozok definialasa
13910 if g=138 then al=9
13920 return
14000 rem /20/ valtozok definialasa-2
14010 if g=44 then al=10
14020 if g=46 then al=11
14030 return
14100 rem /21/ karakterlancok
14110 if g<>34 then al=53
14120 if g=34 then al=54
14130 return
14200 rem /22/ szamjegy
14210 al=57
14220 return
14300 rem /23/ egesz szam-1
14310 if g>47 and g<58 then al=55
14320 if g=46 or g=69 or g=145 then al=56
14330 return
14400 rem /24/ jeloles-1
14410 if g>64 and g<91 then al=4 : return
14420 if g=141 or g=46 or g=44 or g=170 or g=145 then a
l=5 : return
14430 if g=163 or g=143 or g=136 or g=159 or g=166 then
 al=5 : return
```

```
14440 if g=172 or g=173 or g=135 or g=174 or g=175 then
 al=5 : return
14450 if g=176 or g=60 or g=61 or g=62 then al=5 : retu
rn
14460 return
14500 rem /25/ mutato-1
14510 if g=34 then al=68 : return
14520 if g>64 and g<91 thenal=69 : return
14530 return
14600 rem /26/ billentyu jelek
14610 if g<>34 then al=57 : return
14620 al=6
35000 rem Programve9
35010 print#Pr,chr$(13)chr$(13)"A szintaktikus elemzest
 befejeztem."
35020 if fe<>0 then Print#Pr,fe;" hibat dolgoztam fol."

35030 if fe=0 then Print#Pr,chr$(13)"A Program szintakt
ikusan helyes."
35040 Print#ou,chr$(255)chr$(255);
35050 close Pr   close in : close ou : close be
35060 end
38000 rem szemantika
38010 if s%(s-1)=54 then bu$=bu$+g$ : gosub 40000 : gos
ub 42000 : goto 2000
38020 if s%(s-1)=56 then bu$=bu$+g$ : gosub 40000 : gos
ub 42000 : goto 2000
38030 if s%(s-1)=58 then bu$=bu$+g$ : gosub 40000 : gos
ub 42000 : goto 2000
38040 if s%(s-1)=55 then Print#ou,chr$(255)chr$(55);bu$
;: bu$="" : gosub 40000
38050 if s%(s-1)=57 then Print#ou,chr$(255)chr$(57);bu$
;: bu$="" : gosub 40000
38060 if s%(s-1)=59 then Print#ou,chr$(255)chr$(59);bu$
;: bu$="" : gosub 40000
38070 goto 2000
39000 rem semantika
39010 Print#ou,chr$(255)chr$(253)chr$(s%(s-1));
39020 gosub 40000
39030 goto 2000
40000 rem a legfelso szimbolum eltavolitasa
40010 if s%(s)=253 then s=s-2 : return
40020 if s%(s)=252 then s=s-2 : return
40030 if s%(s)=251 then s=s-2 : return
40040 s=s-1 : return
40050 :
41000 rem az alternativa verembe irasa
41010 al%=len(al$(al))
```

```
41020 if al%=0 then gosub 40000 : return
41025 s=s+1
41030 for t9=al% to 1 step -1
41040 s%(s)=asc(mid$(al$(al),t9,1)) : s=s+1
41050 next t9
41060 s=s-1
41065 rem print#pr,chr$(13)chr$(13):for l=sto0step-1:pri
nt#pr,s%(l);:next
41070 return
41080 :
42000 rem uj karakter allitasa
42010 if tl>len(t$) then t$="" : gosub 50000 : tl=1
42020 g$=mid$(t$,tl,1) : g=asc(g$)
42030 tl=tl+1
42035 rem print#pr,chr$(13)"Az aktualis karakter : ";g
42037 rem print#pr,chr$(13)"t$=  ";t$
42040 return
42050 :
49000 rem a nyelvtan
49010 data 148,252,1,253,5,141,252,2,253,8,252,3
49012 data 131,253,2,146,252,4,253,5,46,253,18,252,5,25
5,250
49020 data 162,252,6,253,5,141,253,2,161,252,7,253,5,46
,253,18,250
49030 data 250
49040 data 251,54,253,7,253,24,250
49050 data 251,55,250
49060 data 250
49070 data 253,19,250
49080 data 250
49090 data 138,253,5,253,20,46,253,8,250
49100 data 44,253,5,253,20,250
49110 data 250
49120 data 253,1,253,3,250
49130 data 253,1,253,3,250
49140 data 250
49150 data 142,46,250
49160 data 252,8,167,253,25,46,250
49170 data 252,9,168,253,25,46,250
49180 data 252,10,164,46,250
49190 data 252,11,152,46,250
49200 data 252,12,133,253,14,46,250
49210 data 252,13,134,253,14,46,250
49220 data 252,14,169,253,14,46,250
49230 data 252,15,129,253,5,46,250
49240 data 252,16,149,253,5,46,250
49250 data 252,17,139,253,5,46,250
49260 data 252,18,151,253,5,46,250
```

```
49270 data 252,19,140,253,5,46,250
49280 data 252,20,160,253,17,46,250
49290 data 252,21,128,253,5,170,253,5,145,253,5,46,250
49300 data 252,22,158,253,5,163,253,5,145,253,5,46,250
49310 data 252,23,144,253,5,143,253,5,145,253,5,46,250
49320 data 252,24,137,253,5,136,253,5,145,253,5,46,250
49330 data 252,25,147,253,5,143,253,5,145,253,5,46,250
49340 data 252,26,132,253,5,163,253,5,253,6,252,27,46,2
50
49350 data 252,28,166,253,4,135,253,2,252,23,155,253,2
49355 data 165,252,30,46,250
49360 data 252,31,153,253,5,159,253,2,154,252,32,253,5,
46,250
49370 data 252,33,130,253,5,166,253,4,46,250
49380 data 252,34,171,253,5,163,253,5,172,253,5,173
49385 data 252,35,253,2,154,252,36,46,250
49390 data 252,37,157,253,5,46,250
49400 data 252,38,156,253,5,46,250
49410 data 252,39,150,253,5,46,250
49420 data 252,40,253,11,145,253,5,250
49430 data 252,41,253,5,145,253,5,250
49440 data 252,42,145,253,5,250
49450 data 250
49460 data 253,5,253,15,253,5,250
49470 data 252,43,61,250
49480 data 252,44,174,250
49490 data 252,45,60,250
49500 data 252,46,175,250
49510 data 252,47,62,250
49520 data 252,48,176,250
49530 data 251,56,253,26,253,21,250
49540 data 251,57,250
49550 data 251,58,253,22,253,23,250
49560 data 251,59,250
49570 data 250
49580 data 253,14,253,12,250
49590 data 252,49,46,253,14,253,13,250
49600 data 253,9,250
49610 data 252,50,250
49620 data 253,9,250
49630 data 252,50,250
49640 data 252,51,69,253,10,250
49650 data 252,53,43,253,14,250
49660 data 252,53,45,253,14,250
49670 data 253,14,250
49680 data 34,253,21,34,250
49690 data 253,5,250
49800 for x=1 to 69
```

```
49810 read z% : if z%<>250 then al$(z)=al$(z)+chr$(z%)
: goto 49810
49820 next
49830 return
50000 rem lexikai analizis
50005 zn=0
50010 if i1=0 then if i2=0 then if i3=0 then t$=chr$(25
5) : return
50020 if i1=0 then gosub 51000
50030 if i1>127 and i1<204 then i1$=ba$(i1-128)
50040 if i1=171 then if i2=171 then gosub 51700 : goto
50000
50050 if i1$=" " then if i2$=" " then Print#Pr,i1$;: go
sub 51500 : goto 50000
50070 if i1$="<" and i2=178 then t$=chr$(175) : goto 50
195
50072 if i1$=">" and i2=178 then t$=chr$(176) : goto 50
195
50074 if i1$="/" and i2=178 then t$=chr$(174) : goto 50
195
50080 if i1=34 then gosub 53000 : goto 50210
50085 if i1=32 then if i2=34 then goto 50130
50090 if i2$="=" or i2$="/" or i2$="<" or i2$=">" then
gosub 52000 : goto 50200
50100 if i2$="." or i2$=" " or i2=0 or i2=34 then gosub
 52000 : goto 50200
50110 if i2$="," then gosub 52000 : goto 50200
50120 if i1$="." or i1$="," or i1$="=" then t$=i1$ : go
to 50200
50122 if i1$="/" or i1$="<" or i1$=">" then t$=i1$ : go
to 50200
50130 if i1$=" " then Print#Pr," ";: gosub 51500 : goto
 50000
50190 t$=t$+i1$ : Print#Pr,i1$;: gosub 51500 : goto 500
00
50195 Print#Pr,i1$;: gosub 51500
50197 if i1>127 and i1<204 then i1$=ba$(i1-128)
50200 Print#Pr,i1$;: gosub 51500
50210 return
51000 rem sorkezdet
51005 :
51007 zn=1
51010 gosub 51500 : gosub 51500 : gosub 51500
51020 Print#ou,chr$(255)chr$(254)chr$(i1)chr$(i2);
51030 Print#Pr,chr$(13);str$(i1+i2*256);" ";
51040 gosub 51500 : gosub 51500 : return
51050 :
51500 rem karakter beolvasas
```

```
51505 :
51510 i1$=i2$ : i2$=i3$ : 9et#in,i3$
51520 i1=i2 : i2=i3 : if i3$="" then i3=0 : return
51530 i3=asc(i3$) : return
51540 :
51700 rem sorok a ve9ei9 atolvasva
51710 Print#Pr,i1$; : 9osub 51500 : if i1=0 then return
51720 if i1>127 and i1<204 then i1$=ba$(i1-128)
51730 9oto 51710
51740 :
52000 rem t$ token?
52010 :
52020 t$=t$+i1$ : t=0 : a=asc(left$(t$,1))
52030 if a<65 or a>90 then return
52040 on int((a-65)/5) 9oto 52060,52070,52080,52090,521
00
52050 on (a-64) 9oto 52200,52220,52240,52260,52900
52060 on (a-69) 9oto 52300,52900,52340,52360,52900
52070 on (a-74) 9oto 52900,52380,52400,52420,52900
52080 on (a-79) 9oto 52440,52900,52460,52480,52900
52090 on (a-84) 9oto 52500,52520,52540,52900,52900
52100 9oto 52560
52200 b=0  : c=2  : 9osub 52700 : return : rem a
52220 b=3  : c=4  : 9osub 52700 : rem b
52230 b=44 : c=44 : 9osub 52700 : return : rem bis
52240 b=5  : c=6  : 9osub 52700 : return : rem c
52260 b=7  : c=9  : 9osub 52700 : return : rem d
52300 b=10 : c=10 : 9osub 52700 : rem f
52310 b=43 : c=43 : 9osub 52700 : return : rem fur
52340 b=11 : c=12 : 9osub 52700 : return : rem h
52360 b=13 : c=13 : 9osub 52700 : return : rem i
52380 b=14 : c=14 : 9osub 52700 : return : rem l
52400 b=15 : c=16 : 9osub 52700 : return : rem m
52420 b=17 : c=17 : 9osub 52700 : return : rem n
52440 b=18 : c=20 : 9osub 52700 : return : rem p
52460 b=21 : c=22 : 9osub 52700 : return : rem r
52480 b=23 : c=30 : 9osub 52700 : return : rem s
52500 b=31 : c=34 : 9osub 52700 : return : rem u
52520 b=35 : c=36 : 9osub 52700 : return : rem v
52540 b=37 : c=38 : 9osub 52700 : rem w
52550 b=45 : c=45   9osub 52700 : return : rem ismetles

52560 b=39 : c=42 : 9osub 52700 : return : rem z
52700 for d=b to c
52710 if t$=mi$(d) then t=128+d
52720 next d
52800 if t=0 then return
52810 t$=chr$(t) : return
```

```
52900 return
53000 rem karakterlanc atolvasas
53010 t$=t$+i1$ : print#pr,i1$; : gosub 51500 : if i1<>3
4 then 53010
53020 t$=t$+i1$ : print#pr,i1$; : gosub 51500 : return
53030 :
58000 rem miniatur alapszavak
58005 :
58008 mi$(0)="addiere"
58010 mi$(1)="ausgabegeraet"
58020 mi$(2)="ausgang"
58030 mi$(3)="beginne"
58040 mi$(4)="bilde"
58050 mi$(5)="cuspalte"
58060 mi$(6)="cuzeile"
58070 mi$(7)="dann"
58080 mi$(8)="durch"
58090 mi$(9)="dividiere"
58100 mi$(10)="fliess"
58110 mi$(11)="hintergrund"
58120 mi$(12)="hole"
58130 mi$(13)="ist"
58140 mi$(14)="leer"
58150 mi$(15)="mit"
58160 mi$(16)="multipliziere"
58170 mi$(17)="nach"
58180 mi$(18)="ende"
58200 mi$(20)="programm"
58210 mi$(21)="rahmen"
58220 mi$(22)="rufe"
58230 mi$(23)="schrift"
58240 mi$(24)="schirmfrei"
58250 mi$(25)="schleife"
58260 mi$(26)="sende"
58270 mi$(27)="sonst"
58280 mi$(28)="springe"
58290 mi$(29)="sprungmarke"
58300 mi$(30)="subtrahiere"
58310 mi$(31)="ueber"
58320 mi$(32)="uebertrage"
58330 mi$(33)="uende"
58340 mi$(34)="unterprogramm"
58350 mi$(35)="von"
58360 mi$(36)="vorschub"
58370 mi$(37)="wende"
58380 mi$(38)="wenn"
58390 mi$(39)="zeige"
58400 mi$(40)="zeigez"
```

```
58410 mi$(41)="zeigeas"
58420 mi$(42)="zu"
58430 mi$(43)="fuer"
58440 mi$(44)="bis"
58450 mi$(45)="wiederhole"
58460 mi$(46)="/="
58470 mi$(47)="<="
58480 mi$(48)=">="
58490 return
58500 :
59000 rem basic alapszavak
59005 :
59008 ba$(0)="end"
59010 ba$(1)="for"
59020 ba$(2)="next"
59030 ba$(3)="data"
59040 ba$(4)="input#"
59050 ba$(5)="input"
59060 ba$(6)="dim"
59070 ba$(7)="read"
59080 ba$(8)="let"
59090 ba$(9)="goto"
59100 ba$(10)="run"
59110 ba$(11)="if"
59120 ba$(12)="restore"
59130 ba$(13)="gosub"
59140 ba$(14)="return"
59150 ba$(15)="rem"
59160 ba$(16)="stop"
59170 ba$(17)="on"
59180 ba$(18)="wait"
59190 ba$(19)="load"
59200 ba$(20)="save"
59210 ba$(21)="verify"
59220 ba$(22)="def"
59230 ba$(23)="poke"
59240 ba$(24)="print#"
59250 ba$(25)="print"
59260 ba$(26)="cont"
59270 ba$(27)="list"
59280 ba$(28)="clr"
59290 ba$(29)="cmd"
59300 ba$(30)="sys"
59310 ba$(31)="open"
59320 ba$(32)="close"
59330 ba$(33)="get"
59340 ba$(34)="new"
59350 ba$(35)="tab("
```

```
59360 ba$(36)="to"
59370 ba$(37)="fn"
59380 ba$(38)="spc("
59390 ba$(39)="then"
59400 ba$(40)="not"
59410 ba$(41)="step"
59420 ba$(42)="+"
59430 ba$(43)="-"
59440 ba$(44)="*"
59450 ba$(45)="/"
59460 ba$(46)="^"
59470 ba$(47)="and"
59480 ba$(48)="or"
59490 ba$(49)=">"
59500 ba$(50)="="
59510 ba$(51)="<"
59520 ba$(52)="sgn"
59530 ba$(53)="int"
59540 ba$(54)="abs"
59550 ba$(55)="usr"
59560 ba$(56)="fre"
59570 ba$(57)="pos"
59580 ba$(58)="sqr"
59590 ba$(59)="rnd"
59600 ba$(60)="log"
59610 ba$(61)="exp"
59620 ba$(62)="cos"
59630 ba$(63)="sin"
59640 ba$(64)="tan"
59650 ba$(65)="atn"
59660 ba$(66)="peek"
59670 ba$(67)="len"
59680 ba$(68)="str$"
59690 ba$(69)="val"
59700 ba$(70)="asc"
59710 ba$(71)="chr$"
59720 ba$(72)="left$"
59730 ba$(73)="right$"
59740 ba$(74)="mid$"
59750 ba$(75)="go"
59760 return
59770 :
60000 rem lemezegyseg hibaszatornajanak olvasasa
60010 :
60020 input#be,en,em$,et,es : if en=0 then return
60030 print
60040 print"****  hiba a lemezen!"
60050 print"****  hibaszam :";en
```

```
60060 Print"****  hibauzenet :"
60070 Print"****  ";em$
60080 Print"****  hibas sav : ";et
60090 Print"****  hibas szektor : ";es
60100 Print#be,"i"
60110 close be : close pr
60120 Print : Print"**** A program leallt !"
60130 end
```

4.8 A PARSER OUTPUT kinyomtatása

A PARSER eredményeit, amelyekre a szemantikai elemzésnél és a kódgenerálásnál van szükség, a következő programmal vihetjük ki a képernyőre, ill. a nyomtatóra.

A kivitel eredményeként azt a rövidített változatot kapjuk meg, amely az összes szükséges információt tartalmazza. Vegyük például a *kezdés* nevű programot.

A kezdés nevű program kivitele:

a programazonosító definíciója,
a kezdés azonosítója,
a definíciós rész kezdete,
a definíciós rész vége,
a programazonosító vizsgálata,
a kezdés azonosítója,
a program vége,
a program vége.

Az első program vége a program logikai végét jelenti, a második a program adatállományának fizikai vége.

P.2

```
a Parser output listaja

1000 rem Program a
1010 rem miniatur-syn kiirasahoz
1020 rem
1030 rem ----------------------------------
1040 rem
1050 Print"A nyomtatatot (n) vagy a kepernyot (k) hasz
naljam ?"
1060 Get w$ : if w$="n" then Pr=4 : goto 1085
1070 if w$="k" then Pr=3 : goto 1085
1080 goto 1060
1085 Print"Sorszammal ? (i)"
1087 Get w$ : if w$="" then 1087
1089 if w$="i" then ss=1
1100 oPen Pr,Pr
1110 in=8
1120 oPen in,in,in,"miniatur-syn,s,r"
1130 Get#in,w$
```

```
1140 9et#in,w$ : if w$="" then w=0 : 9oto 1160
1150 w=asc(w$)
1160 if w=253 then Print#Pr,chr$(13);
1165 if w<>253 then 5000
1166 9et#in,w$ : w=asc(w$)
1170 if w=1 then Print#Pr,"Pro9ramazonosito definiciója
           ";: 9oto 1130
1180 if w=2 then Print#Pr,"definicios resz kezdete
           ";: 9oto 1130
1190 if w=3 then Print#Pr,"definicios resz ve9e
           ";: 9oto 1130
1200 if w=4 then Print#Pr,"Pro9ramazonosito vizs9alata
           ";: 9oto 1130
1210 if w=5 then Print#Pr,"Pro9ram ve9e
           ";: 9oto 1130
1220 if w=6 then Print#Pr,"alPro9ramazonosito definicio
ja         ";: 9oto 1130
1230 if w=7 then Print#Pr,"alPro9ramazonosito vizs9alat
a          ";: 9oto 1130
1240 if w=8 then Print#Pr,"kiiras
           ";: 9oto 1130
1250 if w=9 then Print#Pr,"kiiras soremelessel
           ";: 9oto 1130
1260 if w=10 then Print#Pr,"soremeles
            ";: 9oto 1130
1270 if w=11 then Print#Pr,"kePernyotorles
            ";: 9oto 1130
1280 if w=12 then Print#Pr,"kursoroszloP
            ";: 9oto 1130
1290 if w=13 then Print#Pr,"kursorsor
            ";: 9oto 1130
1300 if w=14 then Print#Pr,"ascii kiiras
            ";: 9oto 1130
1310 if w=15 then Print#Pr,"eszkoz
            ";: 9oto 1130
1320 if w=16 then Print#Pr,"keretszin
            ";: 9oto 1130
1330 if w=17 then Print#Pr,"hatterszin
            ";: 9oto 1130
1340 if w=18 then Print#Pr,"irasszin
            ";: 9oto 1130
1350 if w=19 then Print#Pr,"adatbevitel
            ";: 9oto 1130
1360 if w=20 then Print#Pr,"ertekadas
            ";: 9oto 1130
1400 if w=21 then Print#Pr,"osszeadas
            ";: 9oto 1130
1410 if w=22 then Print#Pr,"kivonas
            ";: 9oto 1130
```

```
1420 if w=23 then Print#Pr,"szorzas
               ";: goto 1130
1430 if w=24 then Print#Pr,"osztas
               ";: goto 1130
1440 if w=25 then Print#Pr,"hatvanyozas
               ";: goto 1130
1450 if w=26 then Print#Pr,"kepzes
               ";: goto 1130
1460 if w=27 then Print#Pr,"pont
               ";: goto 1130
1500 if w=28 then Print#Pr,"dontes
               ";: goto 1130
1510 if w=29 then Print#Pr,"kulonben
               ";: goto 1130
1520 if w=30 then Print#Pr,"dontes vege
               ";: goto 1130
1530 if w=31 then Print#Pr,"vegtelen ciklus
               ";: goto 1130
1540 if w=32 then Print#Pr,"vegtelen ciklus vege
               ";: goto 1130
1550 if w=33 then Print#Pr,"kimenet
               ";: goto 1130
1560 if w=34 then Print#Pr,"ciklus
               ";: goto 1130
1600 if w=35 then Print#Pr,"cikluskezdet
               ";: goto 1130
1610 if w=36 then Print#Pr,"ciklusveg
               ";: goto 1130
1620 if w=37 then Print#Pr,"ugras jelzo
               ";: goto 1130
1630 if w=38 then Print#Pr,"ugras
               ";: goto 1130
1640 if w=39 then Print#Pr,"hivas
               ";: goto 1130
1650 if w=40 then Print#Pr,"lebegopontos szam
               ";: goto 1130
1660 if w=41 then Print#Pr,"valtozo
               ";: goto 1130
1700 if w=42 then Print#Pr,"kepzes-1
               ";: goto 1130
1710 if w=43 then Print#Pr,"operator =
               ";: goto 1130
1720 if w=44 then Print#Pr,"operator /=
               ";: goto 1130
1730 if w=45 then Print#Pr,"operator <
               ";: goto 1130
1740 if w=46 then Print#Pr,"operator <=
               ";: goto 1130
1750 if w=47 then Print#Pr,"operator >
               ";: goto 1130
```

```
1760 if w=48 then Print#Pr,"operator >=
              ";: goto 1130
1800 if w=49 then Print#Pr,"decimalis helyzet
              ";: goto 1130
1810 if w=50 then Print#Pr,"nincs kitevo
              ";: goto 1130
1820 if w=51 then Print#Pr,"kitevo
              ";: goto 1130
1830 if w=52 then Print#Pr,"Plus
              ";: goto 1130
1840 if w=53 then Print#Pr,"minus
              ";: goto 1130
1850 Print#Pr,"hiba !!!!! ";
1860 stop
5000 rem rutin w>54
5010 if w=255 then Print#Pr,chr$(13)"Program vege";: go
to 13000
5020 if w=254 and ss=1 then Print#Pr,chr$(13)"sorszam :
 ";: goto 14000
5025 if w=254 then 14000
5100 if w=55 then Print#Pr,chr$(13)"azonosito : ";: got
o 10000
5110 if w=57 then Print#Pr,chr$(13)"karakterlanc : ";:
goto 11000
5120 if w=59 then Print#Pr,chr$(13)"egesz szam : ";: go
to 12000
10000 rem azonosito beolvasasa
10010 get#in,w$ : if w$=chr$(255) then 1140
10020 Print#Pr,w$;
10030 goto 10010
11000 rem karakterlanc beolvasas
11010 get#in,w$ : if w$=chr$(255) then 1140
11020 Print#Pr,w$;
11030 goto 10010
12000 rem egesz szam beolvasas
12010 get#in,w$ : if w$=chr$(255) then 1140
12020 Print#Pr,w$;
12030 goto 10010
13000 rem Programvege
13010 close Pr
13020 close in
13030 end
14000 rem sorszam
14010 get#in,w$ : if w$="" then z=0 : goto 14020
14015 z=asc(w$)
14020 get#in,w$ : if w$="" then 14025
14022 z=z+asc(w$)*256
14025 if ss=1 then Print#Pr,z;
14030 goto 1130
```

P.3

```
MintaPelda : 'szin'

Programazonosito definicioja
azonosito : szin
definicios resz kezdete
definicios resz vege
kepernyotorles
hatterszin
azonosito : fekete
keretszin
azonosito : feher
irasszin
azonosito : feher
kurzorsor
egesz szam : 12
kurzoroszlop
egesz szam : 17
ascii kiiras
egesz szam : 18
kiiras szemelessel
karakterlanc : szin
ascii kiiras
egesz szam : 146
hatterszin
azonosito : feher
keretszin
azonosito : fekete
Programazonosito vizsgalata
azonosito : szin
Program vege
Program vege


MintaPelda : 'kezdes'

Programazonosito definicioja
azonosito : kezdes
definicios resz kezdete
definicios resz vege
Programazonosito vizsgalata
azonosito : kezdes
Program vege
Program vege
```

MintaPelda : 'kiiras'

```
Pro9ramazonosito definiciója
azonosito : kiiras
definicios resz kezdete
definicio= resz ve9e
kiiras soremelessel
karakterlanc : Pelda
kiiras
karakterlanc : szove9ek
kiiras
karakterlanc : kiirasara.
Pro9ramazonosito vizs9alata
azonosito : kiiras
Pro9ram ve9e
Pro9ram ve9e
```

MintaPelda : 'kiirasb'

```
Pro9ramazonosito definiciója
azonosito : kiirasb
definicios resz kezdete
definicios resz ve9e
kePernyotorles
kursoPsor
e9esz szam : 5
kursoroszloP
e9esz szam : 5
ascii kiiras
e9esz szam : 18
kiiras
karakterlanc : iras a kePernyore
ascii kiiras
e9esz szam : 146
soremeles
soremeles
soremeles
kiiras soremelessel
karakterlanc : iras a kePernyore
Pro9ramazonosito vizs9alata
azonosito : kiirasb
Pro9ram ve9e
Pro9ram ve9e
```

```
MintaPelda : 'nyomtato'

Programazonosito definicioja
azonosito : nyomtato
definicios resz kezdete
definicios resz vege
eszkoz
azonosito : nyomtato
kiiras soremelessel
karakterlanc : Most a nyomtatunk
eszkoz
azonosito : kepernyo
kiiras soremelessel
karakterlanc : Ismet a kepernyore irunk.
Programazonosito vizsgalata
azonosito : nyomtato
Program vege
Program vege


MintaPelda : 'definicio'

Programazonosito definicioja
azonosito : definicio
definicios resz kezdete
azonosito : otto
azonosito : emma
azonosito : benno
definicios resz vege
Programazonosito vizsgalata
azonosito : definicio
Program vege
Program vege


MintaPelda : 'adatbevitel

Programazonosito definicioja
azonosito : be
definicios resz kezdete
azonosito : kora
definicios resz vege
kepernyotorles
kiiras
karakterlanc : kora:
adatbevitel
azonosito : kora
kiiras
```

```
karakterlanc : On
kiiras
azonosito : kora
kiiras soremelessel
karakterlanc : eves.
programazonosito vizsgalata
azonosito : be
program vege
program vege


MintaPelda : 'aritmetika'

programazonosito definicioja
azonosito : aritmetika
definicios resz kezdete
azonosito : monika
azonosito : thomas
azonosito : cecilia
azonosito : benno
azonosito : erno
azonosito : otto
definicios resz vege
kepernyotorles
soremeles
kiiras soremelessel
karakterlanc : A lebegoPontos aritmetika bemutatasa.
soremeles
ertekadas
lebegoPontos szam
egesz szam : 2
nincs kitevo
azonosito : monika
ertekadas
lebegoPontos szam
egesz szam : 5
nincs kitevo
azonosito : thomas
ertekadas
valtozo
azonosito : thomas
azonosito : cecilia
kiiras
karakterlanc : monika =
kiiras soremelessel
azonosito : monika
kiiras
karakterlanc : thomas =
kiiras soremelessel
azonosito : thomas
kiiras
```

```
karakterlanc : cecilia =
kiiras soremelessel
azonosito : cecilia
osszeadas
azonosito : monika
azonosito : thomas
azonosito : benno
soremeles
kiiras
karakterlanc : 2 + 5 =
kiiras soremelessel
azonosito : benno
kivonas
azonosito : cecilia
azonosito : benno
azonosito : enno
kiiras
karakterlanc : 7 - 2 =
kiiras soremelessel
azonosito : enno
szorzas
azonosito : thomas
azonosito : monika
azonosito : otto
kiiras
karakterlanc : 5 * 2 =
kiiras soremelessel
azonosito : otto
osztas
azonosito : cecilia
azonosito : monika
azonosito : enno
kiiras
karakterlanc : 5 / 2 =
kiiras soremelessel
azonosito : enno
hatvanyozas
azonosito : monika
azonosito : thomas
azonosito : benno
kiiras
karakterlanc : 2 hoch 5 =
kiiras soremelessel
azonosito : benno
Programazonosito vizsgalata
azonosito : aritmetika
Program vege
Program vege
```

```
MintaPelda : 'fu99venyek'

Pro9ramazonosito definicioJa
azonosito : fu99venyek
definicios resz kezdete
azonosito : a
azonosito : b
azonosito : c
azonosito : d
azonosito : e
azonosito : f
azonosito : 9
azonosito : Piviertel
definicios resz ve9e
ertekadas
lebe9oPontos szam
e9esz szam : 4
nincs kitevo
azonosito : a
ertekadas
lebe9oPontos szam
e9esz szam : 4
nincs kitevo
azonosito : b
ertekadas
lebe9oPontos szam
e9esz szam : 3
decimalis helyzet
e9esz szam : 1415
nincs kitevo
azonosito : c
osztas
azonosito : c
azonosito : b
azonosito : Piviertel
kePernyotorles
kiiras soremelessel
karakterlanc : fu99venyek
soremeles
kePzes
azonosito : e9esz
azonosito : c
kePzes-1
azonosito : d
Pont
kiiras
karakterlanc :  c =
kiiras soremelessel
```

```
azonosito : c
kiiras
karakterlanc :  d =
kiiras soremelessel
azonosito : d
kepzes
azonosito : abszolut
azonosito : a
pont
kiiras
karakterlanc :  abszolut a  =
kiiras soremelessel
azonosito : a
kepzes
azonosito : actangens
azonosito : piviertel
kepzes-1
azonosito : e
pont
kiiras
karakterlanc :  actangens piviertel =
kiiras soremelessel
azonosito : e
kepzes
azonosito : cosinus
azonosito : piviertel
kepzes-1
azonosito : e
pont
kiiras
karakterlanc :  cosinus piviertel =
kiiras soremelessel
azonosito : e
kepzes
azonosito : exponens
azonosito : b
kepzes-1
azonosito : e
pont
kiiras
karakterlanc :  exponent 4 =
kiiras soremelessel
azonosito : e
kepzes
azonosito : gyok
azonosito : piviertel
kepzes-1
azonosito : e
```

```
Pont
kiiras
karakterlanc :  egesz Piviertel =
kiiras soremelessel
azonosito : e
kePzes
azonosito : logaritmus
azonosito : b
kePzes-1
azonosito : e
Pont
kiiras
karakterlanc :  logaritmus 4 =
kiiras soremelessel
azonosito : e
kePzes
azonosito : tar
azonosito : b
kePzes-1
azonosito : f
Pont
kiiras
karakterlanc :  tar 4 =
kiiras soremelessel
azonosito : f
kePzes
azonosito : veletlen
azonosito : a
kePzes-1
azonosito : g
Pont
kiiras
karakterlanc :  veletlen 4 =
kiiras soremelessel
azonosito : g
kePzes
azonosito : elojel
azonosito : b
kePzes-1
azonosito : g
Pont
kiiras
karakterlanc :  elojel 4 =
kiiras soremelessel
azonosito : g
kePzes
azonosito : sinus
azonosito : Piviertel
```

```
kePzes-1
azonosito : f
Pont
kiiras
karakterlanc :  sinus Piviertel =
kiiras soremelessel
azonosito : f
kePzes
azonosito : negyzetgyok
azonosito : b
kePzes-1
azonosito : g
Pont
kiiras
karakterlanc :  negyzetgyok 4 =
kiiras soremelessel
azonosito : g
azonosito : tangens
azonosito : Piviertel
kePzes-1
azonosito : f
Pont
kiiras
karakterlanc :  tangens Piviertel =
kiiras soremelessel
azonosito : f
kiiras soremelessel
karakterlanc : jo?
Programazonosito vizsgalata
azonosito : fuggvenyek
Program vege
Program vege

MintaPelda : 'Paros'

Programazonosito definicioja
azonosito : Paros
definicios resz kezdete
azonosito : szam
azonosito : seged
azonosito : ketto
azonosito : zero
azonosito : segedv
definicios resz vege
kePernyotorles
soremeles
ertekadas
lebegoPontos szam
```

```
egesz szam : 2
nincs kitevo
azonosito : ketto
kiiras
karakterlanc : Kerek egy egesz szamot
adatbevitel
azonosito : szam
soremeles
ertekadas
valtozo
azonosito : szam
azonosito : seged
osztas
azonosito : seged
azonosito : ketto
azonosito : seged
kepzes
azonosito : egesz
azonosito : seged
pont
osztas
azonosito : szam
azonosito : ketto
azonosito : segedv
ertekadas
lebegopontos szam
egesz szam : 0
decimalis helyzet
egesz szam : 0
nincs kitevo
azonosito : zero
dontes
azonosito : seged
operator =
azonosito : zero
kiiras
azonosito : szam
kiiras soremelessel
karakterlanc : Paros
kulonben
kiiras
azonosito : szam
kiiras soremelessel
karakterlanc : Paratlan
dontes vege
programazonosito vizsgalata
azonosito : Paros
program vege
program vege
```

```
MintaPelda : 'valasztas'

Programazonosito definicioja
azonosito : valasztas
definicios resz kezdete
azonosito : teszt
azonosito : egy
definicios resz vege
kepernyotorles
soremeles
kiiras
karakterlanc : nyomtato (1) / kepernyo (2)
adatbevitel
azonosito : teszt
soremeles
ertekadas
lebegoPontos szam
egesz szam : 1
decimalis helyzet
egesz szam : 0
nincs kitevo
azonosito : egy
dontes
azonosito : teszt
operator =
azonosito : egy
eszkoz
azonosito : nyomyato
kiiras soremelessel
karakterlanc : nyomtato
eszkoz
azonosito : kepernyo
kulomben
kiiras soremelessel
karakterlanc : kepernyo
dontes vege
Programazonosito vizsgalata
azonosito : valasztas
Program vege
Program vege


MintaPelda : 'dontesteszt'

Programazonosito definicioja
azonosito : dontesteszt
definicios resz kezdete
azonosito : harom
azonosito : negy
```

```
definicios resz vege
kepernyotorles
soremeles
ertekadas
lebegoPontos szam
egesz szam : 3
decimalis helyzet
egesz szam : 0
nincs kitevo
azonosito : harom
ertekadas
lebegoPontos szam
egesz szam : 4
decimalis helyzet
egesz szam : 0
nincs kitevo
azonosito : negy
kiiras soremelessel
karakterlanc : dontesek:
dontes
azonosito : harom
operator =
azonosito : negy
kiiras soremelessel
karakterlanc : =hiba !
kulonben
kiiras soremelessel
karakterlanc : =nincs hiba !
dontes
azonosito : harom
operator >
azonosito : negy
kiiras soremelessel
karakterlanc : >hiba !
kulonben
kiiras soremelessel
karakterlanc : >nincs hiba !
dontes
azonosito : harom
operator >=
azonosito : negy
kiiras soremelessel
karakterlanc : >=hiba !
kulonben
kiiras soremelessel
karakterlanc : >=nincs hiba !
dontes
azonosito : harom
```

```
operator /=
azonosito : negy
kiiras soremelessel
karakterlanc : /=nincs hiba !
dontes
azonosito : harom
operator <
azonosito : negy
kiiras soremelessel
karakterlanc : <nincs hiba !
dontes
azonosito : harom
operator <=
azonosito : negy
kiiras soremelessel
karakterlanc : <=nincs hiba !
kulonben
kiiras soremelessel
karakterlanc : <=hiba !
dontes vege
kulonben
kiiras soremelessel
karakterlanc : <hiba !
dontes vege
kulonben
kiiras soremelessel
karakterlanc : /=hiba !
dontes vege
dontes vege
dontes vege
dontes vege
Programazonosito vizsgalata
azonosito : dontesteszt
Program vege
Program vege


MintaPelda : 'vegtelen'

Programazonosito definicioja
azonosito : vegtelen
definicios resz kezdete
azonosito : szam
azonosito : egy
definicios resz vege
ertekadas
lebegoPontos szam
egesz szam : 1
```

```
decimalis helyzet
e9esz szam : 0
nincs kitevo
azonosito : e9y
ertekadas
lebe9oPontos szam
e9esz szam : 0
decimalis helyzet
e9esz szam : 0
nincs kitevo
azonosito : szam
ve9telen ciklus
azonosito : e9esz
kiiras soremelessel
azonosito : szam
osszeadas
azonosito : e9y
azonosito : szam
azonosito : szam
ve9telen ciklus ve9e
azonosito : e9esz
Pro9ramazonosito vizs9alata
azonosito : ve9telen
Pro9ram ve9e
Pro9ram ve9e


MintaPelda : 'szaz'

Pro9ramazonosito definicioJa
azonosito : szaz
definicios resz kezdete
azonosito : szam
azonosito : e9y
azonosito : szaz
definicios resz ve9e
ertekadas
lebe9oPontos szam
e9esz szam : 1
decimalis helyzet
e9esz szam : 0
nincs kitevo
azonosito : e9y
ertekadas
lebe9oPontos szam
e9esz szam : 0
decimalis helyzet
e9esz szam : 0
nincs kitevo
```

```
azonosito : szam
ertekadas
lebegoPontos szam
egesz szam : 100
decimalis helyzet
egesz szam : 0
nincs kitevo
azonosito : szaz
vegtelen ciklus
azonosito : egesz
kiiras soremelessel
azonosito : szam
osszeadas
azonosito : egy
azonosito : szam
azonosito : szam
kimenet
azonosito : egesz
azonosito : szaz
operator =
azonosito : szam
vegtelen ciklus vege
azonosito : egesz
Programazonosito vizsgalata
azonosito : szaz
Program vege
Program vege


MintaPelda : 'kilepesteszt'

Programazonosito definicioja
azonosito : kilepesteszt
definicios resz kezdete
azonosito : gudrun
azonosito : emno
azonosito : monika
definicios resz vege
kepernyotorles
soremeles
kiiras soremelessel
karakterlanc : test
soremeles
ertekadas
lebegoPontos szam
egesz szam : 2
decimalis helyzet
egesz szam : 0
nincs kitevo
```

```
azonosito : gudrun
ertekadas
lebegoPontos szam
egesz szam : 2
decimalis helyzet
egesz szam : 0
nincs kitevo
azonosito : monika
ertekadas
lebegoPontos szam
egesz szam : 7
decimalis helyzet
egesz szam : 0
nincs kitevo
azonosito : enno
vegtelen ciklus
azonosito : ciklusa
vegtelen ciklus
azonosito : ciklusb
vegtelen ciklus
azonosito : ciklusc
vegtelen ciklus
azonosito : ciklusd
vegtelen ciklus
azonosito : ciklusе
vegtelen ciklus
azonosito : ciklusf
vegtelen ciklus
azonosito : ciklusg
vegtelen ciklus
azonosito : ciklush
kimenet
azonosito : ciklush
azonosito : gudrun
operator /=
azonosito : enno
kiiras soremelessel
karakterlanc : hibas ciklush
vegtelen ciklus vege
azonosito : ciklush
kiiras soremelessel
karakterlanc : Jo ciklush
kimenet
azonosito : ciklusg
azonosito : gudrun
operator <
azonosito : enno
kiiras soremelessel
karakterlanc : hibas ciklusg
```

```
vegtelen ciklus vege
azonosito : ciklusg
kiiras soremelessel
karakterlanc : jo ciklusg
kimenet
azonosito : ciklusf
azonosito : gudrun
operator <=
azonosito : enno
kiiras soremelessel
karakterlanc : hibas ciklusf
vegtelen ciklus vege
azonosito : ciklusf
kiiras soremelessel
karakterlanc : jo ciklusf
kimenet
azonosito : cikluse
azonosito : gudrun
operator <=
azonosito : monika
kiiras soremelessel
karakterlanc : hibas cikluse
vegtelen ciklus vege
azonosito : cikluse
kiiras soremelessel
karakterlanc : jo cikluse
kimenet
azonosito : ciklusd
azonosito : enno
operator >
azonosito : monika
kiiras soremelessel
karakterlanc : hibas ciklusd
vegtelen ciklus vege
azonosito : ciklusd
kiiras soremelessel
karakterlanc : jo ciklusd
kimenet
azonosito : ciklusc
azonosito : enno
operator >=
azonosito : monika
kiiras soremelessel
karakterlanc : hibas ciklusc
vegtelen ciklus vege
azonosito : ciklusc
kiiras soremelessel
karakterlanc : jo ciklusc
```

```
kimenet
azonosito : ciklusb
azonosito : gudrun
operator >=
azonosito : monika
kiiras soremelessel
karakterlanc : hibas ciklusb
vegtelen ciklus vege
azonosito : ciklusb
kiiras soremelessel
karakterlanc : jo ciklusb
kimenet
azonosito : ciklusa
azonosito : gudrun
operator =
azonosito : monika
kiiras soremelessel
karakterlanc : hibas ciklusa
vegtelen ciklus vege
azonosito : ciklusa
kiiras soremelessel
karakterlanc : jo ciklusa
kiiras soremelessel
karakterlanc : vege
Programazonosito vizsgalata
azonosito : kilepesteszt
Program vege
Program vege

mintaPelda ' szamlalas

10 Programm szamlala  ist
20 fliess index,alsohatar,felsohatar.
30 beginne
40 uebertrage 1.0 nach alsohatar.
50 uebertrage 10.0 nach felsohatar.
60 fuer index von alsohatar bis felsohatar wiederhole
70 zeigez index.
80 sende.
90 Pende szamlalas.

MintaPelda : 'Partest'

Programazonosito definicioja
azonosito : Partest
definicios resz kezdete
azonosito : indexa
```

```
azonosito : indexb
azonosito : indexc
azonosito : alsohatar
azonosito : felsohatar
azonosito : szam
azonosito : egy
definicios resz vege
ertekadas
lebegoPontos szam
egesz szam : 1
decimalis helyzet
egesz szam : 0
nincs kitevo
azonosito : alsohatar
ertekadas
lebegoPontos szam
egesz szam : 10
decimalis helyzet
egesz szam : 0
nincs kitevo
azonosito : felsohatar
ertekadas
lebegoPontos szam
egesz szam : 0
decimalis helyzet
egesz szam : 0
nincs kitevo
azonosito : szam
ertekadas
lebegoPontos szam
egesz szam : 1
decimalis helyzet
egesz szam : 0
nincs kitevo
azonosito : egy
ciklus
azonosito : indexa
azonosito : alsohatar
azonosito : felsohatar
cikluskezdet
ciklus
azonosito : indexb
azonosito : alsohatar
azonosito : felsohatar
cikluskezdet
ciklus
azonosito : indexc
azonosito : alsohatar
azonosito : felsohatar
```

```
ciklusKezdet
osszeadas
azonosito : egy
azonosito : szam
azonosito : szam
ciklusVeg
ciklusVeg
ciklusVeg
kiiras soremelessel
karakterlanc : eredmeny :
kiiras soremelessel
azonosito : szam
ProgramazonositoVizsgalata
azonosito : Partest
Program VegE
Program VegE


MintaPelda : 'ugras'

ProgramazonositoDefiniciója
azonosito : ugras
definicios resz kezdete
azonosito : egy
azonosito : teszt
definicios resz vege
ertekadas
lebegoPontos szam
egesz szam : 1
decimalis helyzet
egesz szam : 0
nincs kitevo
azonosito : egy
ugras jelzo
azonosito : kezdes
adatbevitel
azonosito : teszt
soremeles
dontes
azonosito : teszt
oPerator =
azonosito : egy
kulonben
ugras
azonosito : kezdes
dontes vege
ProgramazonositoVizsgalata
azonosito : ugras
Program vege
Program vege
```

```
MintaPelda : 'aProgteszt'
Programazonosito definicioja
azonosito : aProgteszt
definicios resz kezdete
azonosito : teszt
azonosito : egy
definicios resz vege
ertekadas
lebegoPontos szam
egesz szam : 1
decimalis helyzet
egesz szam : 0
nincs kitevo
azonosito : egy
ugras jelzo
azonosito : kezdes
kiiras soremelessel
karakterlanc : alProgram 1/2 ?
adatbevitel
azonosito : teszt
soremeles
dontes
azonosito : teszt
operator /=
azonosito : egy
hivas
azonosito : a
kulonben
hivas
azonosito : b
dontes vege
ugras
azonosito : kezdes
Programazonosito vizsgalata
azonosito : aProgteszt
alProgramazonosito definicioja
azonosito : a
kiiras soremelessel
karakterlanc : alProgram a.
alProgramazonosito vizsgalata
azonosito : a
alProgramazonosito definicioja
azonosito : b
kiiras soremelessel
karakterlanc : alProgram b.
alProgramazonosito vizsgalata
azonosito : b
Program vege
Program vege
```

5. A SZEMANTIKAI ELEMZÉS ÉS A KÓDGENERÁLÁS

Ez az alprogram a MINIATUR programon belül a szintaktikai elemzéssel a miniatur syn adatállományába írt részeket elemzi. A mintapéldákat az előző oldalakon megtaláljuk.

A következő programban a 61 000-es sortól kezdődően az az alprogram található, amely lehetővé teszi ezeknek a részeknek kódolt formában való kiadását. A program betöltése után a RUN 61 000 utasítást adjuk. A nyomtatón ekkor számok jelennek meg, ezeket a szemantikai szimbólumok táblázata alapján dekódolhatjuk. A szemantikai elemzés során azután a MINIATUR programból egy ASSEMBLER program készül. Az előállított ASSEMBLER programot a LOAD "MINI–ASS",8" utasítással tölthetjük be és listáztathatjuk. Így a MINIATUR programot összehasonlíthatjuk az ASSEMBLER programmal. A szemantikai elemzés és a kódgenerálás működését úgy értjük meg a legjobban, ha követjük a MINIATUR programok fordítását.

Az assembler nyelvű programot a gépi kódban lévő ASSEMBLER-rel lehet lefordítani. A programnévként az ASSEMBLER megindítása után a MINI–ASS nevet kell megadni.

P.4

a szemantikai elemzes listaja

```
500 dim b$(500): dim t%(500)
510 dim i(40): iz=0 : lz=0
1000 print"Sorszammal forditsam ? (i/n)"
1010 get w$ : if w$="" then 1010
1020 if w$="i" then sp=1
1030 print"Protokoll a nyomtatora (n) vagy a kepernyore
 (k) keruljon ?"
1040 get w$ : if w$="" then 1040
1050 if w$="n" then pr=4 : goto 1080
1060 if w$="k" then pr=3 : goto 1080
1070 goto 1040
1080 print"Kerem az adatlemezt !"
1090 print"<CRETURN>"
1100 get w$ : if w$="" then 1100
1110 open pr,pr
1120 open 15,8,15,"i"
1130 print#15,"s:mini-ass"
```

```
1140 input#15,en,em$,et,es
1150 if en<>1 then 60020
1160 ou=9 : open ou,8,ou,"0:mini-ass,p,w"
1170 gosub 60000
1180 in=8 : open in,8,in,"0:miniatur-sym,s,r"
1190 gosub 60000
1200 pc=2049 : rem basic kezdocim
1210 p1=int(pc/256)
1220 p2=pc-p1*256
1230 print#ou,chr$(p2)chr$(p1);
2000 rem *** alprogram kiirasa
2010 ou$="lda #14" : gosub 59000
2020 ou$="jsr $ffd2" : gosub 59000
2030 ou$="lda #144" : gosub 59000
2040 ou$="jsr $ffd2" : gosub 59000
2050 ou$="lda #6" : gosub 59000
2060 ou$="sta 53281" : gosub 59000
2070 ou$="sta 53280" : gosub 59000
2080 ou$="jmp declaration": gosub 59000
2090 ou$="zvor .m" : gosub 59000
2100 ou$="lda #13" : gosub 59000
2110 ou$="jsr $ffd2" : gosub 59000
2120 ou$="rts" : gosub 59000
2130 ou$="aus .m" : gosub 59000
2140 ou$="jsr $bddd" : gosub 59000
2150 ou$="ldx #0" : gosub 59000
2160 ou$="ausp .m" : gosub 59000
2170 ou$="lda $100,x" : gosub 59000
2180 ou$="beq ause" : gosub 59000
2190 ou$="jsr $ffd2" : gosub 59000
2200 ou$="inx" : gosub 59000
2210 ou$="bne ausp" : gosub 59000
2220 ou$="ause .m" : gosub 59000
2230 ou$="rts" : gosub 59000
2240 ou$="ein .m" : gosub 59000
2250 ou$="lda #$3f" : gosub 59000
2260 ou$="jsr $ffd2" : gosub 59000
2270 ou$="ldx #0" : gosub 59000
2280 ou$="ein1 .m" : gosub 59000
2290 ou$="jsr $ffcf" : gosub 59000
2300 ou$="sta $0220,x" : gosub 59000
2310 ou$="inx" : gosub 59000
2320 ou$="cmp #$d" : gosub 59000
2330 ou$="bne ein1" : gosub 59000
2340 ou$="jsr $ffd2" : gosub 59000
2350 ou$="lda #$02" : gosub 59000
2360 ou$="sta $23" : gosub 59000
2370 ou$="lda #$20" : gosub 59000
```

```
2380 ou$="sta $22" : gosub 59000
2390 ou$="dex" : gosub 59000
2400 ou$="txa" : gosub 59000
2410 ou$="jsr $b7b5" : gosub 59000
2420 ou$="rts" : gosub 59000
2430 ou$=".c" : gosub 59000
2440 ou$=".t "+chr$(34)+"sor : "+chr$(34) : gosub 59000
2450 ou$="sPur .m" : gosub 59000
2460 ou$="jsr zvor" : gosub 59000
2470 ou$="ldy # ch" : gosub 59000
2480 ou$="lda # cl" : gosub 59000
2490 ou$="jsr $able" : gosub 59000
2500 ou$="rts" : gosub 59000
2990 ou$="declaration .m" : gosub 59000
5000 rem szemantikai ciklus
5010 get#in,i$
5020 get#in,i$: if i$="" then i=0 : goto 5040
5030 i=asc(i$)
5035 if i=55 then 9000
5040 if i<>253 then 6000
5050 get#in,i$ : i=asc(i$)
5060 if i=1 then 10000
5070 if i=2 then 10500
5080 if i=3 then 11000
5090 if i=4 then 11500
5100 if i=5 then 12000
5110 if i=6 then 12500
5120 if i=7 then 13000
5130 if i=8 then 13500
5140 if i=9 then 14000
5150 if i=10 then 14500
5160 if i=11 then 15000
5170 if i=12 then 15500
5180 if i=13 then 16000
5190 if i=14 then 16500
5200 if i=15 then 17000
5210 if i=16 then 17500
5220 if i=17 then 18000
5230 if i=18 then 18500
5240 if i=19 then 19000
5250 if i=20 then 19500
5260 if i=21 then 20000
5270 if i=22 then 20500
5280 if i=23 then 21000
5290 if i=24 then 21500
5300 if i=25 then 22000
5310 if i=26 then 22500
5320 if i=27 then 23000
```

```
5330 if i=28 then 23500
5340 if i=29 then 24000
5350 if i=30 then 24500
5360 if i=31 then 25000
5370 if i=32 then 25500
5380 if i=33 then 26000
5390 if i=34 then 26500
5400 if i=35 then 27000
5410 if i=36 then 27500
5420 if i=37 then 28000
5430 if i=38 then 28500
5440 if i=39 then 29000
5500 print#pr,"szemantikai hiba !!!!!!"
5510 stop
6000 if i=254 then 30000
6010 if i=255 then 31000
6020 print#pr,"szemantikai hiba !!!!!!"
6030 stop
9000 rem  valtozok definicioja
9010 if df=0 then gosub 56040
9020 if df=0 then me$="a valtozok magyarazata a definic
iosreszben !":goto 58700
9030 gosub 56040
9040 be$=il$ : ty%=3 : gosub 57000
9050 if sc=1 then me$="az azonosito mar definialt !":go
to 58700
9060 gosub 57500
9070 ou$=be$+" .bl 5" : gosub 59000
9080 goto 5020
9090 :
10000 rem programazonosito definicio
10005 gosub 56000
10010 be$=il$ : ty%=1 : gosub 57000
10020 if sc=1 then me$="a programazonosito mar definial
t !":goto 58700
10030 gosub 57500
10040 goto 5020
10050 :
10500 rem definicios resz kezdete
10505 ou$="jmp definitionsende": gosub 59000
10507 ou$=be$+"immersys .bl 5" : gosub 59000
10510 df=1 : goto 5000
10520 :
11000 rem definicios resz vege
11005 ou$="definitionsende .m": gosub 59000
11010 df=0 : goto 5000
11020 :
11500 rem programazonosito vizsgalat
```

```
11505 gosub 56000
11510 be$=i1$ : ty%=1 : gosub 57000
11520 if sc=0 then me$="a Programazonosito nem definial
t !":goto 58700
11540 goto 5020
11550 :
12000 rem Proram vege
12010 ou$="lda #$76" : gosub 59000
12020 ou$="ldy #$a3" : gosub 59000
12030 ou$="jsr $able" : gosub 59000
12040 ou$="jmp $a480" : gosub 59000
12050 ou$=".end" : gosub 59000
12060 goto 5000
12070 :
12500 rem alProgramazonosito definicio
12505 gosub 56000
12510 be$=i1$ : ty%=2 : gosub 57000
12520 if sc=1 then me$="a Programazonosito mar definial
t !":goto 58700
12530 gosub 57500
12535 ou$="u-"+be$+" .m" : gosub 59000
12540 goto 5020
12550 :
13000 rem alProgramazonosito vizsgalat
13005 gosub 56000
13010 be$=i1$ : ty%=2 : gosub 57000
13020 if sc=0 then me$="a Programazonosito nem definial
t !":goto 58700
13030 ou$="rts" : gosub 59000
13040 goto 5020
13050 :
13500 rem kiiras
13510 gosub 13600
13520 goto 5020
13530 :
13600 gosub 55000
13610 if i<>255 then me$="uj szimbolumot varok !": gosu
b 58500 : return
13630 gosub 55000
13640 if i=55 then i1$="": gosub 56040: goto 13800
13650 i1$="": gosub 56340
13652 d9=d9+1
13654 d9$=mid$(str$(d9),2,len(str$(d9))-1)
13656 ou$="jmp de"+d9$ : gosub 59000
13660 ou$=".c" : gosub 59000
13670 ou$=".t "+chr$(34)+i1$+chr$(34) : gosub 59000
13675 ou$="de"+d9$+" .m": gosub 59000
13680 ou$="ldy # ch" : gosub 59000
```

```
13690 ou$="lda # cl" : gosub 59000
13700 ou$="jsr $able" : gosub 59000
13710 return
13800 be$=i1$ : ty%=3 : gosub 57000
13810 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
13820 gosub 54000
13830 ou$="jsr aus" : gosub 59000
13840 return
13850 :
14000 rem kiiras soremelessel
14010 gosub 13600
14020 ou$="jsr zvor" : gosub 59000
14030 goto 5020
14040 :
14500 rem soremeles
14510 ou$="jsr zvor" : gosub 59000
14520 goto 5000
14530 :
15000 rem kepernyotorles
15010 ou$="jsr $e544" : gosub 59000
15020 goto 5000
15030 :
15500 rem kursor oszlop
15510 gosub 56500
15520 if val(i1$)>40 then me$="oszlopszam nagyobb mint
40 !": goto 58700
15530 if val(i1$)<1 then me$="oszlopszam kisebb mint 1
!": goto 58700
15540 ou$="sec": gosub 59000
15550 ou$="jsr $fff0" : gosub 59000
15560 ou$="clc": gosub 59000
15570 ou$="ldy #"+str$(val(i1$)-1): gosub 59000
15580 ou$="jsr $fff0" : gosub 59000
15590 goto 5020
15600 :
16000 rem kursorsor
16010 gosub 56500
16020 if val(i1$)>24 then me$="sorszam nagyobb mint 24
!": goto 58700
16030 if val(i1$)<1 then me$="sorszam kisebb mint 1 !":
 goto 58700
16040 ou$="sec": gosub 59000
16050 ou$="jsr $fff0" : gosub 59000
16060 ou$="clc": gosub 59000
16070 ou$="ldx #"+str$(val(i1$)-1): gosub 59000
16080 ou$="jsr $fff0" : gosub 59000
16090 goto 5020
```

```
16100 :
16500 rem ascii kiiras
16510 gosub 56500
16520 ou$="lda #"+i1$: gosub 59000
16530 ou$="jsr $ffd2" : gosub 59000
16540 goto 5020
16550 :
17000 rem eszkoz
17010 gosub 56000
17020 if i1$="kepernyo" then 17200
17030 if i1$="nyomtato" then 17300
17040 m@$="ismeretlen eszkoz !": goto 58700
17200 rem kepernyo
17210 ou$="jsr zvor" : gosub 59000
17220 ou$="jsr $ffcc" : gosub 59000
17230 ou$="lda #4": gosub 59000
17240 ou$="jsr $ffc3" : gosub 59000
17250 goto 5020
17300 rem nyomtato
17310 ou$="lda #4": gosub 59000
17320 ou$="sta 184": gosub 59000
17330 ou$="sta 186": gosub 59000
17340 ou$="lda #7": gosub 59000
17350 ou$="sta 185": gosub 59000
17360 ou$="lda #0": gosub 59000
17370 ou$="sta 183": gosub 59000
17380 ou$="jsr $ffc0" : gosub 59000
17390 ou$="ldx #4": gosub 59000
17400 ou$="jsr $ffc9" : gosub 59000
17410 goto 5020
17500 rem keret
17510 gosub 56000
17520 ff=-1
17530 if i1$="fekete" then ff=0
17540 if i1$="feher" then ff=1
17550 if i1$="piros" then ff=2
17560 if i1$="turkisz" then ff=3
17570 if i1$="lila" then ff=4
17580 if i1$="zold" then ff=5
17590 if i1$="kek" then ff=6
17600 if i1$="sarga" then ff=7
17610 if i1$="narancs" then ff=8
17620 if i1$="barna" then ff=9
17630 if i1$="vilagospiros" then ff=10
17640 if i1$="szurkea" then ff=11
17650 if i1$="szurkeb" then ff=12
17660 if i1$="szurkec" then ff=15
17670 if i1$="vilagoszold" then ff=13
```

```
17680 if i1$="vila9osbarna" then ff=14
17690 if ff=-1 then me$="ismeretlen szin !": 9oto 58700

17700 ou$="lda #"+str$(ff): 9osub 59000
17710 ou$="sta 53280": 9osub 59000
17720 9oto 5020
18000 rem hatter
18010 9osub 56000
18020 ff=-1
18030 if i1$="fekete" then ff=0
18040 if i1$="feher" then ff=1
18050 if i1$="piros" then ff=2
18060 if i1$="turkisz" then ff=33
18070 if i1$="lila" then ff=4
18080 if i1$="zold" then ff=5
18090 if i1$="kek" then ff=6
18100 if i1$="sar9a" then ff=7
18110 if i1$="naranc9" then ff=8
18120 if i1$="barna" then ff=9
18130 if i1$="vila9osPiros" then ff=10
18140 if i1$="szurkea" then ff=11
18150 if i1$="szurkeb" then ff=12
18160 if i1$="szurkec" then ff=15
18170 if i1$="vila9oszold" then ff=13
18180 if i1$="vila9osbarna" then ff=14
18190 if ff=-1 then me$="ismeretlen szin !": 9oto 58700

18200 ou$="lda #"+str$(ff): 9osub 59000
18210 ou$="sta 53281": 9osub 59000
18220 9oto 5020
18500 rem iras
18510 9osub 56000
18520 ff=-1
18530 if i1$="fekete" then ff=144
18540 if i1$="feher" then ff=5
18550 if i1$="piros" then ff=28
18560 if i1$="purpur" then ff=156
18580 if i1$="zold" then ff=30
18590 if i1$="kek" then ff=31
18600 if i1$="sar9a" then ff=158
18610 if i1$="narancs" then ff=159
18690 if ff=-1 then me$="ismeretlen szin !": 9oto 58700

18700 ou$="lda #"+str$(ff): 9osub 59000
18710 ou$="jsr $ffd2": 9osub 59000
18720 9oto 5020
19000 rem adatbevitel
19010 9osub 56000
```

```
19020 be$=i1$ : ty%=3 : gosub 57000
19030 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
19040 ou$="jsr ein" : gosub 59000
19050 gosub 53000
19060 goto 5020
19400 rem ertekadas valtozobol
19410 gosub 56000
19420 be$=i1$ : ty%=3 : gosub 57000
19430 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
19440 gosub 54000
19450 gosub 56030
19460 be$=i1$ : ty%=3 : gosub 57000
19470 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
19480 gosub 53000
19490 goto 5020
19500 rem ertekadas
19510 va$=""
19520 gosub 55000: gosub 55000: gosub 55000
19524 if i=41 then 19400
19530 gosub 56500: va$=i1$
19535 gosub 55000: gosub 55000
19540 if i<>49 then 19600
19550 va$=va$+".": gosub 56500: va$=va$+i1$
19555 gosub 55000: gosub 55000
19600 if i<>51 then 19900
19610 va$=va$+"e"
19620 gosub 55000: gosub 55000
19630 if i=59 then 19700
19640 if i=52 then va$=va$+"+"
19650 if i=53 then va$=va$+"-"
19700 i1$="": gosub 56540
19710 va$=va$+i1$
19900 gosub 51000
19910 gosub 56000
19920 be$=i1$ : ty%=3 : gosub 57000
19930 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
19940 d9=d9+1
19942 d9$=mid$(str$(d9),2,len(str$(d9))-1)
19946 ou$="jmp de"+d9$ : gosub 59000
19950 ou$="dd"+d9$+" .m": gosub 59000
19952 ou$=" .b"+str$(v0) : gosub 59000
19953 ou$=" .b"+str$(v1) : gosub 59000
19954 ou$=" .b"+str$(v2) : gosub 59000
19955 ou$=" .b"+str$(v3) : gosub 59000
```

```
19956 ou$=" .b"+str$(v4) : gosub 59000
19960 ou$="de"+d9$+" .m": gosub 59000
19962 hi$=be$: be$="dd"+d9$: gosub 54000
19970 be$=hi$: gosub 53000
19980 goto 5020
19990 :
20000 rem osszadas
20010 gosub 56000
20020 be$=i1$ : ty%=3 : gosub 57000
20030 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
20040 gosub 54000
20050 gosub 56030
20060 be$=i1$ : ty%=3 : gosub 57000
20070 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
20080 ou$="ldy #hb-"+be$: gosub 59000
20090 ou$="lda #lb-"+be$: gosub 59000
20100 ou$="jsr $b867" : gosub 59000
20110 gosub 56030
20120 be$=i1$ : ty%=3 : gosub 57000
20130 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
20140 gosub 53000
20150 goto 5020
20500 rem kivonas
20510 gosub 56000
20520 be$=i1$ : ty%=3 : gosub 57000
20530 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
20540 gosub 54000
20550 gosub 56030
20560 be$=i1$ : ty%=3 : gosub 57000
20570 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
20580 ou$="ldy #hb-"+be$: gosub 59000
20590 ou$="lda #lb-"+be$: gosub 59000
20600 ou$="jsr $b850" : gosub 59000
20610 gosub 56030
20620 be$=i1$ : ty%=3 : gosub 57000
20630 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
20640 gosub 53000
20650 goto 5020
21000 rem szorzas
21010 gosub 56000
21020 be$=i1$ : ty%=3 : gosub 57000
21030 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
```

```
21040 gosub 54000
21050 gosub 56030
21060 be$=i1$ : ty%=3 : gosub 57000
21070 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
21080 ou$="ldy #hb-"+be$: gosub 59000
21090 ou$="lda #lb-"+be$: gosub 59000
21100 ou$="jsr $ba28" : gosub 59000
21110 gosub 56030
21120 be$=i1$ : ty%=3 : gosub 57000
21130 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
21140 gosub 53000
21150 goto 5020
21500 rem osztas
21510 gosub 56000
21520 be$=i1$ : ty%=3 : gosub 57000
21530 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
21540 hi$=be$
21550 gosub 56030
21560 be$=i1$ : ty%=3 : gosub 57000
21570 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
21575 gosub 54000
21580 ou$="ldy #hb-"+hi$: gosub 59000
21590 ou$="lda #lb-"+hi$: gosub 59000
21600 ou$="jsr $bb0f" : gosub 59000
21610 gosub 56030
21620 be$=i1$ : ty%=3 : gosub 57000
21630 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
21640 gosub 53000
21650 goto 5020
22000 rem hatvanyozas
22010 gosub 56000
22020 be$=i1$ : ty%=3 : gosub 57000
22030 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
22040 gosub 52000
22050 gosub 56030
22060 be$=i1$ : ty%=3 : gosub 57000
22070 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
22080 gosub 54000
22090 ou$="jsr $bf7b" : gosub 59000
22110 gosub 56030
22120 be$=i1$ : ty%=3 : gosub 57000
```

```
22130 if sc=0 then me$="az azonosito nem definialt !":9
oto 58700
22140 9osub 53000
22150 9oto 5020
22500 rem fu99venyek
22510 9osub 56000
22520 ff$=""
22530 if i1$="abszolut" then ff$="$bc58"
22540 if i1$="actan9ens" then ff$="$e30e"
22550 if i1$="cosinus" then ff$="$e264"
22560 if i1$="exPonens" then ff$="$bfed"
22570 if i1$="e9esz" then ff$="$bccc"
22580 if i1$="lo9aritmus" then ff$="$b9ea"
22590 if i1$="tan" then ff$="$b80d"
22600 if i1$="veletlen" then ff$="$e097"
22610 if i1$="elojel" then ff$="$bc39"
22620 if i1$="sinus" then ff$="$e26b"
22630 if i1$="negyzetgyok" then ff$="$bf71"
22640 if i1$="tan9ens" then ff$="$e2b4"
22650 if ff$="" then me$="ismeretlen fu99veny !": 9oto
58700
22660 9osub 56030
22670 be$=i1$ : ty%=3 : 9osub 57000
22680 if sc=0 then me$="az azonosito nem definialt !":9
oto 58700
22690 9osub 54000
22700 ou$="jsr "+ff$ : 9osub 59000
22705 9osub 55000: 9osub 55000
22710 if i=27 then 22800
22720 9osub 56000: 9osub 55000: 9osub 55000
22800 be$=i1$ : ty%=3 : 9osub 57000
22810 if sc=0 then me$="az azonosito nem definialt !":9
oto 58700
22820 9osub 53000
22830 9oto 5000
23000 rem Pont
23500 rem dontes
23505 ef=ef+1: iz=iz+1: i(iz)=ef
23506 ef$=mid$(str$(i(iz)),2,len(str$(i(iz))))
23510 9osub 56000
23520 be$=i1$ : ty%=3 : 9osub 57000
23530 if sc=0 then me$="az azonosito nem definialt !":9
oto 58700
23540 9osub 54000
23550 9osub 55000: 9osub 55000
23560 oP=i
23570 9osub 56000
23580 be$=i1$ : ty%=3 : 9osub 57000
```

```
23590 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
23600 ou$="ldy #hb-"+be$: gosub 59000
23610 ou$="lda #lb-"+be$: gosub 59000
23620 ou$="jsr $bc5b" : gosub 59000
23630 if op=43 then ou$="cmp #0" : gosub 59000
23640 if op=43 then ou$="beq th"+ef$ : gosub 59000
23650 if op=44 then ou$="cmp #0" : gosub 59000
23660 if op=44 then ou$="bne th"+ef$ : gosub 59000
23670 if op=45 then ou$="cmp #$ff" : gosub 59000
23680 if op=45 then ou$="beq th"+ef$ : gosub 59000
23690 if op=46 then ou$="cmp #1" : gosub 59000
23700 if op=46 then ou$="bne th"+ef$ : gosub 59000
23710 if op=47 then ou$="cmp #1" : gosub 59000
23720 if op=47 then ou$="beq th"+ef$ : gosub 59000
23730 if op=48 then ou$="cmp #$ff" : gosub 59000
23740 if op=48 then ou$="bne th"+ef$ : gosub 59000
23800 ou$="jmp el"+ef$ : gosub 59000
23810 ou$="th"+ef$+" .m" : gosub 59000
23820 goto 5020
24000 rem kulonben
24005 ef$=mid$(str$(i(iz)),2,len(str$(i(iz))))
24010 ou$="jmp en"+ef$ : gosub 59000
24020 ou$="el"+ef$+" .m" : gosub 59000
24030 goto 5000
24500 rem dontes vege
24510 ef$=mid$(str$(i(iz)),2,len(str$(i(iz))))
24520 ou$="en"+ef$+" .m" : gosub 59000
24525 iz=iz-1
25000 rem vegtelen ciklus
25005 gosub 56000
25010 be$=i1$ : ty%=4 : gosub 57000
25020 if sc=1 then me$="a ciklusazonosito mar definialt
 !":goto 58700
25030 gosub 57500
25035 ou$="e-"+be$+" .m" : gosub 59000
25040 goto 5020
25500 rem vegtelen ciklus vege
25505 gosub 56000
25510 be$=i1$ : ty%=4 : gosub 57000
25520 if sc=0 then me$="a ciklusazonosito nem definialt
 !":goto 58700
25530 ou$="jmp e-"+i1$ : gosub 59000
25535 ou$="ee-"+i1$+" .m" : gosub 59000
25540 goto 5020
26000 rem kimenet
26005 gosub 56000
26010 be$=i1$ : ty%=4 : gosub 57000
```

```
26020 if sc=0 then me$="a ciklusazonosito nem definialt
 !":goto 58700
26030 ss$=be$
26040 gosub 56030
26050 be$=i1$ : ty%=3 : gosub 57000
26060 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
26070 gosub 54000
26080 gosub 55000: gosub 55000
26090 op=1
26100 gosub 56000
26110 be$=i1$ : ty%=3 : gosub 57000
26120 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
26130 ou$="ldy #hb-"+be$: gosub 59000
26140 ou$="lda #lb-"+be$: gosub 59000
26150 ou$="jsr $bc5b" : gosub 59000
26160 d9=d9+1
26170 d9$=mid$(str$(d9),2,len(str$(d9))-1)
26200 if op=43 then ou$="cmp #0" : gosub 59000
26210 if op=43 then ou$="bne sz-"+d9$ : gosub 59000
26220 if op=44 then ou$="cmp #0" : gosub 59000
26230 if op=44 then ou$="beq sz-"+d9$ : gosub 59000
26240 if op=45 then ou$="cmp #255" : gosub 59000
26250 if op=45 then ou$="bne sz-"+d9$ : gosub 59000
26260 if op=46 then ou$="cmp #1" : gosub 59000
26270 if op=46 then ou$="beq sz-"+d9$ : gosub 59000
26280 if op=47 then ou$="cmp #1" : gosub 59000
26290 if op=47 then ou$="bne sz-"+d9$ : gosub 59000
26300 if op=48 then ou$="cmp #255" : gosub 59000
26310 if op=48 then ou$="beq sz-"+d9$ : gosub 59000
26440 ou$="jmp ee-"+ss$ : gosub 59000
26450 ou$="sz-"+d9$+" .m" : gosub 59000
26480 goto 5020
26500 rem indexciklus
26510 gosub 56000
26520 be$=i1$ : ty%=3 : gosub 57000
26530 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
26540 dx$=be$
26550 gosub 56030
26560 be$=i1$ : ty%=3 : gosub 57000
26570 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
26580 ux$=be$
26590 gosub 56030
26600 be$=i1$ : ty%=3 : gosub 57000
26610 if sc=0 then me$="az azonosito nem definialt !":g
oto 58700
```

```
26620 ox$=be$
26630 s$(sl)=dx$ : sl=sl+1
26640 goto 5020
27000 rem cikluskezdet
27010 be$=ux$ : gosub 54000
27020 be$=dx$ : gosub 53000
27030 ou$="ia-"+s$(sl-1)+" .m" : gosub 59000
27040 be$=dx$ : gosub 54000
27050 ou$="ldy #hb-"+ox$: gosub 59000
27060 ou$="lda #lb-"+ox$: gosub 59000
27070 ou$="jsr $bc5b" : gosub 59000
27080 d9=d9+1
27090 d9$=mid$(str$(d9),2,len(str$(d9))-1)
27100 ou$="cmp #1" : gosub 59000
27110 ou$="bne ia"+d9$ : gosub 59000
27120 ou$="jmp ie-"+s$(sl-1) : gosub 59000
27130 ou$="ia"+d9$+" .m" : gosub 59000
27140 ou$="lda #$e8": gosub 59000
27150 ou$="ldy #$bf": gosub 59000
27160 ou$="jsr $b867" : gosub 59000
27170 be$=dx$ : gosub 53000
27180 goto 5000
27500 rem indexcciklus ve9e
27505 ou$="jmp ia-"+s$(sl-1) : gosub 59000
27510 ou$="ie-"+s$(sl-1)+" .m" : gosub 59000
27520 sl=sl-1
27530 goto 5000
28000 rem u9ras jelzo
28010 gosub 56000
28020 ou$="s-"+i1$+" .m" : gosub 59000
28030 goto 5020
28500 rem u9ras
28510 gosub 56000
28520 ou$="jmp s-"+i1$ : gosub 59000
28530 goto 5020
29000 rem hivas
29010 gosub 56000
29020 ou$="jsr u-"+i1$ : gosub 59000
29030 goto 5020
30000 rem sorszam
30010 if sp=1 then ou$="jsr spur" : gosub 59000
30020 get#in,i$ : if i$="" then i$=chr$(0)
30030 if sp=1 then ou$="ldx #"+str$(asc(i$)) : gosub 59
000
30040 get#in,i$ : if i$="" then i$=chr$(0)
30050 if sp=1 then ou$="lda #"+str$(asc(i$)) : gosub 59
000
30060 if sp=1 then ou$="jsr $bdcd" : gosub 59000
```

```
30070 goto 5000
30080 :
31000 rem Programm vege
31010 Print#ou,cchr$(0)chr$(0);
31020 close ou
31030 close in
31040 close 15
31050 close Pr
31055 Print : Print
31060 Print"Vege a szemantikai analizisnek."
31065 Print :Print
31070 Print"Az assemblert a kovetkezokeppen tolthetjuk
be :"
31075 Print : Print
31080 Print"load"+chr$(34)+"assembler"+chr$(34)+",8"
31085 Print : Print
31090 Print"Az assemblerprogramot a kovetkezokeppen tol
thetjuk be :"
31095 Print : Print
31100 Print"load"+chr$(34)+"mini-ass"+chr$(34)+",8"
31110 end
51000 rem va$ atalakitasa
51005 if val(va$)=0 then 51200
51010 v1=0: v5=0: v6=129: v7=val(va$): if v7<0 then v1=
128: v7=abs(v7)
51020 if v7<1 and v7<>0 then v6=v6-1 : v7=v7*2 : goto 5
1020
51030 v8=v7
51040 if v8>=2 then v8=int(v8/2) : v5=v5+1 : goto 51040

51050 v0=v5+v6
51060 v7=(v7-2↑v7)/2↑v7
51070 v7=v7*128: v1=v1+int(v7):v7=v7-int(v7)
51080 v7=v7*256: v2=int(v7):v7=v7-v2
51090 v7=v7*256: v3=int(v7):v7=v7-v3
51100 v7=v7*256: v4=int(v7):v7=v7-v4
51110 return
51200 v0=0: v1=0: v2=0: v3=0: v4=0: return
52000 rem valtozo az argumentumba kerul
52010 ou$="ldy #hb-"+be$: gosub 59000
52020 ou$="lda #lb-"+be$: gosub 59000
52030 ou$="jsr $ba8c" : gosub 59000
52040 return
53000 rem fac a  valtozoba kerul
53010 ou$="ldy #hb-"+be$: gosub 59000
53020 ou$="ldx #lb-"+be$: gosub 59000
53030 ou$="jsr $bbd4" : gosub 59000
53040 return
```

```
54000 rem vatozo a fac-ba  kerul
54010 ou$="ldy #hb-"+be$: gosub 59000
54020 ou$="lda #lb-"+be$: gosub 59000
54030 ou$="jsr $bba2" : gosub 59000
54040 return
54050 :
55000 rem kovetkezo szimbolum allitasa
55010 get#in,i$ : if i$="" then i=0 : return
55020 i=asc(i$) : return
55030 :
56000 rem azonositot var
56005 i1$=""
56010 gosub 55000
56020 if i<>255 then me$="uj szimbolumot varok !": gosu
b 58500: return
56030 gosub 55000
56040 i1$="":if i<>55 then me$="azonositot varok !": go
sub 58500: return
56050 gosub 55000
56060 if i=255 then return
56070 i1$=i1$+i$ : goto 56050
56080 :
56300 rem karakterlancot var
56305 i1$=""
56310 gosub 55000
56320 if i<>255 then me$="uj szimbolumot varok !": gosu
b 58500: return
56330 gosub 55000
56340 if i<>57 then me$="karakterlancot varok !": gosub
 58500: return
56350 gosub 55000
56360 if i=255 then return
56370 i1$=i1$+i$ : goto 56050
56380 :
56500 rem egesz szamot var
56505 i1$=""
56510 gosub 55000
56520 if i<>255 then me$="uj szimbolumot varok !": gosu
b 58500: return
56530 gosub 55000
56540 if i<>59 then me$="egesz szamot varok !": gosub 5
8500: return
56550 gosub 55000
56560 if i=255 then return
56570 i1$=i1$+i$ : goto 56050
56580 :
57000 rem azonosito mar definialt?
57005 sc=0
```

```
57010 for t=0 to l
57020 if b$(t)=be$ then if t%(t)=ty% then sc=1
57030 next t
57040 return
57050 :
57500 rem azonosito bevitel
57510 b$(l)=be$: t%(l)=ty%
57520 l=l+1
57530 return
57540 :
58000 rem hibara varas
58010 Print#Pr,chr$(13)
58020 Print#Pr,me$
58030 Print#Pr,chr$(13)
58040 Print#Pr,"<RETURN>"
58050 get w$ : if w$<>chr$(13) then 58050
58060 goto 5000
58070 :
58500 rem hibara varas
58510 Print#Pr,chr$(13)
58520 Print#Pr,me$
58530 Print#Pr,chr$(13)
58540 Print#Pr,"<RETURN>"
58550 get w$ : if w$<>chr$(13) then 58550
58560 return
58700 rem hibara varas
58710 Print#Pr,chr$(13)
58720 Print#Pr,me$
58730 Print#Pr,chr$(13)
58740 Print#Pr,"<RETURN>"
58750 get w$ : if w$<>chr$(13) then 58750
58760 goto 5020
58770 :
59000 rem ou$ atirasa ou-ba
59010 pc=pc+len(ou$)+5
59020 p1=int(pc/256)
59030 p2=pc-p1*256
59040 zl=zl+10
59050 z1=int(zl/256)
59060 z2=zl-z1*256
59070 Print#ou,chr$(p2)chr$(p1)chr$(z2)chr$(z1);ou$;chr
$(0);
59080 Print#Pr,str$(z1)+" "+ou$
59090 ou$=""
59100 return
59120 :
60000 rem hibacsatorna olvasasa
60010 input#15,en,em$,et,es : if en=0 then return
```

```
60020 print#pr,chr$(13)chr$(13)
60030 print#pr,"Hiba a lemezen !"
60040 print#pr,"hiba szama : ";em
60050 print#pr,"hibauzenet : ";em$
60060 print#pr,"hibas sav  : ";et
60070 print#pr,"hibas szektor : ";es
60080 print#pr,"Program leallt !"
60090 close ou
60100 close in
60110 close 15
60120 close pr
60130 end
61000 open 8,8,8,"miniatur-syn,s,r"
61010 open 4,4
61020 get#8,i$ : if st=64 then 61100
61025 if i$="" then i=0 : goto 61040
61030 i=asc(i$)
```

6. AZ ASSEMBLER PROGRAMNYELV

6.1 Rövid bevezetés az assembler programnyelvbe

A bevezetést azoknak szánjuk, akik még nem programoztak assembler nyelven vagy gépi kódban. Nem kívánunk teljes értékű gépi kód ismertetőt adni, csak azokat az utasításokat tárgyaljuk meg, amelyeket a továbbiakban használni fogunk. A könyvünkben ismertetett assembler nyelv a 6502-es, ill a 6510-es mikroprocesszor összes utasítását és címzési módját tartalmazza.

Ha az assembler nyelven való programozás elvét néhány utasításon keresztül megismertük, akkor a későbbiekben nem jelent problémát ismereteinket továbbfejleszteni. Számítógépünk mikroprocesszorának nagy előnye az egyszerű felépítés, és így jól alkalmazható tanulásra is. A mikroprocesszor egyszerű felépítése ellenére nagyon jó teljesítőképességű, amint azt az Olvasó tapasztalni fogja.

A 6502-es és a 6510-es mikroprocesszorok között nem teszünk különbséget, mivel ugyanazokat az utasításokat fogadják el.

A gépi programozásról csak annyit jegyezzünk meg, hogy az ASSEMBLER nevű program az assembler nyelvű parancsokat gépi kódra le tudja fordítani.

Ezért elégszünk meg a sokkal egyszerűbb assembler nyelven való programozással.

Ebben a fejezetben a számítógépünk felépítését egyszerűsítve fogjuk tárgyalni.

A számítógép egy rekeszekből felépített tárból és a mikroprocesszorból áll, amely az egyes rekeszek tartalmát meg tudja változtatni. A rekeszeket 0-tól 65 535-ösig számozzuk. Az assembler nyelvű programokban a számokat nem tízes, hanem tizenhatos számrendszerben adjuk meg, mivel a tizenhatos számrendszer közelebb áll a számítógép felépítéséhez.

Kezdők számára azonban a tizenhatos számrendszer problémát okozhat, ezért a továbbiakban a tízes számrendszerbeli alakot is használni fogjuk.

A tizenhatos számrendszerbeli számokat $-jellel kezdjük. Ezek szerint a tár $0000-tól $FFFF-ig terjed. A DISASSEMBLER nevű programban található egy rutin, amelyik a számokat egyik számrendszerből a másikba átalakítja.

A tárban a programokat a mikroprocesszor hajtja végre. Ezek a programok az adatokat viszik át a tár egyik helyéről a másikba, ill. megváltoztatják azok értékét.

Ebben a fejezetben főleg az adatátviteli parancsokkal foglalkozunk majd.

Az adatok értékének módosítására a C64-es programjait fogjuk használni.

A mikroprocesszor működésének megértéséhez ismerkedjünk meg a regiszterek fogalmával.

A mikroprocesszor regiszterei közül az alábbiakkal foglalkozunk:

- akkumulátor;
- X regiszter;
- Y regiszter;
- állapotregiszter.

Az akkumulátor a mikroprocesszornak az a munkaregisztere, amelyet a legtöbb művelet végrehajtásához használunk.

Az X és az Y segédregisztereket gyakran számlálásra használjuk. Indexregisztereknek is nevezik őket (l. a címzési módokat).

Az állapotregiszter a processzor állapotával kapcsolatos információkat tartalmazza.

Nézzük a következő példát!

A 2055-ös tárhely tartalmát a 7256-os tárhelyre kell átvinni. A 2055-ös tárhely tartalma az akkumulátorba kerül, majd az akkumulátor tartalmát a 7256-os tárhelyre visszük át. Az ehhez szükséges utasítások:

```
LDA 2055
STA 7256
```

Az LDA azt jelenti, hogy töltsd fel az akkumulátort, az STA pedig azt, hogy tárold az akkumulátor tartalmát. A rövidítéseket mnemoniknak nevezzük. A mnemonikok után ún. operandusok állnak. A mikroprocesszor az operandusokból tudja meg, hogy a parancs melyik tárhelyre vonatkozik. Az operandusok tízes vagy tizenhatos számrendszerbeli számok és szimbólumok lehetnek. A mi assembler nyelvű programunkban a szimbólumoknak mindig betűvel kell kezdődniük, ezután bármilyen karaktersorozattal folytatódhatnak és hosszuk tetszőleges lehet. Az assembler nyelvű programban a szimbólumokhoz mindig egy szám rendelődik és a gépi kódra való fordítás után mindig ez a szám áll a szimbólum helyett. Egy számnak a szimbólumhoz való rendelése az ASSEMBLER-t vezérlő ún. pszeudoutasítással megy végbe. Erről részletesebben Az ASSEMBLER című fejezetben olvashatunk.

A mikroprocesszor számára különböző lehetőségek vannak arra, hogy megtalálja azt a tárcímet, amellyel az aktuális utasítás során dolgoznia kell. A különböző címzési lehetőségeket az LDA paranccsal mutatjuk be. A címzési lehetőségek minden parancsnál mások, de a legtöbb címzési lehetőség az LDA parancsnál van.

Közvetlen címzés

A közvetlen címzésnél az utasítás operandusa nem címet, hanem értéket jelent, vagyis az akkumulátorba a 77-es számot kell betölteni.

```
LDA #77
```

Az akkumulátor tartalma tehát nem a 77-es címen lévő érték, hanem 77 lesz!

Nullás lapú címzés

Mit jelent a nullás lap? A tárat 256 laposnak képzelhetjük el (a számozás 0-tól 255-ig tart). A tár első lapja az ún. nullás lap. Erre a lapra a fent említett speciális címzési mód parancsai érvényesek. Ezekkel a gépi kódú parancsok rövidebben és gyorsabban végrehajthatók. A nullás lapra általában olyan változókat helyezünk, amelyekre a program futása során többször is szükségünk van. Ennél a címzésnél az operandus értéke 0 és 255 között lehet.

```
LDA 77
```

A 77-es tárrekesz tartalma az akkumulátorba kerül.

Nullás lapú címzés X index-szel

```
LDX #1
LDA 77, X
```

A fenti két utasítás hatása a következő:

1) az X regiszterbe direkt módon 1-et töltünk;
2) az akkumulátorba a 77 + X = 78. tárrekesz tartalma kerül. Az X regiszter tartalma tehát hozzáadódik a címhez, s így jön létre az új cím az LDA parancs számára.

Az indexregiszterrel való címzés lehetőséget nyújt arra, hogy egy gépi kódú program végrehajtása után határozzuk meg azt a tárcímet, amelyikről olvasni kell. Ennek a lehetőségnek nagyon sok előnye van.

Nullás lapú címzés Y index-szel

Ez a címzés csak két parancsnál, az LDX-nél és az STX-nél lehetséges.

LDX operandus, Y
STX operandus, Y

Az STX az LDX ellentéte. Míg az LDX-szel az X regiszterbe töltünk be adott értéket, az STX paranccsal az X regiszter tartalmát adott tárhelyre visszük ki.

A cím kiszámítása az X regiszternél leírtakhoz hasonló, itt azonban az Y regiszter tartalmát kell figyelembe venni.

Abszolút címzés

Az abszolút címzésnél az operandus értéke 0 és 65 535 közé esik, így minden tárrekesz címezhető.

LDA 47 000

A 47 000-es tárrekesz tartalma az akkumulátorba kerül.

Abszolút címzés X index-szel

Ugyanúgy működik, mint a nullás lapú címzés X index-szel, csak az operandus 0-tól 65 535-ig terjedhet.

Abszolút címzés Y index-szel

Ez is megegyezik a nullás lapú Y index-szel való címzéssel, a különbség csak az, hogy az operandus 0-tól 65 535-ig terjedhet. Ez a címzési mód egy sor további parancsra ad lehetőséget.

Abszolút indirekt címzés

Erre a címzési módra csak egyetlen parancsnál (JMP) van lehetőség, de a címképzés további lehetőségeinek alapjául szolgál. Szükség lehet olyan címek használatára is, amelyeket valóban ki lehet számítani. A kiszámított címet két tárrekeszbe helyezzük, és felhasználáskor a parancsban a két tárrekesz közül csak az elsőnek a címét kell megadni. A mikroprocesszor a tárból először az aktuális címet hozza elő. Miért van két tárhelyre szükség? Azért, mert egy tárhelyen csak 0-tól 255-ig terjedő értékeket lehet megadni. Így egy lapnak a helyeire tudunk hivatkozni, de tudnunk kell még azt is, hogy melyik lapról van szó. Hogyan kell kiértékelni a két tárrekeszben lévő értéket? Ha az érték 8 tizenhatos számrendszerbeli szám, akkor egyszerű a dolgunk. Az első két számjegy az oldalt, a második kettő az oldalon belüli helyet adja meg.

A $AAD2 tárhelynél kezdődik a szükséges cím. A cím legyen $FFD2. Ilyenkor a $AAD2-ben $D2, a $AAD3-ban pedig a $FF érték van.
Az assemblernyelven való programozásnál a címet mindig a fenti módon tároljuk.

A mikroprocesszor a parancsokat a tárból olvassa ki, és ha JMP parancsot talál, akkor a következő parancsot a JMP utáni címről olvassa, ill. számolja ki.

```
JMP $FFD2
JMP ($FFD2)
```

Az első parancsnál abszolút címzést alkalmaztunk és a következő parancs a $FFD2-ben van.

A második parancsnál indirekt címzést írtunk.

A következő parancs azon a címen található, amely a $FFD2 és a $FFD3 rekeszekben van.

Indexelt indirekt címzés

Ez a címzés csak az X regiszterrel lehetséges.

```
LDX #10
LDA (20, X)
```

Az akkumulátorba töltendő tárcímet úgy kapjuk meg, hogy az operandus értékéhez (a mi esetünkben 20) hozzáadjuk az X regiszter tartalmát.

$20 + 10 = 30$

Ebből a mikroprocesszor tudja, hogy a 30-as és a 31-es tárrekeszekben cím található. Legyen például a 30-as rekesz tartalma 10, a 31-esé pedig 1, így a cím $10 + 1*256 = 266$-os. Az akkumulátorba tehát a 266-os tárrekesz tartalma kerül. Az operandus értéke 0 és 255 közé eshet.

Indirekt indexelt címzés

Ebben az esetben az Y az indexregiszter.

```
LDY #10
LDA (20), Y
```

A mikroprocesszor a 20-as és a 21-es tárrekeszekben címet vár. Legyen a 20-as rekesz tartalma 10, a 21-esé pedig 1. Így ez a cím a 10+1*256 = 266-os tárrekeszre mutat. Ehhez a címhez hozzá kell adni az Y regiszter tartalmát, ami esetünkben 10. Így az akkumulátorba a 266+10 = 276-os tárrekesz tartalma kerül. Az operandus értéke 0-tól 255-ig terjedhet, a mi esetünkben ez 20 volt. Mindenképpen nullás lapú címről van szó.

Relatív címzés

Ez a címzési mód az utasítások speciális csoportjánál a feltételes vezérlést átadó utasításoknál fordul elő. Mit jelent a feltételes vezérlést átadó utasítás? Ez olyan átirányító utasítás, amelyet az állapotregiszter tartalmától függően hajtunk végre, így egy programon belül elágazások is szerepelhetnek. Az elágaztatáshoz az állapotregiszter tartalmának ismertetésére van szükség.

Az állapotregiszter 8 bites, ezek tartalma egyenként lekérdezhető. Az első bitet Z-nek nevezzük. Egy bit értéke 0 vagy 1 lehet. Így sok különböző átirányító utasítás kiadására van lehetőségünk, ezek közül mi most csak kettőt emelünk ki, a BEQ-t és a BNE-t.

```
BEQ operandus
BNE operandus
```

Az első példában: ha Z értéke 1, akkor az operandus értékének megadásához ugrunk, ha Z = 0, akkor a soron következő parancsokat hajtjuk végre. A vezérlést átadó utasításhoz képest 128 tárrekesszel lehet vissza és 127 tárrekesszel előre ugrani. Erre vonatkozik a relatív címzés elnevezés.

Az assembler nyelvű programokban a címkéket is lehet használni. Ha az operandus egy ilyen szimbolikus címke, akkor az elágazásnál a program ide ugrik.

A második példában: ha Z = 0, akkor a program nem vár az operandus értékére, hanem a soron következő utasítást hajtja végre. Egyébként az előbbiekkel azonosan működik.

Hogyan kerülnek az információk az állapotregiszterbe? Sok utasítás befolyásolja az állapotregiszter egyes helyeinek a tartalmát. Vannak utasítások, amelyekkel közvetlenül beállítható az állapotregiszter helyeinek a tartalma, más utasításokkal ez a tartalom törölhető. Ezt a későbbiekben részletesebben is leírjuk majd, de előbb nézzük az utolsó címzési módot.

Implicit címzés

A cím magából a parancsból látható. Lássunk ABC sorrendben néhány ilyen parancsot!

DEX : decrement X Reg by one
(csökkentsd X regiszter tartalmát eggyel).

DEY : decrement Y Reg by one
(csökkentsd Y regiszter tartalmát eggyel).

INX : increment X Reg by one
(növeld X regiszter tartalmát eggyel).

INY : increment Y Reg by one
(növeld Y regiszter tartalmát eggyel).

NOP : no operation
(nincs művelet, üres utasítás).

RTS : return from subroutine
(visszatérés az alprogramból).

TAX : transfer Accumulator to X Reg
(az akkumulátor tartalmát vidd X regiszterbe).

TAY : transfer Accumulator to Y Reg
(az akkumulátor tartalmát vidd Y regiszterbe).

TYA : transfer Y Reg to Accumulator
(Y regiszter tartalmát vidd az akkumulátorba).

TXA : transfer X Reg to Accumulator
(X regiszter tartalmát vidd az akkumulátorba).

Még néhány olyan utasítást ismertetünk, amelyekről részletesen az irodalomjegyzékben felsorolt könyvekben olvashatunk.

CMP operandus:
compare Accumulator and Memory
(az akkumulátor és a tár tartalmának összehasonlítása).

CPX operandus:
compare X Reg and Memory
(az X regiszter és a tár tartalmának összehasonlítása).

CPY operandus:
compare Y Reg and Memory
(az Y regiszter és a tár tartalmának összehasonlítása).

JMP operandus:
jump to new location
(ugorj egy új utasítás címére).

JSR operandus:
jump subroutine
(ugorj egy alprogramra).

LDA operandus:
load Accumulator
(tölts az akkumulátorba).

LDX operandus:
load X Reg
(tölts az X regiszterbe).

LDY operandus:
load Y Reg
(tölts az Y regiszterbe).

STA operandus:
store Accumulator
(tárold az akkumulátor tartalmát).

STX operandus:
store X Reg
(tárold az X regiszter tartalmát).

STY operandus:
store Y Reg
(tárold az Y regiszter tartalmát).

Ezzel a fejezettel azt szerettük volna bemutatni, hogy az assembler nyelvű program az utasításokat hogyan hajtja végre. Az assembler nyelven való programozás ismertetésével sok jó könyv részletesen foglalkozik. A fenti fejezet elolvasása után ezekben a könyvekben már sok minden ismerős lesz.

A következőkben az ASSEMBLER és a DISASSEMBLER programok utasításait tárgyaljuk és ezekről egy-egy teljes listát mellékelünk.

6.2 Az ASSEMBLER

Az ASSEMBLER-re akkor van szükségünk, ha egy assembler nyelven írt programot gépi kódra akarunk fordítani. Miből áll egy ASSEMBLER program?

Egyrészt olyan utasításokból, amelyeket assembler nyelvből gépi kódra kell fordítani, másrészt olyanokból, amelyek az ASSEMBLER-t a programmal kapcsolatos információkkal látják el. Ezeket nevezzük pszeudoutasításoknak, mivel a gépi kódú programban már nem jelennek meg. A DISASSEMBLER ezeket az utasításokat már nem tudja visszafordítani. Azok az utasítások, amelyek a gépi kódra való fordítást végzik, annak ellenére, hogy a programok nagyon leegyszerűsítettek és nem tartalmazzák az assembler nyelvű program minden információját, a DISASSEMBLER-rel mégis visszafordíthatók.

Az assembler nyelvű programokat ugyanúgy lehet írni és tárolni, mint a BASIC programokat.

A megjegyzéseket az assembler nyelvű programokban ; jellel kezdjük. Ha az ASSEMBLER egy ; jelet talál, akkor tudja, hogy a sor hátralevő részét figyelmen kívül hagyhatja. A megjegyzések egy soron belül bárhol kezdődhetnek.

```
10;          Ez egy megjegyzés, ami
20;          a sor végéig tart.
30 LDA 12;   töltsd az akkumulátorba
40;          a 12-es tárrekesz tartalmát.
```

Az üres helyek elválasztójelek, az ASSEMBLER programok legkisebb építőelemeit választják el egymástól. Egy utasítás a sor végéig tart. Minden sorba csak egy ASSEMBLER utasítást szabad írni.

Operandusok

Az operandusok 10-es vagy 16-os számrendszerbeli számok és betűvel kezdődő, tetszőleges hosszúságú karaktersorozatok lehetnek.

10-es számrendszerben	15
	1000
16-os számrendszerben	$FFFF
	$0D
	$1234
karaktersorozatok	otto
	címke 1
	mintapélda

Az akkumulátorokra vonatkozó léptető utasítások:

```
ASL ACCU
LSR ACCU
ROL ACCU
ROR ACCU
```

Az implicit címzési utasításokat, mint pl. a BRK-t a megszokott módon kell felírni.

Direkt címzés

Az utasítás felépítése: mnemonikkal kezdődik, utána egy üres hely, ezután #, majd tetszés szerinti üres hely következik és, végül az operandus.

```
LDA # otto
AND # otto
ADC # 13
ADC # 13
CMP # $12FF
```

Nullás lapú és abszolút címzés index nélkül

Az utasítás felépítése: mnemonik, legalább 1 üres hely, operandus. Az operandus nagyságától függően nullás lapú vagy abszolút címzés megy végbe. Ha az operandus egy név, amelyet az assembler nyelvű programban csak később határozunk meg, akkor mindig abszolút címzés megy végbe, mivel az ASSEMBLER a forrásszöveget (az assembler nyelvű programot) csak egyszer olvassa el, hogy időt takarítson meg.

```
ORA otto
STA 234
LDA $FE
STX 12345
```

Nullás lapú és abszolút címzés indexszel

Az utasítás felépítése: mnemonik, üres hely, operandus, vessző, végül X vagy Y, aszerint, hogy melyik regisztert használjuk.

```
STX otto, Y
STY otto, X
STA $44, X
LDA 123, X
```

Indexelt indirekt címzés

Az utasítás felépítése: mnemonik, tetszőleges számú üres hely (1 kötelező), kerek nyitó zárójel, tetszőleges számú üres hely, operandus, tetszőleges számú üres hely, vessző, tetszőleges számú üres hely, egy X, végül kerek záró zárójel.

```
LDA (otto, X)
STA ($AA, X)
```

Indirekt indexelt címzés

Az utasítás felépítése: mnemonik, legalább egy üres hely, kerek nyitó zárójel, tetszőleges számú üres hely, operandus, tetszőleges számú üres hely, kerek záró zárójel, sok üres hely, vessző, tetszőleges számú üres hely, végül egy Y.

```
LDA (otto) , Y
STA (123) , Y
```

Indirekt abszolút címzés

Erre a címzési módra csak a JMP utasításnál van lehetőség.

```
JMP ( 12345 )
```

Relatív címzés

Ez a címzési mód csak a feltételes ugró utasításoknál használatos.

Az utasítás felépítése: mnemonik, legalább egy üres hely, operandus.

Az operandus ebben az esetben egy címke. Erről még később lesz szó.

```
BCC címke-1
BPL kimenet
```

A pszeudoutasítások az ASSEMBLER-t vezérlik, ezért csak közvetve hatnak a gépi kódú programra. Az utasítások mindig egy ponttal kezdődnek. A legtöbb pszeudoutasításnak is van mnemonikja.

Ha egy MINIATUR fordítóval készített assembler nyelvű programot vizsgálunk, sok probléma magától érthetővé válik, és egy olyan példagyűjteménnyel rendelkezünk, amit bármikor tovább bővíthetünk. Ha már kellő gyakorlatunk van, akkor a lehető legjobb assembler nyelvű programot tudjuk készíteni.

START utasítás

A START utasítás határozza meg azt a tártartományt, ahova az ASSEMBLER program kerül. Az utána következő operandus jelöli ki a kezdőcímet.

```
.START 2047
.END
```

.END utasítás

A .END utasítás közli az ASSEMBLER-rel, hogy a programnak itt vége van.

.MARKE vagy .M utasítás

Ezzel az utasítással definiálunk. A címkékhez az a tárcím rendelődik, ahol a következő gépi utasítás képződik. A címkét arra is használhatjuk, hogy az akkumulátort a fenti tárhely tartalmával feltöltsük.

```
Címke 1 .MARKE
Címke 1 .M
```

.EQU vagy .E utasítás

Ezzel az utasítással lehet a változóknak értéket adni. Az ASSEMBLER futása után a változó helyén mindig a változó értéke szerepel. A .EQU utasítással egy változónak csak egyszer adhatunk értéket.

```
GÁBOR  .EQU $FEFE
BALÁZS .EQU 123
ANNA   .EQU ÉVA
ANNA   .E   ÉVA
```

.VAREQU vagy .V utasítás

Ez az utasítás egy változó értékének módosítására szolgál.

```
ANNA .VAREQU KATI
ANNA .V      ILDI
```

.BLOCK vagy .BL utasítás

Ezzel az utasítással adatok számára foglalunk helyet az assembler nyelvű programban. Az utasítás mögött álló operandus a lefoglalandó tárhelyek számát adja meg. Ezek az utasítás után következő helyeken lesznek.

```
.BLOCK  555
.BL     PALI
```

.TEXT vagy .T utasítás

Ezzel az utasítással tároljuk a szöveget attól a helytől kezdődően, ahol az utasítás áll. A karaktersorozat elejét és végét idézőjellel kell ellátni. Az első idézőjelet nem tároljuk, de az utolsót igen. Egy nulla értékű karakter is kerül a karaktersorozat végére. Ezt az utasítást általában az adott szöveg későbbi kiírásához használjuk. Tudnunk kell a címet is, ahol a szöveg áll, ezt a címet átadjuk egy ROM (Read Only Memory; magyarul: csak olvasásra használható tár) rutinnak és ideugrunk. Ezután megkezdődik a szöveg kiírása. Nézzük ehhez a .COUNT utasításnál lévő példát is!

```
.TEXT „halló itt vagyok!"
.T    „ez nagyszerű!"
```

.BYTE vagy .B utasítás

Az utasítás lefoglalja a következő tárhelyet és belehelyezi az operandus értékét. Az operandus értéke ennek megfelelően 0-tól 255-ig terjedhet.

```
.BYTE 66
.B    FERI
```

.DBYTE vagy .DB utasítás

Egy 16 bit információt tartalmazó operandus értékét két 8 bites információra alakítja. Ennek a kétszer 8 bites információnak lefoglalja a következő két tárhelyet és belehelyezi az információt. A behelyezés sorrendje: felső byte (magas helyiértékű), alsó byte (alacsony helyiértékű).

```
.DBYTE 256
.DB    254
```

Az első utasítás hatására 1 és 255, a másodikéra 0 és 254 kerül a tárba.

.WORD vagy .W utasítás

Ez az utasítás a .DBYTE utasításhoz hasonló, de itt először az alsó, majd a felső byte tartalma tárolódik.

.COUNT vagy .C utasítás

Az utasítás hatása a következő: amikor az ASSEMBLER elindul, akkor a CL és a CH szimbólumok egy táblázatba kerülnek. Ezek a szimbólumok értéket kapnak. A következő információ címe két 8 bites adattá alakul át. A CL egyenlő lesz az alsó, a CH pedig a felső byte tartalmával. A későbbi utasításokban előforduló CH és CL változók helyébe azok értéke kerül. Az értékeket a .COUNT utasítás aktualizálja.

Írjuk ki: Anna egy aranyos kislány

```
JMP TEXT-1                      ;ugorjuk át a mondatot.
.COUNT
.TEXT „Anna egy aranyos kislány"
TEXT-1 .MARKE                   ;Az ugrás címkéje.
LDY  #CL                        ;Az ugrás mutatóját töltsük
LDA  #CH                        ;a ROM rutinba.
JSR  $BA1E                      ;Ugrás a ROM-ra.
LDA  #13                        ;Soreltolás, karakterbetöltés.
JSR  $FFD2                      ;Ugrás az operációs
                                rendszer kiíró rutinjára.
```

Remélem kedvet kaptak az assembler nyelvű programozáshoz?!

P.5

az assembler listaja

```
45000 open15,8,15 : Print#15,"i" : Printchr$(14)
45010 Print" *******  Kerem helyezze be  ******"
45020 Print" *******    az adatlemezt    ******"
45030 Print" *******     <RETURN>        ******"
45040 Get f$:if f$<>chr$(13) then 45040
45050 Print#15,"i"
45070 inPut#15,en,em$,et,es
45100 if en<>0 then 45010
45110 close15
47000 goto 49999
48000 rem
48010 Print" *******     <RETURN>        ******"
```

```
48020 get w$:if w$<>chr$(13) then 48020
48030 sys 2076
49997 print chr$(14)
49998 :
49999 rem ********  assembler
50000 :
50001 open 15,8,15 : print#15,"i"
50002 dim f$(150),l$(500),l(500),n$(200),n(200),s$(400)
,s(400)
50003 l=2 : f=0 : pc=2129 : pm=pc : n=0 : s=0 : l$(0)="
ch" : l$(1)="cl" : po=0
50004 ti$="000000"
50005 print" Az assembler program neve : "
50006 input"    ";na$
50007 for t=2 to len(na$) : if mid$(na$,t,1)<>" " then
next
50008 na$=left$(na$,t-1) : print""
50010 gosub 60000
50011 open 8,8,8,"0:"+na$+",p,r" : gosub 50800 : if en<
>0 then 48000
50012 get#8,g$,h$ : if st<>0 then 48000
50014 g=asc(g$) : h=asc(h$)
50016 ad=g+256*h : a1=ad-2 : goto 50500
50020 gosub 50400
50030 if p=0  then 50500
50050 if p=59 then print"megjegyzes";: goto 50500
50060 if p=32 then 50700
50065 if p=34 then 50420
50240 if p>127 and p<204 then 51000
50250 if p<128 and p>32 then b$=b$+chr$(p)
50260 goto 50020
50400 g=h:p=g:a1=a1+1:get#8,h$:ifst<>0thenprint:print".
end hianyzik?":goto62300
50405 if h$="" then h=0: goto 50415
50410 h=asc(h$)
50415 return
50420 b$=b$+chr$(p)
50430 gosub 50400 :if p=34 then b$=b$+chr$(p) : goto 50
020
50440 if p=0  then 50030
50450 goto 50420
50499 :
50500 rem ***** uj sor kezdodik
50501 :
50503 if b$<>"" then gosub 51200
50504 if t0$="1" then f=f+1: f$(f)=zn$+" pszeudoutasita
st varom ": printf$(f)
50505 t0$="n" : f$="n" : ex=0
50507 if a1=ad then 50512
```

```
50510 a3=a1 : for a2=a3 to ad-1 : gosub 50400 : next a2
50512 ad=g+256*h
50515 gosub 50400 : gosub 50400
50527 zn=g+256*h : zn$="sor : "+str$(zn)
50530 print : print"sor : ";zn;
50535 gosub 50400
50540 goto 50020
50541 :
50700 rem ***** blank-et talalt
50701 :
50710 if b$="" then 50020
50712 if left$(b$,1)=chr$(34) then b$=b$+chr$(p) : goto
 50020
50715 if b$="#" or b$="(" then 50020
50717 if h=32 then 50020
50718 if ex=1 then 50020
50719 if t0$<>"n" and h>169 and h<175 then gosub 50400
: ex=1 : goto 50070
50720 gosub 51200
50730 goto 50020
50731 :
50800 rem ****** hibacsatorna
50801 :
50810 input#15,en,em$,et,es : if en=0 then return
50820 print" ***** Hiba a lmezen !  *****" : close 8
50830 print" ***** hiba szama      : ";en
50840 print" ***** hibauzenet      : ": print" *****  "
em$
50850 print" ***** hibas sav       : ";et
50860 print" ***** hibas szektor   : ";es
50870 goto 62300
51000 rem **** interpreter code atvaltasa
51001 :
51010 h$=""
51012 if p<140 then p9=41116 : p6=p-127 : goto 51020
51014 if p<160 then p9=41160 : p6=p-139 : goto 51020
51016 if p<180 then p9=41244 : p6=p-159 : goto 51020
51018 p9=49483-8192 : p6=p-179
51020 for t1=1 to p6
51030 p9=p9+1 : if peek(p9)<128 then 51030
51040 next t1
51050 p9=p9+1 : if peek(p9)<128 then h$=h$+chr$(peek(p9
)) : goto 51050
51060 h$=h$+chr$(peek(p9)-128)
51090 b$=b$+h$
51095 goto 50020
51096 :
51200 rem ******  tokent talalt / kiertekel
```

```
51201 :
51203 Print b$+" ";
51205 mn$="n"
51207 if t0$="i" then 9osub 55400 : 9oto 51290
51208 if b$<>".vare9u" and f$="d" then 9osub 54580
51210 if left$(b$,1)="." then 9osub 51400 : 9oto 51290
51220 if t0$="p" then 9osub 54000 : 9oto 51290
51222 if t0$="r" then 9osub 56500 : 9oto 51290
51225 if t0$="m" then 9osub 54800 : 9oto 51290
51230 if len(b$)=3 and t0$="n" then 9osub 53000
51240 if mn$="J" then 51290
51270 if t0$="1" then f=f+1: f$(f)=zn$+" pszeudoutasita
st varom ": Printf$(f)
51280 if t0$="n" then la$=b$ : t0$="1"
51290 b$=""
51300 return
51310 :
51400 rem ******  pszeudo-oPeratort talalt
51401 :
51405 ps=0
51410 if b$=".end" then ps=1
51420 if b$=".e9u" or b$=".e" then ps=2
51430 if b$=".start" then ps=3
51440 if b$=".block" or b$=".bl" then ps=4
51450 if b$=".byte" or b$=".b" then ps=5
51460 if b$=".dbyte" or b$=".db" then ps=6
51470 if b$=".text" or b$=".t" then ps=7
51480 if b$=".word" or b$=".w" then ps=8
51490 if b$=".vare9u" or b$=".v" then ps=9
51495 if b$=".marke" or b$=".m" then ps=10
51497 if b$=".count" or b$=".c" then ps=11
51500 onPs9osub51600,52000,52100,52200,52300,52400,5250
0,52600,52700,52800,52900
51510 if ps=0 then Print"******    nincs pszeudo ******"
51515 if ps<>0 then t0$="p"
51520 return
51521 :
51600 rem ******     .end
51601 : close 8 : 9osub 50800
51602 Print : Print"   ******       Pro9ramhossz          ****
*":Print
51603 if po=0 then Print"   ******          ";pc-pm+2129-2047

51604 if po=1 then Print"   ******          ";pc-pm
51605 Print"   ******                  byte            ******"
51606 9osub 61000
51610 Print"   ******                  Ve9e            ******"
51640 Print"   ******                  Hiba            ******"
```

```
51643 print"  ****                <RETURN>        ****"
51645 get w$:if w$<>chr$(13) then 51645
51650 for t=1 to f
51660 print f$(t)
51670 for g=1 to 1000 : next
51680 next
51681 print"  **** a gepikodu program":print"  **** new
e : "+na$+"-cel"
51685 gosub 62300
51691 end
51781 :
52000 rem ******     .equ
52001 :
52010 ps$="e"
52020 return
52021 :
52100 rem ******     .start
52101 :
52110 ps$="s"
52120 return
52121 :
52200 rem ******     .block
52201 :
52210 ps$="k"
52220 if la$<>"" then b$=la$ : gosub 54500
52221 :
52230 if f$="d" then gosub 54580
52240 return
52300 rem ******     .byte
52301 :
52310 ps$="b"
52320 if la$<>"" then b$=la$ : gosub 54500
52321 :
52330 if f$="d" then gosub 54580
52340 return
52341 :
52400 rem ******     .dbyte
52401 :
52410 ps$="d"
52420 if la$<>"" then b$=la$ : gosub 54500
52421 :
52430 if f$="d" then gosub 54580
52440 return
52441 :
52500 rem ******     .text
52501 :
52510 ps$="t"
52520 if la$<>"" then b$=la$ : gosub 54500
```

```
52521 :
52530 if f$="d" then gosub 54580
52540 return
52541 :
52600 rem ******      .word
52601 :
52610 ps$="w"
52620 if la$<>"" then b$=la$ : gosub 54500
52621 :
52630 if f$="d" then gosub 54580
52631 :
52640 return
52700 rem ******      .varegu
52701 :
52710 ps$="v" : f$="n"
52720 return
52721 :
52800 rem ******      .marke
52801 :
52810 t0$="i" : b$=la$ : gosub 54500
52820 if f$="d" then gosub 54580
52830 return
52831 :
52900 rem ******      .count
52901 :
52910 t0$="i"
52920 l(0)=int(pc/256) : l(1)=pc-l(0)*256
52930 return
52931 :
53000 rem ******      mnemonic ?
53001 :
53005 w=0
53010 t=62
53020 gosub 53700
53030 te$=left$(t$,3)
53031 :
53035 rem print te$,b$
53040 if b$=te$ then w=val(mid$(t$,6,1)) : ts=t : goto
53570
53050 if b$<te$ then t=asc(mid$(t$,4,1)) : goto 53210
53060 t=asc(mid$(t$,5,1))
53210 if t=91 then w=0 : goto 53570
53220 goto 53020
53570 if w=0 then mn$="n" : return
53580 if w=1 then gosub 55500 : return
53590 if w=2 then gosub 55700 : return
53600 gosub 56000 : return
```

```
53601 :
53700 rem *** t$ toltese t cimre
53701 :
53705 on int(t/10)-2 goto 53710,53720,53730,53740,53750
,53760,53770
53710 on t-34 gosub 53801,53802,53803,53804,53805: retu
rn
53720 on t-399osub53806,53807,53808,5380953810,53811,53
812,53813,53814,53815
53721 return
53730 on t-499osub53816,53817,53818,53819,53820,53821,5
3822,53823,53824,53825
53731 return
53740 on t-599osub53826,53827,53828,53829,53830,53831,5
3832,53833,53834,53835
53741 return
53750 on t-699osub53836,53837,53838,53839,53840,53841,5
3842,53843,53844,53845
53760 on t-799osub53846,53847,53848,53849,53850,53851,5
3852,53853,53854,53855
53761 return
53770 gosub 53856 : return
53771 :
53800 -
53801 t$="adc[[369[[[[1265#7r23275#8k3626d[[[[237d[[[[8
379[[#59361[[[[4271[[[[52":return
53802 t$="and#%329[[[[1225[[J23235$8J3622d[[J4233d[[[[8
339[[$59321?1[[4231[[[[52":return
53803 t$="asl[[30a[[[[:106[[[[3216[[[[620e[[[[231e,1*18
3":return
53804 t$="bcc$^290y1w1;2":return
53805 t$="bcs[(2b0u1@6;2":return
53806 t$="beq[[2f0d1p1;2":return
53807 t$="bit&-324$7$2322cJ1$423":return
53808 t$="bmi[[230i1o1;2":return
53809 t$="bne*,2d0$111;2":return
53810 t$="bPl[[2109101;2":return
53811 t$="brk+.100[[[[01":return
53812 t$="bvc[/250f121;2":return
53813 t$="bvs[[270h191;2":return
53814 t$="clc)8118e3e601":return
53815 t$="cld[[1d8434501":return
53816 t$="cli13158:3:501":return
53817 t$="clv[[1b8[[[[01":return
53818 t$="cmP263c9[[[[12c547=132d5487262cd637323dd46748
3d9[[[[93c1616242d1[[[[52":return
53819 t$="cpx[[3e0+1(112e4n7n232ec[[[[23":return
53820 t$="cpy573c0[[[[12c4[[[[32cc[[[[23":return
```

```
53821 t$="dec[[3c6[[[[32d6[[[[62ce[[[[23de[[[[83":retur
n
53822 t$="dex4;1ca424401":return
53823 t$="dey[:188[[[[01":return
53824 t$="eor[[349[[[[1245:7c23255:8c3624d[[c4235d:6c58
359[[[[93411[[4251[[[[52":return
53825 t$="inc9<3e652<132f6[[[[62ee[[[[23fe[[[[83":retur
n
53826 t$="inx[=1e8[[n101":return
53827 t$="iny[[1c8714101":return
53828 t$="jmP0m34cc1:4236ck1k4<3":return
53829 t$="jsr[[320[[[[23":return
53830 t$="lda?a3a9[[[[12a5a1v132b5b3a362adb4a423bd[[[[8
3b9@3b593a1[[[[42b1[[[[52":return
53831 t$="ldx[[3a2@7b212a6[[[[32b631[[72ae[[[[23beb1519
3":return
53832 t$="ldy@d3a0&1'112a4[[[[32b4@8[[62ac[[[[23bcx1@58
3":return
53833 t$="lsr[[34a:1[[:146[[[[3256[[[[624e[[[[235e[[[[8
3":return
53834 t$="noP:e1ea;1n401":return
53835 t$="ora[[309[[%11205e7%23215e8%3620de1%4231d[[[[8
319[[e59301-1[[4211[[[[52":return
53836 t$="Phabi148:2>101":return
53837 t$="PhP[[108e2e401":return
53838 t$="Pla9[168#2>201":return
53839 t$="PlPhk128)1)201":return
53840 t$="rol[[32a$1[[:126[[[[323f[[[[622e[[[[233e%5m18
3":return
53841 t$="ror;136a#1[[:166[[[[3276[[[[626e#4[[237e;5a58
3":return
53842 t$="rti[[140[[[[01":return
53843 t$="rtsfs160.1/101":return
53844 t$="sbc[[8e9[[[[12e5[[[[32f5n8;262ed53;323fdn6;48
3f9[[[[93e1[[[[42f1[[[[52":return
53845 t$="secn[138$3$601":return
53846 t$="sedor1f8n3n501":return
53847 t$="sei[[178#3#601":return
53848 t$="sta9[385t1s13295t2s2628dt3s3239d[[[[8399[[[[9
381[[[[4291[[[[52":return
53849 t$="stxPw38691[[3296[[[[728e[[[[23":return
53850 t$="sty[[384r6[[3294r7[[628c[[[[23":return
53851 t$="taxtv1aa@2@401":return
53852 t$="tay[y1a8a2@101":return
53853 t$="tyau[198r2z101":return
53854 t$="tsx[[1ba[[[[01":return
53855 t$="txaxz18ar1r301":return
53856 t$="txs[[19ar5r401":return
```

```
54000 rem ***** Pszeudo-oPerator oPerandus
54010 9osub 55200
54015 if Ps$="t" then b$=b$+chr$(0)
54020 if Ps$="s" then Pc=b : Pm=Pc : Po=1
54030 if Ps$="e" then 9oto 54350
54040 if Ps$="k" then Pc=Pc+b : for t=1 to b : c0=0 : 9
osub 60100 : next t
54050 if Ps$="b" and b<256 then c0=b : 9osub 60100 : Pc
=Pc+1
54060 if Ps$="b" and b>=256 then 9osub 55400
54070 if Ps$="d" then 9osub 57620 : c0=b2 : 9osub 60100
 : c0=b1 : 9osub 60100 : Pc=Pc+2
54080 if Ps$="w" then 9osub 57620 : 9osub 57060
54090 if Ps$="t" then fort=2tolen(b$)-1:c0=asc(mid$(b$,
t,1)):9osub60100:Pc=Pc+1:next t
54100 if Ps$="v" then l(v9)=b
54340 t0$="i" : return
54350 b$=la$ : 9osub 54500
54360 if f$="d" then 9osub 54580 : 9oto 54340
54370 l(l-1)=b : 9oto 54340
54500 rem **** cimket talalt & beolvas
54501 :
54510 for t=0 to l
54520 if l$(t)<>b$ then next
54540 if t<>l+1 then f$="d" : v9=1 : return
54550 l$(l)=b$ : l(l)=Pc : v9=l : l=l+1 : la$=""
54560 return
54570 :
54580 f=f+1 : f$(f)=zn$+"  "+b$+"  tobbszorosen defini
lt." : Printf$(f) : return
54800 rem ***** mnemonichoz cimket keres   valtozok cim
ei
54801 :
54805 y=len(b$)
54810 if b$="accu" then oP%=1 : ce%=10 : by=1 : 9osub 5
7000 : return
54820 if left$(b$,1)="#" then 58000
54830 if left$(b$,1)="(" and right$(b$,3)=",x)" then 58
400
54840 if left$(b$,1)="(" and right$(b$,1)=")" then 5820
0
54850 if left$(b$,1)="(" and right$(b$,3)="),y" then 58
600
54860 if right$(b$,2)=",x" then 58800
54870 if right$(b$,2)=",y" then 59000
54880 9oto 59200
54881 :
55200 rem ***** oPerandus   b$ - b dec
```

```
55201 :
55205 if left$(b$,1)=chr$(34)then b=0 : return
55210 if left$(b$,1)="$" then gosub57800 : return
55220 if asc(left$(b$,1))>47 and asc(left$(b$,1))<58 th
en gosub 57900 : return
55230 for t=0 to l : if l$(t)<>b$ then next
55240 if t=l+1 then gosub 62400
55245 if t=l+1 then print chr$(13);b$;" meg nem definia
lt."
55250 ift=l+1and(ce%=1orce%=4orce%=5) then b=255 : s$(s
)=b$ : s(s)=pc : s=s+1:return
55254 rem    print"55254 b=256*256"
55255 if t=l+1 then b=256*256-1 : s$(s)=b$ : s(s)=pc :
s=s+1 :return
55257 if k7=1 then k7=0 : return
55260 b=l(t) : return
55261 :
55300 rem ****** dec b$ - b ------>  hex
55301 :
55310 he$=mid$(b$,2,len(b$)-1)
55320 gosub 59700
55330 b=de : return
55331 :
55400 rem ****** nem lehet operandus - hiba
55401 :
55410 f=f+1
55420 f$(f)=zn$+"  "+b$+"  mint mint operandus nem lehe
tseges.": printf$(f)
55430 return
55431 :
55500 rem ****** egy byte-os parancs
55501 :
55510 me$=b$ : op%=1 : ce%=0
55520 gosub 55600
55530 c0=co : gosub 60100
55540 pc=pc+1
55550 t0$="i" : mn$="j"
55560 return
55561 :
55600 rem ******  co gepikod beallitasa
55601 :
55610 t1=len(t$)-6
55620 for t=0 to t1/8-1
55630 if ce%=asc(mid$(t$,13+t*8,1))-48 then 55685
55640 next t
55660 f=f+1
55670 f$(f)=zn$+"  cimzes nem megengedett.": printf$(f)
55680 co=234 : return
```

```
55681 :
55685 if oP%<>val(mid$(t$,14+t*8,1)) then 55660
55690 he$=mid$(t$,7+t*8,2) : gosub 59700
55695 co=de : return
55699 :
55700 rem ***** relativ cimzesu Parancs
55701 :
55710 me$=b$ : oP%=2 : ce%=11
55720 gosub 55600
55730 c0=co : gosub 60100
55740 Pc=Pc+1
55750 t0$="r" : mn$="j"
55760 return
55761 :
56000 rem ***** valtozo cimzesu Parancs
56001 :
56010 me$=b$ : t0$="m" : mn$="j" : return
56011 :
56500 rem *****  oPerandus relativ cimzeshez
56501 :
56510 if left$(b$,1)<>"$" then 56600
56520 gosub57800 : gosub 55300
56530 if b>255 then 55670
56540 c0=b : gosub 60100
56550 Pc=Pc+1
56560 t0$="i"
56570 return
56571 :
56600 if asc(left$(b$,1))>47 and asc(left$(b$,1))<58 th
en gosub 57900 : goto 56530
56601 :
56610 for t=0 to l : if l$(t)<>b$ then next
56620 if t=l+1 then n$(n)=b$ : n(n)=Pc : n=n+1 : b=255
: goto 56530
56630 b=255-Pc+1(t) : goto 56530
56631 :
57000 rem ****  co-t keres & Poke-ol
57001 :
57010 if f$="j" then t0$="i" : return
57020 gosub 55600
57030 c0=co : gosub 60100 : Pc=Pc+1
57040 if by=1 then t0$="i" : return
57050 if by=2 then c0=b1 : gosub 60100 : Pc=Pc+1 : t0$=
"i" : return
57060 c0=b1:gosub 60100 : c0=b2 : gosub 60100 : Pc=Pc+2
 : t0$="i" : return
57061 :
57590 rem *****  oPerandus test es b1/b2 osztas
```

```
57591 :
57600 if b<0 then 57650
57610 if b<256 then by=2 : b1=b : b2=0 : return
57620 if b>65536 then 57650
57630 by=3 : b2=int(b/256) : b1=b-b2*256 : return
57650 f=f+1 : f$(f)=zn$+"  "+str(b)+"  nem ervenyes.":
printf$(f) : f$="j" : return
57699 :
57700 rem ***** hexaszam ?
57701 :
57800 for t=2 to len(b$) : te=asc(mid$(b$,t,1))
57810 if te>47 and te<58 or te>64 and te<71 then next :
 gosub 55300 : return
57830 f=f+1 : f$(f)=zn$+"  "+b$+"  nem hexaszam.": prin
tf$(f) : f$="j"
57840 b=0 : return
57889 :
57890 rem *****  decimalis szam ?
57900 for t=1 to len(b$)-1 : te=asc(mid$(b$,t,1))
57910 if te>47 and te<58 then next : b=val(b$) : return

57930 f=f+1 : f$(f)=zn$+"  "+b$+"  nem decimalis szam."
: printf$(f) : f$="j"
57940 b=0 : return
57979 :
57980 rem *****  az operandus elvalasztasa a mnemonic-t
ol
57981 :
58000 b$=mid$(b$,2,y-1) : ce%=1 : gosub 59400 : op%=by
: goto 57000
58200 b$=mid$(b$,2,y-2) : gosub 59400 : op%=3 : ce%=12
: goto 57000
58400 b$=mid$(b$,2,y-4) : ce%=4 : gosub 59400 : op%=by
: goto 57000
58600 b$=mid$(b$,2,y-4) : ce%=5 : gosub 59400 : op%=by
: goto 57000
58800 b$=mid$(b$,1,y-2) : gosub 59400 : op%=by : ce%=6
58810 if by=3 then ce%=8
58820 goto 57000
59000 b$=mid$(b$,1,y-2) : ce%=7 : gosub 59400 : op%=by
59010 if by=3 then ce%=9
59020 goto 57000
59200 b$=mid$(b$,1,y) :ce%=3 : gosub 59400 : op%=by
59210 if by=3 then ce%=2
59220 goto 57000
59299 :
59300 rem *****  b$ ----> b
59301 :
```

```
59400 gosub 55200 : goto 57600
59599 :
59700 rem ***** hex ----> dec
59701 :
59710 de=0
59720 for t=len(he$) to 1 step -1
59730 h2$=mid$(he$,t,1)
59740 for t1=1 to 16
59750 if mid$("0123456789abcdef",t1,1)<>h2$ then next t
1
59760 de=de+(t1-1)*16^(len(he$)-t)
59770 next t : return
59780 :
60000 rem *****  csatorna megnyitas - adatok
60010 print#15,"s:"+na$+"ko"
60020 input#15,en,em$,et,es : rem print "  **";left$(em
$,13);"**"
60030 open 9,8,9,"0:"+na$+"ko"+",p,w" : gosub 50800 : i
f en<>0 then stop
60035 print#9,chr$(0)chr$(0);
60040 return
60100 rem ***** kiiras a lemezre
60110 co%=co%+1
60120 if co%<254 then bl$=bl$+chr$(c0) : return
60130 print#9,bl$;
60135 rem      ?"bl$=";bl$
60140 bl$=chr$(c0) : co%=1
60150 return
60151 :
60200 rem *****  az utolso blokk atadasa
60205 rem    ?"bl$=";bl$,len(bl$)
60206 :
60210 print#9,bl$; : return
60211 :
61000 rem *****  pass 11
61001 :
61005 if st<>64 then 50800
61010 gosub 60200
61020 close 9 : gosub 50800 : if en<>0 then 48000
61021 :
61022 rem      open 9,8,9,na$+"ko,p,r"
61024 rem      get#9,b$ : if st<>0 then close 9 : goto 6
1030
61026 rem      if b$="" then print0; : goto 61024
61028 rem      print asc(b$); : goto 61024
61030 rem      print asc(b$) : close 9 : goto 50800 : if
 en<>0 then stop
61040 print#15,"s:"+na$+"-cel"
```

```
61050 input#15,en,em$,et,es : rem print "  **";left$(em
$,13);"**"
61060 open 9,8,9,"0:"+na$+"-cel"+",p,w" : gosub 50800 :
 if en<>0 then 48000
61062 p7=pc : pc=pm : t2=int(pm/256) : t3=pm-t2*256
61063 if po=1 then print#9,chr$(t3)chr$(t2);
61064 if po=0 then print#9,chr$(1 )chr$(8 );
61065 gosub 62100
61066 rem    print"st="!st
61067 :
61070 open 8,8,8,"0:"+na$+"ko"+",p,r"
61072 get#8,b$
61073 get#8,b$
61075 rem    print"st=";st
61090 s(s)=65000 : n(n)=65000 : t2=0 : t3=0
61100 if s(t2)=65000 and n(t3)=65000 then 61600
61110 if s(t2)=65000 then t4=n(t3) : t5$=n$(t3) : t3=t3
+1 : t6$="r" : goto 61150
61140 if s(t2)<n(t3) then t4=s(t2)+1 : t5$=s$(t2) : t2=
t2+1 : t6$="a" : goto 61150
61145 t4=n(t3) : t5$=n$(t3) : t3=t3+1 : t6$="r"
61150 for t=0 to l
61160 if l$(t)<>t5$ then next
61170 if t=l+1 then f=f+1:f$(f)=b$+" nem definialt.": g
oto 61100
61180 if t6$="r" then 61500
61190 b=l(t) : gosub 57630
61200 get#8,g$ : if g$="" then g$=chr$(0)
61205 if pc<>t4 then print#9,g$;: pc=pc+1 : goto 61200
61210 get#8,h$ : if h$="" then h$=chr$(0)
61220 ifasc(g$)=255andasc(h$)=255thenprint#9,chr$(b1)ch
r$(b2);:pc=pc+2:goto 61100
61230 ifasc(g$)=255andb<256thenprint#9,chr$(b);h$;:pc=p
c+2:goto 61100
61240 f=f+1:f$(f)=str$(pc)+"  cimke nem megengedett v.
meg nem definialt"
61250 goto 61100
61500 b=l(t)-n(t3-1)-1
61510 if pc<>t4 then get#8,g$: if g$="" then g$=chr$(0)

61515 if pc<>t4 then print#9,g$;: pc=pc+1 : goto 61510
61520 if b>127 then f=f+1:f$(f)="Nem lehetseges "+str$(
pc)+" relativ ugras."
61540 print#9,chr$(b);:pc=pc+1 : get#8,g$ : goto 61100
61600 get#8,g$ : if g$="" then g$=chr$(0)
61601 if st=0 then print#9,g$;:goto 61600
61603 print#9,g$;
61605 print#9,chr$(0)chr$(0)chr$(0);
```

```
61610 close 9 : gosub 50800 : if en<>0 then 48000
61620 close 8 : gosub 50800 : if en<>0 then 48000
61630 Print#15,"s:"+na$+"ko"
61640 inPut#15,en,em$,et,es : rem Print em$
61645 close 15
61650 rem      oPen 8,8,8,na$+"-cel,P,r"
61660 rem      Get#8,b$ : if st<>0 then 62000
61670 rem      if b$="" then Print0;: goto 61660
61680 rem      Print asc(b$);: goto 61660
62000 rem      if b$="" then Print0;: close 8 : return
62005 rem      Print asc(b$);: close 8
62010 return
62100 rem *****   basic fej
62105 if Po=1 then return : rem a Program szamara
62110 Print#9,chr$(79)chr$(8)chr$(1)chr$(0)chr$(158)chr
$(32)chr$(50)chr$(49);
62120 Print#9,chr$(50)chr$(57)chr$(32)chr$(32)chr$(32)c
hr$(32)chr$(32);
62130 Print#9,chr$(86)chr$(79)chr$(76)chr$(75)chr$(69)c
hr$(82)chr$(32);
62140 Print#9,chr$(38)chr$(32)chr$(69)chr$(78)chr$(78)c
hr$(79)chr$(32);
62150 Print#9,chr$(32)chr$(32)chr$(32)chr$(32)chr$(32)c
hr$(32)chr$(32);
62160 Print#9,chr$(32)chr$(32)chr$(32)chr$(32)chr$(32)c
hr$(32)chr$(32);
62170 Print#9,chr$(32)chr$(32)chr$(32)chr$(32)chr$(32)c
hr$(32)chr$(32);
62180 Print#9,chr$(32)chr$(32)chr$(32)chr$(32)chr$(32)c
hr$(65)chr$(83);
62190 Print#9,chr$(83)chr$(69)chr$(77)chr$(66)chr$(76)c
hr$(69)chr$(82);
62200 Print#9,chr$(32)chr$(32)chr$(32)chr$(32)chr$(32)c
hr$(32)chr$(32);
62210 Print#9,chr$(32)chr$(32)chr$(49)chr$(57)chr$(56)c
hr$(52)chr$(0);
62220 Print#9,chr$(0)chr$(0);
62230 return
62300 rem *****  vege
62305 na$=" "+na$: na$=mid$(na$,2,len(na$)-1)
62330 Print"****     A Program befejezodott !     ****"
62340 Print"****          A celProgramot          ****"
62350 Print"****          a kovetkezo modon       ****"
62360 Print"****          tolthetJuk be           ****"
62365 Print"":Print"load"+chr$(34)+na$+"-cel"+chr$(34)+
",8,1"
62400 rem ***** cimket keres es a hb-be es az lb-be rak
ja.
```

```
62410 if left$(b$,3)="hb-" then 62440
62420 if left$(b$,3)="lb-" then 62440
62430 return
62440 k7$=mid$(b$,4,len(b$)-3) : for t=0 to l
62450 if l$(t)=k7$ then 62480
62460 next t
62470 return
62480 k7=1
62490 if left$(b$,3)="hb-" then b=int(l(t)/256) : retur
n
62500 b=l(t)-int(l(t)/256)*256 : return
```

6.3 A DISASSEMBLER

A DISASSEMBLER-re akkor van szükségünk, ha gépi kódú programot akarunk elemezni.

Az ASSEMBLER-rel gépi kódú programokat írhatunk, amelyeket önállóan vagy egy MINIATUR, ill. egy BASIC program részeként futtathatunk. A DISASSEMBLER-nek az a feladata, hogy a gépi kódú programot mnemonikus írásmódra visszafordítsa. A DISASSEMBLER BASIC program, ezért a tárban tetszőleges helyre tehető. Ha egy gépi kódú programot pl. a tár 2047-es rekeszétől a 10000-esig terjedő helyeire töltöttünk be, akkor a DISASSEMBLER-t a 10002-es helytől kezdődően tárolhatjuk. Gépeljük be közvetlen üzemmódban az alábbi sorokat:

```
POKE 44, INT (10002/256)
POKE 43, 10002–256*PEEK(44)
POKE 10001–1,0
```

Ezután a DISASSEMBLER-t a következő utasítással hívhatjuk be:

```
LOAD „DISASSEMBLER",8
```

Ha a gépi kódú programot a 2047-es és a 12000-es tárhelyeken kívül helyeztük el, akkor az utolsó sort közvetlenül is beadhatjuk. A DISASSEMBLER használata után ne felejtsük el a számítógépet eredeti állapotába visszaállítani. Ez a

```
POKE 43,1
POKE 44,8
```

utasításokkal lehetséges.

Ha meg akarjuk tudni, hogy a számítógépben lévő program melyik tárhelyig tart, akkor ezt a következő utasítással érhetjük el:

```
PRINT PEEK(45) + PEEK(46)*256
```

Töltsük be a DISASSEMBLER-t és indítsuk el a RUN paranccsal! Ekkor megjelenik a DISASSEMBLER parancsait tartalmazó menü. Nézzük egyenként a parancsokat!

M : MENUE

Az M betű megnyomásával az adott parancs befejezése után ismét megjelenik a menü. Így bármikor tájékozódhatunk az alkalmazható parancsokról.

F : FREIER PLATZ

Ez a parancs a tárban lévő szabad helyekről tájékoztat. Megadja azoknak a helyeknek a számát, amelyeknek címe nagyobb a DISASSEMBLER végének címénél.

Z : DE Z NACH HEX

Ezzel a paranccsal tízes számrendszerbeli számot tizenhatos számrendszerbelire alakíthatunk.

X : HEX NACH DE Z

Ez pedig tizenhatos számrendszerbeli számot alakít tízes számrendszerbeli számmá.

A : ADRESSEN SETZEN

Ezzel közölhetjük a DISASSEMBLER-rel, hogy melyik tártartományban kívánunk dolgozni.

H : ZEIGER WEITER

A program indításakor a mutató (pointer) azt a tárhelyet jelzi, amelyiket az előző paranccsal kezdetként kijelöltünk. A H billentyű lenyomásával a mutató eggyel feljebb megy, és kiírja a képernyőre a megfelelő tárrekesz tartalmát.

N : ZEIGER ZURUECK

Ez a parancs a mutatót eggyel visszaállítja és kiírja a tárrekesz tartalmát.

P : POKE

A P billentyű lenyomásával megváltoztathatjuk annak a rekesznek a tartalmát, amelyiket a mutató jelez. Az új érték beírása után nyomjuk le a RETURN billentyűt, ezzel a rekesz tartalma megváltozik, és a mutató eggyel feljebb kerül.

E : BYTE EINSETZEN

A parancs hatására a beszúrandó byte-okra vonatkozó kérdés jelenik meg. A megfelelő szám beírása után nyomjuk le a RETURN billentyűt. A kijelölt tártartományon belül a pillanatnyi mutatóállástól kezdődően a rekeszek tartalma a beszúrandó byte-ok számával eltolódik. A szabaddá váló tárhelyek 234 decimális értékkel töltődnek fel. Ez a mikroprocesszor NOP (NO OPERATION) parancsának felel meg.

L : BYTE LOESCHEN

Meg kell adnunk a törlendő byte-ok számát, erre a mutató állásától kezdődően a megadott számú byte törlődik. A kiválasztott tártartomány maradék része ennek megfelelően lejjebb tolódik.

Y : SYS(XXXXX)

Az Y billentyű lenyomásával a megadott tárhelyen kezdődő gépi kódú program elindul.

D : DISS & DRUCK

A D billentyű lenyomásával egy olyan programot nyomtathatunk ki, amelyet a DISASSEMBLER tizenhatos vagy tízes számrendszerbe lefordított. Ehhez előbb jeleznünk kell, hogy a kezdő- és a végcímet tizenhatos vagy tízes számrendszerben fogjuk-e megadni. Ha egy J-től eltérő billentyűt nyomunk le, akkor tízes számrendszerben kell a címet megadni. A műveletet a RETURN billentyű lenyomásával fejezzük be.

F5 : DISS & PRINT DEZ

Ennek a parancsnak a hatására a program tízes számrendszerbeli formában jelenik meg a képernyőn.

F7 : DISS & PRINT HEX

Hasonló az F5-ös parancshoz, de itt tizenhatos számrendszerbeli formában jelenik meg a program.

S : SAVE DISK

A parancs a tár tetszőleges területeinek a tartalmát lemezre viszi.

O : OLD DISK

Ezzel a paranccsal a lemezen lévő információkat tölthetjük a számítógép tárjába.

Az Olvasó türelmesen próbálja ki a DISASSEMBLER parancsait!

P.6

a disassembler listaja

```
9995 rem********** memoria terulet
9996 :
10032 :
30020 poke 650,128 : em=49151 : am=40960 : a1=0 : la=am

30026 :
30027 rem********* menu
30029 :
30040 print" e : byte beiras"
30045 print" l : byte torles"
30050 print" h : mutato tovabb"
30055 print" n : mutato vissza"
30056 print" z : dec --> hex"
30057 print" x : hex --> dec"
30060 print" y : sys("jam")"
30065 print" a : cim allitas"
30068 print" s : save disk"
30070 print" o : old disk"
30071 print" f : szabad hely      "
30072 print" p : poke"
30073 print" m : menu"
30074 print" d : dis & nyomtat"
30075 print" f5: dis & print dec"
30076 print" f7: dis & print hex"
30080 get w$ : if w$="" then 30080
30085 if w$="h" then la=la+1 : print la;peek(la) : goto
 30080
```

```
30090 if w$="n" then la=la-1 : print la;peek(la) : goto
 30080
30100 if w$="e" then gosub 31000 : goto 30040
30102 if w$="z" then gosub 32200
30104 if w$="x" then gosub 32400
30110 if w$="l" then gosub 32000 : goto 30040
30120 if w$=chr$(135) then gosub 44300
30130 if w$=chr$(136) then gosub 44300
30131 if w$="f" then print"szabad hely :";fre(0)
30132 if w$="s" then gosub 40000
30133 if w$="o" then gosub 41000
30135 if w$="a" then input"kezdocim :";an
30136 if w$="p" then print la;: input po : poke la,po :
 la=la+1
30137 if w$="y" then print"   sys(";an;")";: sys (an) :
 print" vege"
30138 if w$="m" then 30040
30139 if w$="d" then 44000
30140 if w$="a" then input"vegcim :";en : la=an
30141 :
30150 goto 30080
30151 :
30997 rem ********* byte beiras
30999 :
31000 input"Hany byte-ot akar beirni";by
31010 for t=en to la+by step-1: poke t,peek(t-by): next

31020 for t=la to la+by-1: poke t,234: next
31030 return
31032 :
31997 rem ********* byte torles
31999 :
32000 input"Hany byte-ot akar torolni";by
32010 for t=la to la+by : poke t,peek(t+by): next
32020 return
32022 :
32200 rem ********* dec --> hex
32202 :
32210 input "melyik decimalis szamot";de
32220 gosub 45000
32230 print"a hexa szam : ";he$
32240 return
32241 :
32400 rem ********* hex --> dec
32402 :
32410 input "melyik hexa szamot";he$
32420 gosub 32600
32430 print"a decimalis szam : ";de
```

```
32440 return
32441 :
32600 rem ****** hex --> dec konverzio
32602 :
32610 de=0
32620 for t=len(he$) to 1 step-1
32630 h2$=mid$(he$,t,1)
32640 for t1=1 to 16
32650 if mid$("0123456789abcdef",t1,1)<>h2$ then next t1
32660 de=de+(t1-1)*16↑(len(he$)-t)
32670 next t
32680 return
32681 :
39997 rem ****** tarolas lemezen
39998 :
40000 Print:inPut"    az adat neve";w$
40005 a1=0 : a2=0
40007 Print"  tarolas"
40010 inPut" mettol";a1
40020 inPut" meddig";a2
40025 ac=8 : rx=8 : ry=1 :gosub 42200 : sys 65466
40030 gosub 42500
40040 ac=len(w$) : rx=175 : ry=2 : gosub 42200 : sys 65
469
40050 ac=251 : rx=Peek(253) : ry=Peek(254) : : gosub 42
200 : sys 65496
40060 gosub 42900
40070 return
40071 :
40997 rem ***** toltes a lemezrol
40998 :
41000 inPut" az adat neve";w$
41005 a1=0 : a2=0
41007 Print"  toltes"
41010 inPut" mettol";a1: a2=0
41025 ac=8 : rx=8 : ry=1 :gosub 42200 : sys 65466
41030 gosub 42500
41040 ac=len(w$) : rx=175 : ry=2 : gosub 42200 : sys 65
469
41050 ac=0 : rx=251 : ry=252 : gosub 42200 : sys 65493
41060 gosub 42900
41070 return
41071 :
42200 rem ***** sys
42201 :
42210 Poke 780,ac
42220 Poke 781,rx
42230 Poke 782,ry
```

```
42280 return
42281 :
42500 rem ******** 680-regiszter allitasa
42580 rem **   251 :l.b. toltes/tarolas kezdocime
42590 rem **   252 :h.b. toltes/tarolas kezdocime
42600 rem **   253 :l.b. toltes/tarolas vegcim + 1
42610 rem **   254 :h.b. toltes/tarolas vegcim + 1
42640 rem **   687-700 :=nev
42641 :
42760 if a1=0 and a2=0 then a1=an : a2=en
42770 poke 251,a1-int(a1/256)*256
42780 poke 252,int(a1/256)
42790 poke 253,a2+1-int((a2+1)/256)*256
42800 poke 254,int((a2+1)/256)
42810 for t=687 to 686+len(w$)
42820 poke t,asc(mid$(w$,t-686,1))
42830 next
42840 return
42841 :
42900 rem ******** hibacsatorna olvasasa
42901 :
42910 open 15,8,15
42930 input#15,ef,em$,et,es
42940 if ef=0  then close 15: return
42950 print"** Hiba a lemezen! **"
42960 print"** hiba szama  : ";ef
42970 print"** hibauzenet : ";em$
42980 print"** hibas sav   : ";et
42990 print"** hibas szektor : ";es
42991 :
42992 :
43000 close 15: return
43001 :
44000 rem ******** dis & nyomtatas
44002 :
44010 print" lista kinyomtatasa"
44030 open 4,4
44033 input "hexa ? (i)";ef$
44035 if ef$<>"i" then 44050
44040 input" mettol";he$: gosub 32600: a1=de
44045 input" meddig";he$: gosub 32600: a2=de: goto 44065
44050 input" mettol";a1
44060 input" meddig";a2
44065 print" megszakitas <RETURN>-nel"
44080 if a1>a2 then close4 : goto 30027
44085 get ef$: if ef$=chr$(13) then close 4: goto 30027
```

```
44090 gosub 48500
44100 Print#4,Pr$
44110 a1=a1+oe
44120 goto 44080
44299 :
44300 rem ****** dis & Print hex/dec
44302 :
44320 a9=la : a1=a9
44330 gosub 48500
44335 if w$=chr$(135) then Pr$=mid$(Pr$,38,len(Pr$)-35)
 : goto 44350
44340 Pr$=left$(Pr$,35)
44350 Print Pr$
44360 get ef$: if ef$=chr$(13) then la=a1 : return
44370 a1=a1+oe : goto 44330
44371 :
45000 rem **** dec --> hex konverzio 4-jegyu
45001 :
45002 el=3
45005 he$=""
45010 for d=el to 0 step-1
45020 h1=16^d:h2=int(de/h1):he$=he$+mid$("0123456789abc
def",h2+1,1):de=de-h2*h1
45030 next
45040 return
45050 :
45060 rem **** dec --> hex konverzio 2-jegyu
45062 :
45065 el=1: gosub 45005 : return
45099 :
45100 rem *****     oe=0 / hiba
45102 :
45110 Pr$=Pr$+" *****"
45120 hi$=hi$+"  *******"
45130 a1=a1+1
45140 return
45199 :
45200 rem *****     oe=1 / 1 byte
45201 :
45210 Pr$=Pr$+"       "
45220 hi$=hi$+"         "
45240 return
45299 :
45300 rem *****     oe=2 / 2 byte
45302 :
45310 de=Peek(a1+1) : gosub 45060
45320 Pr$=Pr$+" "+he$+"    "
45330 de=Peek(a1+1) : hi$=hi$+" "+right$("   "+str$(de),
```

```
3)+"     "
45340 return
45399 :
45400 rem *****     oe=3 / 3 byte
45402 :
45410 for t=1 to 2
45420 de=peek(a1+t) : gosub 45060
45430 pr$=pr$+" "+he$
45440 de=peek(a1+t) : hi$=hi$+"   "+right$("   "+str$(de)
,3)
45450 next
45460 return
45499 :
45500 rem *****     egy bytos utasitas
45502 :
45510 pr$=pr$+"        "
45520 return
45699 :
45700 rem *****     kozvetlen
45702 :
45710 gosub 48300
45720 pr$=pr$+" #"+he$+"    "
45730 hi$=hi$+" #"+de$+"    "
45740 return
45799 :
45900 rem *****     absolut
45902 :
45910 gosub 48400 : gosub 45960
45920 pr$=pr$+"    "
45930 hi$=hi$+"    "
45940 return
45960 pr$=pr$+"   "+he$ : hi$=hi$+"   "+de$ : return
46099 :
46100 rem *****     0. lap
46102 :
46110 gosub 48300 : gosub 45960
46120 pr$=pr$+"    "
46130 hi$=hi$+"    "
46140 return
46199 :
46300 rem *****     (ind,x)
46302 :
46310 gosub 48300
46320 pr$=pr$+" ("+he$+",x)"
46330 hi$=hi$+" ("+de$+",x)"
46340 return
46499 :
46500 rem *****     (ind),y
```

```
46502 :
46510 gosub 48300
46520 pr$=pr$+" ("+he$+"),y"
46530 hi$=hi$+" ("+de$+"),y"
46540 return
46599 :
46700 rem *****     0. laP ,x
46702 :
46710 gosub 48300 : gosub 45960
46720 pr$=pr$+" ,x"
46730 hi$=hi$+" ,x"
46740 return
46799 :
46900 rem *****     0. laP ,y
46902 :
46910 gosub 48300 : gosub 45960
46920 pr$=pr$+" ,y"
46930 hi$=hi$+" ,y"
46940 return
46999 :
47100 rem *****     abszolut x
47102 :
47110 gosub 48400 : gosub 45960
47120 pr$=pr$+" ,x"
47130 hi$=hi$+" ,x"
47140 return
47199 :
47300 rem *****     abszolut y
47302 :
47310 gosub 48400 : gosub 45960
47320 pr$=pr$+" ,y"
47330 hi$=hi$+" ,y"
47340 return
47399 :
47500 rem *****     akku
47501 :
47510 pr$=pr$+"  accu  "
47520 hi$=hi$+"  accu  "
47530 return
47699 :
47700 rem *****     relativ
47702 :
47710 if peek(a1+1)>127 then 47750
47720 sP=a1+2+peek(a1+1)
47730 goto 47800
47740 :
47750 sP=a1-254+peek(a1+1)
47760 :
```

```
47800 de=sP : 9osub 45000
47810 pr$=pr$+"  "+ri9ht$("  $"+he$,5)+"    "
47820 hi$=hi$+"  "+ri9ht$("   "+str$(sP),5)+"    "
47830 return
47899 :
47900 rem *****     indirekt
47902 :
47910 9osub 48400
47920 pr$=pr$+" ("+he$+")  "
47930 hi$=hi$+" ("+de$+")  "
47940 return
47999 :
48100 rem *****     hiba
48102 :
48110 pr$=left$(pr$,16)+"                  "
48120 hi$=left$(hi$,23)+"  "
48130 if oP<32 or oP>95 then hi$=hi$+"nincs ascii-karak
ter": return
48140 hi$=hi$+"ascii-karakter : "+chr$(oP)
48150 return
48299 :
48300 rem ** 1 dec/hex byte feldol9ozasa
48302 :
48310 de=Peek(a1+1)    d9=de
48320 9osub 45060
48330 he$="  $"+he$
48340 de$=right$("     "+str$(d9),5)
48360 return
48399 :
48400 rem ** 2 dec/hex byte feldol9ozasa
48402 :
48410 de=Peek(a1+1)+Peek(a1+2)*256 : d9=de
48420 9osub 45000
48430 he$="$"+he$
48440 de$=right$("  "+str$(d9),5)
48460 return
48499 :
48500 rem ** disassembler ciklus
48502 :
48510 de=a1
48520 9osub 45000
48530 pr$=he$ : hi$="    "+str$(a1)
48540 oP=Peek(a1)
48550 9osub 49000
48560 de=oP
48570 9osub 45060
48580 pr$=pr$+"  "+he$
48590 hi$=hi$+"  "+ri9ht$(" "+str$(oP),3)
```

```
48600 on oe+1 gosub 45100,45200,45300,45400
48610 pr$=pr$+"    "+me$
48615 hi$=hi$+"    "+me$
48620 if ce%>7 then 48627
48622 on ce%+1 gosub 45500,45700,45900,46100,46300,4650
0,46700,46900
48625 goto 48630
48627 on ce%-7 gosub 47100,47300,47500,47700,47900,48100
48630 pr$=pr$+"    "+hi$
48640 return
48641 :
49000 rem ***** a me$,oe,ce% valtozokat keresi
49001 :
49010 te$="k" : se=4
49020 t=asc(te$)
49030 gosub 53700
49040 he$=mid$(t$,7+se*8,2) : gosub 32600
49050 if op=de then 49500
49060 if op<dethente$=mid$(t$,9+se*8,1):se=val(mid$(t$,
10+se*8,1))-1:goto49200
49070 te$=mid$(t$,11+se*8,1): se=val(mid$(t$,12+se*8,1)
)-1:goto49200
49200 if se=-1 then me$="*" : oe=0 : ce%=13 : return
49210 goto 49020
49500 me$=left$(t$,3)
49510 oe=val(mid$(t$,14+se*8,1))
49520 ce%=asc(mid$(t$,13+se*8,1))-48
49530 return
53700 rem *** t$ toltese t cimre
53705 on int(t/10)-2 goto 53710,53720,53730,53740,53750
,53760,53770
53710 on t-34 gosub 53801,53802,53803,53804,53805: return
53720 on t-39gosub53806,53807,53808,53809,53810,53811,5
3812,53813,53814,53815
53721 return
53730 on t-49gosub53816,53817,53818,53819,53820,53821,5
3822,53823,53824,53825
53731 return
53740 on t-59gosub53826,53827,53828,53829,53830,53831,5
3832,53833,53834,53835
53741 return
53750 on t-69gosub53836,53837,53838,53839,53840,53841,5
3842,53843,53844,53845
53751 return
53760 on t-79gosub53846,53847,53848,53849,53850,53851,5
3852,53853,53854,53855
```

```
53761 return
53770 gosub 53856 : return
53771 :
53801 t$="adc[[369[[[[1265#7k23275#8k3626d[[[[237d[[[[8
379[[#59361[[[[4271[[[[52":return
53802 t$="and#%329[[[[1225[[j23235$8j3622d[[j4233d[[[[8
339[[$59321?1[[4231[[[[52":return
53803 t$="asl[[30a[[[[:106[[[[3216[[[[620e[[[[231e,1*18
3":return
53804 t$="bcc$'290y1w1;2":return
53805 t$="bcs[(2b0u1@6;2":return
53806 t$="beq[[2f0d1P1;2":return
53807 t$="bit&-324$7$2322cj1$423":return
53808 t$="bmi[[230i1o1;2":return
53809 t$="bne*,2d88111;2":return
53810 t$="bPl[[2109101;2":return
53811 t$="brk+.100[[[[01":return
53812 t$="bvc[/250f121;2":return
53813 t$="bvs[[270h191;2":return
53814 t$="clc)8118e3e601":return
53815 t$="cld[[1d8434501":return
53816 t$="cli13158:3:501":return
53817 t$="clv[[1b8[[[[01":return
53818 t$="cmP263c9[[[[12c547=132d5487262cd637323dd46748
3d9[[[[93c1616242d1[[[[52":return
53819 t$="cPx[[3e0+1(112e4n7n232ec[[[[23":return
53820 t$="cPy573c0[[[[12c4[[[[32cc[[[[23":return
53821 t$="dec[[3c6[[[[32d6[[[[62ce[[[[23de[[[[83":return
53822 t$="dex4;1ca424401":return
53823 t$="dey[:188[[[[01":return
53824 t$="eor[[349[[[[1245:7c23255:8c3624d[[c4235d:6c58
359[[[[9341l1[[4251[[[[52":return
53825 t$="inc9<3e652<132f6[[[[62ee[[[[23fe[[[[83":return
53826 t$="inx[=1e8[[n101":return
53827 t$="iny[[1c8714101":return
53828 t$="jmP0m34cr1:4236ck1k4<3":return
53829 t$="jsr[[320[[[[23":return
53830 t$="lda?a3a9[[[[12a5a1v132b5b3a362adb4a423bd[[[[8
3b9@3b593a1[[[[42b1[[[[52":return
53831 t$="ldx[[3a2@7b212a6[[[[32b631[[72ae[[[[23beb1519
3":return
53832 t$="ldy@d3a0&1'112a4[[[[32b4@8[[62ac[[[[23bcx1@58
3":return
53833 t$="lsr[[34a:1[[:146[[[[3256[[[[624e[[[[235e[[[[8
3":return
53834 t$="noPce1ea;1n401":return
```

```
53835 t$="ora[[309[[%11205e7%23215e8%3620de1%4231d[[[[8
319[[e59301-1[[4211[[[[52":return
53836 t$="Phabi148:2>101":return
53837 t$="PhP[[108e2e401":return
53838 t$="Pla9[168#?>201":return
53839 t$="PlPhk128)1)201":return
53840 t$="rol[[32a$1[[:126[[[[3236[[[[622e[[[[233e%5m18
3":return
53841 t$="ror/136a#1[[:166[[[[3276[[[[626e#4[[237e/5a58
3":return
53842 t$="rti[[140[[[[01":return
53843 t$="rtsfs160.1/101":return
53844 t$="sbc[[3e9[[[[12e5[[[[32f5n8)262ed53)323fdn6)48
3f9[[[[93e1[[[[42f1[[[[52":return
53845 t$="secn[138$3$601":return
53846 t$="sedor1f8n3n501":return
53847 t$="sei[[178#3#601":return
53848 t$="sta9[385t1s13295t2s2628dt3s3239d[[[[8399[[[[9
381[[[[4291[[[[52":return
53849 t$="stxPw38691[[3296[[[[728e[[[[23":return
53850 t$="sty[[384r6[[3294r7[[628c[[[[23":return
53851 t$="taxtv1aa@2@401":return
53852 t$="tay[@1a8a2@101":return
53853 t$="tyauy198r2z101":return
53854 t$="tsx[L1ba[[[[01":return
53855 t$="txaxz18ar1r301":return
53856 t$="txs[[19ar5r401":return
```

7. A C 64-ES OPERÁCIÓS RENDSZERE ÉS BASIC INTERPRETERJE

Egy számítógép nemcsak mikroprocesszorból és tárból áll, ezeken kívül még különböző perifériák is csatlakoznak hozzá, mint pl. a képernyő és a nyomtató, az adattárolók, vagyis a lemezegység, ill. a kazettás egység. Ahhoz, hogy ezeket a készülékeket használni tudjuk, a mikroprocesszornak speciális programokra van szüksége. Ezek a programok a számítógépben vannak és a számítógép operációs rendszerét (pontosabban annak egy részét) alkotják.

A számítógépünk eleve ismeri a BASIC programozási nyelvet. A BASIC programok megértéséhez a számítógép ún. BASIC interpreterrel rendelkezik. Az interpreter nem más, mint a processzor nyelvén írt olyan program, amely a BASIC utasításokat elemzi és kiváltja az ezeknek megfelelő gépi műveleteket.

Ebben a fejezetben azzal foglalkozunk, hogy miként lehet ezeket a programokat (az operációs rendszert és az interpretert) egyéni céljainknak megfelelően hasznosítani. A programok alprogramokból épülnek fel, amelyeknek át kell adnunk a szükséges információkat. Ezt legtöbbször az X regiszterrel és az akkumulátorral tehetjük meg.

Az operációs rendszer alprogramjait akkor használjuk, ha valamelyik perifériára van szükségünk. De mire valók az interpreter alprogramjai? Minden programnyelvnek el kell látnia bizonyos feladatokat, mint pl.: a lebegőpontos aritmetikában való számolást, vagy a feltételes szerkezetek kezelését stb.

Gyors futást eredményező és nagyhatékonyságú gépi kódú programok írása időt rabló munka, ezért az ilyen rutinok írására csak különleges esetekben szánjuk rá magunkat.

Ez a fejezet egyrészt speciálisan a C 64-essel foglalkozik, de megjegyezzük, hogy a legtöbb mikroszámítógép operációs rendszere és interpreterje sokban hasonlít a COMMODORE-hoz. A legtöbb számítógéphez megfelelő kézikönyvek vannak, amelyekben utánanézhetünk annak, hogy hol találhatók a számítógépben az egyes rutinok és hogyan lehet ezekkel a rutinokkal a paramétereket közölni.

A kurzor kezelése

A kurzor kezeléséhez olyan alprogram szükséges, amely különböző paraméterekkel hívható. A kurzornak a képernyő adott pozíciójába való helyezéséhez az X regiszterbe a sor, az Y regiszterbe az oszlop számát kell beírni. Ezután

CLC paranccsal kell nullázni a CARRY BIT-et (átvitel), és a $FF0-nál kezdődő alprogramra kell ugrani. Ezután egy utasításban mindig vagy csak az oszlopnak, vagy csak a sornak a számát módosítjuk. Amikor azonban az oszlopot meg akarjuk változtatni, nem tudjuk, hogy melyik sorban van éppen a kurzor, ezért először le kell kérdezni a kurzor helyét. Ez ugyancsak az előbbi alprogrammal valósítható meg, de ilyenkor a CARRY BIT-et SEC paranccsal be kell állítani. A C 64-esnél a sorokat és az oszlopokat nullától kezdődően számozzuk, ezért pl.: a 17. oszlop sorszáma 16. lesz. Helyezzük pl. a kurzort a 22. oszlopba:

```
SEC
JSR $FFF0
CLC
LDY #21
JSR $FFF0
```

A képernyőtörlés

A képernyő törlésére a JSR $E544 alprogram szolgál.

Karakter kivitele a képernyőre

A karakter kiviteléhez, annak ASCII kódját az akkumulátorba kell tölteni, majd a $FFD2-ben kezdődő alprogramra kell ugrani. Vigyük ki pl. az A betűt a képernyőre:

```
LDA #65
JSR $FFD2
```

Karaktersorozat kiíratása a képernyőre

Rendelkezésünkre áll egy olyan rutin, amely egy egész karaktersorozatot ír ki a képernyőre, ha az idézőjelek közé van zárva. A kezdő idézőjel címét az Y regiszterbe és az akkumulátorba kell tölteni, majd a kiíró alprogramra, a $AB1E-re kell ugrani.

A lap száma az akkumulátorba, az adott lapon lévő hely címe az Y regiszterbe kerül. A lapra vonatkozó információt a cím magasabb helyiértékű részének (felső byte), a lapon lévő helyet a cím alacsonyabb helyiértékű részének (alsó byte) hívják. A felső byte-ot az ASSEMBLER-ben HB-vel (Hight Byte), az alsó byte-ot LB-vel (Low Byte) jelöljük. Írjuk ki a $1000-esben kezdődő karaktersorozatot.

```
Cím .EQU $1000
LDY LB-cím
LDA HB-cím
JSR $AB1E
```

A soremelés

Ugyanazt az alprogramot használjuk, mint amivel az egy karaktert lehet kiíratni. A soremelés ASCII értéke 13 (a képernyőn való soremeléshez elegendő a kocsi-vissza billentyű egyszeri megnyomása).

```
LDA #13
JSR $FFD2
```

Ezeket a feladatokat a ZVOR alprogram végzi el.

```
ZVOR .CÍMKE
LDA #13
JSR $FFD2
RTS
```

Kiíratás a nyomtatóval

Először meg kell nyitnunk a nyomtató számára egy kiviteli csatornát. Ehhez a rekeszekbe a következő információkat kell tölteni:

183 a file név hossza,
184 a logikai file szám,
185 másodlagos cím,
186 készülékszám.

Ha a file-nak nem adunk nevet, akkor a 183-asba 0-t kell írni, mivel a név hossza 0.

```
LDA #0
STA 183
```

A logikai file szám és a készülékszám 4.

```
LDA #4
STA 184
STA 186
```

A másodlagos cím 7-es, így a nyomtatót nagy- és kisbetűre kapcsoljuk.

```
LDA #7
STA 185
```

Ezután az állomány megnyitására szolgáló alprogramra ugrunk.

```
JSR $FFC0
```

A csatornaszámot az X regiszterbe kell tölteni. Az adatkivitel ezen a csatornán fog végbemenni. A következő két parancs után a kiírás a nyomtatón jelenik meg.

```
LDX #4
JSR $FFC9
```

Kiírás ismét a képernyőre

A biztonság kedvéért a nyomtatópuffer kiürítésére egy soremelés parancsot adunk.

```
JSR ZVOR
```

Ezután kijelöljük a képernyőt.

```
JSR $FFCC
```

Most zárjuk le a 4-es csatornát. Ehhez a csatornaszámot az akkumulátorba kell tölteni.

```
LDA #4
JSR $FFC3
```

A LEBEGŐPONTOS ARITMETIKA

Műveleten két lebegőpontos szám összeadását, kivonását, szorzását, osztását vagy hatványozását értjük.

A továbbiakban feltételezzük, hogy a lebegőpontos számokat a belső ábrázolási módban tároltuk.

A műveletek végrehajtására többféle lehetőségünk van. Ezek közül most kettőt ismertetünk.

a) Kijelölünk két tárterületet a lebegőpontos számoknak. A műveletek kezdetekor a változókat az így kijelölt tárrekeszbe visszük, majd a kívánt műveletnek megfelelő alprogramra ugrunk. Az eredményt egy ún. célváltozóba töltjük.

b) A két tártartomány neve, ahova a lebegőpontos számokat töltenünk kell a FAC, ill. az ARG. Erről részletesebben *A Commodore 64-es belső felépítése* c. DATA BECKER – NOVOTRADE kiadvány ROM listájából tájékozódhatunk.

A következő alprogramokat fogjuk használni:

egy változó átvitele az FAC-be;

a FAC tartalmának átvitele egy változóba;

egy változó és a FAC tartalmának összeadása;

a FAC tartalmának kivonása egy változóból;

egy változó szorzása a FAC tartalmával;

egy változó osztása a FAC tartalmával;

egy változó hatványozása a FAC tartalmával.

Valamennyi művelet a következő példa szerint megy végbe.

változó 1 művelet változó 2 eredmény változó 3

Először a változó 2 a FAC-ba töltődik. Végrehajtódik változó 1-gyel a művelet, az eredmény, ami FAC-ban van, a változó 3-ba kerül.

Nézzük végig a különböző eseteket!

1) *Egy változó átvitele a FAC-ba*

A változó címe az akkumulátorba, ill. az Y regiszterbe kerül, mégpedig a következőképpen: az Y regiszterbe a cím felső byte-ja, az akkumulátorba a cím alsó byte-ja kerül, ezután ugrás a $BBA2-ben kezdődő alprogramra.

Vigyük be pl. a Kati változó értékét a FAC-ba:

```
LDY #HB-Kati
LDA #LB-Kati
JSR $BBA2
```

2) *A FAC tartalmának átvitele egy változóba*

A változó címét az X és az Y regiszterben kell megadni. X regiszterbe az alsó, az Y regiszterbe a felső byte tartalma kerül. Az alprogram a $BBD4-ben kezdődik. A FAC tartalmát vigyük át az Anna nevű változóba:

```
LDX #LB-Anna
LDY #HB-Anna
JSR $BBD4
```

3) *Egy változó és a FAC tartalmának összeadása*

A változó címét töltsük be az akkumulátorba, ill. az Y regiszterbe, és ugorjunk a $B867-ben kezdődő alprogramra. Ez az alprogram a változó értékét először az ARG tártartományba viszi, majd hozzáadja a FAC tartalmát, és elhelyezi az eredményt a FAC-ban. A további pontokban ismertetésre kerülő műveletek a fentihez hasonlóan működnek, csupán az alprogramok címe más és más.

Adjuk össze a Balázs nevű változó tartalmát a FAC tartalmával.

```
LDY #HB-Balázs
LDA #LB-Balázs
JSR $B867
```

4) *A FAC tartalmának kivonása egy változóból*

Lásd a *3)* pontot, de a cím $B850.

5) *Egy változó szorzása a FAC tartalmával*

Lásd a *3)* pontot, de a cím $BA28.

6) *Egy változó osztása a FAC tartalmával*

Lásd a *3)* pontot, de a cím $BB0F.

7) *Egy változó hatványozása a FAC tartalmával*

Ezt a műveletet az operációs rendszernek azzal a rutinjával végezzük el, amelyik az ARG-ot hatványozza a FAC tartalmával.

A hatványozandó változót tehát először be kell vinni az ARG-be.

Hatványozzuk pl. az Anna változót a FAC tartalmával:

```
LDY #HB-Anna
LDA #LB-Anna
JSR $BABC ; Anna nach ARG
JSR $BF7B ; ARG hoch FAC
```

Függvények

A függvények értékének kiszámítása a következő:

a változó értékét a FAC-ba töltjük, ezután az adott függvénynek megfelelő alprogramra ugrunk, végül a FAC tartalmát a változóba töltjük.

A hívható függvények és az azoknak megfelelő alprogramok címei a következők:

Függvény	Kezdőcím
absolut (abszolút érték)	$BC58
actangens (árkusz tangens)	$E30E
cosinus (koszinusz)	$E264
exponent (exponenciális)	$BFED
integer (egészre kerekítés)	$BCCC
logarithmus (logaritmus)	$B9EA
speicherwert (tárérték)	$B80D
zufall (véletlenszám)	$E097
vorzeichen (előjel)	$BC39
sinus (szinusz)	$E26B
quadratwurzel (négyzetgyök)	$BF71
tangens (tangens)	$E2B4

Képezzük Feri abszolút értékét és tegyük Zoliba.

Töltsük be Feri értékét a FAC-ba:

```
LDY #HB-FERI
LDA #LB-FERI
JSR $BBA2
```

Az abszolútérték-függvény hívása:

```
JSR $BC58
```

A FAC értékének átvitele Zoliba:

```
LDY #HB-ZOLI
LDX #LB-ZOLI
JSR $BBD4
```

Az interpreter, ill. az operációs rendszer különböző rutinokkal rendelkezik az adatok ki- és bevitelére.

Az adatkivitel

A lebegőpontos számok a tárban ún. belső ábrázolású alakban vannak. Erről az alakról ASCII kódú számokká kell azokat alakítani. Ehhez először be kell vinni a számokat a FAC-ba. A FAC-ban lévő számokat egy rutin átalakítja és a $0100-es címnél kezdődő tártartományba viszi. Ezt a tártartományt be- és kiviteli puffernek, röviden puffernek nevezzük. A pufferbe 12 karakter fér. Az átalakítás után a rutin egy kettes számrendszerbeli 0-t ír a pufferbe utolsó karakterként. Így mindig pontosan tudjuk, hogy milyen hosszú az a karaktersorozat, amit ki akarunk írni. A kiírás a már előbb említett rutinnal, vagy egy általunk írt programmal megy végbe.

Nézzünk egy általunk írt adatkiviteli alprogramot!

Töltsük a változó tartalmát a FAC-be:

```
LDA #LB-Név
LDY #HB-Név
JSR $BBA2
JSR AUS ;az adatkiviteli rutin hívása
```

Az adatkiviteli rutin

```
AUS .MARKE
JSR $BDDD       ;a FAC tartalmát $0100-ba töltjük
LDX # 0         ;X regiszter törlése
AUSP .M         ;adatkiviteli ciklus
LDA $0100,X     ;a karakter betöltése az akkumulátorba
BEQ AUSE        ;ha akku = 0, akkor vége
JSR $FFD2       ;az akkumulátor tartalmának kiírása
INX             ;X regiszter tartalmát 1-gyel növeljük
BNE AUSP        ;az adatkiviteli ciklushoz
AUSE :M         ;ciklusvég
RTS             ;vége a feladatnak
```

JSR $BDDD	A FAC tartalmát a megfelelő ASCII karakterekké alakítja.
LDA $0100,X	A $0100 + X-ben lévő karaktert az akkumulátorba viszi.
BEQ AUSE	Kettes számrendszerbeli 0-t tölt az akkumulátorba, lezzel beállítja a 0-dik bitet és az AUSE-hoz ugrik.
JSR $FFD2	Az adatkiviteli rutinhoz ugrik, amely kiírja az akkumulátor tartalmát.
BNE AUSP	A BNE parancs végrehajtása gyorsabb, mint a JMP parancsé. Mivel INX tartalma egy 0-tól különböző szám, a nullabitet a parancs kinullázza. Ennek az a következménye, hogy végbemegy az ugrás.

Az adatbevitel

Az adatbevitel végrehajtásához is írunk egy alprogramot.

```
BE .MARKE
```

Az adatbevitelnél jelenjen meg egy kérdőjel a képernyőn. A kérdőjel ASCII kódja $3F.

```
LDA #$3F
JSR $FFD2 ;"?" kiírása
```

A bevitt karaktereket a $0220-as tárhelytől kezdődően helyezzük el. Indexregiszterként az X regisztert használjuk. A bevitelt végrehajtó rutin kezdőcíme a $FFCF. Az utoljára bevitt karakter után le kell nyomni a RETURN billentyűt (ASCII kód $D):

```
LDX #0
BE1 .MARKE
JSR $FFCF
STA $220,X
INX
CMP #$D
BNE BE1
```

A Return billentyű lenyomása után a

```
JSR $FFD2
```

paranccsal soremelést végzünk.

Ezután a puffer mutatóját és a beviteli mező hosszúságát megadjuk egy rutinnak. Ez a bevitt karaktereket belső formátumra alakítja és a FAC-ba helyezi el.

```
LDA #$02
STA $23
LDA #$2P
STA $22
DEX
TXA
JSR $B7B5
RTS
```

Ezek után a saját programunkba már csak a következőket kell írni:

```
JSR BE
LDX #LB-Név
LDY #HB-Név
JSR $BBD4
```

és ezzel értéket kapott a változó.

A sorszámok kiíratása

A program működését követve az interpreter egy rutinja kiírja annak a sornak a számát, ahol a program megszakadt (BREAK IN...). A sorszám alsó és felső byte-ra bontva jelenik meg. Ezek az X regiszterbe és az akkumulátorba kerülnek. A rutin a $BDCD címnél kezdődik.

A 259-es sorszám kiírása:

```
LDY #3
LDA #1
JSR $BDCD
```

A programnyelv felépítése a következő:

```
wenn A                          Operator B
dann
- -
- -                            Blokk 1
- -
sonst
- -
- -                            Blokk 2
- -
wende
```

Ha az A feltételt a B operátor kielégíti, akkor a programot a *sonst* utasításig tovább kell folytatni. Ha a feltétel nem teljesül, akkor a *sonst* utáni részre kell ugrani. Vizsgáljuk meg, hogyan megy ez végbe a feltételek szóbajövő eseteinél. Az interpreternek egy rutinja két lebegőpontos számot képes összehasonlítani. A rutin kezdőcíme $BC5B. Az első lebegőpontos szám címének a FAC-ban kell lennie, a másodikét a már ismertetett módon az akkumulátorba és az Y regiszterbe kell bevinni. Az alprogram végrehajtása után az összehasonlítás eredménye az akkumulátorból olvasható ki.

Akkumulátor = 0 A két érték azonos.
Akkumulátor = 1 Az első változó nagyobb, mint a második.
Akkumulátor = 255 Az első változó kisebb, mint a második.

Nézzük az egyes műveleteket!

Operátor: =

Az első változó átvitele a FAC-ba

```
LDA #LB-Változó 1
LDY #HB-Változó 1
JSR $BBA2
```

A második változó címének átadása

```
LDA #LB-Változó 2
LDY #HB-Változó 2
```

Ugrás az összehasonlító rutinra

```
JSR $BC5B
```

Az eredmény kiértékelése

```
CMP #0 ;akkumulátor összehasonlítása 0-val
BNE sonst
```

A program további utasításai következnek

```
JMP wende     ;vége az első blokknak
sonst .MARKE ;a második blokk kezdete
```

A második blokk utasításai

```
wende .MARKE ;feltételes utasítás vége.
```

A CMP #0 utasítással egyenlőség teljesülését vizsgáljuk. Ha az összehasonlításra kerülő értékek azonosak, akkor nem kerül sor ugrásra, ellenkező esetben ugrani kell. A többi műveletnél is ugyanezzel az elvi felépítéssel találkozunk, különbségek csak a feltételek kiértékelésénél vannak, ezért csak ezt vizsgáljuk.

Operátor: / =

A / = feltétel kiértékelése:

```
CMP#0
BEQ sonst
```

A CMP #0 utasítással a változók azonosságát vizsgáljuk. Ha azonosak, akkor a *sonst*-hoz ugrunk, ha különböznek, folytatjuk a programot.

Operátor: >

A > feltétel kiértékelése:

```
CMP #1
BNE sonst
```

Ha az első változó > a második változónál, akkor a program folytatódik, különben a *sonst*-hoz ugrunk.

Operátor: < =

A < = feltétel kiértékelése:

```
CMP #1
BEQ sonst
```

Az utasítás működése megegyezik az előző utasításéval.

Operátor: <

A < feltétel kiértékelése:

```
CMP #255
BNE sonst
```

Operátor : > =

A > = feltétel kiértékelése:

```
CMP #255
BEQ sonst
```

Egymásba szerkesztett feltételeknél a címkék helyes kijelölésére ügyelni kell! Lásd *A szemantikai elemzés és a kódgenerálás* c. fejezetben.

A fordítandó ciklusutasítás szerkezete:

```
fuer változó 1 von változó 2 bis változó 3 wiederhole
– –
– – blokk
– –
sende
```

A ciklusutasítás működése: először a változó 1-hez kezdőértékként a változó 2 értékét rendeljük. A ciklus elejét címkével jelöljük, majd megvizsgáljuk, hogy a változó 1 értéke nagyobb-e, mint változó 3 értéke. Amennyiben igen, akkor a ciklus végére ugrunk, ha nem, akkor a változó 1 értékét eggyel megnöveljük és végrehajtjuk a blokkban lévő utasításokat, ezután a ciklus elejét jelző címkére ugrunk.

```
fuer ottó von enno bis benno wiederhole
– –
– –
– –
sende
```

Legyen enno értéke 1, benno értéke 3.

```
otto értéke felveszi 1-et.
otto nagyobb már, mint benno?
Nem, ezért legyen otto = otto + 1.
Az utasítások végrehajtása.
otto = 2-vel ugrás a vizsgálathoz.
otto nagyobb, mint benno?
Nem, ezért legyen otto = otto + 1.
Az utasítások végrehajtása.
otto = 3-mal ugrás a vizsgálathoz.
otto nagyobb, mint benno?
Nem, ezért legyen otto = otto + 1.
Az utasítások végrehajtása.
otto = 4-gyel ugrás a vizsgálathoz.
Igen, ezért ugrás a ciklus végére.
```

Látjuk tehát, hogy a ciklus magját képező utasításokat háromszor hajtottuk végre.

Nézzük a fenti ciklusnak megfelelő assembler nyelvű programot!

otto értéke felveszi enno értékét.
enno értékének bevitele a FAC-ba:

```
LDY #HB-enno
LDA #LB-enno
JSR $BBA2
```

A FAC értékének beírása ottoba:

```
LDY #HB-otto
LDA #LB-otto
JSR $BBD4
```

A címke elhelyezése a ciklus elejére:

```
CIKLUS .MARKE
```

Ellenőrizzük, otto értéke nagyobb-e benno értékénél és otto értékének bevitele a FAC-ba:

```
LDY #HB-otto
LDA #LB-otto
JSR $BBA2
```

benno címének átadása:

```
LDY #HB-benno
LDA #LB-benno
JSR $BC5B
```

Az értékek összehasonlítása (nagyobb-e?):

```
CMP #1
BEQ CIKLUSVÉG
```

Nem, tovább.

A FAC értékének növelése eggyel:

```
LDA #$E8
LDY #$BF
JSR $B867
```

A FAC értékének beírása ottoba:

```
LDY #HB-otto
LDA #LB-otto
JSR $BBD4
```

A blokk utasításai következnek, ciklusvég

```
JMP CIKLUS          ;vissza az elejére
CIKLUSVÉG .MARKE ;vége a ciklusnak
```

Néhány magyarázat

A CMP #1 és BEQ címke utasításokat már ismerjük az előző fejezetből.

Az összehasonlító rutin nem változtatja meg a FAC értékét, ezért a FAC-hoz egyet hozzá lehet adni. Az összehasonlítás előtt otto értékét a FAC-ba kell tölteni, mivel még nem tudjuk, hogy a FAC miként kerül majd felhasználásra magában a ciklusban.

Az interpreter a $BFE8 címen egy 1-est tárol, amit a ciklusváltozó értékének növeléséhez használ. Ezután a FAC értéke a következő ciklusléphez otto-ban megőrződik.

Ha több egymásba szerkesztett ciklus van, akkor a címkék neveit ez esetben is megfelelő körültekintéssel kell megválasztani.

A kimenettel rendelkező ciklusokat a feltételes utasításokhoz hasonlóan kell kezelni.

Mi hiányzik még?

A számok belső ábrázolású alakban való előállításának módját A *szemantikai elemzés és a kódgenerálás* c. fejezetben találhatjuk meg.

A képernyő színének kiválasztásához a színeknek megfelelő számokat (kódokat) a tár speciális helyein kell elhelyezni. A színek kódjait a számítógép kézikönyvből kereshetjük ki.

Ezzel az ismertetéssel kívántunk kedvet ébreszteni az operációs rendszer és az interpreter alaposabb megismeréséhez.

8. HOGYAN TEHETJÜK PROGRAMJAINKAT RÖVIDEBBÉ?

A könyvünkben ismertetett programok számítógépünk tárkapacitásának nagy részét programszöveggel töltik ki úgy, hogy bővítésre nem túl sok hely marad.

Ha BASIC fordítóval rendelkezünk, akkor a programok terjedelmét fordítással csökkenthetjük, feltéve, hogy a fordítónk el tudja végezni a feladatot.

Ha nincs fordítónk, akkor a programokat kézi módszerrel rövidíthetjük, de ez nagyon fáradságos. A könyv szerzője a *Data Welt* c. folyóirat 4/1984-es számában javasolt egy megoldást, amelyet itt ismét bemutat.

Ki ne gondolt volna arra, hogy milyen jó lenne egy-egy BASIC nyelvű programot olyan röviddé és gyorssá tenni, amennyire csak lehetséges és mindezt egyetlen billentyű benyomásával.

Legkésőbb akkor gondolunk komolyan arra, hogyan lehetne tárhelyet megtakarítani, ha a rendelkezésünkre álló tárkapacitás végére értünk. Ilyenkor először azokat a megjegyzéssorokat töröljük, amelyekről azt gondoljuk, hogy feleslegesek. Ez az eljárás azonban hamar megbosszulja magát, ha később valamilyen változtatáshoz helyet keresünk a programban. Az összes felesleges üres karakter eltávolítása hosszabb programok esetében különben is nagyon aprólékos és hosszadalmas munka. Több sor összevonását az editor csupán nyolcvan karakternyi sorhosszig engedélyezi, bár a COMMODORE kézikönyvből tudjuk, hogy a számítógép 256 karakternyi hosszúságú sorokat tud feldolgozni!

A sorok átszámozásának lehetőségét általában csak olyan személyek mérlegelik, akik számíthatnak programozói segítségre.

Ha a felvetett javaslatokat akár csak egy rövid programon próbáljuk ki, akkor hamarosan kiderül, hogy sikerült ugyan helyet megtakarítani és a program működését meggyorsítani, de a program nagyon áttekinthetetlenné és a javítások számára szinte hozzáférhetetlenné vált. Emellett ez a módszer ugyancsak időigényes, és nagy koncentrációt is követel, mert egy sort csak akkor lehet a másikhoz kapcsolni, ha meggyőződtünk róla, hogy a megszüntetendő sorhoz nem címeztünk ugrást. Végül is nem azért vásároltunk számítógépet, hogy órák hosszat a programlista mellett üljünk, és üres sorokat távolítsunk el! Jobb, ha ezt a munkát is maga a számítógép végzi!

Az itt ismertetésre kerülő BASIC COMPRESSOR programmal számítógépünk programjainkat kedvező esetben max. 55%-kal rövidebb, és 40%-kal gyorsabb futásúvá alakíthatja. A hely-, ill. időmegtakarítás mértéke természetesen függ attól, hogy ki és milyen stílusban írta a programot. A szerző saját programjait

a BASIC COMPRESSOR átlagosan 45%-kal rövidebbé és 35%-kal gyorsabbá tette, annak ellenére, hogy nem túlságosan nagyvonalú a tárhelyek felhasználásában, amint ez a BASIC COMPRESSOR listájából is kitűnik.

A program kezelése

A biztonság kedvéért először is kapcsoljuk ki, majd újra be a számítógépet. Ezután töltsük be azt a programot, amelyet tömöríteni akarunk. Írjuk be a következő két sort:

```
POKE 43, (PEEK(45)+256*PEEK(46)-2)AND255
POKE 44, (PEEK(45)+256*PEEK(46)-2)/256
```

Ezzel a BASIC program kezdetét a tömörítendő program végére helyeztük át. Töltsük most be a BASIC COMPRESSOR-t és RUN paranccsal indítsuk el.

Hogyan működik a BASIC COMPRESSOR?

Először is megkérdezi, hogy hány byte hosszúak lehetnek a tömörített program sorai. Vegyük észre, hogy itt a belső sorhosszról van szó! A képernyőn megjelenő sorok általában hosszabbak, mint a belső sorhossz, mert a kulcsszavak (pl. GOSUB) egyetlen karakterként tárolódnak. A LIST utasítás azonban csak olyan sorokat tud feldolgozni, amelyek a képernyőn legfeljebb 256 karakter hosszúak. Számítógépünk tehát hosszabb sorokat tud előállítani, mint amilyeneket listázni tud!

A számítógép minden feltett kérdésre egy javaslattal válaszol, amelyet a program kezelője felülírással megváltoztathat. A bevitelt a RETURN billentyű lenyomásával fejezzük be.

A program azután azt kérdezi meg, hogy mennyi legyen a tömörített program első sorának száma, majd, hogy milyen különbséggel történjék a sorok további számozása. Végül azt is tudni szeretné, hogy kb. hány sorból áll a tömörítendő program? Ha ilyenkor túl nagy számot adunk meg, akkor előfordulhat, hogy az OUT OF MEMORY hibakiírás jelenik meg. Ha a BAD SUBSCRIPT hibakiírás tűnik fel, akkor túl kicsi számot adtunk meg.

A BASIC COMPRESSOR háromszor megy végig a tömörítendő programon és kiadja az éppen feldolgozott sor számát.

Miután a BASIC COMPRESSOR befejezte munkáját, a tárban már csak az összetömörített programunk található. Ne ijedjünk azonban meg, ha programunk összetömörített változatát alig ismerjük fel!

Hogy teszteljük a BASIC COMPRESSOR-t?

Nos, egészen egyszerűen – önmagával!

Töltsük be a BASIC COMPRESSOR-t, vigyük be a két, fent említett sort, és töltsük be a BASIC COMPRESSOR-t még egyszer.

A program RUN paranccsal való indítása után válaszoljunk a program mindegyik kérdésére egy egy RETURN paranccsal.

A program futása után a tömörített BASIC COMPRESSOR-t tároljuk. Óvatosságból kapcsoljuk ki, majd be a számítógépet. Ezután töltsük be az eredeti BASIC COMPRESSOR-t, adjuk be a szokásos két sort, és töltsük be a tárolt tömörített BASIC COMPRESSOR-t. Indítsuk el a programot, és válaszoljunk minden egyes kérdésére RETURN paranccsal. Miután a program lefutott, a VERIFY utasítással állapítsuk meg, hogy a tárban lévő program megegyezik-e a külső tárban rögzített, és ellenőrzésünkkor betöltött tömörített BASIC COMPRESSOR-ral. Ha igen, akkor sem a gépelésnél, sem a kezelésnél nem követtünk el hibát. Mielőtt a BASIC COMPRESSOR-t kipróbáljuk, mindenesetre készítsünk egy biztonsági példányt magunknak, mert a program megsemmisíti önmagát, miután hibátlanul lefutott!

Tanulmányozzuk nyugodt körülmények között a BASIC COMPRESSOR-t és a programunk tömörített változatát! Ezután bizonyára gyorsan felismerjük a programban rejlő lehetőségeket. Használjuk ki ezeket, írjunk jól dokumentált programokat, amelyeket aztán fáradság nélkül tömöríttethetünk.

P7.

a basic compressor listája

```
53000 rem ==========================
53010 rem =       volker sasse     =
53020 rem =    basic-compressor    =
53030 rem =  vc20 & cbm64 reszere  =
53040 rem =    basic version 2.0   =
53050 rem ==========================
53090 rem :
53100 rem ==========================
53110 rem =   utasitas              =
53120 rem =       #  =kurson balra  =
53130 rem =       &  =kurson lefele =
53140 rem =       '  =clr           =
53150 rem ==========================
54010 print"'&&&    basic-compressor"
54020 print"&&&    maximalis sorhosz"
54021 input"     60####";zm
54022 print"&     elso sorszam"
54023 input"     10####";sa
```

```
54024 print"&     a sorszam novekmenye"
54025 input"      10####";ab .
54027 print"&     a sorok szama"
54028 input"      300####";a9
54030 dim a(a9),m%(a9),vs(a9)
54040 a1=4608
54045 if peek(64787)=56 and peek(64788)=48 then l=2048
54050 h=peek(44)*256+peek(43):z=l-1:n1=l
54055 print"& 1. menet    sor : "
54060 z=z+1 : if z>h then 54175
54070 t=peek(z)
54080 if t=34 then gosub 57500
54090 if t=32 then poke z,7 : goto 54060
54100 if t=139 then gosub 57300 : goto 54060
54110 if t=167 then gosub 56200 : goto 54060
54120 if t=137 then gosub 56400 : goto 54060
54130 if t=141 then gosub 56400 : goto 54060
54140 if t=0 then z=z+3 : gosub 57700 : goto 54060
54150 if t=58 then gosub 57650 : goto 54060
54160 if t=143 then gosub 57600 : goto 54060
54170 goto 54060
54175 rem ========== 1. menet vege
54176 m%(1)=1
54177 gosub 57100
54178 rem ========== 2. menet vege
54180 for t=1 to vo : zn=vs(t) : gosub 57000 : next
54200 t1=1
54210 for t=1 to zl
54220 if m%(t)=1 then a(t1)=a(t) : t1=t1+1
54230 next
54235 for t=t1 to zl : a(t)=0 : next
54240 zl=t1-1
54300 z=l-1 : pz=l-1 : z4=1
54305 print:print"& 3. menet   sor : "
54310 z=z+1 : if z>h then 54400
54320 t=peek(z)
54340 if t=7 then 54310
54345 if t=34 then gosub 58000 : if t=34 then 54310
54350 if t=167 then pz=pz+1:poke pz,167:gosub56100:goto
54310
54360 if t=137 then pz=pz+1:poke pz,137:gosub55800:goto
54310
54370 if t=141 then pz=pz+1:poke pz,141:gosub55800:goto
54310
54380 if t=0 then gosub 56600 : goto 54310
54390 pz=pz+1 : poke pz,t : goto 54310
54400 pz=pz-1 : poke pz-2,0 : poke pz-1,0
54410 h=int(pz/256) : i=pz-int(pz/256)*256
```

```
54412 Poke 828,i : Poke 829,h
54413 rem ========== 3. menet vege
54414 Poke l,0
54415 Print:Print"&    ";z-Pz;"byte-tal kevesebb!"
54420 Poke 43,1 : Poke 44,18 : if l=2048 then Poke 44,8

54430 Poke 45,Peek(828) : Poke 46,Peek(829) : clr : end

55398 :
55400 rem ====== tovabbi sorszamok
55401 :
55410 z=z+1 : t=Peek(z)
55420 if t=7 then 55410
55430 if t=44 then Pz=Pz+1 : Poke Pz,44 : goto 55410
55440 if t>47 and t<58 then gosub 55600 : goto 55400
55450 z=z-1 : return
55598 :
55600 rem ====== sorszam felismero es Potolo
55602 :
55610 z=z-1 : zn$=""
55620 z=z+1
55622 if Peek(z)>47andPeek(z)<58thengosub55700: goto 55
620
55630 zn=val(zn$) : z=z-1
55640 for t1=1 to zl : if a(t1)<zn then next
55650 rem ====== t1 uj sorszam
55660 zn$=str$((t1-1)*ab+sa)
55670 for t1=2 to len(zn$)
55672 Pz=Pz+1 : Poke Pz,asc(mid$(zn$,t1,1))
55674 next
55680 return
55681 :
55700 zn$=zn$+chr$(Peek(z)) : return
55702 :
55800 rem ====== sorszamok goto / gosub utan
55802 :
55810 z=z+1 : t=Peek(z)
55820 if t=7 then 55810
55830 if t>47 and t<58 then gosub 55600 : gosub 55400 :
 return
55840 z=z-1 : return
55841 :
56000 rem ====== sorszam then / goto / gosub lezaras
56001 :
56010 z=z-1 : zn$=""
56020 z=z+1
56022 if Peek(z)>47andPeek(z)<58thengosub55700: goto 56
020
```

```
56030 zn=val(zn$)
56040 if zn<=a(z1) then gosub 57000 : goto 56055
56050 vo=vo+1 : vs(vo)=zn
56055 z=z-1
56060 return
56061 :
56100 rem ====== sorszam then utan
56102 :
56110 z=z+1 : t=Peek(z)
56120 if t=7 then 56110
56130 if t=137 then Pz=Pz+1:Poke Pz,137:gosub55800: ret
urn
56140 if t=141 then Pz=Pz+1:Poke Pz,141:gosub55800: ret
urn
56150 if t>47 and t<58 then gosub 55600 : return
56160 z=z-1 : return
56161 :
56200 rem ====== sorszam then utan
56202 :
56210 z=z+1 : t=Peek(z)
56220 if t=32 then Poke z,7 : goto 56210
56230 if t=137 then gosub56400 : return
56235 if t=141 then gosub56400 : return
56240 if t>47 and t<58 then gosub 56000 : return
56250 z=z-1 : return
56252 :
56400 rem ====== sorszamok goto / gosub utan
56402 :
56410 z=z+1 : t=Peek(z)
56420 if t=32 then Poke z,7 : goto 56410
56430 if t>47 and t<58 then gosub 56000 : gosub 56800 :
 return
56440 z=z-1 : return
56559 :
56600 rem ====== fel lehet oldani a sorszamokat ?
56610 rem ====== sorszamok valtoztatasa
56620 zn=Peek(z+3)+Peek(z+4)*256
56622 Print spc(8-len(str$(zn)))str$(zn);
56625 if z=h-1 then 56650
56630 if a(z4)=zn then 56650
56640 Pz=Pz+1 : Poke Pz,58 : z=z+4 : return
56650 Pz=Pz+1 : Poke Pz,0
56655 Poke n1,Pz+1-int((Pz+1)/256)*256
56657 Poke n1+1,int((Pz+1)/256)
56660 n1=Pz+1 : z5=(z4-1)*ab+sa
56665 Poke Pz+3,z5-int(z5/256)*256 : Poke Pz+4,int(z5/2
56)
56670 Pz=Pz+4 : z4=z4+1 : z=z+4 :return
```

```
56671 :
56800 rem ====== tovabbi sorszamok
56802 :
56810 z=z+1 : t=Peek(z)
56820 if t=32 then Poke z,7 : 9oto 56810
56830 if t=44 then 56810
56840 if t>47 and t<58 then 9osub 56000 : 9osub 57300 :
9oto 56800
56850 z=z-1 : return
56852 :
57000 rem ====== zn keresese a()-ban es tarolasa m%()-b
en
57001 :
57010 for z4=1 to zl : if a(z4)<zn then next
57020 m%(z4)=1
57030 return
57032 :
57100 rem ====== sorhossz me9allaPitasa
57102 :
57105 Print:Print"& 2. menet   sor : "
57110 z=l-1 : t=0
57120 z=z+1 : 9=Peek(z)
57130 if 9=0 then 57160
57140 if 9<>7 then zc=zc+1
57150 9oto 57120
57160 t=t+1 : if zz+zc>zm then m%(t-1)=1 : zz=zc : zc=0

57163 Print sPc(8-len(str$(zn)))str$(zn);
57165 Print sPc(8-len(str$(zn)))str$(zn);
57170 if m%(t)=1 then zz=0 : zc=0 : 9oto 57200
57180 zz=zz+zc
57190 zc=0
57200 if z=h-1 then return
57201 z=z+4 : 9oto 57120
57300 rem ====== if / on eseten a kovetkezo sor tarolasa
57302 :
57310 m%(zl+1)=1 : return
57312 :
57500 rem ====== strin9ek atolvasasa
57520 :
57530 z=z+1 : t=Peek(z) : if t<>34 and t<>0 then 57530
57540 return
57560 :
57600 rem ====== kommentsorok corlese
57602 :
57605 if Peek(z-1)=58 then Poke z-1,7 : Poke z,7 : 9oto
 57620
```

```
57606 if Peek(z-1)=7 and Peek(z-2)=58 then Poke z-2,7 :
 Poke z,7 : goto 57620
57610 zl=zl-1 : for z3=z-5 to z : Poke z3,7 : next
57620 z=z+1 : if Peek(z)<>0 then Poke z,7 : goto 57620
57630 z=z+3 : gosub 57700 : return
57632 :
57650 rem ====== kettosPontsorok torlese
57652 :
57655 if Peek(z-5)=0 and Peek(z+1)=0 then 57660
57657 return
57660 zl=zl-1 : for z3=z-5 to z : Poke z3,7 : next
57670 z=z+4 : gosub 57700 : return
57671 :
57700 rem ====== sorszamok tarolasa
57720 :
57730 zl=zl+1 : a(zl)=Peek(z)+Peek(z+1)*256
57735 Print spc(8-len(str$(a(zl))))str$(a(zl));
57740 z=z+1 : return
58000 rem ====== karakterlanc atvitel
58010 Pz=Pz+1 : Poke Pz,34
58020 z=z+1 : t=Peek(z)
58030 if t=0 or t=34 then Pz=Pz+1 : Poke Pz,34 :return
58040 Pz=Pz+1 : Poke Pz,t : goto 58020
```

P8.

```
a tomoritett compressor listaja
1 Print"{&&&    basic-compressor":Print"&&&      maximal
is sorhosz":inPut"      60####";zm:Print"&    elso sorsz
am":inPut"      10####";sa:Print"&    a sorszam novekmen
ye":inPut"      10####";ab:Print"&    a sorok szama":inP
ut"     300####";a9
2 dima(a9),m%(a9),vs(a9):al=4608:ifPeek(64787)=56andPee
k(64788)=48thenl=2048
3 h=Peek(44)*256+Peek(43):z=l-1:n1=l:Print"& 1. menet
  sor : "
4 z=z+1:ifz>hthen15
5 t=Peek(z):ift=34then9osub77
6 ift=32thenPokez,7:9oto4
7 ift=139then9osub76:9oto4
8 ift=167then9osub49:9oto4
9 ift=137then9osub54:9oto4
10 ift=141then9osub54:9oto4
11 ift=0thenz=z+3:9osub87:9oto4
12 ift=58then9osub84:9oto4
13 ift=143then9osub79:9oto4
14 9oto4
15 m%(1)=1:9osub67:fort=1tovo:zn=vs(t):9osub65:next:t1=
```

```
1:fort=1tozl:ifm%(t)=1thena(t1)=a(t):t1=t1+1
16 next:fort=t1tozl:a(t)=0:next:zl=t1-1:z=l-1:pz=l-1:z4
=1:print:print"& 3. menet   sor : "
17 z=z+1:ifz>hthen25
18 t=peek(z):ift=7then17
19 ift=34then9osub88:ift=34then17
20 ift=167thenpz=pz+1:pokepz,167:9osub44:9oto17
21 ift=137thenpz=pz+1:pokepz,137:9osub36:9oto17
22 ift=141thenpz=pz+1:pokepz,141:9osub36:9oto17
23 ift=0then9osub57:9otp17
24 pz=pz+1:pokepz,t:9oto17
25 pz=pz-1:pokepz-2,0:pokepz-1,0:h=int(pz/256):i=pz-int
(pz/256)*256:poke828,i:poke829,h:poke1,0:print:print"&
   ";z-pz;"byte-tal kevesebb!":poke43,1:poke44,18:ifl=2
048thenpoke44,8
26 poke45,peek(828):poke46,peek(829):clr:end
27 z=z+1:t=peek(z):ift=7then27
28 ift=44thenpz=pz+1:pokepz,44:9oto27
29 ift>47andt<58then9osub31:9oto27
30 z=z-1:return
31 z=z-1:zn$=""
32 z=z+1:ifpeek(z)>47andpeek(z)<58then9osub35:9oto32
33 zn=val(zn$):z=z-1:fort1=1tozl:ifa(t1)<znthennext
34 zn$=str$((t1-1)*ab+sa):fort1=2tolen(zn$):pz=pz+1:pok
epz,asc(mid$(zn$,t1,1)):next:return
35 zn$=zn$+chr$(peek(z)):return
36 z=z+1:t=peek(z):ift=7then36
37 ift>47andt<58then9osub31:9osub27:return
38 z=z-1:return
39 z=z-1:zn$=""
40 z=z+1:ifpeek(z)>47andpeek(z)<58then9osub35:9oto40
41 zn=val(zn$):ifzn<=a(zl)then9osub65:9oto43
42 vo=vo+1:vs(vo)=zn
43 z=z-1:return
44 z=z+1:t=peek(z):ift=7then44
45 ift=137thenpz=pz+1:pokepz,137:9osub36:return
46 ift=141thenpz=pz+1:pokepz,141:9osub36:return
47 ift>47andt<58then9osub31:return
48 z=z-1:return
49 z=z+1:t=peek(z):ift=32thenpokez,7:9oto49
50 ift=137then9osub54:return
51 ift=141then9osub54:return
52 ift>47andt<58then9osub39:return
53 z=z-1:return
54 z=z+1:t=peek(z):ift=32thenpokez,7:9oto54
55 ift>47andt<58then9osub39:9osub61:return
56 z=z-1:return
57 zn=peek(z+3)+peek(z+4)*256:printspc(8-len(str$(zn)))
str$(zn);:ifz=h-1then60
```

```
58 ifa(z4)=znthen60
59 pz=pz+1:Pokepz,58:z=z+4:return
60 pz=pz+1:Pokepz,0:Poken1,pz+1-int((pz+1)/256)*256:Pok
en1+1,int((pz+1)/256):n1=pz+1:z5=(z4-1)*ab+sa:Pokepz+3,
z5-int(z5/256)*256:Pokepz+4,int(z5/256):pz=pz+4:z4=z4+1
:z=z+4:return
61 z=z+1:t=Peek(z):ift=32thenPokez,7:goto61
62 ift=44then61
63 ift>47andt<58thengosub39:gosub76:goto61
64 z=z-1:return
65 forz4=1tozl:ifa(z4)<znthennext
66 m%(z4)=1:return
67 Print:Print"& 2. menet    sor  :  ":z=l-1:t=0
68 z=z+1:q=Peek(z):ifq=0then71
69 ifq<>7thenzc=zc+1
70 goto68
71 t=t+1:ifzz+zc>zmthenm%(t-1)=1:zz=zc:zc=0
72 PrintsPc(8-len(str$(zn)))str$(zn);:ifm%(t)=1thenzz=0
:zc=0:goto74
73 zz=zz+zc:zc=0
74 ifz=h-1thenreturn
75 z=z+4:goto68
76 m%(zl+1)=1:return
77 z=z+1:t=Peek(z):ift<>34andt<>0then77
78 return
79 ifPeek(z-1)=58thenPokez-1,7:Pokez,7:goto82
80 ifPeek(z-1)=7andPeek(z-2)=58thenPokez-2,7:Pokez,7:go
to82
81 zl=zl-1:forz3=z-5toz:Pokez3,7:next
82 z=z+1:ifPeek(z)<>0thenPokez,7:goto82
83 z=z+3:gosub87:return
84 ifPeek(z-5)=0andPeek(z+1)=0then86
85 return
86 zl=zl-1:forz3=z-5toz:Pokez3,7:next:z=z+4:gosub87:ret
urn
87 zl=zl+1:a(zl)=Peek(z)+Peek(z+1)*256:PrintsPc(8-len(s
tr$(a(zl)))str$(a(zl));:z=z+1:return
88 pz=pz+1:Pokepz,34
89 z=z+1:t=Peek(z):ift=0ort=34thenpz=pz+1:Pokepz,34:ret
urn
90 pz=pz+1:Pokepz,t:goto89
```

IRODALOMJEGYZÉK

1) AHO, ALFRED V. – ULLMAN, JEFFREY D: PRINCIPLES OF COMPILER DESIGN. 3. Auflage 1979 Addison Wesley

2) AHO, ALFRED V. – ULLMAN, JEFFREY D.: THE THEORY OF PARSING, TRANSLATION, AND COMPILING. Volume I: Parsing. 1972 Prentice-Hall. Volume II: Compiling. 1973 Prentice-Hall

3) ANGERHAUSEN–BRÜCKMANN–ENGLISCH–GERITS: 64 – INTERN. 3. Auflage 1983 Data-Becker Düsseldorf

4) BACKHOUSE, ROLAND C.: SYNTAX OF PROGRAMMING LANGUAGES. 1979 Prentice-Hall London

5) ENGLISCH–SZCZEPANOWSKI: DAS GROßE FLOPPY-BUCH. 2. Auflage 1984 Data-Becker Düsseldorf

6) ENGLISCH, LOTHAR: DAS MASCHINENSPRACHEBUCH ZUM COMMODORE 64. 1984 Data-Becker Düsseldorf

7) GRIES, DAVID: COMPILER CONSTRUCTION FOR DIGITAL COMPUTERS. 1971 New York–London–Sydney–Toronto

8) LEVENTHAL, LANCE A. – SAVILLE, WINTHORP: 6502 ASSEMBLY LANGUAGE SUBROUTINES: 1982 Berkeley

9) MAYER, OTTO: SYNTAXANALYSE. 1978 Zürich

10) PETERSON, JAMES I.: COMPUTER ORGANISATION AND ASSEMBLY LANGUAGE PROGRAMMING. 1978 New York

11) TREMBLAY, JEAN-PAUL – SORENSON, PAUL G.: AN IMPLEMENTATION GUIDE TO COMPILER WRITING. 1982 Mc Graw Hill

12) RAETO WEST: PROGRAMMING THE PET/CBM. 1982 London

13) WAITE, WILLIAM M. – GOOS, GERHARD: COMPILER CONSTRUCTION. 1984 New York

JEGYZETEK

Ára: 298,–Ft

SZÁMÍTÁSTECHNIKA A KÖNYVESBOLTOKBAN

NOVOTRADE – 2 C ÁRUHÁZ
1136 Bp., Balzac u. 35. Tel.: 402-954

ÁLLAMI KÖNYVTERJESZTŐ V. – NOVOTRADE 2C

BUDAPEST

Táncsics Könyvesbolt
1073 Lenin krt. 17.
Telefon: 422–178

BUDAPEST

Műszaki Könyváruház
1061 Liszt Ferenc tér 9.
Telefon: 420-353

MŰVELT NÉP KÖNYVTERJESZTŐ V. – NOVOTRADE 2C

PÉCS

Zrínyi Miklós Könyvesbolt
7621 Jókai u. 25.
Telefon: 72-12835

DEBRECEN

Szak- és ismeretterjesztő Könyváruház
4024 Hunyadi u. 8.
Telefon: 52-23237

SZOMBATHELY

Savaria Könyvesbolt
9700 Mártírok tere 1.
Telefon: 94-12341

VESZPRÉM

Kölcsey Ferenc Könyvesbolt
8200 Cserhát út 7.

BÉKÉSCSABA

Radnóti M. könyvesbolt
5600 Tanácsköztársaság út 2.
Telefon: 25-207

GYŐR

Pattantyús Á. Géza Szakkönyvesbolt
9021 Molnár Ferenc u. 9.

SZEGED

Tömörkény Könyvesbolt
6720 Lenin krt. 48.
Telefon: 62-21453

SZOLNOK

Szigligeti Könyvesbolt
5000 Ságvári krt. 35.
Telefon: 56-11133

MISKOLC

Chip-kuckó
3530 Tanácsház tér 14.

Minden érdeklődőt szeretettel vár
az ÁKV, a Művelt Nép és a NOVOTRADE RT!