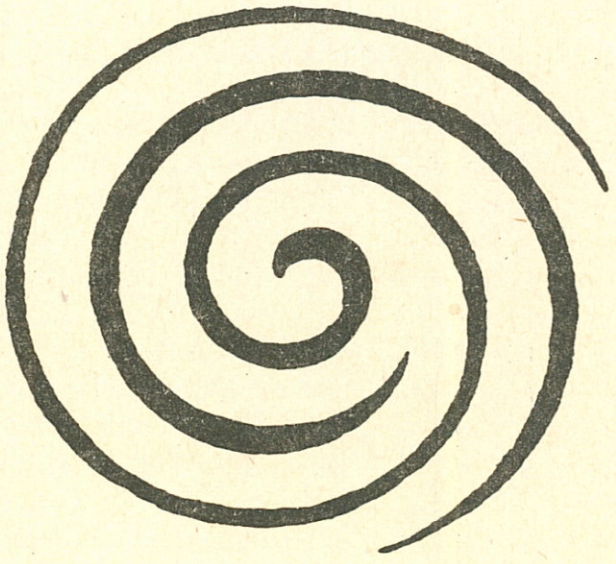
R. Baumgartner
S. Hansjakob W. Praxl

Turbo-Pascal

Elmélet és gyakorlat

R.Baumgartner
S.Hansjakob W.Praxl



Turbo-Pascal

Elmélet és gyakorlat

A könyv eredeti címe: Turbo–Pascal in Theorie und Praxis. Für MSDOS/PCDOS–Rechner mit den Turbo–Pascal–Versionen 2.0 und 3.0 (1986)

Fordította: KÖRTVÉLYESI GÉZÁNÉ
Lektorálta: SZOLEK ANDRÁS

Változatlan utánnyomás, 1989

A kiadásért felel RÉNYI GÁBOR, a NOVOTRADE RT. igazgatója
Budapest, 1987
Felelős szerkesztő: SIBA LÁSZLÓ
Műszaki szerkesztő: ERDŐSI ZOLTÁN
A szedés készült az OKISZ Labor – AGROÉPSZER GT. Nyomdájában
Készült a Somogy Megyei Nyomdaipari Vállalat Kaposvári üzemében
Felelős vezető: MIKE FERENC igazgató

ISBN 963 585 008 5

TARTALOMJEGYZÉK

ELŐSZÓ

Ez a könyv azokat az ismereteket tartalmazza, amelyekre a *Turbo–Pascal*-lal a rendszerközeli programozás területén az utóbbi években tettünk szert. Szeretnénk e rendkívüli fordítóprogram (Compiler) által a felhasználó számára megnyitott lehetőségeket és távlatokat bemutatni. Sok olyan problémát, amelyhez korábban assemblerprogramozás volt szükséges, most magasszintű programnyelv segítségével áttekinthetően oldhatunk meg.

Mivel az ilyen típusú programozáshoz egyrészt az adott operációs rendszer, másrészt az alkalmazott programnyelv részletes ismerete kell, ezért a könyv első részében a szükséges alapokkal foglalkozunk. A második rész a gyakorlat, teljes programozási példákat tartalmaz, amelyek a programozás alkalmazott fogásait és módszerét mutatják be.

Az átfogó témakör miatt a *Standard–Pascal* programnyelvet ismertnek tételezzük fel, a *Turbo–Pascal* bővítéseivel viszont behatóbban foglalkozunk.

Reméljük olyan könyv kerül az Olvasó elé, amely a hivatásos programozók számára tájékoztató- és rendezőelvként szolgál, a hobbiból programozók számára pedig megvilágítja a *Turbo–Pascal*-ban végzett igényes munka alapjait és a technikáját.

Szívélyes köszönetet mondunk ezen a helyen mindazoknak, akik e könyv elkészülésében részt vettek.

Külön megköszönjük *Herbert Kolbe* mérnök úrnak azt a fáradságos munkáját, amellyel a kéziratot átolvasta és javaslataival értékes szolgálatokat tett.

Ez a könyv nem jött volna létre a *VICTOR TECHNOLOGIES Inc.*, barátságos támogatása nélkül, segítségével több számítógép, valamint dokumentáció és szoftver állt rendelkezésünkre.

Bécs, 1985 novemberében

Robert Baumgartner

Silvia Hansjakob

Wolfgang Praxl

Mielőtt hozzákezdenénk. . .

A programpéldákat *IBM-PC XT* és *SIRIUS 1* (VICTOR 9000) gépeken teszteltük, miközben ügyeltünk arra, hogy a rendszerközeli programozás ellenére a hardver függőséget elkerüljük.

Az eredeti kiadó (IWT-Verlag GmbH) *Software–Service* szolgálatánál a könyvben szereplő programpéldák eredeti forrásszöveggel, lemezen is hozzáférhetőek, valamint a NOVOTRADE RT. kiadványát terjesztő bolthálózatban magyar forrásszöveggel.
DOS-t írtunk minden olyan helyen, ahol a mondottak mind a *PC–DOS*-ra, mind az *MS–DOS*-ra érvényesek.

A hexadecimális számokat a *Turbo–Pascal*-ban használatos formában az eléjük tett $-jellel nyomtattuk.
A programok magyarítása során a jobb olvashatóság kedvéért, helyenként ékezetes betűket is használtunk. (Ford.)

A magyar kiadás előkészítése során igyekeztünk minden, a köznyelvben még meg nem honosodott számítástechnikai kifejezésnek magyar megfelelőt keresni. Ebben a szándékunkban nem a mindenáron magyarítási törekvés vezérelt minket, hanem a szakzsargonban kevésbé járatos Olvasóknak szerettünk volna a könyv könnyebb megértésében segítséget nyújtani. A magyar kifejezések egy része már a szakirodalomban megjelent, többségük használata azonban még kiforratlan, ill. bizonyos szavak esetében először találkozhat az Olvasó a megfelelő magyar kifejezés megkeresésének kísérletével. A könyvben következetesen törekedtünk a magyar kifejezések alkalmazására – az angol megfelelőt legalább az első előfordulásnál zárójelben feltüntetve –, de a magyarázó szövegrészekben alkalmazott rövidítések, mozaikszavak természetesen az elterjedt angol nyelvű kifejezések megfelelői.

Kiadó

I. RÉSZ

ELMÉLET

1. A PC–DOS és az MS–DOS alapjai

Az *MS–DOS* jelenleg a 16 bites mikroprocesszorra a legelterjedtebb operációs rendszer. Többek között ez az operációs rendszer működik csekély eltéréssel *PC–DOS* néven az *IBM–PC*-n is.

Ebben a fejezetben először ezt az operációs rendszert megalapozó 8088/8086-os mikroprocesszort (továbbiakban 8088/8086-os) ismertetjük, majd az arra felépített tár szervezését és végül az utolsó fejezetben rátérünk arra, hogyan használjuk az operációs rendszer nagyteljesítményű és hasznos rutinjait a *DOS-megszakításokon* (interrupt) keresztül. A mikroprocesszor kimerítő és az itt közölt alapokon messze túlmenő leírást – magyar nyelven *Grohman–Eichler:* A 8086/8088-as processzor c. Data Becker–Novotrade kiadványban – *Sargent és Shoemaker* [9] könyvében találja meg az Olvasó a 8088/8086-os Assembler nyelvét és hardver részletként való alkalmazását egyaránt. *Scanlon* [10] könyve a 8088/8086-os Assembler nyelvéről és *Norton* [7] könyve az *IBM–PC*-ről már német nyelven is megjelent.

1.1. A 8088/8086-OS MIKROPROCESSZOR REGISZTEREI

A mikroszámítógép lényeges építőeleme a *központi vezérlőegység* (Central Processing Unit – CPU). Ezért a mikroszámítógép leírásakor mindig meg kell adni, melyik processzort építették be központi vezérlőegységként. A mikroprocesszorokat durván 8 bites, 16 bites és 32 bites mikroprocesszorokra osztjuk fel a számolóregiszterek és az adatsínek – amelyek a tárban az adatátvitelt teszik lehetővé – mérete szerint. Jelenleg az *Intel* által kifejlesztett 8088, 8086, 80186 és 80286 jelű 16 bites mikroprocesszorok a legelterjedtebbek. A 8088-as és a 8086-os mikroprocesszorokhoz kifejlesztett operációs rendszerek a következő fejezetekben ismertetésre kerülő *MS–DOS* és a *PC–DOS*. Ez a két mikroprocesszor nagyon hasonló egymáshoz, a különbség az, hogy a 8088-as adat busza 8 bites, a 8086-osé pedig 16 bites, így bár a 8088-as minden adatművelete kissé lassú, de korábbi és olcsóbb perifériaelemeket igényel.

A 8088/8086-os írásmód azt jelenti, hogy az említett processzor akár mint 8088-as, akár mint 8086-os alkalmazható.
A 8088/8086-os regisztereinek pontos leírása előtt rögzítsük az alapfogalmakat.

Szó: egy szó két byte-ból áll.
Byte: egy byte 8 bitből áll, a szó bal oldali byte-ot *magasabb helyi értékű byte*-nak, ill. felsőbyte-nak (high order byte) a jobb oldalit pedig *alacsonyabb helyi értékű byte*-nak, ill. alsóbyte-nak (low order byte) nevezzük.
Bit: egy byte vagy egy szó bitjeit nullával kezdve jobbról balra, a kis helyi értékű felől a nagy helyi értékű felé számozzuk.
Kbyte, kbyte: A K nagybetűvel 1000 byte-ot, a k kisbetűvel 1024 byte-ot jelzünk.

A 8088/8086-os processzornak 14 regisztere van. Az egyes regisztereket igen általánosan, míg másokat csak különleges, egészen körülhatárolt feladatok ellátására használhatjuk. A következő ábra a regiszterek funkcionális csoportokra osztott áttekintését mutatja.

ADATREGISZTEREK

Akkumulátor	AX	AH	AL	Accumulator
Bázisregiszter	BX	BH	BL	Base
Számlálóregiszter	CX	CH	CL	Count
Adatregiszter	DX	DH	DL	Data

SZEGMENSREGISZTEREK

Kódszegmens	CS	Codesegment
Adatszegmens	DS	Datasegment
Veremszegmens	SS	Stacksegment
Extra-szegmens	ES	Extrasegment

MUTATÓ- ÉS INDEXREGISZTEREK

Veremmutató	SP	Stackpointer
Bázismutató	BP	Basepointer
Forrásindex	SI	Sourceindex
Célindex	DI	Destination Index

PARANCSREGISZTER

Utasításszámláló	IP	Instruction Pointer

KAPCSOLÓREGISZTER

Kapcsolók		Flags

Adatregiszterek

Az adatregiszterek csoportja az összes többi regiszterrel ellentétben nemcsak 16 bites regiszterként (AX, BX, CX és DX), hanem 8 bites regiszterként is használhatóak (az AH az AL-lel, BH a BL-lel, a CH a CL-lel, valamint a DH a DL-lel, ahol a H felső-, az L alsóbyte jelentésű).

Az adatregiszterek univerzális regiszterek, amelyek sokféle gépi utasításhoz használhatók, bár mindegyiknek saját feladata is van:

AX az *akkumulátor,* mindegyik számolási művelethez és minden be- és kimeneti művelethez használható:
BX a *bázisregiszter,* a tárban végzett címzésekhez szükséges (indirekt címzés);
CX a *számlálóregiszter,* némely gépi utasításnál ciklusszámlálóként működik;
DX az *adatregiszter,* a dupla pontossági értékekkel végzett számolási műveletekhez és a be- és a kimeneti műveletekhez egyaránt használható.

Szegmensregiszterek

A szegmensregisztereknek a tár címzésekor van különleges jelentőségük, amelyre a következő fejezetben visszatérünk. Előzetesen tekintsük az alábbiakat:

CS a *kódszegmensregiszter,* a program kódrészeinek címzésére;
DS az *adatszegmens-regiszter,* az adatterületek címzésére;
SS a *veremszegmens-regiszter,* a verem címzésére való, ill.
ES az *extraszegmens,* ami lényegében tartalék címregiszter.

Mutató- és indexregiszterek

A mutató- és indexregiszterek az általános aritmetikai és a logikai műveletekhez is használhatók, azonban:
SI a *forrásindex* és
DI a *célindex* a tárterületek címzésére;
BP a *bázismutató* és
SP a *veremmutató* a verem címzésére használatos elsősorban.

A verem (stack) a tár tetszőleges helyén rögzített terület, amelyben az alprogram hívásakor a paraméter és a visszatérési cím átadása történik. A 2.5. alfejezetben foglalkozunk a *Turbo–Pascal* veremműveleteivel.

Utasításszámláló

Az **IP** utasításszámláló a programfutáshoz nélkülözhetetlen regiszter, mivel ez tartalmazza a soron következő gépi utasítás címét.

Kapcsolóregiszter

A kapcsolóregiszter lényegesen különbözik a többi regisztertől. Képzeljünk el egy kapcsolóbitekből álló gyűjteményt, amely – az autó műszerfalán lévő ellenőrzőlámpákhoz hasonlóan – a processzor aktuális állapotát mutatja. A következő ábrán látható, a 8088-as processzor kilenc kapcsolójának elrendezése.

				11	10	9	8	7	6	4	2	0
				OF	DF	IF	TF	SF	ZF	AF	PF	CF

Bit	0	CF	Átvitel	(Carry)
Bit	2	PF	Paritás	(Parity)
Bit	4	AF	Segédátvitel	(Auxiliary Carry)
Bit	6	ZF	Nulla	(Zero)
Bit	7	SF	Előjel	(Sign)
Bit	8	TF	Csapda	(Trap)
Bit	9	IF	Megszakítás	(Interrupt)
Bit	10	DF	Irány	(Direction)
Bit	11	OF	Túlcsordulás	(Overflow)

Három kapcsolóval a processzor állapota állítható be:
DF az *iránykapcsoló* (Direction Flag) meghatározza, hogy melyik irányból kell a karakterláncot kifejezni;
IF a *megszakításkapcsoló* (Interrupt Flag) meghatározza, hogy a processzor végrehajtsa-e a periféria megszakítási kérelmét; (0 = a periféria megszakítási kérelmét figyelmen kívül hagyja, 1 = a periféria megszakítási igényét azonnal elfogadja);
TF a *csapdakapcsoló* (Trap Flag) a processzort – tesztelési célokból – lépésenkénti végrehajtási módba állítja.

A másik hat kapcsolót általában az artimetikai és a logikai műveletek állítják be:
CF az *átvitelkapcsoló* (Carry Flag) értéke 1 lesz, ha egy számolási művelet egy regiszterben már nem ábrázolható eredményt ad, megtartja azokat a biteket is, amelyek a regiszterből az eltoló művelet során már kiesnének:

PF a *paritáskapcsoló* (Parity Flag) értéke 1, ha egy művelet eredménye páros számú 1-es értékű bitekből áll, különben 0;

AF a *segéd-átvitelkapcsoló* (Auxiliary Carry Flag) 1 értékű, ha egy számolási művelet eredménye a legalsó 4 bitben okoz hibát, a kapcsoló a BCD- és a decimális aritmetikai műveleteknél szükséges;

ZF a *nullakapcsoló* (Zero Flag) értéke 1, ha egy logikai vagy aritmetikai művelet eredménye nulla;

SF az *előjelkapcsoló* (Sign Flag) értéke 1, ha az előjeles számokkal végzett művelet eredménye negatív érték;

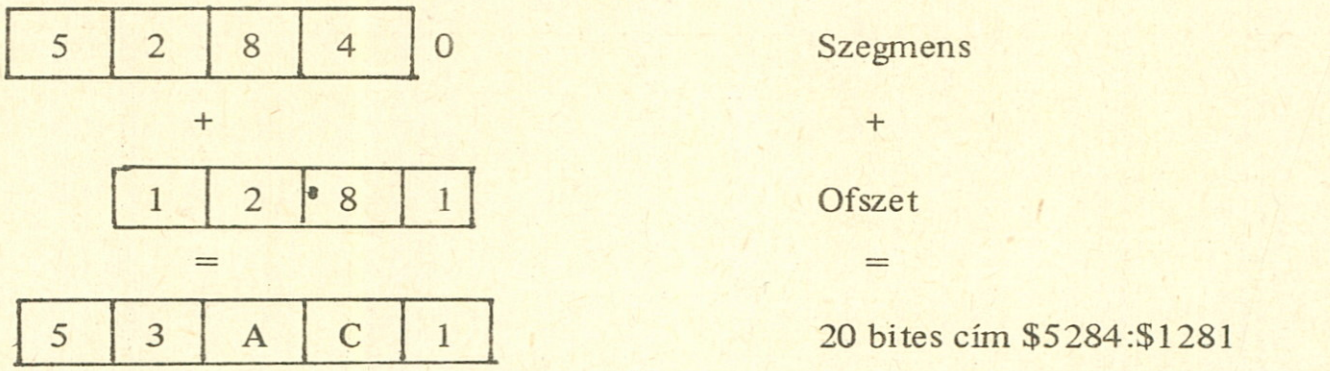
OF a *túlcsordulászkapcsoló* (Overflow Flag) értéke 1, ha a processzor egy számolási művelet eredményét előjeles számmal már nem képes ábrázolni.

1.2. A TÁR CÍMZÉSE

A tárat byte-onként címezzük. Mivel a 8088-as és a 8086-os regiszterei 16 bitesek belátható, hogy csak 65 535 byte, azaz 64 kbyte címezhető. Ténylegesen azonban a processzor 1024 kbyte-ig tud címezni.

És hogyan működik a címzés?

Egy szegmensregiszter egy másik regiszterrel egy 20 bites ún. fizikai címet (hexadecimálisan 5 pozíció) alkot. A szegmensregiszter tartalma betöltődik a 20 bites fizikai cím bal oldali részébe a másik regiszter tartalma – amelyet ofszetnek nevezünk – ehhez hozzáadódik. A tár címzésére nézzünk meg egy példát.



A következő gépi utasítás címét a CS-regiszterből (kódszegmens) és az IP-regiszterből (utasításszámláló) képezzük. Tegyük fel, hogy a CS értéke $5284 és IP értéke $1281. A CS értékét jobbról egy nullával kiegészítjük és a $52840-hez adjuk a $1281-et, így a következő utasítás fizikai címe

$$\$52840 + \$1281 = \$53AC1$$

Egy fizikai címet tehát a szegmens és az ofszet **segmens: ofszet** alakban való megadásával, példánkban

$$\$5284:\$1281$$

egyértelműen rögzíthetünk. Ez azonban azt is jelenti, hogy egy adott tárcímet több szegmens: offszet is meghatározhat.
Mivel a szegmensregisztert jobbról mindig egy nullával egészítjük ki, ezért egy szegmens mindig csak a tár egy-egy 16 byte-os területének a határánál kezdődhet. Ezeket a 16 byte nagyságú közöket blokknak (block vagy paragraph) nevezzük.
Egy rögzített szegmensérték mellett az ofszettel 64 kbyte terület címezhető, ezért erre gyakran mint szegmensre hivatkozunk.

Ahogy a regiszterek nevénél és felosztásánál ezt már megsejthettük, a címek szegmensértékét csak a szegmensregiszterek tárolhatják.
Azonosan egy tárcím képzésével egy verem címét az SS (veremszegmens) és az SP (veremmutató) regiszterpár adja, míg a DS (adatszegmens) és az ES (extra-szegmens) a különféle adat-, mutató- és indexregiszterekkel alkothat címeket.

1.3. A TÁRTERÜLET FELOSZTÁSA

20 bites fizikai címekkel címezhető a 8088-as teljes rendelkezésre álló tárterülete, azaz több mint egymillió byte. Ennek a tárterületnek csak egyes részei állnak szabadon a rendelkezésünkre, mivel bizonyos hardver-, ill. operációs rendszerelemek meghatározott területeket lefoglalnak.
A tár állandó részekre tagolása első látásra szigorú korlátozásnak tűnik, azonban – amint azt a következő fejezetben látni fogjuk – nagyon hasznos, hála a kielégítő nagyságú szabad tárterületnek. Nézzük meg az IBM–PC XT és a SIRIUS 1 tárfelosztását.

Az IBM–PC XT tárfelosztása

$00000–$003FF	Megszakítási rutinok
$00400–$005FF	ROM BIOS, BASIC-re mutató paraméterek
$00600–$9FFFF	Szabad tárterület
$A0000–$AFFFF	Videobővítés
$B0000–$BFFFF	Monokrom képernyőtár
$B8000–$CFFFF	Színes és grafikus képernyőtár
$D0000–$EFFFF	Kazetta ROM
$F0000–$F5FFF	Szabad ROM
$F6000–$FDFFF	ROM BASIC
$FE000–$FFFFF	ROM BIOS

A SIRIUS 1 tárfelosztása

$00000–$003FF	Megszakítási rutinok
$00400–$0047F	ROM BIOS paraméterek
$00480–$00C7F	Logo
$00C80–$01C7F	1. karakterkészlet
$01C80–$02C7F	2. karakterkészlet
$02C80–$7EEEF	Szabad tárterület
$7FFF0–$DFFFF	BIOS és MS–DOS
$E0000–$EFFFF	Tárban leképzett I/O
$F0000–$F3FFF	Képernyőtár
$F4000–$FBFFF	Jövőbeni bővítésre lefoglalva
$FC000–$FFFFF	ROM kiegészítés

4.4. A TÁRBAN KÉPZETT I/O MÓDSZERE

Ebben az alfejezetben a képernyő tárban való leképzésével (memory-mapping) foglalkozunk.

Minden képernyőkimenetű perifériának van saját tárja, amely minden egyes képponthoz tárolja a hozzá tartozó információt. Ez a tár a perifériának tartozéka, de a számítógép ellenőrzésével működik.
A tárban képzett képernyő (memory-mapped video) esetében ez a tároló a címezhető tár része. Leköti ugyan a tár egy részét, de lényegesen gyorsabb és rugalmasabb képernyőkezelést nyújt.
A képernyőkezelés gyors, mert a képernyőtartalom olvasása és írása valójában a tár olvasása és írása, ami jelenleg a leggyorsabb írás- és olvasásművelet, valamint rugalmas, mert így a képernyő is programozható minden magasszintű programnyelven.

A tárban képzés elvét egyes számítógépeknél pl. SIRIUS 1, a számítógép portjaira is megvalósították. A 8088/8086-osnál úgy aknázták ki a lehetőségeket, hogy az I/O pont programozójának olyan érzése van, mintha a tárba írna, miközben utasításai egy hardverelemen a port felé továbbfutnak. Az eljárás a sebesség rovására megy, de lehetővé teszi a port programozását olyan magasszintű nyelven is, amelyben – ellentétben a *Turbo–Pascal*-lal – nincs közvetlen port eltérési lehetőség.

1.5. AZ OPERÁCIÓS RENDSZER ÉS A FELHASZNÁLÓI PROGRAMOK KAPCSOLATA

A rendszerközeli programozáshoz elengedhetetlen az operációs rendszer kapcsolódási pontjainak pontos ismerete. Szerencsére a *Turbo–Pascal*-ban megengedett a *DOS* funkcióinak közvetlen hívása, így a programozó számára az összes lehetőség nyitva áll. A 2.8. alfejezetben mutatjuk be ennek a konkrét alkalmazását a *Turbo–Pascal*-ban. Először azonban ismerkedjünk meg a kapcsolódási pontokkal.
Minden *DOS*-ban létezik a tárban egy a megszakítási rutinok címeit tartalmazó táblázat. Ezek a címek csak akkor aktivizálódnak, ha a 8088/8086-ost a programvégrehajtás közben erre utasítják.

A megszakítások két csoportját különböztetjük meg:
– hardvermegszakítások;
– szoftvermegszakítások.

A hardvermegszakításokat a perifériák (pl. a lemezmeghajtó) használják, jelezve a processzornak, hogy a figyelmére igényt tartanak. Ezzel megtakarítható, hogy ezeket az egységeket a processzor állandóan figyelje, így a számítógép sebessége növelhető.
Szoftvermegszakítás lép fel egy programban pl. akkor, ha nullával osztunk. A *DOS* alatt szoftvermegszakításokat kell használni az operációs rendszer rutinjainak hívásához is. Nagy előnyt jelent, hogy az újabb operációs rendszerváltozatokban a standard rutinok megváltoztathatók anélkül, hogy az azokat hívó utasításokat változtatni kellene.
A megszakítások be vannak sorszámozva $00-tól $FF-ig. Minden sorszám a megszakítási vektor egy elemének felel meg. Ezek az *INT* assembly-utasítással hívhatók. A paraméterátadás a regiszterekben történik.

Példa: *INT 21H* a $21-es megszakítást eredményezi.

A *Turbo–Pascal*-ban ez az *Inter*-eljárással érhető el.

Példa: *Inter ($21, Register)*

A *DOS* az alábbi megszakításokat teszi lehetővé a programozónak:

$20 program vége;
$21 DOS-funkciók hívása;
$22 DOS programvég rutin;
$23 megszakítás a felhasználó által (Control–C, ill. ^C);
$24 megszakítás nem javítható hiba miatt;
$25 közvetlen olvasás;
$26 közvetlen írás;
$27 program befejezése rezidens módon.

Számunkra különösen érdekes az $21-es megszakítás, az ún. *DOS-funkciók* hívása.

A *DOS-funkciók* két csoportja
DOS 1.xx

Hagyományos karakter I/O:	$01, . . . , $0C	
Hagyományos adatkezelés:	$0D, . . . , $24,	$27, . . . , $29
Hagyományos rendszerkezelés:	$25, . . . , $26,	$2A, . . . , $2E

DOS 2.xx

Bővített rendszerkezelés:	$2F, . . . , $38,	$4C, . . . , 4F, $54
File-könyvtárkezelés:	$39, . . . , $3B,	$47
Bővített adatkezelés:	$3C, . . . , $46,	$56, $57
Bővített tárkezelés:	$48, . . . , $4B	

Az operációs rendszer funkcióinak alkalmazása bizonyos veszélyekkel jár, amelyekkel tisztában kell lennie a rendszerközeli program készítőjének.
Így lehet pl. programhibákat a számítógép ki-be kapcsolása nélkül elhárítani. Összeállítottunk néhány szabályt, amelyeket saját tapasztalataink alapján megszívlelendőnek tartottunk az Olvasó számára:

- használja, ameddig csak lehet a *Turbo* belső függvényeit és eljárásait;
- csak akkor alkalmazzon *DOS*-megszakítást, ha ez feltétlen szükséges, ha a *Turbo-Pascal* nem tartalmaz ilyen funkciót (pl. tartalomjegyzék-olvasás);
- gondosan dokumentálja az operációs rendszerhívásokat, gondoljon a tárhatásokra és a mellékhatásokra;
- az ilyen programokat csak azután tesztelje, ha az elkészült vagy a megváltoztatott programot már lemezen tárolta.

1.5.1. DOS-megszakítások

$20 **A program vége** (Terminate Program)
Ez a megszakítás a program szokásos módon való befejezése. Az adatpuffer kiürül, a file-bővítések és az írásra nyitott file-ok elvesznek. A file-ban végzett változtatások a file zárása (Close) nélkül is megmaradnak. A *DOS 2.xx*-ben ezt a funkciót a $4C vagy a $31 látja el.

$21 **DOS-funkció hívása** (Funkction Request)
A *DOS*-funkció hívásokat bővebben az 1.5.2. pontban tárgyaljuk.

A következő három megszakítás ($22–$24) a *DOS* belső céljait szolgálja és ezeket a programozó nem használhatja. Szükség esetén az adott célhoz rendelt megszakítási rutinra kerül a vezérlés.

$22 **DOS programvég rutin** (Terminate Address)

$23 **Megszakítás Controll–C-vel** (Exit Address)

$24 **Megszakítás nem javítható I/O hiba miatt** (Fatal Error Abort Address)

Hibalehetőségek:
(0) írás írásvédett lemezre
(1) ismeretlen meghajtó
(2) a meghajtó nem üzemképes (pl. a lemez nem rögzített)
(3) elvi hiba
(4) adathiba (CRC hibás)
(5) hiba az adathordozó szerkezetében
(6) keresési hiba (Techn. Ref.-ből)
(7) ismeretlen adathordozó típus
(8) nem létező szektor
(9) nincs papír a nyomtatóban
(A) íráshiba
(B) olvasási hiba
(C) általános hiba

$25 **Közvetlen olvasás** (Absolute Disk Read)
Ez a megszakítás hívja a *DOS BIOS-t* (Basic Input/Output System)
Előzetesen: a meghajtó számát az AH-ba;
az olvasni kívánt szektorok számát a CH-be;
a logikai szektort, amelynél az olvasást kezdeni kell a DX-be;
a puffer címet, amelybe másolni kell az DS:BX-be kell betölteni.
Eredmény: azok a kapcsolók, amelyek a *DOS*-hívás előtt PUSHF segítségével rögzítődtek (a veremben vannak), az aktuális információkat tartalmazzák. Ezért szükséges a POPF utasítás, mert különben a veremrend felbomlik.

Az aktuális átvitelkapcsoló (0) megmutatja, hogy a kívánt adatok a pufferben vannak-e. Ha az átvitelkapcsoló beállított, akkor az AL a *DOS* hibakódot ($24-es megszakítás) tartalmazza és az AH pedig:

$80 nincs válasz
$40 nyomkeresési hiba
$20 elvi hiba
$10 olvasási hiba (CRC hibás)
$08 DMA-túlcsordulás
$04 szektort nem talál
$03 más védelem miatt nem tud írni
$02 címet nem talál
$01 más hiba

üzenetkódokat tartalmazhatja.

$26 **Közvetlen írás** BIOS-szal (Absolute Disk Write)
Ez a megszakítás fordított adatátviteli irányú, mint a $25-ös. Egyébként a hívása és az eredménye azonos.

$27 **A programvég** (Terminate But Stay Resident)
– a program betöltve marad

A *DOS 2.xx*-ben ez a funkció a $31-essel hívható.

1.5.2. Általános eljárás DOS-funkció hívásakor

Mielőtt a $21-es *DOS*-megszakítást hívnánk, az AH regiszterben (az AX felsőbyte-ja) meg kell adnunk, milyen *DOS*-funkciót kívánunk elérni. Sok funkció elérése további paramétereket is megkövetel.

Előzetesen néhány rendszeresen ismétlődő kifejezést kell megemlítenünk:

– *ASCIIZ* (American Standard Code for Information Interchange with a Zero) olyan *ASCII*-jellánc, amely *NUL*-jellel ($00) végződik.
– A leggyakoribb file-jellemzők (attributumok):
$01 csak olvasható file (Read_only)
$02 rejtett file (Hidden)
$04 rendszerfile (System)
$08 az adathordozó neve, címkéje (Volume)
$10 az adathordozó tartalomjegyzéke (Directory)
$20 a file-t megváltoztatták az utolsó lemezre írás óta (Archive)

Hagyományos I/O műveletek $01, . . . $0C (DOS 1.xx)
A billentyűzetről végzett beolvasáshoz és a képernyőre, ill. a nyomtatóra való kiíráshoz állnak ezek a funkciók rendelkezésre.
Azért nevezik ezeket a funkciókat hagyományosnak, mert a *DOS 2.00*-tól kezdve már a *Xenix* operációs rendszerhez tervezett I/O műveletek is működnek.

A funkciók leírása

$01 **Bevitel a billentyűzetről** (Keyboard Input)
Funkció: várakozás a standard beviteli egység (billentyűzet) jelére.
Előzetesen: $01 az AH-ban.
Eredmény: a jelet az AL-be visszaadja és ugyanakkor megjeleníti a képernyőn (standard kiviteli eszköz), ezt *Echo*-nak is nevezik.
A^C ($03) kiváltja a $23-as megszakítást.

$02 **Kivitel a képernyőre** (Display Output)
Funkció: jelet ad a standard kiviteli egységre (képernyő).
Előzetesen: $02 az AH-ban, a jel a DL-ben van.
Eredmény: a jel megjelenik a képernyőn, a kurzor egy hellyel jobbra tolódik.
A^C ($03) hatástalan.

$03 **Bevitel választható egységről** (Auxiliary Input)
Funkció: várakozás a választható beviteli egység (pl. a soros port) jelére.
Előzetesen: $03 az AH-ban.
Eredmény: a jelet visszaadja az AL-nek.
A^C ($03) hatástalan.

$04 **Kivitel választható egységre** (Auxiliary Output)
Funkicó: jelet ad a választható kiviteli egységre (pl. a soros portra).
Előzetesen: $04 az AH-ban, a jel a DL-ben van.
Eredmény: a DL-ben levő jelet a választható kiviteli egységre (pl. soros portra) adja.
A^C ($03) hatástalan.

$05 **Kivitel a nyomtatóra** (Printer Output)
Funkció: jelet ad a standard nyomtatóra (LST).
Előzetesen: $05 az AH-ban, a jel DL-ben van.
Eredmény: a DL-ben lévő jel a standard nyomtatón (LST) megjelenik.
A^C hatástalan.

$06 **I/O a standard periférián** (Standard Console I/O)
Funkció: jel kiírása a standard kiviteli egységre (képernyőre), vagy egy már bevitt jel olvasása a standard beviteli egységről (billentyűzetről).

a) *Olvasáskor* (I)
Előzetesen: $06 az AH-ban, $FF a DL-ben van.
Eredmény: ha beviszünk egy jelet, akkor a nullakapcsoló üres marad és az AL tartalmazza a bevitt jelet, ellenkező esetben $00-t.
A^C ($03) hatástalan, a $03 jel továbbítódik.

b) *Iráskor* (O)
Előzetesen: $06 az AH-ban, a kiviteli jel a DL-ben van.
Eredmény: a DL-ben levő jel a standard kiviteli egységen (képernyőn) megjelenik.
A^C ($03) hatástalan.

$06 **Közvetlen bevitel a standard perifériáról** (Direct Console Input)
Funkció: várakozás a billentyűzet jelére.
Előzetesen: $07 az AH-ban.
Eredmény: a bevitt jelet AL-ben visszaadja.
Nincs *Echo* a képernyőn (a standard kiviteli egységen).
A ^C ($03) hatástalan.

$08 **Bevitel a billentyűzetről** – rejtett (Keyboard Input)
Funkció: várakozás a standard beviteli egység (billentyűzet) jelére
Előzetesen: $08 az AH-ban.
Eredmény: a bevitt jelet az AL-ben visszaadja.
Nincs *Echo* a képernyőn (standard kiviteli egységen).
A ^C ($03) kiváltja a $23-as megszakítást.

$09 **Karakterlánc írása** (Print String)
Funkció: karakterlánc küldése a standard kiviteli egységre (képernyőre).
Előzetesen: $09 az AH-ban. A DS:DX-nek egy olyan jelláncra kell mutatnia, amelyet a $-jel ($24) zár le.
Eredmény: a karakterláncot a $-jel nélkül kiírja a standard kiviteli egységre (képernyőre).
A ^C ($03) hatástalan.

$0A **Pufferelt bevitel a billentyűzetről** (Buffered Keyboard Input)
Funkció: jellánc pufferezett bevitele a standard beviteli egységről (billentyűzetről).
Előzetesen: $0A az AH-ban. A DS:DX-nek olyan előkészített beviteli tartományra kell mutatnia, amelynek a formátuma
– maximális beviteli hossz (1 byte, be kell állítani);
– tényleges beviteli hossz (1 byte);
– beviteli puffer (max. beviteli hossz +2 byte).
Eredmény: mindaddig várja a standard beviteli egységről a jelbevitelt, amíg le nem nyomjuk az *Enter*-billentyűt ($0D, Carriage Return, adatvége).
A korábban megadott max. beviteli hossz túllépésekor a további karaktereket elnyeli és a standard kimenetre (képernyőre) a $07 jelet (Bell) küld.
A bevitt jeleket az *Enter*-jel nélkül a beviteli pufferben tárolja.
A ^C ($03) hatástalan.

$0B **A standard beviteli egység állapotának ellenőrzése** (Check Standard Input Status)
Funkció: a beviteli puffer ellenőrzése, van-e karakter a billentyűzet pufferében.
Előzetesen: $0B az AH-ban.
Eredmény: ha a standard beviteli egység pufferében várakozik egy karakter, akkor az AL tartalma a funkcióhívás után $FF, különben $00.
A ^C ($03) hatástalan.

$0C **A billentyűzetpuffer törlése és egy beviteli funkció hívása** (Clear Keyboard Buffer and in voke a Keyboard Function)
Funkció: a standard beviteli egység (billentyűzet) pufferét törli és végrehajtja az 1, 6, 7, 8 beviteli feladatok egyikét.
Előzetesen: $0C az AH-ban, az AL-ben csak 1, 6, 7 vagy 8 állhat mint a kívánt beviteli funkció.
Eredmény: a standard beviteli egység puffere törlődik és hívja az AL szerinti beviteli funkciót.

Hagyományos file-kezelés $0D. . .$24, $27. . .$29 (DOS 1.xx)

A *DOS 1.xx* hagyományos file-kezelése paraméter területként az állomány-ellenőrző blokkot (File Control Block–FCB) használja, amelyben a meghajtóazonosító, a file-név, a kiterjesztés és a rekordhossz van megadva, valamint a lemez átviteli címet (Disk Transfer Address-t DTA), amely a *DOS*-nak az az aktuális paraméterterülete, ill. pufferterülete, ahol olvasáskor és íráskor az adatforgalom végbemegy.

Az *FCB* az alábbi mezőkből áll:

meghajtószám	1 byte
file-név	8 karakter
kiterjesztés	3 karakter
aktuális blokk	2 byte (128 byte alapértelmezés szerint)
rekordhossz (byte-ban)	2 byte (megnyitás után 128 az értéke)
file-nagyság	4 byte
az utolsó változat dátuma	2 byte (bitbeosztás –jjjjjjjmmmmmddddd)
az utolsó változat időpontja	2 byte (bitbeosztás –hhhhhmmmmmmmssssss)
rendszer által fenntartott tartalék terület	10 byte
aktuális rekord	1 byte (relatív rekordszám a blokkban)
relatív rekord	4 byte (relatív rekordszám a file-ban)

A bővített *FCB*-t a következő mezők egészítik ki:

kapcsoló (Flag)	1 byte $FF-el
foglalás (Reserve)	5 byte
jellemző (Attribute)	1 byte (Volume, Directory, System, Hidden, System, Hidden, Read_only)
FCB	FCB byte-k

A funkciók leírása

$0D File-puffer törlése (Disk Reset)

Funkció: az összes file-puffer törlése. Végrehajtása előtt az összes file-t le kell zárni. A nyitva maradt file-ok állapota definiálatlan, a tartalomjegyzék változatlan marad.

Előzetesen: $0D az AH-ban.

Eredmény: az összes adatpuffer kitörölve.

$0E A meghajtó kiválasztása (Select Disk)

Funkció: a megadott meghajtó lesz az alapértelmezés szerinti meghajtó.

Előzetesen: $0E az AH-ban, a meghajtó száma (0=A, 1=B,. . .) a DL-ben van.

Eredmény: a kívánt meghajtó lett az alapértelmezett (Default) meghajtó és az AL-ben a rendszer által ismert meghajtók (hajlékony- és merevlemezes egységek) darabszáma áll.

$0F File megnyitása (Open File)

Funkció: létező file megnyitása. Az FCB-ben megadott meghajtó megkeresi a kívánt file-t.

Előzetesen: $0F az AH-ban, a DS:DX-nek egy meg nem nyitott FCB-re (unopened FCB) kell mutatnia.
Eredmény: ha az AL $00 tartalmú, akkor a file-t megnyitja.
Az FCB-t a file-tartalomjegyzék adataival aktualizálja.
Amennyiben a file-t nem találja, akkor az AL értéke $FF lesz és a $16-os file létrehozás (Create a File) funkcióval nyitható meg a file.

$10 File lezárása (Close File)
Funkció: megnyitott file lezárása.
Előzetesen: $10 az AH-ban, a DS:DX-nek egy megnyitott FCB-re (opened FCB) kell mutatnia.
Eredmény: a FCB-be lerakott értékek átkerülnek a file-tartalomjegyzékbe és az AL értéke $00 lesz. Amennyiben az adatfile nem zárható le, mert pl. a hajlékony lemezt kicserélték, akkor AL értéke $FF lesz.

$11 Első file-név keresés (Search for First Entry)

$12 További file-név keresés (Search for Next Entry)
Funkció: a file-tartalomjegyzék olvasása.
Előzetesen: $11 (első kereséskor), ill. $12 (további keresés esetén) az AH-ban, a DS:DX-nek egy meg nem nyitott FCB-re, vagy bővített FCB-re kell mutatnia, amely a kereső kritériumot tartalmazza. A kereső kritérium vagy egyértelmű file-név, vagy file-névcsoport, amelyet helyettesítő karakterek (wild cards) formával (*,?) adunk meg és a $29 file-név elemző (Parse Filename) funkció viszi át az FCB-be.
Eredmény: ha a keresés eredményes volt, akkor az alkalmazott FCB-típus egy másolata a DTA-ra kerül. A másolat egy file-nevet tartalmaz a file-tartalomjegyzékből. Az AL értéke $00.
Amennyiben nem talál (további) file-nevet, akkor az AL értéke $FF lesz.
Mindkét funkció alkalmazható helykímélő módon, ha a DTA-t a $1A átállítás (Set DTA) funkcióval a DS:DX-re tesszük.

$13 File-név törlés a tartalomjegyzékből (Delete File)
Funkció: file-név törlése a tartalomjegyzékből.
Előzetesen: $13 az AH-ban, a DS:DX-nek meg nem nyitott FCB-re kell mutatnia, amely egyértelmű file-nevet vagy helyettesítő karakteres (*,?) formával megadott file-névcsoportot tartalmaz.
Eredmény: ha az AL tartalma $00, akkor az összes kívánt file-nevet törölte. Amennyiben egyetlen file-t sem törölt, akkor AL tartalma $FF lesz.

$14 Soros olvasás (Sequential Read)
Funkció: a file egy rekordjának *soros* (szekvenciális) beolvasása.
Előzetesen: $14 az AH-ban, a DS:DX-nek megnyitott FCB-re kell mutatnia. Az FCB aktuális rekord nevű mezőjében talált sorszámnak megfelelő rekordot beolvassa.
Eredmény: a rekordot a DTA-pufferbe másolja, ha eléri a file végét (End of file – EOF), akkor az AL tartalma
$03, ha a pufferbe való adatmásolás után a maradék pufferterület $00-val töltötte fel, vagy
$01, ha több adatot nem másol (EOF);
$02, ha a DTA-puffer hibás méretű, s így nem fér az egész rekord a pufferbe (ekkor a DTA-ofszet nagyobb, mint 65 535 mínusz a rekord-hossz).
$00, ha a beolvasás zavartalan, az FCB aktuális rekord nevű mezőjét aktualizálja.

$15 **Soros írás** (Sequential Write)

Funkció: a rekord file-ba való *soros* (szekvenciális) beírása.

Előzetesen: $15 az AH-ban, a DS:DX-nek egy megnyitott FCB-re kell mutatnia. Az FCB aktuális rekord nevű mezőjében talált sorszámnak megfelelő rekordot beírja.

Eredmény: a rekordot kimásolja a DTA-pufferből, ha eléri az adathordozó végét, akkor AL tartalma

$03, ha a pufferbe való adatmásolás után a maradék pufferterületet $00-al töltötte fel, vagy

$01, ha több adatot nem másol (a lemez megtelt);

$02, ha a DTA-puffer hibás méretű, s így nem fér az egész rekord a pufferbe (ekkor a DTA-ofszet nagyobb, mint 65 535 mínusz a rekordhossz).

$00, ha az írás zavartalan, a FCB aktuális rekord nevű mezőjét aktualizálja.

$16 **File létrehozása** (Create a File)

Funkció: file létrehozása és megnyitása.

Előzetesen: $16 az AH-ban, a DS:DX-nek egy meg nem nyitott FCB-re kell mutatnia, amely a kívánt file-nevet tartalmazza.

Eredmény: ha a file-tartalomjegyzékben a file-nevet megtalálja, akkor azt újra felhasználja, miközben a file hosszát nullára állítja. Az AL ekkor $00-t tartalmaz. A file-tartalomjegyzékbe új file bevitele esetén az AL tartalma $FF lesz.

$17 **File átnevezése** (Rename File)

Funkció: egy file, ill. egy file-csoport átnevezése.

Előzetesen: $17 az AH-ban, a DS:DX-nek egy *meg nem nyitott módosított FCB*-re kell mutatnia, amely a kívánt file-nevet vagy file-névcsoportot tartalmazza.

A módosított FCB formája:

meghajtóegységszám	1 byte
file-név_1	8 karakter
kiterjesztés_1	3 karakter
tartalék	6 byte
file-név_2	8 karakter
kiterjesztés_2	3 karakter

Amennyiben a file-név_2 vagy a kiterjesztés_2 kérdőjelet (?) tartalmaz, akkor a ? helyének a megfelelő karaktereket átveszi a file-név_1, ill. a kiterjesztés_1-ből.

Eredmény: hívás után az AL tartalma

$00, ha a végrehajtás zavartalan;

$FF, ha nem tudta az átnézést elvégezni, mert pl. a file-név már szerepelt a tartalomjegyzékben.

$19 **A meghajtó lekérdezése** (Current Disk)

Funkció: az alapértelmezett meghajtó lekérdezése.

Előzetesen: $19 az AH-ban.

Eredmény: az AL tartalmazza az alapértelmezés szerinti meghajtó azonosítóját (0=A, 1=B,. . .).

$1A **A lemezátviteli cím beállítása** (Set Disk Transfer Address)

Funkció: a DTA-t a kívánt tárcímre állítja.

Előzetesen: $1A az AH-ban, a DS:DX-nek a kívánt tárcímre kell mutatnia.

Eredmény: a DTA az így meghatározott tárcímet tartalmazza.

$1B **File-elhelyezési tábla tartalma** (Allocation Table Information – csak a DOS 1.xx-ben)

Funkció: a rezidens file-elhelyezési tábla (File Allocation Table – FAT) címének a behozása.

Előzetesen: $1B az AH-ban.

Eredmény: a DS:BX a funkció hívása után az alapértelmezett meghajtó FAT-jára mutat. A DX a (Allocation Unit – AU) *felosztási egységek* darabszámát, az AL az AU-hoz tartozó szektorok számát és a CX az egyes szektorok nagyságát (byte-ban) tartalmazza.

DOS 2.0 változattól a FAT-ok tárolása nem rezidens módon valósul meg, ezért a FAT-nak nem kell feltétlenül teljesnek lennie. A DS:(BX–1) a *hulladék* (dirty) byte $01-gyel mutatja, hogy a FAT módosított felépítésű. Ezt a byte-ot csak a $0D file-puffer törlésével szabad $00-ra állítani.
A file-elhelyezési tábla (FAT) 12 bites (1,5 byte-os) elemekből áll, amelyben minden egyes elem egy *cluster*-hez van rendelve. A cluster definíció szerint egy vagy több lemezszektorból áll. A file-tartalomjegyzék minden file-hoz tartalmazza az első, a FAT pedig a további clusterek számát, láncolva.

Példa: az *a.dat* nevű file a $013 AU-val kezdődik, amint ez a file-tartalomjegyzékből kiderül. A $013-as FAT-elemben a $021-es érték, a $021-es elemben a $065-ös érték áll és a $065-ös elemben található meg a file végét jelző $FF8-as érték.

A FAT első eleme mindig $FFF-et, a második eleme pedig mindig $FXX-et tartalmaz, ahol XX a *média byte* (Media Descriptor byte).
XX lehetséges értékei:

$FF kétoldalas lemez (Dual Sided)	8 szektor/sáv;
$FE egyoldalas lemez (Single Sided)	8 szektor/sáv;
$FD kétoldalas lemez (Dual Sided)	9 szektor/sáv;
$FC egyoldalas lemez (Single Sided)	9 szektor/sáv;
$F8 merevlemez (Fix Disk).	

A további FAT elemek lehetséges tartalma:

$000 AU használaton kívül;
$002–$EEF adat-AU;
$FF0–$FF7 hibás AU (formatálási hiba);
$FF8–$FFF EOF, az AU-lánc vége.

A $25-ös és a $26-os megszakításokhoz a logikai szektorszám meghatározása:
Logikai szektorszám = (AU–2) * AU-kénti_szektorok_száma + az_adattartomány_kezdete.

$1C **A file-elhelyezési tábla információi a meghajtóról** (Allocation Table Information for specific Drive – csak DOS 1.xx)

Funkció: valamelyik meghajtó rezidens file-felosztási tábla (FAT) címének behívása.

Előzetesen: $1C az AH-ban, a kívánt meghajtó száma a DL-ben van.

Eredmény: a DS:BX a hívás után a kívánt meghajtó FAT-jára mutat. DX a clusterek darabszámát, AL az AU-nkénti szektordarabszámot és CX egy szektornak byte-okban mért hosszát tartalmazza.

A *DOS 2.0* változattól a FAT-ok tárolása nem rezidens módon valósul meg, ezért a FAT-nak nem kell feltétlenül teljesnek lennie.
DS:(BX−1) a jegyzék (Listy) byte $01 értékével jelzi, hogy a FAT módosított felépítésű. Ezt a byte-ok csak a $0D file-puffer törlésével szabad $00-ra állítani.

$21 **Közvetlen olvasás** (Random Read)
Funkció: egy rögzített hosszúságú rekord közvetlen elérése a file-ban.
Előzetesen: $21 az AH-ban, a DS:DX-nek nyitott FCB-re kell mutatnia. Az FCB-mező relatív rekord nevű mezője kijelöli, hogy melyik rekordot kell a DTA-pufferbe másolni.
Eredmény: az AL tartalma
$00, ha az olvasás zavartalan, az FCB aktualizálódott;
$03, ha a file véget ért (EOF) a pufferbe még adatok másolódtak és a maradék terület $000-val töltődött fel;
$02, ha a DTA-pufferbe nem fér be az egész rekord, ilyenkor a DTA ofszetje nagyobb, mint 65 535 mínusz a rekordhossz;
$01, minden ellenkező esetben.

$22 **Közvetlen írás** (Random Write)
Funkció: rögzített hosszúságú rekord közvetlen beírása a file-ba.
Előzetesen: $22 az AH-ban, a DS:DX-nek nyitott FCB-re kell mutatnia. Az FCB relatív rekord nevű mezője jelöli ki, hogy az adatfile melyik pozíciójára másolódjon a DTA puffer tartalma.
Eredmény: az AL tartalma
$00, ha az írás zavartalan, az FCB relatív rekord nevű mezője aktualizálódott;
$02, ha a DTA pufferba nem fér be az egész rekord, ilyenkor a DTA ofszetje nagyobb, mint 65 535 mínusz a rekordhossz;
$01, minden ellenkező esetben.

$23 **A file mérete** (File Size)
Funkció: file méretének a lekérdezése.
Előzetesen: $23 az AH-ban, a DS:DX-nek meg nem nyitott FCB-re kell mutatnia. A rekordhossz nevű mezőt be kell állítani.
Eredmény: az FCB-ben megadott meghajtóhoz fordul, megkeresi a kívánt file-t. Amennyiben az AL tartalma $00, akkor az FCB-t a file-tartalomjegyzék értékeivel aktualizálta és az FCB relatív rekord nevű mezője a file rekordjainak a számát tartalmazza. Ha a file nem található meg a lemezen, AL értéke $FF lesz.

$24 **A relatív rekord nevű mező beállítása** (Set Random Record Field)
Funkció: az FCB relatív rekord mezőjének az állítása.

Előzetesen: $24 az AH-ban, a DS:DX-nek nyitott FCB-re kell mutatnia.
Eredmény: az FCB relatív rekord mezőjét a file azon pozíciójára állítja, amelyet az FCB aktuális blokk és az aktuális rekord nevű mezőinek a tartalmából határoz meg.

$27 **Közvetlen blokkolvasás** (Random Block Read)
Funkció: a $21-es közvetlen olvasáshoz hasonló eredményt nyújt, de egyszerre több rekordot, ún. blokkot tud kezelni.
Előzetesen: $27 az AH-ban, a DS:DX-nek nyitott FCB-re kell mutatnia. A CX pedig a blokkban levő rekordok számát tartalmazza.

Eredmény: AL tartalma
$00, ha az olvasás zavartalan, az FCB aktualizálódott;
$03, ha az adatfile végét elérte (EOF), még adatok másolódtak a pufferbe és a maradék pufferterület $00-val töltődött fel;
$02, ha a DTA-pufferbe nem fér be az egész rekord, ilyenkor a DTA ofszetje nagyobb, mint 65 535 mínusz a rekordhossz;
$01, minden ellenkező esetben.

$28 **Közvetlen blokkírás** (Random Block Write)
Funkció: a $22-es közvetlen íráshoz hasonló eredményt nyújt, de egyszerre több rekordot, ún. blokkot tud kezelni.
Előzetesen: $28 az AH-ban, a DS:DX-nek nyitott FCB-re kell mutatnia. A CX a blokkban levő rekordok számát tartalmazza.
Eredmény: az AL tartalma
$00, ha az olvasás zavartalan, a FCB aktualizálódott;
$02, ha a DTA-pufferbe nem fér be az egész rekord, a DTA ofszetje ekkor nagyobb, mint 65 535 mínusz a rekordhossz;
$01, minden ellenkező esetben.

$29 **File-név elemzés** (Parse Filemane)
Funkció: a karakterlánc (a file megnevezése) ellenőrzése és bevitele az FCB-be.
Előzetesen: $29 az AH-ban, az elemzésvezérlés (Parse) az AL-ben van. A DS:SI a karakterláncra, a DS:DX-nek meg nem nyitott FCB-re kell mutatnia.

Elemzésvezérlés:
7 6 5 4 3 2 1 0 vezérlőbitek
———— ———0 az elemzés az első elválasztójelnél befejeződik.
———— ———1 a vezető-elválasztójelen átmegy.
———— ——0— ha nincs meghajtószám megadva, nullát vesz fel.
———— ——1— új meghajtóazonosító kerül az FCB-be.
———— —0—— ha nincs file-név megadva, a file-nevet törli az FCB-ben.
———— —1—— ha nincs file-név megadva, a file-név változatlan marad az FCB-ben.
———— 0——— ha nincs kiterjesztés megadva, törli a kiterjesztést az FCB-ben.
———— 1——— ha nincs kiterjesztés megadva, a kiterjesztés változatlan marad az FCB-ben.
4-től 7-ig a biteket figyelmen kívül hagyja, ignorálja.

Az elválasztójelek: ., :, „ ;, +, /, <, >, =, ”, a *szóköz*, a *vagy*-jel, a *szögletes zárójel* és a *backslash* (\).
A *-jel hatására a file-névben és a kiterjesztésben a mező még üres helyeit ? -lel tölti ki.
Eredmény: az AL visszajelzése
$00 rendben;
$01 * vagy ? található a file-név megadásban;
$FF hibás a meghajtóazonosító.

Hagyományos rendszerkezelés $00, $25, . . . , $26, $2A, . . . , $2E (DOS 1.xx)
Ezekkel a funkciókkal a program befejezése és az operációs rendszerhez való visszatérés, továbbá a rendszerdátum és rendszeridő kiolvasása, ill. új beállítása lehetséges.

$00 **A program vége** (Program Terminate – csak DOS 1.xx-ben)
Funkció: a program befejezése.
Előzetesen: $00 az AH-ban.
Eredmény: a file-puffereket törli (a file-tartalomjegyzék nem aktualizálódik).

$25 **A megszakítási rutincím beállítása** (Set Interrupt Vector)
Funkció: egy új megszakítás beépítése.
Előzetesen: $25 az AH-ban, az AL a megszakítás számát és a DS:DX a megszakítási rutin kezdőcímét tartalmazza.
Eredmény: a megszakítási rutin kezdőcíme a megszakításvektorba kerül.

A *DOS 2.00* változattól a $35-ös *megszakítási rutincím lekérdezés* (Get Interrupt Vector) funkcióval a kezdőcímek az eredeti állapotba hozhatók.

$26 **Új programszegmens létrehozása** (Create a New Program Segment – csak DOS 1.xx-ben)
Funkció: új programszegmens kijelölése.
Előzetesen: $26 az AH-ban, a DX-nek egy szegmenssszámot kell tartalmaznia, amelyben az új programszegmens létrejön.
Eredmény: az aktuális programszegmens első $100 byte-ját az új szegmens kezdetére másolja.
DOS 2.00 változattól már a $4B *program töltés és végrehajtás* (Load and Execute Program) funkciót kell használni.

$2A **A rendszerdátum lekérdezése** (Get Date)
Funkció: a rendszerdátum lekérdezése, ill. átadása az adatregiszterekbe.
Előzetesen: $2A az AH-ban.
Eredmény: a rendszerdátumból a CX az évet (1980–2099 között – binárisan),
a DH a hónpot (1–12 között – binárisan) és
a DL a napot (1–31 között – binárisan) tartalmazza.

$2B **A rendszerdátum beállítása** (Set Date)
Funkció: a rendszerdátum beállítása.
Előzetesen: $2B az AH-ban,
a CX-et az évnek (1980–2099 között),
a DH-t a hónapnak (1–12 között) és
a DL-t a napnak (1–31 között) megfelelő bináris értékre kell állítani.
Eredmény: ha a bevitt dátum minden komponense helyes, akkor az AL értéke 00, különben $FF lesz.

$2C **A rendszeridő lekérdezése** (Get Time)
Funkció: a rendszeridő lekérdezése, ill. átadása az adatregiszterekbe.
Előzetesen: $2C az AH-ban.
Eredmény: a rendszeridőből
a CH az órákat (0–23 között – binárisan),
a CL a perceket (0–59 között – binárisan),
a DH a másodperceket (0–59 között – binárisan), és
a DL a századmásodperceket (0–99 – között binárisan) tartalmazza.
Az AL-ben jelennek meg a hét napjai (0 = Vasárnap, 1 = Hétfő, . . .) bináris kódokban.

$2D **A rendszeridő beállítása** (Set Time)
Funkció: a rendszeridő beállítása
Előzetesen: $2D az AH-ban,
a CH-t az óráknak (0–23 között),
a CL-t a perceknek (0–59 között),
a DH-t a másodperceknek (0–59 között) és
a DL-t a századmásodperceknek (0–99 között) megfelelő bináris értékre kell állítani.

Eredmény: ha a bevitt rendszeridő minden komponense helyes volt, akkor az AL értéke $00, különben $FF lesz.

$2E. Írásellenőrzés be/ki (Set/reset Verify Switch)
Funkció: az írásellenőrzés be-, ill. kikapcsolása.
Előzetesen: $2E az AH-ban, és az AL tartalma szerint az adatok lemezre való kiírásakor, ha
$00, akkor nincs írásellenőrzés (quick and dirty), ill., ha
$01, akkor az írásellenőrzés biztosan működik.
Eredmény: az írásellenőrzés be-, ill. kikapcsolt állapotban van.

Bővített rendszerkezelés $2F, . . . , $38, $4C, . . . , $4F, $54 (DOS 2.xx)

A *DOS 2.xx* sok új, a programozó számára jelentős segítséget nyújtó, funkciót tartalmaz.

$2F A lemezátviteli cím lekérdezése (Get DTA)
Funkció: az aktuális lemezátviteli cím lekérdezése, ill. átadása egy szegmensregiszterbe.
Előzetesen: $2F az AH-ban.
Eredmény: az ES:BX a DTA címét tartalmazza.
$1A Lemezátviteli cím beállítása (Set DTA) funkcióval állítható be a DTA a kívánt értékre.

$30 DOS-verziószám lekérdezése (Get DOS Version Number)
Funkció: a futó operációs rendszer verziószámának lekérdezése, ill. átadása az adatregiszterbe.
Előzetesen: $30 az AH-ban.
Eredmény: a funkció hívása után az AL tartalmazza a *DOS* alapverziószámot (0=1, 2=2, 3=3), az AH tartalmazza az alverziószámot (pl. 1=10, 11), a DX és a BX tartalma nulla.

$31 A program befejezése a programkód megőrzésével (Get Terminate Process and remain resident)
Funkció: a program befejezése, a programkód rezidens marad. Ez lehetővé teszi a futó program befejezését úgy, hogy a tár tartalma megőrződik. (Pl. Side Kick (TM), PoPup (TM). . .)
Előzetesen: $31 az AH-ban, az AL pedig visszatérési kódot (Return Code) tartalmazhat, amelyet parancsfile-ból (batch) az ERRORLEVEL hibaszint paranccsal vagy a $4D *visszatérési kód meghatározása* (Retrieve the Return Code from a Sub-Process) funkcióval kérdezhetünk le.
A DX a rezidensként megmaradó tárterület nagyságát tartalmazza blokkokban (16 byte-onként) megadva.
Eredmény: a program befejeződik és rezidens marad.

$33 Control-C ellenőrzés (Check)
Funkció: a $03-as megszakítás a ^C (Control-C) a standard beviteli egységen keresztül beadva, néhány meghatározott *DOS*-funkciót megszakít.
Előzetesen: $33 az AH-ban,
$01 az AL-ben és a DL-ben;
az összes DOS-funkció megszakítása;
$01 az AL-ben és $00 a DL-ben:
az eredeti állapot visszaállítása,

$00 az AL-ben:
a DL-be a rendszer aktuális állapotát jelző érték kerül.
Eredmény: az AL-nek a hívás előtti tartalma szerint a ^C ellenőrzés be-, ill. kikapcsolt állapotú, ill. a kapcsolási állapotot a DL mutatja.

$35 **Megszakítási rutincím lekérdezés** (Get Vector)
Funkció: a megszakítási rutincímnek a lekérdezése.
Előzetesen: $35 az AH-ban, az AL-nek a kívánt információkat adó rutin számát kell tartalmaznia.
Eredmény: ES:BX a megszakítási rutin kezdőcímét tartalmazza.
$25 A megszakítási rutincím beállítása (Set Interrupt Vector) funkcióval lehet ezt a címet a kívánt értékre állítani.

$36 **A lemez szabad területének lekérdezése** (Get Disk Free Space)
Funkció: a meghajtóban levő lemez helyfoglalási viszonyairól nyújt információkat.
Előzetesen: $36 az AH-ban, a DL-nek a meghajtó számát kell tartalmaznia (0=alapértelmezett, 1=A, 2=B . . .)
Eredmény: az AX tartalma $FFFF, ha a megadott számú meghajtó nem létezik.
Egyébként:
a BX a szabad clusterek,
a DX a lemezen lehetséges max. clusterek, valamint
a CX szektoronként a byte-ok és az AX clusterenként a szektorok számát tartalmazza.
Utalás: A $1B és $1C funkciók a file-elhelyezési tábla címét is megadják, ez azonban a *DOS 2.0* -tól már nem nyújt biztos információt.

$38 **A forgalmazás helyétől függő információk behozása** (Get Return Country Dependend Information)
Funkció: a forgalomba hozás helyétől függő információi az operációs rendszernek.
Előzetesen: $38 az AH-ban, és az AL-ben a $00-ás funkciókód (a DOS 2.0 és a 2.11 verziónál) van.
A DS:DX-nek egy 32 byte nagyságú terület címét kell tartalmaznia, amelybe a rendszer az információkat teszi.
Eredmény: a terület tartalma 2 byte-os formában

	idő	*dátum*
0 USA-ban	h:m:s	m/d/j
1 Európában	h:m:s	d/m/j
2 Japánban	j/m/d	h:m:s
pénzjegyek	2 byte (ASCIIZ)	
ezredespont	2 byte (ASCIIZ)	
tizedespont	2 byte (ASCIIZ)	
tartalék	24 byte	

$4C **Az eljárás befejezése** (Terminate a Process – Exit)
Funkció: lehetővé teszi a futó eljárás befejezését.
Előzetesen: $4F az AH-ban, az AL egy visszatérési kódot tartalmazhat, amelyet parancsfile-ból az ERRORLEVEL paranccsal vagy a $4D alfolyamat visszatérési kódjának meghatározása funkcióval lehet megkapni.
Eredmény: lezárja a $3C file létrehozása és a $3D file megnyitása funkciókkal megnyitott file-okat. A program befejeződik és a vezérlés a következő eljáráshívásra kerül.

$4D **A visszatérési kód meghatározása** (Retrieve the Return Code of a Sub-Process)
Funkció: egy olyan részfolyamat visszatérési kódját adja, amelyet a $31-es vagy a $4C funkcióval fejeztünk be.
Előzetesen: $4D az AH-ban.
Eredmény: az AL az alkalmazott kódot tartalmazza,
az AH tartalma
$00 normális befejezéskor,
$01 ^C-vel történt megszakításkor,
$02 készülékhiba esetén és
$03 a $31-es funkció használatakor.

$4E **Első file(-ok) keresése a tartalomjegyzékben** (Find First Matching File)

$4F **További file(-ok) keresése a tartalomjegyzékben** (Find Next Matching File)
Funkció: egy file-név vagy egy file-névcsoport keresése a file-tartalomjegyzékben.
Előzetesen: $4E az AH tartalma az első híváskor, ill. $4F minden további híváskor.
DS:DX-nek egy ASCIIZ karakterláncra kell mutatnia, amely a meghajtóazonosítót, a keresési utat (path) és a file-nevet vagy a helyettesítő karakterekkel meghatározott file-névcsoportot tartalmazza.
A CX-nek egy file-jellemzőt kell tartalmaznia (Volume, Directory, System, Hidden, Read_only).
Eredmény: a lemezátviteli címen (DTA) az alábbi információkat eredményezi:

foglalt a DOS számára	21 byte
file jellemző	1 byte
rögzítési idő	2 byte
rögzítési dátum	2 byte
file-méret	4 byte
	(egy szó alsó-felső byte-ja)

egyértelmű file-név
(file-név kiterjesztéssel) 13 byte ASCIIZ
Ha a file-tartalomjegyzék végére ér az AX tartalma $13 (l. a hibakódoknál) lesz.

$54 **Az írásellenőrzés állapota** (Get Verify State)
Funkció: az írásellenőrzés állapotának lekérdezése.
Előzetesen: $54 az AH-ban
Eredmény: AL tartalma $01, ha az írásellenőrzés bekapcsolt és $00, ha kikapcsolt állapotban van.

File-könyvtárkezelés $39, . . ., $3B, $47 (DOS 2.0-tól)

Ez a funkciócsoport a file-ok kezelését, könyvtározását teszi lehetővé, ahogy az a XENIX-ben is működik. Módot nyújt az aktuális könyvtár (alkönyvtár) megváltoztatására (CHDIR, ill. CD), új könyvtár (alkönyvtár) nyitására (MKDIR, ill. MD), üres könyvtár (alkönyvtár) törlésre (RMDIR, ill. RD) és az aktuális könyvtár (alkönyvtár) lekérdezésére (GETDIR, ill. GD).

$39 **Új alkönyvtár megnyitása** (Make a Subdirectory – MKDIR)

$3A **Alkönyvtár törlése** (Remove a Subdirectory Entry – RMDIR)

$3B **Alkönyvtár váltása** (Change the current Subdirectory – CHDIR)

Funkciók: könyvtár előállítása, törlése, váltása.

Előzetesen: $39, $3A vagy $3B az AH-ban,

a DS:DX-nek egy karakterláncra kell mutatnia, amely a meghajtóazonosítót és a keresési utat (path) tartalmazza.

Eredmény: AX tartalma

$00, ha minden akadálytalanul ment, különben a 3-as vagy az 5-ös hibakód jelentkezik az AX-ben.

$47 **Aktuális könyvtár lekérdezése** (Get current Directory – GETDIR)

Funkció: megmutatja az aktuális könyvtár felépítését, az alkönyvtárstruktúrát.

Előzetesen: $47 az AH-ban, a DL-nek a meghajtószámot (0=alapértelmezett, 1=A,. . .) kell tartalmaznia és a DS:SI-nek egy 64 hosszú ASCIIZ karakterláncra kell mutatnia, amelynek a keresési utat kell tartalmaznia.

Eredmény: a DS:SI a keresési utat tartalmazza, a meghajtóazonosító és a vezetőjel (backslash) nélkül, az AX tartalma pedig $00. A hibát az AX-ben SF jelzi.

Bővített file-kezelés $3C, . . . , $46, $56, $57 (DOS 2.0-tól)

A *DOS*-felhasználó az FCB helyett már csak egy hivatkozási file-számot kap, ez az ún. *kezelő* (Handle). Minden műveletnek hivatkoznia kell a handle-ra és így kizárhatók az FCB-n meg nem engedett műveletek. Ezért a *Turbo–Pascal*-ban a 3-as verziótól kezdve a FCB helyett a *file-illesztő blokk* (File Interface Blokk – FIB) a paraméterterület.

A FIB felépítése:

kezelő	2 byte
rekordhossz	2 byte
pufferofszet	2 byte
pufferméret	2 byte
puffermutató	2 byte
a puffer vége	2 byte
keresési út	64 byte (ASCIIZ)

$3C **File-létrehozás** (Create a File)

Funkció: új, üres file létrehozása.

Előzetesen: $3C az AH-ban, a DS:DX-nek egy ASCIIZ karakterláncra kell mutatnia, amely a meghajtóazonosítót, a keresési utat és a file nevét tartalmazza. A CX-ben egy file-jellemző lehet (pl. system, hidden, read_only).

Eredmény: az AX-be egy handle-t ad. A file nyitva áll írásra és olvasásra, ha a funkció végrehajtása után az átvitelkapcsoló beállított (értéke 1), akkor az AX-ben hibakód van. A $5-ös hibakód arra utal, hogy a teljes lemez vagy egy azonos nevű file írásvédett.

A $43-as funkcióval lehet a file-jellemzőket módosítani.

$3D **File-megnyitás** (Open a File)

Funkció: file megnyitása.

Előzetesen: a DS:DX-nek olyan karakterláncra kell mutatnia, amely a meghajtóazonosítót, a keresési utat és a file nevét tartalmazza.

Az AL-ben a hozzáférési mód található:

0 = csak olvasás

1 = csak írás

2 = írás és olvasás

Eredmény: az AX-be egy handle-t ad. A file nyitva áll és a file-mutató a file első karakterére mutat.
A végrehajtás után, ha az átvitelkapcsoló beállított, akkor az AX hibakódot tartalmaz, a $2-es, $4-es, $5-ös és SC hibakódok valamelyikét.

A $42-es funkcióval lehet a file-mutatót mozgatni és a $57-essel pedig a file rögzítési dátumát és idejét beállítani, ill. lekérdezni.

$3E **A file-feldolgozás lezárása** (Close a File Handle)
Funkció: a handle-hoz tartozó file feldolgozásának befejezése.
Előzetesen: $3E az AH-ban, a handle a BX-ben van.
Eredmény: a file feldolgozása befejeződik. A végrehajtás után, ha az átvitelkapcsoló beállított, akkor az AX-ben a $6-os hibakódot eredményezheti.

$3F **Olvasás file-ból vagy I/O egységről** (Read from a File or Device)
Funkció: kívánt darabszámú karaktert olvas be egy file-ból vagy egy perifériáról.
Előzetesen: $3F az AH-ban, a handle a BX-ben van.
A DS:DX-nek a puffer címére kell mutatnia, amelybe a CX-ben megadott számú karaktert kell az I/O egységről vagy a file-ból átvinni.
Eredmény: az AX tartalmazza a visszatéréskor a ténylegesen beolvasott karakterek számát. A funkció végrehajtása után, ha az átvitelkapcsoló beállított, akkor az AX-be a $5-ös vagy a $6-os hibakód kerül.

$40 **Írás file-ba vagy I/O egységre** (Write to a File or Device)
Funkció: kívánt darabszámú karaktert küld egy file-ba vagy egy I/O egységre.
Előzetesen: $40 az AH-ban, a handle a BX-ben van.
DS:DX-nek arra a pufferre kell mutatnia, amelyből a CX-ben meghatározott számú karaktert kell átadni.
Eredmény: az AX tartalmazza a ténylegesen kiírt karakterek számát. A végrehajtás után, ha az átvitelkapcsoló beállított, akkor az AX-be a $5-ös vagy a $6-os hibakód kerül.

$41 **File törlése a file-tartalomjegyzékből** (Delete a file from a specified Directory -- UNLINK)
Funkció: file törlése a file-tartalomjegyzékből. Az írásvédett file-okat ezzel a funkcióval nem lehet törölni.

Előzetesen: $41 az AH-ban, a DS:DX-nek egy olyan ASCIIZ karakterláncra kell mutatnia, amelynek a meghajtóazonosítót, a keresési utat és a törlendő file nevét kell tartalmaznia. A funkció végrehajtása után, ha az átvitelkapcsoló beállított, akkor az AX-be a $2-es vagy a $5-ös hibakód kerül.

$42 **A file-mutató mozgatása** (Move File read/write Pointer – LSEEK)
Funkció: A file-mutató mozgatása.
Előzetesen: az AL tartalma a mozgatási módra utal;
0 esetén CX:DX-ben az adatfile kezdetétől számított ofszet áll,
1 esetén CX:DX tartalma hozzáadódik az adatfile vége pozícióhoz.
A BX-ben a handle van.
Eredmény: a DX:AX tartalmazza visszatéréskor az új pozíciót.
A végrehajtása után, ha az átvitelkapcsoló beállított, akkor az AX-be a $1-es vagy a $6-os hibakód kerül.

$43 **A file-jellemző megváltoztatása** (Change File Mode – CHMOD)

Funkció: a file file-jellemzőjének megváltoztatása.

Előzetesen: $43 az AH-ban, az AL-nek vagy $00-t (a jellemző olvasása) vagy $01-et (a jellemző beállítása) kell tartalmaznia, a handle a BX-ben van.
A DS:DX egy olyan ASCIIZ karakterláncra mutat, amely a meghajtóazonosítót, a keresési utat és a file nevét tartalmazza.

Eredmény: ha az AL-ben $00 volt, akkor a DX a file-jellemzőt tartalmazza;
ha az AL-ben $01 volt, akkor a DX-ben az új file-jellemzők vannak.
A funkció végrehajtása után, ha az átvitelkapcsoló beállított, akkor az AX-ben a $2-es, a $3-as vagy a $5-ös hibakódok valamelyike áll.

$44 **Egységek I/O ellenőrzése** (I/O Control for Devices – IOCTL)

Funkció: az egységekkel és a file-okkal végzett I/O műveletek lekérdezése és beállítása.

A berendezésre vonatkozó lekérdezési információk:

Előzetesen: $44 az AH-ban, $01 az AL-ben, a handle a BX-ben van.

Eredmény: a DX az alábbi információt tartalmazza:

15. bit	tartalék
14. bit	CNTL
13. bit	tartalék
12. bit	Network
11–08. bit.-ig	tartalék
07. bit	ISdev
06. bit	EOF
05. bit	BIN (binary)
04. bit	tartalék
03. bit	ISclk
02. bit	ISnul
01. bit	IScot
00. bit	IScin

Az ISdev bit megmutatja, hogy az adatcsatornához I/O egység (1) vagy file (0) tartozik.
Ha ISdev=1 (I/O egységre mutat), akkor a további bitek jelentése:

EOF=0	az adatbevitel vége
BIN=1	a ^Z-t ($1A), ASCII mód, EOF-nak tekinti
BIN=0	bináris mód
ISclk=1	CLOCK egység
ISnul=1	NUL egység
IScot=1	CONsol kimeneti egység
IScin=1	CONsol bemeneti egység
CNTL=0	az egység a $4402 és a $4403 funkciókkal a vezérlést *nem képes* átvenni
CNTL=1	az egység a $4402 és a $4403 funkciókkal a vezérlést *át tudja* venni

Ha ISdev=0 (file-ra mutat), akkor a további bitek jelentése:
EOF=0 file vége.
A 0-tól 5-ig levő bitek a meghajtóazonosító számjelét tartalmazzák (0=A, 1=B, . . .).

Az egységre vonatkozó beállítási információk

Előzetesen: $44 az AH-ban, $02 az AL-ben, a handle a BX-ben van.
A DX a vezérlő információt tartalmazza, míg a DH-ban $00-nak kell lennie.

Eredmény: a vezérlő információkat átveszi.

Az egységre vonatkozó lekérdezési/beállítási információk a vezérlő csatornán keresztül

Előzetesen: $44 az AH-ban $02 (lekérdezéséhez) vagy $03 (beállításhoz) az AL-ben, a handle a BX-ben van. A DS:DX arra a tartományra mutasson, amelybe vagy amelyből a CK tartalmának megfelelő számú karaktert az egységbe át kell vinni.

Eredmény: az AX tartalmazza a ténylegesen átvitt karakterek számát.
Ha a CNTL bittartalma nulla, akkor hiba van.

A berendezésre vonatkozó lekérdezési/beállítási információk a meghajtóazonosító szám segítségével

Előzetesen: $44 az AH-ban, $04 (lekérdezéshez) vagy $5 (beállításhoz) az AL-ben.
A BL megadja a meghajtót (0=alapértelmezett, 1=A, . . .).
A DS:DX-nek egy olyan területre kell mutatnia, amelybe vagy amelyből a CX tartalmának megfelelő számú karaktert az egységről vagy az egységre át kell vinni.

Eredmény: az AX tartalmazza a ténylegesen átvitt karakterek számát.
Ha a CNTL bittartalma nulla. akkor hiba van.

A berendezés működőképességének lekérdezése

Előzetesen: $44 az AH-ban, $06 (bevitelhez) vagy $07 (kiíráshoz) az AL-ben.
A BH-ban a handle van.

Eredmény: az AX tartalma $00, ha a berendezés működésre kész, ellenkező esetben $FF.

$45 **A handle másolása** (Duplicate a File Handle – Dup)

$46 **A handle erőszakolt másolása** (Force a Duplicate of a Handle – Fdup)

Funkció: egy handle másolatának előállítása.

Előzetesen: $45 (Dup) vagy
$46 (Fdup, ha a CX-et használni kell) az AL-ben, a handle a BX-ben van és a CX a második handle-t tartalmazza.

Eredmény: a másolat file-mutatója mindig ugyanarra a helyre mutat, mind az eredetié. Amennyiben a két mutató valamelyikét mozgatjuk, akkor a másik követi.
Az új handle-t az AX fogja tartalmazni.
Ha nincs több szabad handle, akkor $4-es hibát jelez a DUP.
Az Fdup lehetővé teszi, hogy a CX-ben megadjuk a második handle-t (a másolat számára).
Ha a CX-ben megadott handle nyitva van, akkor ezt a másolás előtt lezárja.

$56 **A file átnevezése** (Rename a File)

Funkció: lehetőséget ad a file-ok nevének megváltoztatására.

Előzetesen: $56 az AH-ban, a DS:DX egy olyan ASCIIZ karakterláncra mutasson, amely a meghajtóazonosítót, a keresési utat és a file nevét tartalmazza. Az ES:DI egy olyan ASCIIZ karakterláncra mutasson, amely a file új nevét tartalmazza.

Eredmény: a file új file-nevet kap.
Az AX tartalma $00 hibátlan végrehajtás esetén, vagy a $3-as, a $5-ös és a $11-es hibakód lehet.

$57 **A file rögzítési dátumának és idejének lekérdezése, beállítása** (Get/Set a Files Date and Time)

Funkció: a rögzítés dátumát és idejét lehet lekérdezni, ill. módosítani.

Előzetesen: $57 az AH-ban, az AL-ben $00 (lekérdezéshez) vagy $01 (beállításhoz) legyen.

A BX-ben a handle van.
A CX az idő, a DX a dátum megadására való.
Eredmény: a funkció végrehajtása után, ha az átvitelkapcsoló beállított, akkor az AX-be a $1-es vagy a $6-os hibakód kerül.

Bővített tárkezelés $48, . . . , $4B (DOS 2.0-tól)

$48 **A tár lefoglalása** (Allocate Memory)
Funkció: a rendelkezésre álló tartalék tárterületek igénybevétele.
Előzetesen: $48 az AH-ban, a BX-nek a blokkok kívánt darabszámát kell tartalmaznia. A blokkok száma: (karakterek száma + 16)/16.
Eredmény: az AX a funkció hívása után az igényelt tartomány kezdőcímét, a BX a rendelkezésre álló blokkok max. darabszámát tartalmazza.
A funkció végrehajtása után, ha az átvitelkapcsoló beállított, akkor az AX-be a $7-es vagy a $8-as hibakód kerül.

$49 **A tárfoglalás törlése** (Free Allocated Memory)
Funkció: a lefoglalt tárterület felszabadítása.
Előzetesen: $49 az AH-ban, az ES felszabadítandó tárterületre mutat (amelyet a $48-as funkcióval kellett igényelni).
Eredmény: a funkció végrehajtása után, ha az átvitelkapcsoló beállított, akkor az AX-be a $7-es vagy a $9-es hibakód kerül.

$4A **A tár blokkfoglalásának módosítása** (Modify allocated Memory Blocks – SETBLOCK)
Funkció: a tárterület felosztásának módosítása.
Előzetesen: $4A az AH-ban, a BX-nek a blokkok kívánt darabszámát kell tartalmaznia. A blokkok száma: (karakterek száma + 16)/16.
Az ES-ben a módosítandó terület kezdőcíme álljon.
Eredmény: a BX tartalmazza a visszatérés után a rendelkezésre álló blokkok max. számát.
A funkció végrehajtása után, ha az átvitelkapcsoló beállított, akkor az AX-be a $7-es, a $8-as vagy a $9-es hibakód kerül.

$4B **A program betöltése és végrehajtása** (Load and Execute Program – EXEC)
Funkció: egy overlay-betöltés vagy egy programfile betöltése és végrehajtása.
A program betöltése előtt a programszegmens-prefixet (Program Segment Prefix – PSP) tölti fel:

INT $20 hívása	2 byte ($0)
a legmagasabb tárcím (Top of Memory)	2 byte ($2)
tartalék	1 byte ($4)
DOS-hívások (Long DOS–Call)	5 byte ($5)
a programvég címe (Terminate Address)	4 byte ($A)
^C címe (Break Address)	4 byte ($E)
kritikus hiba címe (Critical Error Address)	4 byte ($12)
tartalék	6 byte
környezetszegmens (Environment Segment)	2 byte ($2C)
tartalék	10 byte
FCB_1	16 byte ($5C)
FCB_2	16 byte ($6C)
parancssor és DTA (Command Line and DTA)	128 byte ($80)

A program betöltése és végrehajtása

Előzetesen: $4B az AH-ban és $00 az AL-ben. A DS:DX-nek egy olyan karakterláncra kell mutatnia, amely a meghajtóazonosítót, a keresési utat és a kívánt programfile nevét (COM vagy EXE kiterjesztéssel) tartalmazza.

Az ES:BX-ben egy paraméterblokk címének kell állnia. A paraméterblokk felépítése:

környezetcím (Enviroment Adress – DS(PSP):S2C)	2 byte
a parancssorba átmásolásra kerülő terület címe	4 byte
az első FCB címe ($5C)	4 byte
a második FCB címe ($6C)	4 byte

Eredmény: a programot betölti és végrehajtja. Az AX tartalma ekkor $00 lesz.

Megjegyzés: minden megnyitott file-hoz és I/O egységhez az alprogramok is hozzáférhetnek.

A funkció végrehajtása után, ha az átvitelkapcsoló beállított, akkor az AX-ban a $1-es, a $2-es, a $5-ös, a $8-as, a $A vagy a $B hibakód áll.

Overlay betöltése

Előzetesen: $4B az AH-ban és $03 az AL-ben. A DS:DX-nek egy olyan karakterláncra kell mutatnia, amely a meghajtóazonosítót, a keresési utat és a kívánt overlay-file nevét (kiterjesztéssel) tartalmazza. Az ES:BX-ben egy paraméterblokk címének kell állnia. A paraméterblokk felépítése:

a szegmens címe, amelybe az overlay-file betöltődik	2 byte
áthelyezési faktor (Relocation Factor), amelyet a betöltéshez kell használni	4 byte

Eredmény: az overlay betöltődik. Az AX ekkor $00. Ha az átvitelkapcsoló beállított akkor a $1-es, a $2-es, a S5-ös, a $8-as, a $A vagy a $B hibakódot tartalmazza.

1.5.3. Hibakódok

A *DOS* hibás utasítás esetén az akkumulátorba (AX) egy hibakódot küld ($0 és $58 között), ha az átvitelkapcsoló beállított.

Pl.:
- $0 hibátlan (ok)
- $1 nem létező funkció (invalid function number)
- $2 nem létező file-név (file not found)
- $3 nem létező keresési út (path not found)
- $4 túl sok nyitott file (too many open files)
- $5 az elérés visszautasítva (access denied)
- $6 érvénytelen handle (invalid handle)
- $7 sérült a tárellenőrző (memory control blokk destroyed)
- $8 a tár nem elegendő (insutficent memory)
- $9 érvénytelen tárblokk cím (invalid memory block address)
- $A érvénytelen környezet (invalid environment)
- $B érvénytelen forma (invalid format)
- $C érvénytelen elérési kód (invalid acces code)
- $D érvénytelen adat (invalid data)
- $E fenntartott
- $F érvénytelen meghajtóazonosító (invalid drive was specified)
- $10 az aktuális file-tartalomjegyzék törlésére tett kísérlet (attempt to remove the current directory)
- $11 nem azonos egység (not same device)
- $12 nincs több file (no more files) stb.

2. Az operációs rendszer csatlakozási pontjai a Turbo–Pascalhoz

A *Turbo–Pascal* egyik különleges előnye az, hogy a magasszintű nyelvek összes egyéb lehetősége mellett hozzáférést nyújt az operációs rendszerhez és a hardverhez. Ebben a fejezetben a rendszerközeli programozáshoz szükséges parancsokat, eljárásokat és funkciókat, valamint ezek felhasználási lehetőségeit részletezzük.

2.1. PARAMÉTERÁTADÁS AZ OPERÁCIÓS RENDSZERBŐL A PROGRAMBA

A *DOS* parancssorban – híváskor – paramétereket lehet átadni a főprogramnak. A paraméterek átadása a programblokkban – a COM kiterjesztésű file-ok első 256 byte-ja ($100) – keresztül történik. A *Turbo–Pascal 3.0* verziójában már előre definiált funkciók állnak rendelkezésre, amelyek nagyon megkönnyítik a paraméterátadást. Mielőtt a régebbi *Turbo–Pascal* verziók paraméterátadásával foglalkoznánk, a 3.0 verzió kényelmesebb lehetőségét ismertetjük.

Paraméterátadás a 3.0 verzióban

Az előre meghatározott funkciók

ParamCount	paraméter:	nincs
	funkció típusa:	integer
ParamStr	paraméter:	egy egész szám
	funkció típusa:	string

A ParamCount a parancssorban megadott paraméterek darabszámát adja. Figyelembe kell venni, hogy a paraméterek legalább egy szóközzel el vannak egymástól választva.
A ParamStr az egész számmal kiválasztott paraméter értékét adja.

Példa:

```
{ ParTeszt.pas }
{ Pr. I. 2.1.  }

PROGRAM ParTeszt;    { Parameter teszt }
VAR I: integer;
BEGIN
   for i:=1 to ParamCount
     do begin
       writeln (i:2, '. parameter:  ',ParamStr(i));
     end;
END.
```

A könyv II. részében az 1.11. és a 4.3. programpéldákban alkalmaztuk ezeket a funkciókat.

Paraméterátadás a Turbo-verziókban

A paraméterátadás csak a következőkben leírt módon lehetséges – a *DOS* parancssorról a főprogramba – a *Turbo–Pascal 1.0* és *2.0* verziójában, de működik a *3.0* verzióban is.
A program betöltését követően a COM file-ban a programblokk $80-as ofszetjétől kezdve helyezkedik el a programnév után álló paramcssor tartalma. Az ofszet $80-as byte-ja tartalmazza ennek a parancssornak a hosszát. A programblokk kezdetének a címe CS:0, így a minket érdeklő rekordot CS:$80-nal címezhetjük.
A következő ábra mutatja, hogyan helyezkedik el a programblokkban a *DOS* parancssor tartalma.

Parancssor:	A:>parmtest p1 p2 p3
Programblokk:	CS:$00 (a . COM file kezdete)
	CS:$80–$89 09 p1 p2 p3

Az ábrából az is kitűnik, hogy a programnév és a paraméterek közötti elválasztójel is bekerül a programblokkba. Nagyon fontos, hogy a paraméterértékeket közvetlenül a program kezdetén kiolvassuk a programblokkból és a megfelelő segédváltozókba betöltsük, mivel az első file-műveletnél a CS:$80-as cím felülíródik.

Példa:

```
{ P_Teszt.pas }
{ Pr. I. 2.2. }

PROGRAM P_Teszt;
VAR Param_s : string[255] absolute Cseg:$80;
    Param_l : byte absolute Cseg:$80;
BEGIN
    Writeln('Parameter: ',param_s);
    Writeln('Hossz:     ',param_l);
END.
```

INLINE

Az *Inline* utasítással – az utasítási rész tetszőleges helyén – a *Turbo–Pascal* programba gépi kódú részprogramot iktathatunk be. Ehhez, a ferde vonallal elválasztott kódelemeket gömbölyű zárójelek közé téve az Inline utasítás mögé kell írni.

Hová kell a gépi kódú utasítás?

Az olyan ritka esetekben, ahol a *Turbo–Pascal*-ban nincs a kívánt gépi utasításnak megfelelője. Pl. a regiszter *PUSH* és *POP* utasításai a minden regisztert megváltoztató *INTR* hívás előtt.

Továbbá, gyors rutinokat gépi kódban lehet megírni a leghatékonyabban.

A gépi kódban való programozás igen fáradságos. Eredményes kivitelezéséhez Assembler ismeretek is szükségesek. A Függelékben megtalálható a gépi utasítások felsorolása.

2.3. KÜLSŐ ALPROGRAMOK HÍVÁSA

A *Turbo–Pascal*-ban két lehetőség van a külső alprogramok behívására.

1. A *Turbo–Pascal* alprogram:
   az *Include* paranccsal beszúrható a főprogram forrásszövegébe. Az include-file használatát a 3.2. alfejezetben írjuk le részletesen.
2. A gépi kódú alprogram:
   a tárgykódba való fordítás közben szúrjuk a programba. Ezeket ugyanúgy, mint az összes többi *Turbo–Pascal* alprogramot mint *Function*-t vagy mint *Procedure*-t az átadandó paraméterekkel együtt kell deklarálni. A deklarációkor a külső alprogramot az erre a célra fenntartott *External* szóval jelöljük, amelyhez hozzáfűzzük a gépi kódú programfile nevét. A típusok, a konstansok és a változók deklarációja, valamint az utasítási rész elmarad.

Példa:

```
{ ExtProgr.pas }
{ Pr. I. 2.3. }


VAR  s : string[255];
     i : integer;

Function Caps(c:char) : char; EXTERNAL 'Caps.bin';
FUNCTION Ascii(c:char): byte; EXTERNAL 'Asciich';

BEGIN
      readln(s);
      for i:=1 to Length(s) do Write(Caps(s[i]);
      for i:=1 to Length(s) do Write(ascii(s[i]);

END.
```

A beolvasott karaktert nagybetűre formálja át (caps külső funkció), ezután minden jel ASCII kódját kinyomtatja (ascii külső funkció).

A *caps* külső funkció a *caps.bin* file-ban az *ascii* külső funkció az *asciich.com* file-ban van tárolva, ez utóbbiról a kiterjesztést azért nem kell a programban megadni, mert a COM file-kiterjesztés az alapértelmezett.

Csak akkor hívható be egy külső gépi kódú alprogram, ha azt assembly nyelven készítették és két feltételnek eleget tesz.

1. Mindegyik szegmensregisztert az alprogram kezdetén a verembe el kell tenni.
2. Nem lehet semmilyen hivatkozás az adatszegmensre, ezért nem használhatók lokális változók.

2.4. PROGRAMOK ÖSSZELÁNCOLÁSA

Különösen a nagy, képernyővezérelt alkalmazásoknál célszerű a szoftver felosztása több programra.
Hogyan kell ezeket a programokat összefűzni anélkül, hogy a felhasználónak a *DOS* parancssorban programokat és paramétereket kellene megadnia?
A *DOS*-t ismerők azonnal a parancsfile-t (Betch) ajánlanák, amelyben több parancssor és az azokat vezérlő utasítások találhatók. Azonban a *Turbo–Pascal*-ban is van egyszerű lehetőség arra, hogy a főprogramból egy vagy több külső programot vezéreljünk. Két eljárás áll ehhez rendelkezésre:
a *Chain* és az *Execute*.

Mindkét eljárás mellé paraméterként egy file-változót kell megadni, amelyet előzőleg az *Assign* eljárással a meghívandó program tárgykódjához rendelünk.
A meghívásra kerülő programot vagy *COM* kiterjesztésű file-ként a fordítóprogram C opciójával, vagy *CHN* kiterjesztésű file-ként H opcióval kell előállítani. A *COM* kiterjesztésű file aktivizálása az *Execute* eljárással, míg a *CHN* kiterjesztésű file aktivizálása *Chain* eljárással valósul meg.

A *COM* és a *CHN* kiterjesztésű file-ok között az a különbség, hogy *CHN* kiterjesztésű file nincs a *Turbo–Pascal* fordító könyvtárában (Library) tárolva, ezért ezek a file-ok kisebb tárhelyet foglalnak el és így gyorsabban betölthetők. Fontos, hogy az első behívásra kerülő program COM kiterjesztésű legyen.
A lényeges különbség a programláncolás és az alprogram hívása között az, hogy az alprogramból a visszatérés után a programfutás a hívást követő első utasítással folytatódik, míg a *Chain* vagy az *Execute* eljárással való programhíváskor az új program főprogramként töltődik a tárba anélkül, hogy a visszatérési cím tárolásra kerülne.

A forrásprogram fordítása
Ha a *Turbo–Pascal* kézikönyvben utánanézünk a programláncok egyes programjai fordításának azt találjuk, hogy a kódszegmensnek, az adatszegmensnek és a veremszegmensnek egy meghatározott minimális nagyságot el kell érnie ahhoz, hogy később ne legyen zavaró hatással, de lehetséges más megoldás is!

Közös változó
Két lehetőség is van arra, hogy egy programlánc egyes programjaiból az azonos változók között adatátadás menjen végbe.

a) Abszolút címzés
A közös változók az *absolute* változójellemzővel (attributummal) a programozó által ismert tárcímekre kerülnek. A módszer hátrányai nyilvánvalóak, az abszolút címek kijelölésekor gondoskodni kell arról, hogy az operációs rendszer tárkezelését ne zavarják, valamint ezek a programok aligha futtathatók változtatás nélkül más számítógépen.

b) Közös globális változó
A változók a programdeklaráció sorrendjében kerülnek a tárba. Így, ha a több programban közösen alkalmazott változók mindig a programok deklarációs részei kezdetén, azonos sorrendben, azonos típusmegjelöléssel szerepelnek, akkor ezek a programok a tár egyazon helyéhez fordulnak, s az itt rögzített értéket egyaránt használhatják.

Példa:
A vezérlő programban MENÜ kerül a képernyőre. Választás szerint vagy befejeződik a feldolgozás, vagy a három feldolgozó program (feldolg1.pas, feldolg2. pas, feldolg3.pas) egyike hívódik meg. A feldolgozó programból a vezérlés mindig visszatér a főprogramba.

```
{ Vezerlo.pas }
{ Pr. I. 2.4. }

PROGRAM Vezerlo;

VAR
    parameter  :   string[10];
    valasz     :   byte;
    p1,p2,p3   :   file;
BEGIN
   Assign(p1,'feldolg1.chn');
   Assign(p2,'feldolg2.chn');
   Assign(p3,'feldolg3.chn');
   ClrScr;
   GotoXY(30,3);  Write('MENU');
   GotoXY(30,8);  Write('1 ... Letrehozas');
   GotoXY(30,10); Write('2 ... Feldolgozas');
   GotoXY(30,12); Write('3 ... Nyomtatas');
   GotoXY(30,14); Write('4 ... Vege');
   repeat
   GotoXY(30,20); WRITE('Az On valasztasa: ');
   Readln(valasz)
   until (valasz>=1) and (valasz<=4);
   case valasz of
     1: begin
          parameter:='p1 valasztas';
          Chain(p1);
        end;
     2: begin
          parameter:='p2 valasztas';
          Chain(p2);
        end;
     3: begin
          parameter:='p3 valasztas';
          Chain(p3);
        end;
     4: ;
   end;
END.
```

```
{ Feldolg1.pas }
{ Pr. I. 2.4/a }

PROGRAM Feldolg1;

TYPE

   t_s10 = string[10];

VAR

   par1 : t_s10;
   main : file;
```

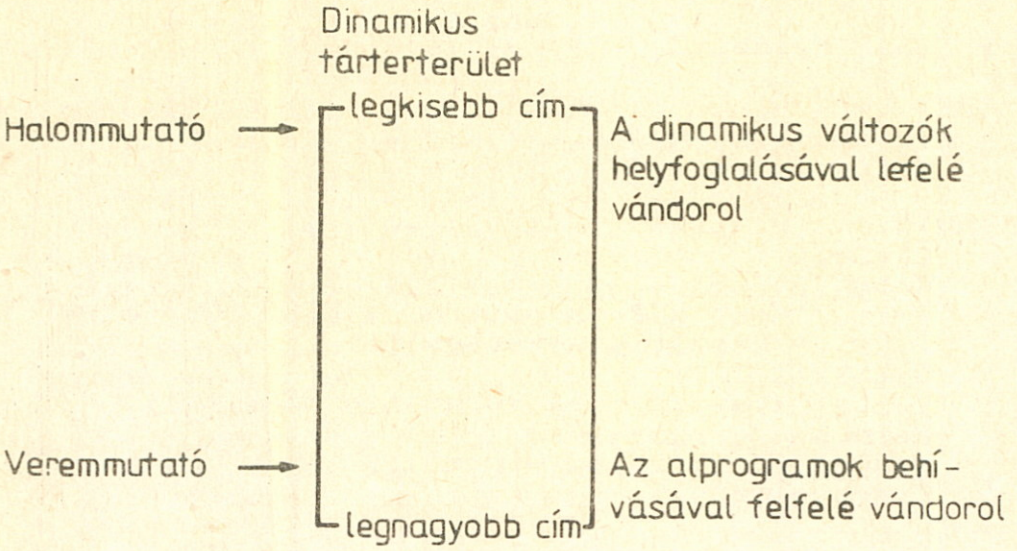
```
BEGIN

   ClrScr;
   GotoXY(1,1Ø);
   Write('Most az 1-es pontra, a LETREHOZAS-ra' +
         ' kerult a vezerles');
   GotoXY(1,12);
   Write('<return>-nel kerheti a MENU-t');
   GotoXY(1,24);
   Write('parameter: ',parl);
   ReadLn;
   Assign(main,'vezerlo.com');
   Execute(main);
END.
```

```
{ Feldolg2.pas }
{ Pr. I. 2.4/b }

PROGRAM Feldolg2;

VAR

   main : file;

BEGIN
   ClrScr;
   GotoXY(1,1Ø);
   Write('Most a 2-es pontra, a FELDOLGOZAS-ra' +
         ' kerult a vezerles');
   GotoXY(1,12);
   Write('<return>-nel kerheti a MENU-t');
   ReadLn;
   Assign(main,'vezerlo.com');
   Execute(main);
END.
```

```
{ Feldolg3.pas }
 { Pr. I. 2.4/c }

 PROGRAM Feldolg3;

VAR

   main : file;

BEGIN
   ClrScr;
   GotoXY(1,1Ø);
   Write('Most a 3-as pontra, a NYOMTATAS-ra' +
         'kerult a vezerles');
   GotoXY(1,12);
   Write('<return>-nel kerheti a MENU-t');
   ReadLn;
   Assign(main,'vezerlo.com');
   Execute(main);
END.
```

2.5. HALOM ÉS VEREM

A halom (Heap) és a verem (Stack) a dinamikus tárterület két része. Ezt a területet azért nevezik dinamikusnak, mert a lefoglalása az adatterülettel ellentétben nem a fordítás során, hanem a futtatáskor történik. A dinamikus tárterületnek a programfuttatáshoz szükséges minimális nagysága a fordítás után kizáródik, s rögtön utána a maximális dinamikus tárterület nagysága – amelyet a számítógép tárkapacitása határol be – is megjelenik.
Program behívásakor a *halommutató* a tárolóterület legkisebb címére, a *veremmutató* pedig a legnagyobb címre mutat. A program végrehajtása során ez a két mutató az igénybe vett tárolóterületnek megfelelően egymás felé vándorol. Ha találkozik a két mutató, akkor a program (Heap/Stack-Kollision) futási hibával megszakad. A következő ábra szemlélteti ezt a tárkezelést.



A verem: futtatás közben a dinamikus tárterület veremmutatóval kezelt részét használják ideiglenesen az alprogramok kifejezései, lokális változói, paraméterei és visszatérési címei. A sokszorosan egymásba ágyazott alprogramok – ami pl. a rekurzív hívásoknál gyakran előfordul – a dinamikus tárterület nagy részét lefoglalják. A lefoglalt tárterület a legfelső hívási szintre való visszatéréskor felszabadul.
A programozó számára nincs ésszerű lehetőség arra – de nincs is rá szükség –, hogy a veremkezelésbe aktívan belenyúljon.

A halom: a dinamikus tárterület halommutatóval kezelt részét használják ideiglenesen a dinamikus változók. Ezért, ha egy programban a *New* eljárással nagymennyiségű változóstruktúrának foglalunk helyet, akkor ezzel a dinamikus tárterület nagy részét lefoglaljuk.

Dinamikus tárkezelés: a halom kezelésére a *Turbo–Pascal*-ban az alábbi kifejezések definiáltak:

HeapPtr a halom első szabad helyére mutat;
New a halomba tesz egy meghatározott változót;
Dispose a *New* eljárással lefoglalt területet felszabadítja;
Mark a halommutató aktuális értékét tárolja;
Release a halommutatót a megadott értékre állítja;
GetMem tárterületet foglal le a halomban;
FreeMem a GetMem eljárással lefoglalt területet felszabadítja;
MemAvail a dinamikus tár még szabad területének a nagyságát adja.

HEAPPTR

A *HeapPtr* előre definiált mutatóváltozó, amely a halom első szabad helyére mutat. A mutatót csak a legnagyobb elővigyázatossággal lehet megváltoztatni, mozgatni.

NEW és DISPOSE

New eljárás: (paraméter: mutató)
a mutatóval meghatározott adatstruktúrát a halomba teszi s ezért a halommutató a veremmutató felé vándorol.
Dispose eljárás: (paraméter: mutató)
az utolsó *New* eljárással lefoglalt területet ismét felszabadítja. Amennyiben a felszabadítandó terület után a halomban új területeket foglaltunk már le, a *Dispose* által való felszabadítás nem lehetséges, hatástalan.

A *Standard–Pascal*-lal való kompatibilitás miatt a *Dispose* eljárást minden *Pascal* -fordítónak ismernie kell.

Példa:

```
{ Heap1.pas    }
{ Pr. I. 2.5.  }

TYPE t_rech = record
                Kulcs : integer;
                info  : string[255];
              end;
VAR p  : ^t_rech;

BEGIN
   New(p);
   ...
   Dispose(p);
```

Az eljárás alkalmazása megtalálható a II. rész 1.11. programpéldájában.

MARK és RELEASE

Mark eljárás: (paraméter: mutató)
a halommutató aktuális értékét tárolja a paraméterként megadott mutatóba.
Release eljárás: (paraméter: mutató)
a halommutatót a paraméter értékére állítja.
A *Mark* és a *Release* eljárás arra való, hogy segítségükkel a halomban szabad tárterületet állítsunk elő.

Példa:

```
{ Heap2.pas  }
{ Pr. I. 2.6.}

TYPE t_rech = record
                Kulcs  : integer;
                info   : string[255]
              end;
VAR p : ^t_rech;
    m : ^integer;

BEGIN
   mark(m);
   new(p);
   ...
   new(p);
   ...
   release(m);
```

Minden tárterület, amelyet a *Mark* hívás után foglal le a program a halomban, a *Release* eljárással ismét felszabadul.

GETMEM, FREEMEM és MEMAVAIL

GetMem eljárás: (paraméter: mutató
tárterület – integer)

a halomban egy összefüggő tárterületet foglalunk le, amely a mutatóváltozóval meghatározott helyen kezdődik és amelynek nagyságát blokkokban az integer számmal rögzítjük (1 blokk = 16 byte).

FreeMem eljárás: (paraméter: mutató
tárterület – integer)

GetMem által lefoglalt tárterületet lehet felszabadítani.

MemAvail függvény: (paraméter: nincs
függvénytípus: integer)

a függvény a halom összesen rendelkezésre álló területét mutatja blokkokban (1 blokk = 16 byte). A függvényeknek az alkalmazását a 2.8. alfejezetben (*DOS*-funkciók hívása) találjuk meg a $48-as és $49-es funkciók leírásánál.

2.6. KÖZVETLEN HOZZÁFÉRÉS A TÁRHOZ ÉS A REGISZTEREKHEZ

Az előző fejezetben megismertük, hogy a tár bizonyos részei az operációs rendszer meghatározott feladataihoz lefoglaltak. Ahhoz, hogy a tár ezen részeit hasznosítani tudjuk elengedhetetlen, hogy a lefoglalt területeket céltudatosan lekérdezhessük.
A *Turbo–Pascal*-ban erre a célra előre meghatározott függvények és változók állnak rendelkezésünkre

Seg:	változó címszegmens értéke
Ofs:	változó címofszet értéke
Addr:	a változó teljes tárcíme
Ptr:	tárcím számítása
Cseg:	a CS-regiszter tartalma
Dseg:	a DS-regiszter tartalma
Sseg:	az SS-regiszter tartalma
Absolute:	változók címzése
Mem:	a tárból 1 byte közvetlen kiolvasása
MemW:	a tárból 1 szó (2 byte) közvetlen kiolvasása

SEG és OFS

Seg függvény: (paraméter: tetszőleges típusú változó
függvénytípus: integer)
a paraméterként megadott változó címének a szegmensértékét adja.
Ofs függvény: (paraméter: tetszőleges típusú változó
függvénytípus: integer)
a paraméterként megadott változó címének az ofszet értékét adja.

Példa:

```
{ Cimkepz.pas }
{ Pr. I. 2.7. }

VAR puffer : string[10];
    es, dx : integer;

BEGIN
   es:=Seg(puffer);
   dx:=Ofs(puffer)+1;
```

Azoknál a *DOS*-funkció hívásoknál, amelyek karakterláncokat formálnak át, a karakterlánc címét egy regiszterpár kapja.
A *Seg* és *Ofs* segítségével ezek a címek kinyomozhatók. Egy karakterlánc ofszetjének meghatározásakor mindig ügyelnünk kell arra, hogy a karakterlánc címe az első byte-tal, amely a karakterlánc hosszát tartalmazza, vagy anélkül (mint a példánkban) kerül átadásra.
A fenti két funkcióval foglalkozik a 2.8. alfejezet számos példája, valamint 3.14. és a 3.16. programpéldák az I. részben.

ADDR

Addr függvény: (paraméter: tetszőleges típusú változó
függvénytípus: mutató)
a változó tárcímét adja, a bal oldali 16 bit a cím szegmensértékét, a jobb oldali 16 bit pedig az ofszetértékét tartalmazza.

Példa:

```
{ Address.pas }
{ Pr. I. 2.8. }

TYPE t_rek = record
                kulcs  : integer;
                info   : string[80];
             end;

VAR a  :  t_rek;
    p  : ^t_rek;

BEGIN
   p^:=Addr(a);
```

Az *Addr(a)* azonos eredményt ad, mint a *Ptr(seg(a), ofs(a))*.
Az *Addr* függvény használatát a 2.8. alfejezetben több példában bemutatjuk.

PTR

Ptr függvény: (paraméter: szegmens — integer
ofszet — integer
függvénytípus — mutató)

a két integer a paraméterből egy 32 bites mutatót készít, amely a bal oldali 16 bitben az első paramétert (szegmens), a jobb oldali 16 bitben a második paramétert (ofszet) tartalmazza.

Példa:

```
{ Pr. I. 2.9. }


TYPE sor = string[80];

VAR     x : sor;
        p : ^sor;

BEGIN
   p:=Ptr(Cseg,$20);
   Writeln(p^);
END.
```

A függvénynek egy alkalmazása megtalálható a 2.8. alfejezetben a S35-ös *DOS*-funkció hívás példában.

CSEG, DSEG és SSEG

Előre definiált integer típusú változók és a *CS, DS,* valamint az *SS* regiszterek értékét tartalmazzák.

Példa:

```
{ Pr. I.2.10. }

VAR i : integer;
BEGIN
    i := Cseg;
END.
```

Az előre definiált változóknak a használatakor arra kell ügyelni, hogy ezek különböző értékeket tartalmaznak aszerint, hogy a program lefordított-e és COM file-ként vagy *TURBO*-n belül lesz végrehajtva. A *TURBO*-n belül a *CS* a *TURBO.COM* értékét tartalmazza és ez általában *nem* a kívánt eredmény.
Ezeknek az előre definiált változóknak a használatával foglalkozik a 2.8. alfejezetben a $25-ös és $4B funkció leírása, valamint a 2.3. programpélda a II. részben.

ABSOLUTE

Az *Absolute* változójellemző segítségével egy változó deklarálásakor a helye meghatározható a tárban.
A meghatározás módszere lehet
abszolút: rögzített tárcímre és
relatív: egy másik, már deklarált változóhoz viszonyítva.

Abszolút címzés:
a módszer mindenekelőtt akkor szükséges, ha a tárnak egy előre meghatározott részéhez, pl. a képernyőkimenet tárterületéhez akarunk hozzáférni.

Példa:

VAR *x: integer absolute* $400:$0,

Az egész típusú változót a munkatárban a $E400 címre teszi le.
A helymeghatározás történhet egy előre definiált regeiszterterületre – *Cseg, Dseg* vagy $*seg* – is.

Példa:

VAR *x: string* [10] *absolute* Cseg: $20,

Relatív címzés:
a módszerrel megkerülhető a *Standard–Pascal* igen szigorú típusátalakítási elve.

Példa:

```
{ Pr. I. 2.11. }

VAR betu : char;
    szam : byte absolute betu;

{ Az ABC kinyomtatasa }

BEGIN
   for szam:=65 to 90 do write(betu);
END.
```

A *szam* változóval ugyanazt a tárterületet kérdezzük le, mint a *betu* változóval.
Az alkalmazásra a 3.3. program ad példát.

MEM ÉS MEMW

A *Mem* segítségével közvetlenül egy byte és a *MemW* segítségével közvetlenül egy szó (2 byte) kérdezhető le a tárban. A tárcímet szegmens: ofszet formában kell megadni.

Példa:

```
{ Pr. I. 2.12. }

VAR b : byte;
    i : integer;
BEGIN
    b:=Mem[Dseg:$8Ø];
    Mem[Dseg:$8Ø]:=Ø;
    i:=MemW[$Ø:$FE];
    MemW[Ø:Ø]:=Ø;
END.
```

A példa első utasítására a DS:$80 tárcímen levő byte tartalma a *b* változóba kerül, a harmadik utasításra a $0:$FE tárcímen levő szó (két byte) tartalma kerül az *i* változóba.
A második utasításra a tárban egy byte értéke, míg a negyedik utasításra a tárban két byte értéke változik meg, nulla lesz.

Nyilvánvaló, hogy a tárból, ill. a tárba az értékadás a *Mem* útján csak egy byte hosszúságú változókkal (*Bybe* vagy *Char* típusúval), ill. a *MemW* alkalmazásakor csak két byte hosszúságú változókkal (*Integer* vagy *Word* típusúval) lehetséges.
És még egy **példa:**

```
{ Pr. I. 2.13. }

VAR x : string[1Ø];
    l : byte;

BEGIN
    l:=Mem[seg(x):ofs(x)]
END.
```

Ahogy ezt a 3.1. alfejezetben még részletesebben ismertetjük majd, egy karakterlánc első byte-jában a karakterlánc hossza áll; példánkban *l* tehát az *x* hossza (a karakterek darabszáma).

INTR

A *DOS* a jól használható rutinok egész sorát bocsátja a programozó rendelkezésére. Ezeket a rutinokat megszakításoknak nevezik és a megszakításszámmal azonosítják. A *Turbo–Pascal* számára a *DOS*-megszakítások hozzáférhetőségét az *INTR* eljárás teszi lehetővé.

Intr eljárás: (paraméter: megszakításszám – integer
regiszter – t_reg)

t_reg definiálása: **TYPE**

```
t_reg = rekord
ax, bx, cx, dx,
bp, si, di,
ds, es, flags:, integer;
end;
```

A *DOS*-megszakítások olyan fontos és annyi lehetőséget nyújtó eljárások, hogy ezzel a témával két alfejezet is – az 1.5. és a 2.8. – foglalkozik.

MSDOS

A legjelentősebb az eddig ismertetett *DOS*-megszakítások közül a $21-es, amellyel ismét egy egész sor *DOS*-rutin válik hívhatóvá. A $21-es megszakítást a *DOS-funkciók hívásának* nevezik. A hívható funkciók számozva vannak. A hívásra kerülő funkció száma kerül az AH regiszterbe.

Ezek a *DOS* funkció híváskor az *Intr($21, regiszter)* eljárás segítségével, vagy a speciálisan e célra rendelkezésre álló *MsDos* eljárással érhetők el.

MSDOS eljárás: (paraméter: regiszter – t_reg)
(a t_reg definícióját l. előbb)

A *DOS*-funkcióhívások részletes leírása az 1.5. és a 2.8 alfejezetben találhatók.

A *DOS*-nak a felhasználói programhoz való kapcsolódási lehetőségeinek általános leírását az 1.5. alfejezetben már közöltük. Itt példákkal mutatjuk be, hogyan használhatók ezek a *Turbo–Pascal*-ban.

2.8.1. DOS-megszakítások

$20 **Program vége** (Terminate Program)
HALT;

$21 **DOS-funkcióhívás** (Function Request)
MSDOS(r);
vagy

INTR($21,r);

$27 **Programvég foglalt tárterülettel** (Terminate But Stay Resident)
DOS.20 változattól a $31-es azonos funkciójú

2.8.2. DOS-funkciók

Definíciók (a további példákhoz):

```
{ Pr. I. 2.14. }

TYPE

   regs = record {8086/88 regiszterei}
      ax, bx, cx, dx,
      bp, si, di,
      ds, es,
      flags
        : integer;
   end;


{ Pr. I. 2.14/a }


t_nev  = array[1..8] of char;
t_ext  = array[1..3] of char;

t_fcb  = record      {File Conrol Block}
  meghajto_sz       : byte;
  file_nev          : t_nev;
  kiterjesztes      : t_ext;
  akt_blokk,
  rekord_hossz,
  file_meret_f,     { felso byte }
  file_meret_a,     { also byte  }
  datum,
  ido,
  tartalek_1,
  tartalek_2,
  tartalek_3,
```

```
    tartalek_4        : integer;
    akt_rekord        : byte;
    rel_rekord_f,                    { a felso byte }
    rel_rekord_a      : integer;     { az also byte }
  end;

  VAR
    ch      : char;    {Be-/Kivitel}
    r       : regs;
```

A funkciók hívása

$00 **Program vége** (Terminate Program)
Halt:

$01 **Bevitel a billentyűzetről és megjelenítés a képernyőn** (Read Keyboard and Echo)

```
{ Pr. I. 2.15. }

r.ax:=$0100;
MsDos(r);    {Control-c megszakitja}
ch:=Chr(Lo(r.ax));
```

$02 **Kivitel a képernyőre** (Display Character)

```
{ Pr. I. 2:16. }

r.ax:=$0200;
r.dx:=Ord(ch);       {Control-C megszakitja}
MsDos(r);
```

$03 **Bevitel választható egységről** (Auxiliary Input)

```
{ Pr. I. 2.17. }

r.ax:=$0300;
MsDos(r);            {Control-C megszakitja}
ch:=Chr(Lo(r.ax));
```

$04 **Kivitel választható egységre** (Auxiliary Output)

```
{ Pr. I 2.18. }

r.ax:=$0400;
r.dx:=Ord(ch);       {Control-C megszakitja}
MsDos(r);
```

$05 Karakternyomtatás (Print Character – LST Device)

```
{ pr. I. 2.19. }


r.ax:=$Ø5ØØ;
r.dx:=Ord(ch);      {Control-C megszakitja}
MsDos(r);
```

$06 Közvetlen Be/Kivitel (Direct Console I/O) – várakozás nélkül

```
{ Pr. I. 2.2Ø. }


r.ax:=$Ø6ØØ;        {ch=$FF eseten olvasasi kiserlet}
r.dx:=Ord(ch);      {egyebkent ch lesz kiadva}
MsDos(r);           {Control-C nem szakitja meg}
if Zero_Flag
 then
   ch:=Chr(Lo(r.dx)); {kulonben Lo(r.ax)=Ø}
```

$07 Közvetlen bevitel a billentyűzetről (Direct Console Input)
– várakozással, megjelenítés nélkül

```
{ Pr. I. 2.21. }


r.ax:=$Ø7ØØ;
MsDos(r);       {Control-C nem szakitja meg}
ch:=Chr(Lo(r.ax));
```

vagy

```
Read(kbd,ch);  {TURBO-ban, minden trukk nelkul}
```

$08 Olvasás a billentyűzetről (Read Keyboard)
– megjelenítés nélkül

```
{ Pr. I. 2.22.}


r.ax:=$Ø8ØØ;
MsDos(r);       {Control-C megszakitja}
ch:=Chr(Lo(r.ax));
```

$09 Karakterlánc írása a képernyőre (Display String)

```
{ Pr. I. 2.23. }


sor:='String kiiras $';
r.ax:=$Ø9ØØ;
r.ds:=Seg(sor);
r.dx:=Ofs(sor)+1;  {+1 Hosszbyte!}
MsDos(r);
```

vagy

```
Write(str:length(sor)-1);

        {TURBO-ban, minden trukk nelkul}
```

$0A Pufferelt bevitel a billentyűzetről (Buffered Keyboard Input)

```
{ Pr. I. 2.24. }


TYPE
  t_puf = record
    puf_max: byte;
    sor    : string[30];
    end;

VAR
  puf  : t_puf;

BEGIN
r.ax:=$0A00;
r.ds:=Seg(puf);
r.dx:=Ofs(puf);
puf.puf_max:=30;
MsDos(r);          {Control-C megszakitja}
```

vagy

```
Buflen:=SizeOf(sor);
Read(sor);

                        {TURBO-ban, minden trukk nelkul}
```

$0B Billentyűzet állapotának ellenőrzése (Check Keyboard Status)

```
{ Pr. I. 2.25. }


r.ax:=$0B00;
MsDos(r);
if Lo(r.ax)=255
 then
   Write('Karakter a pufferben');
```

vagy

```
if Keypressed    {TURBO-ban, minden trukk nelkul}
 then
 ;     {...}
```

$0C Billentyűzetpuffer törlése és olvasás a billentyűzetről (Flush Buffer, Read Keyboard)

```
{ Pr. I. 2.26. }


VAR
  sub_funktion : integer;    {1,6,7,8 vagy $A}

r.ax:=$0C00 or sub_funktion; {a fuggveny sorszama}
MsDos(r);
if Lo(r.ax) = 0
 then
   Write('A billentyu puffer ures, nincs tovabb');
```

$0D Az adatpuffer kiürítődik és szabad állapotjelzést kap (Disk Reset)

```
{ Pr. I. 2.27. }


{ FIGYELEM! A file-tartalomjegyzek
            nem kerul aktualizalasra }

r.ax:=$ØDØØ;
MsDos(r);
```

$0E Az alapértelmezett (default) meghajtó kiválasztása (Select Disk)

```
{ Pr. I. 2.28. }


VAR

  megh : char;  {Meghajto 'A'..'P'}

BEGIN
r.ax:=$ØEØØ;
r.dx:=Ord(megh)-Ord('A');  {A=Ø, B=1, C=2,....}
MsDos(r);
WriteLn('A meghajtok darabszama:',Lo(r.ax));
```

$0F File megnyitása (Open File)

```
{ Pr. I. 2.29. }


VAR

  fcb : t_fcb;
  {Az OPEN elott $29-es (file-nev elemzes) hivassal
   egy karakterlancot kell az FCB-be kuldeni}

BEGIN
r.ax:=$ØFØØ;
r.ds:=Seg(fcb);  {nem lehet nyitva}
r.dx:=Ofs(fcb);
MsDos(r);
if Lo(r.ax)<>Ø
then
  Write('A file nem letezik!');

  {Az OPEN utan az FCB aktualizalasra kerult}
```

vagy

```
Assign(FilVar,Filenev);

                    {TURBO-ban, minden trukk nelkul}
```

$10 File lezárása és a tartalomjegyzék aktualizálása (Close File)

```
{ Pr. I. 2.3Ø. }


r.ax:=$1ØØØ;
r.ds:=Seg(fcb);   {Megnyitottnak kell lennie}
r.dx:=Ofs(fcb);
```

```
MsDos(r);
if Lo(r.ax)<>Ø
 then
   Write('a file nem letezik!');
```

vagy

```
close(FilVar);    {TURBO-ban, minden trukk nelkul}
```

$11 Első file-név keresése (Search for First Entry)

```
{ Pr. I. 2.31. }


TYPE
  t_bovitett_fcb = record
    flag,
    tartalek_1        : byte;
    tartalek_2,
    tartalek_3        : integer;
    attribute          : byte;
    fcb               : t_fcb;
  end;
VAR
  bovitett_fcb        : t_bovitett_fcb;

{ Elobb a $1A, majd a $29 -es DOS funkciot meg kell
  hivni ! }

r.ax:=$11ØØ;
r.ds:=Seg(bovitett_fcb);
r.dx:=Ofs(bovitett_fcb);
     {FCB nem lehet nyitva!}
bovitett_fcb.flag:=$FF;  {file attribute kereseshez:
                  $1Ø=directory
                  $Ø4=system
                  $Ø2=hidden
                  $Ø8=(csak) vol_label,kulonben Ø}
bovitett_fcb.attribute:=$16;
MsDos(r);
if Lo(r.ax) = Ø
 then
   Write( 'File (',
          bovitett_fcb.fcb.file_nev,' ',
          bovitett_fcb.fcb.kiterjesztes,
          ') megtalalva')
 else                                  {Lo(r.ax) = $FF}
   Write('Nem talalt file-t');
```

$12 További file-név keresése (Search for Next Entry)

```
{ Pr. I. 2.32. }


{ A $11 es $12-es DOS funkciohivasok kozott
  semmilyen mas I/O funkciohivas nem lehet.
  A DTA-t hivasonkent ujra fel kell epiteni }

r.ax:=$12ØØ;
r.ds:=Seg(bovitett_fcb);
r.dx:=Ofs(bovitett_fcb);
     {Az FCB-t csak a DOS $11-es hasznalhatta}
MsDos(r);
if Lo(r.ax) = Ø
```

```
then
 Write( 'File (',
        bovitett_fcb.fcb.file_nev,'.',
        bovitett_fcb.fcb.kiterjesztes,
        ') megtalalva')
else   {Lo(r.ax) = $FF}
  Write('Nincs tobb file');
```

$13 File-név törlése a tartalomjegyzékből (Delete File)

```
{ Pr. I. 2.33. }

{ Elobb $29-es (file-nev elemzes) funkcioval
  a file-nevet meg kell adni. A  *.* megadas
  ????????.??? file_nevve konvertalodik. }

r.ax:=$13ØØ;
r.ds:=Seg(fcb);
r.dx:=Ofs(fcb);
    {FCB nem lehet nyitva!}
MsDos(r);
if Lo(r.ax) = Ø
  then
    Write('A torles sikeres')
  else   {Lo(r.ax) = $FF}
    Write('Nem letezo file(-ok)');
```

$14 Soros olvasás (Sequential Read)

```
{ Pr. I. 2.34. }



r.ax:=$14ØØ;
r.ds:=Seg(fcb);
r.dx:=Ofs(fcb);
     {FCB-nek nyitva kell lennie!}
MsDos(r);
case Lo(r.ax) of
 Ø : {az aktualis blokk/record beolvasasra kerult};
 1 : {EOF - a file vegenek elerese};
 2 : {DTA terulet tul kicsi};
 3 : {EOF -  a rekordterulet reszben fel van toltve};
end;
```

vagy

```
{reset(FilVar);}
Read(FilVar,rekord);

                          {TURBO-ban, minden trukk nelkul}
```

$15 Soros írás (Sequential Write)

```
{ Pr. I. 2.35. }


{ Elobb $1A hivassal DTA-t a rekord teruletre kell
  allitani. }

r.ax:=$15ØØ;
r.ds:=Seg(fcb);
r.dx:=Ofs(fcb);
```

```
    {FCB-nek nyitva kell lennie!}
MsDos(r);
case Lo(r.ax) of
 Ø : {az aktualis blokk/record kiirasra kerult};
 1 : {lemez tele van};
 2 : {DTA terulet tul kicsi};
end;
```

vagy

```
{Reset(FilVar);}
Write(FilVar,rekord);

                    {TURBO-ban, minden trukk nelkul}
```

$16 File létrehozása (Create File)

```
{ Pr. I. 2.36. }


{Elobb a $29-est (file-nev elemzes) kell meghivni,
 vagy a file-nev adott az FCB-ben}

r.ax:=$16ØØ;
r.ds:=Seg(fcb);
r.dx:=Ofs(fcb);
    {FCB nem lehet megnyitva!}
MsDos(r);
if Lo(r.ax) = Ø
 then
   Write('A file elkeszult')
 else   {Lo(r.ax) = $FF}
   Write('Nincs hely a file-tartalomjegyzekben');
```

vagy

```
Assign(FilVar,filenev);
Rewrite(FilVar);
                {TURBO -ban, minden trukk nelkul}
```

$17 File átnevezése (Rename File)

```
{ Pr. I. 2.37. }


TYPE

  t_mod_fcb = record
    meghajto      : byte;
    file_nev      : t_nev;
    kiterjesztes  : t_ext;
    maradek_1,
    maradek_2,
    maradek_3     : integer;
    uj_nev        : t_nev;
    uj_kit        : t_ext;
    maradek_4     : array[1..8] of byte;
              end;
```

```
VAR

 mod_fcb          : t_mod_fcb;

BEGIN
mod_fcb.meghajto      :=Ø;
mod_fcb.file_nev      :='????????';
mod_fcb.kiterjesztes  :='DTA';
mod_fcb.uj_nev        :='????????';
mod_fcb.uj_kit        :='DTA';

r.ax:=$17ØØ;
r.ds:=Seg(mod_fcb);
r.dx:=Ofs(mod_fcb);
MsDos(r);
if Lo(r.ax) = Ø
 then
   Write('Az atnevezes sikeres')
 else   {Lo(r.ax) = $FF}
   Write('A file-t nem talalja',
         'vagy az uj nev mar letezik');
```

vagy

```
Assign(FilVar,filenev);
Rename(FilVar,Uj_filenev);

                        {TURBO -ban, minden trukk nelkul}
```

$19 A meghajtó lekérdezése (Current Disk)

```
{ Pr. I. 2.38. }


r.ax:=$19ØØ;
MsDos(r);
Writeln('Az aktualis meghajto',
        Chr(Ord('A')+Lo(r.ax)),':');
```

$1A A lemezátviteli cím beállítása (Set Disk Transfer Addres)

```
{ Pr. I. 2.39. }


{ Az alapertelmezes szerin a DTA a Cseg:$8Ø-on van }

r.ax:=$1AØØ;
r.ds:=Seg(fcb);
r.dx:=Ofs(fcb);
MsDos(r);
```

$21 Rekord közvetlen olvasása (Random Read)

```
{ Pr. I. 2.4Ø. }


r.ax:=$21ØØ;
r.ds:=Seg(fcb);
r.dx:=Ofs(fcb);
```

```
        {FCB-nek nyitva kell lennie!}
fcb.rel_rekord_a:=rekord_sorszam;
MsDos(r);
case Lo(r.ax) of
Ø : {rendben};
1 : {EOF, file-veg elerese};
2 : {a DTA tul kicsi};
3 : {EOF, a rekordterulet reszben feltoltve},
end;
```

vagy

```
Seek(FilVar,rekord_sorszam);
Read(FilVar,rekord);

                        {TURBO -ban, minden trukk nelkul}
```

$22 Rekord közvetlen írása (Random Write)

```
{ Pr. I. 2.41. }

r.ax:=$21ØØ;
r.ds:=Seg(fcb);
r.dx:=Ofs(fcb);
     {FCB-nek nyitva kell lennie!}
fcb.rel_rekord_a:=rekord_sorszam;
MsDos(r);
case Lo(r.ax) of
Ø : {rendben};
1 : {lemez tele van};
2 : {DTA tul kicsi};
end;
```

vagy

```
Seek(FilVar,rekord_sorszam);
Write(FilVar,rekord);

                        {TURBO -ban, minden trukk nelkul}
```

$23 A file méretének lekérdezése (File Size)

```
{ Pr. I. 2.42. }

r.ax:=$23ØØ;
r.ds:=Seg(fcb);
r.dx:=Ofs(fcb);
     {FCB nem lehet megnyitva!}
fcb.rekord_hossz:=Rekord_hossz;
MsDos(r);
if Lo(r.ax) = Ø
 then
   Write('A file tartalmaz',
         fcb.rel_rekord_a,'rekordot')
 else     {Lo(r.ax) = $FF}
   Write('A file nincs a',
         'file-tartalomjegyzekben');
```

vagy

```
Write('A file tartalmaz',
      FileSize(FilVar),
      'rekordot');

                        {TURBO -ban, minden trukk nelkul}
```

$24 **A relatív rekord nevű mező beállítása** (Set Relative Record)
– az aktuális blokk és rekord nevű mezők tartalma alapján

```
{ Pr. I. 2.43. }


r.ax:=$24ØØ;
r.ds:=Seg(fcb);
r.dx:=Ofs(fcb);
      {FCB-nek nyitva kell lennie!}
fcb.akt_blokk :=rekord_sorszam div 128;
fcb.akt_rekord:=rekord_sorszam mod 128;
MsDos(r);
```

vagy

```
Seek(FilVar,Rekord_sorszam);
                        {TURBO -ban, minden trukk nelkul}
```

$25 **A megszakítási rutincím beállítása** (Set Vector)
– az aktuális program számára

```
{ Pr. I. 2.44. }


r.ax:=$25ØØ or Interrupt_Sorszam;
r.ds:=Cseg;
r.dx:=Ofs(Interrupt eljaras);
MsDos(r);
```

$27 **Közvetlen blokkolvasás** (Random Block Read)

```
{ Pr. I. 2.45. }


r.ax:=$27ØØ;
r.ds:=Seg(fcb);
r.dx:=Ofs(fcb);
      {FCB-nek nyitva kell lennie!}
r.cx:=blokk_szam;
fcb.rekord_hossz:=blokk_meret;
fcb.rel_rekord_a:=blokk_sorszam;
MsDos(r);
case Lo(r.ax) of
 Ø : {rendben};
 1 : {EOF, a file vege}
      Write(r.cx,'blokk beolvasva');
 2 : {DTA tul kicsi};
 3 : {EOF, a terulet reszben feltoltve};
end;
```

vagy

```
seek(FilVar,blokk_sorszam);
Blockread(FilVar,terulet,
        blokk_szam,
        tenylegesen_beolvasott);
                {TURBO -ban, minden trukk nelkul}
```

$28 Közvetlen blokkírás (Random Block Write)

```
{ Pr. I. 2.46. }


r.ax:=$28ØØ;
r.ds:=Seg(fcb);
r.dx:=Ofs(fcb);
      {FCB-nek nyitva kell lennie!}
r.cx:=blokk_szam;

{ Ha blokk_szam = Ø, akkor a file_meret mezo
  aktualizalodik a rel_rekord mezo alapjan   }

fcb.rekord_hossz:=blokk_meret;
fcb.rel_rekord_a:=blokk_sorszam;
MsDos(r);
case Lo(r.ax) of
 Ø : {rendben};
 1 : {a lemez tele van};
 2 : {a DTA tul kicsi};
end;
```

vagy

```
Seek(FilVar,blokk_sorszam);
Blockwrite(FilVar,terulet,
          blokk_szam);
                {TURBO -ban, minden trukk nelkul}
```

$29 File-név elemzés (Parse File Name)

– az FCB feltöltése egy karakterláncból álló ellenőrzött file-névvel.

Elemzésvezérlés:

7654 3210 a vezérlőbitek;
———— ———0 az elemzés az első elválasztójelnél befejeződik;
———— ———1 a vezető-elválasztójelen átmegy;
———— ——0— az alapértelmezett meghajtóhoz fordul, ha nincs meghajtó megadva;
———— ——1— a meghajtó kijelölése;
———— —0—— a file-nevet a FCB-ben 8 szóközzel tölti fel, ha nem adtunk meg file-nevet;
———— —1—— a file-név megadása;
———— 0——— a kiterjesztést a FCB-ben 3 szóközzel tölti fel, ha nem adtunk meg kiterjesztést;
———— 1——— a kiterjesztés megadása.

4-től 7-ig terjedő biteket nem veszi figyelembe.

A szeparátorok: ., ,, :, ;, +, /, <, >, =, ”, [,], *backslash, szóköz, vagy*-jel

a * a file-névben és a kiterjesztésben a mező egyéb karakterekkel fel nem töltött részét ? -lel tölti fel.

```
{ Pr. I. 2.47. }

filenev:='C:ALKAMAR.GDS';
r.ax:=$29ØF;  {ellenorzesvezerles ----1111 = $ØF}
r.ds:=Seg(filenev);
r.si:=Ofs(filenev)+1;
r.es:=Seg(fcb);
r.di:=Ofs(fcb);
      {az FCB nem lehet nyitva!}
MsDos(r);
case Lo(r.ax) of
 Ø    : {nincs helyettesito karakter};
 1    : {helyettesito karaktert talalt};
 $FF  : {hibas meghajto};
end;
```

$2A A rendszerdátum lekérdezése (Get Date)

```
{ Pr. I. 2.48. }

r.ax:=$2AØØ;
MsDos(r);
ev:=r.cx;              {198Ø .. 2Ø99}
ho:=Hi(r.dx);          {1 .. 12}
nap:=Lo(r.dx);         {1 .. 31}
napok:=Lo(r.ax);       {Ø=vasarnap, ... ,
                        6=szombat}
```

$2B A rendszerdátum beállítása (Set Date)

```
{ Pr. I. 2.49. }

r.ax:=$2BØØ;
r.cx:=ev;              {198Ø .. 2Ø99}
r.dx:=(ho shl 8) or nap;
                       {1 .. 12}
                       {1 .. 31}
MsDos(r);
if Lo(r.ax) = Ø
 then
   Writeln('Datum rendben')
 else { $FF }
   Writeln('Ervenytelen datum');
```

$2C A rendszeridő lekérdezése (Get Time)

```
{ Pr. I. 2.5Ø.}

r.ax:=$2CØØ;
MsDos(r);
ora:=Hi(r.cx);             {Ø .. 23}
perc:=Lo(r.cx);            {Ø .. 59}
masodperc:=Hi(r.dx);       {Ø .. 59}
szazadmp :=Lo(r.dx);       {Ø .. 99)
```

$2D A rendszeridő beállítása (Set Time)

```
{ Pr. I. 2.51. }

r.ax:=$2DØØ;
r.cx:=(ora shl 8) or perc;
                  {Ø .. 23, Ø .. 59}
r.dx:=(masodperc shl 8) or szazadmp;
                  {Ø .. 59, Ø .. 99}
MsDos(r);
if Lo(r.ax) = Ø
 then
   Writeln('Idomegadas rendben')
 else          { $FF }
   Writeln('Ervenytelen idomegadas');
```

$2E Írásellenőrzés-kapcsoló beállítása vagy törlése (Set/Reset Verify Flag)

```
{ Pr. I. 2.52. }

r.ax:=$2EØØ;  {nincs irasellenorzes
               (quick and dirty)}
r.ax:=$2EØ1;  {irasellenorzessel}
MsDos(r);
```

$2F A lemezátviteli cím lekérdezése (Get Disk Transfer Address)

```
{ Pr. I. 2.53. }

r.ax:=$2FØØ;
MsDos(r);
Disk_Transfer_Address:=
        Ptr(r.es,r.bx);
```

$30 DOS-verziószám lekérdezése (Get DOS Version Number)

```
{ Pr. I. 2.54. }

r.ax:=$3ØØØ;
MsDos(r);
verzio    :=Hi(r.ax);
alverzio  :=Lo(r.ax);
```

$31 A program befejezése a programkód megőrzésével (Keep Process) — a programvég elérése után is megmarad a lefoglalt terület és annak a tartalma

```
{ Pr. I. 2.55. }

r.ax:=$31ØØ or Exit_Code;
r.dx:=lefoglalt_terulet;
      {paragrafusban megadva, azaz kar_szam div 16}
MsDos(r);
```

$33 ^C ellenőrzés (Controll–C Check) – normális esetben csak a $01-es, a $08-as és a SOC DOS-funkcióknál kerül ellenőrzésre, hogy volt-e futás közben ^C-vel megszakítás.
Ezzel a funkcióval elérhető, hogy a többi *DOS*-funkció is megszakítható legyen.
Azoknál a programoknál, amelyek a $06-os és a $07-es funkciókkal a ^C-t felismemék, a $07-es ellenőrzést ki kell kapcsolni.

```
{ Pr. I. 2.56. }

if kerdes
 then
   r.ax:=$3300    {00 = kerdes}
 else begin
   r.ax:=$3301;   {01 = beallitas}
   if bekapcsolas
    then
      r.dx:=1
    else
      r.dx:=0;
end;
MsDos(r);

if kerdes
 then if r.dx = 1
   then Writeln('ellenorzi')
   else Writeln('nem ellenorzi');
```

$35 Megszakítási rutincím lekérdezése (Get Interrupt Vector)

```
{ Pr. I. 2.57. }

r.ax:=$3500 or megszakitas_sorsz;
                    {0 .. $FF}
MsDos(r);
intr_cim:=Ptr(r.es,r.bx);
```

$36 A lemez szabad területének lekérdezése (Get Disk Free Space)

```
{ Pr. I. 2.58. }

r.ax:=$3600;
r.dl:=meghajto;
MsDos(r);
if r.ax=$FFFF
 then
   Writeln('Rossz meghajto azonosito')
 else begin
   Free_Clusters:=r.bx;
   Cluster_per_disk:=r.dx;
   Bytes_per_Sector:=r.cx;
   Sectors_per_Cluster:=r.ax;
   Free:=Free_Clusters * Sectors_per_Cluster
                       * Bytes_per_Sector;
     end;
```

$38 A forgalmazás helyétől függő információk behozása (Return Country Dependent Information)

```
{ Pr. I. 2.59. }

TYPE

  t_asciz_2=array[1 .. 2] of char;
  t_asciz_5=array[1 .. 5] of char;
VAR
  CDI_puf : record
     datum_ido          : integer;
     currency_symbol    : t_asciz_5;
     n1000_sep,
     dec_sep     : t_asciz_2;
     tartalek    : string[24];
           end;

BEGIN
  FillChar(CDI_puf,SizeOf(CDI_puf),0);
  r.ax:=$3800;
  r.ds:=Seg(CDI_puf);
  r.dx:=Seg(CDI_puf);
  MsDos(r);
  writeln(r.ax);  {=2 az orszag nem talahato}
  with CDI_puf do
  Writeln (' dt:',datum_ido,
                          {0 - USA ... h:m:s m/d/y}
                          {1 - Eu .... h:m:s d/m/y}
                          {2 - Japan . y/m/d h:m:s}
           ' cs:', currency_symbol,
           ' ls:', n1000_sep,
           ' ds:', dec_sep;
END.
```

$39 Alkönyvtár létrehozása (Create Sub-Directory)

```
{ Pr. I. 2.60. }


r.ax:=$3900;
ut:=ut+#0;  {ASCIIZ keszites}
r.ds:=Seg(ut);
r.dx:=Ofs(ut)+1;
MsDos(r);
if (r.flags and $0001) = 1 {atvitelkapcsolo beallitva}
 then case r.ax of
   3 : Writeln('Helytelen keresesi ut');
         {a keresesi ut vagy a konyvtarnev
          szintaktikusan hibas}
   5 : Writeln('Igeny elutasitva');
         {a konyvtar mar letezik vagy
          nincs tobb hely}
end;
```

vagy

```
MkDir(ut);  {TURBO -ban, minden trukk nelkul}
```

$3A Alkönyvtár törlése (Remove a SUB-Directory Entry)

```
{ Pr. I. 2.61. }

r.ax:=$3AØØ;
ut:=ut+#Ø;      {ASCIIZ keszitese}
r.ds:=Seg(ut);
r.dx:=Ofs(ut)+1;
MsDos(r);
if (r.flags and $ØØØ1) = 1
          {atvitelkapcsolo beallitva}
  then case r.ax of
    3 : Writeln('Helytelen keresesi ut');
          {a keresesi ut vagy a konyvtarnev
           szintaktikusan hibas}
    5 : Writeln('Igeny elutasitva');
          {A konyvtar nem torolheto,nem ures}
   16 : Writeln('Az aktualis konyvtar torleset kerte)
end;
```

vagy

```
RmDir(ut);  {TURBO -ban, minden trukk nelkul}
```

$3B Alkönyvtárváltás (Change Current SUB-Directory)

```
{ Pr. I. 2.62. }

r.ax:=$3BØØ;
ut:=ut+#Ø;      {ASCIIZ keszitese}
r.ds:=Seg(ut);
r.dx:=Ofs(ut)+1;
MsDos(r);
if (r.flags and $ØØØ1) = 1
        {atvitelkapcsolo beallitva}
 then case r.ax of
   3 : Writeln('Helytelen keresesi ut');
         {a keresesi ut vagy a konyvtarnev
          szintaktikusan hibas}
end;
```

vagy

```
ChDir(ut);  {TURBO -ban, minden trukk nelkul}
```

$3C File-létrehozás (Create a File)

```
{ Pr. I. 2.63. }
r.ax:=$3CØØ;
ut:=ut+#Ø;         {ASCIIZ keszitese}
r.cx:=Attribute; {R/W,Hidden,System}
r.ds:=Seg(ut);
r.dx:=Ofs(ut)+1;
MsDos(r);
if (r.flags and $ØØØ1) = 1
      {atvitelkapcsolo beallitva}
  then case r.ax of
```

```
      3 : Writeln('Helytelen keresesi ut vagy filenev');
          {A keresesi ut vagy a file-nev
           szintaktikusan hibas}
      4 : Writeln('Tul sok file van megnyitva');
      5 : Writeln('Igeny elutasitva');
          {A file- jellemzo hibas vagy a file
           mar letezik, vagy nincs tobb hely}
  end;
```

vagy

```
  Assign(FilVar,Ut+Filenev);
  Rewrite(FilVar);        {TURBO -ban, minden trukk nelkul}
```

$3D File-megnyitás (Open a File)

```
{ Pr. I. 2.64. }

r.ax:=$3DØØ or tipus;
    {tipus = Ø csak olvasas
             1 csak iras
             2 olvasas es iras}
ut:=ut+#Ø;              {ASCIIZ keszitese}
r.cx:=Attribute;  {R/W, Hidden, System}
r.ds:=Seg(ut);
r.dx:=Ofs(ut)+1;
MsDos(r);
if (r.flags and $ØØØ1) = 1
     {atvitelkapcsolo beallitva}
  then case r.ax of
     2 : Writeln('A file nem talalhato');
     3 : Writeln('Helytelen keresesi ut');
          {A keresesi ut vagy a file-nev hibas'}
     4 : Writeln('Tul sok file van nyitva');
     5 : Writeln('Igeny elutasitva');
          {A file-jellemzo hibas, vagy a file
           meg nem letezik, vagy nincs tobb hely}
    12 : Writeln('tipus nem Ø..2');
    end
  else
   handle:=r.ax;
     {A file feldolgozasi sorszama}
```

vagy

```
Assign(FilVar,Ut+Filenev);
Rewrite(FilVar);      {TURBO -ban, minden trukk nelkul}
```

$3E A file-feldolgozás lezárása (Close a File Handle)

```
{ Pr. I. 2.65. }
r.ax:=$3DØØ or tipus;
r.bx:=handle;
MsDos(r);
if (r.flags and $ØØØ1) = 1
     {atvitelkapcsolo beallitva}
  then case r.ax of
     6 : Writeln('A handle ervenytelen');
end;
```

vagy

```
Close(FilVar);    {TURBO -ban, minden trukk nelkul}
```

$3F Olvasás file-ból vagy I/O egységről (Read from File/Device)

```
{ Pr. I. 2.66. }

r.ax:=$3FØØ;
r.cx:=kar_szam;
r.ds:=Seg(puf);
r.dx:=Ofs(puf);
MsDos(r);
if (r.flags and $ØØØ1) = 1
      {atvitelkapcsolo beallitva}
    then case r.ax of
      5 : Writeln('Igeny visszautasitva');
          {Mod=1 - az olvasas nem megengedett}
      6 : Writeln('A file nincs megnyitva');
      else tenylegesen_beolvasott:=r.ax;
         end
    else tenylegesen_beolvasott:=r.ax;
```

vagy

```
Assign(FilVar,ut+filenev);
Reset(FilVar);  {TURBO -ban, minden trukk nelkul}

Blockread(FilVar,puf,
        kar_szam,
        tenylegesen_beolvasott);
```

$40 Írás file-ba vagy I/O egységre (Write to a File/Device)

```
{ Pr. I. 2.67. }

r.ax:=$4ØØØ;
r.cx:=kar_szam;
r.ds:=Seg(puf);
r.dx:=Ofs(puf);
MsDos(r);
if r.flags and $ØØØ1 = 1
        {atvitelkapcsolo beallitva}
   then case r.ax of
     5 : Writeln('Igeny visszautasitva');
         {Mod=Ø, az iras nem megengedett}
     6 : Writeln('A file nincs megnyitva');
     else tenylegesen_kiirt:=r.ax;
        end
   else tenylegesen_kiirt:=r.ax;
```

vagy

```
Assign(FilVar,ut+filenev);
Rewrite(FilVar,1);   {TURBO -ban, minden trukk nelkul}
BlockWrite(FilVar,puf,
           kar_szam,
           tenylegesen_kiirt);
```

$41 File-könyvtárbejegyzés törlése (Delete a Directory Entry)

```
{ Pr. I.2.68 }


r.ax:=$41ØØ;
ut:=ut+#Ø;          {ASCIIZ keszitese}
r.ds:=Seg(ut);
r.dx:=Ofs(ut)+1;
MsDos(r);
if (r.flags and $ØØØ1) = 1
      {atvitelkapcsolo beallitva}
  then case r.ax of
    2 : Writeln('A file nem talalhato');
    5 : Writeln('Igeny elutasitva');
      end;
```

vagy

```
Assign(FilVar,Ut+Filenev);
Erase(FilVar);       {TURBO -ban, minden trukk nelkul}
```

$42 A file-mutató mozgatása (Move a File Pointer)

```
{ Pr. I. 2.69. }
r.ax:=$42ØØ or mozgatasirany;
    {= Ø a file kezdetetol}
    {= 1 relativ modon, aktualis pozicio + tavolsag}
    {= 2 EOF + tavolsag}
r.cx:=tavolsag_f;    {a 32 bites file-mutato felso}
r.dx:=tavolsag_a;    {... es also byte-ja}
r.bx:=handle;
MsDos(r);
case r.ax of
   1 : Writeln('A mozgatasirany nem Ø, 1, 2');
   6 : Writeln('A file nincs megnyitva');
end;
```

vagy

```
                        {TURBO -ban, minden trukk nelkul}
Seek(FilVar,tavolsag);
LongSeek(FilVar,tavolsag);
```

$43 A file-jellemző megváltoztatása (Change Attributes)

```
{ Pr. I. 2.7Ø. }


r.ax:=$43ØØ or funkcio;
    {funkcio=Ø r.cx - ben kapjuk}
    {        =1 r.cx - bol atveszi}
r.ds:=Seg(ut);
r.dx:=Ofs(ut)+1;
if funkcio=1
```

```
 then r.cx:=attribute;     {R/W, Hidden, System}
MsDos(r);
if (r.flags and $ØØØ1) = 1
  then case r.ax of
       1 : Writeln('A funkcio nem Ø..1');
       3 : Writeln('A keresesi utat talalja');
             {a keresesi ut szintaktikusan hibas}
       5 : Writeln('Igeny elutasitva');
             {A file-jellemzo hibas  }
       end
  else if funkcio=Ø
   then Writeln('A file-jellemzo:',r.cx);
```

L. még a *File Dir* nevű programpéldát.

$44 Egységes I/O ellenőrzése (I/O Control for Devices)
L. *Keszulekek* nevű programpéldát.

$45 A handle másolása (Duplicate a File Handle)
Egy megnyitott file-t úgy duplikál, hogy a régi és az új file-mutató ugyanazon file-ra, és ezen belül ugyanazon byte-ra mutat.

```
{ Pr. I. 2.71. }


r.ax:=$45ØØ;
r.bx:=handle;
MsDos(r);
if (r.flags and $ØØØ1) = 1
 then case r.ax of
   4 : Writeln('Tul sok file van megnyitva');
   6 : Writeln('A file nincs megnyitva');
   end
 else
  uj_handle:=r.ax;
```

$46 A handle erőszakolt másolása (Force a Duplicate of a File Handle)
Egy megnyitott file-t úgy duplikál, hogy a file-mutató ugyanazon file-ra és ezen belül ugyanazon byte-ra mutat. Amennyiben a handle már megnyitott, akkor ezt a másolás előtt lezárja.

```
{ Pr. I. 2. 72. }


r.ax:=$46ØØ;
r.bx:=handle;
r.cx:=uj_handle;
MsDos(r);
if (r.flags and $ØØØ1) = 1
 then case r.ax of
   4 : Writeln('Tul sok file van megnyitva');
   6 : Writeln('A file nincs megnyitva');
   end;
```

$47 Az aktuális file-tartalomjegyzék szolgáltatása (Return Text of Current Directory)

```
{ Pr. I. 2.73. }

VAR
  ut : String[64];

r.ax:=$4700;
Fillchar(ut,SizeOf(ut), 0);
r.ds:=Seg(ut);     {Kezdocim}
r.di:=Ofs(ut)+1;
r.dx:=meghajto;    {meghajto azonositas}
MsDos(r);
if (r.flags and $0001) = 1
 then case r.ax of
  15 : Writeln('Hibas meghajto',
                '(a meghajto nem letezik)');
       end
 else begin
       ut[0]:=Char(SizeOf(ut));
       ut[0]:=Char(Pos(#0,ut));
       {Hossz rogzitese}
       end;
```

vagy

```
GetDir(meghajto,ut);  {TURBO -ban, minden trukk nelkul}
```

$48 A tár lefoglalása (Allocate Memory)

```
{ Pr. I.  2.74.}
TYPE
  t_byte_a = array[0..10] of byte;
VAR
  Mem_pointer: ^t_byte_a;
  Mem_kezdet :   integer;

BEGIN

r.ax:=$4800;
r.bx:=igenyelt_tarterulet;
     {paragrafusokban, ahol
      egy paragrafus 16 byte!}
MsDos(r);
if (r.flags and $0001) = 1
 then begin
       case r.ax of
       7 : Writeln('Szettort struktura');
       8 : Writeln('Tulzott koveteles');
       end;
       Writeln('Az igenyelheto tarterulet',
            r.bx,'paragrafusokban');
       end
 else begin
       Mem_pointer:=Ptr(r.ax,0);
       Mem_kezdet:=r.ax;
       end;
```

vagy

```
GetMem(p,igenyelt_byteszam);
          {TURBO -ban, minden trukk nelkul}
```

$49 A tárfoglalás törlése (Free Allocated Memory)

```
{ Pr. I. 2.75. }


r.ax:=$49ØØ;
r.es:=Mem_kezdet;
MsDos(r);
if (r.flags and $ØØØ1) = 1
 then begin
      case r.ax of
      7 : Writeln('Szettort struktura');
      9 : Writeln('Nem $48 hivassal lett lefoglalva');
      end;
      end;
```

vagy

```
FreeMem(p,igenyelt_byteszam);
                    {TURBO -ban, minden trukk nelkul}
```

$4A A tár blokkfoglalásának módosítása (Modify Allocated Memory Blocks)

```
{ Pr. I. 2.76. }


r.ax:=$4AØØ;
r.bx:=igenyelt_tarterulet;
      {paragrafusokban, ahol
       egy paragrafus 16 byte!}
r.es:=Cseg;
MsDos(r);
if (r.flags and $ØØØ1) = 1
 then begin
      case r.ax of
      7 : Writeln('Szettort struktura');
      8 : Writeln('Tulzott koveteles');
      9 : Writeln('Nem $48-hivassal lett lefoglalva');
      end;
      Writeln('Az igenyelheto tarterulet',
              r.bx,'paragrafusban');
      end;



Assign(FilVar,ut+filenev);
Execute(FilVar);   {TURBO -ban, minden trukk nelkul -
                    nincs igazi megfeleloje!}
```

$4B Program betöltése és végrehajtása (Load and Execute a Program)

```
{ ProgVegr.pas }

{ Pr. I. 2.77. }

TYPE t_sor = string[8Ø];
PROCEDURE Program_Vegrehajtas(prg,cmd : t_sor);

TYPE
  dummy = char;
  ptr   = ^dummy;
VAR
  i,
  j        : integer;
  env      : integer absolute Cseg:$2C;
  fcb1     : array[1..32] of char absolute cseg:$5C;
  fcb2     : array[1..32] of char absolute cseg:$6C;

  blk      : record
             env      : integer;
             cmd,
             fcb1,
             fcb2     : ptr;
             end;

BEGIN
  ClrScr;
  r.ax:=$4AØØ;    {a tar felszabaditasa}
  r.es:=Cseg;
  r.bx:=$1ØØØ;    {.com max. 64 kbyte}
  MsDos(r);

  prg:=prg+#Ø;    {ASCIIZ Keresesi ut + programfile-nev}
  cmd:=' '+cmd+#Ø; {ASCIIZ parancssor}
  blk.env:=env;
  blk.cmd:=ptr(Seg(cmd),Ofs(cmd)+1);
  blk.fcb1:=addr(fcb1);
  blk.fcb2:=addr(fcb2);

  r.ax:=$4BØØ;
  r.ds:=Seg(prg);
  r.dx:=Ofs(prg)+1;
  r.es:=Seg(blk);
  r.bx:=Ofs(blk);

  MsDos(r);

  if (r.flags and $ØØØ1) = 1
   then case r.ax of
          1 : Writeln('A funkcio nem 1 vagy 3');
          2 : Writeln('A programot nem talalja');
          8 : Writeln('Tul keves a szabad tarterulet');
         1Ø : Writeln('A $2C nem mutat sehova');
         11 : Writeln('.EXE fejresze hibas');
        end;
END;
```

vagy

```
{ Pr. i. 2.77/a }

Assign(FilVar,ut+programnev);
Execute(FilVar);
                    {TURBO -ban, minden trukk nelkul}
```

$4C Az eljárás befejezése (Terminate a Process)

```
{ Pr. I. 2.78. }

r.ax:=$4C00 or vege_kod;
                    {ERRORLEVEL a Dos 2.0-tol}
MsDos(r);
```

vagy

```
Halt(vege_kod);  {TURBO -ban, minden trukk nelkul}
```

$4D A visszatérési kód meghatározása (Retrieve the Return of a Child)

```
{ Pr. I. 2.79. }

r.ax:=$4D00;
MsDos(r);
Writeln('Vege a programnak');
case Hi(r.ax) of
   0 : Writeln(Lo(r.ax));
   1 : Writeln('^C');
   2 : Writeln('"sulyos hiba"');
   3 : Writeln('$31-es rezidens');
end;
```

$4E File(-ok) keresése a tartalomjegyzékben (Find Match File)

$4F További file(-ok) keresése a tartalomjegyzékben (Step Through a Directory Matching Files)

```
{ OlvDir.pas    }
{ Pr. I. 2.80. }

{$I regs_FCB.pas}
TYPE
  t_Search_Status = (uj,meg_valami,vege,hibas);
  t_file = record
      attr            : byte;
      ido,
      datum,
      meret_a,
      meret_f         : integer;
      nev             : array[1..13] of char;
    end;
  t_sor = string[80];

  t_file_blk=record
      res      : array[1..21] of byte;
      entry    : t_file
    end;
```

```
VAR
  file_blk    : t_file_blk;
  Search_Status:t_Search_Status;

FUNCTION Keres(ut    : t_sor; attr : integer;
               VAR allomany : t_file_blk)     :boolean;

VAR
  Search_Ok : boolean;

PROCEDURE Set_Disk_Transfer_Address;
  BEGIN
    r.ax:=$1AØØ;
    r.ds:=Seg(allomany);
    r.dx:=Ofs(allomany);
    MsDos(r);
  END;

BEGIN
  Search_Ok:=false;

  case Search_Status of
    uj:begin
         Set_Disk_Transfer_Address;
         Fillchar(allomany,SizeOf(allomany),Ø);
         allomany.entry.nev:='                ';
         ut:=ut+#1;       {ASCIIZ}

         r.ax:=$4EØØ;
         r.ds:=Seg(ut);
         r.dx:=Ofs(ut[1]);
         r.cx:=attr;
         MsDos(r);

         if r.ax = 18
          then Search_Status:=vege
          else if r.ax=2
           then Search_Status:=hibas
           else begin
            Search_Status:=meg_valami;
            Search_Ok:=true;
           end;
       end;
    meg_valami:
      begin
         allomany.entry.nev:='
         r.ax:=$4FØØ;
         MsDos(r);
         if (r.ax)=18
          then Search_Status:=vege
          else begin
            Search_Status:=meg_valami;
            Search_Ok:=true;
          end;
       end;
vege:;
hibas:;
end;
    Keres:=Search_Ok;
END;
```

L. még a *File Dir* nevű programot.

$54 Az írásellenőrzés állapotának lekérdezése (Return Current Setting of Verify)

```
{ Pr. I. 2.82. }


r.ax:=$54ØØ;
MsDos(r);
Writeln('Irasellenorzes',r.ax);
```

$56 A file átmásolása másik tartalomjegyzékbe és/vagy átnevezése (Move/Rename a Directory Entry)

```
{ Pr. I. 2.83 }


r.ax:=$56ØØ;
ut:=ut+#Ø;              {ASCIIZ karakterlanc keszitese}
r.ds:=Seg(ut);
r.dx:=Ofs(ut)+1;
uj_ut:=uj_ut+#Ø;        {ASCIZZ karakterlanc keszitese}
r.es:=Seg(uj_ut);
r.di:=Ofs(uj_ut)+1;
MsDos(r);
if (r.flags and $ØØØ1) = 1  {atvitelkapcsolo beallitott}
  then case r.ax of
     2 : Writeln('file not found- a file nem talalhato');
     5 : Writeln('acces denied- az eleres visszautasitva');
                 {csak konyvtart adott meg,
                  vagy a file mar letezik}
    17 : Writeln('not same device- a meghajto nem azonos');
       end;
```

$57 A file rögzítési dátumának és időpontjának beállítása vagy lekérdezése *(Get/Set Date/Time of File)*

```
{ Pr. I. 2.84. }
r.ax:=$57ØØ or funkcio;   {Ø = lekerdezes,
                           1 = beallitas}
r.bx:=handle;
if funkcio=1
 then begin             {15---1Ø---5---1-}
  r.cx:=datum;          {EEEEEEEHHHHNNNNN}
  r.dx:=ido;            {OOOOOPPPPPPMMMMM}
end;
MsDos(r);
if (r.flags and $ØØØ1) = 1
 then case r.ax of
   1 : Writeln('hibas  funkcio - nem Ø vagy 1');
   6 : Writeln('hibas handle - a file nincs megnyitva');
     end
  else
   if funkcio=Ø
    then begin                  {15---1Ø---5---1-}
      Writeln(r.cx shr 9,'.', {EEEEEEEHHHHNNNNN}
              (r.cx and $Ø1EØ) shr 5,'.',
              (r.cx and $ØØ1F));
      Writeln(r.dx shr 11,':', {OOOOOPPPPPPMMMMM}
              (r.dx and $Ø7EØ) shr 5,':',
              (r.dx and $ØØ1F));
    end;
```

3. A Turbo–Pascal további lehetőségei

Az előző fejezetben túlnyomórészt új parancsokról volt szó, amelyek a rendszerközeli programozásnál hasznosak. Ezeken kívül a *Turbo–Pascal* egy sor fontos bővítést is tartalmaz, amelyek mindenekelőtt az adattípusokra és a file-kezelésre vonatkoznak. Sok új eljárás és függvény bővíti a *Standard–Pascal* alapelvét és segíti a programozót fáradságos munkájában.

3.1. ADATTÍPUSOK

3.1.1. Bitek

Mivel a tárat, mint azt az *MS–DOS/PC–DOS* alapjaival foglalkozó fejezetben már említettük, byte-onként kell címeznünk, a bitek a *Turbo–Pascal*-ban adattípusként nem állnak explicit alakban rendelkezésre. Mégis az integer, ill. byte számok megkerülésével az egyes bitek manipulálhatók. Erre a célra az operátorok két csoportja létezik, amelyek valójában az assembler programozás köréből származnak:

— az eltoló operátorok és
— a logikai operátorok.

Tekintsük a számok bináris ábrázolását az operátorok működésének jobb megértéséhez.

Példa: 12 binárisan 1100

Az egyszerűség kedvéért a vezető nullákat elhagyjuk, mivel az átalakítás végeredményére nincsenek befolyással.

Operátorok:

a) *Eltoló operátorok*

SHL (shift left): balra tolja a számot az adott helyiértékhez képest. A bal szélen álló helyek elvesznek, míg a jobb oldalt nullákkal tölti fel.

SHR (shirt right) értelem szerint ugyanúgy hat jobbra.

Az egy hellyel történő balra tolás 2-vel való szorzásnak felel meg, az egy hellyel történő jobbra tolás pedig 2-vel való osztásnak.

Példa: 5 shl 2 jelentése: 101 shl 2 eredménye: 10100(20)
13 shr 1 jelentése: 1101 shr 1 eredménye: 110(6)

b) *Logikai operátorok*

Ezeket a logikai kifejezésekből ismert operátorokat használhatjuk az egész számokhoz is.

Definíció: legyen A és B két integer szám, ill. a és b két tetszőleges bit azonos pozíción (0. hely, 1. hely stb.) az A és B számból.

A **and** *B:*

a	*b*	eredmény
1	1	1
1	0	0
0	1	0
0	0	0

pl. A = 0111010000000101
B = 0000100101010101
0111110101010101

A **or** *B*

a	*b*	eredmény
1	1	1
1	0	1
0	1	1
0	0	0

pl. A = 0111010000000101
B = 0000100101010101
0111110101010101

A xor *B* *a b* eredmény

1	1 0
1	0 1
0	1 1
0	0 0

pl. *A* = 0111010000000101
B = 0000100101010101
0111110101010000

not *A* *a* eredmény

1	0
0	1

pl. *A* = 0111010000000101
1000101111111010

A következő program ezeknek az operátoroknak a felhasználási lehetőségeit példázza.

```
{ BITPL.PAS    }
{ Pr. I. 3.1. }

PROGRAM Bit_pl;

TYPE
   t_bitpos = Ø..15;
   t_irany  = (be,ki);

VAR
  i,j : integer;

PROCEDURE Bit_valt (VAR szam : integer;
                        pos  : t_bitpos;
                        irany  : t_irany);

{  Az egeszszam Ø-tol 15-ig terjedo bitjei egymastol
   fuggetlenul
   be- vagy kikapcsolhatok  }

   BEGIN
     if irany = ki
      then szam:=szam xor (1 shl pos)
      else szam:=szam or  (1 shl pos);
   END; {.....................................}

FUNCTION Bit_Beallit (szam : integer;
                      pos  : t_bitpos) : boolean;

   BEGIN
     bit_Beallit:=(szam and (1 shl pos)) shr pos = 1;
   END; {.....................................}

FUNCTION paratlan (szam : integer) : boolean;

   BEGIN
     paratlan:=bit_Beallit(szam,1);
   END; {.....................................}

PROCEDURE Binarisszam (szam : integer);

   VAR i : t_bitpos;
```

```
  BEGIN
    for i:=15 downto Ø
     do if bit_Beallit(szam,i)
         then Write('1')
         else Write('Ø');
  END; {.....}


PROCEDURE Csere (VAR i,j : integer);

{ Az eljaras i es j valtozok tartalmat felcsereli
  segedvaltozo hasznalata nelkul. }

   BEGIN
     i:=i xor j;
     j:=i xor j;
     i:=j xor i;
   END; {.....}

BEGIN
  ClrScr;
  Lowvideo;
  Writeln('SZAMOK');
  GotoXY(1,4);
  Write('Binaris szam:                ');
  Write('        Decimalis szam:');
  HighVideo;
  for i:=1 to 32767
   do begin
     GotoXY(16,4);
     Delay(1ØØ);
     Binarisszam(i);
     GotoXY(55,4);
     Write(i:5);
   end;
END.
```

3.1.2. Byte-ok

A *byte* adattípusú változók 8 bitből állnak és értékük 0 és 255 között lehet. Elsősorban rendszerprogramozáshoz használjuk ezeket, a tárelemek, ill. a regiszterek kezelésére.

Operátorok

Az *integer* számokhoz alkalmazható operátorok mindegyikét használhatjuk a *byte* számokhoz is.

A kettes komplemens alapján lehetséges egy byte-ban negatív számmal is számolni, amelyek azonban a kivitelkor pozitív számként vannak ábrázolva. A kettes komplemens az informatika alapmondataként tekinthető és a következőképpen működik: negatív számokat olyan bináris számokként ábrázolja, amelyekben a biteket a bináris számábrázolás szerint invertálja (0 helyett 1 és fordítva) és 1-et hozzáad. Az így előállt szám minden műveletben használható anélkül, hogy az előjellel törődnünk kellene; automatikusan helyes, ha a bal szélső bitet előjelként értelmezzük és a túlcsordulást levágjuk.

Példa: tekintsük a –3-as számot.

A 3 binárisan 8 bit hosszban:	00000011
a szám inverze:	11111100
hozzáadva 1-et:	11111101
A 4 bináris ábrázolásban:	00000100

Adjuk össze a bináris –3-at és 4-et:

```
 11111101
 00000100
 --------
 10000001
```

Az eredmény 9 bites, a bal szélen levő bitet a *byte* adattípus miatt levágja. Ezzel eredmény 1, ami helyes.

Amennyiben az eredményt közvetlenül integer számmá alakítanánk, hibás ere ményt, 257-et kapnánk, mivel megmaradt a bal szélső bit.

Adjuk össze a –4-et és a 3-at. Az eredmény –1, ami 11111111-nek felel meg. szám kiíráskor 255-ként jelenik meg.

A következő programpéldán kívül a további programok is tartalmaznak példákat a *by* adattípusra.

```
{ BYTEPL.PAS  }
{ Pr. I. 3.2. }

PROGRAM Byte_pl;

VAR
   a,b,c : byte;
   d     : integer;

BEGIN
  ClrScr;
  a:=-3;
  b:=4;
  c:=a+b;
  d:=a+b;
  Writeln('Pelda a Byte adattipusra');
  Writeln('------------------------');
  Writeln;
  Writeln('a = ',a:3,'   (-3) mint pozitiv szam abr.')
  Writeln('b = ',b:3);
  Writeln('c = ',c:3,'   Byte    - Eredmeny');
  Writeln('d = ',d:3,'   Integer - Eredmeny');
END.
```

3.1.3. Karakterláncok

A *Standard–Pascal*-ban a karakterláncokat *tömörített karaktertömböknek* (packed arrays of char) definiálták. Ennek az a hátránya, hogy nem segíti a változtatható hosszúságot. A *Turbo–Pascal*-ban ezt a hátrányt megszüntették, mert használható az előre definiált *string* adattípus. Az ilyen típusú változó különböző hosszúságú (meghatározott felső határig) lehet. Az egyes elemeket, amelyek a *char* típushoz tartoznak *tömb*ként (array) kezeli. A *string*-változóban legfeljebb 255 jel tárolható, mert az első byte-ban a változó aktuális hossza áll és egy byte legnagyobb értéke 255 lehet.

A *string*-ek konstansként is szerepelhetnek a programban, ahol mint karakterláncot, felsővesszők között kell megadni.

A következő program egy teljes sor ASCII kódra való átalakítását mutatja. Képzés közben az egyes kódértékekből ellenőrző számot összegez.

```
{ SORKONV.PAS }
{ Pr. I. 3.3  }

PROGRAM Sorkonv;

{ Pelda teljes sor ASCII-kodra torteno konvertalasara
  az 'ord' fuggveny alkalmazasa nelkul' }

TYPE
   t_sor       = string [80];
   t_ascii_sor = array [0..80] of byte;

VAR
   sor       : t_sor;
   ascii_sor : t_ascii_sor absolute sor;
   i         : byte;
   prf       : integer;

BEGIN
  ClrScr;
  Writeln('Adja meg a sort ! ');
  Writeln;
  Readln(sor);
  Writeln;
  Write('Ellenorzo szam: ');
  for i:=1 to ascii_sor[0]
   do prf:=prf+ascii_sor[i];
  Write(prf);
END.
```

Az előző fejezethez hasonlóan vizsgáljuk meg az operátorokkal végzett stringműveleteket.

Operátorok

Elvben csak két alapvető művelet áll rendelkezésre: az összefűzés és az összehasonlítás.

a) *Összefűzés*

A + operátort a stringműveletek során két vagy több karakterlánc összefűzésére használjuk.

b) *Összehasonlítás*

Az ismert relációs operátorokat $<$, $>$, $<=$, $>=$, $=$, $<>$, használhatjuk a karakterláncokhoz is, hogy logikai kifejezéseket kapjunk.

Két karakterlánc összehasonlításakor balról jobbra haladva az egyes jeleket hasonlítjuk össze egymással (miközben a jelek ASCII kódját vizsgáljuk). Ha az egyik karakterlánc hamarabb véget ér, de végig azonos volt a másikkal, úgy az a kisebb. A karakterláncok csak akkor egyenlőek, ha azonos hosszúságuk és azonos karakterekből állnak.

Példa:	*'ABC'*	kisebb, mint	*'ABCD'*
	'123'	egyenlő	*'123'*
	'A'	nagyobb, mint	*'1'*
	'A'	nagyobb, mint	*'12'*

A Turbo–Pascal egész sor előre definiált eljárást és függvényt ad a karakterláncok kezeléséhez:

- Delete karakter törlése;
- Insert karakter beszúrása;
- Copy karakter másolása;
- Concat karakter összefűzése;
- Str szám átformálása karakterlánccá;
- Val karakterlánc átalakítása számmá;
- Lenght karakterlánc hossza;
- Pos a karakter pozíciója.

Ezekkel a karakterláncok kényelmesen feldolgozhatók. A következő oldalakon alaposan megvizsgáljuk a működésüket, mivel ezeket a függvényeket és eljárásokat gyakran alkalmazzuk. A további példák e könyv szinte minden programjában megtalálhatók.

DELETE

A *Delete* eljárást használjuk egy stringváltozóból adott számú karakter meghatározott helytől kiinduló kitörléséhez.

Paraméterek:

- stringváltozó (string)
- pozíció (byte vagy integer)
- törlendő karakterek darabszáma (byte vagy integer)

Különleges esetek:

- ha a pozíció a karakterláncon kívül van, mincs művelet;
- ha a pozíció és a darabszám nagyobb, mint a karakterlánc hossza, ekkor csak a karakterlánc belsejében törli a karaktereket;
- ha a pozíció megadás nagyobb, mint 255, akkor az a futás közben hibajelzést eredményez.

A következő példa a *Delete* alkalmazását mutatja. Egy szekvenciális file-t olvas be és minden rekord egy részét kitörli.

```
{ TORLES.PAS  }
{ Pr. I. 3.4. }

PROCEDURE Torles (VAR f,g  : text;
                  tol, db : byte);

VAR
   sor : string[255];

BEGIN
   while not Eof(f)
    do begin
      Readln(f,sor);
      Delete(sor,tol,db);
      Writeln(g,sor);
   end;
END; {.....}
```

INSERT

Az *Insert* eljárás pontosan a *Delete* eljárás ellentéte. Akkor alkalmazzuk, ha egy stringváltozóba kell adott számú karaktert meghatározott helytől kiindulva beilleszteni.

Paraméterek:
- stringváltozó (string)
- beillesztésre váró karakterek (string)
- pozíció (byte vagy integer)

Különleges esetek:
- ha a pozíció a karakterláncon kívül van, a beillesztendő karaktereket hozzáfűzi;
- ha a pozíció és a beszúrandó karakterek darabszáma nagyobb, mint a karakterlánc hossza, akkor csak a karakterlánc hosszán belül elférő karaktereket illeszti be;
- ha a pozíciómegadás nagyobb, mint 255, akkor az a futás közben hibajelzést eredményez.

Az előző oldalon levő eljárást megváltoztattuk, hogy az *Insert* felhasználását bemutathassuk.

```
{ BESZURAS.PAS }
{ Pr. I. 3.5.  }

PROCEDURE Beszuras (VAR f,g : text;
                        poz    : integer;
                        s      : t_str255);

VAR
   sor : string[255];

BEGIN
   while not Eof(f)
    do begin
      Readln(f,sor);
      Insert(s,sor,poz);
      Writeln(g,sor);
    end;
END;
```

COPY

A *Copy* függvényt a karakterlánc egy részének kivitelére használjuk.

Paraméterek:
- karakterlánc (string)
- pozíció (byte vagy integer)
- a másolásra kerülő karakterlánc hossza (byte vagy integer)

Függvényérték:
- részlánc (string)

Különleges esetek:
- ha a pozíció a karakterláncon kívül esik, akkor a kiadott részlánc üres,
- ha a pozíció és a darabszám nagyobb a karakterlánc hosszánál, akkor csak a karakterlánc hosszán belüli karakterek kerülnek kimásolásra;
- ha a pozíciómegadás nagyobb mint 255, akkor az a futás közben hibajelzést eredményez.

Következő programpéldánk a *Copy* függvény alkalmazását mutatja. További példa található a *Val* eljárásnál.

```
{ MASOLAS.PAS }
{ Pr. I. 3.6. }

PROCEDURE Masolas (VAR f, g : text;
                       tol, db : byte);

VAR
   sor : string[255];

BEGIN
  while not Eof(f)
   do begin
      Readln(f,sor);
      Writeln(g,Copy(sor,tol,db));
   end;
END;
```

LENGTH

A *Length* függvény a karakterlánc aktuális hosszát közli.

Paraméterek:
– karakterlánc (string)

Függvényérték:
– hossz (byte)

Különleges esetek:
Nincsenek. Minden hibát futás közbeni hibaként kezel.
A függvény a *Concat* függvény leírásához csatolt programpéldában fordul elő.

POS

A *Pos* függvény egy karakterlánc egy adott részláncának kezdő pozícióját keresi meg.

Paraméterek:
– részlánc (string)
– karakterlánc (string)

Függvényérték:
– pozíció (byte)

Különleges esetek:
– ha a részláncot nem találja, a függvény értéke 0. Minden más hibát futás közbeni hibaként kezel.

A függvény alkalmazása a *Val* eljárás programpéldájában is megtalálható.

CONCAT

A *Concat* függvény több részlánc összefűzésére használható. Ez azonban lényegesen kényelmesebb a + operátor felhasználásával.
Ez a függvény csak a többi *Pascal*-nyelvvel való kompatibilitást biztosítja.

Paraméterek:

— részlánc (string)
— részlánc (string)
.
.
.
— részlánc (string)

Függvényérték:

— karakterlánc (string)

Különleges esetek:

Nincsenek. Minden hibát futás közbeni hibaként kezel.
A következő programpélda a *Concat* felhasználásával a karakterlánc megfordítását végzi.

```
{ VISSZA.PAS  }
{ Pr. I. 3.7. }

FUNCTION Vissza (s : t_str255) : t_str255;

VAR
   uj_s : t_str255;
   i    : integer;

BEGIN
  uj_s:='';
  for i:=Length(s) downto 1 do uj_s:=Concat(uj_s,s[i]);
  Reverse:=uj_s;
END;
```

STR

Az *Str* eljárással egy számot karakterlánccá alakíthatunk át.
A *Write* írásutasításhoz hasonlóan, formátumrögzítő paramétert adhatunk mellé.

Paraméterek:

— szám, esetleg formátummegadással (byte, integer vagy real)
— változó a karakterek felvételére (string)

Különleges esetek:

Nincsenek. Minden hibát futás közbeni hibaként kezel.
Érdekes felhasználást mutat a következő program. Egy szám jegyeinek számát úgy állapítja meg, hogy a számot karakterlánccá alakítja.

```
{ SZAMKONV.PAS }
{ Pr. I. 3.8.  }

PROCEDURE Szamkonv (ossz, dec : integer);
{ Egy szam szamjegyeinek szamat adja. }

VAR
   z_str : string[255] ;
   i,j   : integer;
   z     : real;
```

```
BEGIN
  Readln(z);
  Str(z:ossz:dec,z_str);
  i:=1;
  while z_str[i] = ' ' do i:=i+1;
  j:=Length(z_str);
  while z_str[j] = '0' do j:=j-1;
  Writeln(j-i);
END;
```

VAL

Fontos eljárás. A segítségével egy karakterlánc számmá alakítható át, azaz az előző oldalon leírt *Str* eljárás ellentéte.

Paraméterek:

- karakterlánc (string)
- változó a szám felvételére (byte, integer vagy real)
- hibaváltozó (byte vagy integer)

A hibaváltozó értéke akkor 0, ha az átalakítás sikeres volt; különben a karakterláncban annak a pozíciónak az értékét tartalmazza, ahol az átalakítás közben is hiba keletkezett.
A karakterlánc nem tartalmazhat sem vezető, sem befejező szóközt.

Különleges esetek:
Nincsenek. Minden hibát futási hibaként kezel.
A programpélda egy egyszerű összeadási műveletet mutat be.

```
{ ERTEK.PAS   }
{ Pr. I. 3.9. }

FUNCTION Ertek (Formula : t_str20) : real;

VAR
   z1,z2 : real;
   p,hiba : integer;

BEGIN
  p:=Pos('+',formula);
  if p > 0
   then begin
     Val(Copy(formula,1,p-1),z1,hiba);
     Val(Copy(formula,p+1,Length(formula)),z2,hiba);
     if hiba = 0
      then ertek:=z1+z2
      else ertek:=9999;
   end;
END;
```

3.2. INCLUDE FILE-OK

Olyan programrészletek, amelyek a fordítás során kerülnek a főprogramba. Ha a fordítóprogram *Include* utasítást talál, leáll a főprogram fordításával, a megadott file-t beolvassa és lefordítja, csak ezután folytatja a főprogram fordítását. Igy a lefordított program úgy néz ki, mintha az include file kódjai a főprogramban lettek volna.
Az include file-okat kétféleképpen használhatjuk:
— programokhoz, amelyek nagyobbak az *editor* által feldolgozható méretnél, valamint
— a strukturált programozás eszközeként.

A Turbo—Pascal nem engedi meg program szétválasztva való lefordítását és utólagos összefűzését. Az include file-ok alkalmazása az egyetlen lehetőség gyakran alkalmazott programrészleteknek leválasztására és több programban való felhasználására anélkül, hogy fizikailag bármelyik programba be kellene másolnunk a file-okat.
Ezek a programrészek lehetnek konstans deklarációk vagy típus deklarációk, de eljárások vagy a főprogram más részei is.
A fordítóprogram *I* paramétere segítségével specifikálható az include eljárás, ezután a file nevének kell következnie, amelynek a típusa tetszés szerinti lehet.

3.3. OVERLAY FILE-OK

Néha szükséges olyan programok kifejlesztése, amelyek nagyobb terjedelműek, mint a rendelkezésre álló tárterület. E probléma megoldása a *Turbo–Pascal*-ban kétféle technikával lehetséges:

– overlay file-okkal és
– programösszefűzéssel, amelyet a következő fejezetben tárgyalunk.

Az *Overlay* kifejezés *föléhelyezést* jelent, megadja ennek a technikának az alapvető működésmódját.

A lefordított főprogram egy sor eljárást és függvényt fog össze és elkülönítve azokat egy vagy több file-ban tárolja. A módszer automatikus, ha az eljárás-, ill. függvénydeklarációban megadjuk az *Overlay* megjelölést.

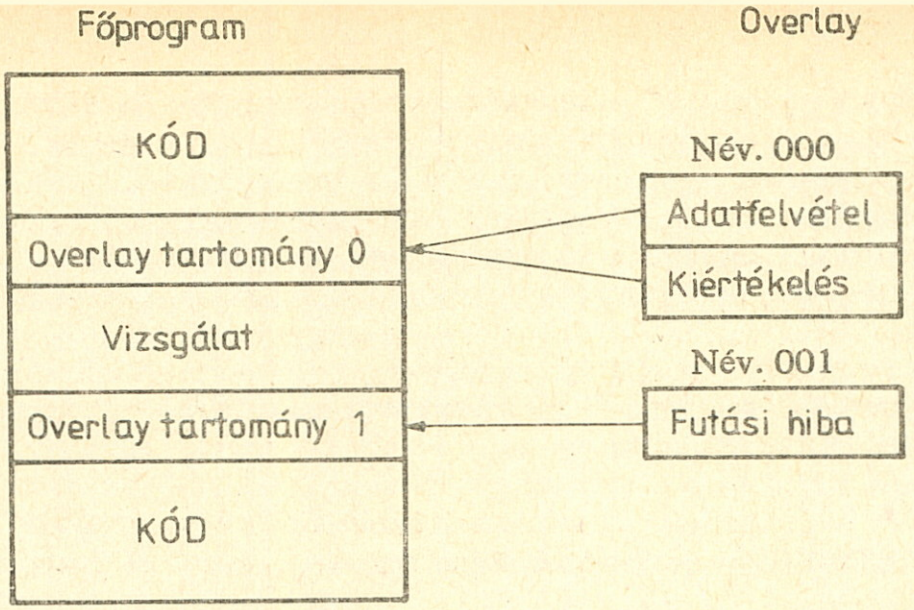
Példa:

```
Overlay PROCEDURE Adatfelvétel;
BEGIN
...
END;
Overlay PROCEDURE Kiértékelés;
BEGIN
...
END;
FUNCTION Vizsgálat: boolean;
BEGIN
...
END;
Overlay PROCEDURE Futási hiba;
BEGIN
...
END;
```

Az overlay file-ok neve megegyezik a főprogramével és a file-kiterjesztés segítségével átszámozódnak (.000-tól .999-ig.), miközben a lefordított kódokat ugyanabba a file-ba rakja mindaddig, amíg az eljárások és a függvények *Overlay* megjelöléssel megszakítás nélküli sorban következnek. Amint normális deklarációt talál vagy újabb főprogram kezdetét, akkor lezárja az overlay file-t, ha ismét talál overlay eljárást vagy függvényt, akkor új overlay file-t nyit.

A két első eljárás tehát a fordítás után a Név.000 file-ban, az utolsó eljárás pedig a Név.001 file-ban található.

A főprogram futásakor a tárban két overlay tartományt foglal le. Az első a Név.000 file alprogramjához kell, a másik pedig a Név.001 file-hoz. A tartomány nagyságát az overlay file-ban elhelyezett legnagyobb alprogram szabja meg. Amikor az alprogramot a főprogram behívja, akkor a kódot az overlay file-ból a tárba áttölti, mivel az overlay file-ból mindig csak egy eljárás tölthető be a tárba, az ilyen file-ban együtt található alprogramok nem hívhatók egyszerre. Amennyiben erre mégis szükséges volna, akkor egy második overlay file-t is le kell foglalni (l. a példát és a következő ábrát).



Az overlay technika hátránya természetesen az az időveszteség, amelyet az alprogramkódok betöltése okoz, ezért a betöltés idejét lehetőség szerint csökkenteni kell. Az alábbi szabályok betartását ajánljuk:

- az overlay file alprogramjait nem szabad gyakran hívni, ha mégis szükséges, akkor közben nem szabad újabb programot az overlay file-ból betölteni,
- a hajlékonv lemez helyett jobb a merev lemez használata, mivel ez lényegesen gyorsabb hozzáférésű tároló,
- lehetőség szerint kevés, nagy kiterjedésű overlay file-t alakítsunk ki, ne sok kisméretűt.

A *Turbo–Pascal* az overlay file-jait alapesetben az aktuális meghajtó aktuális könyvtárában keresi. A file eléréséhez (pl. egy másik meghajtóról vagy könyvtárból) szükségünk lehet keresési út specifikálására. Az *Ovrpath* eljárást használhatjuk erre a célra.

```
{ AKTMEGH.PAS  }
{ Pr. I. 3.1Ø. }

PROCEDURE Akt_Meghajto_Beall;

VAR
   meghajto : char;

BEGIN
  Writeln('Az Overlay-file nincs az aktualis lemezen ');
  Write('Adja meg a megfelelo meghajtot (A,B,C) : ');
  Readln(meghajto);
  Ovrpath(meghajto);
END;
```

Az utasítás hatására a következő overlay alprogram meghívásakor az overlay file-t a keresési úttal kijelölt meghajtó könyvtárából tölti be. Ugyanez elérhető természetesen a *Chdir* eljárással is, de ez azt jelentené, hogy az összes többi file-t is ebben a könyvtárban keresné a program.
Az *Ovrpath* eljárás előnye abban áll, hogy a keresési út specifikálása csak overlay file-okra érvényes.

3.4. A FILE-KEZELÉS ELVI ALAPJAI

A *Turbo–Pascal*-ban a file-okat *adatstruktúráknak* tekintjük és akként is kezeljük, a tömbökkel vannak rokonságban. A különbség az, hogy a tömbök végesek (mivel a tárterület korlátozott), míg a file-ok potenciálisan végtelenek (a külső tárolóközeg kapacitása korlátozza csupán). Egy hajlékony lemezen rendelkezésre álló hely többnyire többszöröse a tár adatfeldolgozásra használható területének, ez is igazolja a fenti szemléletet.
A *Standard–Pascal*-lal szemben a *Turbo*-ban egész sor bővítés van, mint például a közvetlen hozzáférésű file-ok, a típus nélküli file-ok és az egységfile-ok. A kezelésükre használatos függvények és eljárások mellett a már megismert adatátviteli módszereket hatékony parancsok is támogatják. A későbbiekben ezeket részletesebben is tárgyalni fogjuk.
A fent említettek szerint a file-okat változóknak fogjuk fel és a típusának a hozzárendelésével definiáljuk. Ennek következtében és a hozzáférési mód szerint az alábbi file-típusok lehetségesek.

a) *Szekvenciális file-ok*
Ennél a file-típusnál az elemek csak egymás után dolgozhatók fel. A file-ok tehát vagy csak olvasásra, vagy csak írásra megnyitott állapotban lehetnek. Egy file módosításához, a változatlan rekordokat le kell másolni, a módosítandó rekordokat pedig ki kell cserélni. Ez a művelet időigényes és ezért a dialógusszerű feldolgozáshoz nem alkalmas.
– szövegfile-ok (az egységfile-okkal)
– strukturált file-ok
– típus nélküli file-ok.

b) *Közvetlen elérésű file-ok*
Itt a rekordszám segítségével, amelyet külön kell kezelnünk tetszés szerinti rekordhoz nyúlhatunk, így a file egyes rekordonként olvasható és írható. Ideális a dialógushoz:
– strukturált file-ok
– típus nélküli file-ok.

A *Turbo–Pascal* programokból lehetőségünk van a file-ok átnevezésére és törlésére a *Rename* és az *Erase* eljárásokkal. Ezek a parancsok mindegyik file-típusra érvényesek. Példa a szövegfile-oknál található.

File-illesztő blokk (File Interface Block–FIB)
A *Turbo–Pascal* minden file-hoz rendel egy *FIB*-et, amely néhány, a file kezelésére vonatkozó információt tartalmaz. A további információk a *DOS* által kezelt file-tartalomjegyzékben találhatók meg.
A *FIB* felépítése a file-típusnak megfelelően (szövegfile, strukturált file stb.) különböző.

3.4.1. Szekvenciális file-ok

3.4.1.1. Szövegfile-ok

Bár a szövegfile-ok is mutatnak bizonyos szerkezetet (a sorokra gondolunk), nem soroljuk a strukturált file-okhoz, azaz elhatároljuk a rekordokkal meghatározott file-októl. Ebben különbözünk a *Turbo–Pascal* kézikönyvtől, amely a strukturált file-okat típussal rendelkező file-oknak nevezi. Nem tartjuk helytelennek azt, hogy a szövegfile-okat

strukturálatlan file-oknak tekintsük, noha a szövegfile-oknak is van típusuk. Továbbá, csak a szövegfile-oknál van különbseg az egységfile-ok és a lemezes file-ok között. A többi file-típus csak lemezes file-ként alkalmazható.
A *FIB* leírása a készülékfile-oknál található, mivel előzőleg még néhány alapvető dolgot meg kell tárgyalnunk. Az ott megadott információ a lemezes file-okra is vonatkozik.

Lemezes file-ok
A szövegfile-ok egyes karakterek sorozatából állanak. Az alapvető struktúra a különálló soroké. Ezeket a kocsi vissza (Carriage Return – CR) és a soremelés (Line Feed – LF) zárja le. Mivel szekvenciális file-ról van szó, ez csak vagy olvasható, vagy írható. A programban a következőképpen kell definiálni:

VAR bevitel,kivitel:*text;*

A *text*-típus mutatja a *Turbo–Pascal*-nak, hogy a *bevitel* és a *kivitel* változóknál szövegfile-ról van szó. Az első felhasználás előtt a változókat a *DOS*-ban érvényes file-névhez kell rendelnünk. Ezt a program, utasítási részében az *Assign* eljárással tesszük meg.

Példa: *Assign* (bevitel, 'inp.dta')
Assign (kivitel, 'out.dta')

Ezzel összekapcsoltuk az *inp.dta* file-nevet a *bevitel* változóval és az *out.dta* file-nevet a *kivitel* változóval. Meg kell adnunk, hogy az adatállományt olvasni vagy létrehozni kell-e? Erre való a két standard eljárás a *Reset* és a *Rewrite.*

Példa: *Reset* (bevitel)
Rewrite (kivitel)

Az első parancs a bevitelt, a második a kivitelt készíti elő. Ebben a pillanatban a következő esetek különböztethetők meg.

A bevitel esetében:
- ha az *inp.dta* file nem létezik, I/O hibát (01) jelez. A *Turbo–Pascal* a tartalomjegyzék átvizsgálása után megállapította, hogy nem lehetséges hozzárendelés;
- ha az *inp.dta* létezik minden rendben.

A kivitel esetében:
- ha az *out.dta* file nem létezik, minden rendben van, a *Turbo–Pascal* megkísérel a tartalomjegyzékbe a file-ról bejegyzést készíteni. Amennyiben a tartalomjegyzék megtelt, akkor is I/O hibát jelez, de ebben az esetben az F1-es hibaüzenetet küldi.

- ha az *out.dta* file létezik, a szóban forgó file-t felülírja. Nincs I/O hibajelzés.

Mihelyt I/O-hibát állapít meg, a *Turbo–Pascal* megszakítja a programot és megadja a *programszámlálót* (programcounter – PC) és a hibaszámot. Ez elkerülhető, az *I* fordítóprogram-paraméter segítségével (ne keverjük össze az Include direktívával), ekkor a hibarutinok meghívását magunk is vezérelhetjük.

Beállítás:
I+ meghívja a *Turbo–Pascal* hibarutinjait (alapbeállítás),
I– a program a hibák ellenére továbbfut.
Az *I* beállításakor meg kell hívni az *IO result* standard függvényt és hiba esetén a hibát ki kell küszöbölni. Az *IO result* értéke 0, ha hibátlan a futás. Minden más érték hibát jelent.

Bemutatunk egy *Turbo–Pascal* programrészletet, amelynek egy file-t kell kivitelre előkészítenie. Megvizsgálja, hogy van-e azonos nevű file és átnevezi azt, mielőtt a *Rewrite* eljárás végrehajtásra kerülne.

```
{ NYITAS.PAS }
{ Pr. 3.11.  }

FUNCTION Nyitas (VAR szoveg : text;
                     nev   : t_dat_nev) : boolean;

{  A fuggveny az alabbi ertekeket szolgaltathatja:
   True : sikeres megnyitas
   False: egyebkent
   Az esetleg letezo azonos nevu file-t elobb
   atnevezi. }

VAR
   str_veg  : integer;
   nev_regi : t_dat_nev;

BEGIN
   Assign(szoveg,nev);
   {$I-}
   Reset(szoveg);
   {$I+}
   if IOresult = 0
    then begin
      Close(szoveg);
      nev_regi:=nev;
      str_veg:=pos('.',nev);
      if str_veg <> 0
       then
         nev:=Copy(nev,1,pos('.',nev)-1)+'.BAK'
       else
         nev:=nev+'.BAK';
       Rename(szoveg,nev);
       Close(szoveg);
       Assign(szoveg,nev_regi);
     end;
   {$I-}                    { Betelt az alkonyvtar? }
   Rewrite(szoveg);
   {$I+}
   Open_out:=IOresult = 0;
END; {.....................................}
```

Eddig minden rendben ment, az adatmozgatás végrehajtható. Ez a *Read, Readln, Write, Writeln* eljárás segítségével valósul meg, amelyek a Standard–Pascalból már ismertek és ezért itt nem kívánjuk bővebben magyarázni.
Sajnos a *Turbo–Pascal*-ban megvalósított file-kezelésnek is van hátránya, a *Put* és a *Get* standard eljárás nem ismert. A file-mutatón keresztül való hozzáférés nem megengedett. Az adatátvitelt mindenkor a fent említett eljárásokkal kell megvalósítani.
Ez azzal a következménnyel jár, hogy a file-mutató átállítása és a változók értékadása egyszerre történik és csak ezután következhet az ellenőrzés. A jó öreg *Put* és *Get* eljárással a programozó a kezében tartotta a file-mutatóval való manipulálás lehetőségét, ill. a mutatóval történő összehasonlítás lehetőségét anélkül, hogy változót kellett volna deklarálnia. Ezek hiánya a *Turbo*-ban pótváltozók felhasználását teszi néha szükségessé. A feldolgozás után a file-t fel kell szabadítani, ami a *Close* eljárással valósítható meg.

Példa: *Close* (bevitel)

A *Close* eljárás használata a kiviteli file-oknál feltétlenül szükséges, mert különben a belső puffert nem üríti ki és a tartalomjegyzék-bejegyzéseket nem zárja le, s végül ebben az esetben a file üres marad.
Szövegfile-oknál általában 128 byte-os belső pufferrel dolgozunk. Bővíthető, ha a file-változó definíciója mögött megadjuk a kívánt puffernagyságot. Minél nagyobb a puffer, annál kevesebb lemezműveletet kell végezni a *Turbo–Pascal*-ból.

Példa: **VAR** bevitel: *text* [512]

Az alkalmazáshoz az eddig megtárgyalt lehetőségeken kívül van néhány függvény, amelyek a be/kivitel státuszáról felvilágosítást adnak.

a) *File-vég*
A file-vég az *Eof* függvénnyel lekérdezhető.
Példa: if *Eof* (bevitel) **then** *Close* (bevitel)
Ahogy a példából látható az *Eof* Boolean-függvény, értéke *igaz* (true), ha a file végére ért, különben *hamis* (false). A szövegfile végét akkor érjük el, ha a következő beolvasott karakter^Z.
Az *Eof* függvény egy változata a *SeekEof*. Ez már akkor is file-véget jelez, ha már csak *szóköz, tabulátor* vagy *CR és LF* van a file végéig.

Példa: if *SeekEof* (bevitel) **then** *Writeln* ('nincs több adat');

b) *Sorvég*
A fenti függvényhez hasonlóan működik az *Eoln* és a *SeekEoln*. Mindkettő a sor végét állapítja meg, ha a következő beolvasott karakter a *CR*. A *SeekEoln* a szóközöket és tabulátorokat átugorja.
A következő programpélda a *SeekEof* függvény alkalmazását mutatja, egy szövegfile szavait számlálja meg feltéve, hogy a szövegben nincs üres sor, csak utána. Az algoritmus a normális *Eof* függvénnyel nem működne, mert az az üres sorokat is szavaknak számlálná.

```
{ SZAVAK.PAS    }
{ Pr. I. 3.12.  }

FUNCTION Szavak (var f : t_dat)' : integer;

VAR
   w,
   i     : integer;
   sor   : string[255];
   blank : boolean;
   ch    : char;

BEGIN
   Reset(f);
   w:=Ø;
   blank:=true;
   while not Seekeof(f)
    do begin
      Readln(f,sor);
      for i:=1 to Length(sor)
       do begin
         ch:=sor[i];
         if (ch = ' ') and (not blank)
```

```
            then begin
              w:=w+1;
              blank:=true;
            end
            else if ch <> ' ' then blank :=false;
          end;
        if not blank then w:=w+1;
      end;
    szavak:=w;
    Close(f);
END; {.....................................}
```

A szöveg-file-ok feldolgozásához további függvények és eljárások állnak rendelkezésre

a) A meglevő file-ok bővítésére az *Append* eljárás alkalmazható, amely nem csinál mást, mint a file-mutatót a file végére állítja. Ezután sorokat fűzhetünk a file-hoz. Az eljárást a *Rewrite* helyett használjuk, amely a file-mutatót a kezdetre állítaná.

Példa: *Append* (kivitel);
Write (kivitel, 'File vége');

b) A file-puffer kiürítése a *Flush* eljárással valósítható meg.

Példa: *Flush* (kivitel);

Egységfile-ok

Egy készülék a *DOS*-ban file-nak számít, amelyet előre definiált név azonosít. Ezzel a kivitel átirányítása egyszerűen a logikai név megváltoztatásával lehetséges (l. a *COPY* parancsot a *DOS*-ban). A *Turbo–Pascal* a készülékfile-okkal lemezes file-ként foglalkozik, egy kivétellel: ha készülékfile-t nyitunk meg, nincs pufferhasználat (kivétel: *CON*). Ez azt jelenti, hogy jel-jel után kerül kivitelre (ami a nyomtató esetében ésszerű is). Bizonyos esetekben kívánatos a puffer használata. Ezt a *D* fordítóprogram-paraméter segítségével érhetjük el.

Beállítás:
D+ nincs pufferhasználat az egységfile-hoz (alapbeállítás);
D– a készülékfile-hoz pufferhasználatot rendel.

Az alábbi készülékneveket a *DOS* használja.

CON *CONSOLE:* a bevitel a billentyűzetről történik, a kivitel a képernyőn jelenik meg. Ennél a készülékfile-nál a bevitel pufferhasználattal történik, így lehetséges az editálás, amíg le nem nyomjuk a RETURN-t. Speciális szerkesztőbillentyűk:
– ezek a *DOS*-változattól függenek, de legtöbbjük általánosan működik.

Backspace,
DEL vagy ^*H*: a cursor egy hellyel balra lép és kitörli az ott található karaktert;
^*Z:* lezárja a bevitelt és *Eof* jelzést tesz a végére. Amennyiben nincs minden változónak értéke az olvasási utasításban
– a *char* változókat ^*Z*-re állítja;
– a *string* változókat 0-ra (hossz=0) állítja;
– a *numerikus* változókat változatlanul hagyja.

LST *LISTER:* a *DOS*-ban hozzárendelt nyomtatót kezeli.

AUX *AUXILIARY:* kisegítő berendezés mind a bevitel, mind a kivitel megvalósítható (általában COM1).

NUL *NULL:* speciális file, amely nem tárol adatot, ha ezen keresztül megy a kivitel, és nem ad adatot, ha rajta keresztül olvasunk be.

A *NUL*-file segítségével eltüntethetünk akár egy egész adatfolyamot hibajelzés nélkül. Ez akkor ésszerű, ha pl. valaki nem kíván mindig új listákat létrehozni, ekkor a program összes parancsát meghagyja és ideiglenesen *Assign*-t készít a *NUL* file-ra. A *Write* utasítások működnek, de semmit sem írnak ki. Az adatokat az operációs rendszer úgymond lenyeli. Természetesen a *NUL* file-ok *Input*-ként is felhasználhatók, ebben az esetben az EOF azonnal teljesül, a *Reset* nem jelez hibát.

A *Turbo–Pascal*-ban a következő kiegészítő egységfile-ok léteznek.

CON: *CONSOLE* a bevitel a billentyűzetről (Echoval), a kivitel a képernyőn keresztül történik. Ennél az egységfile-nál a bevitel pufferhasználattal valósul meg (128 byte) és így lehetséges az editálás mindaddig, amíg *Return-t* nem nyomjuk le.

A speciális szerkesztőbillentyűk:
általában a *Backspace* és az *ESC* billentyűk minden egyes billentyűzet tartozékai, de a *Control* billentyű és egy betű együtt használva helyettesítheti azokat.

Backspace,
DEL vagy ^*H:* a cursor egy hellyel visszalép és az ott levő karaktert törli;
ESC vagy ^*X:* a cursor a sorkezdetre megy, minden bevitt karaktert töröl;
^*D:* az utolsó bevitt karakterlánc karakterenkénti kiíratása;
^*R:* az utolsó bevitt sor kiíratása;
^*Z:* a bevitt sor befejezése és az EOF jelzés beállítása, ha nincs minden változónak értéke az olvasási utasításban, akkor
– a *char* változókat ^Z-re állítja,
– a *string* változókat 0-ra (hossz=0) állítja,
– a numerikus változókat változatlanul hagyja.

LST: *LISTER* a *DOS*-ban hozzárendelt nyomtatót kezeli.

AUX: *AUXILIARY* segédberendezés, lehetséges akár bevitel, akár kivitel ezen a berendezésen keresztül.

TRM: *TERMINAL* a bevitel a billentyűzetről (Echoval), a kivitel a képernyőre történik. A szerkesztési lehetőség ugyanúgy létezik, minta **CON:** esetében.

KBD: *KEYBOARD* a bevitel a billentyűzetről történik (Echo nélkül). Nincs szerkesztési lehetőség.

USR: *USER* egy a felhasználó által meghatározott be/kiviteli egységet fog használni. Általában a bevitelhez a **KBD:** és a kivitelhez a **CON:** használatos.

INP: *INPUT* a DOS standard beviteli file-ra vonatkozik, ha megegyezik a **CON:**-nal (ez a normális eset), akkor ugyanaz a hatás, mint a **CON**-nál (kettőspont nélkül). A bevitel közvetlenül a *DOS*-on keresztül történik a *Turbo–Pascal* speciális szerkesztőfunkciói nélkül.

OUT: *OUTPUT* a DOS standard kiviteli file-ra vonatkozik. Gyakorlatilag megegyezik a **CON:**-nal.

ERR: *ERROR* a DOS Standard hibafile-ra vonatkozik, ha megegyezik a **CON:**-nal, a hibákat a képernyőn adja ki.

Figyelem: a *kettőspont* fontos, erről tudja a *Turbo–Pascal,* hogy a saját egységfile-ja és nem a DOS-é.
A felhasználó az összes egységfile-hoz tetszés szerint illesztheti a *Turbo–Pascal* be/kimeneti meghajtóját igényei szerint. Különösen alkalmas ehhez az **USR:** készülék. Az illesztéskor a saját eljárásofszetje az előre meghatároz ott címmutatóba kerül. Természetesen két meghajtó létezik minden olyan készülékhez, amely be- és kivitelre egyaránt felhasználható aszerint, hogy olvasó- vagy íróeljárást hívunk meg. A *Turbo–Pascal* meghajtói lényegében a megfelelő DOS-funkciók (Interrupt $21) meghívásából állnak.

Az alábbi táblázat áttekintést ad a standard meghajtók címmutatóiról. Minden eljárásnak *Char* típusú értékparamétere van és a kiviteli jelet ezen keresztül kapja meg. A címmutatónak azonos a neve a meghajtóval és még egy *Ptr* toldatot kap.

Meghajtó	Használja	Típus	DOS–CALL
ConIn	**CON:, TRM:, KBD:**	Char funkció	$8
ConOut	**CON:**	Eljárás	$2
LstOut	**LST:**	Eljárás	$5
AuxOut	**AUX:**	Eljárás	$4
AuxIn	**AUX:**	Char funkció	$3
UsrOut	**USR:**	Eljárás	$2
UsrIn	**USR:**	Char funkció	$8
ConSt	*Keypressed*	Boolean funkció	$11

A *ConSt* meghajtó különlegessége, hogy ezt a boolean funkciót nem készülékhez, hanem a *billentyűfigyelés* (keypressed) funkcióhoz rendelték.
A következő program saját be/kiviteli meghajtó használatára példa. A *TraceOn* és *TraceOff* eljárások segítségével lehetővé teszi, hogy a képernyőkivitelt egy file-ba jegyzőkönyvezzük. Az új meghajtó neve *New_ConOut.*
A képernyőre kivitelhez rövid időre a standard meghajtót használjuk. Ez pl. az általunk megírt meghajtó rutinban lévő *Write* utasításnál szükséges, mert különben az rekurzív híváshoz vezetne (a *Write* pontosan a meghajtót hívná újra stb.).

```
{ NYOMKOV.PAS  }
{ Pr. I. 3.13. }

PROGRAM Nyomkovetes;

TYPE
    t_nev = string[12];

VAR
    bs_file          : text;
    regi_conoutptr,i : integer;

PROCEDURE Uj_ConOut (c : char);

BEGIN
   Write(bs_file,c);
   conoutptr:=regi_conoutptr;
   Write(c);
   conoutptr:=Ofs(Uj_ConOut);
END; {.....................................................}

PROCEDURE TraceBe (bs_file_nev : t_nev);
```

```
BEGIN
   Assign(bs_file,bs_file_nev);
   Rewrite(bs_file);
   regi_conoutptr:=conoutptr;
   conoutptr:=Ofs(Uj_ConOut);
END; {.....}

PROCEDURE TraceKi;

BEGIN
   Close(bs_file);
   conoutptr:=regi_conoutptr;
END; {.....}

BEGIN
   TraceBe('BS.DTA');
   for i:=1 to 1Ø do Writeln(i:5,i*i:5);
   TraceKi;
END.
```

Input és *Output*

Minden Pascal változatban két egységfile van standardként előre meghatározva, az *Input* és az *Output*. Ezeket nem kell az első programsorokban definiálni. Arra való, amint azt nevük is mutatja, hogy a billentyűzetről a bevitelt és a képernyőn keresztüli kivitelt megvalósítsák. Ha a be/kiviteli eljárásokban nincs file-megadás, akkor az olvasóeljárásnál mindig *Input*-tal lefoglalt beviteli file-t és a kiviteli eljárásnál mindig *Output*-tal lefoglalt kiviteli file-t nyit meg.

Az *Input*-tal megvalósított bevitel lehetőségei:

Ha hiányzik a beviteli file megadása, akkor a *Turbo–Pascal* másképp viselkedik, mint amikor az input specifikálva van. Ez a *B* fordítóprogram-paraméterrel is elérhető, úgy hogy az input megadását mindenféleképpen megtakaríthatjuk.

Beállítás:

B+ mint input megadása nélkül (standard beállítás);

B– mint input megadással.

Az alábbi áttekintés összefoglalja a különbségeket:

B–, *Input:* a ^*Z*-t EOF-ként értelmezi, az olvasóutasításban felsorolt változókat kötelezően beolvassa. A *Carriage Return*-t nem követi *Echo*.

B+: a ^*Z*-t nem értelmezik EOF-ként, az olvasóutasításban felsorolt változók, beolvasása nem kötelező. A *Carriage Return*-t *Echo* követi.

A *Readln* esetében természetesen a *Carriage Return*-t mindig *Echo* követi a beállítástól függetlenül.

Bevitelkor előfordulhat, hogy csak meghatározott hosszúságú változókat szeretnénk megengedni. Legtöbbször a *string* változóknál. Erre a célra az előre definiált *Buflen* változót alkalmazhatjuk. Mint már előbb említettük, az *Input* egységfile-hoz 128 byte-os puffer tartozik, amely megengedi a szerkesztést. Ennek a puffernek a kívánt hosszát a *Buflen*-nel adjuk meg. A változót a beolvasási parancs előtt kell a kívánt értékre beállítani. A beolvasás után a változó értéke ismét 128 lesz.

Példa: *Buflen:*=12;
Readln (file-név);

Az újabb *DOS* verziókban lehetséges az úgynevezett *Piping*. Ez azt jelenti, hogy – mivel a készülékeket mint file-okat kezeli –, lehetséges a be- és kiviteli file-okat a program behívásakor specifikálni. Ehhez azonban a fordító támogatása szükséges. Több program hívható egymás után láncolva, pl. a kivitel az első, a bevitel a második program stb.

A *Turbo–Pascal*-ban ez a *G* és *P* paraméterekkel érhető el, a *G Get*, a *P* pedig *Put* jelentésű.

Beállítás:

GO az *Input*-tal kezelt beviteli file **CON:**, a puffer 128 byte (standard beállítás);
Gn a puffermagyságot az *n*-nel adjuk meg, a beviteli file az **INP:** (a *DOS*-ban specifikált standard beviteli file);
PO az *Output*-tal kezelt kiviteli file a **CON:**, a puffer Ø byte (standard beállítás);
Pn a puffermagyságot az *n*-nel adjuk meg, a kiviteli file az **OUT:** (a *DOS*-ban specifikált standard kiviteli file).

A paramétereket a programban elsőként kell alkalmazni, ezenkívül a *D–* fordítóparamétert is meg kell adni a fordításkor, mert különben nincs pufferhasználat (kivétel **INP:**). A következő program beolvas egy *Turbo–Pascal* programot, majd feloldja az include file-hivatkozásokat. Ez segíti a be/kivitel átirányítását. A II. rész 1. fejezetében a *hasításos keresés* leírásához mellékelt nyomtatóprogrammal úgy kombinálható a 3.14. programpélda, hogy ennek a programnak a kimenete egyben a nyomtatóprogram bemenete legyen, s így az include file-okat tartalmazó programok is nyomtathatók.

```
{ FILEINCL.PAS }
{ Pr. I. 3.14. }

{$G512,P512,D-}

PROGRAM File_Beszur;

VAR
   f      : text;
   sor    : string[80];
   i      : integer;
   nev    : string[12];

BEGIN
  while not Eof
   do begin
     Readln(sor);
     Writeln(sor);
     i:=Pos('{$I',sor);
     if i=1
      then begin
        i:=4;
        While (sor[i] = ' ') and (i < length(sor)) do
              i:=i+1;
        nev:='';
        While (sor[i] <> ' ') and (sor[i] <> '}') and
              (i < length(sor))
          do begin
            nev:=nev+sor[i];
            i:=i+1;
          end;
          Assign(f,nev);
          Reset(f);
          while not Eof(f)
           do begin
             Readln(f,sor);
             Writeln(sor);
           end;
           Close(f);
        end;
    end;
END.
```

File-illesztő blokk (File Interface – FIB)
A következő leírás a lemezes file-okra és a szövegfile-okra egyaránt vonatkozik.
A *FIB* közvetlenül a file-változókhoz van rendelve, azaz a file-változó a *FIB*-nek a neve. Ofszetként definiálva a változó helyét adja meg a tárban. Fogadjuk el az alábbi definíciókat:

```
TYPE t_fib = record
               fiel-jel: integer;
               kapcsoló_byte,
               karpuffer: byte;
               puffer_offset,
               puffer_nagysag,
               puffer_mutato,
               puffer_vege: integer;
               adat_ut: array [1..63] of char;
               end;
```

Az egyes mezők leírása:

file_jel — ha a file-t *Reset*-tel vagy *Rewrit*-tal nyitottuk meg, akkor itt a DOS file-megjelölése, lezárt file-nál pedig 4351 ($FFFF) áll;

kapcsoló_byte — az első bit a file-típust adja meg
0 lemezes-file-t jelent
1-től 5-ig készülék file-t jelent
1 – *CON:, TRM:*
2 – *KBD:*
3 – *LST:*
4 – *AUX:*
5 – *USR:*
az 5. bit az előolvasás kapcsolója (1 = igen)
a 6. bit 1-es értéke kiviteli file-t jelent
a 7. bit 1-es értéke beviteli file-t jelent;

karpuffer — csak olyan egység file-oknál kell megadni, amelyeknél az alapértelmezés szerint nincs puffermegadás;

puffer_offset — a puffertároló kezdete, a puffertároló *FIB* szegmensében található;

puffer_nagyság — a puffer mérete byte-okban;

puffer_mutató — a puffer aktuális byte-jára mutat;

puffer_vége — a puffer végét ofszetként adja meg;

adat_ut — egyértelmű keresési út meghajtóval, könyvtárnévvel és file-névvel, ahogy azt az *Assign*-ban megadtuk.

```
VAR
    fib : t_fib;
      f : text absolute fib:
```

Az *absolute* megadással a *FIB* változó kezdetét ugyanarra a tárhelyre tesszük, mint az *f* változót. A *fib* változó definíciója ezzel egyben a *FIB*-et is jelenti.
A következő program a *FIB* saját célra való felhasználását mutatja. Egy olyan eljárásról van szó, amely egy file-változót kap paraméterenként és bizonyos információkat kiolvas a *FIB*-ből. Itt másképp nyúltunk a *FIB*-hez, mint az előbb leírt módszernél, most az előre definiált *Mem* függvényt hívtuk segítségül.

```
{ FILEINFO.PAS }
{ Pr. I. 3.15. }

PROCEDURE File_info (VAR f: text);

VAR
   i, start_seg, start_ofs : integer;

BEGIN
  Writeln('File-Informacio a FIB-bol');
  Writeln('-------------------------');
  Writeln;
  Write('Nev ');
  i:=12;
  start_seg:=Seg(f);
  start_ofs:=Ofs(f);
  while Mem[start_seg:start_ofs+i] <> Ø
   do begin
     Write(chr(Mem[start_seg:start_ofs+i]));
     i:=i+1;
   end;
  Writeln;
  Writeln;
  Write('A file egy ');
  start_ofs:=start_ofs+2;
  case Mem[start_seg:start_ofs] and 7 of
    Ø   : Write('lemezes file, ');
    1..5: Write('file-kent kezelt egyseg, ');
  end;
  Writeln;
  Writeln;
  Write('amely nyitott: ');
  if Mem[start_seg:start_ofs] and 6 = 1
   then writeln('input-ra')
   else writeln('output-ra');
END; {.....}
```

3.4.1.2. Strukturált file-ok

Ezek a file-ok a *Standard–Pascal*-ból már ismertek, nincs *ASCII* formátumuk és ezért csak lemezes file-ként lehetséges a felhasználásuk. Az ilyen file-ok alkalmazásának két nagy előnye van:

- gyorsabb, mivel nincs adatkonverzió;
- kisebb helyigényű, mert az adatok a belső ábrázolásuk szerint kerülnek rögzítésre.

A szöveges file-oknál megismert függvények és eljárások közül itt is használható:

- az *Assign;*
- a *Reset, Rewrite;*
- a *Read, Readln, Write, Writeln;*
- az *Eof;*
- a *Close* és
- a *Flush.*

Továbbá, van néhány olyan függvény, amely a strukturált file-ok kezelését megkönnyíti:

- a *Filesize* függvény a file méretét adja meg, megszámlálja a file-t felépítő struktúrák előfordulásait;
- a *Filepos* függvény megadja a file feldolgozásra kerülő aktuális elemének pozícióját.

S végül a strukturált file-okhoz tartozó *FIB* definícióját adjuk meg.

File-csatoló blokk (FIB)

Ez erősen különbözik a *text* file-ok *FIB*-jétől.

```
TYPE t_fib = record
               file_jel:   integer;
               rec-hossz:  integer;
               filler:     array [1..5] of integer;
               adat_ut:    array [1..63] of char;
               end;
```

file_jel — ha a file-t *Reset*-tel vagy *Rewrite*-tal nyitottuk meg, akkor itt a *DOS* megjelölése, lezárt file-nál pedig 4351, azaz ($FFFF) áll;

rec_hossz — a file rekordjának hossza byte-okban megadva;

filler — a strukturált file-oknál nincs pufferhasználat;

adat_ut — egyértelmű keresési út meghajtóval, könyvtárnévvel és file-névvel, ahogy azt a *Assign*-ban megadtuk.

Most a file-változó és a *FIB* kezdetét ismét egymásra definiáljuk:

```
VAR
    fib : t_fib;
      f : t_fib absolute t_fib;
```

A következő programpéldában a *File_méret* függvény meghatározza egy strukturált file méretét a *FIB* felhasználásával.

```
{ FILESIZE.PAS }
{ Pr. I. 3.16. }

PROGRAM File_meret;

LABEL 99;

TYPE
   t_szemely   = record
                   nev     : string[3Ø];
                   cim     : string[5Ø];
                   telefon : string[1Ø];
                 end;

   t_szemfile = file of t_szemely;

VAR
   szem_file : t_szemfile;
   szemely   : t_szemely;

FUNCTION File_meret (VAR f: t_szemfile) : integer;

BEGIN
  if MemW[seg(f):ofs(f)+4] = Ø
   then File_meret:=MemW[seg(f):ofs(f)+2]*filesize(f);
END; {.....................................................}


BEGIN
  Assign(szem_file,'Szemelyek.dta');
  Rewrite(szem_file);
  ClrScr;
  Writeln('Szemelyi adatok felvetele');
  Gotoxy(1,4);
  Write('Befejezesul nevkent "VEGE"-t kell megadni');
  Gotoxy(',8);
  Write(' ev     :');
  Gotoxy( ,1Ø);
```

```
    Write('Cim     :');
    Gotoxy(1,12);
    Write('Telefon:');
    Gotoxy(1,16);
    Write('A file merete    :');
    repeat
      with szemely
       do begin
         Gotoxy(10,8);
         Read(nev);
         if nev = 'VEGE' then goto 99;
         Gotoxy(10,10);
         Read(cim);
         Gotoxy(10,12);
         Read(telefon);
       end;
      Write(szem_file,szemely);
      Gotoxy(19,16);
      Write(File_meret(szem_file));
      Gotoxy(10,8);
      ClrEol;
      Gotoxy(10,10);
      ClrEol;
      Gotoxy(10,12);
      ClrEol;
    until 1 <> 1;
    99:ClrScr;
END.
```

3.4.1.3. Típus nélküli file-ok

Néha szeretnénk lemezes file-okkal dolgozni anélkül, hogy azok pontos szerkezetét ismernénk. Pl. file-másoláskor vagy egyszerűen a file-törlésekor. Erre a célra a *Turbo–Pascal* egy különlegessége, a *típus nélküli file-ok* használhatók.
Normális esetben a *Turbo–Pascal* felvesz egy szektorpuffert, amelyet a lemezről 128 byte-tal tölt fel. Ebből a pufferből az információk, a file struktúrájának megfelelően, rekordonként kerülnek átvitelre. A típus nélküli file-oknál ez a puffer elmarad, a rekordhossz alapértelmezés szerint 128 byte, az átvitel közvetlenül az adatterületre kerül. Természetesen ehhez másik olvasó-, íróeljárást készítettek.
A tárban való helymegtakarítás mellett további előnyt nyújtanak a típus nélküli file-ok, a minden más file-típusnál lényegesen gyorsabb adatátvitellel.

Az alábbi eljárásokat és függvényeket alkalmazhatjuk:
- *Assign*
- *Reset, Rewrite*
- *Eof*
- *Filesize*
- *Filepos*
- *Close*

A típus nélküli definíciója a file-típus elhagyásával valósul meg.

Példa: **VAR** adatok: *file;*
Az átvitelhez a *Blockread* és a *Blockwrite* eljárások állnak a rendelkezésünkre.

a) *Blockread*
(paraméterek: file-változó, változó, rekordszám, eredmény)
- azt a típus nélküli file-t kell megadnunk, amelyből olvasni akarunk;
- a változóba az információkat az első byte-tól kiindulva tölti. Fontos, hogy a

változó elég nagy legyen a rekord befogadására, mert különben más változók kerülnek felülírásra;
- a 128 byte-os rekordok kívánt darabszáma;
- az eredményváltozó tartalmazza a ténylegesen átvitt rekordok darabszámát, megadása nem kötelező.

b) *Blockwrite*
(paraméterei azonosan a *Blockread*-éval)
A *FIB* a típus nélküli file-oknál ugyanolyan, mint a strukturált file-oknál. Az ott elmondottak itt is érvényesek.

A 3.17.-es *Konvert* nevű program a file-típusok megváltoztatására való, egyben a *Blockread* és *Blockwrite* eljárások alkalmazását mutatja.

A program működése röviden:
különbséget tesz a szövegfile-ok (változó rekordhossz) – amelyekben az egyes rekordokat *CR* és *LF* választja el – és a közvetlen hozzáférésű file-ok (fix rekordhossz) között.
Az egyik file-formáról a másikra való áttérés lehetőségei:

a) *fix file-ról fix file-ra*
- ha a beviteli rekordhossz nagyobb mint a kimenő rekordhossz, akkor a program átviszi a rekordot (változás nélkül) amíg lehet és levágja azokat a karaktereket, amelyek már nem férnek el a kimeneti rekordban ellenben,
- ha a beviteli rekordhossz kisebb, vagy azonos a kimenő rekordhosszal, akkor az esetleges fölös helyeket a kimenő rekordban szóközökkel tölti fel;
- ezek *1:1* arányú átvitelnek felelnek meg.

b) *fix file-ról változó file-ra*
- ha nagyobb a beviteli, mint a kiviteli rekordhossz, akkor a bemenő rekordot a kimenő rekordba addig a karakterig helyezi el, ameddig belefér, a kimenő rekord feltöltése után *CRLF*-et fűz hozzá és új kimenő rekordot kezd mindaddig, amíg a beviteli rekord el nem fogy és csak ezután kezd a következő beviteli rekord feldolgozásához;
- ha a beviteli rekordhossz kisebb, vagy azonos a kiviteli rekordhosszal, a kiviteli rekordot teljesen feltölti a beviteli rekordokkal és azután *CRLF*-fel kiírja, az utolsó, teljesen át nem vitt beviteli rekord maradéka a következő kiviteli rekordba kerül;
- az átviteli arány tehát ebben az esetben *1: m*-hez.

c) *változó file-ról fix file-ra*
- ha nagyobb a beviteli rekordhossz, mint a kiviteli rekordhossz, akkor az átviteli arány *1 : m*-hez, egy beviteli rekord *m* darab kiviteli rekordba (fix hosszúságú) kerül, a *CRLF*-et nem veszi figyelembe;
- ellenkező esetben a program *n* beviteli rekordot olvas be, eltünteti a *CRLF*-et és összefűzi egy kiviteli rekordba.

d) *változó file-ról változó file-ra*
- az átviteli arány *1 : m*-hez, ha a beviteli rekordhossz a nagyobb;
- ellenkező esetben a program *n* beviteli rekordot olvas be, eltünteti a *CRLF*-et, a rekordokat összefűzi egy kiviteli rekordba, CRLF-tel ellátja és kiírja.

A paramétereket megadhatjuk közvetlenül a program meghívásakor, vagy később a menü kitöltésével.

Példa: konvert imp, 10F,out,10V

Jelentés szerint, egy 10-es fix rekordhosszal rendelkező *imp* beviteli file-t 10-es változó rekordhosszú *out* kiviteli file-re konvertálja.

```
{ KONVERT.PAS  }
{ Pr. I. 3.17. }

{$U-}
{$C-}

PROGRAM KONVERT;

{ FIX_FIX  sli >  slo  1:1 levagassal                    }
{          sli <  slo  1:1 szokoz feltoltessel           }
{                                                        }
{ VAR_FIX  sli <= slo  n:1 osszefuzessel                 }
{          sli >  slo  1:m darabolassal                  }
{                                                        }
{ VAR_VAR  sli <= slo  n:1 osszefuzessel                 }
{          sli >  slo  1:m darabolassal                  }
{                                                        }
{ FIX_VAR  sli <= slo  1:1 ^M^J hozzafuzesevel           }
{          sli >  slo  1:m darabolassal es               }
{                      ^M^J hozzafuzessel                }

CONST
   Verzio    = 'Ver1.ØØ';
   inp       = true;
   outp      = false;
   c_var_max = 25Ø;
   c_fix_max = 25Ø;

TYPE
   t_sor = string[c_var_max];
   t_jel  = (olvasott,irt);

CONST
   z          : array[1..4] of t_sor =
     ('Input allomany  . . . . . ?          ',
      ' Rekordhossz<1V..25ØV/1F..25ØF> . . ?          ',
      'Output allomany . . . . . ?          ',
      ' Rekordhossz<1V..25ØV/1F..25ØF> . . ?          ');


VAR
   ch       : char;
   cmd,
   fni,
   s_sli,
   fno,
   s_slo,
   v,
   v2       : string[c_var_max];
   f,
   f2       : array[Ø..c_fix_max] of char;
   i_i,
   i_o      : real;
   i,
   j,
   l,
   sli,
   slo,
   vor,
   tat      : integer;
   chi,
```

```
    cho     : char;
    fvi,
    fvo     : text[$5000];
    ffi,
    ffo     : file;

PROCEDURE Hiba (ErrNo, ErrOfs : Integer);

BEGIN
   GotoXY(1,13);
   HighVideo;
   Writeln('*** KONVERT felbeszakitva ***',Chr(7));
   Halt;
END; {.....................................................}

FUNCTION Olv_ch : char;

VAR
   ch : char;
   r  : record
          ax,bx,cx,dx,
          bp,si,di,
          ds,es,
          flags : integer;
        end;

BEGIN
   r.ax:=$0600;
   r.dx:=$00FF;
   MsDos(r);
   if Lo(r.dx)=$FF
    then ch:=Chr(Lo(r.ax))
    else ch:=#0;
   if ch=#3
    then Hiba($0100,0);
   Olv_ch:=ch;
END; {.....................................................}

PROCEDURE Jel (i : real;
               x : t_jel);

BEGIN
   if x = olvasott
    then GotoXY(10,11)
    else GotoXY(31,11);
   Write(i:5:0);
   ch:=Olv_ch;
END; {.....................................................}

FUNCTION Max (a,b : integer) : integer;

BEGIN
   if a > b
    then max:=a
    else max:=b;
END; {.....................................................}

FUNCTION Min (a,b : integer) : integer;

BEGIN
   if a < b
    then min:=a
    else min:=b;
END; {.....................................................}

FUNCTION Elemez (VAR str     : t_sor;
                     pattern : char   ): t_sor;
```

```
VAR
   i : integer;
   s : t_sor;

BEGIN
   while (str[1] = ' ') and (length(str) > Ø)
    do Delete(str,1,1);
   i:=Pos(pattern,str+pattern);
   if i <> Ø
    then begin
      Elemez:=Copy(str,1,i-1);
      Delete(str,1,i);
    end
    else Elemez:='';
END;

FUNCTION Sl_vizsg  (     s_sl   : t_sor;
                     VAR ch     : char) : integer;
VAR
   i,
   sl : integer;

BEGIN
   Val(s_sl,sl,i);
   ch:=s_sl[Length(s_sl)];
   if ch in ['V','v']
    then begin
      if not sl in [1..255] then sl:=Ø
    end
    else if ch in ['F','f']
          then begin
            if not sl in [1..255]
             then sl:=Ø;
          end
          else begin
            ch:='?';
            sl:=Ø;
          end;
   ch:=UpCase(ch);
   Sl_vizsg:=sl;
END; {.....................................................}

PROCEDURE IO_vizsg (IO_in : boolean;
                    fn    : t_sor;
                    IO_nr : integer);
BEGIN
   if IO_nr <> Ø
    then begin
      GotoXY(1,13);
      HighVideo;
      Write('*** File (',fn,') nem  talalhato ***',
            Chr(7));
      if IO_in then Halt;
    end;
END; {.....................................................}



PROCEDURE Fix_Var;

BEGIN
   Assign(ffi,fni);
   {$I-}
   Reset(ffi,sli);
   {$I+}
   IO_vizsg(inp,fni,IOresult);
```

```
    Assign(fvo,fno);
    {$I-}
    Rewrite(fvo);
    {$I+}
    IO_vizsg(outp,fno,IOresult);

    while not Eof(ffi)
     do begin
       Fillchar(f,c_fix_max,' ');
       Blockread(ffi,f,vor,tat);
       i_i:=i_i+tat;
       Jel(i_i,olvasott);
       for i:=Ø to tat-1
        do begin
          j:=Ø;
          while j < sli
           do begin
             l:=Min(slo,sli-j);
             Fillchar(v[1],c_var_max,' ');
             Move(f[i*sli+j],v[1],l);
             v[Ø]:=Chr(l);
             while (v[Length(v)]=' ') and
                   (Length(v) > Ø)
              do Delete(v,Length(v),1);
             Writeln(fvo,v);
             i_o:=i_o+1;
             j:=j+l;
             ch:=Olv_ch;
           end;
        end;
        Jel(i_o,irt);
     end;

    Close(ffi);
    Write(fvo,^Z);
    Close(fvo);
END; {.....................................................}

    PROCEDURE Fix_Fix;

    BEGIN
       Assign(ffi,fni);
       {$I-}
       Reset(ffi,sli);
       {$I+}
       IO_vizsg(inp,fni,IOresult);

       Assign(ffo,fno);
       {$I-}
       Rewrite(ffo,slo);
       {$I+}
       IO_vizsg(outp,fno,IOresult);

       while not Eof(ffi)
        do begin
          Fillchar(f,Sizeof(f),' ');
          Blockread(ffi,f,vor,tat);
          i_i:=i_i+tat;
          Jel(i_i,olvasott);
          for i:=Ø to tat-1
           do begin
             Fillchar(f2,Sizeof(f2),' ');
             Move(f[i*sli],f2[Ø],sli);
             Blockwrite(ffo,f2,1);
             i_o:=i_o+1;
             Jel(i_o,irt);
           end;
        end;
```

```
   Close(ffi);
   Close(ffo);
END; {.....}


PROCEDURE Var_Var;

BEGIN
   Assign(fvi,fni);
   {$I-}
   Reset(fvi);
   {$I+}
   IO_vizsg(inp,fni,IOresult);

   Assign(fvo,fno);
   {$I-}
   Rewrite(fvo);
   {$I+}
   IO_vizsg(outp,fno,IOresult);

   if sli <= slo
    then
     while not Eof(fvi)
      do begin
        i:=Ø;
        Fillchar(v2[1],c_var_max,' ');
        while i < slo
         do begin
           Fillchar(v[1],c_var_max,' ');
           Readln(fvi,v);
           i_i:=i_i+1;
           Jel(i_i,olvasott);
           Move(v[1],v2[i+1],min(Length(v),slo-i));
           i:=i+sli;
         end;
        v2[Ø]:=Chr(c_var_max);
        while (v2[Length(v2)]=' ') and (Length(v2)>Ø)
         do Delete(v2,Length(v2),1);
        Writeln(fvo,v2);
        i_o:=i_o+1;
        Jel(i_o,irt);
      end
    else
     while not Eof(fvi)
      do begin
        Fillchar(v[1],c_var_max,' ');
        Readln(fvi,v);
        i_i:=i_i+1;
        Jel(i_i,olvasott);
        i:=Ø;
        while i < sli
         do begin
           Fillchar(v2[1],c_var_max,' ');
           Move(v[i+1],v2[1],Min(slo,sli-i));
           i:=i+slo;
           v2[Ø]:=Chr(slo);
           while (v2[Length(v2)]=' ') and
                 (Length(v2) > Ø)
            do Delete(v2,Length(v2),1);
           Writeln(fvo,v2);
           i_o:=i_o+1;
           Jel(i_o,irt);
         end;
      end;
```

```
    Close(fvi);
    Write(fvo,^Z);
    Close(fvo);
END; {.....}

PROCEDURE Var_Fix;

BEGIN
    Assign(fvi,fni);
    {$I-}
    Reset(fvi);
    {$I+}
    IO_vizsg(inp,fni,IOresult);

    Assign(ffo,fno);
    {$I-}
    Rewrite(ffo,slo);
    {$I+}
    IO_vizsg(outp,fno,IOresult);

    if sli <= slo
     then begin
      while not Eof(fvi)
       do begin
         i:=Ø;
         Fillchar(f[Ø],Sizeof(f),' ');
         while i < slo
          do begin
            Fillchar(v[1],c_var_max,' ');
            Readln(fvi,v);
            i_i:=i_i+1;
            Jel(i_i,olvasott);
            Move(v[1],f[i],Min(Length(v),slo-i));
            i:=i+sli;
          end;
         Blockwrite(ffo,f,1);
         i_o:=i_o+1;
         Jel(i_o,irt);
       end;
     end
     else begin
       while not Eof(fvi)
        do begin
          Fillchar(v[1],c_var_max,' ');
          Readln(fvi,v);
          i_i:=i_i+1;
          Jel(i_i,olvasott);
          i:=Ø;
          while i < sli
           do begin
             l:=Min(slo,Length(v)-i);
             Fillchar(f[Ø],slo,' ');
             Move(v[i+1],f[Ø],l);
             i:=i+slo;
             Blockwrite(ffo,f,1);
             i_o:=i_o+1;
             Jel(i_o,irt);
           end;
        end;
     end;

   Close(fvi);
   Close(ffo);
END; {.....}
```

```
BEGIN
   ErrorPtr:=Ofs(Hiba);
   cmd:='';
   sli:=Ø;
   slo:=Ø;

   for i:=1 to ParamCount
    do cmd:=cmd+ParamStr(i);
   for i:=1 to Length(cmd)
    do cmd[i]:=UpCase(cmd[i]);

   fni  :=Elemez(cmd,',');
   s_sli:=Elemez(cmd,',');
   fno  :=Elemez(cmd,',');
   s_slo:=Elemez(cmd,',');

   sli:=Sl_vizsg(s_sli,chi);
   slo:=Sl_vizsg(s_slo,cho);

   ClrScr;
   HighVideo;
   Write('KONVERT  ',Verzio);
   LowVideo;

   GotoXY( 1, 3); Write(z[1],fni);
   GotoXY( 1, 4); Write(z[2],s_sli);
   GotoXY( 1, 6); Write(z[3],fno);
   GotoXY( 1, 7); Write(z[4],s_slo);

   if fni = ''
    then repeat
      Buflen:=35;
      GotoXY(45, 3);
      ClrEol;
      Read(fni);
    until fni <> '';

   if sli = Ø
    then repeat
      Buflen:=6;
      GotoXY(45, 4);
      ClrEol;
      Read(s_sli);
      sli:=Sl_vizsg(s_sli,chi);
    until (sli <> Ø) and (chi in ['F','V']);

    if fno = ''
     then repeat
       Buflen:=35;
       GotoXY(45, 6);
       ClrEol;
       Read(fno);
     until fni <> '';

    if slo = Ø
     then repeat
       Buflen:=6;
       GotoXY(45, 7);
       ClrEol;
       Read(s_slo);
       slo:=Sl_vizsg(s_slo,cho);
     until (slo <> Ø) and (cho in ['F','V']);

     vor:=c_fix_max div sli;

     GotoXY( 1, 9);
     Write('KONVERT ',fni,' (',sli,chi,') ---> ',
                      fno,' (',slo,cho,') ');
```

```
        GotoXY(1,11);
        Write('Olvasott: ');
        GotoXY(18,11);
        Write('Irt : ');

        i_i:=Ø.Ø;
        i_o:=Ø.Ø;

        GotoXY(1,1Ø);

        if chi='V'
         then if cho='F'
                then Var_Fix
                else Var_Var
         else if cho='F'
                then Fix_Fix
                else Fix_Var;
END.
```

3.4.2. Közvetlen hozzáférésű file-ok

Lehetővé teszik a *Turbo–Pascal*-ban programozónak, hogy a rekordokat tetszőleges sorrendben feldolgozza.
A dolog kulcsa a rekordszám, amellyel a rekord megtalálható, azonosítható.

3.4.2.1. Strukturált file-ok

Minden eddig említett függvény és eljárás, amely a soros, strukturált file-okhoz rendelkezésre áll, itt is használható. Ezekhez jön még a *Seek* eljárás, amellyel a file-mutatót kell a helyes pozícióra állítani, csak ezután lehet a kívánt rekordot beolvasni vagy kiírni.

Figyelem! A *Turbo–Pascal* a rekordokat egymás után, sorrendben tárolja. A programozónak kell arról gondoskodnia, hogy a file-t a tényleges rekordhosszal kezelje.

3.4.2.2. Típus nélküli file-ok

Közvetlen hozzáférésű file-ként is használhatók, itt a fix rekordhossz 128 byte. Minden, amit a soros, típus nélküli file-ról megállapítottunk, a közvetlen hozzáférésű file-okra is igaz.

A *Seek* eljárás a rekordok megtalálását segíti.

Az utolsó példánk ebben a fejezetben egy egyszerű programot mutat be, amely strukturált, közvetlen hozzáférésű file-t használ termékadatok feldolgozására. A file-t először inicializáljuk és minden rekordot mint még nem létezőt megjelölünk. A cikkszám a hozzáférési kulcs a rekordok bevitelére, ill. módosítására.

```
{ DIREKTFK.PAS }
{ Pr. I. 3.18. }

PROGRAM KozvEleres;

TYPE
   t_elnevezes = string[2Ø];

   t_cikk  = record
                sorsz        : integer;
                elnevezes    : t_elnevezes;
                ar           : real;
                raktarkeszl  : integer;
                letezik      : boolean;
             end;

   t_cikkfile = file of t_cikk;

VAR
   cikk_file   : t_cikkfile;
   cikk        : t_cikk;
   cikksorsz   : integer;
   valasz      : byte;

PROCEDURE init;

VAR
   i : integer;

BEGIN
   ClrScr;
   Write('Cikk allomany init ');
   for i:=1 to 1ØØØ
    do begin
     if i mod 1ØØ = Ø then Write('.');
     with cikk
      do begin
        sorsz:=i;
        letezik:=false;
      end;
     Write(cikk_file,cikk);
    end;
END;

PROCEDURE AdatMenu;

BEGIN
   GotoXY(1,4);
   Write('Az adatbevitel vegen  Cikksorszam=Ø ');
   GotoXY(1,8);
   Write('Cikksorsz       :');
   GotoXY(1,1Ø);
   Write('Elnevezes       :');
   GotoXY(1,12);
   Write('A               :');
   GotoXY(1,14);
   Write('Raktarkeszlet   :');
END;

PROCEDURE Sorok_torl;

BEGIN
   GotoXY(18,8);
   ClrEol;
   GotoXY(18,1Ø);
   ClrEol;
```

```
    GotoXY(18,12);
    ClrEol;
    GotoXY(18,14);
    ClrEol;
END;

PROCEDURE FeltoltMenu (cikk : t_cikk);

BEGIN
   with cikk
    do begin
      GotoXY(18,8);
      Write(sorsz);
      GotoXY(18,1Ø);
      Write(elnevezes);
      GotoXY(18,12);
      Write(ar);
      GotoXY(18,14);
      Write(raktarkeszl);
    end;
END;


PROCEDURE Modos;

LABEL 99;

VAR nev : t_elnevezes;

BEGIN
  ClrScr;
  Writeln('Cikk adatok modositasa');
  AdatMenu;
  repeat
     GotoXY(18,8);
     Read(cikksorsz);
     if cikksorsz = Ø then goto 99;
     Seek(cikk_file,cikksorsz);
     Read(cikk_file,cikk);
     if cikk.letezik then FeltoltMenu(cikk);;
     with cikk
      do begin
       GotoXY(18,1Ø);
       nev:=elnevezes;
       Read(elnevezes);
       if elnevezes = ''then elnevezes:=nev;
       GotoXY(18,12);
       Read(ar);
       GotoXY(18,14);
       Read(raktarkeszl);
       letezik:=true;
      end;
    Seek(cikk_file,cikksorsz);
    Write(cikk_file,cikk);
    Sorok_torl;
   until 1<>1;
   99:;
END;

PROCEDURE Lista;

VAR
   i : integer;

BEGIN
   ClrScr;
   Writeln('Cikk Lista');
```

```
   AdatMenu;
   for i:=1 to 999
    do begin
      Seek(cikk_file,i);
      Read(cikk_file,cikk);
      if cikk.letezik
       then begin
        FeltoltMenu(cikk);
        delay(1000);
       end;
    end;
END;
BEGIN
   Assign(cikk_file,'cikk.dta');
   repeat
     ClrScr;
     LowVideo;
     Writeln('Pelda kozvetlen eleresu file-ra ');
     Writeln('------------------------------ ');
     GotoXY(1,5);
     Write('Cikk adatok feldolgozasa');
     GotoXY(1,8);
     HighVideo;
     Write('1');
     LowVideo;
     Write(' Elokeszites');
     GotoXY(1,10);
     HighVideo;
     Write('2');
     LowVideo;
     Write(' Adatfeldolgozas');
     GotoXY(1,12);
     HighVideo;
     Write('3');
     LowVideo;
     Write(' Listazas');
     GotoXY(1,14);
     HighVideo;
     Write('4');
     LowVideo;
     Write(' Vege');
     GotoXY(1,18);
     Write('Valasztas --> ');
     HighVideo;
     repeat
       GotoXY(15,18);
       ClrEol;
       Read(valasz);
     until valasz in [1..4];
     case valasz of
       1: begin
            Rewrite(cikk_file);
            Init;
            Close(cikk_file);
          end;
       2: begin
            Reset(cikk_file);
            Modos;
            Close(cikk_file);
          end;
       3: begin
            Reset(cikk_file);
            Lista;
            Close(cikk_file);
          end;
       else;
     end;
  until valasz = 4;
END.
```

II. RÉSZ

GYAKORLAT

1. Rendezés és kezelés

A rendezés és a keresés a programozás gyakori részproblémái. Egy nem megfelel megoldás a programot jelentősen lelassítja (és ki akar ma lassú programot írni?). A utóbbi harminc évben nagyszámú rafinált kereső- és rendezőelvet fejlesztettek ki.

Könyvünkben ezt a terjedelmes ismeretanyagot nem közölhetjük kimerítően. A érdeklődő olvasónak *D.E. Knuth* [5] művét ajánljuk, aki a több mint 700 oldala könyvében ezt a témát a legrészletesebben tárgyalja.

Ebben a fejezetben arra összpontosítottunk, hogy a véleményünk szerinti leggyorsabb rendező és kereső algoritmusokat előállítsuk és ezek programjait *Turbo–Pascal*-ban kifejlesszük úgy, hogy közben csak tömb array adatszerkezetre szorítkozunk.

Az algoritmusok analízise matematikailag általában tökéletes, arra törekedtünk, hogy egyszerűbb meggondolásokat alkalmazzunk, hogy a matematikában kevésbé gyakorlott Olvasó számára se jelentsen nehézséget megértésük. Akiket azonban elsősorban a programok érdekelnek, természetesen felhasználhatják azokat anélkül, hogy az analízist el kellene olvasniuk.

1.1. RENDEZŐ ELJÁRÁSOK

1.1.1. Alapvető meggondolások

Mielőtt az egyes algoritmusokat megvizsgálnánk, a problémát kell pontosan meghatároznunk és néhány elvi kérdést meggondolnunk.
Definiáljunk egy tömböt:

```
VAR s :array   1..c_max   of record
                            key   :t_key;
                            info  :t_infor;
                          end;
```

A c_max olyan nagy legyen, hogy a tömb illeszkedjen a rendelkezésre álló tárterülethez. A *key* változó a kulcsot tartalmazza, ez a mértékadó a rendezésben, az *info* változó a további információkat jelképezi, amelyek a rendezéshez jelentéktelenek. Feladatunk olyan eljárást keresni, amely a kulcsra növekvő sorrendet állít elő a tömbben. A rendezés után megvalósul:

$$s[1] \cdot key <= s[2] \cdot key <= \ldots <= s[c\_max] \cdot key$$

Néhány osztályba soroljuk a könyvünkben tárgyalásra kerülő eljárásokat:
– beszúrás,
– kicserélés,
– összeválogatás,
– kiválasztás.
Ez az osztályozás szubjektív, de egy abszolút osztályozás aligha valósítható meg. A

Shellsort példából is kitűnik, hogy néhány algoritmus több osztályhoz is tartozhat aszerint, hogy milyen nézőpontból vizsgáljuk a működésüket.

Az egyes osztályok rövid leírása.

– beszúrás, az *i*-edik elemet az *(i–1)*-edik, már rendezett elem után helyezzük el (1 <= i <= *c_max*);

l az 1.1.2 pontot – rendezés beszúrással,
l az 1.1.3 pontot – Shell rendezés (Shellsort),

– kicserélés, azokat a kulcspárokat, amelyeket még nem rendeztünk addig cserélgetjük, amíg többé nem találhatók ilyen párok.
l. az 1.1.4. pontot – gyorsrendezés (Quicksort),

– összeválogatás, a tömböt különféle résztömbökre osztjuk, ezeket külön rendezzük, majd összeválogatjuk,
l. az 1.1.3. pontot – Shell rendezés,

– kiválasztás, az *i*-edik elemet a rendezetlen elemek közül kiválasztjuk és a saját helyére tesszük,
l. az 1.1.5. pontot – fastruktúrás rendezés (Heapsort).

Célunk, az eljárások értelmezése mellett annak a megállapítása, hogy a tömbök rendezésében ezek mennyire hatékonyak. Hivatkozni fogunk ezért az adatrekordok kezelése során a kulcsok összehasonlítási gyakoriságára S (darabszám), valamint az adatrekordok mozgási gyakoriságára B (darabszám). Nem a pontos érték érdekel minket, amely gyakran csak bonyolult képlettel számítható, hanem a nagyságrend.
A *P. Bachmann* által 1892-ben feltalált írásmód, az *O–Notation*, segítségével egyszerűsíthetjük a képleteket.
Egyenlőségi jelet írhatunk oda is, ahol a két kifejezés nem teljesen azonos. Lényegében ez becslést jelent, és kimondja, hogy a képletben nincs magasabb fokú kifejezés.

Példa: $\frac{1}{2}N^3 + 6N^2 = O(N^3)$, illetve

$$\frac{1}{2}N^3 + 6N^2 = \frac{1}{2}N^3 + O(N^3)$$

A második képlet szigorúbb közelítés.
A kulcs-összehasonlításoknál az S-re egy alsó határt adhatunk meg, amelyet azonban meg kell gondolnunk.

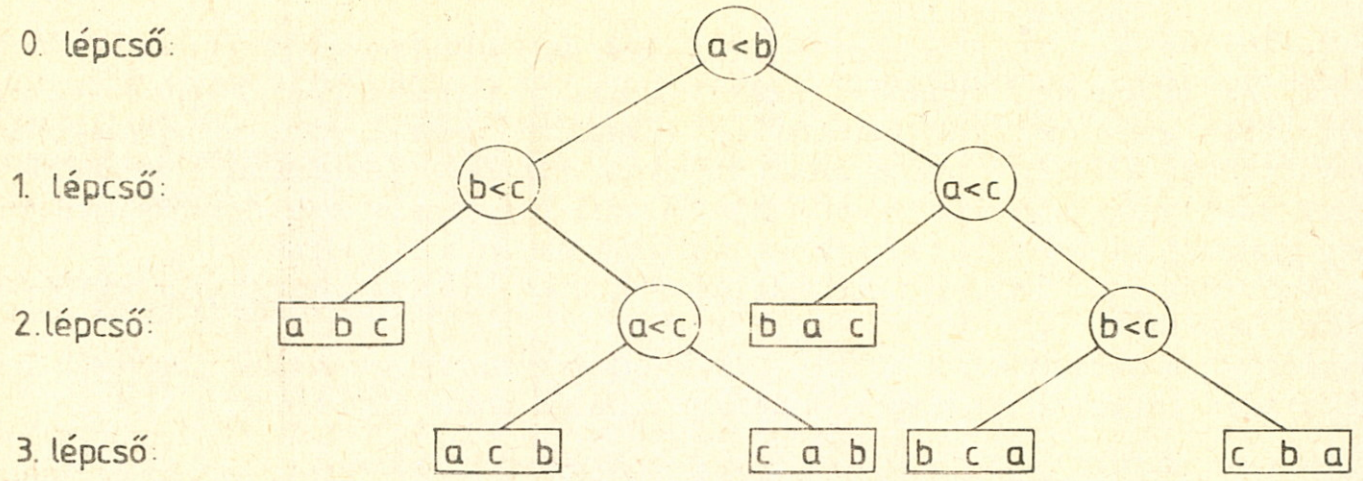
A kulcs-összehasonlítások legkisebb számának meghatározása

Tételezzük fel, hogy a három rendezésre kerülő szám: *a, b, c*. Az egyszerűség kedvéért legyenek a számok különbözőek. Az *a* számot írhatjuk az első, a második vagy a harmadik helyre. Ez három lehetőség. Egy hely már foglalt, a *b* számnak már csak két lehetősége van. A *c* számára csak egy hely maradt. Mivel az egyes számok számára a helyválasztás egymástól független, az egyes elhelyezési lehetőségeket össze kell szorozni. Az esetünkben, tehát a 3 szám elrendezése: 3 · 2 · 1 = 3! féle képpen lehetséges.

Általános esetben:
n szám lehetséges elrendezéseinek a száma = *n*!
Tekintsük most a három szám egy tetszőleges elrendezését. Annak megállapítására, hogy melyik elrendezés állt elő összehasonlítjuk párosával az egyes számokat. Minden összehasonlítás két kimenetű aszerint, hogy *igaz* vagy *sem*. A *Pascal* programrészletet azonnal megadhatjuk:

```
{ S. II. 1.1 }

if a < b
   then if b < c
           then Writeln('a<b<c')
           else if a < c
                   then Writeln('a<c<b')
                   else Writeln('c<a<b')
   else if a < c
           then Writeln('b<a<c')
           else if b < c
                   then Writeln('b<c<a')
                   else Writeln('c<b<a')
```

Az elrendezés helyett kiírathatjuk magát a három számot is a megállapított sorrendben, ami megfelel egy rendezésnek.



Az ábra megmutatja, hogy milyen áttekinthetően ábrázolható az ún. összehasonlítási fán a rendezés lefutása. Elkészítettük az algoritmus (esetünkben a *Pascal* programlészlet) összehasonlítási fáját, amelyben minden összehasonlításnak egy körrel ábrázolt csomópont felel meg. A körbe írtuk a végrehajtott összehasonlítást. A csomópontokról két kimenet van, az úgynevezett *ágak*. Ezek vagy további csomópontokhoz vezetnek, vagy a fa *leveleihez*, amelyeket téglalappal jelöltünk.
Azért nevezték el fának, mert kis fantáziával egy fejtetőre állított fa formájára hasonlít. Azokat a fákat, amelyeknél egy csomót mindig pontosan csak két elágazás követ *bináris* fáknak nevezzük. A legfelső csomót gyökérnek nevezzük és a 0. lépcsőn található. A csomók különböző lépcsőkre való felosztásával a fa magasságát is definiálhatjuk; egy fa magassága a lépcsők számával egyenlő.

Egy rendező eljáráshoz célszerűen kevés összehasonlítást kívánunk alkalmazni, ezért az összehasonlítási fájának minimális magasságúnak kell lennie. A minél nagyobb számú csomó melletti minél alacsonyabb faméret eléréséhez az kell, hogy az i-edik lépcsőn levő csomók száma kétszeres legyen az $(i-1)$-edik lépcsőn levőkhöz képest. Ezzel a *levelek* száma a bináris fa k magasságából becsülhető:

a levelek darabszáma max. 2^k

Amennyiben tudjuk a *levelek* darabszámát, akkor a fenti összefüggés alapján azonnal megbecsülhetjük a fa magasságát;
ha a levelek száma n, akkor a magasság legalább G (lg(n)) ahol *G* a legközelebbi nagyobb egészszámot jelenti, mivel a magasság csak egész szám lehet.
Eszmefuttatásunk eredményeképpen a kezdeti kérdés az összehasonlítások minimális számát illetően megválaszolható:

Minden algoritmusnak, amely a rendezést összehasonlításra építi, legalább G (lg(n!)) összehasonlítást kell tennie.

Ahogy láttuk az összehasonlítási fa fontos eszköz az algoritmusok elkészítéséhez. Nemcsak rendező, hanem kereső eljárásokhoz is használható, ahogy ezt a fejezet második részében majd újból megállapíthatjuk.

A sok elmélet után végre kezdjük!

1.1.2. Rendezés beszúrással

Ez a fontos rendező algoritmus arra a módszerre vezethető vissza, amelyet a kártyások előszeretettel alkalmaznak; az egyik kártyát a másik után nézik meg és a már összerendezett kártyákhoz beszúrják. A gondolatból algoritmust úgy kaphatunk, hogy feltételezzük, hogy a kártyákat sorban az asztalra rakjuk és ha új kártyát kell beszúrnunk, akkor a nagyobb értékű kártyákat egy hellyel jobbra toljuk és az új kártyát a szabad helyre tesszük.
Ezt az eljárást a tömbökre alkalmazva a *beszúrással való rendezés* módszeréhez jutunk. A kezdő index a 2 (mivel az első elem automatikusan rendezett) és a befejező index az *N*. Az aktuális elem számára az *i* indexszel keressük a beszúrási helyet a már rendezett (*i*–*1*) elem között.
A beszúrási hely megtalálására több lehetőségünk van. Elsőként tegyük fel, hogy az aktuális elemet a *h* segédmezőbe mentjük és a tőle balra álló elemeket sorban addig toljuk egy hellyel jobbra, míg a beszúrási helyet megtaláljuk. Ekkor a segédmező tartalmát erre a helyre beírjuk.
Nézzük a rendezés menetét egy példán. A már rendezett elemeket aláhúzással jelöltük.

Példa:

Kezdő sorrend:	23	43	44	13	76	90	76	10
i=2:	23	43	44	13	76	90	76	10
i=3:	23	43	44	13	76	90	76	10
i=4:	13	23	43	44	76	90	76	10
i=5:	13	23	43	44	76	90	76	10
i=6:	13	23	43	44	76	90	76	10
i=7:	13	23	43	44	76	76	90	10
i=8:	10	13	23	43	44	76	76	90

Magyarázat:
A 23-as elem rendezett.
i=2: a 43-at összehasonlítja 23-mal, ha 23 kisebb mint 43, akkor a beszúrási helyet rögtön megtalálta, nincs átrendezés.
i=3: a 44-et összehasonlítja a 43-mal, ha a 43 kisebb mint 44, akkor a beszúrási helyet rögtön megtalálta, nincs átrendezés.
i=4: a 13-at összehasonlítja 44-gyel, ha a 13 kisebb mint 44, akkor át kell rendezni. Ebből a célból 13-at a *h* változóba tesszük, 44-et egy hellyel jobbra

toljuk. A *h* tartalmát most 43-mal hasonlítjuk össze, a 43 is jobbra vándorol. A *h* tartalmát a 23-mal összehasonlítva, a 23-nak is egy hellyel jobbra kell tolódnia. Mivel nincs több elem, találtunk egy beszúrási helyet, ezzel a *h* tartalmát az első helyre fogjuk beírni.

i=5: nincs jobbra tolás.

i=6: nincs jobbra tolás.

i=7: a 90-et jobbra toljuk.

i=8: A 90, 76, 76, 44, 43, 23 és 13 értékű elemeket jobbra toljuk.

Végül a RendHivo keretprogramot közöljük, amely a rendező eljárás ellenőrzésére való. Éppen ezért ajánlatos ezt az eljárást a rendező programhoz include file-ként csatolni, mert akkor több program használhatja. A rendező eljárás cseréje ezáltal könnyen megvalósítható.

```
{ RENDHIVO.PAS }
{ Pr.*II. 1.1. }

PROGRAM RendHivo;

CONST
   c_max = 1ØØ;

TYPE
   t_r    = record
              kulcs  : integer;
              info : string[8Ø];
            end;

   t_s    = array [Ø..c_max] of t_r;

VAR
   s : t_s;
   i : integer;

PROCEDURE Csere (i,j : integer);

VAR
   h : t_r;

BEGIN
   h:=s[i];
   s[i]:=s[j];
   s[j]:=h;
END; {.....}

PROCEDURE Ir_s;

VAR
   i : integer;

BEGIN
   for i:=1 to c_max
    do begin
      Write(s[i].kulcs:8);
      if i mod 1Ø = Ø then Writeln;
    end;
END; {.....}
```

```
{$I RENDEZ5.PAS}     { A megf. rendezo pr. beillesztese}

BEGIN
  for i:=1 to c_max do s[i].kulcs:=trunc(random*32767);
  Writeln('BEVITEL:');
  Ir_s;
  Rendez(c_max);
  Writeln;
  Writeln('KIVITEL:');
  Ir_s;
END.
```

A keretprogramot a Beszurrend eljárás követi. Rögtön ezután bemutatunk egy második, gyorsabb változatot, amelyben egy ún. végjelet alkalmaztunk. A *found* logikai változót ezzel megtakarítottuk, bár fel kellett vennünk egy további táblázati elemet 0 indexszel. Ebbe először a *Maxint* értéke kerül inicializálásként, így biztosítottuk, hogy a *repeat* ciklus befejeződjön.

A beszúrással való rendezés lassú verziója:

```
{ RENDEZ1.PAS  }
{ Pr. II. 1.2. }

PROCEDURE Rendez (n : integer);

{ Rendezes beszurasos modszerrel, lassabb valtozat }

VAR
   i,j    : integer;
   h      : t_r;
   talal : boolean;

BEGIN
   for i:=2 to n
    do begin
      if s[i].kulcs < s[i-1].kulcs
       then begin
         j:=i;
         h:=s[i];
         repeat
           j:=j-1;
           s[j+1]:=s[j];
           if j = 1
            then talal:=true
            else talal:=s[j-1].kulcs <= h.kulcs;
         until talal;
         s[j]:=h;
       end;
    end;
END; {.....................................................}
```

A gyorsabb verzió végjellel:

```
{ RENDEZ2.PAS  }
{ Pr. II. 1.3. }

PROCEDURE Rendez (n : integer);

{ Rendezes beszurassal, gyoprsabb valtozat }

VAR
   i,j : integer;
   h   : t_r;

BEGIN
   s[Ø].kulcs:=-maxint;
   for i:=2 to n
    do begin
      if s[i].kulcs < s[i-1].kulcs
       then begin
         j:=i-1;
         h:=s[i];
         repeat
           s[j+1]:=s[j];
           j:=j-1
         until s[j].kulcs <= h.kulcs;
         s[j+1]:=h;
       end;
    end;
END; {.....}
```

A beszúrással való rendezés analízise

Tekintsük meg a javított változatot.

Az S függvény leírása

A működés szempontjából elsősorban a *Repeat* ciklus a mértékadó, mert ez igényli a legnagyobb időráfordítást az eljárásban. Megállapíthatjuk, hogy három eset különböztethető meg.

a) A legkedvezőbb eset
Ebben az esetben csak

$$c\_max - 1$$

összehasonlítást kellett elvégezni. Ez az összehasonlítások legkisebb száma, amelyek a tömb rendezettségének megállapításához szükségesek. Ez a *legkedvezőbb* érték!

b) Az átlagos eset
Először el kell végeznünk c_max-1 összehasonlítást, ahhoz azonban, hogy az i-edik rekordot is helyesen rendezzük legkevesebb *0*, legrosszabb esetben $i-1$ további összehasonlítást (jelölést) kell elvégeznünk. Az egyes kulcsok azonos valószínűségénél ez átlagban $(i-1)/2$ összehasonlítást jelent. Nekünk tehát

$$1/2 + 2/2 + \ldots + (c\_max-1)/2 = (1+2+3+4+\ldots + c\_max-)/2$$

összehasonlítást kell végrehajtanunk. Mivel a számok összege 1-től c_max-1-g az alábbi képlet szerint számítható,

$$c\_max*(c\_max-1)/2$$

Az eredmény behelyettesítés után:

$$c\_max*(c\_max-1)/4$$

az S átlagos darabszámára. Ehhez kell hozzáadnunk a c_max−1-hez szükséges összehasonlításokat. Az összesítés eredménye:

$$S(c\_max) = (c\_max^2 + 3*c\ max-4)/4 = O(c\_max^2)$$

Az eljárás tehát normális esetben lényegesen több összehasonlítást igényel, mint a feltétlenül szükséges $lg(c\_max!)$.

c) A legrosszabb eset
Megfordított sorrendű rendezéskor a ráfordítás az előbbi meggondolásból adódik:

$$2+3+\ldots+c\_max = ((c\_max^2 + c\_max)/2)-1 = O(c\_max^2)$$

A B függvény leírása

a) A legkedvezőbb eset
Ebben az esetben 0 mozgás van a *Repeat ciklusban, míg a For* ciklusban

$$2*(c\_max-1)$$

számú elemmozgatás kell.

b) Az átlagos eset
Az a) pont szerinti $2*(n-1)$ mozgatást minden esetben meg kell csinálni. Ehhez adódik hozzá a *Repeat* ciklusban

$$(1+2+\ldots+c\_max-1)/2 = (c\_max^2 - c\_max)/4$$

mozgatás, ami összegezve a B függvényre a következő

$$B(c\_max) = (c\_max^2 + 7*c\_max-8)/4 = O(c\_max^2)$$

c) A legrosszabb eset
Az a) pontban megállapított mozgatásokhoz ebben az esetben még

$$(1+2+\ldots+c\_max-1) = (c\_max^2 - c\_max)/2$$

mozgatást kell hozzáadni. Ezzel az összesítés

$$B(c\_max) = (c\_max^2 + 3*c-max-4)/2 = O(c\_max^2)$$

lesz.

A fentiekből világos, hogy a *beszúrással való rendezés* a már rendezett tömbökhöz alkalmazható legkedvezőbben.
A beszúrási hely felkeresését a *Keresés*-nél leírt bináris módszerrel tovább gyorsíthatjuk. Azonban az adatrekordok eltolása a kereső módszertől független, így a sebességen csak keveset nyerhetünk.
H. Nagler (1960) analízisében azt tanácsolja, hogy a leírt módszerrel ne rendezzünk 128 rekordnál többet.

1.1.3. Sell rendezés (Shellsort)

Jobb rendezési módot talált *Donald L. Shell* 1959-ben, és ez a módszer az ő nevét viseli: Shell rendezés.
A módszer azon alapszik, hogy elkerüli a beszúrással való rendezés nagy hátrányát – mindig csak a közvetlenül szomszédos kulcsok kerülnek összehasonlításra, ami átlagos esetben nagyszámú felesleges mozgatáshoz vezet –, ha először az egymástól távolabb fekvő kulcsokat hasonlítjuk össze, úgy hamarabb kerülhetnek a saját rendezett helyükre.
A lépésközt páratlan számra választva kezdjük pl. a 7-tel és rendezzük az elemeket 7-es közzel. Ezután csökkentjük a közt 5-re és ezzel rendezzük az összes elemet. Erre következik a 3-as lépésközzel való rendezés, majd végül az 1-es lépésközű rendezés biztosítja, hogy az egész tömb rendezett lesz. Lényegében tehát a *Shell rendezés* csak a különböző lépésközökkel végzett feldolgozás miatt különbözik a beszúrással való rendezéstől.
A lépésközöket tetszés szerint választhatjuk, csak a kettő hatványait kell elkerülnünk. Ezeket azért, mert így az egyik menetben összehasonlított kulcsokat a következő menetben újra összehasonlítanánk.
Az egyszerűség kedvéért a lépésközre az alábbi képletet adjuk:
inc: = *n* inicializálás
inc: = (inc div 3) +1 minden menetre.
Tekintsük a rendezés menetét ismét egy példán. Három menetben rendeztünk 3, 2, 1 lépésközökkel. Az egyes láncokban összehasonlított elemeket aláhúzással jelöltük.

Példa:

Kezdeti sorrend:	23 43 44 13 76 90 76 10
1. menet inc=3	
i=4:	13 43 44 23 76 90 76 10
i=5:	13 10 44 23 43 90 76 76
i=6:	13 10 44 23 43 90 76 76
2. menet inc=2	
i=3:	13 10 43 23 44 90 76 76
i=4:	13 23 44 43 10 76 76 90
3. menet inc=1:	
i=2:	
Magyarázat:	10 13 23 43 44 76 76 90

1. menet
i=4: A lánc 23-as eleme már rendezett, ezért az induló-érték is 4. Ezután összehasonlítja a 13-at 23-mal és megcseréli. Mivel a következő elem a láncban a 76 nagyobb, mint 23, nincs csere.
i=5: A 43 a láncban már rendezett. A következő érték a 76 összehasonlítva 43-mal nem eredményez cserét. A 10 összehasonlítva 76-tal és 43-mal azt eredményezi, hogy a 10 elfoglalja a 43-as helyét.
i=6: Összehasonlítás 44 és 90 között, nem eredményez cserét.

2. és 3. menet

Az eljárás az első menettel analóg, csak a lépéstávolság változik. Felismerhető, hogy az 1-es lépésköz a beszúrással való rendezést valósítja meg (de kevesebb mozgatással). A *Shell rendezést* ezzel megismertük. Az alábbi példa a beszúrással való rendezés utolsó változataként alkalmazza a Shell-féle rendezést.

```
{ RENDEZ3.PAS  }
{ Pr. II. 1.4. }

PROCEDURE Rendez (n : integer);

{ Rendezes Shell modszerrel }

VAR
   lepes,kezd,i,j  : integer;
   h               : t_r;

BEGIN
   lepes:=n;
   s[Ø].kulcs:=-maxint;
   repeat
     lepes:=lepes div 3 + 1;
     for kezd:=1 to lepes
      do begin
        i:=kezd+lepes;
        while i <= n
         do begin
           if s[i].kulcs < s[i-lepes].kulcs
            then begin
              j:=i;
              h:=s[i];
              repeat
                j:=j-lepes;
                s[j+lepes]:=s[j];
              until s[j-lepes].kulcs <= h.kulcs;
              s[j]:=h;
            end;
            i:=i+lepes;
         end;
        end;
   until lepes = 1;
END; {.....................................................}
```

A Shell rendezés analízise

Meglepő módon a rendezés matematikai analízise csak nagyon nehezen végezhető el. Mai ismereteink szerint nem létezik pontos számítás az S és B függvényt illetően, ezért kísérletekre hagyatkoztunk, amelyek azt mutatták, hogy megfelelően nagy c_max esetén a mozgatások száma

$$B(c\_max) = O(c\_max^{1.25})$$

Ez mindenesetre lényeges javulást jelent a beszúrással való rendezéshez képest.

1.1.4. Gyorsrendezés (Quicksort)

Az alapváltozatot, amely csere elven alapszik 1962-ben *C.A.R. Hoare* hozta nyilvánosságra.
Először ki kell keresnünk egy ún. *pivot* elemet és a tömböt két résztömbre kell osztanunk. Az eljárás a következő; balról indulva addig vizsgálja a tömböt, míg a *pivot* elemnél *nagyobb vagy egyenlő* elemet nem talál, majd jobbról indulva keres a tömb elemei között addig, míg a *pivot* elemnél *kisebb vagy egyenlő* elemet nem talál. Ekkor a két elemet felcseréli és újra kezdi a vizsgálatot. Ezért a *gyorsrendező* eljárást a cserealgoritmusok osztályába soroljuk. A tömb vizsgálata és felosztása abbamarad, ha a kétoldali vizsgálat során valahol találkozik. Eddig a tömböt két félre osztotta, ahol a bal oldali elemek kisebb-egyenlőek, a jobb oldaliak pedig nagyobb-egyenlőek mint a *pivot* elem. Ezután az egész eljárás rekurzív módon a résztömbökhöz fordul addig, míg a résztömbök már csak egy elemet tartalmaznak, amelyek világos, hogy rendezetteknek tekinthetők.
Tekintsük a rendezési eljárást egy példán. A pivot elemet az alábbi képlettel választjuk:
pivot:=$S((j+i)$ div 2).key
ahol i az alsó, j a felső határt jelenti.

Példa:

Kezdő sorrend:	23	43	44	13	76	90	76	10
1. felosztás i=1, j=8								
pivot: 13	10	13	44	43	76	90	76	23
2. felosztás i=1, j=2								
pivot: 10	10	13	44	43	76	90	76	23
3. felosztás i=3, j=8								
pivot: 76	10	13	44	43	23	76	90	76
4. felosztás i=3, j=6								
pivot: 43	10	13	23	43	44	76	90	76
5. felosztás i=5, j=6								
pivot: 44	10	13	23	43	44	76	90	76
6. felosztás i=7, j=8								
pivot: 90	10	13	23	43	44	76	76	90

Magyarázat:

1. felosztás i=1 , j=8
   A képlet alapján a 13 a pivot elem. A balról induló keresésnél a megtalált elem a 23, jobbról pedig a 10. Megcseréljük a két elemet. Az új keresésnél balról a 43-at találjuk, jobbról egészen a pivot elemig (13) kell keresnünk. A 43 és a 13 helyet cserélnek.
   Bár a további kereséssel balról a 44 adódik, azonban a jobb oldali index =2 kisebb lesz a bal oldalinál, ezért a felosztás befejeződik.
   A két résztömb (1,2) és (3,8).

2. felosztás i=1 , j=2
   Mivel a két elem a 10 és a 13 már rendezett, a 10-es pivot elemre nézve nem eredményez cserét a vizsgálat. Az új résztömbök (1,1) és (2,2) már csak egy-egy elemmel rendelkeznek, ezért rendezettnek tekinthetők.

3. felosztás i=3 , j=8
   A pivot elem a 76. Ez érdekes eset, mert a 76 kétszer fordul elő. A balról megtalált

elem a 76. Ezt a 23-mal megcseréljük. Ezután a 90 és a másodikként előforduló 76 kerül cserére.
Az így kapott résztömbök a (3,6) és (7,8).

4. felosztás i=3 , j=6
   A 44 és a 23 cserél helyet. A (3,3) így készre rendezett lesz, az (5,6) résztömböt még vizsgálni kell.

5. felosztás i=5 , j=6
   Nem eredményez cserét. Az (5,5) és (6,6) rendezett.

6. felosztás i=7 , j=8
   Pivot elemként a 90 számítható és a 76-tal kell felcserélni. A két egyelemű tömb a (7,7) és (8,8) definíció szerint rendezett.

Az eljárás *Turbo–Pascal*-ban:

```
{ RENDEZ4.PAS  }
{ Pr. II. 1.5. }

PROCEDURE Rendez (i,j : integer);

{ Rendezes un. gyorsrendezo modszerrel }

VAR
   a,e,pivot : integer;

BEGIN
   if i < j
    then begin
      pivot:=s[(j+i) div 2].kulcs;
      a:=i-1;
      e:=j+1;
      repeat
        repeat
          a:=a+1;
        until s[a].kulcs >= pivot;
        repeat
          e:=e-1;
        until s[e].kulcs <= pivot;
        if a < e then csere(a,e);
      until a >= e;
      if a = e
       then begin
         rendez(i,a-1);
         rendez(e+1,j);
       end
       else begin
         rendez(i,e);
         rendez(a,j);
       end;
    end;
END; {.....................................................}
```

A gyorsrendező analízise

A pivot elem megválasztása a futási idő tekintetében jelentős szerepet játszik. Különösen kerülni kell a feleslegesen sok rekurzív hívást, ez akkor adódik, ha mindig a legkisebb elemet választjuk pivot elemként. Válasszuk példának a következő tömböt:

23 61 45 10 38 51 76

Az előbbi elgondolásunk szerint válasszuk pivot elemnek a 10-et, így 6 elem hosszúságú nem üres rösztömböt kapunk. A 10-hez tartozó résztömböt rendezettnek tekintjük:

10 61 45 23 38 51 76

Ezután, ha a 23-at választjuk, az a (2,2) és (3,7) felosztáshoz vezet.

10 23 45 61 38 51 76

Tovább a legkisebb választható elemet pivot elemként választva következik a 38 és így tovább. Miért olyan rossz ez a felosztás?

Az eljárás minden egyes rekurzív meghívásakor tárolásra kerülnek a kezelési adatok, azaz a paraméterek és a lokális változók. A *Pascal*-ban ez a *Runtime-stack* segítségével történik, amely fenntartott tárolóterület az eljárásmeghívásoknak. Amennyiben a gyorsrendező, mint az előbbi példánkban, hatszor rekurzív módon meghívja saját magát, akkor a kezelési adatokat hatszor kell tárolni. Visszafelé haladva ezek természetesen felszabadulnak, de a tárolás értékes gépidőt köt le, így a rendezés lelassul.
Amennyiben a mindenkori legelső tömbelemet választanánk pivot elemnek, akkor a helyzet még rosszabb lenne, mert amikor a tömb már rendezett, akkor áll elő a legkedvezőtlenebb helyzet. (Gondoljunk arra, hogy a beszúrással való rendezés adta ebben az esetben a legkedvezőbb eredményt.)
A pivot elem választása abban az esetben a legkedvezőbb, ha az a tömböt két közel egyforma résztömbre osztja fel.
Természetesen a gyakorlatban nem ésszerű minden elemet ehhez megvizsgálni, marad tehát a véletlen választás (néhány összehasonlítás után).

Az S függvény számítása
Az S(n), a kulcs-összehasonlítások száma, amelyeket a gyorsrendező egy n hosszúságú tömbön végrehajt és B(n) az elemcserék száma. Előljáróban kijelenthetjük, hogy

$$S(1) = S(0) = 0$$

Két esetet kell megkülönböztetnünk.

a) A két *Repeat* ciklus $a=e$ esetén áll le.
Ez azt jelenti, hogy a felosztás végén a pivot elem középen található. Az ilyen felosztáshoz n elem esetén $n+1$ összehasonlítás szükséges.

b) A két *Repeat* ciklus $a>e$ esetén áll le.
Ez a helyzet akkor áll elő, ha a pivot elemet a két részblokk egyikének belsejébe cseréltük. Az összehasonlítások száma ekkor eggyel több mint az előbbi esetben, tehát $n+2$.
Minket elsősorban az S nagyságrendje érdekel, elhagyjuk a két eset közötti különbséget és n db összehasonlítással számolunk. A résztömbök méretét k, ill. $n-k$ hosszúságúra választva az összehasonlítások száma c_max elem esetén

$$S(c\_max) = c\_max + S(c\_max-k)$$

Az egyenlet megoldásához meg kell határoznunk k értékét. Vizsgáljuk meg először a *legkedvezőtlenebb* esetet.

a) A legkedvezőtlenebb eset
Létrejönnek a c_max-1, c_max-2, c_max-3,. . .,1 tömbök, s ez az egyenletünk szempontjából így írható le:

$$S(c\_max) = c\_max + S(c\_max-1)$$

Az egyenlet megoldására a kisebb értékeket helyettesítsük be fokozatosan az egyenletbe:

$$\begin{aligned} S(c\_max) &= c\_max+S(c\_max-1) \\ &= c\_max+c\_max-1+S(c\_max-2) = \\ &= c\_max+c\_max-1+c\_max-2+\ldots+2 \end{aligned}$$

Az utolsó érték a láncban könnyen ellenőrizhető számítással

$$S(1) = 0$$
$$S(2) = 2+S(0) = 2$$

Ezzel azonban újra a számok összegét kaptuk 1-től c_max-ig.
A legkedvezőtlenebb esetre tehát

$$S(c\_max) = ((c\_max^2+c\_max)/2)-1 = O(c\_max^2)$$

érvényes.

A gyorsrendező legkedvezőtlenebb viselkedése esetén tehát, éppen azonos az összehasonlítások száma a beszúrással való rendezéshez tartozó összehasonlítások számával. Gyenge teljesítmény ez, egy magát *gyorsrendező*-nek nevező módszertől! Hogyan néz ki azonban egy átlagos eset?

b) Átlagos eset
Bizonyítás nélkül közöljük a közelítő értéket

$$S(c\_max) = O(c\_max*\lg(c\_max))$$

Ez kiváló eredményességről tanúskodik.

c) A legjobb eset

Könnyen belátható, hogy a legkedvezőbb eset akkor lép fel, ha a pivot elem választásával telibe találunk. Ekkor a tömböt minden esetben két megközelítően azonos félre osztjuk (egyszerűsítve úgy vesszük, mintha teljesen azonos nagyságúak lennének). Ebben az esetben S nagyságát az alábbi egyenlet adja:

$$S(c\_max) = c\_max + 2*S(c\_max/2)$$

A fentebb már megismert behelyettesítő módszerrel adódik, hogy

$$\begin{aligned} S(c\_max) &= c\_max+2*S(c\_max/2) \\ &= c\_max+2*(c\_max/2+2*S(c\_max/4)) \\ &= c\_max+2*(c\_max/2+2*(c\_max/4+2*S(c\_max/8))) \end{aligned}$$

Helyettesítsünk be annyiszor, hogy S elérje a 2-es értéket. A behelyettesítések számának meghatározásához írjuk fel:

$$c\_max/2^k = 2$$

Mindkét oldal logaritmusát képezve

$$\lg(c\_max) = k+1$$

Ezzel az $S(c\_max)$ számot könnyen meghatározhatjuk; ha az első elemet c_max/1-nek definiáljuk, akkor k a 0 és $\lg(c\_max)-1$ közötti értékeket veszi fel. Tehát $\lg(c\_max)$ elemünk van.

$$\begin{aligned} S(c\_max) &= c\_max+2*(c\_max/2+2*(c\_max/4+S(c\_max/8+\ldots)) = \\ &= c\_max+c\_max+c\_max+\ldots+c\_max = c\_max*\lg(c\_max) \end{aligned}$$

A B függvény meghatározása

a) A legkedvezőtlenebb eset
Válasszuk a következő speciális esetet:

91 85 77 43 11 22 18 34

A pivot elem legyen a 11. Ez azt eredményezi, hogy a 91 és a 34, a 85 és a 18, a 77 és a 22 valamint a 43 és a 11 helyet cserélnek.
Belátható, hogy kedvezőtlen esetben n elem esetén $n/2$ cserét kell végrehajtani. Mivel mindkét résztömb $n/2$ elemet tartalmaz, adódik az egyenlet:

$$B(c\_max) = c\_max/2 + 2 * B(c\_max/2)$$

A S függvény számítása alapján ismerjük az eredményt:

$$B(c\_max) = c\_max * \lg(c\_max)$$

b) Átlagos eset
Bizonyítás nélkül közöljük az alábbi képletet

$$B(c\_max) = 0.23*(c\_max+1)*\lg(c\_max)$$

c) Legjobb eset
Ez az eset akkor áll elő, ha egy cserét sem kell végrehajtani, például az alábbi kiindulási sorrend esetében:

12 23 34 45 56 67 78

A legkedvezőbb esethez tartozó egyenlet

$$B(c\_max) = 0$$

Bármilyen jók is a gyorsrendező értékei, mégis c_max kisebb értékénél az a hátránya jelentkezik, hogy a beszúrással való rendezésnél lassabb lesz. A két módszer kombinációja csökkenti a rekurzív hívások okozta tár igényt is.
Példaként tekintsünk meg egy olyan rendezőprogramot, amely a 20 alatti elemszám esetén a beszúrással való rendezést alkalmazza.

```
{ RENDEZ5.PAS  }
{ Pr. II. 1.6. }

PROCEDURE Rendez (n : integer);

VAR
    i,j       : integer;

    PROCEDURE Gyors_rend (i,j : integer);

    { Gyorsrendezes max. 20 elemre }

    VAR
    a,e,pivot : integer;

    PROCEDURE Beszur_rend (k,n : integer);

    VAR
     i,j    : integer;
     h      : t_r;
     talal : boolean;

    BEGIN
     for i:=k+1 to n
      do begin
        if s[i].kulcs < s[i-1].kulcs
         then begin
           j:=i;
           h:=s[i];
           repeat
             j:=j-1;
             s[j+1]:=s[j];
             if j = 1
              then talal:=true
              else talal:=s[j-1].kulcs <= h.kulcs;
           until talal;
           s[j]:=h;
         end;
      end;
    END; {.....}




    BEGIN
    if i-j < 20
     then Beszur_rend(i,j)
     else
      if i < j
       then begin
         pivot:=s[(j+i) div 2].kulcs;
         a: i-1;
         e:=j+1;
         repeat
           repeat
             a:=a+1;
           until s[a].kulcs >= pivot;
           repeat
             e:=e-1;
           until s[e].kulcs <= pivot;
           if a < e then csere(a,e);
         until a >= e;
         if a = e
          then begin
```

```
                Gyors_rend(i,a-1);
                Gyors_rend(e+1,j);
              end
              else begin
                Gyors_rend(i,e);
                Gyors_rend(a,j);
              end;
          end;
      END;

BEGIN
i:=Ø;
j:=n;
Gyors_rend(i,j)
END;
```

1.1.5. A fastruktúrás rendezés (Heapsort)

A *J.W. Williams* által feltalált és 1964-ben bemutatott módszer az alábbi egyszerű elven alapszik: a még rendezetlen elemek egyikét kiválasztva, azt a végleges helyére kell tenni. Különlegessége az, hogy a rendezetlen elemeket fa (heap) formában helyezi el.
Ennek a heap-nek nincs köze a *Pascal*-ban a dinamikus változókhoz alkalmazott tárterület-elnevezéshez.
Definíció: a bináris fa, amelyet az *s* tömb ábrázol, az alábbi feltételeket elégíti ki:

$$s[i] >= s[2*i], \text{ha } 1 <= i <= c\_max/2$$

és

$$s[i] >= s[2*i+1], \text{ha } 1 <= i < c\_max/2$$

Tömbből mindig alkothatunk bináris fát a következők szerint:

a) Az 1 csomópont a gyökér
b) Az *i* csomópont, a $2*i$ csomópont bal oldali és a $2*i+1$ ($1<=i<c\_max/2$) a jobb oldali követője.

Az eljárás két fázisra tagozódik:

1. A fa készítése az esetlegesen rendezett tömbből. Vegyük figyelembe, hogy minden tetszőlegesen feltöltött tömb felfogható bináris faként.
   A fát úgy állítjuk elő, hogy egy résztömböt veszünk, amelyet fokozatosan balról tovább szélesítünk. Kezdjük a következő résztömbbel

$$c\_max \text{ div } 2,\ldots,c\_max,$$

   ahol minden elem

$$(c\_max \text{ div } 2)+1,\ldots c\_max$$

   indexszel ellátott levél és ezért elölről tekintve fát alkotnak.
   Az első közös csomópont tehát a c_max div 2 indexet kapja.
   Erre kell megvizsgálni, hogy a fatulajdonságot teljesíti-e, ha nem, akkor a követő két csomópont közül a nagyobbal fel kell cserélni (feltéve, hogy van két követő). Ezzel a tömböt egy hellyel balra szélesítjük és ezt az új csomópontot ismét megvizsgáljuk. Az eljárást addig kell folytatni, amíg az első csomóponthoz nem érkezünk, amelyet szintén meg kell vizsgálnunk.

2. Az első (legnagyobb) elem cseréje az utolsóval.
Először az 1. fázist alkalmazzuk a c_max hosszúságú tömbre, ezzel a legnagyobb elem az 1. helyre kerül. Most következik a 2. fázis. Ezt követően az 1. fázist alkalmazzuk a *c_max*−1 hosszúságú tömbre, amelyet a 2. fázis követ a cserével, majd ismét az 1. fázist alkalmazzuk a *c_max*−2 hosszúságú tömbre stb. Ha elérünk az 1 hosszúsághoz, rendezett tömbünk lesz.

Tekintsük ismét a rendezés menetét a példánkon:

Kezdő sorrend:	23 43 44 13 76 90 76 10
1. fázis	
Résztömb 4, . . ., 8	23 43 44 13 76 90 76 10
Résztömb 3, . . ., 8	23 43 90 13 76 44 76 10
Résztömb 3, . . ., 8	23 76 90 13 43 44 76 10
Résztömb 1, . . ., 8	90 76 76 13 43 44 23 10
2. fázis	
Csere 1 <—> 8	10 76 76 13 43 44 23 90
1. fázis	
Résztömb 3, . . ., 7	10 76 76 13 43 44 23 90
Résztömb 2, . . ., 7	10 76 76 13 43 44 23 90
Résztömb 1, . . ., 7	76 43 76 13 10 44 23 90
2. fázis	
Csere 1<—> 7	23 43 76 13 10 44 76 90
1. fázis	
Résztömb 3, . . ., 6	23 43 76 13 10 44 76 90
Résztömb 2, . . ., 6	23 43 76 13 10 44 76 90
Résztömb 1, . . ., 6	76 43 44 13 10 23 76 90
2. fázis	
Csere 1 <—> 6	23 43 44 13 10 76 76 90
1. fázis	
Résztömb 2, . . ., 5	23 43 44 13 10 76 76 90
Résztömb 1, . . ., 5	44 43 23 13 10 76 76 90
2. fázis	
Csere 1 <—> 5	10 43 23 13 44 76 76 90
1. fázis	
Résztömb 2, . . ., 4	10 43 23 13 44 76 76 90
Résztömb 1, . . ., 4	43 10 23 13 44 76 76 90
2. fázis	
Csere 1 <—> 4	13 10 23 43 44 76 76 90

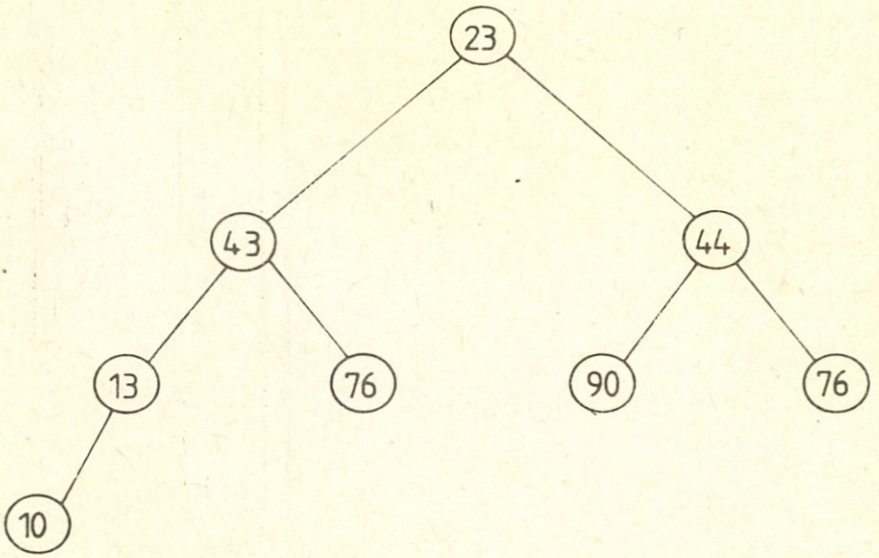
1. fázis
Résztömb 1, . . ., 3 23 10 13 43 44 76 76 90

2. fázis
Csere 1 <—> 3 13 10 23 43 44 76 76 90

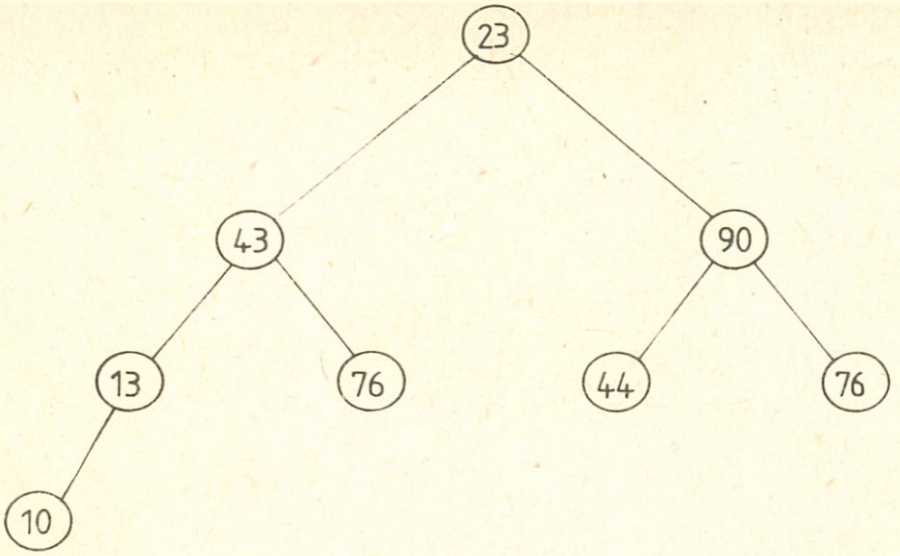
1. fázis
Résztömb 1,2 13 10 23 43 44 76 76 90

2. fázis
Csere 1 <—> 2 10 13 23 43 44 76 76 90

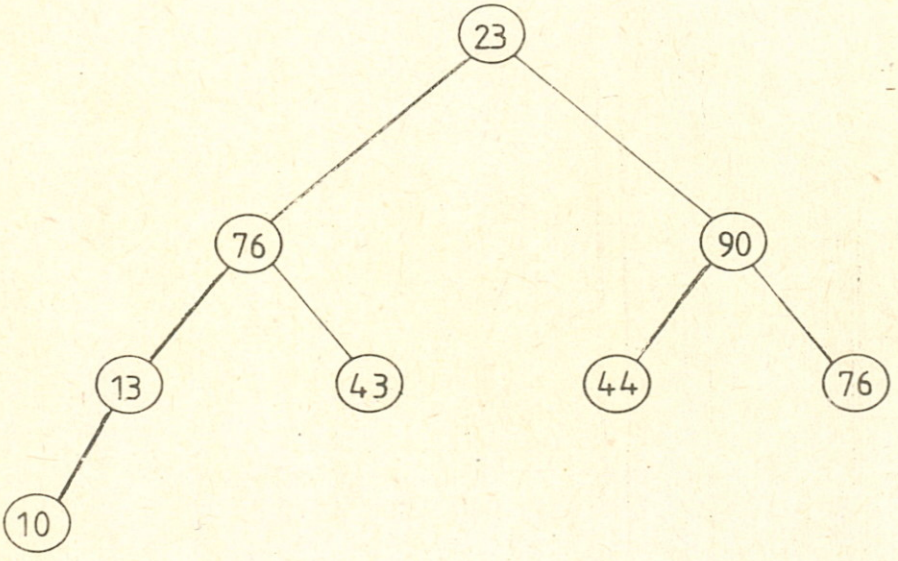
Magyarázat:
Az alábbi ábra azt a bináris fát mutatja, amelyet a kezdő tömbhöz rendelhetünk.



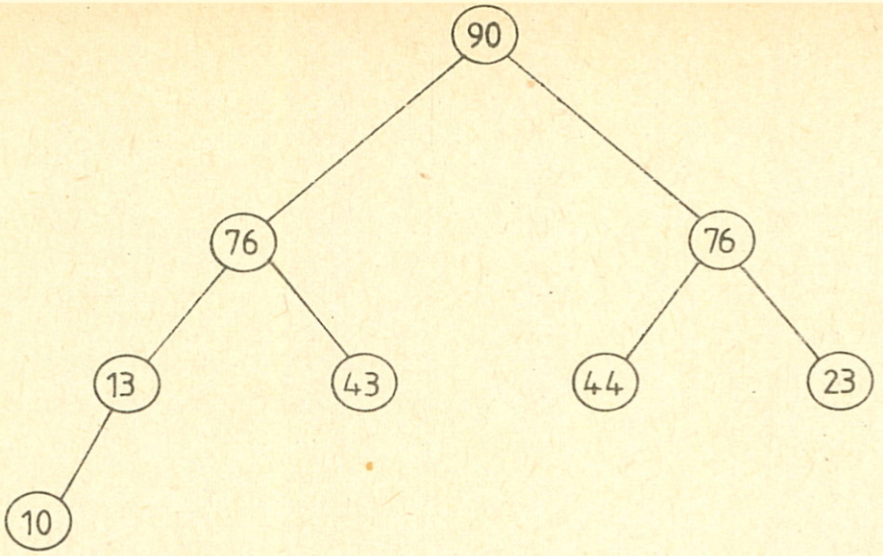
1. fázis:
Résztömb 4,. . .,8:
A 13-as elemet az egyetlen követővel a 10-zel kell összehasonlítani.
Mivel 13 > 10, a helyén marad.
Résztömb, 3,. . .,8:
A résztömb gyökerét 44-et a követő elemekkel 90-nel és 76-tal kell összehasonlítani és a
90-essel kicserélni.



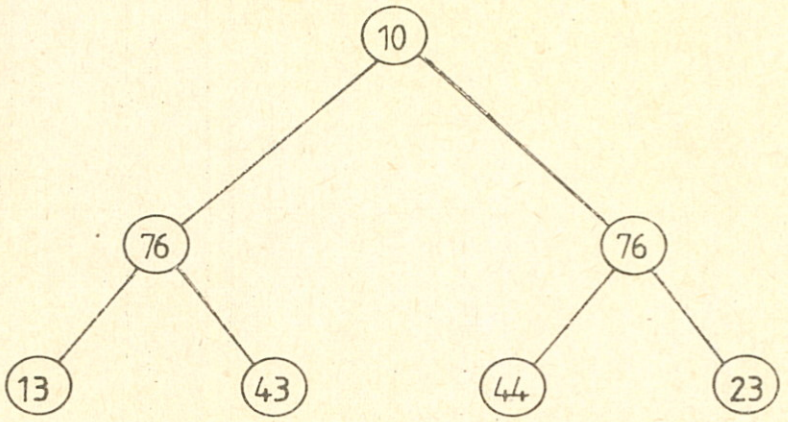
Résztömb 2,. . .,8:
A 43-as elemet kell a 13-as és a 76-os elemmel összehasonlítani és a 76-ossal kicserélni.



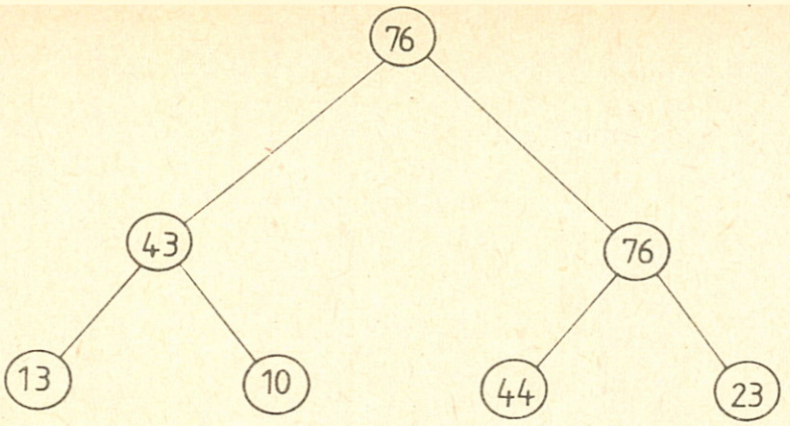
Résztömb 1,. . .,8:
A 23-as elemet a 90-essel, továbbá a 76-ossal kell kicserélni. . .



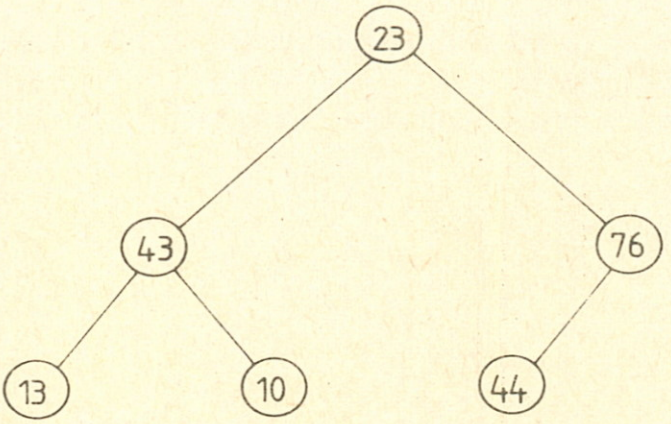
2. fázis:
Az első és utolsó elem cseréje azt eredményezi, hogy most a 10-es a gyökér elem. Egyidejűleg a tömb hossza 1-gyel csökkent, mivel a 90-es a legnagyobb elem már a saját helyén van.



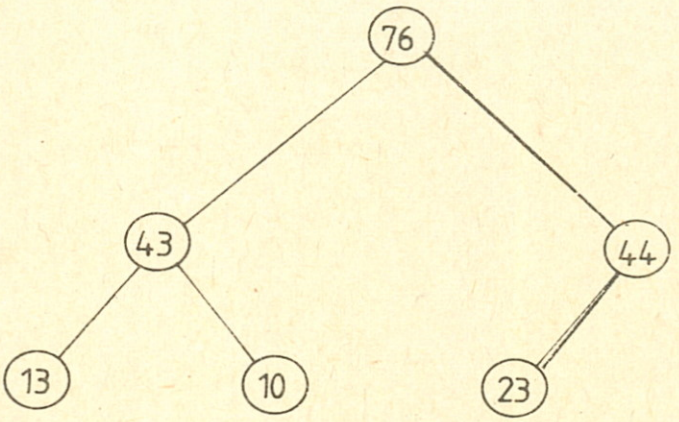
1. fázis:
Résztömb 3,. . .,7:
A 76-ot a 13-mal és 43-mal kell összehasonlítani. Nincs csere.
Résztömb 2,. . .,7:
A 76-ot a 23-mal és 44-gyel kell összehasonlítani. Nincs csere.
Résztömb 1,. . .,7:
A 10-et kell a 76-tal és a 76-tal összehasonlítani. Csere a bal oldali követővel, majd a 43 mal.



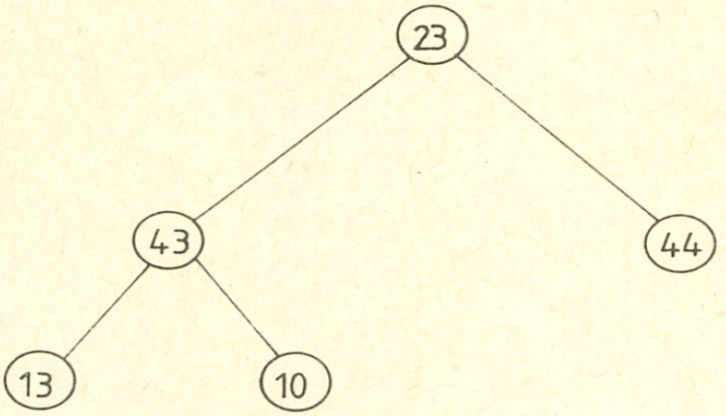
2. fázis:
A 76-os és 23-as cseréje a 23-at a fa gyökerébe helyezi. A tömb mérete 6-ra csökkent.



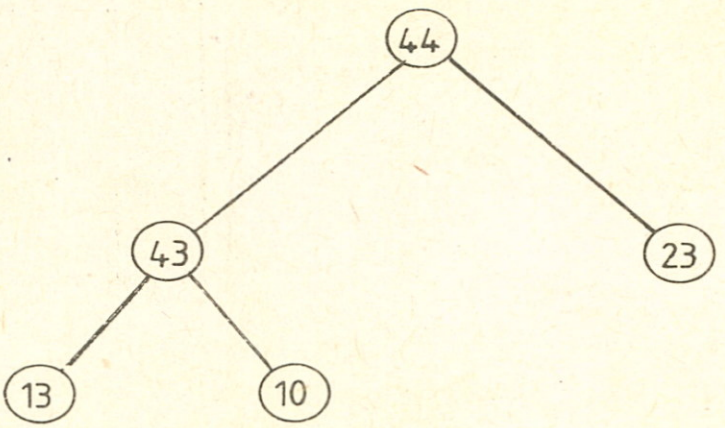
1. fázis:
Résztömb 3,. . .,6:
A 76-os elemet a 44-gyel kell összehasonlítani. Nincs csere.
Résztömb 2,. . .,6:
A 43-at kell a 13-mal és a 10-zel összehasonlítani. Nincs csere.
Résztömb 1,. . .,6:
A 23-at kell a 43-mal és a 76-tal összehasonlítani. Csere először 76-tal, ezt követően a 44-es elemmel.



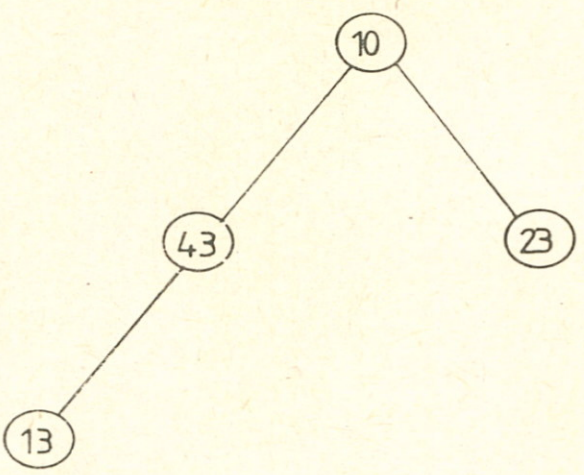
2. fázis:
A 76-os és 23-as cseréje a fa gyökerébe viszi a 23-at. A tömb mérete 5-re csökkent.



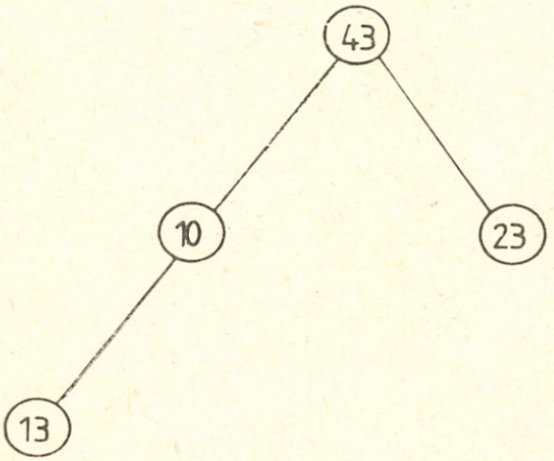
1. fázis:
Résztömb 2,. . .,5:
A 43-as elemet a 13-mal és 10-zel kell összehasonlítani. Nincs csere.
Résztömb 1,. . .,5:
A 23-as elemet a 44-gyel és 43-mal kell összehasonlítani. Csere a 44-essel.



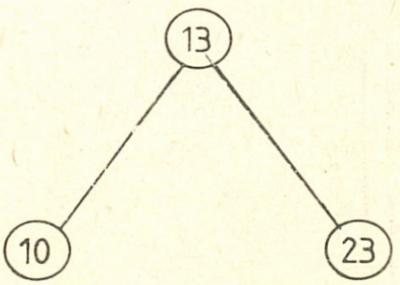
2. fázis:
A 44-es és 10-es cseréje a 10-est a fa gyökerébe teszi. A tömb hossza 4-re csökken:



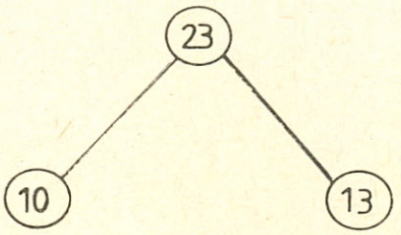
Résztömb 2,. . .,4:
A 43-as és 13-as elemet kell összehasonlítani. Nincs csere.
Résztömb 1,. . .,4:
10-es elemet a 43-mal és 23-mal kell összehasonlítani. Csere a 43-assal.



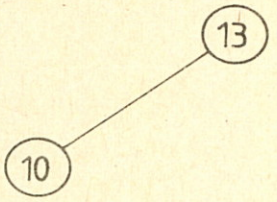
2. fázis:
A 43-as és 13-as cseréjével a 13-as a fa gyökerébe kerül. A tömb mérete 3-ra csökken.



1. fázis:
Résztömb 1,3:
A 13-as elemet a 23-assal kell kicserélni.



2. fázis:
A 23-as és a 13-as cseréje ismét a fa gyökerébe viszi a 13-at.
A tömb hossza 2-re csökken.



1. fázis:
Résztömb 1,2:
A 13-as elemet a 10-zel összehasonlítva, nincs csere.

2. fázis:
A 13-as és a 10-es cseréje a tömb végleges rendezését jelenti.
A következőkben a fastruktúrás rendezés (Heapsort) eljárásnak *Turbo-Pascal* változatát ismertetjük.

```
{RENDEZ7.PAS    }
{ Pr. II. 1.7. }

PROCEDURE Rendez (n : integer);

{ Rendezes fastrukturaval (heapsort) }

VAR
   i : integer;

   PROCEDURE verem (i,j : integer);

   VAR
      k : integer;

   BEGIN
      k:=2*i;
      if k <= j
       then begin
         if k < j then if s[k].kulcs < s[k+1].kulcs
                       then k:=k+1;
         if s[i].kulcs < s[k].kulcs
          then begin
            csere(i,k);
            verem(k,j);
          end;
       end;
   END; {.....................................}

BEGIN
   for i:=n div 2 downto 1 do verem(i,n);
   for i:=n downto 2
    do begin
      csere(1,i);
      verem(1,i-1);
    end;
END; {.....................................}
```

A fastruktúrás rendezés analízise

Első pillantásra ez a rendezés nem tűnik hatékonynak. Tekintsük az időfelhasználást. Mivel a *B* függvény kiértékelése egyszerűbb, ezzel kezdjük az analízist.
A *B* függvény meghatározása:

a) A legkedvezőtlenebb eset.
Tekintsük először a fastruktúrás eljárást:
Tételezzük fel, hogy minden elem a mindenkori gyökér pozíción felcserélődik, amilyen gyakran csak lehet. A tömb adott *i* és *j* határainál, amelyek *részfát* alkotnak, azt kell megállapítanunk, hogy milyen magasságú a részfa. Az *i* indexet addig szorozhatjuk 2-vel, amíg nagyobb lesz *j*-nél.
Ez az alábbi képlethez vezet:

$$i*2^k <= j$$

átrendezve:

$$2^k <= j/i$$

mindkét oldalt logaritmálva:

$$k = G(\lg(j/i))$$

A fejezet elején már említettük, hogy a *G* függvény a legközelebbi nagyobb egész szám. Helyettesítsük be az $m = G(c\_max/2)$ kifejezést, ezzel az első **For** ciklusban az eljárás *m* darabszámú nem rekurzív hívását végezhetjük el. Összesen tehát

$$\Sigma\ (G(g\ (c\_max/k))) \quad (\text{ahol } k = 1,2,\ldots,m)$$

Ha nem a legközelebbi nagyobb egész számot vesszük, akkor a közelítés hibája csak mértéktartóan nő. Az alábbi logaritmikus azonosságok felhasználásával

$$\lg(x/y) = \lg(x) - \lg(y) \text{ és } \lg(x*y) = \lg(x) + \lg(y)$$

átírhatjuk a képletünket:

$$\Sigma\ (\lg(c\_max) - \lg(k)) = m * \lg(c\_max) - \lg(m!) \text{ (ahol } k=1,\ldots,m)$$

Ha képletgyűjteményünkben megtalálhatjuk (a Stirlings-féle közelítés alkalmazásánál *m*!-hoz) az alábbi érdekes összefüggést:

$$\lg(m!) \text{ körülbelül } m * \lg(m) - m$$

Továbbá, mivel

$$m = G(c\_max/2)$$
$$\lg(m) = \lg(c\_max) - 1$$

miután mindkét oldalt logaritmáltuk. Ezzel

$$m * \lg(c\_max) - \lg(m!)$$

értékű körülbelül

$$m * \lg(c\_max) - (\lg(c\_max - 1)) * m + m = O(c\_max)$$

Következőnek becsüljük meg a második **For** ciklus ráfordítását.
A fenti eredményt felhasználhatjuk, mivel ismét a fa eljárást használjuk. Itt $c\_max - 1$ elemcserét kell hozzávenni. Az összeg a következő:

$$\Sigma\ (\lg(k)) + c\_max - 1 \quad (\text{ahol } k = 2,3,\ldots,c\_max)$$

Amely

$$\lg(c\_max!) + c\_max - 1$$

A fenti becslés felhasználásával a közelítés

$$c\_max * \lg(c\_max) - 1$$

B értéke összesen tehát:

$$B(c\_max) = O(c\_max * \lg(c\_max))$$

b) Átlagos eset
Az analízis ebben az esetben igen bonyolult. A gyakorlati vizsgálatok közel azonos értéket adnak, mint a legrosszabb esetnél:

$$B(c\_max) = O(c\_max * \lg(c\_max))$$

c) A legjobb eset
Nem egyszerű átlátni, mivel úgy tűnik, hogy a fordított rendezésnél adódna a legjobb érték. Az első **For** ciklusban nincs csere, azonban a második **For** ciklusban végrehajtott cserékkel a legkisebb elem a gyökérpozícióba kerül. Emiatt ismét néhány cserét kell végrehajtani, hogy előállítsuk a fát.
Szerencsére tudjuk, hogy a legjobb érték semmi esetre sem lehet rosszabb, mint

$$O(c\_max * \lg(c\_max))$$

A S függvény meghatározása
A kulcsok összehasonlítási száma közelítően a mozgatások számának kétszerese. Ezért átvehetjük onnan az eredményeket.
A legkedvezőtlenebb, az átlagos és a legjobb esetben

$$S(c\_max) = O(c\_max * \lg(c\_max))$$

1.2.1. Alapvető meggondolások

Már a fejezet első részében is kizárólag a tömbökre koncentráltunk. Alkalmaztuk a már ismert adatdefiníciót.

```
{ Seged1.2 }
{ S. II. 1.2. }

VAR s : array[1..c_max] of record
                             kulcs : t_kulcs;
                             adat  : t_adat;
                           end;
```

A c_max konstansnak olyan nagynak kell lennie, hogy a tömb illeszkedjen a rendelkezésre álló tárterülethez.
Feladatunk a megadott kulcshoz tartozó információ elérése. Míg a rendező eljárások kiindulópontja mindig azonos volt, a kereső eljárások között vannak olyanok, amelyek bizonyos rendet feltételeznek. A módszerek osztályozása nehéz. Kétféle módszer különböztethető meg:

a) Amelyik nem tételez fel előzetes rendezettséget.
   Pl.: A lineáris keresés (1.2.2. pont)

b) Amelyik bizonyos rendezettséget feltételez
   Pl.: A bináris keresés (1.2.3. pont)
   A hasításos keresés (Hashing) (1.2.4. pont)

Az eljárás sebességének becslésére itt csak az *S* függvényt, a kulcsok, összehasonlítások számát kell meghatározni.
Vegyük legelőször a legegyszerűbb és leggyakrabban alkalmazható algoritmust, a lineáris keresést, amelynél az elemeknek a tömbben való rendezettsége a legkevésbé sem érdekes.

1.2.2. A lineáris keresés

Először az első elemmel kezdi és a keresett elem kulcsával összehasonlítja. Egyenlőség esetén már meg is találta az elemet. Ha nem egyenlő, veszi a második elemet, összehasonlítja stb. A keresés akkor fejeződik be, ha megtalálta, vagy ha a tömb utolsó indexét c_max-ot elérte. Ha az index c_max, akkor a keresett elem a tömbben nem található.
Az algoritmus annyira egyszerű, hogy a példát megtakarítjuk és rögtön a programra térünk. Az ebben található Lin_Keres függvény ismét alkalmazza a beszúrással való rendezésnél megismert trükköt a kulcsnak a tömbhöz való hozzáfűzését, így a vizsgálat nem fut túl az utolsó tömbelemen.

```
{ LinKer.PAS   }
{ Pr. II. 1.8. }

PROGRAM Linearis_Kereses;

CONST
   c_max = 1Ø1;

TYPE
   t_r   = integer;
   t_s   = array [1..c_max] of t_r;

VAR
   s       : t_s;
   kulcs, i : integer;

FUNCTION Lin_Keres (keres_kulcs : integer;
                                n         : integer) : integer;

VAR
   i : integer;

BEGIN
   s[n+1]:=keres_kulcs;
   i:=1;
   while s[i]<> keres_kulcs do i:=i+1;
   Lin_Keres:=i;
END; {.....................................................}

BEGIN
   for i:=1 to c_max-1 do s[i]:=trunc(random*32767);
   for i:=1 to c_max-1
    do begin
      Write(s[i]:7);
      if i mod 1Ø = Ø then writeln;
    end;
   Writeln;
   Write('KERESENDO KULCS: ');
   Read(kulcs);
   Writeln;
   i:=Lin_Keres(kulcs,c_max-1);
   if i < 1Ø1 then
              Writeln('HELY: ',i:5)
              else
              Writeln('A MEGADOTT KULCS NINCS A TABLABAN')
END.
```

A lineáris keresés analízise

Az S függvény meghatározása:

a) A legkedvezőtlenebb eset

A legkedvezőtlenebb eset beláthatóan akkor lép fel, ha a keresett elem nem található a tömbelemek között. A beírt kulcsot is beszámítva a szükséges összehasonlítások száma

$$c\_max + 1$$

b) Átlagos eset

Tekintsük az egyik kulcsot a másik után és szorozzuk a szükséges összehasonlítások számát a kulcsok előfordulási valószínűségével. Azonos valószínűség esetén ennek az értéke

$$1/(c\_max+1).$$

Ezzel

$$1/(c\_max+1)+2/(c\_max+1)+\ldots+(c\_max+1)/(c\_max+1)$$

A c_max+1 magyarázataként utalunk arra az esetre, ha az összehasonlítási kulcsot csak a hozzáfűzött kulcsban a c_max+1-edik elemben találja meg. Az 1-től n-ig terjedő számok ismert összegzési képletének segítségével egyszerűsíthetjük képletünket

$$(c\_max+1)*(c\_max+2)/2*(c\_max+1)=(c\_max+2)/2 = O(c\_max)$$

c) A legjobb eset
Akkor lép fel, ha az első elem rögtön a keresett. Ebben az esetben

$$S(c\_max) = 1$$

Mivel ez érvényes minden itt említett kereső eljárásra, a jövőben az analízisnél a legjobb esetet elhagyjuk.

1.2.3. Bináris keresés

Az előző fejezetben megtárgyalt algoritmus a rövid tömbökre alkalmas, hosszabbak esetén több mint lassú. Ha a tömböt az előzőekben ismertetett módszerek egyikével már rendeztük, akkor lényegesen jobb eljárást alkalmazhatunk.
Az első elem helyett a c_max div 2 indexű elemmel kezdünk, és azt összehasonlítjuk a keresési kulccsal, ha a keresési kulcs kisebb, akkor a keresett elem a tömb alsó részében található, ha a keresési kulcs nagyobb, akkor a felső részében. Egyenlőség esetén megtaláltuk az elemünket. A módszert rekurzív módon alkalmazva az érintett félhosszúságú résztömbre akkor fejezzük be a keresését, ha megtaláltuk a keresett elemet vagy a résztömb nagysága elérte az 1-et.

Példa:

a rendezett tömb;	10 13 17 23 43 44 76 86 89 90 96 99	
a keresett elem a 17-es		
a bal oldali fél	10 13 17 23 43	
a jobb oldali fél	17 23 43	
a bal oldali fél	17	megvan a keresett elem.

Magyarázat:
A 17-es és a 44-es összehasonlítása után a bal oldali felet tovább vizsgálja, mivel a keresési kulcs kisebb. Ezután a 13-assal hasonlítja össze, majd a jobb oldali félben keres tovább. A 23-assal való összehasonlítás mutatja, hogy a keresett elemnek attól balra kell elhelyezkednie. A következő összehasonlítás végül megtalálja a 17-es elemet.
Közlünk egy rekurzív és egy nem rekurzív *Turbo–Pascal* eljárást és a hozzá tartozó keretprogramot. A rendezéshez az 1.1.5. pontban tárgyalt fastruktúrás rendezést alkalmaztuk.

```
{ KERHIVO.PAS  }
{ Pr. II. 1.9. }

PROGRAM RendHivo;

CONST
   c_max = 500;

TYPE
   t_r    = record
              kulcs  : integer;
              info : string[80];
            end;

   t_s    = array [1..c_max] of t_r;

VAR
              s : t_s;
              i : integer;
   keres_kulcs : integer;

PROCEDURE Csere (i,j : integer);

VAR
   h : t_r;

BEGIN
   h:=s[i];
   s[i]:=s[j];
   s[j]:=h;
END; {.....................................................}

{$I BINKER1.PAS}     { A megf. kereso elj. beillesztese }
{$I RENDEZ7.PAS}     { A fastrukturas (heap) rendezes   }

BEGIN
  for i:=1 to c_max do s[i].kulcs:=trunc(random*32767);
  Rendez(c_max);
  Writeln('BEVITEL:');
  for i:= 1 to c_max
   do begin
      Write(s[i].kulcs:7);
      if i mod 10 = 0 then Writeln;
   end;
  Writeln;
  Write('KERESENDO KULCS: ');
  Read(keres_kulcs);
  Writeln;
  Writeln('HELY: ',Bin_Ker(1,c_max,keres_kulcs));
END.


{ BINKER1.PAS     }
{ Pr. II. 1.9./A }

FUNCTION Bin_Ker (l, r, keres_kulcs : integer):integer;

VAR
   m : integer;

BEGIN
   if r < l
    then Bin_Ker:=0
    else begin
      m:=(l+r) div 2;
      if keres_kulcs < s[m].kulcs
```

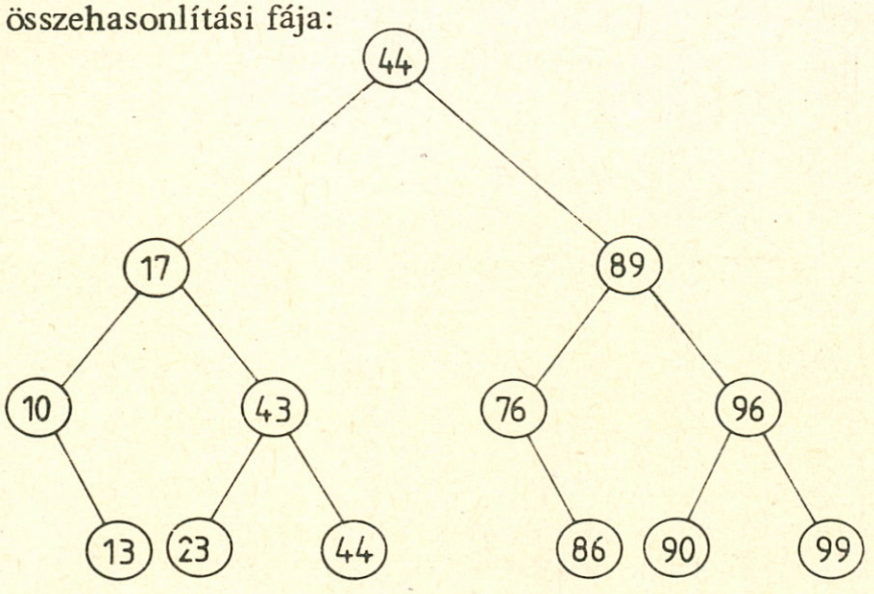
```
        then Bin_Ker:=Bin_Ker(l,m-1,keres_kulcs)
        else
         if keres_kulcs > s[m].kulcs
          then Bin_Ker:=Bin_Ker(m+1,r,keres_kulcs)
          else Bin_Ker:=m;
    end;
END; {.....}



{ BINKER2.PAS     }
{ Pr. II. 1.9./B }

FUNCTION Bin_Ker (l, r, keres_kulcs : integer):integer;

VAR
   m : integer;

BEGIN
   repeat
     m:=(l+r) div 2;
     if keres_kulcs < s[m].kulcs
      then r:=m-1
      else
       if keres_kulcs > s[m].kulcs
        then l:=m+1
        else begin
          Bin_Ker:=m;
          exit;
        end;
   until r < l;
   Bin_Ker:=Ø;
END; {.....}
```

A bináris keresés analízise

a) A legkedvezőtlenebb eset
A lineáris kereséshez hasonlóan itt is az a legkedvezőtlenebb eset, ha a keresett elem a tömbnek nem eleme. Az S előállítására megrajzoltuk az algoritmus összehasonlítási fáját, l. még az 1.1.1. pontot). Közben azt az egyszerűsítést követjük, hogy mindkét összehasonlítást egy csomópontként ábrázoltuk.

A tömb: 1Ø 13 17 23 43 44 76 86 89 9Ø 96 99

A bináris keresés összehasonlítási fája:



Az összehasonlítási fa bináris, ezért a magassága

$$\lg(c\_max + 1)$$

Minden választáshoz két összehasonlítás kell, ezért az összehasonlítások száma a legkedvezőtlenebb esetben

$$2 * \lg(c\_max+1) = O(\lg(c\_max))$$

b) Átlagos eset
Az összehasonlítási fa segítségével kimutatható, hogy az eredményes átlagos keresés értéke

$$O(\lg(c\_max))$$

Ezzel az átlagos és legkedvezőtlenebb esethez tartozó érték elég nagy *c_max* esetén alig különbözik!
A bináris keresés nagy *c_max* esetén bámulatos teljesítményű. Például: 1000 elemű tömbből 10 összehasonlítással eléri a keresett elemet. Ennek ellenére, létezik egy – néhány esetben – még eredményesebb eljárás a hasítás (Hashing). Előbb azonban lássuk az 1.1.2. pontban megígért változatot a beszúrással történő rendezéshez, ahol a beszúrási helyek megtalálására a bináris keresést alkalmazzuk.

```
{ RENDEZ6.PAS    }
{ Pr. II. 1.10.  }

PROCEDURE Rendez (n : integer);

{ Rendezes beszurassal, binaris keresessel javitva }

VAR
   i,j,k : integer;
   h     : t_r;

   FUNCTION Bin_Ker (l,r,keres_kulcs:integer):integer;

   VAR
      m : integer;

   BEGIN
      repeat
        m:=(l+r) div 2;
        if keres_kulcs < s[m].kulcs
         then r:=m-1
         else if keres_kulcs > s[m].kulcs
                then l:=m+1
                else begin
                  Bin_Ker:=m;
                  exit;
                end;
      until r < l;
      Bin_Ker:=l;
   END; {.....................................}

BEGIN
   for i:=2 to n
    do begin
      if s[i].kulcs < s[i-1].kulcs
```

```
            then begin
              h:=s[i];
              k:=Bin_Ker(1,c_max,..[i].kulcs);
              for k:=i-1 downto k do move(s[k],s[k+1],2);
              s[k]:=h;
            end;
        end;
END;
```

1.2.4. A hasításos keresés (Hashing)

Míg az előbbi pontban megtárgyalt bináris keresés feltételezte, hogy a tömb elemei rendezettek, a hasításos keresés egy lépéssel tovább megy. Megköveteli, hogy az elemek egy függvény segítségével kerüljenek tárolásra és ugyanez a függvény számítsa ki az adott kulcsú elem indexét. E függvény segítségével a keresett elem gyorsan megtalálható.
Ha a kulcsok száma kicsi, kereshetünk olyan egyértelmű indexfüggvényt, amely az összefüggő indexszámokra az egyes kulcsokat leképezi. Azonban gyakran a lehetséges kulcsok nagy darabszámával találkozunk. Például, ha egy címjegyzékkezelőt írunk és a hozzáférési kulcsnak a családnevet választjuk, akkor max. 20 karakter hosszúságú név esetén már kb.

$$26^{20} \text{ a lehetséges kulcsok száma!}$$

Ez természetesen minden mikroszámítógépnek túl nagy szám. Ráadásul ebben a gigantikus tömbben csak kevés hely volna kihasználva. A többi üres maradna. Ezért indexfüggvényeket szerkesztettek, amelyek több lehetséges kulcsot egy helyre képeznek le a tömbben. Természetesen arra is vezethet ez a leképzés, hogy egy kulcsot olyan helyre kellene írni, amely már foglalt. Ebben az esetben belép az *ütközési* kezelés, amelyet még tárgyalni fogunk. Először a hasítófüggvényre koncentráljunk, így nevezzük ebben az esetben az indexfüggvényt.
Az áttekinthetőség kedvéért tekintsük először a működését:

a) a tömb inicializálása, hogy minden hely szóközzel legyen feltöltve;
b) az elemek beszúrása a hasítófüggvény segítségével, az ütközések feloldása az ütközési kezelőrutin segítségével;
c) az elemek keresése a hasítófüggvényük kiszámításával, ha az elemek nem a számított helyükön találhatók, akkor felkutatásuk az ütközési kezelőrutin segítségével.

A hasítófüggvény kiválasztása
A függvénynek két fontos követelményt kell kielégíteni:

a) a könnyű és gyors számíthatóság;
b) a meghatározott ismérveknél a kulcshalmozódás elkerülése.

A szokásos hasítófüggvény a kulcsot részekre hasítja (innen a neve).

Kétféle megoldást mutatunk be:

1. *Redőzés.* A kulcs széthasítása különféle részekre és a részek kombinálása szorzással vagy összeadással.

2. *Modulképzés.* A kulcs átszámolása egész számmá (esetleg az 1-nél közöltek szerint) és maradékképzés (MOD-függvény) a tömbnagysággal. Itt a tömbméretre a legjobb választás egy primszám, amely a kulcsok egyenletes elosztását garantálja.

1. Nyílt címzés

A legegyszerűbb megoldás, ha már lefoglalt helyre bukkannak, hogy az indexet 1-gyel növeljük addig, amíg szabad helyet nem találunk (ha elérjük a tömb felső határát, akkor 1-gyel kezdjük újra). A körforgás és a prímszámra választott tömbmérettel végzett maradékképzés a biztosíték arra, hogy minden helyet megvizsgál.

új_hely=(hasítóérték+i) mod tömbméret

Ezt *lineáris* módszernek nevezik. A hátránya kézenfekvő: Az eljárás elősegíti a másodlagos tárolódás (cluster) képződést. Ez azt jelenti, hogy a letett elemek olyan hosszú lánca áll elő, amely a beszúrási időt és a keresési időt számottevően megnyújtja. Az elsődleges torlódás képződésről akkor beszélünk, ha már a hasítófüggvény túl hosszú, egymást követő kulcsokat ad.
A szituáció javítható azzal, hogy egy második, úgynevezett belső elválasztó (Inkrementum) függvényt használunk a legközelebbi szabad hely elérésére. Ez a függvény természetesen a kulcsértékből indul ki. A neve újrahasító (Rehashing) vagy kontrahasító (Double hashing).

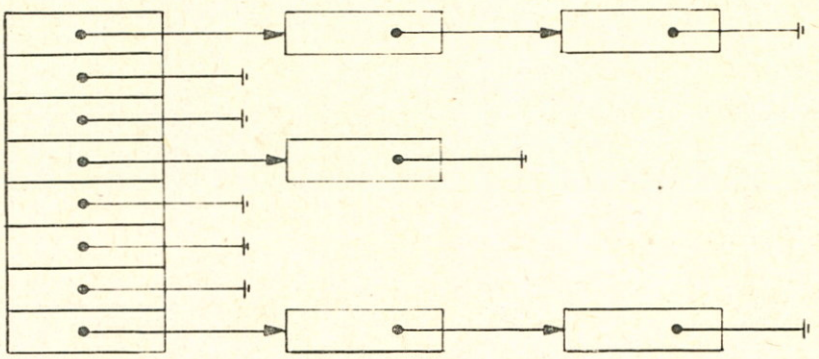
A továbblépéshez szükséges függvényérték meghatározása után ismét a lineáris módszer kerül alkalmazásra.

$$\text{új\_hely} = (\text{hasítóérték} + i * \text{inkérték}) \bmod \text{tömbméret}$$

Van még más lehetőség is természetesen, csak arra kell ügyelni, hogy a megvizsgált helyek az egész tömböt lefedjék.

2. Láncolás (Chaining)

Bár a fejezet elején a tömb adatszerkezetének leírására korlátozódtunk, a láncolást a listaszerkezet segítségével kívánjuk tárgyalni, mivel az jó lehetőséget ad az ütközések bemutatására. Az elvet a következő ábra mutatja:



A hasítótáblázatba való bevitel mutatókat tartalmaz az ütközési láncra vonatkozóan. Ezek *Nil*-ek, ha nincs elembevitel. Mivel a mutató értékére csak 2 byte-ot kell feláldozni, ez komoly helymegtakarítást jelent elég nagy adatrekordok esetében. Ha az adatrekordok rövidek, akkor a helyszükségletre a mutatóhasználat rossz hatásfokú.

Mielőtt megkísérelnénk a hasítás analízisét, közlünk egy olyan programpéldát, amely a hasításos keresést alkalmazza.

A program három funkciójú:

a) Beolvas egy *Turbo–Pascal* programot, majd kinyomtatja úgy, hogy a *Turbo–Pascal* kulcsszavak kövér betűkkel kiemeltek.

b) Továbbá keresztreferencia-listát készít azokról a szavakról, amelyek nincsenek kövér betűkkel nyomtatva. Közben meg lesznek különböztetve
   – az eljárások
   – a függvények
   – a változók
   – a konstansok
   – a típusok stb.

c) Külön listára felírja az összes eljárás- és függvénynevet az áttekinthetőség kedvéért.

Működés:
A programot karakterenként olvassa be és kiválasztja a *Turbo–Pascal* kulcsszavakat. Ehhez a bináris keresést használja, amely erre a legalkalmasabb, mivel a *Turbo–Pascal* kulcsszavai kötöttek, s ezért táblázatban rendezve tárolhatók.

A szavak könyvelése, amelyeket a keresztreferencia-listába bevitt, a hasítás segítségével valósult meg (nyílt címzés mint ütközési kezelés). Az egyes táblázati elemek tartalmazzák a kulcsszót és egy mutatót, amely a kulcsszóhoz tartozó sorszámokat tartalmazó rekordra mutat. Ez a dinamikusan generált rekord legfeljebb 20 ilyen sorszámot vehet fel, ha további számok kerülnek elő, egyszerűen új rekordot generál és a megfelelő mutatóhoz fűzi. Az utolsó rekord mutatója mindig *Nil* tartalmú. A kulcsszavak típusát is a hasítótáblázat elemeiben tárolja.

A program beolvasása után a hasítótömb (Hasharray) rendezésre kerül, mert a hasítófüggvény nem rendez. Ezután minden elemet, amely bejegyzést tartalmaz, kinyomtat.

```
{ NYOMTAT.PAS    }
{ Pr. II. 1.11   }

PROGRAM Nyomtat;

CONST
   c_kszo       =  49;
   c_maxsor     =  58;
   c_tombmeret  = 997;
   c_max        =  20;
   c_deklmax    =  50;
```

```
TYPE
   t_sor      =  string [255];
   t_nev      =  string [12];
   t_tetel    =  string [20];
   t_p_znr    =  ^t_znr;
   t_znr      =  record
                    max    : integer;
                    znr    : array [1..c_max] of integer;
                    p_next: t_p_znr;
                 end;

   t_typ        = (fu,pr,va,co,ty,null);

   t_elem    = record
                    kulcs : t_tetel;
                    art : t_typ;
                    p_next : t_p_znr;
                end;

VAR
   ch  : char;
   str,bez,tetel : t_sor;
   j,cc,zz,sz,n : integer;
   sor : t_sor;
   prg_in,inc_in : text;
   prg_nev, inc_nev : t_nev;
   fej : string [50];
   i:integer;
   file_vege, inc, normal : boolean;
   hasittomb : array [1..c_tombmeret] of t_elem;
   typ : t_typ;
   deklare : array [1..c_deklmax] of t_sor;
   dekl_voll : integer;


CONST
   kulcs_szo : array [1..c_kszo] of string[9] =

      ('ABSOLUTE','AND','ARRAY','BEGIN','BOOLEAN',
       'BYTE','CASE','CHAR','CONST','DIV','DO','DOWNTO'
       'ELSE','END','EXTERNAL','FILE','FOR','FORWARD',
       'FUNCTION','GOTO','IF','IN','INLINE','INTEGER',
       'LABEL','MOD','NIL','NOT','OF','OR','PACKED',
       'PROCEDURE','PROGRAM','REAL','RECORD','REPEAT'
       'SET','SHL','SHR','STRING','TEXT','THEN','TO'
       'TYPE','UNTIL','VAR','WHILE','WITH','XOR');

{$I JELOLES.PAS}

PROCEDURE Rendez (n : integer);

{ Rendezes fastrukturavel (heap) }

VAR
   i : integer;

   PROCEDURE Csere (i,j : integer);

   VAR
      h : t_elem;

   BEGIN
      h:=hasittomb[i];
      hasittomb[i]:=hasittomb[j];
      hasittomb[j]:=h;
   END; {.....}
```

```
PROCEDURE Fa (i,j : integer);

VAR
   k : integer;

BEGIN
   k:=2*i;
   if k <= j
    then begin
    if k < j
     then if hasittomb[k].kulcs<hasittomb[k+1].kulcs
               then k:=k+1;
      if hasittomb[i].kulcs < hasittomb[k].kulcs
       then begin
         Csere(i,k);
         Fa(k,j);
       end;
    end;
END; {.....}


BEGIN
   for i:=n div 2 downto 1 do fa(i,n);
   for i:=n downto 2
    do begin
      Csere(1,i);
      Fa(1,i-1);
    end;
END; {.....}

FUNCTION Bin_Ker (l, r : integer;
                  keres_kulcs : t_tetel) : integer;

VAR
   m : integer;

BEGIN
   repeat
     m:=(l+r) div 2;
     if keres_kulcs < kulcs_szo[m]
      then r:=m-1
      else
       if keres_kulcs > kulcs_szo[m]
        then l:=m+1
        else begin
          Bin_Ker:=m;
          exit;
        end;
   until r < l;
   Bin_Ker:=Ø;
END; {.....}

FUNCTION Hasito (tetel : t_sor) : integer;

VAR
   i,sum : integer;

BEGIN
   sum:=Ø;
   for i:=1 to length(tetel) do sum:=sum+ord(tetel[i]);
   Hasito:=(sum mod c_tombmeret)+1;
END; {.....}

PROCEDURE Init_hasittomb;

VAR
   i : integer;
```

```
BEGIN
   for i:=1 to c_tombmeret do hasittomb[i].kulcs:=' ';
END; {.....}

FUNCTION Foglalt (w : integer) : boolean;

BEGIN
   Foglalt:=hasittomb[w].kulcs <> ' ';
END; {.....}

PROCEDURE Bevitel (tetel : t_sor);

VAR
   i,inkertek,ertek,uj_hely : integer;
   p : t_p_znr;

   PROCEDURE Znr_bevitel(i:integer);

   VAR
      p,q    : t_p_znr;
      talalt : boolean;

   BEGIN
      p:=hasittomb[i].p_next;
      repeat
        if p^.max < c_max
         then talalt:=true
         else
          if p^.p_next = nil
           then begin
             new(q);
             q^.max:=Ø;
             q^.p_next:=nil;
             p^.p_next:=q;
             p:=q;
             talalt:=true;
           end
           else p:=p^.p_next;
      until talalt;
      with p^
       do begin
         max:=max+1;
         znr[max]:=zz;
       end;
   END; {.....}

   BEGIN
      ertek:=Hasito(tetel);
      if Foglalt(ertek)
       then begin
         if hasittomb[ertek].kulcs = tetel
          then begin
            Znr_bevitel(ertek);
            exit;
          end;
         inkertek:=ord(tetel[1]);
         i:=Ø;
         repeat
           i:=i+1;
           uj_hely:=(ertek+inkertek) mod c_tombmeret;
           if hasittomb[uj_hely].kulcs = tetel
            then begin
              Znr_bevitel(uj_hely);
              exit;
            end;
         until (not Foglalt(uj_hely)) or (uj_hely=ertek);
         if uj_hely = ertek
```

```
        then begin
          Writeln('A hasitotomb tul kicsi! VEGE!      ');
          Halt;
        end;
        ertek:=uj_hely;
     end;
    new(p);
    p^.max:=1;
    p^.znr[1]:=zz;
    p^.p_next:=nil;
    with hasittomb[ertek]
     do begin
       kulcs:=tetel;
       art:=typ;
       p_next:=p;
     end;
END; {.....................................................}

PROCEDURE Kiir_hasittomb;

VAR
   p_run : t_p_znr;
   i,j,k : integer;

BEGIN
   Rendez(c_tombmeret);

   Writeln(lst,chr(12));
   Writeln(lst,'KERESZT REF. LISTA');
   Writeln(lst,'------------------');
   Writeln(lst);
   Writeln(lst);
   for i:=1 to c_tombmeret
    do if foglalt(i)
         then begin
           case hasittomb[i].art of
             fu:  Write(lst,'FUNCTION  ');
             pr:  Write(lst,'PROCEDURE ');
             va:  Write(lst,'VAR       ');
             co:  Write(lst,'CONST     ');
             ty:  Write(lst,'TYPE      ');
             else Write(lst,'          ');
           end;
           Write(lst,hasittomb[i].kulcs,' ':
                 22-Length(hasittomb[i].kulcs));
           p_run:=hasittomb[i].p_next;
           k:=1;
           while p_run <> nil
            do begin
              for j:=1 to p_run^.max
               do begin
                 Write(lst,p_run^.znr[j]:5);
                 if k mod 11 = 0
                  then begin
                    Writeln(lst);
                    Write(lst,' ':32);
                  end;
                  k:=k+1;
               end;
               p_run:=p_run^.p_next;
            end;
            Writeln(lst);
         end;
END; {.....................................................}
```

```
PROCEDURE Process_file (VAR f : text);

VAR
   i,j,k : integer;

   PROCEDURE Hozz_ch;

   LABEL 99;

   BEGIN
      if cc = Length(sor)
       then begin
         zz:=zz+1;
         if Eof(f)
          then begin
            Close(f);
            file_vege:=true;
            goto 99;
          end;
         if zz mod c_maxsor =1 then Write(lst,chr(12));
         Writeln(lst);
         Write(lst,zz:4,':  ');
         Readln(f,sor);
         sor:=sor+' ';
         cc:=0;
       end;
      cc:=cc+1;
      ch:=sor[cc];
      99:;
   END; {.....................................}

   PROCEDURE Megjegyzes;

   VAR
      ex : boolean;

   BEGIN
      Write(lst,ch);
      Hozz_ch;
      while not ((ch = '}') or eof(f) or ((ch = ')')
                                  and ex))
       do begin
         Write(lst,ch);
         Hozz_ch;
         ex:=false;
         if ch = '*'
          then begin
            Write(lst,ch);
            Hozz_ch;
            ex:=true;
          end;
       end;
      Write(lst,ch);
      Hozz_ch;
    END; {.....................................

    BEGIN
       Hozz_ch;
       file_vege:=false;
       repeat
        if ch in ['a'..'z','A'..'Z']
         then begin
           bez:='';
           tetel:='';
           repeat
             tetel:=tetel+ch;
             bez:=bez+upcase(ch);
```

```
  Hozz_ch;
until
 not (ch in ['a'..'z','A'..'Z','_','Ø'..'9']);
if Bin_Ker(1,c_kszo,bez) <> Ø
 then begin
   if bez = 'FUNCTION'
    then begin
      typ:=fu;
      dekl_voll:=dekl_voll+1;
      deklare[dekl_voll]:=sor;
    end
    else
      if bez = 'PROCEDURE'
       then begin
         typ:=pr;
         dekl_voll:=dekl_voll+1;
         deklare[dekl_voll]:=sor;
       end
       else
        if bez = 'VAR'
         then typ:=va
         else if bez = 'CONST'
                then typ:=co
                else if bez = 'TYPE'
                       then typ:=ty
                       else if bez = 'BEGIN'
                              then typ:=null
                              else
                                 if bez ='PROGRAM'
                                    then begin
                                      dekl_voll:=
                                      dekl_voll+1;
                                   deklare[dekl_voll]
                                              :=sor;
                                    end;


    Bekapcs,
    Write(lst,tetel);
    Kikapcs;
   end
   else begin
     Write(lst,tetel);
     Bevitel(tetel);
     if typ in [fu,pr] then typ:=null;
   end;
end
else
 if ch = ''''
  then begin
    Write(lst,ch);
    Hozz_ch;
    while not ((ch = '''') or Eof(f))
     do begin
       Write(lst,ch);
       Hozz_ch;
     end;
    Write(lst,ch);
    Hozz_ch;
  end
  else
   if ch = '('
    then begin
      Write(lst,ch);
      Hozz_ch;
      if ch = '*' then Megjegyzes;
    end
```

```
            else
             if ch = '{'
              then Megjegyzes
              else begin
                Write(lst,ch);
                Hozz_ch;
              end;
   until file_vege;
END; {.....}

BEGIN
   sor:='';
   ch:=' ';
   cc:=Ø;
   zz:=Ø;
   n:=1;
   while n <= paramcount
    do begin
      str:=paramstr(n);
      if pos('.',str) = Ø then str:=str+'.pas';
      Assign (prg_in,str);
      {$I-}
      Reset (prg_in);
      {$I+}
      init_hasittomb;
      typ:=null;
      dekl_voll:=Ø;
      if IOresult <> Ø
         then Writeln(chr(7),'nem talalhato: ',str)
         else Process_file(prg_in);
      Kiir_hasittomb;
      Writeln(lst,chr(12));
      Reset(prg_in);
      n:=n+1;
    end;
END.
```

A hasításos keresés analízise

Sajnos a hasítás használatakor az eseteket nem olyan könnyű megkülönböztetni, mint a bináris keresésnél. Túl sok faktor, mint például a hasítófüggvény, a számításához szükséges idő és természetesen a hasítótömb mérete a várt kulcs vonatkozásában, továbbá az ütközési kezelés játszik meghatározó szerepet. Definiáljuk ezért a hasítótáblázat töltési faktorát a *Lambdát:*

Legyen a bevitt elemek száma n és a tömb mérete c_max, így

$$\text{Lambda} = n/c\_max$$

Ez azt jelenti, hogy nyílt címzés esetén az üres táblázathoz Lambda = 0 és a teli táblázathoz Lambda = 1 tartozik.

A láncolás alkalmazásakor a dinamikus listák miatt a Lambda egynél lényegesen nagyobb.

Különbséget teszünk az ütközési kezelés szerint és feltételezzük, hogy a hasítófüggvény számításához szükséges idő nem szignifikáns.

a) *Láncolás*

Ez a könnyebb eset. Ha a keresés eredménytelen k hosszúságú lánc esetén k

összehasonlítást kell elvégezni. Tételezzük fel, hogy az elemek a hasítótáblázatban azonos valószínűséggel helyezkednek el, akkor a vizsgálat alatt álló listában a várt elemek darabszáma: Lambda. Ezért az eredménytelen keresés esetére az összehasonlítások átlagos darabszáma

$$S(c\_max) = \text{Lambda}$$

Eredményes keresés esetén ugyanúgy, mint a lineáris keresésnél, a lista felét kell megvizsgálni. Némi elhanyagolással írhatjuk:

$$S(c\_max) = \text{Lambda}/2$$

b) *Nyílt címzés*
Ez nehezebben képezhető (mértani sorral és hasonló módszerekkel) és mindenekelőtt az ütközési kezeléstől (lineáris inkrementum függvény) függ. A lineáris ütközési kezelés esetére kész összefüggést közlünk, ha a keresés sikertelen:

$$(1+1/(1-\text{Lambda}^2))/2$$

valamint, ha sikeres:

$$1+1(1-\text{Lambda}))/2$$

2. Fordító (Compiler) és értelmező (Interpreter)

A programkészítő gyakran kerül szembe olyan problémával, hogy felhasználói adatbevivő programot kell készítenie és elemeznie kell a legkülönbözőbb programokat.

A *LOLA* programnyelv és a hozzá tartozó fordító bemutatásával – amennyire lehetséges – ismertetjük, hogyan valósítható meg a *Turbo–Pascal*-ban egyszerű bevitellel rendelkező programnyelv elve minimális programozási ráfordítással.

2.1. A LOLA PROGRAMNYELV

Minden programnyelv alapja a szintaxis és a szemantika meghatározása.

A szintaxis rögzíti, hogy milyen karakterláncok alkothatják a nyelv jól kialakított programjait és milyenek nem, a szemantika pedig az egyes programok jelentését adja meg.

Programnyelvünk definíciójaként először (példákon keresztül megvilágítva) a szemantikát közöljük, és ezután a szintaxisdiagramok segítségével, a programok formális felépítését adjuk meg.

A *LOLA* ezt a nevet *Lola Montez* tiszteletére kapta. Ezen az egyszerű nyelven mutatjuk be, hogyan lehet erre a célra fordítót kifejleszteni.

2.1.1. Adatkoncepció

A *LOLA* nem tartalmaz változókat más programnyelvek, mint például a *Pascal* értelmezése szerint.
Ehelyett tárolót használ, amelynek elemei *real* típusú értékeket vehetnek fel. Ezeket az elemeket közvetlen és közvetett módon címezhetjük, így mégha bonyolultan is, szimulálhatunk tömböket. Annak jelölésére, hogy egy tárolórekesz tartalmáról van szó, az indexmegadást gömbölyű zárójelbe írjuk.

Ha egy tárolócím az indexszámítás után nem egész szám, akkor a legközelebbi kisebb egész számot alkalmazza címnek.

Példa: (10 + (1)) Ha az (1) tartalma 3.3, akkor a kiértékelés szerinti szám 13.3, a tárolócím pedig 13 lesz.

A konstansokat a program kezdetén deklarálhatjuk. Minden konstans automatikusan *real* típusú.

Példa: KONSTANS pi = 3.14159, és = 1985;

A konstansnevek maximális hossza 10 karakter lehet.

2.1.2. Aritmetikai kifejezések

Az aritmetikai kifejezés konstansokból vagy tárolóelemekből, operátorokból és standard függvényekből áll. A számítás mindig *real* típusú eredményt ad. A tárolóelemekkel konstansokkal és standard függvényekkel végezhető érvényes műveletek:
**hatványozás
* szorzás
/ osztás
+ összeadás
– kivonás
Megjegyzés: csak pozitív egész kitevőre értelmezett (Ford.)
A kiértékelés sorrendje: hatványozás, ezután szorzás és osztás, majd összeadás és kivonás. Azonos sorrendiség esetén balról jobbra történik a kiértékelés. Csúcsos zárójelet akkor használunk, ha a számítási sorrendet magunk akarjuk megkötni.

Példák: 2+3**2−10 eredménye 1
<2+3>**2−10 eredménye 15
12/5*2 eredménye 4, és egy tárolórekesz elérésére való.

A Standard függvények:

abs ⟨x⟩	abszolút érték
gyök ⟨x⟩	négyzetgyök
cos ⟨x⟩	koszinusz
sin ⟨x⟩	színusz
tan ⟨x⟩	tangens
log ⟨x⟩	tizes alapú logaritmus
ln ⟨x⟩	természetes alapú logaritmus
exp ⟨x⟩	exponenciális függvény
egészszám⟨x⟩	egész rész

Példa: exp ⟨ ln<(1)> + ln <(2)>⟩ (1) és (2) szorzata

2.1.3. Logikai kifejezések

Az aritmetikai kifejezésekkel ellentétben, amelyek a *LOLA*-ban mindig valós számot adnak eredményül, a logikai kifejezések értéke igaz vagy hamis. A relációs operátorok az alábbi módon érvényesek a *LOLA* nyelvben.

KISEBB	kisebb
NAGYOBB	nagyobb
=	egyenlő
KISEBB=	kisebb egyenlő
NAGYOBB =	nagyobb egyenlő
#	nem egyenlő

Standard függvényként:

PÁRATLAN ⟨x⟩ megállapítja, hogy ⟨x⟩ páros vagy sem.

Példák:

12 KISEBB 100	igaz értéket ad
(1)+(2)#(1)+(2)	hamis értéket ad

2.1.4. Szöveg konstansok

A *LOLA*-ban szövegkonstanst csak az IR-utasításban használhatunk, hossza legfeljebb 50 karakter lehet.

2.1.5. Programszerkezet

Minden programot a PROGRAM kulcsszóval kezdünk, amelyet a programnév követ. Ezután a konstansokat deklaráljuk. Ha egynél több *LOLA* utasítás fordul elő, akkor ezeket a KEZDET – VÉG utasítászáró jellel kell közrefogni.
A programot pont zárja le.

Példa:

```
{ Negyzet.lola }
{ L. II. 1.    }

PROGRAM  Negyzet;
KONSTANS ketto = 2, beadat = 1, i = Ø;
KEZDET
      OLV (beadat);
      ISMETLES INDUL 1 -> (i) BEFEJEZ (i) NAGYOBB (beadat):
      IR (i),"**2 = ",(i)**ketto
VEG.
```

2.1.6. LOLA utasítások

A *LOLA* utasításokat tetszésünk szerint kis- vagy nagybetűkkel írhatjuk. Elválasztásukra legalább egy szóközt kell használni.

ÉRTÉKADÓ UTASÍTÁS

Formája: aritmetikai kifejezés —> (aritmetikai kifejezés)

Az értékadás balról jobbra történik, mindkét kifejezés kiértékelése után. Jobb oldalon mindig a tárolórekesz-megadás álljon.

Példák:

12 —> (1)
(x)*(y) —> (eredmény)
(((1))) —> (1)

Az utolsó példa indirekt címzést mutat, amelynél először az 1 jelű tárolórekesz tartalmát veszi fel, majd ezen a címen lévő tárolórekesz tartalmát stb. . .

UTASÍTÁSZÁRÓ JEL

Formája: KEZDET *LOLA* utasítások VÉG

Az utasítászáró jel összefüggő utasításként deklarálja a KEZDET és VÉG között levő *LOLA* utasításokat. Jelentése analóg a *Pascal*-beli Begin-End utasítással. Az utasításokat pontosvesszővel zárjuk, kivétel a VÉG előtti utolsó utasítás, amely után tilos pontosvesszőt tenni.

Példa: l. az AMEDDIG utasítást

CIKLUS UTASÍTÁSOK

1. Az AMEDDIG utasítás

Formája: AMEDDIG logikai kifejezés: *LOLA* utasítás

Minden ismétlés előtt kiértékeli a logikai kifejezést és ha igaz, végrehajtja az ismétlést. Különben az AMEDDIG után következő utasítással folytatja.

Példa:

```
{ Szamlal.lola }
{ L. II. 2.    }


   1 -> (szamlalo);
   Ø -> (osszeg);
   AMEDDIG (szamlalo) KISEBB 1ØØ :
      KEZDET
      (osszeg) + (szamlalo) -> (osszeg);
      (szamlalo) + 1        -> (szamlalo)
      VEG;
```

2. Az ISMÉTLÉS utasítás
Formája: ISMÉTLÉS INDUL értékadás BEFEJEZ logikai kifejezés: *LOLA* utasítás

vagy
Formája: ISMÉTLÉS INDUL értékadás BEFEJEZ logikai kifejezés LÉPÉSKÖZ aritmetikai kifejezés: *LOLA* utasítás

Az ISMÉTLÉS utasítás tárolóhelyet inicializál a kezdő értékkel, amelyet a lépésközzel növel, amíg a logikai kifejezés hamis értékű. A kettőspont utáni LOLA utasítást minden értékre végrehajtja. Ha a logikai kifejezés már az első iteráció előtt igaz értékű, akkor az ismétlést egyszer sem hajtja végre.

Példa:

```
{ Ciklus.lola }
{ L. II. 3    }

   Ø -> (osszeg);
   ISMETLES INDUL 1 -> (i) BEFEJEZ (i) NAGYOBB 1ØØ:
            (osszeg) + (i) -> (osszeg)
```

HA UTASÍTÁS

Formája: HA logikai kifejezés AKKOR *LOLA* utasítás
A megadott feltétel teljesülése esetén a *LOLA* utasítást végrehajtja, különben a következő utasítással folytatja.

Példa:
HA páratlan < (be_adat) > AKKOR (be_adat) – 1 –> (be_adat)
A *páratlan* függvény a (be_adat) címen lévő tárolórekeszhez fordul és ha értéke páratlan, akkor értékéből 1 levonásra kerül.

STOP UTASÍTÁS

Formája: STOP
Ez az utasítás a programfutás befejezését okozza.

BE/KIVITELI UTASÍTÁSOK

1. OLV utasítás
Formája: OLV (aritmetikai kifejezés)
Valós szám beolvasását várja a billentyűzetről és az aritmetikai kifejezés által meghatározott tárterületen tárolja.

Példa: OLV (kezdkm)

2. IR utasítás
Formája: IR (aritmetikai kifejezés)
vagy: IR text
vagy: IR aritmetikai kifejezés
Az IR utasítás több argumentumot is tartalmazhat vesszővel elválasztva. Az előre definiált CR konstans is használható, amelynek a hatása kocsi vissza (Carriage Return) és soremelés (Line Feed).

Példák:
IR „HŐMÉRSÉKLET STATISZTIKA", CR
IR (kezdkm),(végkm),(végkm)–(kezdkm)
IR sin ‹‹szög›*PI/180›

CÍM UTASÍTÁS

Formája: CIM aritmetikai kifejezés: *LOLA* utasítás

Ha a *LOLA* utasítás cím kifejezést tartalmaz, akkor a megadott aritmetikai kifejezést ofszetként hozzáadja. Ez az utasítás a tömbök feldolgozásához hasznos.

Példa: l. benzinfogyasztási példát.

Ezzel megismertük az összes *LOLA* utasítást. Most néhány program következik, amelyek a felhasználásukat mutatják be. Végül szintaxisdiagramok formájában összefoglalást adunk.

2.1.7. Példák

A következő *LOLA* program az 1-től n-ig terjedő számok összegét számítja iteratív módon.

```
{ Osszeg.lola }
{ L.  II. 4.  }

PROGRAM Osszeg;
KONSTANS osszeg   = Ø;
         vegertek = 1;
         i        = 2;
KEZDET
  IR "Kerem a vegerteket megadni: ";
  OLV (vegertek);
  HA (vegertek) KISEBB Ø AKKOR
     KEZDET
     IR "Sajnos rossz";
     STOP
     VEG;
  Ø -> (osszeg);
  ISMETLES INDUL 1 ->(i) BEFEJEZ (i) NAGYOBB (vegertek)
     (osszeg)+(i) -> (osszeg);
  IR "Osszeg: ",(osszeg)
VEG.
```

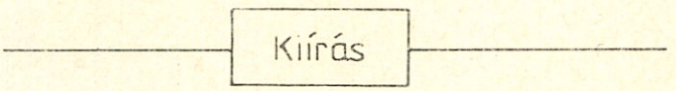
Az alábbi program a benzinfogyasztás mértékét számolja 100 kilométerre. Felváltva olvassa be a kezdeti kilométerállást, a végső kilométerállást és a felvett benzinmennyiséget, majd kiírja a számított értékeket.

```
{ Benzfogy.lola }
{ L.  II. 5.    }

PROGRAM BenzFogy;
KONSTANS kezdkm = 1,
         vegkm  = 2,
         benzin = 3,
         n      = Ø,
         i      = 1;
KEZDET
   IR "Mennyi adat van? ";
   OLV (n);
   ISMETLES INDUL 3 -> (i) BEFEJEZ (i) NAGYOBB (n)*3
            LEPESKOZ 3: CIM (i):
               KEZDET
                  IR CR,"Kezdeti km allas : ";
                  OLV (kezdkm);
                  IR CR,"Veg km allas     : ";
                  OLV (vegkm);
```

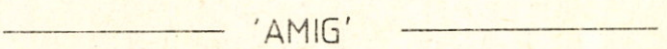
```
                    IR CR,"Benzin              : ";
                   OLV (benzin)
                  VEG;
    IR CR,CR,
       "BENZIN FOGYASZTASI STATISZTIKA",CR,CR,
       "   KEZD. KM   VEG KM   DIFF.      BENZIN  ",
       "FOGYASZTAS",CR,
       "   ----------------------------------------",
       "----------";
    ISMETLES INDUL 3 -> (i) BEFEJEZ (i) NAGYOBB (n)*3
             LEPESKOZ 3: CIM (i):
               IR CR, (kezdkm),(vegkm),(vegkm)-(kezdkm),
               (benzin),(benzin)/<(vegkm)-(kezdkm)>*1ØØ
VEG.
```

2.1.8. Szintaxisdiagramok

A szintaxisdiagram irányított gráf, amelyen áthaladva megállapítható, hogy a program szintaktikusan helyes-e.
A nyilak mellett, amelyek az áthaladás irányát adják meg, a jelkép két fajtáját különböztetjük meg:

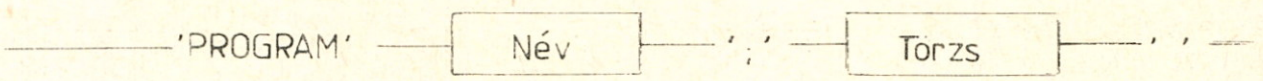


A négyszög további *kifejezés* szintaxis diagram számára van fenntartva, amelyet erre a helyre képzelünk.

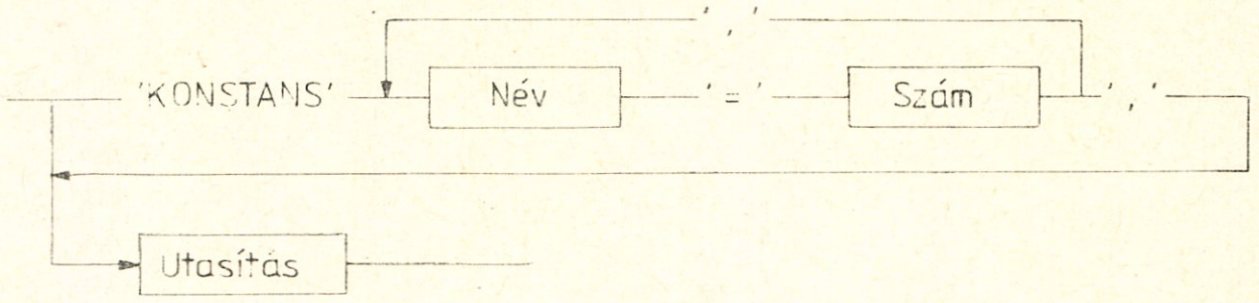


A két aposztrof között állnak az úgynevezett terminál szimbólumok, ezek a programszövegben ténylegesen előforduló karakterek.
Ezzel megadhatjuk a *LOLA* szintaxis diagramját.

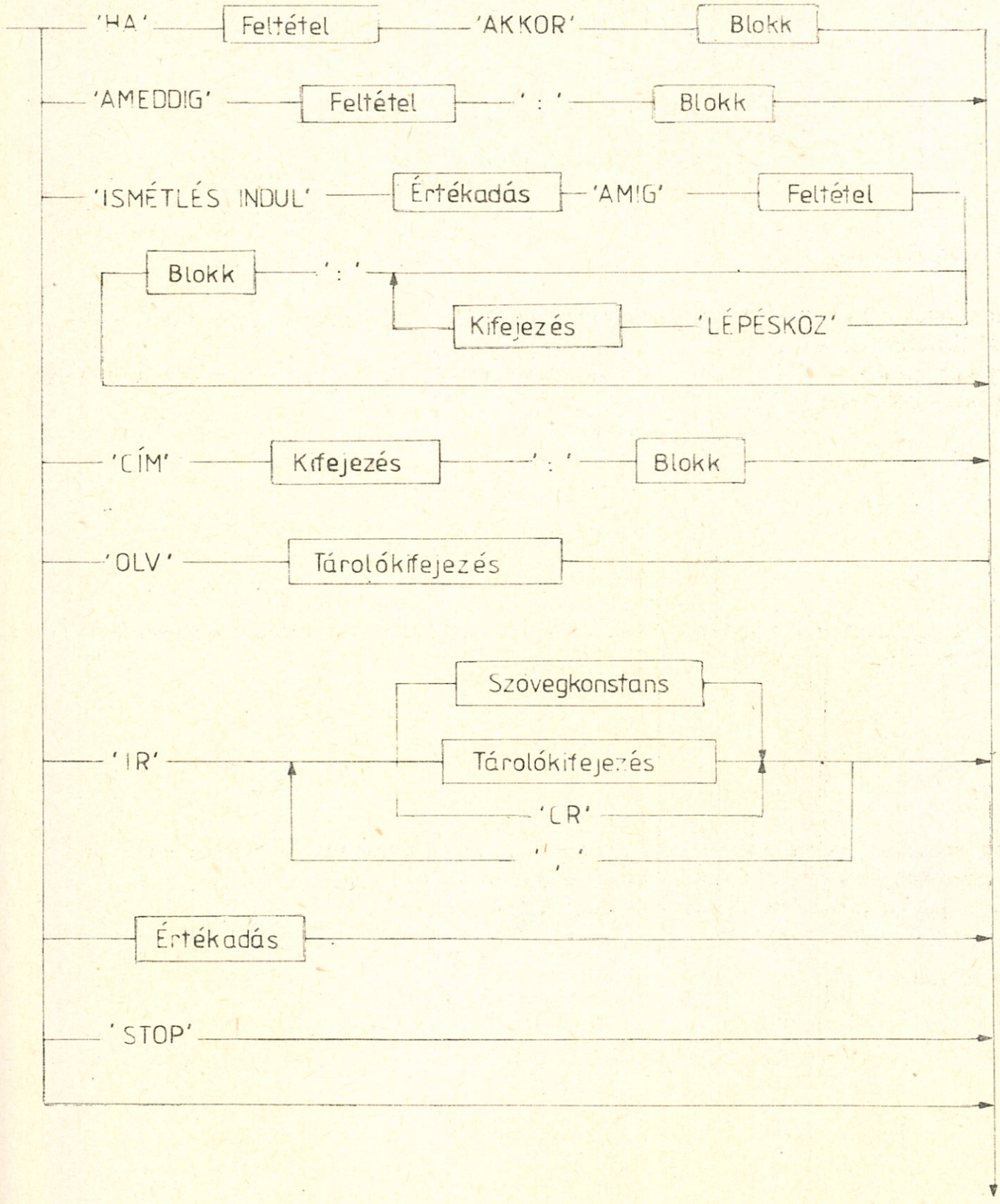
Program



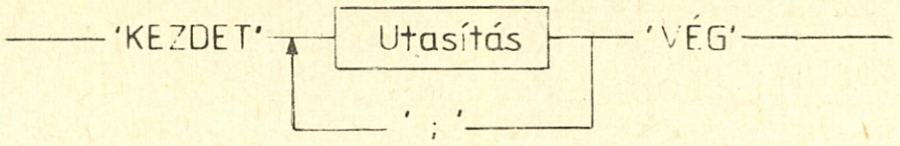
Törzs



Utasítás

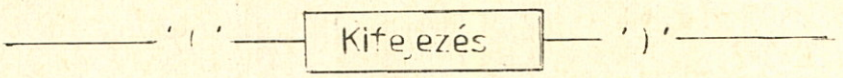

'HA'
Feltétel
'AKKOR'
Blokk
'AMEDDIG'
Feltétel
':'
Blokk
'ISMÉTLÉS INDUL'
Értékadás
'AMIG'
Feltétel
Blokk
':'
Kifejezés
'LÉPÉSKÖZ'
'CÍM'
Kifejezés
':'
Blokk
'OLV'
Tárolókifejezés
Szövegkonstans
'IR'
Tárolókifejezés
'CR'
','
Értékadás
'STOP'

Blokk:

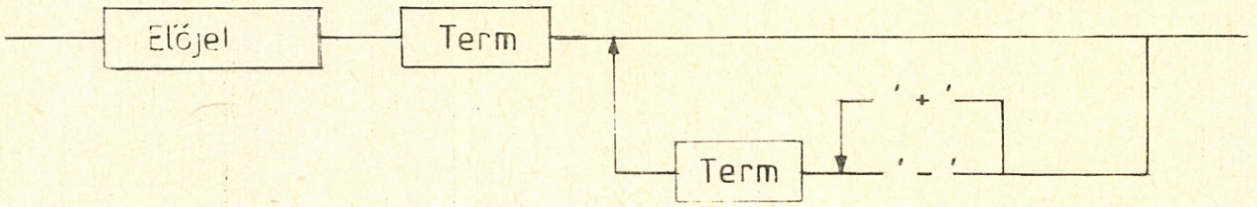


Értékadás

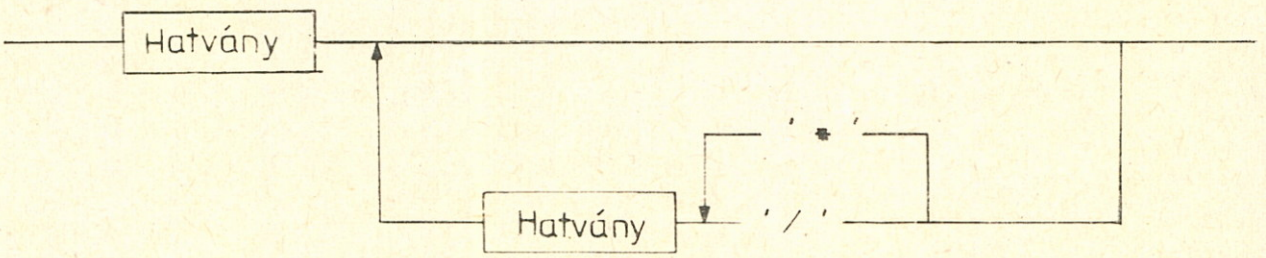
Tárolókifejezés



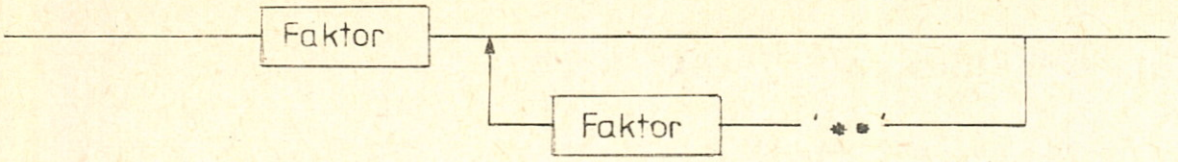
Kifejezés



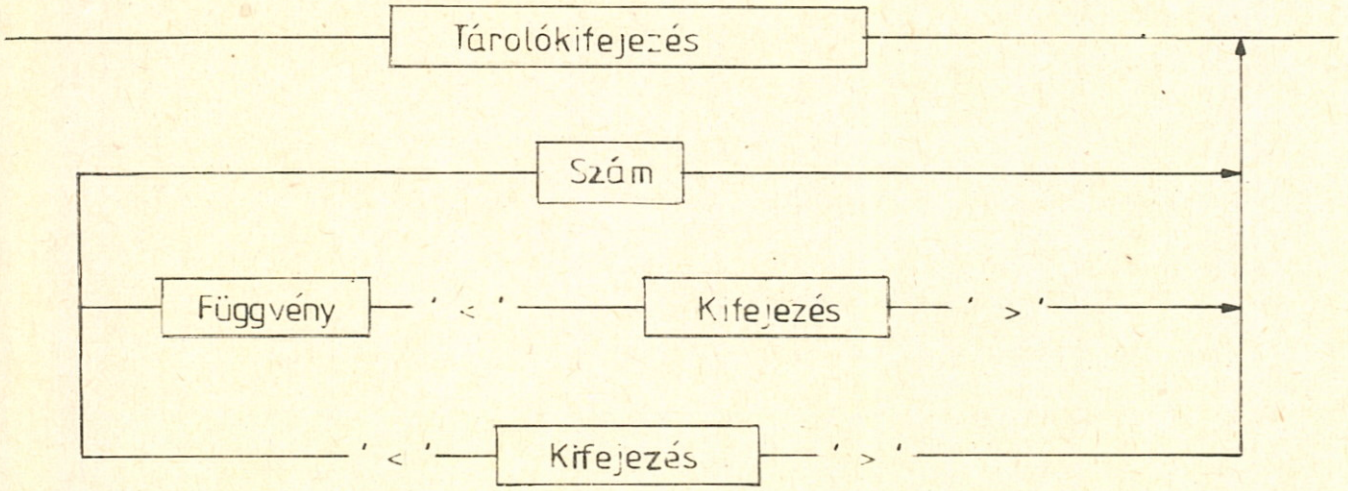
Term



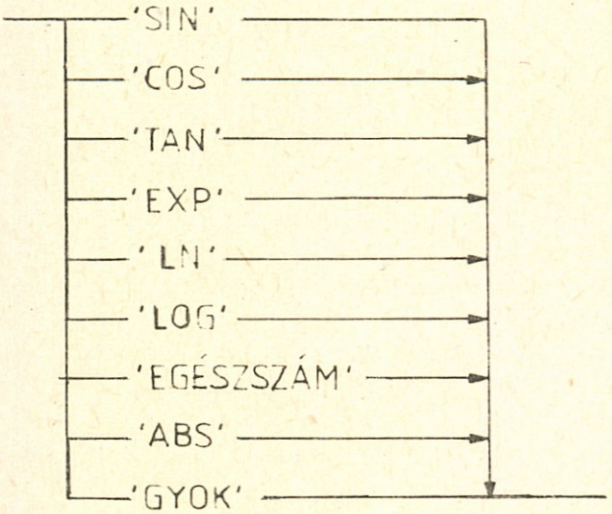
Hatvány



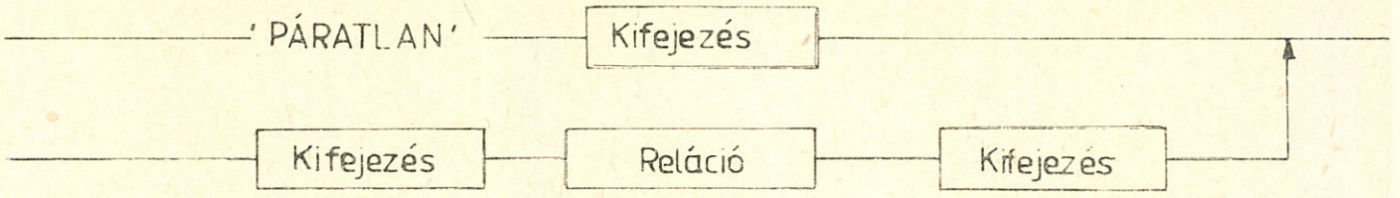
Faktor



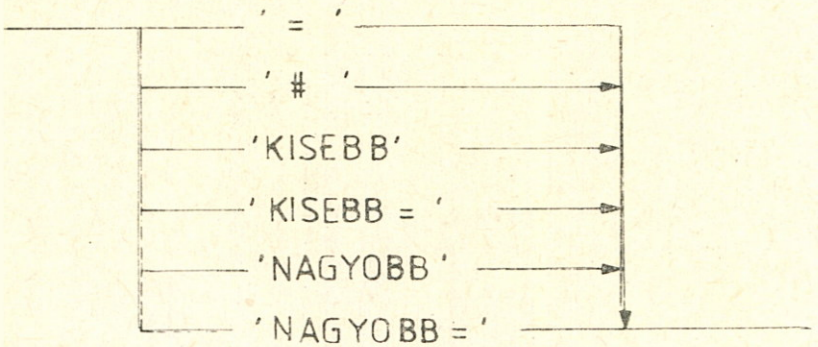
Függvény



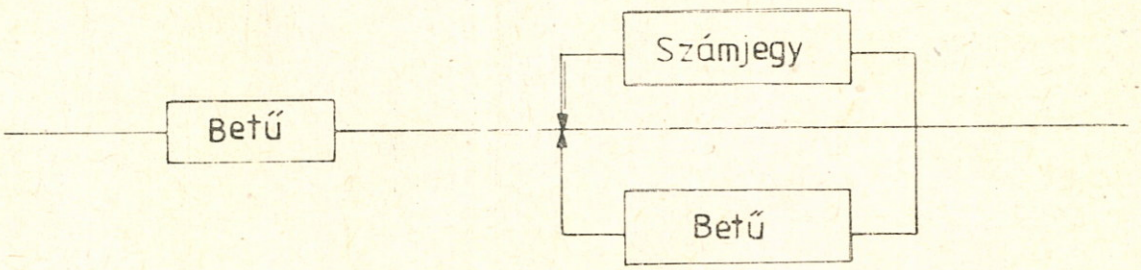
Feltétel:



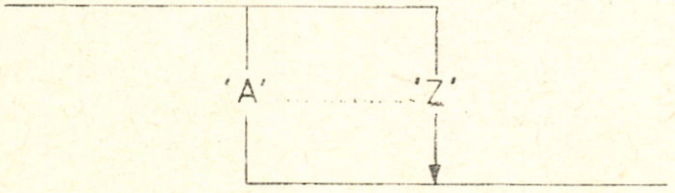
Reláció:



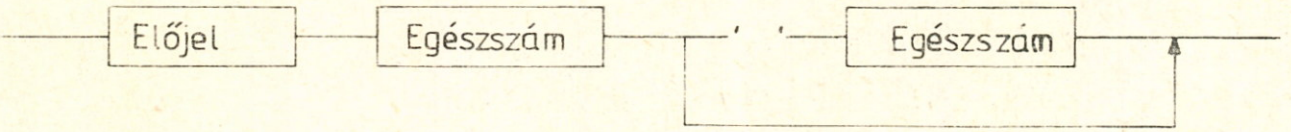
Név:



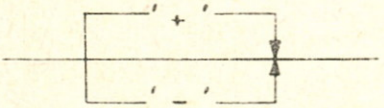
Betű:



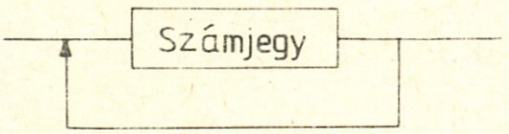
Szám:



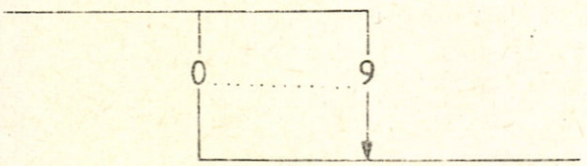
Előjel:



Egészszám:



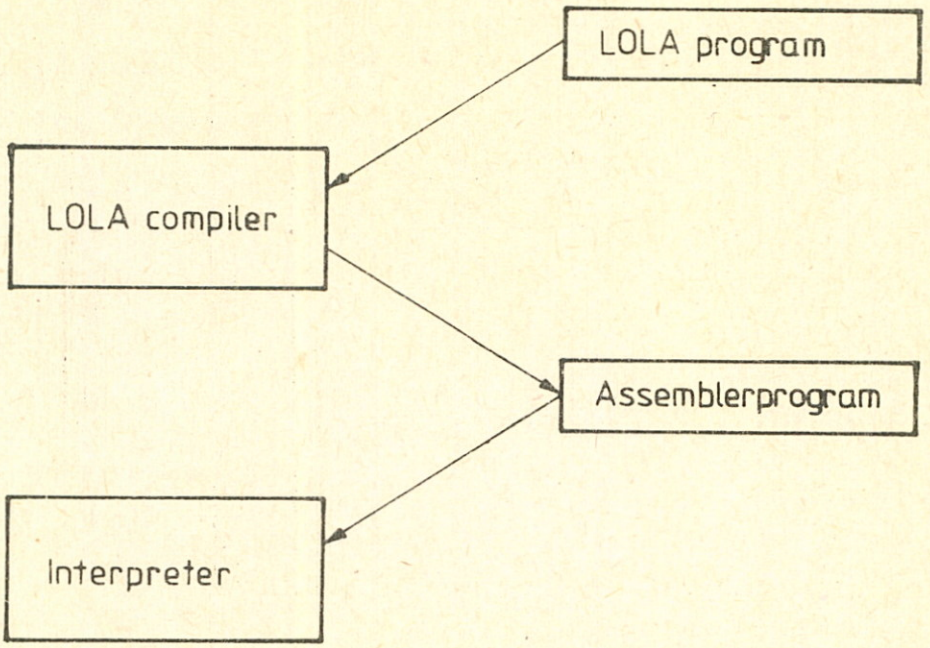
Számjegy:



2.2. AZ ABSZTRAKT GÉP

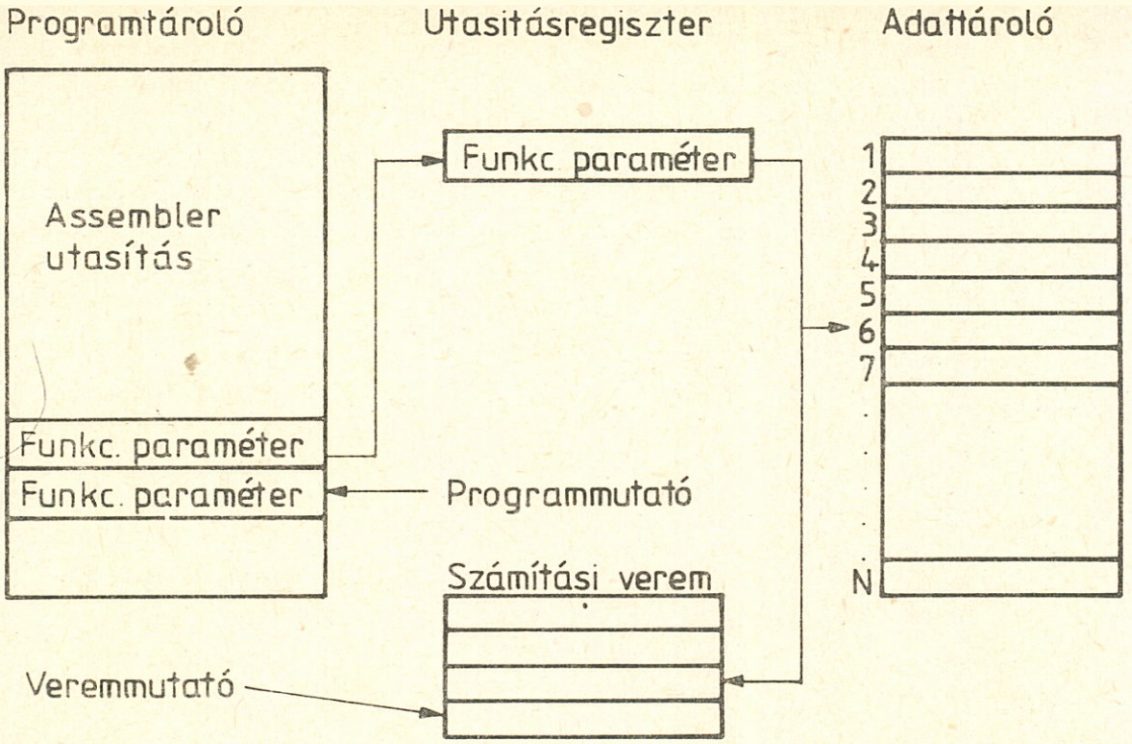
2.2.1. Alapvető meggondolások

A fordítók természetesen azzal a géppel vannak szoros kapcsolatban, amelyre megírták azokat. Ez azt jelenti, hogyha egy fordítót új gépen alkalmaznak, akkor a kódgenerálást újra el kell végezni.
Ezt a problémát úgy kerülhetjük el, hogy tervezünk egy feltételezett gépet és írunk egy programot (interpretert), amely ezt a gépet szimulálja.
Ennek az a hátránya, hogy a programfeldolgozás ideje megnő, viszont sok esetben az *interpreterrel* elért feldolgozási sebesség is megfelelő.
Az alábbi ábra a fordítási és programkivitelezési folyamatot mutatja:



2.2.2. Absztrakt gép tervezése

A következő ábra a gép számunkra érdekes elemeit mutatja be. A programtárolóban találhatók az assembler-utasítások, amelyeket végrehajtáskor az utasításregiszterbe tölt át. A programmutató mindig a következő végrehajtandó utasításra mutat.
A *LOLA* direkt hozzáférést valósít meg az adattárolóhoz, ezt mi az utasításregiszterrel címezhető tömbként realizáljuk. A számításokat a verem segítségével kell végeznünk. Ez a számítási verem az adattárolót is címezi, valamint az utasításregiszterből és az adattárolóból operandusokat fog a számítási verembe tölteni.
A karakterláncokat, amelyek *LOLA* programban csak az IR utasítás konstansaiként szerepelhetnek, lánctömbben tárolja.



UTASÍTÁSOK

Az assembler utasítások megkonstruálását természetesen a *LOLA* nyelv messzemenően befolyásolja. Végül is egy lehetőleg egyszerű leképzést kell a forráskódra megvalósítani. Az utasítások felosztása és megfogalmazása a fentiek ellenére erősen az egyéni ízlésre hagyatkozik. Így összevonhattuk volna az **OPR** és **FKN** utasítást vagy több utasításban alkalmazhattuk volna egynél több paramétert.

Paraméterezett utasítások:

Funkció	*Paraméter*	
CON	k	A real típusú *k* konstans betöltése a számítási verembe.

A következő paraméterek egész számok.

STA	a	A számítási verem legfelső elemének az adattároló *a* címére való tárolása.
JMP	a	Ugrás az *a* számú assembler utasításra.
JPC	a	Ugrás az *a* számú assembler utasításra, ha az előző feltétel teljesül.
JPN	a	Ugrás az *a* számú assembler utasításra, ha a közvetlen megelőző feltétel *nem* teljesül.
WRA	a	Az *a* címen levő lánc beírása a karakterlánc-tárolóba.
FKN	a	Az *a* függvény végrehajtása a számítási veremben. A függvények az aritmetikai kifejezésekben szereplő *LOLA* függvényeknek felelnek meg.

a = 0: ABS
1: COS
2: EXP
3: LOG
4: LN
5: SIN
6: TAN
7: GYÖK
8: EGÉSZ SZÁM

OPR a Az *a* műveletek végrehajtása a számítási veremben.
a = 0: Programstop
A következő műveletek az aritmetikai műveleteket realizálják és a legfelső és a közvetlen alatta fekvő elemre vonatkoznak.
1: negatív előjel
2: összeadás
3: kivonás
4: szorzás
5: osztás
6: hatványozás
Műveletek, amelyek valószínűségi értéket adnak.
Egyelemű:
7: páratlan
Kételemű:
8: egyenlő
9: nem egyenlő
10: kisebb
11: kisebb egyenlő
12: nagyobb
13: nagyobb egyenlő.

Paraméter nélküli utasítások:
Az alábbi utasítások a számítási verem legfelső elemeit címként használják, kiegészítve az esetleges ofszettel, amely a CIM *LOLA* utasítással állítható be.

LOD Az adattároló egy elemének betöltése a verembe.
STO A legfelső elem levétele címként és az alatta levő elem tárolása az adattárolóba.
INP *LOLA*-szám beolvasása a billentyűzetről és tárolása az adattárolóba.

Egyeb utasítások:
WRS A számítási verem legfelső elemének kiírása a képernyőre.
WRC Kocsi vissza és soremelés kiadása a képernyőre.

Ezzel az általunk szimulálni kívánt absztrakt gép szerkezeti felépítését és viselkedését rögzítettük. A hozzá tartozó interpreter megírása előtt megtervezzük a fordítót, amely a *LOLA*-t a fent leírt assembler utasításokra lefordítja.

2.3. FORDÍTÓ TERVEZÉSE A LOLA-HOZ

2.3.1. Alapszerkezet

A fordítóprogramunk szerkezete a feladatából adódik:
– a *LOLA* program elemzése és
– az assembler programok előállítása az absztrakt gépre.
Az előállítás az elemzéssel szoros kötöttségben van, ami azt jelenti, hogy amint az elemzés alapján elegendő információ áll rendelkezésre egy assembler programrészlet azonosítására, akkor azt azonnal generálja. Ennek ellenére az áttekinthetőség kedvéért először csak az elemzéssel foglalkozunk.

2.3.2. LOLA programok elemzése

2.3.2.1. Lexikális elemzés

Amikor egy szöveget olvasunk, mindegy, hogy könyvet vagy programot, a megértés első lépése az, hogy megállapítjuk minden egyes elem formáját és jelentését. Ezután képezünk az egyes betűkből egy magasabb szintet: a szót. A *LOLA* program fordításakor hasonlóan járunk el. Eljárást írunk, amely az egyik karaktert a másik után beolvassa és ezekből szimbólumot képez. A fordítóban ezt a programrészt *lexikális elemzésnek* nevezzük.
Ezt a folyamatot egy példán keresztül mutatjuk be.
LOLA programrészlet:

```
HA (1) = 1 AKKOR 2 -> (3)
```

A lexikális elemzéssel előállított szimbólumsor (a vesszőket csak a jobb áttekinthetőségért szúrtuk be) így néz ki:
hasym,bzjelsym,numsym,jzjelsym,egyenlősym,umsym,akkorsym,numsym,
eredmsym,bzjelsym,numsym,jzjelsym
Ezzel a következő pontban leírt szintaxiselemzést a mellékes dolgoktól megszabadítjuk és a fordítási folyamat gyorsabb lesz.
Összefoglalva a feladatokat:
– *LOLA* programszöveg beolvasása;
– A szöveg esetleges protokolljának elkészítése;
– Szimbólumképzés;
– Felismert szimbólumok értékének elkészítése.
Programpéldánkban a lexikális elemzést a Hozz_sym eljárással valósítottuk meg, amely minden meghívásakor a soron következő szimbólumot adja.
A Hozz_sym-et egy másik eljárás szolgálja ki, a Hozz_ch, amely mindig a soron következő karaktert adja.

2.3.2.2. Szintaktikus elemzés

Miután a programszöveg a lexikális elemzésen túl van, következik a nehezebb rész, a szintaktikus felépítés felismerése. Mit értsünk ezalatt?
Ha természetes nyelvről van szó, akkor az a felismerés, hogy egy mondat helyesen formált-e, többnyire intuitív tapasztalati alapon áll. Így aki érti a magyar nyelvet, az alábbi mondatot:

Szilvia, Dezső és Róbert könyvet ír.

érvényes magyar nyelvű mondatnak tartja, de a következőt:

Könyvet és Szilvia Dezső, Róbert ír.

már helytelennek, bár azonos szavak, szimbólumok fordulnak elő benne. A fordító szintaktikai elemzőjének hasonló feladata van. Azt, hogy érvényes programról van-e szó, csak a programnyelvben lefektetett szabályok meghatározása alapján dönthetjük el.
Esetünkben még emellett a definiált konstans értékeket táblázatba kell vinni úgy, hogy a kódgeneráláskor elérhetőek legyenek.
Bár ez a folyamat az első látásra talán bonyolultnak tűnik, az ehhez szükséges *Turbo–Pascal*-kód a 2.1.8. pontban definiált szintaxisdiagramból könnyen kivehető.

N. Wirth [13] közöl egy eljárást, amellyel a fordítás messzemenően automatizálható.

A figyelmünket egy olyan problémára kell összpontosítanunk, amellyel minden programozó – aki fordítóprogramot ír – kénytelen többé-kevésbé megküzdeni: a *szintaktikai* hibakezelésre.

A legegyszerűbb megoldás az elemzés félbeszakítása hiba esetén. Ennek az eljárásnak természetesen vannak hátrányai, amelyeket könnyen beláthatunk.

Jobb módszer egy hiba felfedezése után megkísérelni az elemzés folytatását. Ehhez néhány feltételezést kell tennünk. A találó feltételezések kiválasztása bonyolult és messzemenően tapasztalati értékeken nyugszik. Erősen függ a fordítandó nyelvtől is. Ennek ellenére van néhány általános szabály *(N. Wirth* után):

1. A hiba megállapítása után az elemző átugorja a szöveget mindaddig, amíg olyan szimbólumra nem akad, amelynek a hibás alkalmazása nagyon valószínűtlen. Ennél újból kezdődhet az analízis.
2. Minden meghívott eljárás megkísérli a hiba *javítását.* Ez azt jelenti, hogy át kell adnunk az eljárásnak egy olyan paramétert is, amely az érvényes követő szimbólumok összességét tartalmazza.
3. A paramétert a stop szimbólumok tömegével kibővítjük. Ezeknek az a feladata, hogy a kifejtett követő szimbólum (pl. pontosvessző) esetén a program továbbolvasását megakadályozza.
4. Teszt eljárást írunk, amely a továbbolvasásról gondoskodik.
5. Esetenként a teszt eljárást az eljárás elején is meghívhatjuk.

A szöveg továbbolvasásával végzett elemzésnél a fentiek ellenére is könnyen követhetünk el követési hibát. A hibás konstans deklaráció után a továbbolvasás arra vezethet például, hogy a meghatározott nevet később, mint nem kedlaráltat értelmezi.

2.3.2.3. Szemantikus elemzés

A lexikális és szintaktikus programhibák mellett még egy sor olyan hiba létezik, amelyek csak a program futtatásakor fedezhetők fel.
Például a következő programrészlet fordításakor nem kapunk hibajelzést, mégis a program végrehajtásakor nullával való osztás állhat elő:

```
{ Talan.lola }
{ L. II. 6.  }


PROGRAM Talan;
KONSTANS i=1;
KEZDET
  OLV (i);
  IR "EREDMENY:",12/(i)
VEG.
```

Bizonyos hibákat már a fordítási folyamat közben megállapíthatunk. Az azonban, hogy a vizsgálat milyen mértékű legyen, a programozó és a programnyelv ügye.
Például az alábbi változatban levő nullával való osztás már a fordítás közben felfedezhető:

```
{ Biztos.lola }
{ L. II. 7.   }


PROGRAM Biztos;
KONSTANS i=1;
KEZDET
  OLV (i);
  IR "EREDMENY:",12/<(i)-(i)>
VEG.
```

A szemantikai vizsgálat eredményeként a felhasználó számára az a legkedvezőbb, ha fordítás közben a lehető legtöbb hiba kiderül. Ez azonban gyakran a fordítóprogram-írás ráfordítási költségének kérdését is felveti. Példánkban az egyszerűség kedvéért nem alkalmaztunk kimerítő szemantikus elemzést.

```
{ ELEMZO.PAS   }
{ Pr. II. 2.1. }

PROGRAM Lola_Elemzo;

CONST
   c_kszo      =  28;  { Foglalt kulcsszavak szama      }
   c_ascii     = 128;  { ASCII-jelek szama              }

   c_szjmax    =   6;  { Szamjegyek maximalis szama     }
   c_azmax     =  1Ø;  { Azonosito maximalis hossza     }
   c_strmax    =  8Ø;  { Karakterlanc maximalis hossza  }
   c_errmax    =  3Ø;  { A hibasorszam maximuma         }
   c_adrmax    = 5Ø3;  { Az adatok maximalis cime + 3   }

   c_tabmax    = 1ØØ;  { A konstanstabla merete         }

TYPE
 t_symbol = (abssym,cossym,expsym,logsym,lnsym,sinsym,
             tansym, gyoksym,eszamsym,

      adrsym,kezdsym,befejezsym,akkorsym,vegesym,indulsym,
      konstsym,olvsym,progsym,irsym,
      lepessym,ameddigsym,stopsym,hasym,
      ismetlessym,

      aposym,azsym,crsym,osztsym,kettospontsym,eredmsym,
      egyenlosym,nagyobbsym,negyenlosym,uressym,
      bzarosym,bzjelsym,kisebbsym,kegyenlosym,vesszosym,
      szorsym,minussym,numsym,
      plussym,hatvsym,pontsym,jzjelsym,jzarosym,
      pontosvsym,ptlsym,nemegyenlosym);

   t_az       =  string [c_strmax];
   t_a        =  Ø..c_adrmax;
   t_symset   =  set of t_symbol;

VAR
   ch  : char;      { Utolso beolvasott karakter  }
   sym : t_symbol;  { Utolso beolvasott szimbolum }
   az  : t_az;      { Utolso beolvasott azonosito }
   num : real;      { Utolso beolvasott szam      }
```

```
    dec         : real;
    tab_ind,
    j           : integer;
    ssym        : array [char] of t_symbol;
    tabla       : array [Ø..c_tabmax] of
                  record
                       nev        : t_az;
                       konst      : real;
                  end;


    err,
    cc            : integer;
    prg_in      : text;
    prgnev      : string[12];
    sor          : string[8Ø];
    valasz      : char;
    sorsz       : integer;
    ok          : boolean;

CONST
    szo : array [1..c_kszo] of t_az =
    (
     'ABS',
     'AKKOR',
     'AMEDDIG',
     'BEFEJEZ',
     'CIM',
     'COS',
     'CR',
     'EGESZSZAM',
     'EXP',
     'GYOK',
     'HA',
     'INDUL',
     'IR',
     'ISMETLES',
     'KEZDET',
     'KISEBB',
     'KONSTANS',
     'LEPESKOZ',
     'LN',
     'LOG',
     'NAGYOBB',
     'OLV',
     'PARATLAN',
     'PROGRAM',
     'SIN',
     'STOP',
     'TAN',
     'VEG'
    );

wsym : array [1..c_kszo] of t_symbol =
(
 abssym,
 akkorsym,
 ameddigsym,
 befejezsym,
 adrsym,
 cossym,
 crsym,
 eszamsym,
 expsym,
 gyoksym,
 hasym,
```

```
      indulsym,
      irsym,
      ismetlessym,
      kezdsym,
      kisebbsym,
      konstsym,
      lepessym,
      lnsym,
      logsym,
      nagyobbsym,
      olvsym,
      ptlsym,
      progsym,
      sinsym,
      stopsym,
      tansym,
      vegesym
     );

    konstbegsys : t_symset = [konstsym];
    befbegsys   : t_symset = [kezdsym,hasym,ameddigsym,
                              irsym,olvsym,
                              ismetlessym,adrsym,stopsym,
                              bzjelsym];
    faktbegsys  : t_symset = [azsym,numsym,bzarosym,
                              sinsym,
                              tansym,bzjelsym,
                              expsym,logsym,lnsym,
                              gyoksym];

PROCEDURE Allj;

BEGIN
   Halt;
END; {.....................................................}


PROCEDURE Hiba (n : integer);

VAR
   f : string[5Ø];
   i : integer;

BEGIN
   case n of
     1: f:='A szam tul nagy';
     2: f:='A konstans tul nagy';
     3: f:='A cim pozitiv egeszszamnak kell lennie';
     4: f:='Itt "=" jelnek kell lennie "->" helyett';
     5: f:='Szamot var';
     6: f:='Itt "=" jelnek kell lennie';
     7: f:='Nevet var';
     8: f:='Ervenytelen faktor';
     9: f:='Ismeretlen nev';
    1Ø: f:='">" -t var';
    11: f:='"<" -t var';
    12: f:='")" -t var';
    13: f:='"(" -t var';
    14: f:='Nincs relacio';
    15: f:='A tarolo megadasat varja';
    16: f:='";" -t var';
    17: f:='"VEG" -t var';
    18: f:='Itt "->" -nek kell lennie';
    19: f:='"AKKOR" -t var';
    2Ø: f:='Ervenytelen feltetel';
    21: f:='"PROGRAM" -t var';
    22: f:='A "." hianyzik';
```

```
      23: f:='A nev tul hosszu';
      24: f:='Itt "INDUL" -nak kell lennie';
      25: f:='Itt "BEFEJEZ"-nek kell lennie';
      26: f:='":" -t var';
      27: f:='";" nem megengedett';
    end;
    if valasz = 'I'
     then begin
       for i:=1 to cc-1 do Write('_');
       Writeln('^ ',f);
     end;
    err:=err+1;
    if err > c_errmax
     then begin
       Close(prg_in);
       Allj;
     end;
END; {.....................................................}


PROCEDURE Hozz_sym;

VAR
   i,j,k : integer;

   PROCEDURE Hozz_ch;

   BEGIN
      if cc = Length(sor)
       then begin
         if Eof(prg_in)
          then begin
            Writeln('Nem vart program vege');
            Allj;
          end;
          Readln(prg_in,sor);
          sorsz:=sorsz+1;
          if valasz = 'I'
           then Writeln(sor)
           else begin
             Gotoxy(18,5);
             Write(sorsz:5);
           end;
          sor:=sor+' ';
          cc:=Ø;
       end;
      cc:=cc+1;
      ch:=sor[cc];
   END; {.....................................................}

BEGIN
   while ch = ' ' do Hozz_ch;
   if ch in ['a'..'z','A'..'Z']
    then begin
      az:='';
      repeat
        az:=az+Upcase(ch);
       Hozz_ch;
      until not (Upcase(ch) in
            ['a'..'z','A'..'Z','Ø'..'9']);
      if Length(az) > c_azmax then Hiba(21);
      az:=copy(az,1,c_azmax);
      i:=1;
      j:=c_kszo;
      repeat
```

```
      k:=(i+j) div 2;
      if az <= szo[k] then j:=k-1;
      if az >= szo[k] then i:=k+1;
    until i > j;
    if i-1 > j
     then sym:=wsym[k]
     else sym:=azsym;
  end
  else
   if ch in ['Ø'..'9']
    then begin
      j:=Ø;
      num:=Ø;
      sym:=numsym;
      while ch in ['Ø'..'9']
       do begin
         num:=1Ø*num+(ord(ch)-ord('Ø'));
         j:=j+1;
         Hozz_ch;
       end;
      if j > c_szjmax then Hiba(1);
      if ch = '.'
       then begin
         Hozz_ch;
         j:=Ø;
         dec:=1;
         while ch in ['Ø'..'9']
          do begin
            j:=j+1;
            dec:=dec*Ø.1;
            num:=num+(ord(ch)-ord('Ø'))*dec;
            Hozz_ch;
          end;
         if j > c_szjmax then Hiba(1);
       end;
    end
    else
     if ch = '-'
     then begin
       Hozz_ch;
       if ch = '>'
        then begin
          sym:=eredmsym;
          Hozz_ch;
        end
        else sym:=minussym;
     end
     else
      if ch = '"'
       then begin
         Hozz_ch;
         while ch <> '"' do Hozz_ch;
         sym:=aposym;
         Hozz_ch;
       end
       else

       if ch = '*'
        then begin
          Hozz_ch;
          if ch = '*'
           then begin
             sym:=hatvsym;
             Hozz_ch;
           end
           else sym:=szorsym;
        end
```

```
            else begin
              sym:=ssym[ch];
              Hozz_ch;
            end;
    if (sym = nagyobbsym) and (ch = '=')
     then begin
       sym:=negyenlosym;
       Hozz_ch;
     end;
    if (sym = kisebbsym) and (ch = '=')
     then begin
       sym:=kegyenlosym;
       Hozz_ch;
     end;
END; {.....................................................}

PROCEDURE Teszt (s1,
                 s2 : t_symset;
                 n  : integer);

BEGIN
    if not (sym in s1)
     then begin
       Hiba(n);
       s1:=s1+s2;
       while not (sym in s1) do Hozz_sym;
     end;
END; {.....................................................}


PROCEDURE Torzs (fsys : t_symset);

    PROCEDURE Bevitel;

    BEGIN
       tab_ind:=tab_ind+1;
       with tabla[tab_ind]
        do begin
          nev:=az;
          if num > c_adrmax
           then begin
             Hiba(2);
             num:=Ø;
           end;
          konst:=num;
       end;
    END; {.....................................................}

    FUNCTION Pozicio (az : t_az) : integer;

    VAR
       i : integer;

    BEGIN
      tabla[Ø].nev:=az;
      i:=tab_ind;
      while tabla[i].nev <> az do i:=i-1;
      Pozicio:=i;
    END; {.....................................................}

    PROCEDURE Konstansdef;

    BEGIN
       if sym = azsym
        then begin
          Hozz_sym;
          if sym in [egyenlosym,eredmsym]
```

```
          then begin
            if sym = eredmsym then Hiba(4);
            Hozz_sym;
            if sym = numsym
             then begin
               Bevitel;
               Hozz_sym;
             end
             else Hiba(5);
          end
          else Hiba(6);
      end
      else Hiba(7);
END; {.....................................}



PROCEDURE Utasitas;

VAR
   i : integer;

   PROCEDURE Kifejezes (fsys : t_symset);

      PROCEDURE Term (fsys : t_symset);

         PROCEDURE Hatvany (fsys : t_symset);

            PROCEDURE Faktor (fsys : t_symset);

            VAR
               i : integer;

            BEGIN
               Teszt(faktbegsys,fsys+[eredmsym,pontosvsym],8);
               while sym in faktbegsys
                do begin
                   if sym = bzjelsym
                    then begin
                      Hozz_sym;
                      Kifejezes([jzjelsym]+fsys);
                      if sym = jzjelsym
                       then Hozz_sym
                       else Hiba(12);
                    end
                    else
                     if sym = azsym
                      then begin
                        i:=Pozicio(az);
                        if i=Ø then Hiba(9);
                        Hozz_sym;
                      end
                      else
                       if sym = numsym
                        then Hozz_sym
                        else
                         if sym = bzarosym
                          then begin
                            Hozz_sym;
                            Kifejezes([jzarosym]+   s);
                            if sym = jzarosym
                             then Hozz_sym
                             else Hiba (1Ø);
                          end
```

```
                                else
                                 if sym in [abssym,cossym,expsym,
                                            lnsym,eszamsym,sinsym,
                                            gyoksym,logsym,
                                            tansym]
                                  then begin
                                    Hozz_sym;
                                    if sym = bzarosym
                                     then begin
                                       Hozz_sym;
                                       Kifejezes([jzarosym]+fsys);
                                       if sym = jzarosym
                                        then Hozz_sym
                                        else Hiba(10);
                                     end
                                     else Teszt([],fsys,8);
                                  end;
                    end;
        END; {.....................................}

            BEGIN
               Faktor(fsys+[szorsym,osztsym,hatvsym]);
               while sym = hatvsym
                do begin
                  Hozz_sym;
                  Faktor(fsys+[szorsym,osztsym,hatvsym]);
                end;
    END; {.....................................}

         BEGIN
            Hatvany(fsys+[szorsym,osztsym]);
            while sym in [szorsym,osztsym]
             do begin
               Hozz_sym;
               Hatvany(fsys+[szorsym,osztsym]);
            end;
END; {.....................................}


BEGIN
   if sym in [plussym,minussym]
    then begin
      Hozz_sym;
      Term(fsys+[plussym,minussym]);
    end
    else Term(fsys+[plussym,minussym]);
   while sym in [plussym,minussym]
    do begin
      Hozz_sym;
      Term(fsys+[plussym,minussym]);
    end;
END; {.....................................}

PROCEDURE Feltetel (fsys : t_symset);

VAR relop : t_symbol;

BEGIN
  if sym = ptlsym
        then begin
             Hozz_sym;
             Kifejezes(fsys);
        end
        else begin
             Kifejezes([nemegyenlosym,egyenlosym,
             kisebbsym,nagyobbsym,kegyenlosym,
             negyenlosym]+fsys);
```

```
                if not (sym in [egyenlosym,
                     nemegyenlosym,
                     kisebbsym,nagyobbsym,
                     kegyenlosym,negyenlosym])
                   then Hiba(14)
                   else begin
                        relop:=sym;
                        Hozz_sym;
                        Kifejezes(fsys);
                        e.d;
                end;
END; {.....................................}

PROCEDURE IrUtasitas;

BEGIN
   if sym = aposym
    then Hozz_sym
    else if sym = crsym
          then Hozz_sym
          else Kifejezes(fsys);
END; {.....................................}


  PROCEDURE OlvUtasitas;
    *
  BEGIN
     if sym = bzjelsym
      then begin
        Hozz_sym;
        Kifejezes([jzjelsym]+fsys);
        if sym = jzjelsym
         then Hozz_sym
        else Hiba(12);
      end
      else Hiba(15)
  END; {.....................................}

BEGIN
   if sym = hasym
    then begin
      Hozz_sym;
      Feltetel([akkorsym]+fsys);
      if sym = akkorsym
       then Hozz_sym
       else Hiba(19);
      Utasitas;
    end
    else
     if sym = ameddigsym
      then begin
        Hozz_sym;
        Feltetel([kettospontsym]+fsys);
        if sym = kettospontsym
         then Hozz_sym
         else Hiba(26);
        Utasitas;
      end
      else
       if sym = irsym
        then begin
          Hozz_sym;
          IrUtasitas;
          while sym = vesszosym
           do begin
             Hozz_sym;
             IrUtasitas;
           end;
          end
```

```
else
 if sym = olvsym
  then begin
    Hozz_sym;
    OlvUtasitas;
    while sym = vesszosym
     do begin
       Hozz_sym;
       OlvUtasitas;
     end;
  end
  else
    if sym = kezdsym
     then begin
       Hozz_sym;
       Utasitas;
       while sym in [pontosvsym]+befbegsys
        do begin
           if sym = pontosvsym
            then Hozz_sym
            else Hiba(16);
          Utasitas;
        end;
       if sym = vegesym
        then Hozz_sym
        else Hiba(17);
     end
     else
      if sym = ismetlessym
       then begin
         Hozz_sym;
         if sym = indulsym
          then Hozz_sym
          else Hiba(24);
         Kifejezes(fsys);
         if sym = eredmsym
          then Hozz_sym
          else Hiba(18);
         if sym = bzjelsym
          then Hozz_sym
          else Hiba(12);
         Kifejezes([jzjelsym]+fsys);
         if sym = jzjelsym
          then Hozz_sym
          else Hiba(13);
         if sym = befejezsym
          then Hozz_sym
          else Hiba(25);
         Feltetel(fsys);
        if sym = lepessym
         then begin
           Hozz_sym;
           Kifejezes(fsys);
         end;
        if sym = kettospontsym
         then Hozz_sym
         else Hiba(26);
        Utasitas;
      end
      else
       if sym = adrsym
        then begin
          Hozz_sym;
          Kifejezes(fsys);
          if sym = kettospontsym
           then Hozz_sym
```

```
                        else Hiba(26);
                      Utasitas;
                    end
                    else
                     if sym = stopsym
                      then Hozz_sym
                      else if sym in faktbegsys
                             then begin
                               Kifejezes(fsys);
                               if sym = eredmsym
                                then Hozz_sym
                                else Hiba(18);
                               if sym = bzjelsym
                                then Hozz_sym
                                else Hiba(13);
                               Kifejezes([jzjelsym]+fsys);
                               if sym = jzjelsym
                                then Hozz_sym
                                else Hiba(12);
                             end;

END; {.....................................................}


BEGIN
   repeat
     if sym = konstsym
      then begin
        Hozz_sym;
        repeat
          Konstansdef;
          while sym = vesszosym
           do begin
             Hozz_sym;
             Konstansdef;
           end;
          if sym = pontosvsym
           then Hozz_sym
           else Hiba(16);
        until sym <> azsym;
      end;
   until not (sym = konstsym);
   Utasitas;
   Teszt(fsys,[],22);
END; {.....................................................}

PROCEDURE Hozz_valasz;

BEGIN
   repeat
     read(valasz);
     valasz:=Upcase(valasz);
   until valasz in ['N','I'];
END; {.....................................................}


BEGIN
   ClrScr;
   Lowvideo;
   Write('KEREM A PROGRAM NEVET (Forras):    ');
   repeat
     Highvideo;
     Read(prgnev);
     Lowvideo;
     Assign(prg_in,prgnev);
     {$I-}
```

```
    Reset(prg_in);
    {$I+}
    ok:=IOresult = Ø;
    if not ok
     then begin
        GotoXY(1,3);
        write('A PROGRAM NEM TALALHATO! VEGE(I/N)? ');
        Highvideo;
        Hozz_valasz;
        if valasz = 'I' then Allj;
        GotoXY(1,3);
        ClrEol;
        GotoXY(35,1);
        ClrEol;
     end;
  until ok;
  GotoXY(1,3);
  Write('PROGRAM LISTAZAS (I/N) ?  ');
  Highvideo;
  Hozz_valasz;
  if valasz = 'N'
   then begin
     GotoXY(1,5);
     Lowvideo;
     Writeln('ELEMZETT SOROK: ');
     Highvideo;
   end
   else begin
     Writeln;
     Writeln;
   end;

  sorsz:=Ø;
  err:= Ø;
  sor:= '';
  cc:= Ø;
  tab_ind:= Ø;
  for ch:=chr(Ø) to chr(c_ascii-1) do
                  ssym[ch]:=uressym;
  ssym['+']:=plussym;
  ssym['-']:=minussym;
  ssym['*']:=szorsym;
  ssym['/']:=osztsym;
  ssym['(']:=bzjelsym;
  ssym[')']:=jzjelsym;
  ssym['=']:=egyenlosym;
  ssym[',']:=vesszosym;
  ssym['.']:=pontsym;
  ssym['#']:=nemegyenlosym;
  ssym['<']:=bzarosym;
  ssym['>']:=jzarosym;
  ssym[';']:=pontosvsym;
  ssym['"']:=aposym;
  ssym[':']:=kettospontsym;
  ch:=' ';
  Hozz_sym;
  Teszt([progsym],konstbegsys+befbegsys,21);
  if sym = progsym then Hozz_sym;
  if sym = azsym
   then Hozz_sym
   else Hiba(7);
  if sym = pontosvsym then Hozz_sym else Hiba(16);
  Torzs([pontsym]+konstbegsys+befbegsys);
  if sym <> pontsym then Hiba(22);
     Writeln;
     Writeln;
     Writeln('***',err:4,' *** HIBA LEPETT FEL');
     Close(prg_in);
  END.
```

A *LOLA* programot a szintaktikus elemzés automatikusan részegységekre bontja. Ezt a tulajdonságot az assemblerkódok előállításakor kihasználhatjuk. Amint egy szintaktikus blokk (pl. egy AMEDDIG ciklus) lezárul, a hozzá tartozó kód generálása is befejeződik.
Itt azonban problémába ütközünk.
Néha olyan assembler utasításokat kell képeznünk, amelyekhez még nem áll rendelkezésünkre az összes szükséges információ. Ezt a HA utasítással kapcsolatos példán magyarázzuk meg:
LOLA-példa:

```
{ Ha.lola    }
{ L. II. 8. }

   HA (1) = Ø AKKOR
      KEZDET
         (3) -> (1);
         OLV (3)
      VEG;
```

Assemblerkód:

```
 0 CON    1
 1 LOD
 2 CON    0
 3 OPR    8
 4 JPN   11   <——  A 'JPN' utasítás címe először
 5 CON    3
 6 LOD
 7 CON    1
 8 STO
 9 CON    3
10 INP        <——  itt lesz ismert
11 OPR    0
```

Elháríthatjuk a kellemetlen helyzetet azzal, hogy a kódot nem adjuk ki azonnal, hanem közben egy tömbben tároljuk, így először az ugrási címet nyitva hagyjuk (a programban 0-ra generálva) és megjegyezzük az utasítás indexét. Miután az ugrási cím ismert, a megjegyzett címmel visszanyúlunk a kódtárolóba és az ugró utasítást kijavítjuk.
Ezt az utasításfoltozást a szakma *Fixup*-nak nevezi. További különlegességet mutat az aritmetikai kifejezések feldolgozása. Ezeket, a *LOLA* definíció szerint, úgynevezett Infix kifejezésként adjuk meg. Ez azt jelenti, hogy a műveleti jel két operandus között áll.

Példa: (i) + 123/3*⟨45 + (k)⟩

Zárójelekkel megváltoztathatjuk a műveletek végrehajtási sorrendjét. Mivel a célgépünk azonban csak korlátos mennyiségű tárolórekesszel rendelkezik a számolás céljára, ezeket a kifejezéseket számításokra alkalmasabb formára kell hozni. *Lukasiewitz* lengyel matematikus feltalált egy lehetőséget a zárójelek teljes elhagyására. Az aritmetikai kifejezések ilyen előállítási módját ezért gyakran *lengyel logikának* vagy *Postfix* ábrázolásnak nevezzük, ahol a műveleti jel a két operandus mögött található.
Az elv magyarázataképpen tekintsük az alábbi példát:

LOLA-kifejezés Infix formában:
1+23*⟨2+9/3⟩

LOLA-kifejezés Postfix formában:
1 23 + 2 9 3 / + *
A Postfix képlet számítása balról jobbra:
Kezdet 1 23 + 2 9 3 / + *
1. lépés 24 2 9 3 / + *
2. lépés 24 2 3 + *
3. lépés 24 5 *
4. lépés 120

Láthatjuk, hogy a Postfix kifejezésnél megtakarítottuk a zárójelezést. Ennek alapján az Infix kifejezések elemzésekor a Postfix kód generálását kell beépítenünk. Ez egyszerű, mivel csak a műveleti jel kiadásával kell késlekednünk. Azaz először megadjuk az operandusokat, majd aztán a műveleti jelet.

Mielőtt a fordító komplett listáját közölnénk, tárgyaljuk meg a 2.1.7. pont összegprogramja kapcsán az assemblerkód előállítását.
Előbb azonban meg kell világítanunk egy problémát az ISMETLES és a CIM *LOLA*-utasítással összefüggésben:

Hogyan lehet felismerni az assemblerkódok feldolgozásánál a ciklusszámláló címét, ill. a lépésköz nagyságát?
Ezenkívül a CIM utasítás számára egy esetleges ofszettel kell megjelölnünk, ami azt jelenti, hogy szükségünk van további három tárolóhelyre.
Mivel gépünknek nincs speciális regisztere erre a célra, az adattároló legfelső címeit használjuk megoldásként.
Példánkban az adattároló nagysága 503 elemet tesz ki, az utolsó hármat a következőképpen használjuk:
Cím 501 tartalmazza a ciklusszámláló címét,
Cím 502 tartalmazza a lépésnagyságot,
Cím 503 tartalmazza a CIM utasítással beállított ofszetértéket (a standard érték: 0),
Ezután az előkészítés után nézzük a *LOLA*-utasításokat és a hozzá tartozó assembler utasításokat:

```
PROGRAM Összeg;
KONSTANS  összeg     = 0,
          végérték   = 1,
          i          = 2;

KEZDET
    IR „KÉREM A VÉGÉRTÉKET MEGADNI”;
    0  WRA        0     / írj a lánctároló 0 indexével karakterláncot OLV (végérték);
    1  CON  1.00000     / töltsd be az 1 konstanst
    2  INP        0     / olvass a billentyűzetről az adattároló 1 címére.

HA (végérték) KISEBB 0 AKKOR
    3  CON  1.00000     / töltsd be az 1 konstanst
    4  LOD        0     / töltsd be az adattároló 1 címét
    5  CON  0.00000     / töltsd be a 0 konstanst
    6  OPR       10     / kisebb?
    7  JPN       10     / ugorj 10-re, ha nem igaz
```

```
KEZDET
    IR "sajnos rossz";
    8  WRA          1 / írj a lánctartó 1 indexével karakterláncot
    STOP
    9  OPR          0 / Programstop
    VEG;
    0  -> (összeg);
    10    CON    0.00000 / töltsd be a 0 konstanst
    11    CON    0.00000 / töltsd be a 0 konstanst = az "összeg" címe
    12    STOP           / tárold a 0-t az adattár 0 címére

ISMÉTLÉS INDUL 1 -> i BEFEJEZ i NAGYOBB végérték:
    13    CON    1.00000 / töltsd be az 1 konstanst
    14    CON    2.00000 / töltsd be a 2 konstanst = az "i" címe
    15    STA        501 / tárold a 2-t az adattár 501-es címére
    16    CON  501.00000 / töltsd be az 501-es konstanst = "i" címének a címe
    17    LOD           / töltsd be az adattár elemét az 501-es címről
    18    STO           / tárold az 1-et az adattár e címére
    19    CON    2.00000 / töltsd be a 2 konstanst = az "i" címe
    20    LOD           / töltsd be az adattár elemét a 2-es címről
    21    CON    1.00000 / töltsd be az 1 konstanst = "végérték" címe
    22    LOD           / töltsd be az adattár elemét 1 címről
    23    OPR         12 / nagyobb?
    24    JPC         45 / ugorj a 45. utasításra, ha igaz
    25    CON    1.00000 / töltsd be az 1 konstanst
    26    CON  502.00000 / töltsd be az 502-es konstanst = lépésköz címe
    27    STO           / tárold az 1-et erre a címre (Default érték)
    összeg + i -> összeg;
    28    CON    0.00000 / töltsd be a 0-t = az "összeg" címe
    29    LOD           / töltsd be az adattár elemét 0 címről
    30    CON    2.00000 / töltsd be a 2 konstanst = az "i" címe
    31    LOD           / töltsd be az adattár elemét; 2 címmel
    32    OPR         2 / összeadás
    33    CON         0 / töltsd be a 0 konstanst = az "összeg" címe
    34    STO           / tárold az összeadás eredményét a 0 címre
    35    CON  501.00000 / töltsd be az 501 konstanst = az "i" címének a címe
    36    LOD           / töltsd be az adattár elemét az 501-es címről
    37    LOD           / töltsd be az adattár elemét = az "i"-t
    38    CON  502.00000 / töltsd be az 502-es konstanst = a lépésköz címe
    39    LOD           / töltsd be az adattár elemét az 502-es címről
    40    OPR         2 / összeadás
    41    CON  501.00000 / töltsd be az 501-es konstanst = az "i" címének a címe
    42    LOD           / töltsd be az adattár elemét az 501-es címről
    43    STO           / tárold az összeadás eredményét az "i"-ben
    44    JMP        19 / ugorj az összehasonlításhoz


IR "Összeg:", (összeg)
    45    WRA          2 / írj a lánctároló 2-es indexével karakterláncot
    46    CON          0 / töltsd be a 0 konstanst = az "összeg" címe
    47    LOD            / töltsd be az adattár elemét a "0" címmel
```

```
48   WRS      0   / írd ki a legfelső stack elemet = "összeg"
49   OPR      0   / programstop
```

A következő oldalakon megtalálható a fordító teljes listája.

```
{ FORDITO.PAS  }
{ Pr. II. 2.2. }

PROGRAM Lola_Fordito;
{$U-,C+}
CONST
   c_kszo      =  28;  { Foglalt kulcsszavak szama      }
   c_ascii     = 128;  { ASCII-jelek szama              }

   c_szjmax    =   6;  { Szamjegyek maximalis szam      }
   c_azmax     =  10;  { Azonosito maximalis hossza     }
   c_strmax    =  80;  { Karakterlanc maximalis hossza  }
   c_errmax    =  30;  { A hibasorszam maximuma         }
   c_adrmax    = 503;  { Az adatok maximalis cime + 3   }

   c_codemax   = 500;  { A kod-stack merete             }
   c_litmax    =  50;  { A string-stack merete          }
   c_tabmax    = 100;  { A konstanstabla merete         }

TYPE
   t_symbol = (abssym,cossym,expsym,logsym,lnsym,
               sinsym,tansym, gyoksym,eszamsym,

       adrsym,kezdsym,befejezsym,akkorsym,vegesym,
       indulsym,konstsym,olvsym,progsym,irsym,
       lepessym,ameddigsym,stopsym,hasym,
       ismetlessym,

       aposym,azsym.crsym,osztsym,kettospontsym,
       eredmsym,egyenlosym,nagyobbsym,negyenlosym,
       uressym,bzarosym,bzjelsym,kisebbsym,
       kegyenlosym,vesszosym,szorsym,minussym,numsym,
       plussym,hatvsym,pontsym,jzjelsym,jzarosym,
       pontosvsym,ptlsym,nemegyenlosym);

   t_az        =  string [c_strmax];
   t_a         =  0..c_adrmax;
   t_symset    =  set of t_symbol;
   t_fkt       =  (con,opr,lod,sto,sta,jmp,jpc,jpn,wrs,
                   wra,wrc,fun,inp);
   t_literal   =  string[80];

   t_instruktion = record
                     case fkt : t_fkt of
                          con : (k : real);
                          opr,lod,sto,sta,jmp,jpc,jpn,
                          wrs,wra,wrc,
                          fun,inp : (a : t_a);
                     end;

   t_code      = array [0..c_codemax] of t_instruktion;

   t_literale = array [0..c_litmax] of t_literal;

   t_pgm       = record
                   code     : t_code;
                   literale : t_literale;
                   end;

   { CON k      "k" konstans betoltese
```

```
      OPR a      "a" muvelet vegrehajtasa
      LOD        valtozo tartalmanak betoltese
      STO        ertek tarolasa valtozoban
      STA a      verem-kifejezes tarolasa az "a" -ba
      JMP a      feltetel nelkuli ugras "a" -ra
      JPC a      ugras "a"-ra, ha a feltetel igaz
      JPN a      ugras "a"-ra, ha a feltetel hamis
      WRS        veremtartalom kiirasa a kepernyore
      WRA a      "a" beirasa a karakterlanctombbe
      WRC        "carriage return" es "line feed" iras
      FUN a      "a" fuggveny vegrehajtasa
      INP        olvasas a veremben levo cimre  }

VAR
   ch  : char;       { Utolso beolvasott karakter  }
   sym : t_symbol;   { Utolso beolvasott szimbolum }
   az  : t_az;       { Utolso beolvasott azonosito }
   num : real;       { Utolso beolvasott szam      }

   dec        : real;
   pgm        : t_pgm;
   lcode      : char;
   code       : t_code;
   zw_literal: t_literal;
   literale   : t_literale;
   code_ind,
   lit_ind,
   tab_ind,
   j          : integer;
   ssym       : array [char] of t_symbol;
   tabla      : array [Ø..c_tabmax] of record
                                         nev   : t_az;
                                         konst : real;
                                         end;
   err,
   cc         : integer;
   prg_in     : text;
   prg_out    : file of t_pgm;
   prgnev     : string[12];
   sor        : string[8Ø];
   valasz     : char;
   sorsz      : integer;
   ok         : boolean;

CONST
   szo : array [1..c_kszo] of t_az =
  (
   'ABS',
   'AKKOR',
   'AMEDDIG',
   'BEFEJEZ',
   'CIM',
   'COS',
   'CR',
   'EGESZSZAM',
   'EXP',
   'GYOK',
   'HA',
   'INDUL',
   'IR',
   'ISMETLES',
   'KEZDET',
   'KISEBB',
   'KONSTANS',
   'LEPESKOZ',
   'LN',
   'LOG',
```

```
    'NAGYOBB',
    'OLV',
    'PARATLAN',
    'PROGRAM',
    'SIN',
    'STOP',
    'TAN',
    'VEG'
  );
  wsym : array [1..c_kszo] of t_symbol =
  (
   abssym,
   akkorsym,
   ameddigsym,
   befejezsym,
   adrsym,
   cossym,
   crsym,
   eszamsym,
   expsym,
   gyoksym,
   hasym,
   indulsym,
   irsym,
   ismetlessym,
   kezdsym,
   kisebbsym,
   konstsym,
   lepessym,
   lnsym,
   logsym,
   nagyobbsym,
   olvsym,
   ptlsym,
   progsym,
   sinsym,
   stopsym,
   tansym,
   vegesym
  );
   mnemonic : array [t_fkt] of string[3] =
      ('CON',
       'OPR',
       'LOD',
       'STO',
       'STA',
       'JMP',
       'JPC',
       'JPN',
       'WRS',
       'WRA',
       'WRC',
       'FUN',
       'INP');

     konstbegsys : t_symset = [konstsym];
     befbegsys   : t_symset = [kezdsym,hasym,ameddigsym,
                               irsym,olvsym,
                               ismetlessym,adrsym,
                               stopsym,bzjelsym];
     faktbegsys  : t_symset = [azsym,numsym,bzarosym,
                               sinsym,tansym,bzjelsym,
                               expsym,logsym,lnsym,
                               gyoksym];

   PROCEDURE Allj;
```

```
BEGIN
   Halt;
END; {.....}

PROCEDURE Hiba (n : integer);

VAR
   f : string[5Ø];
   i : integer;

BEGIN
   case n of
     1: f:='A szam tul nagy';
     2: f:='A konstans tul nagy';
     3: f:='A cimnek pozitiv egeszszamnak kell lennie';
     4: f:='Itt "=" jelnek kell lennie "->" helyett';
     5: f:='Szamot var';
     6: f:='Itt "=" jelnek kell lennie';
     7: f:='Nevet var';
     8: f:='Ervenytelen faktor';
     9: f:='Ismeretlen nev';
    1Ø: f:='">" -t var';
    11: f:='"<" -t var';
    12: f:='")" -t var';
    13: f:='"(" -t var';
    14: f:='Nincs relacio';
    15: f:='A tarolo megadasat varja';
    16: f:='";" -t var';
    17: f:='"VEG" -t var';
    18: f:='Itt "->" -nek kell lennie';
    19: f:='"AKKOR" -t var';
    2Ø: f:='Ervenytelen feltetel';
    21: f:='"PROGRAM" -t var';
    22: f:='A "." hianyzik';
    23: f:='A nev tul hosszu';
    24: f:='Itt "INDUL" -nak kell lennie';
    25: f:='Itt "BEFEJEZ"-nek kell lennie';
    26: f:='":" -t var';
    27: f:='";" nem megengedett';
   end;

    if valasz = 'I'
     then begin
       for i:=1 to cc-1 do Write('_');
       Writeln('^ ',f);
     end;
    err:=err+1;
    if err > c_errmax
     then begin
       Close(prg_in);
       Allj;
     end;
END; {.....}

PROCEDURE Hozz_sym;

VAR
   i,j,k : integer;

   PROCEDURE Hozz_ch;

BEGIN
   if cc = Length(sor)
    then begin
      if Eof(prg_in)
       then begin
```

```
            Writeln('Nem vart program vege');
            Allj;
          end;
          Readln(prg_in,sor);
          sorsz:=sorsz+1;
          if valasz = 'I'
           then Writeln(sor)
           else begin
             Gotoxy(18,5);
             Write(sorsz:5);
           end;
          sor:=sor+' ';
          cc:=Ø;
      end;
    cc:=cc+1;
    ch:=sor[cc];
END; {.....................................}

BEGIN
    while ch = ' ' do Hozz_ch;
    if ch in ['a'..'z','A'..'Z']
     then begin
       az:='';
       repeat
        az:=az+Upcase(ch);
        Hozz_ch;
       until not (Upcase(ch) in
             ['a'..'z','A'..'Z','Ø'..'9']);
       if Length(az) > c_azmax then Hiba(21);
       az:=copy(az,1,c_azmax);
       i:=1;
       j:=c_kszo;
       repeat
         k:=(i+j) div 2;
         if az <= szo[k] then j:=k-1;
         if az >= szo[k] then i:=k+1;
       until i > j;
       if i-1 > j
        then sym:=wsym[k]
        else sym:=azsym;
     end
     else
      if ch in ['Ø'..'9']
       then begin
            j:=Ø;
            num:=Ø;
            sym:=numsym;
            while ch in ['Ø'..'9']
                  do begin
                     num:=1Ø*num+(ord(ch)-ord('Ø'));
                     j:=j+1;
                     Hozz_ch;
                  end;
            if j > c_szjmax then Hiba(1);
            if ch = '.'
              then begin
                   Hozz_ch;
                   j:=Ø;
                   dec:=1;
                   while ch in ['Ø'..'9']
                         do begin
                            j:=j+1;
                            dec:=dec*Ø.1;
                            num:=num+
                                 (ord(ch)-ord('Ø'))*dec;
                            Hozz_ch;
```

```
                        end;
                if j > c_szjmax then Hiba(1);
            end
    end

    else
     if ch = '-'
                then begin
                       Hozz_ch;
                       if ch = '>'
                                then begin
                                     sym:=eredmsym;
                                     Hozz_ch;
                                     end
                                else sym:=minussym;
                     end
                else
                     if ch = '"'
                                then begin
                                     zw_literal:='';
                                     Hozz_ch;
                                     while ch <> '"' do
                                       begin
                                       zw_literal:=
                                       zw_literal+ch;
                                       Hozz_ch;
                                       end;
                                     literale[lit_ind]:=
                                     zw_literal;
                                     sym:=aposym;
                                     Hozz_ch;
                                     end
                                else
                                  if ch = '*'
                                     then begin
                                     Hozz_ch;
                                     if ch = '*'
                                      then begin
                                           sym:=hatvsym;
                                           Hozz_ch;
                                           end
                                      else sym:=szorsym;
                                     end
                                     else begin
                                     sym:=ssym[ch];
                                     Hozz_ch;
                                     end;
if (sym = nagyobbsym) and (ch = '=')
            then begin
            sym:=negyenlosym;
            Hozz_ch;
            end;
if (sym = kisebbsym) and (ch = '=')
            then begin
                  sym:=kegyenlosym;
                  Hozz_ch;
                  end;
   END; {.....................................................}
```

```
PROCEDURE Gen (x  : t_fkt;
               z1 : integer;
               z2 : real);

BEGIN
   if code_ind > c_codemax
    then begin
      Write(' A program tul hosszu');
      Allj;
    end;
   with code[code_ind]
    do begin
      fkt:=x;
      case fkt of
           con:          k:=z2;
           opr,lod,sto,
           sta,jmp,jpc,
           wrs,wra,wrc,
           fun,inp:    a:=z1;
      end;
    end;
   code_ind:=code_ind+1;
END; {.....................................................}

PROCEDURE Teszt (s1,
                 s2 : t_symset;
                 n  : integer);

BEGIN
   if not (sym in s1)
    then begin
      Hiba(n);
      s1:=s1+s2;
      while not (sym in s1) do Hozz_sym;
    end;
END; {.....................................................}

PROCEDURE Kod_lista;

VAR
   i : integer;

BEGIN
   Writeln;
   Writeln;
   Writeln('A GENERALT ASSEMBLER KOD');
   Writeln('-------------------------');
   Writeln;
   Writeln;
   for i:=Ø to code_ind-1 do with code[i]
    do case fkt of
       con: Writeln(i:4,mnemonic[fkt]:4,k:12:5);
       opr,lod,sto,sta,jmp,jpc,jpn,wrs,wra,wrc,inp,fun:
            Writeln(i:4,mnemonic[fkt]:4,a:12);
   end;
END; {.....................................................}

PROCEDURE Torzs (fsys : t_symset);

   PROCEDURE Bevitel;

   BEGIN
      tab_ind:=tab_ind+1;
      with tabla[tab_ind]
       do begin
         nev:=az;
```

```
        if num > c_adrmax
         then begin
           Hiba(2);
           num:=Ø;
         end;
        konst:=num;
    end;
END; {.....}

FUNCTION Pozicio (az : t_az) : integer;

VAR
   i : integer;

BEGIN
  tabla[Ø].nev:=az;
  i:=tab_ind;
  while tabla[i].nev <> az do i:=i-1;
  Pozicio:=i;
END; {.....}

PROCEDURE Konstansdef;

BEGIN
   if sym = azsym
    then begin
      Hozz_sym;
      if sym in [egyenlosym,eredmsym]
       then begin
         if sym = eredmsym then Hiba(4);
         Hozz_sym;
         if sym = numsym
          then begin
            Bevitel;
            Hozz_sym;
          end
          else Hiba(5);
       end
       else Hiba(6);
    end
    else Hiba(7);
END; {.....}


PROCEDURE Utasitas;

VAR
   i,code_ind1,code_ind2 : integer;

   PROCEDURE Kifejezes (fsys : t_symset;
                        fkt  : t_fkt);
   VAR
      plusop : t_symbol;

      PROCEDURE Term (fsys : t_symset);

      VAR
         szorop : t_symbol;

         PROCEDURE Hatvany (fsys : t_symset);

           PROCEDURE Faktor (fsys : t_symset);

           VAR
              i       : integer;
              trigsym : t_symbol;
```

```
BEGIN
Teszt(faktbegsys,fsys+
      [eredmsym,pontosvsym],8);
while sym in faktbegsys
   do begin
      if sym = bzjelsym then
      begin
      Hozz_sym;
      Kifejezes([jzjelsym]+fsys,fkt);
      if sym = jzjelsym
         then Hozz_sym
         else Hiba(12);
      Gen(fkt,Ø,Ø);
      end
      else
      if sym = azsym
         then begin
              i:=Pozicio(az);
              if i=Ø then Hiba(9);
              Gen(con,Ø,tabla[i].konst);
              Hozz_sym;
              end
         else
         if sym = numsym
            then begin
                 Gen(con,Ø,num);
                 Hozz_sym
                 end
            else


            if sym = bzarosym
            then begin
                 Hozz_sym;
                 Kifejezes([jzarosym]+
                           fsys,fkt);
                 if sym = jzarosym
                 then Hozz_sym
                 else Hiba (1Ø);
                 end
            else
            if sym in [abssym,cossym,
                       expsym,lnsym,
                       eszamsym,sinsym,
                       gyoksym,logsym,
                       tansym]
            then begin
                 trigsym:=sym;
                 Hozz_sym;
                 if sym = bzarosym
                 then begin
                      Hozz_sym;
                      Kifejezes
                        ([jzarosym]+fsys
                          ,fkt);
                      Gen(fun,
                          ord(trigsym),Ø);
                      if sym = jzarosym
                      then Hozz_sym
                      else Hiba(1Ø);
                      end
                 else Teszt([],fsys,8);
                 end;
      end;
END; {.........Faktor vege.............}
```

```
      BEGIN
        Faktor(fsys+[szorsym,osztsym,hatvsym]);
        while sym = hatvsym
          do begin
             Hozz_sym;
             Faktor(fsys+[szorsym,osztsym,
                          hatvsym]);
             Gen(opr,6,Ø);
             end;
      END; {..........Hatvany vege.............}

  BEGIN
    Hatvany(fsys+[szorsym,osztsym]);
    while sym in [szorsym,osztsym]
       do begin
          szorop:=sym;
          Hozz_sym;
          Hatvany(fsys+[szorsym,osztsym]);
          if szorop = szorsym
            then Gen(opr,4,Ø)
            else Gen(opr,5,Ø);
          end;
  END; {.............Term vege..................}

BEGIN
  if sym in [plussym,minussym]
     then begin
          plusop:=sym;
          Hozz_sym;
          Term(fsys+[plussym,minussym]);
          if plusop = minussym
             then Gen(opr,1,Ø);
          end
     else Term(fsys+[plussym,minussym]);
     while sym in [plussym,minussym]
           do begin
              plusop:=sym;
              Hozz_sym;
              Term(fsys+[plussym,minussym]);
              if plusop = plussym
                 then Gen(opr,2,Ø)
                 else Gen(opr,3,Ø);
           end;
END; {...........Kifejezes vege.................}
  PROCEDURE Feltetel (fsys : t_symset);

  VAR relop : t_symbol;

  BEGIN
   if sym = ptlsym
     then begin
         Hozz_sym;
         Kifejezes(fsys,lod);
         Gen(opr,7,Ø);
     end
     else begin
          Kifejezes([nemegyenlosym,egyenlosym,
                     kisebbsym,nagyobbsym,
                     kegyenlosym,
                     negyenlosym]+fsys,lod);
          if not (sym in [egyenlosym,nemegyenlosym,
                          kisebbsym,nagyobbsym,
                          kegyenlosym,negyenlosym])
            then Hiba(14)
            else begin
                 relop:=sym;
```

```
                Hozz_sym;
                Kifejezes(fsys,lod);
                case relop of
                     egyenlosym : Gen(opr,8,Ø);
                     nemegyenlosym : Gen(opr,9,Ø);
                     kisebbsym : Gen(opr,1Ø,Ø);
                     negyenlosym : Gen(opr,13,Ø);
                     nagyobbsym : Gen(opr,12,Ø);
                     kegyenlosym : Gen(opr,11,Ø);
                end;
          end;
     end;
END; {.....................................}

PROCEDURE IrUtasitas;

BEGIN
    if sym = aposym
     then begin
          Gen(wra,lit_ind,Ø);
          lit_ind:=lit_ind+1;
          Hozz_sym ;
          end
     else if sym = crsym
           then begin
                Gen(wrc,Ø,Ø);
                Hozz_sym;
                end
           else begin
           Kifejezes(fsys,lod);
           Gen(wrs,Ø,Ø);
           end;
END; {.....................................}

    PROCEDURE OlvUtasitas;

    BEGIN
    if sym = bzjelsym
     then begin
       Hozz_sym;
       Kifejezes([jzjelsym]+fsys,lod);
       if sym = jzjelsym
        then Hozz_sym
        else Hiba(12);
       Gen(inp,Ø,Ø);
     end
     else Hiba(15)
    END; {.....................................}

BEGIN
    if sym = hasym
     then begin
       Hozz_sym;
       Feltetel([akkorsym]+fsys);
       if sym = akkorsym
        then Hozz_sym
        else Hiba(19);
       code_ind1:=code_ind;
       Gen(jpn,Ø,Ø);
       Utasitas;
       code[code_ind1].a:=code_ind;
     end
     else
      if sym = ameddigsym
       then begin
         code_ind1:=code_ind;
         Hozz_sym;
```

```
    Feltetel([kettospontsym]+fsys);
    code_ind2:=code_ind;
    Gen(jpn,Ø,Ø);
    if sym = kettospontsym
     then Hozz_sym
     else Hiba(26);
    Utasitas;
    Gen(jmp,code_ind1,Ø);
    code[code_ind2].a:=code_ind;
  end



  else
   if sym = irsym
    then begin
      Hozz_sym;
      IrUtasitas;
      while sym = vesszosym
       do begin
         Hozz_sym;
         IrUtasitas;
       end;
      end
      else
       if sym = olvsym
        then begin
          Hozz_sym;
          OlvUtasitas;
          while sym = vesszosym
           do begin
             Hozz_sym;
             OlvUtasitas;
           end;
        end
        else
          if sym = kezdsym
           then begin
             Hozz_sym;
             Utasitas;
             while sym in [pontosvsym]+befbegsys
              do begin
                 if sym = pontosvsym
                  then Hozz_sym
                  else Hiba(16);
                 Utasitas;
              end;
             if sym = vegesym
              then Hozz_sym
              else Hiba(17);
           end
           else
            if sym = ismetlessym
             then begin
               Hozz_sym;
               if sym = indulsym
                then Hozz_sym
                else Hiba(24);
               Kifejezes(fsys,lod);
               if sym = eredmsym
                then Hozz_sym
                else Hiba(18);
               if sym = bzjelsym
                then Hozz_sym
                else Hiba(12);
```

```
      Kifejezes([jzjelsym]+fsys,sto);
      if sym = jzjelsym
       then Hozz_sym
       else Hiba(13);
      Gen(sta,c_adrmax-2,Ø);
      Gen(con,Ø,c_adrmax-2);
      Gen(lod,Ø,Ø);
      Gen(sto,Ø,Ø);
      code_ind1:=code_ind;
      if sym = befejezsym
       then Hozz_sym
       else Hiba(25);
      Feltetel(fsys);
      code_ind2:=code_ind;
      Gen(jpc,Ø,Ø);
      if sym = lepessym
       then begin
         Hozz_sym;
         Kifejezes(fsys,lod);
         Gen(con,Ø,c_adrmax-1);
         Gen(sto,Ø,Ø);
       end
      else begin
         Gen(con,Ø,1);
         Gen(con,Ø,c_adrmax-1);
         Gen(sto,Ø,Ø);
         end;
      if sym = kettospontsym
       then Hozz_sym
       else Hiba(26);
      Utasitas;
      Gen(con,Ø,c_adrmax-2);
      Gen(lod,Ø,Ø);
      Gen(lod,Ø,Ø);
      Gen(con,Ø,c_adrmax-1);
      Gen(] d,Ø,Ø);
      Gen(opr,2,Ø);
      Gen(con,Ø,c_adrmax-2);
      Gen(lod,Ø,Ø);
      Gen(sto,Ø,Ø);
      Gen(jmp,code_ind1,Ø);
      code[code_ind2].a:=code_ind;
    end
    else
     if sym = adrsym
      then begin
        Hozz_sym;
        Kifejezes(fsys,lod);
        Gen(sta,c_adrmax,Ø);
        if sym = kettospontsym
         then Hozz_sym
         else Hiba(26);
        Utasitas;
        Gen(con,Ø,Ø);
        Gen(sta,c_adrmax,Ø);
      end
      else
       if sym = stopsym
        then begin
             Hozz_sym;
             Gen(opr,Ø,Ø);
             end
        else if sym in faktbegsys
              then begin
                Kifejezes(fsys,lod);
                if sym = eredmsym
                 then Hozz_sym
                 else Hiba(18);
                if sym = bzjelsym
```

```
                                    then Hozz_sym
                                    else Hiba(13);
                                  Kifejezes([jzjelsym]+fsys,lod);
                                  Gen(sto,Ø,Ø);
                                  if sym = jzjelsym
                                   then Hozz_sym
                                   else Hiba(12);
                                end;

END; {.............Utasitas vege.....................}

BEGIN
   repeat
     if sym = konstsym
      then begin
        Hozz_sym;
        repeat
          Konstansdef;
          while sym = vesszosym
           do begin
             Hozz_sym;
             Konstansdef;
           end;
          if sym = pontosvsym
           then Hozz_sym
           else Hiba(16);
        until sym <> azsym;
      end;
   until not (sym = konstsym);
   Utasitas;
   Gen(opr,Ø,Ø);
   Teszt(fsys,[],22);
END; {..............Torzs vege.........................}

PROCEDURE Hozz_valasz;

BEGIN
   repeat
     read(valasz);
     valasz:=Upcase(valasz);
   until valasz in ['N','I'];
END; {.....................................................}

BEGIN
   ClrScr;
   Lowvideo;
   Write('KEREM A PROGRAM NEVET (Forras): ');
   repeat
     Highvideo;
     Read(prgnev);
     Lowvideo;
     Assign(prg_in,prgnev);
     {$I-}
     Reset(prg_in);
     {$I+}
     ok:=IOresult = Ø;
     if not ok
      then begin
         GotoXY(1,3);
         write('A PROGRAM NEM TALALHATO! VEGE(I/N)? ');
         Highvideo;
         Hozz_valasz;
         if valasz = 'I' then Allj;
         GotoXY(1,3);
         ClrEol;
         GotoXY(33,1);
         ClrEol;
```

```
    end;
until ok;
GotoXY(1,3);
Write('PROGRAM es KOD LISTAZASA (I/N) ?  ');
Highvideo;
Hozz_valasz;
if valasz = 'N'
 then begin
   GotoXY(1,5);
   Lowvideo;
   Writeln('ELEMZETT SOROK: ');
   Highvideo;
 end
 else begin
   Writeln;
   Writeln;
 end;

sorsz:=Ø;
err:= Ø;
sor:= '';
cc:= Ø;
tab_ind:= Ø;
lit_ind:=Ø;
code_ind:=Ø;
for ch:=chr(Ø) to chr(c_ascii-1) do
                                  ssym[ch]:=uressym;

ssym['+']:=plussym;
ssym['-']:=minussym;
ssym['*']:=szorsym;
ssym['/']:=osztsym;
ssym['(']:=bzjelsym;
ssym[')']:=jzjelsym;
ssym['=']:=egyenlosym;
ssym[',']:=vesszosym;
ssym['.']:=pontsym;
ssym['#']:=nemegyenlosym;
ssym['<']:=bzarosym;
ssym['>']:=jzarosym;
ssym[';']:=pontosvsym;
ssym['"']:=aposym;
ssym[':']:=kettospontsym;
ch:=' ';
Hozz_sym;
Teszt([progsym],konstbegsys+befbegsys,21);
if sym = progsym then Hozz_sym;
if sym = azsym
 then Hozz_sym
 else Hiba(7);
if sym = pontosvsym then Hozz_sym else Hiba(16);
Torzs([pontsym]+konstbegsys+befbegsys);
if sym <> pontsym then Hiba(22);

if valasz = 'I'
 then Kod_lista
 else writeln;
Writeln;
Writeln('***',err:4,' *** HIBA LEPETT FEL');
Close(prg_in);
if err = Ø
then begin
  Writeln;
  Lowvideo;
  Write('AKARJA A KODOT TAROLNI?   (I/N): ');
  Highvideo;
  Hozz_valasz;
```

```
        if valasz = 'N' then Allj;
        Writeln;
        Writeln;
        Lowvideo;
        Write('KEREM A KOD-FILE NEVET! ');
        repeat
          Highvideo;
          Read(prgnev);
          Lowvideo;
          Assign(prg_out,prgnev);
          {$I-}
          Rewrite(prg_out);
          {$I+}
        until IOresult = 0;
        pgm.code:=code;
        pgm.literale:=literale;
        Write(prg_out,pgm);
        Close(prg_out);
      end;
END.
```

Miután – az előző fejezetekben kifejlesztett – a fordító előállította az assemblerkódot, ezt az absztrakt gép már fel tudja dolgozni. Ezt a gépet a *RUN* programmal működtetjük, így az assemblerprogramok a

RUN programnév

segítségével elindíthatók lesznek.

A teljes program a következő oldalakon található meg. A programnév paraméterként való átadására a pufferváltozót a Cseg:$80 abszolút címre definiáltuk. A *Turbo–Pascal* 2.0-ban ez az egyetlen lehetőség, hogy a *DOS* alatti parancssorhoz hozzáférjünk.

```
{ RUN.PAS       }
{ Pr. II. 2.3. }

PROGRAM Run (input,output);

{ Ennek a programnak a leforditott valtozataval,
  RUN.COM - mal lehet LOLA programot vegrehajtatni,
  --- A> RUN LolaProgramNev ---  paranccsal, ahol
  a LolaProgramNev a FORDITO program "KEREM A KOD-FILE
  NEVET!" felszolitasara adott valasz. }

CONST
   c_cimmax   = 5Ø3;
   c_kodmax   = 5ØØ;
   c_szjmax   = 6;
   c_litmax   = 5Ø;

TYPE
   t_a            = Ø..c_cimmax;
   t_fkt          = (con,opr,lod,sto,sta,jmp,jpc,jpn,
                     wrs,wrl,wrc,fun,inp);
   t_literal      = string [8Ø];

   t_utasitas = record
                   case fkt : t_fkt of
                        con              : (k: real);
                        opr,lod,sto,
                        sta,jmp,jpc,
                        jpn,wrl,wrs,
                        wrc,fun,inp      : (a : t_a);
                end;

   t_kod          = array [Ø..c_kodmax] of t_utasitas;

   t_literale     = array [Ø..c_litmax] of t_literal;

   t_pgm          = record
                      kod     : t_kod;
                      literale : t_literale;
                    end;

VAR
   kod       : t_kod;
   puffer    : string[127] absolute cseg:$8Ø;
   prg_in    : file of t_pgm;
   pgm       : t_pgm;
   literale  : t_literale;
   i         : integer;
```

```
PROCEDURE Vegrehajtas;

{ A LOLA fordito altal egy file-ba letett assembler
program vegrehajtasa }

CONST
   c_veremmax  = 5ØØ;
   cr           = 13;

TYPE
   t_tarolo  = array [Ø..c_cimmax] of real;
   t_verem   = array [Ø..c_veremmax] of real;

VAR
   p,s       : integer;          { Programm-, veremmutato }
   utasitas  : t_utasitas;       { Utasitasregiszter }
   sp        : t_tarolo;         { Adattarolo }
   st        : t_verem;          { Verem }
   bevitel   : string [c_szjmax];
   eredm     : integer;
   i         : integer;
   ss        : real;

   PROCEDURE Hiba (n : integer);

   { Futas kozbeni hiba eseten a vegrehajtast
     megszakitja es megfelelo hibauzenetet kuld }

   VAR f : string[4Ø];

   BEGIN
      case n of
        1: f:='Ervenytelen szam';
        2: f:='Negativ szam gyoke';
        3: f:='Az argumentum nem esik -1 es 1 koze';
        4: f:='Az argumentum kisebb, mint Ø';
        5: f:='A kitevo valos es negativ';
      end;
      Writeln('*** FUTAS KOZBENI HIBA ',f);
      Halt;
   END; {.....................................}

BEGIN
   s:=Ø;
   p:=Ø;
   Fillchar(sp,c_cimmax,Ø);

 { A fociklusnak vege van:  OPR Ø  assembler utasitas
   eseten, vagy futas kozbeni hiba fellepesekor }

   repeat
     utasitas:=kod[p];
     p:=p+1;
     with utasitas
      do case fkt of
        con: begin
               s:=s+1;
               st[s]:=k;
             end;
        fun: case a of
               Ø: st[s]:=abs(st[s]);
               1: st[s]:=cos(st[s]);
               2: st[s]:=exp(st[s]);
               3: begin
                    if st[s] < Ø then Hiba(4);
                    st[s]:=ln(st[s])/ln(1Ø);
                  end;
```

```
          4: begin
               if st[s] < Ø then Hiba(4);
               st[s]:=ln(st[s]);
             end;
          5: st[s]:=sin(st[s]);
          6: begin
               if (st[s] > 1) or (st[s] < -1)
                then Hiba(3);
               st[s]:=sin(st[s])/cos(st[s]);
             end;
          7: begin
               if st[s] < Ø then Hiba(2);
               st[s]:=sqrt(st[s]);
             end;
          8: st[s]:=trunc(st[s]);
        end;
opr:    case a of
        Ø: halt;
        1: st[s]:=-st[s];
        2: begin
             s:=s-1;
             st[s]:=st[s]+st[s+1];
           end;
        3: begin
             s:=s-1;
             st[s]:=st[s]-st[s+1];
           end;

        4: begin
             s:=s-1;
             st[s]:=st[s]*st[s+1];
           end;
        5: begin
             s:=s-1;
             st[s]:=st[s]/st[s+1];
           end;
        6: begin
             s:=s-1;
             ss:=1;
             if st[s+1]>=1 then
             for i:=1 to trunc(st[s+1]) do
                 ss:=ss*st[s];
             st[s]:=ss
           end;
        7: st[s]:=ord(odd(trunc(st[s])));
        8: begin
             s.=s-1;
             st[s]:=ord(st[s] = st[s+1]);
           end;
        9: begin
             s:=s-1;
             st[s]:=ord(st[s] <> st[s+1]);
           end;
       1Ø: begin
             s:=s-1;
             st[s]:=ord(st[s] < st[s+1]);
           end;
       11: begin
             s:=s-1;
             st[s]:=ord(st[s] <= st[s+1]);
           end;
       12: begin
             s:=s-1;
             st[s]:=ord(st[s] > st[s+1]);
           end;
       13: begin
             s:=s-1;
```

```
                st[s]:=ord(st[s] >= st[s+1]);
              end;
         14: begin
                s:=s-1;
                if (st[s] <= Ø ) and
                   (st[s+1]-trunc(st[s+1]) <> Ø)
                 then Hiba(5);
                st[s]:=exp(st[s+1]*ln(st[s]));
              end;
       end;
   lod: st[s]:=sp[trunc(st[s]+sp[c_cimmax])];
   sto: begin
          sp[trunc(st[s])]:=st[s-1]+sp[c_cimmax];
          s:=s-2;
        end;

        sta: begin
               sp[a]:=st[s];
               s:=s-1;
             end;
        jmp: p:=a;
        jpc: begin
               if st[s] = 1 then p:=a;
               s:=s-1;
             end;
        jpn: begin
               if st[s] = Ø then p:=a;
               s:=s-1;
             end;
        inp: begin
               repeat
                  Readln(bevitel);
               until Length(bevitel) > Ø;
               Val(bevitel,sp[Trunc(st[s]+
               sp[c_cimmax])],eredm);
               if eredm <> Ø then Hiba(1);
               s:=s-1;
             end;
        wrs: begin
               write(st[s]:1Ø:3);
               s:=s-1;
             end;
        wrl: write(literale[a]);
        wrc: writeln;
      end;
    until p=Ø;
END; {.....}

BEGIN

 { A parancssorbol a program nevenek behozatala }

   Assign(prg_in,Copy(puffer,2,Length(puffer)-1));
   {$I-}
   Reset(prg_in);
   {$I+}
   if IOresult <> Ø
    then begin
      Writeln('PROGRAM NEM TALALHATO');
      Halt;
    end;

   Read(prg_in,pgm);
   kod:=pgm.kod;
   literale:=pgm.literale;
   Vegrehajtas;
END.
```

Az interpreter futási tesztjei a *LOLA* assembler-programokkal figyelemre méltó programfeldolgozási sebességet mutattak. Természetesen mi is tudjuk, hogy a *LOLA* nyelv nem perfekt, azonban a célunk egy olyan egyszerű programnyelv közreadása volt, amely e könyv kereteit nem haladja meg.

3. Számolás nagyobb pontossággal

Az *integer* számokat a *Turbo*–Pascalban egy szóban tároljuk, azaz ábrázolásukhoz 16 bit áll rendelkezésre. Ezzel a –32768 és a +32767 közötti értéktartomány fedhető le. Sok felhasználáshoz azonban nem elegendő ez az értéktartomány és a *real* számábrázolásra való áttérés sem nyújt elegendő értékes jegyet.
Ebben a fejezetben eljárásokat közlünk a tetszőlegesen nagy egész számokkal való négy alapművelet elvégzésére.

3.1. Alapvető megoldások

Mielőtt a négy alapművelethez tartozó algoritmusokon töprengenénk, még két alapvető kérdést kell tisztáznunk.

1. A bővített pontosságú számokat melyik adattípussal lehet a legkedvezőbben ábrázolni?
2. Hogyan ábrázoljuk a negatív számokat?

Az adatok formátuma

Mivel adatformátumunkkal kapcsolatban az a követelmény, hogy egy decimális szám lehetőleg sok jeggyel ábrázolható legyen, bizonyosan tömböt választunk, ahol minden egyes elem a számunk egy számjegyének felel meg. Hogy a számjegyekkel számolni is tudjunk, a byte-tömb (array of byte) kínálkozik tárolóként.

Mivel egy byte számban a 0-tól 255-ig terjedő értéktartomány ábrázolható (l. az elméleti rész 3.1.2. pontját), egy byte-ben legfeljebb 28 decimális számjegy összege helyezhető el (9 * 28 = 252), ha több jegyet akarunk összegezni – ahogy ezt még látni fogjuk, a több mint 28 jegyű számok szorzásakor –, akkor a számokat az egész tömbben (array of integer) kell tárolnunk.

Negatív számok ábrázolása

Az elméleti rész 3.1.2. pontjában már megismerkedtünk a negatív bináris számoknak a kettes komplemens segítségével való ábrázolásával. A bináris számok esetében ennek az ábrázolási formának meggyőző előnyei vannak.

- azonos algoritmus alkalmazható a pozitív számok negatív számmá való átalakítására, ill. megfordítva is;
- az, hogy egy szám negatív vagy pozitív, a szám bal szélső jegyéről felismerhető;
- az alapműveletek algoritmusai a pozitív és negatív számokhoz egyaránt alkalmazhatóak;
- a nullának csak egy ábrázolása van.

Ezeket az előnyöket akkor is felhasználhatjuk, ha a decimális rendszer negatív számait a kettes komplemenssel analóg módon a tízes komplemens képzésével ábrázoljuk; azaz minden számjegyhez felírjuk a 9-re való kiegészítését és az így előálló számhoz hozzáadunk 1-et. A bal szélen túlcsorduló jegyet levágjuk.

Hányadik hely valójában a bal szélen levő? Ez természetesen a választott tömb méretétől függ. A további magyarázatokhoz feltételezzük, hogy 5 jegyet tudunk ábrázolni, amelyeket jobbról balra 1-től 5-ig sorszámozunk.
A tizes komplemenshez való szoktatás kedvéért képezzük a 970 tízes komplemensét:

00970 a kiindulási szám
99029 minden számjegyre képeztük a 9-re való kiegészítő jegyet
+1 hozzáadunk 1-et
99030 a tízes komplemens

Képezzük még a 99030 tízes komplemensét:

99030 a kiindulási szám
00969 minden számjegyre képeztünk a 9-re való kiegészítő jegyet
+1 hozzáadunk 1-et
00970 a tízes komplemens

Azt vártuk, hogy az eredmény éppen az eredeti 970 legyen.

A bal szélen lévő hely – az 5. hely – előjelként szerepel; azaz az 5. helyen lévő 0 pozitív, a 9 pedig negatív számot jelez. Ezzel, tehát –9999 és +9999 értéktartományon belül ábrázolhatunk számokat. Általános érvényű, hogy n helyen -10^n+1 és $+10^n-1$ közötti értéktartományban ábrázolhatók a számok.

3.2. ÖSSZEADÁS

Mivel nem kell azzal törődnünk, hogy negatív vagy pozitív számot kell összeadnunk, az összeadás igen egyszerű eljárás lesz. Ugyanúgy, ahogy ezt papíron végeznénk, jobbról balra haladva az összeadandók megfelelő helyi értékeit összeadjuk, és hozzáadjuk az előző helyről származó maradékot. Az összeg egyes helyi értékét, amely a 10-zel való egész számú osztás maradéka, tároljuk az eredményben és a 10-zel való egész számú osztás eredményét az átvitelhez megjegyezzük.

Egész egyszerűen a második összeadandó tízes komplementerét képezzük és ezzel a kivonást az összeadásra vezettük vissza.

3.4. SZORZÁS

Mint az összeadásnál, itt is átvehető a papíron végzett szorzás módszerére épülő algoritmus. Nézzük meg ezt az algoritmust a 738 * 245 szorzási példán:

```
  738 * 245
 ---------
  3690
 2952
1476
 ---------
180810
```

Ha az első szám számjegyeit a1 ... a5-tel jelöljük, valamint a második számét b1 ... b5-tel, akkor a módszert általánosabb formában az alábbiak szerint írhatjuk:

			*	a5	a4	a3	a2	a1
				b5	b4	b3	b2	b1
				a5*b1	a4*b1	a3*b1	a2*b1	a1*b1
			a5*b2	a4*b2	a3*b2	a2*b2	a1*b2	
		a5*b3	a4*b3	a3*b3	a2*b3	a1*b3		
	a5*b4	a4*b4	a3*b4	a2*b4	a1*b4			
a5*b5	a4*b5	a3*b5	a2*b5	a1*b5				
s9	s8	s7	s6	s5	s4	s3	s2	s1

Két 5 jegyű szám szorzása tehát legfeljebb 10 jegyű eredményt adhat, ahol a 10. helyen csak az átvitel állhat.
Könnyen belátható, hogy ez a megállapítás n jegyű számokra is általánosítható: két n jegyű szám szorzata legfeljebb $2n$ jegyű eredményt ad, ahol a $2n$-edik helyen csak az átvitelből származó jegy állhat.
Tekintsük az indexeket részletesen, amelyeket az egyes összeadandókhoz használunk.
Legyen:

i az összeg számjegyek	$1 \ldots 2n-1$ közötti futóindexe
j az első szám	$1 \ldots n$ helyi értéke közötti futóindex
k a második szám	$1 \ldots n$ helyi értéke közötti futóindex

így belátható, hogy a részszorzatok mindegyikére érvényes

$$i+k = i+1$$

Ebből következően a k indexet

$$k = i-j+1$$

eredményeként állíthatjuk elő.
Az i-edik részszorzatok összegének számítására összegeznünk kell az összes

$$a(j)*b(k),$$

azaz a fentiek felhasználásával

$$a(j)*b(i-j+1)$$

Részszorzatot, ahol j értéke 1-től i-ig változik, és az a, ill. b szám indexe nem lehet n-nél nagyobb.

A *Turbo–Pascal*-ban erre az alábbi *for*-ciklus készíthető:

```
for   j:=1 to i do
      if (j<=n) and (i-j+1<=n) then s(j):=s(j)+a(j)*b(i-j+1);
```

A szorzás eljárásának a megvalósításánál arról kell még gondoskodnunk, hogy minden összegzési helyi értékre csak egy decimális helyi érték legyen tárolva és az így előálló átvitelről kellően gondoskodjunk. Ennek a megoldását már az összeadásnál megtárgyaltuk.

3.5. OSZTÁS

Egész számok osztására van egy egyszerű és ismert eljárás, az osztandóból az osztót annyiszor kell kivonni, amíg az osztandó (maradék) kisebb nem lesz az osztónál. A kivonások száma a hányados. Ezt az egyszerű módszert a már ismert eljárásokkal megvalósíthatjuk. Hátránya azonban, hogy nagy osztandó és kis osztó esetén nagyon sok kivonást kell elvégezni.
Eröltessük meg magunkat és optimalizáljuk ezt a módszert.
Gondolkozzunk a következő példán: 7672 div 28 = 274.
A leírt eljárás értelmében tehát, 274 kivonást kellene elvégeznünk.
Nyúljunk ismét a már bevált papíron végzett művelet módszeréhez. Mit csinálnánk ennél az osztásnál?
Első lépésként visszavezetnénk a feladatot 76 div 28-ra az osztandó 2. és 3. számjegye közé tett kicsiny jellel, ezzel a hányadosnak csak a százas helyi értékét kapnánk, majd a tízes, végül az egyes helyi értékét. De miért éppen az osztandó 2. és 3. helyi értéke közé tettük a jelet kezdéskor és honnan tudjuk, hogy ezzel a hányados százas helyi értékét határoztuk meg?
Ha nem volna olyan nagy gyakorlatunk az osztásban, akkor az osztót jobbról annyi nullával egészítenénk ki, amíg nagyobb nem lesz az osztandónál: 7672 div 28 000. Ezután az osztó végéről lehúzunk egy nullát és rögtön látjuk, hogy a 7672-ből a 2800 pontosan 2-szer vonható le, amíg a maradék a jelenlegi osztónál, 2800-nál kisebb nem lesz. Mivel az osztót két nullával egészítettük ki, a kapott 2-t is két nullával kell kiegészítenünk mielőtt hányadosként figyelembe vennénk.
Hasonlóan járunk el a maradékhelyekkel; közben már nem kell az osztót a 28-tól kezdve megfelelő számú nullákkal kibővítenünk, hanem minden lépésnél lehúzunk egy nullát az előző osztóból.

7672 div 2800 = 2, maradék = 2072, hányados = 200
2072 div 280 = 7, maradék = 112, hányados = 200 + 70
112 div 28 = 4, maradék = 0, hányados = 200 + 70 + 4

Ezzel a 274 kivonás helyett csak 2+7+4 kivonást és néhány eltoló műveletet kellett elvégeznünk a nullák hozzáfűzésére.

Szeretnénk az alapműveleteinket lehetőség szerint kényelmesen felhasználni, ez azt jelenti, hogy a kibővített pontosságú számok beolvasására és kivitelére még eljárásokat kell létrehoznunk.

Az egyszerűbb eset a kivitel lesz, amelyet a programpéldában a Kodolo eljárással valósítottunk meg. Az egésztömböt egyszerűen egy karakterlánccá alakítottuk át.
A bevitel, amelyet a Dekodolo függvénnyel valósítottunk meg, először láncot olvas be és átalakítja azt karakterről karakterre az egésztömb egyik elemévé. Közben természetesen vizsgálnunk kell, hogy csak számértékeket olvasson be és negatív előjel esetén a beolvasott szám tízes komplemensét kell képeznünk. A függvény *igaz* (true) vagy *hamis* (false) értékű aszerint, hogy a láncban csak érvényes, vagy érvénytelen karakterek is voltak.

```
{ Szamolas.pas }
{ Pr. II. 3.1. }

PROGRAM Szamolas;

{ Szamolas nagyobb pontossaggal }

CONST
   n  = 10;
   n2 = 20;

TYPE
   t_di = array[1..n] of integer;
   t_dr = array[1..n2] of integer;
   t_s  = string[n2];

VAR
   a,b,c,r     : t_di;
   s1,s2,szam : t_s;
   muvjel     : char;

PROCEDURE Komplemens (a : t_di; VAR b : t_di);

{ 10-es komplemens szamitasa }

VAR
   u,i : integer;

BEGIN
   u:=1;
   for i:=1 to n
    do begin;
      b[i]:=(9-a[i]+u) mod 10;
      u:=(9-a[i]+u) div 10;
    end;
END; {.....}

FUNCTION Dekodolo (VAR a : t_di; s : t_s):boolean;

{ Karakterlanc szamma konvertalasa }

VAR
   i,j:integer;

BEGIN
   Dekodolo:=true;
   for j:=1 to n do a[j]:=0;
   j:=1;
   if Length(s)>n
```

```
     then Dekodolo:=false
     else begin
       for i:=Length(s) downto 1
        do begin;
          if s[i]='-'
           then begin
             Komplemens(a,a);
             i:=1;
           end
           else
             if (Ord(s[i])<48) or (Ord(s[i])>57)
              then begin
                Dekodolo:=false;
                i:=1;
              end
              else begin
                a[j]:=Ord(s[i])-48;
                j:=j+1;
              end;
        end;
     end;
END; {.....................................................}

PROCEDURE Kodolo (a : t_di; VAR s : t_s);

{ Szam karaktersorozatta alakitasa }

VAR
   i,j:integer;
   b:t_di;

BEGIN
   s:='';
   b:=a;
   if b[n]=9
    then begin;
      s:='-';
      Komplemens(b,b);
    end;
   i:=n-1;
   while b[i]=Ø do i:=i-1;
   for j:=i downto 1 do s:=s+Chr(b[j]+48);
   if s='' then s:='Ø';
END; {.....................................................}


PROCEDURE Osszeadas (a,b : t_di; VAR c : t_di);

          { c:=a+b }
VAR
   i : integer;
   d : t_dr;

BEGIN
   d[1]:=Ø;
   for i:=1 to n
    do begin;
      d[i]:=d[i]+a[i]+b[i];
      d[i+1]:=d[i] div 1Ø;
      d[i]:=d[i] mod 1Ø;
    end;
   for i:=1 to n do c[i]:=d[i];
END; {.....................................................}

PROCEDURE Kivonas (a,b : t_di; VAR c : t_di);

          { c:=a-b }
```

```
BEGIN
   Komplemens(b,b);
   Osszeadas(a,b,c);
END; {.....}

PROCEDURE Szorzas (a,b : t_di; var c : t_di);

          { c:=a*b }

VAR
   i,j : integer;
   d   : t_dr;

BEGIN
   d[1]:=Ø;
   for i:=1 to n2-1
    do begin;
      for j:=1 to i do
        if (i-j+1 <=n) and (j<=n) then
        d[i]:=d[i]+a[i-j+1]*b[j];
      d[i+1]:=d[i] div 1Ø;
      d[i]:=d[i] mod 1Ø;
    end;
   for i:=1 to n do c[i]:=d[i];
END; {.....}

FUNCTION Hasonlitas (a,b : t_di) : char;

{ a<b: Hasonlitas:='<' }
{ a=b: Hasonlitas:='=' }
{ a>b: Hasonlitas:='>' }

VAR
   v : char;
   i : integer;

BEGIN
   v:='=';
   i:=n;
   while (v='=') and (i>=1)
    do begin;
      if a[i]<b[i]
       then v:='<'
       else if a[i]>b[i] then v:='>';
      i:=i-1;
    end;
   Hasonlitas:=v;
END; {.....}

PROCEDURE Balra (a : t_di; inc :integer;
                 VAR b : t_di);

{ "b" az "a" ertek inc hellyel
   balra tolt erteket veszi fel }

VAR
   i : integer;

BEGIN
   for i:=1+inc to n do b[i]:=a[i-inc];
   for i:=1 to inc do b[i]:=Ø;
END; {.....}

PROCEDURE Jobbra (a : t_di; inc : integer;
                  VAR b : t_di);

{ "b" az "a" ertek inc hellyel
```

```
   jobbra tolt erteket veszi fel }

VAR
   i : integer;

BEGIN
   for i:=1 to n-inc do b[i]:=a[i+inc];
   for i:=n-inc+1 to n do b[i]:=Ø;
END; {.....................................................}



PROCEDURE Osztas (a,b : t_di; VAR q : t_di;
                  VAR r : t_di);

{ q:=a div b; r:=a mod b }

VAR
   d,hq,k,null,egy  : t_di;
   z,i              : integer;
   vza,vzb          : boolean;

BEGIN
   if Dekodolo(null,'Ø') then;
   if Dekodolo(egy,'1') then;
   if Hasonlitas(b,null)<>'='    { A Ø-val valo osztas }
    then begin;                  { elkerulese          }
      vza:=false; vzb:=false;
      if a[n]<>Ø
       then begin
         vza:=true;
         Komplemens(a,a);
       end;
      if b[n]<>Ø
       then begin
         vzb:=true;
         Komplemens(b,b);
       end;
      r:=a;
      d:=a;
      q:=null;
      hq:=b;
      z:=Ø;
      while Hasonlitas(hq,d)<>'>'
       do begin
         Balra(hq,1,hq);
         z:=z+1;
       end;
      for i:=z downto 1
       do begin
         k:=null;
         Jobbra(hq,1,hq);
         while Hasonlitas(d,hq)<>'<'
          do begin
            Kivonas(d,hq,d);
            Osszeadas(k,egy,k);
          end;
          Balra(k,i-1,k);
          Osszeadas(q,k,q);
       end;
      r:=d;
      if vza xor vzb then Komplemens(q,q);
    end;
END; {.....................................................}
```

```
BEGIN
   ClrScr;
   repeat
     repeat
       Write('Elso szam (vagy VEGE)   -----> ');
       ReadLn(s1);
     until (s1='VEGE') or Dekodolo(a,s1);
     if s1<>'VEGE'
      then begin;
        repeat
          Write('Masodik szam           -----> ');
          ReadLn(s2);
        until Dekodolo(b,s2);
        repeat
          Write('Muvelet (*,+,-,/)      -----> ');
          ReadLn(muvjel);
        until (muvjel='*') or
              (muvjel='+') or
              (muvjel='-') or
              (muvjel='/');
        case muvjel of
          '*':Szorzas(a,b,c);
          '+':Osszeadas(a,b,c);
          '-':Kivonas(a,b,c);
          '/':begin;
                Osztas(a,b,c,r);
                Kodolo(r,szam);
                Write('         ');
                Writeln('            Maradek:  ',szam);
              end;
        end;
        Kodolo(c.szam);
        WriteLn('                    Eredmeny: ',szam);
        Write('========');
        WriteLn('====================================');
        WriteLn;
      end;
   until s1='VEGE'
END.
```

4. Rendszerprogramozás

Ebben a fejezetben olyan programokat írunk le, amelyek reprezentatív példái a *Turbo–Pascal*-ban való rendszerprogramozásnak. Több, az elméleti részben szereplő függvény és eljárás megtalálható itt, használat közben. Ezen túlmenően ezek a programok a gyakorlatban is jól használhatók.

A példában egy egyszerű file-kezelést valósítottunk meg a *Turbo–Pascal* segítségével. A program használatát két menülista segíti.

1. Menü: *Választás*

   A felsorolt funkciók a program általános vezérlését végzik. A következő választási lehetőségek állnak a rendelkezésünkre:

   Hajtsd végre
   A kiválasztott tevékenység végrehajtása.

   Be file-eleje
   A file-ok első karakterei a képernyő jobb oldalán megjelennek.

   Következő file
   A kurzor egy sorral lejjebb kerül.

   Út
   Egy új elérési út kiválasztása.

   Quit
   Program vége a kiválasztott tevékenység végrehajtása nélkül.

   Előző file
   A kurzor egy sorral feljebb kerül.

2. Menü: *File*

   A file-kezelő tevékenységek a felsorolt funkciókból választhatók ki. A végrehajtás ezután a *Választás* menü **H** funkciójának a kiválasztásával lehetséges.

   Attributum
   A file-jellemzőket lehet megváltoztatni.

   Mozgatás
   A file átmásolása és törlése.

   Duplikálás
   A file másolása.

   Törlés
   A file törlése.

   Átnevezés
   A file átnevezése.

   Vissza
   A kurzor egy sorral feljebb kerül.

A program lényege az, hogy az aktuális vagy a *Választás* menüben kiválasztott könyvtárbejegyzések a *d* táblába kerülnek. A táblaelemeket a képernyőn görgetve (scroll) lehet megjeleníteni és megjelölni.

```
{ FileDir.pas  }
{ Pr. II. 4.1. }

{ Az alabbi DOS-funkciok hivasat mutatjuk be:
  $1A      - Disk Transfer Address beallitasa,
  $43      - File attributumok olvasasa es beallitasa,
  $4E $4F  - Peldakent tartalomjegyzek olvasasa }

{$C+            A program ^C -vel megszakithato }

PROGRAM FileDir;              { Tartalomjegyzek kezelese  }

LABEL 1;                      { Eloreugras cikluson belul }
```

```
CONST
   Verzio   = '1.Ø1';      { Verzioszam       }
   c_o_scrol =    7;       { a legfelso sor   }
   c_u_scrol =   23;       { a legalso sor    }
   c_d_max   = 11ØØ;       { max. file szam   }
   c_f_max   =    7;       { max. funk. szam  }
   c_ch_max  =   18;       { max. kar. szam
                             MENU - ben       }
   c_zb_max  =   2Ø;       { max. kar. szam
                             ENTRY - ben      }
   c_tot     =  'Ø';       { Rejtett file     }
   c_nm      =  '.';       { Nem jelolt       }
   c_elval   =  '-';       { Elvalaszto jel   }
   fent      =  true;
   lent      =  false;

TYPE
   t_sor     = string[8Ø];          { File-nev tipus     }
   t_fun     = string[c_ch_max];    { Funkciok           }
   t_zb      = string[c_zb_max];    { File eleje         }
   t_valaszt = string[c_f_max];     { A funk. "maradeka"}
   t_menu    = array[Ø..c_f_max]    { Funkcio nev tabla  }
               of t_fun;

   t_Search_Status  = (uj,megegy,vege,telj);

   t_file   = record
                 attr    : byte;
                 ido,
                 datum,
                 meret_l,
                 meret_h  : integer;
                 nev     : array[1..13] of char;
              end;
   t_file_blk = record
                 res     : array[1..21] of byte;
                 entry   : t_file;
              end;

   t_d      = record              { A file tabla kepe }
                mark : char;
                de : t_file;
                at : string[6];
                gr : real;
                zb : t_zb;
              end;

CONST
   honap    : array[1..12] {Honap nevek}
               of string[5] =
                ('Jan','Feb','Mar','Apr','Maj','Jun',
                 'Jul','Aug','Sep','Okt','Nov','Dec');

   A_Valaszt   : t_valaszt = 'HBKUQE';   { A funkciok
                                           kezdobetui }
   A_Menu   : t_menu =
              ('Valasztas: ',          { A maradek betuk }
               'ajtsd vegre ',         { Vegrehajtas keres }
               'e file eleje ',
               'ov. file ',
               't  ',
               'uit  ',
               'lozo file ',
               '');

   B_Valaszt   : t_valaszt = 'AMDTAV';   { A funkciok
                                           kezdobetuje }
```

```
B_Menu    : t_menu =
          ('File :',                          { A tobbi betu }
           'ttributum  ',
           'ozgatas    ',
           'uplikalas  ',
           'orles      ',
           'tnevezes   ',
           'issza      ',
           '');

VAR
   i,                          { Ciklusvaltozo      }
   ii,                         { Segedvaltozo       }
   i_d,                        { File-tabla index   }
   i_d_max,                    { max. file-tabla index }
   IOr,                        { IOresult-ot tarolo    }
   y          : integer;
   ch         : char;
   d_zb       : boolean;
   osszeg,
   pmeret      : real;
   d          : array[1..c_d_max]       { File tabla    }
                 of t_d;
   ut,                                  { aktualis file }
   ut_uj,
   biztos_ami_biztos                     {  Ut kezdetben }
              : t_sor;

   r          : record
                   ax,bx,cx,dx,
                   bp,si,di,
                   ds,es,
                   flags          : integer;
                end;

   file_blk      : t_file_blk;
   Search_Status : t_Search_Status;

PROCEDURE Valaszt_Menu (posi : byte;
                        valaszt : t_valaszt;
                        menu : t_menu);
          { A VALASZTAS menu kiirasa }
   VAR
      kivalaszt : byte;

   BEGIN
      GotoXY( 1, posi);
      LowVideo;
      Write(menu[Ø],' ');
      i:=1;
      ii:=1;
      while i <= Length(valaszt)
       do begin
         ii:=ii+Length(menu[i])+1;
         if ii > 8Ø
          then begin
            GotoXY( 1, posi+1);
            Write(' ':Length(menu[Ø])+1);
          end;
         HighVideo;
         Write(valaszt[i]);              { Kezdobetuk   }
         LowVideo;
         Write(menu[i]);                 { Tovabbi betuk }
         i:=i+1;
       end;
   END; {.....................................}
```

```
PROCEDURE Scroll (fel  : boolean);

   BEGIN                    { A scroll terulet eltolasa }
      if fel
       then begin
         GotoXY( 1, c_o_scrol); InsLine;
       end
       else begin
         GotoXY( 1, c_o_scrol); DelLine;
         GotoXY( 1, c_u_scrol);
       end;
   END; {.....................................}

PROCEDURE F_attr(     f_fib : t_sor;    { file-jellemzo }
                      attr  : integer; { a feldolg.-hoz}
                  VAR altes : integer);
                               { be/visszaallitas      }

   BEGIN
      r.ax:=$4300;       { Az akt. jellemzo behozatala }
      r.ds:=seg(f_fib);
      r.dx:=ofs(f_fib[+1]);
      f_fib[Length(f_fib)+1]:=#0;     { ASCIZ keszites }
      MsDos(r);
      if (r.flags and $0001)<>0
       then altes:=$0000
       else altes:=r.cx;

      r.ax:=$4301;               { A 0 jellemzo beall. }
      r.ds:=seg(f_fib);
      r.dx:=ofs(f_fib[+1]);
      f_fib[Length(f_fib)+1]:=#0;
      r.cx:=attr;
      MsDos(r);
   END; {.....................................}

FUNCTION Keres_file (      ut   : t_sor;
                           attr  : integer;
                     VAR allom : t_file_blk): boolean;

   VAR
      Search_Ok : boolean;

   PROCEDURE Set_Disk_Transfer_Address;

      BEGIN
         r.ax:=$1A00;
         r.ds:=Seg(allom);
         r.dx:=Ofs(allom);
         MsDos(r);
      END;
BEGIN
   Search_Ok:=false;

   case Search_Status of
     uj:        begin
                    Set_Disk_Transfer_Address;
                    Fillchar(allom,SizeOf(allom),0);
                    allom.entry.nev:='           ';
                    ut[length(ut)+1]:=^@;
                    r.ax:=$4E00;
                    r.ds:=seg(ut);
                    r.dx:=ofs(ut[1]);
                    r.cx:=attr;
                    MsDos(r);
                    if (r.ax = 18)
```

```
                        then Search_Status:=vege
                        else if (r.ax = 2)
                         then Search_Status:=telj
                         else begin
                           Search_Status:=megegy;
                           Search_Ok:=true;
                         end;
                    end;
      megegy:  begin
                       allom.entry.nev:='
                       r.ax:=$4FØØ;
                       MsDos(r);
                       if (r.ax = 18)
                        then Search_Status:=vege
                        else begin
                          Search_Status:=megegy;
                          Search_Ok:=true;
                        end;
                    end;
      vege:;
      telj:;
    end;
    Keres_File:=Search_Ok;
END; {.....................................}

FUNCTION Kopizas (mit,                    { File masolas }
                  hova : t_sor): boolean;
    VAR
       len,
       regi_mit,
       regi_hova : integer;
       block     : array[1..maxint] of char;
       fb_mit,
       fb_hova   : file;

    BEGIN
       Assign(fb_mit,mit);
       F_attr(mit,Ø,regi_mit);
       {$I-} reset(fb_mit,1); {$I+}
       if IOresult=Ø
           then begin
             Assign(fb_hova,hova);
             F_attr(hova,Ø,regi_hova);
             Rewrite(fb_hova,1);
             while not eof(fb_mit)
              do begin
                 Blockread(fb_mit,block,maxint,len);
                 Blockwrite(fb_hova,block,len);
              end;
             Close(fb_mit);
             F_attr(mit,regi_mit,regi_mit);
             Close(fb_hova);
             F_attr(hova,regi_hova,regi_hova);
             Kopizas:=true;
           end
           else Kopizas:=false;
     END; {.....................................}

PROCEDURE Ir (nr : integer);        { File-sor irasa }

    BEGIN
       Write(d[nr].mark,' ',
             d[nr].de.nev,' ':15-Length(d[nr].de.nev),
             d[nr].at,' ',
             d[nr].de.datum and $ØØ1F       :2,'. ',
             honap[(d[nr].de.datum and $Ø1EØ) shr 5],' ',
             198Ø+(d[nr].de.datum shr 9)    ,' ',
```

```
          d[nr].de.ido shr 11          :2,':',
         (d[nr].de.ido and $Ø7EØ) shr 5:2,':',
          d[nr].de.ido and $ØØ1F       :2,
          d[nr].gr:11:Ø,'  «',
          d[nr].zb,'»');
END; {.....................................}

PROCEDURE Jelent (nr : integer); {Jelentes a file-okrol}

   BEGIN
      GotoXY(67, 1);
      Write(nr:5,' File');
   END; {.....................................}

PROCEDURE File_Eleje;  { A file elejenek a behozatala }

   VAR
      tat : integer;
      f : file;

   FUNCTION S(ssi: t_sor;
              i  : integer): t_zb;
      VAR
         si  : t_sor;
         szb : t_zb,

      BEGIN
         str(i:3,si);
         S:=ssi+si;
      END;

   BEGIN
      y:=c_ó_scrol;
      for i:=1 to i_d_max
       do begin
         Jelent(i);
         Assign(f,ut+d[i].de.nev);
         {$I-} reset(f,1); {$I+}
         IOr:=IOresult;
         if IOresult=Ø
          then begin
            {$I-} BlockRead(f,d[i].zb[1],2Ø,tat); {$I+}
            IOr:=IOresult;
            d[i].zb[Ø]:=chr(tat);
            if IOr<>Ø
             then begin
              d[i].zb:=S('Hiba:',IOr);
              d[i].mark:=c_tot;
             end
             else begin      { hibas kar.-ek elhagyasa }
               for ii:=1 to Length(d[i].zb)
                do if d[i].zb[ii] in [#Ø..#$1F,#$FF]
                     then d[i].zb[ii]:=c_nm;
             end;
          end
          else begin
            d[i].zb:=S('Hiba:',IOr);
            d[i].mark:=c_tot;
          end;;
         Close(f);
        if (i <= c_u_scrol-c_o_scrol+1)
         then begin
           GotoXY( 1, y);
           y:=y+1;
           Ir(i);
         end;
       end;
```

```
      y:=c_o_scrol;
      i_d:=1;
   END; {.....}

PROCEDURE Olv_Dir (ut : t_sor);

   VAR                    { A tartalomjegyzek olvasasa  }
      r_l,
      r_h : real;

   FUNCTION Attrx(x : byte): t_sor;

      VAR
         y : t_sor;

      BEGIN
        y:='ADVSHR';
        if (x and $01)=0 then y[6]:=c_nm;
        if (x and $02)=0 then y[5]:=c_nm;
        if (x and $04)=0 then y[4]:=c_nm;
        if (x and $08)=0 then y[3]:=c_nm;
        if (x and $10)=0 then y[2]:=c_nm;
        if (x and $20)=0 then y[1]:=c_nm;
        attrx:=y;
      END;
   BEGIN
      d_zb:=false;                { Nincs file          }
      FillChar(d,SizeOf(d),0); { File tabla torlese   }
      osszeg:=0;                  { Kar.szamlalo torlese }
      i_d:=0;                     { Direktori olvasasa   }
      Search_Status:=uj;          { DOS-Funkcio: $4E/$4F }
      while Keres_File(ut+'*.*',$003F,file_blk) and
            (i_d < c_d_max)
       do begin
         i_d:=i_d+1;
         d[i_d].de:=file_blk.entry;
         d[i_d].mark:=c_nm;
       end;
      i_d_max:=i_d;
      for i:=1 to i_d_max
       do begin
         d[i].at:=Attrx(d[i].de.attr);
         r_h:=d[i].de.meret_h;  { File meret szamitas }
         if r_h < 0
          then r_h:=65536.+r_h;
         r_l:=d[i].de.meret_l;
         if r_l < 0
          then r_l:=65536.+r_l;
         d[i].gr:=(65535.0*r_h)+r_l;
         osszeg:=osszeg+d[i].gr;
       end;
   END; {.....}


PROCEDURE Fej  (aktualis : boolean);    { Cim irasa   }

   BEGIN
      if not aktualis
       then
         ClrScr;

      GotoXY ( 1, 1);
      HighVideo;
      Write('FileDir: File-kezeles mas modon   ');
      LowVideo;
      Write(' - Verzio ',Verzio);
```

```
    if aktualis
     then begin
       Jelent(i_d_max);              { a file-ok szama }

       Valaszt_Menu(2,A_Valaszt,A_Menu);
       Valaszt_Menu(3,B_Valaszt,B_Menu);

       GotoXY( 1, 5); ClrEol;
       Writeln('Aktualis konyvtar: ',ut,
               '     tartalmaz ',
                osszeg:trunc(ln(osszeg+1)/ln(1Ø.Ø)):Ø,
               ' karaktert');
     end;

   GotoXY( 1, 6);
   HighVideo;
   for i:=1 to 8Ø
    do Write(c_elval);
   LowVideo;
END; {.....................................}


PROCEDURE Present;          { Az elso lap mejelenitese }

   BEGIN
      GetDir(Ø,ut);              { Melyik konyvtar?     }

      if not (ut[Length(ut)] in ['/','\'])
       then ut:=ut+'\';

      Fej(false);               { Fej sor funkcio nelkul }

      GotoXY( 1, 5); ClrEol;
      Write('A tartalomjegyzek olvasasa  ', ut);

      Olv_Dir(ut);               { File tabla feltoltese }
      Fej(true);                { Fej sor akt. verzioval }

      i:=1;                     { Az elso lap kiirasa    }
      y:=c_o_scrol;
      while (i <= c_u_scrol-c_o_scrol+1) and
            (i <= i_d_max)
       do begin
         GotoXY( 1, y);
         y:=y+1;
         Ir(i);
         i:=i+1;
       end;

      y:=c_o_scrol;
      i_d:=1;
   END; {.....................................}

PROCEDURE Vegrehajtas;

   VAR
      attr_beall : boolean;
      A_byte     : byte;
      be         : t_sor;
      f          : file;

   BEGIN
      Scroll(lent);
      Scroll(lent);

      for i_d:=1 to i_d_max
```

```
    do begin                  { File-tabla feldolgozas }
      if Pos(d[i_d].mark,B_Valaszt)<>Ø
       then begin
         ii:=Pos(d[i_d].mark,B_Valaszt);
         Scroll(lent);
         HighVideo;
         Write(B_Valaszt[ii]);
         LowVideo;
         Write(B_Menu[ii],' ',d[i_d].de.nev,' ');
         case ii of
          1: begin          { Jellemzo valtoztatasa }
               Write(' most  [',d[i_d].at,
                                '].  Uj [+¦-]? ');
               buflen:=12;
               Read(be);
               Attr_beall:=true;
               A_byte  :=d[i_d].de.attr;
               while length(be) > Ø
                do begin
                  case be[1] of
                   '+'     : Attr_beall:=true;
                   '-'     : Attr_beall:=false;
                   'R','r': if Attr_beall
                            then A_byte:=A_byte or  $Ø1
                            else A_byte:=A_byte and $FE;
                   'W','w': if Attr_beall
                            then A_byte:=A_byte and $FE
                            else A_byte:=A_byte or  $Ø1;
                   'H','h': if Attr_beall
                            then A_byte:=A_byte or  $Ø2
                            else A_byte:=A_byte and $FD;
                   'S','s': if Attr_beall
                            then A_byte:=A_byte or  $Ø4
                            else A_byte:=A_byte and $FB;
                   'A','a': if Attr_beall
                            then A_byte:=A_byte or  $2Ø
                            else A_byte:=A_byte and $DF;
                  end;
               Delete(be,1,1);
              end;
             r.ax:=$43Ø1;       { Attr. beallitasa }
             be:=ut+d[i_d].de.nev+#Ø;
             r.ds:=Seg(be);
             r.dx:=Ofs(be[1]);
             r.cx:=A_byte;
             MsDos(r);
             if (r.flags and $ØØØ1) <> Ø
              then begin
                Scroll(lent);
                Write(' Hiba: ',be,' Attr=',
                      a_byte,' RC=',r.ax);
              end;
         end;
      end;
    2: begin                    { Mozgatas      }
         IOr:=1;
         while IOr<>Ø
          do begin
            Write(' hova? ');
            buflen:=64;
            Read(be);
            if Kopizas(ut+d[i].de.nev,be)
             then begin
               IOr:=Ø;
               Assign(f,ut+d[i_d].de.nev);
               {$I-} Erase(f); {$I+}
               if IOresult<>Ø
                then
```

```
                        Write(' Torles sikertelen'^G);
                      end
                      else begin
                        Writeln(' Kopizas sikertelen'^G);
                        Scroll(lent);
                      end;
                    end;

              end;
            3: begin                        { Kopizas       }
                 IOr:=1;
                 while IOr<>Ø
                  do begin
                    Write(' hova? ');
                    buflen:=64;
                    Read(be);
                    if Kopizas(ut+d[i].de.nev,be)
                     then IOr:=Ø
                     else begin
                       Write(' sikertelen'^G);
                       Scroll(lent);
                     end;
                  end;
               end;

            4: begin                        { Torles        }
                 Assign(f,ut+d[i_d].de.nev);
                 {$I-} Erase(f); {$I+}
                 if IOresult<>Ø
                  then
                    Write(' sikertelen'^G);
               end;
            5: begin                        { Atnevezes     }
                 IOr:=1;
                 while IOr<>Ø
                  do begin
                    Write(' mire? ');
                    buflen:=11;
                    Read(be);
                    Assign(f,ut+d[i_d].de.nev);
                    {$I-} Rename(f,be); {$I+}
                    IOr:=IOresult;
                    if IOr<>Ø
                     then begin
                       Write(' sikertelen'^G);
                       Scroll(lent);
                     end;
                  end;
               end;
          end;
        end;
     end;
   Scroll(lent);
   Scroll(lent);
   Write('Vegrehajtas befejezodott ');
   Delay(3ØØØ);
   Present;
END; {.....................................}


BEGIN
   GetDir(Ø,biztos_ami_biztos);
                        { Az akt. konyvtar behozatala  }
   osszeg:=Ø;
   ut:=ParamStr(1);     { A parancssorbeli utasitas    }
   {$I-} ChDir(ut); {$I+}     { beallitasa }
```

```
Present;

repeat                          { Adatbeviteli ciklus  }
   GotoXY( 1, y);
   Read(kbd,ch);                { Egy betus parancsok  }
   ch:=UpCase(ch);      { vagy utasitasok olvasasa  }


   if (Pos(ch,B_Valaszt)<>Ø)  { Ismert betu-jel     }
    then begin
      if (d[i_d].mark<>c_tot)
       then begin
         if (ch='V')  { Vissza az elozo betu-jelre }
          then begin
            d[i_d].mark:=c_nm;
            Write(c_nm);
          end
          else begin
            d[i_d].mark:=ch;     { Betu-jel beirasa }
            Write(ch);
          end;
       end;
      ch:='K';
    end;

   if ch='K'                    { Ugras a kov. file-ra  }
    then begin
      i_d:=i_d+1;
      if i_d <= i_d_max
       then begin
         if y < c_u_scrol
          then begin
            y:=y+1;
          end
          else begin
            Scroll(lent);
            Ir(i_d);                  { File sor irasa }
          end;
       end
       else i_d:=i_d_max;
      goto 1;
    end;


   if ch='E'                   { Ugras az elozo file-ra }
    then begin
      i_d:=i_d-1;
      if i_d > Ø
       then begin
         if y > c_o_scrol
          then begin
            y:=y-1;
          end
          else begin
            Scroll(fent);
            Ir(i_d);
          end;
       end
       else i_d:=1;
      goto 1;
    end;

   if ch='U'     { A konyvtart meg kell valtoztatni }
    then begin
      if d[i_d].at[2]='D' { CURrentSOuRce alkonyvt.}
       then begin
         ut_uj:=ut+d[i_d].de.nev;
```

```
    end
    else begin                { CURSOR nem az
                                alkonyvtarnal all }
      GotoXY( 1, 5); ClrEol;
      Write('Uj konyvtar ?');
      buflen:=65;
      Readln(ut_uj);
    end;

  {$I-} ChDir(ut_uj); {$I+}{ Konyvtar beallit. }
  if IOresult = 0
   then begin
     Present;
   end
   else begin          { A regi konyvtar marad  }
     GotoXY(60, 5); Write(^G'NEM talalhato  !');
     Delay(1500);
     GotoXY( 1, 5); ClrEol;
     Writeln('Aktualis Konyvtar: ',ut);
   end;
  goto 1;
end;

      if ch='B'             { File elejenek behozatala }
       then
         File_Eleje;

      if ch='H'                    { Vegrehajtas keres }
       then
         Vegrehajtas;
1:
   until ch='Q';                        { A program vege }

   ChDir(biztos_ami_biztos);{ Konyvtar visszaallitasa }

   ClrScr;                               { Koszones     }
   HighVideo;
   Write('A FileDir program vege ');
   LowVideo;
   Write(' - Verzio ',Verzio);         { Udvozlet       }
   Writeln;
END. {.....................................................}
```

4.2. FILEDUMP

A FileDump programpélda – a *Turbo–Pascal* felhasználásával – lehetővé teszi egy file hexadecimális és karakteres ábrázolását a képernyőn.
A program bemutatja, hogy melyik funkcióbillentyűk hatásosak. Az oldalankénti lapozás ugyanúgy lehetséges, mint a file végére vagy elejére ugrás.

Ez a program bemutatja egy *DOS*-file olvasását és megjelenítését, a bináris értékek átalakítását hexadecimális ábrázolásra és a közvetlenül nem ábrázolható karakterek elkészítését.

```
{ Filedump.pas }
{ Pr. II. 4.2. }

PROGRAM Filedump;

{ Ez a program bemutatja egy DOS file "olvasasat" es
  megjeleniteset. A binaris ertekek hexadecimalisan es
  karakteres abrazolassal is megjelennek. }

{$U-,C-}

CONST
   Verzio     = '1.Ø1';
   c_o_scrol  =     5;      { A scroll-terulet teteje }
   c_u_scrol  =    24;      { A scroll-terulet alja   }
   fent       =  true;
   lent       = false;
   c_fb_max_1=   319;    {(c_u_scrol-c_o_scrol+1)*16-1 }
   c_fb_max   =   32Ø;    { (c_u_scrol-c_o_scrol+1)*16  }
   c_ertek    =   '!';    { Fuggoleges elvalaszto jel   }

TYPE
   t_z        = string[45];
   t_sor      = string[79];
   puffer     = array[Ø..c_fb_max_1] of byte;

CONST
Sor_11:t_z='+----------------------+---------------';
Sor_12:t_z='----------------+----------------------+';
Text_Ø1:t_z='! File Dump   1.Ø1       ! File : _______';
Text_Ø2:t_z='_________________ ! Pozicio : __________ !';
Text_11:t_z='!  1=Vege  (__________) !  2=Lap vissza ';
Text_12:t_z='   3=Lap elore !  4=Kezdet  5=FileVege!';
Sor_21:t_z='+----------------------+---------------';
Sor_22:t_z='----------------+----------------------+';

VAR
   attrib,
   i,
   relpos,
   tat,
   sor        : integer;
   bi         : byte;
   ch         : char;
   f          : file;
   fn         : t_sor;
   fb         : puffer;
   f_pos,
   f_pos_max  : real;
```

```
PROCEDURE Hiba (ErrNo, ErrOfs : Integer);

   BEGIN
      GotoXY( 1,c_o_scrol);
      DelLine; DelLine; DelLine; DelLine;
      GotoXY( 1,c_u_scrol-2);
      write('* Megszakitas *'^G);
      Halt;
   END;

FUNCTION Olv_ch : char;  { Olvasas a billentyuzetrol }

   VAR
      ch : char;
      r  : record
              ax,bx,cx,dx,
              bp,si,di,
              ds,es,
              flags         : integer;
           end;

   BEGIN
      r.ax:=$Ø6ØØ;
      r.dx:=$ØØFF;
      MsDos(r);
      if Lo(r.dx)=$FF
       then ch:=Chr(Lo(r.ax))
       else ch:=#$F8;
      if ch=^C
       then Halt;
      Olv_ch:=ch;
   END; {.....................................}

PROCEDURE Scrol (fel  : boolean);
                          { A scroll terulet eltolasa }
   BEGIN
      if fel
       then begin
         GotoXY( 1, c_o_scrol); InsLine;
       end
       else begin
         GotoXY( 1, c_o_scrol); DelLine;
         GotoXY( 1, c_u_scrol);
       end;
   END; {.....................................}


FUNCTION H (fb_b : byte): t_sor;

   CONST
      t : string[16] = 'Ø123456789ABCDEF';

   VAR
      i : integer;
      x : t_sor;

   BEGIN
      x:='__';
      x[1]:=t[((fb_b and $FØ) shr 4) +1];
      x[2]:=t[((fb_b and $ØF)      ) +1];
      h:=x;
   END;

PROCEDURE F_attr(     f_fib : t_sor;
                      attr  : integer;
                  VAR regi : integer);
```

```
VAR
   r  : record
           ax,bx,cx,dx,
           bp,si,di,
           ds,es,
           flags          : integer;
     end;

BEGIN
   r.ax:=$4300;
   r.ds:=Seg(f_fib);
   r.dx:=Ofs(f_fib[+1]);
   f_fib[Length(f_fib)+1]:=#0;
   MsDos(r);
   if (r.flags and $0001)<>0
    then
      regi:=$0000
    else
      regi:=r.cx;

   r.ax:=$4301;
   r.ds:=Seg(f_fib);
   r.dx:=Ofs(f_fib[+1]);
   f_fib[Length(f_fib)+1]:=#0;
   r.cx:=attr;
   MsDos(r);
   if (r.flags and $0001)<>0
    then
      Write('!');
END; {.....}

PROCEDURE Olv (      fn     : t_sor;  { A file olvasasa }
                     f_pos  : real;
                 VAR fb     : puffer;
                     len    : integer;
                 VAR tat    : integer);

   VAR
      f        : file;

   BEGIN
      Assign(f,fn);
      F_attr(fn,0,attrib);
      {$I-} Reset(f,1);{$I+};
      if IOresult = 0
       then begin
         LongSeek(f,f_pos);
         BlockRead(f,fb,len,tat);
       end
       else begin
         FillChar(fb,len,0);
         tat:=0;
       end;
      Close(f);
      F_attr(fn,attrib,attrib);
   END; {.....}

PROCEDURE Mut_Sor (f_pos : real; sor : integer);
   VAR
      i,
      j : integer;

   BEGIN
      LowVideo;
      GotoXY( 1,sor);
      j:=(sor-c_o_scrol)*16;
      Write(f_pos+j:10:0,'      ');
```

```
      for i:=Ø to 15
       do begin
        if fb[j+i] in [Ø..$1F]
         then begin
           HighVideo;
           Write(h(fb[j+i]));
           LowVideo;
         end
         else begin
           Write(h(fb[j+i]));
         end;
        if i mod 4 = 3 then Write(' ');
        if i mod 8 = 7 then Write(' ');
       end;

      Write(' ',c_ertek,' ');

      for i:=Ø to 15
       do begin
        if fb[j+i] in [Ø..$1F]
         then begin
           HighVideo;
           Write(Chr($4Ø+fb[j+i]));
           LowVideo;
         end
         else begin
           if fb[j+i] in [$2Ø..$7E,$8Ø..$FE]
            then Write(Chr(fb[j+i]))
            else Write(#$F9);
         end;
       end;

      Write(' ',c_ertek);
   END; {.....}

PROCEDURE Uj_Poz (lepes: real);

   BEGIN
      if (f_pos+lepes < Ø)
       then f_pos:=Ø.Ø
       else if (f_pos+lepes >= f_pos_max)
        then f_pos:=f_pos_max-c_fb_max
        else f_pos:=f_pos+lepes;

      GotoXY(68, 2);
      Write(f_pos:1Ø:Ø);

      Olv(fn,f_pos,fb,c_fb_max,tat);

      for i:=c_o_scrol to c_u_scrol
       do begin
         Mut_Sor(f_pos,i);
       end;
   END; {.....}

BEGIN
   ClrScr;
   fn:=ParamStr(1);
   Assign(f,fn);
   F_attr(fn,Ø,attrib);
   {$I-} Reset(f,1);{$I+}
   if IOresult = Ø
    then begin
      f_pos:=LongFilePos(f);
      f_pos_max:=LongFileSize(f);
      Close(f);
      F_attr(fn,attrib,attrib);
```

```
        HighVideo;
        Write(sor_11); WriteLn(Sor_12);
        Write(Text_Ø1); WriteLn(Text_Ø2);
        Write(Text_11); WriteLn(Text_12);
        Write(sor_21); WriteLn(Sor_22);
        LowVideo;
        GotoXY(34, 2);
        Write(fn);
        GotoXY(68, 2);
        Write(f_pos:1Ø:Ø);
        GotoXY(13, 3);
        Write(f_pos_max:1Ø:Ø);
      end
      else
        Halt;

    Uj_Poz(Ø);

    repeat
       ch:=Olv_ch;

       case ch of
            '2' : Uj_Poz(-c_fb_max);
            '3' : Uj_Poz(+c_fb_max);
            '4' : Uj_Poz(-f_pos_max);
            '5' : Uj_Poz(f_pos_max);
       end;
    until ch='1';

    Scrol(lent);
    Scrol(lent);
    Scrol(lent);
    GotoXY( 1, 22);
END.
```

4.3. I/O EGYSÉG

Az I/O egység programpélda – a *Turbo–Pascal* felhasználásával – lehetővé teszi a *DOS* egység meghajtóinak felkutatását.

A program a $44 IO Control *DOS*-funkciót használja, amellyel a DOS-adatcsatorna állapota, készenléte és összeköttetési lehetősége letapogatható.

```
{ EgysEll.pas  }
{ Pr. II. 4.3. }

PROGRAM Egyseg_Ell;

{ Ez a program a $44 DOS funkciok hasznalatat mutatja.
  A Ø, 6 es 7 kodok hasznalata egyszeru.
  A 2 es 3, ill. 4 es 5 -os funkciok csak akkor
  hasznalhatok, ha a megfelelo periferiak rendelkezesre
  allnak. }

{ Az I/O egyseg ellenorzese }

CONST
   verzio = '1.Ø1';

TYPE
   t_sor = string[8Ø];

VAR
   r        :  record
                  ax,bx,cx,dx,
                  bp,si,di,
                  ds,es,
                  flags          : integer;
               end;
   fc,                           {Funkcio kod  :        Ø..7}
   handle,                       {File/egyseg-kezeles}
   i,
   ior,                          {IOresult tarolasara}
   ix,
   j        : integer;
   buf      : array [1..maxint] of char;


PROCEDURE IO_Control (function_code : byte;
                      handle,
                      buf_len       : integer);
   VAR
      lw_nr    : integer absolute handle;

   CONST
      fc_nev : array [Ø..7] of string[4Ø] =
                    (' I/O egyseg rendben          ?',
                     ' I/O egysegdef. beallitasa    ',
                     ' Olv  CX kar    <handle> - bol',
                     ' Ir   CX kar    <handle> - ba ',
                     ' Olv  CX kar    meghajto - bol',
                     ' Ir   CX kar    meghajto - ba ',
                     ' INPUT lehetoseg vizsg.       ',
                     ' OUTPUT lehetoseg vizs.       ');
```

```
BEGIN
   if function_code in [Ø..7]
    then if function_code in [Ø..3,6..7]
     then Writeln('IO_Ellenorzes (handle = ',handle,
                  ', ',fc_nev[function_code],')')
     else Writeln('IO_Ellenorzes (Meghajto = ',
                  Chr(Ord('A')+lw_nr),', ',
                  fc_nev[function_code],')')
    else exit;

   if (function_code in [2..3]) and (buf_len > Ø)
    then begin
      r.ax:=$44ØØ or function_code;
      r.bx:=handle;
      r.ds:=Seg(buf);
      r.dx:=Ofs(buf);
      r.cx:=buf_len;
      MsDos(r);
    end
    else if(function_code in [4..5]) and (buf_len>Ø)
     then begin
      r.ax:=$44ØØ or function_code;
      r.bx:=lw_nr;
      r.ds:=Seg(buf);
      r.dx:=Ofs(buf);
      r.cx:=buf_len;
      MsDos(r);
    end
    else if (function_code in [Ø,6..7])
     then begin
      r.ax:=$44ØØ or function_code;
      r.bx:=lw_nr;
      r.ds:=Seg(buf);
      r.dx:=Ofs(buf);
      r.cx:=Ø;
      MsDos(r);
     end;

   if (r.flags and $ØØØ1)<>Ø
    then begin
      write('DOS $44: HIBA    - ');
      case r.ax of
         1 : Writeln('hibas funkcio - (Ø..7)');
         5 : Writeln('eleres nem megengedett');
         6 : Writeln('hibas handle');
        13 : Writeln('hibas adat');
        else Writeln(r.ax,' ???');
      end;
    end
    else begin
      case function_code of
        Ø,1 : begin
                if (r.dx and $ØØ8Ø) <> Ø
                 then begin
                   Writeln;
                   Writeln('I/O egyseg vizsg.   :');
                   Writeln;
                   if (r.dx and $ØØ4Ø) <> Ø
                    then Writeln('EOF elott, ')
                    else Writeln('EOF utan, ');
                   if (r.dx and $ØØ2Ø) <> Ø
                    then Writeln('BIN-file, ')
                    else Writeln('TEXT-file, ');
                   if (r.dx and $ØØ1Ø) <> Ø
                    then Writeln('SPECIAL, ');
                   if (r.dx and $ØØØ8) <> Ø
                    then Writeln('van CLOCK, ');
```

```
      if (r.dx and $0004) <> 0
       then Writeln('van NUL, ');
      if (r.dx and $0002) <> 0
       then Writeln('van ConOUT, ');
      if (r.dx and $0001) <> 0
       then Writeln('van ConIN, ');
      if (r.dx and $4000) <> 0
       then Writeln
            ('CTRL (FC 2..3 lehet), ')
       else Writeln('CTRL(FC 2..3',
                    ' nem lehet)');
    end
    else begin
      if (r.dx and $4000) <> 0
       then Writeln('EOF van , ')
       else Writeln('EOF hianyzik');;
      Writeln(' Egyseg:',
              (r.dx and $1F00) shr 8);
    end;
  Writeln;
end;


        2..5: begin
                j:=1;
                while j <= buf_len
                 do begin
                   if buf[j] in [' '..#$FE]
                    then Write(buf[j])
                    else Write('<',Ord(buf[j]),'>');
                   j:=j+1;
                  end;
              end;

          6 : begin
                case r.ax of
                    0 : Writeln('IN nem mukodik');
                  $FF : Writeln('IN mukodik');
                end;
              end;
          7 : begin
                case r.ax of
                    0 : Writeln('OUT nem mukodik');
                  $FF : Writeln('OUT mukodik');
                end;
              end;
      end;
    end;
END; {.....}


BEGIN
   ClrScr;
   HighVideo;
   Writeln ('I/O egysegek a rendszerben');
   Writeln ('--------------------------');
   Writeln;
   fc:=0;
   while fc in [0..7]
    do begin
      LowVideo;
      Write('Valassza ki a funkciot (0..7, 8=Vege): ');
      HighVideo;
      repeat
         Buflen:=1;
         Read(fc);
      until fc in [0..8];
```

```
        if fc < 8
         then begin
           Writeln;
           LowVideo;
           Write('Handle/Meghajtoszam (Ø..63): ');
           HighVideo;
           repeat
              Buflen:=2;
              Read(handle);
           until handle in [Ø..63];

           if fc in [2..5]
            then begin
              Writeln;
              Lowvideo;
              Write('Puffermeret  (Ø..32767): ');
              HighVideo;
              repeat
                 Buflen:=5;
                 Read(i);
              until i in [Ø..maxint];
            end;

           if fc in [3,5]
            then begin
              Writeln;
              LowVideo;
              Write('Puffertartalom (Vege Enter-rel): ');
              HighVideo;
              ix:=Ø;
              repeat
                 ix:=ix+1;
                 Read(kbd,buf[ix]);
                 if buf[ix] in [' '..#$FE]
                  then Write(buf[ix])
                  else Write('<',Ord(buf[ix]),'>');
              until (ix >= i) or (buf[ix] in [^M]);
            end;

            Writeln;
            Writeln;

            { Funkcio hivas  }
            IO_Control(fc,handle,i);

          end;
         Writeln;
       end;
END.
```

FÜGGELÉK

1. A 8088/8086-os gépi utasítás

Az *Intel Corp. (USA)* 8088/8086-os mikroprocesszorának gépi kódja.

A rövidítések magyarázata:

m . . . tároló
mr . . . tároló/regiszter
r . . . regiszter
mra . . . regisztercímzési mód (mm rrr aaa)
kk . . . byte–konstans
jjkk . . . szókonstans
ppqq . . . mutató

Címzési módok: *(mm rrr aaa)*

Mód = 00 az operandus a tárolóban,
az *aaa* megadja a címképzést,
nincs displacement

Mód = 01 az operandus a tárolóban,
az *aaa* megadja a címképzést,
dl – 128 . . . +127
Utasításforma: utasításkód *mra dl*

Mód = 10 az operandus a tárolóban,
az *aaa* megadja a címképzést,
dl dh (ofs (var))
Utasításforma: utasításkód *mra dl dh*

Mód = 11: az operandus a regiszterben,
(*w* a 0-dik bit az utasításkódban)
az *rrr* és *w* adja a regisztert,
az *aaa* megadja a címképzést,
Utasításforma: utasításkód *mra* vagy
Utasításforma: utasításkód *m110 dl dh*

A regiszterek kódolása:

rrr	*w*= 0	w=1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

A címzés kódolása:

aaa	mód=00	mód=01	mód=10	mód=11	
				w=0	*w*=1
000	BX+SI	BX+SI+disp	BX+SI+disp	AL	AX
001	BX+DI	BX+DI+disp	BX+DI+disp	CL	CX
010	BP+SI	BP+SI+disp	BP+SI+disp	DL	DX
011	BP+DI	BP+DI+disp	BP+DI+disp	BL	BX
100	SI	SI+disp	SI+disp	AH	SP
101	DI	DI+disp	DI+disp	CH	BP
110	d.adr	BP+disp	BP+disp	DH	SI
111	BX	BX+disp	BX+disp	BH	DI

A műveleti kódok áttekintése (hex):

AAA	37	ASCII adjust for addition
AAA	D5 0A	ASCII adjust for division
AAM	D4 0A	ASCII adjust for multiplication
AAS	3F	ASCII adjust for subtraction
ADC	14 *kk*	Add with carry AL,*kk*
ADC	15 *kk jj*	Add with carry AX,*jjkk*
ADC	81 *m010a dl dh jj kk*	Add with carry mr, *jjkk*
ADC	83 *m010a dl dh jj kk*	Add with carry *mr, jjkk* word
ADC	80 *m010a dl dh kk*	Add with carry *mr, kk*
ADC	82 *m010a dl dh kk*	Add with carry *mr,kk* byte
ADC	10 *mra dl dh*	Add with carry *mr,r* byte
ADC	11 *mra dl dh*	Add with carry *mr,r* word
ADC	12 *mra dl dh*	Add with carry *r,mr* byte
ADC	13 *mra dl dh*	Add with carry *r,mr* word
ADD	04 *kk*	Add AL,*kk*
ADD	05 *kk jj*	Add AX,*jjkk*
ADD	81 *m000a dl dh jj kk*	Add *mr, jjkk*
ADD	83 *m000a dl dh jj kk*	Add *mr, jjkk* word
ADD	80 *m000a dl dh kk*	Add *mr,kk*
ADD	82 *m000a dl dh kk*	Add *mr,kk* byte
ADD	00 *mra dl dh*	Add *mr,r* byte
ADD	01 *mra dl dh*	Add *mr,r* word
ADD	02 *mra dl dh*	Add *r,mr* byte
ADD	03 *mra dl dh*	Add *r,mr* word
AND	24 *kk*	AND AL,*kk*
AND	25 *kk jj*	AND AX, *jjkk*
AND	81 *m100a dl dh jj kk*	And *mr, jjkk*
AND	80 *m001a dl dh kk*	And *mr,kk*
AND	20 *mra dl dh*	AND *mr,r* byte
AND	21 *mra dl dh*	AND *mr,r* word
AND	22 *mra dl dh*	AND *r,mr* byte
AND	23 *mra dl dh*	AND *r,mr* word
CALL	9A *kk jj hh gg*	CALL ⟨far pointer⟩
CALL	E8 *dl dh*	CALL disp 16
CALL	FF *m011a dl dh*	Call *m*
CALL	FF *m010a dl dh*	Call *mr*
CBW	99	Convert byte to word
CLC	F8	Clear carry flag
CLD	FC	Clear direction flag
CLI	FA	Clear interrupt flag
CMC	F5	Complement carry flag
CMP	3C *kk*	Compare AL,*kk*
CMP	3D *kk jj*	Compare AX,*jjkk*
CMP	81 *m111a dl dh jj kk*	Compare *mr, jjkk*
CMP	83 *m111a dl dh jj kk*	Compare *mr, jjkk* word
CMP	80 *m111a dl dh kk*	Compare *mr,kk*
CMP	82 *m111a dl dh kk*	Compare *mr,kk* byte
CMP	38 *mra dl dh*	Compare *mr,r* byte
CMP	39 *mra dl dh*	Compare *mr,r* word
CMP	3A *mra dl dh*	Compare *r,mr* byte

CMP	3B *mra dl dh*	Compare *r,mr* word
CMPSB	A6	Compare byte string
CMPSW	A7	Compare word string
CWB	98	Convert byte to word
CWD	99	Convert word to double word
DAA	27	Decimal adjust for addition
DAS	2F	Decimal adjust for subtraction
DEC	48	Decrement AX
DEC	4D	Decrement BP
DEC	4B	Decrement BX
DEC	49	Decrement CX
DEC	4A	Decrement DX
DEC	FE *m001a dl dh*	Decrement *mr* byte
DEC	FF *m001a dl dh*	Decrement *mr* word
DEC	4E	Decrement SI
DEC	4F	Decrement SI
DEC	4C	Decrement SP
DIV	F7 *m110a dl dh kk jj*	Divide *mr,* jjkk
DIV	F6 *m110a dl dh kk*	Divide *mr,* kk
ESC	D8 *mra dl dh*	Escape *mr*
ESC	D9 *mra dl dh*	Escape *mr*
ESC	DA *mra dl dh*	Escape *mr*
ESC	DB *mra dl dh*	Escape *mr*
ESC	DC *mra dl dh*	Escape *mr*
ESC	DD *mra dl dh*	Escape *mr*
ESC	DE *mra dl dh*	Escape *mr*
ESC	DF *mra dl dh*	Escape *mr*
HLT	F4	Halt
IDIV	F7 *m111a dl dh kk jj*	Integer Divide *mr, jjkk*
IDIV	F6 *m111a dl dh kk*	Integer Divide *mr, kk*
IMUL	F7 *m101a dl dh kk jj*	Integer Multiply *mr,jjkk*
IMUL	F6 *m101a dl dh kk*	Integer Multiply *mr,kk*
IN	EC	IN AL,DX
IN	E4 *kk*	IN AL,*kk*
IN	ED	IN AX,DX
IN	E5 *kk*	IN AX, *kk*
INC	40	Increment AX
INC	45	Increment BP
INC	43	Increment BX
INC	41	Increment CX
INC	47	Increment DI
INC	42	Increment DX
INC	FE *m000a dl dh*	Increment *mr* byte
INC	FF *m000a dl dh*	Increment *mr* word
INC	46	Increment SI
INC	44	Increment SP
INT	CD	Interrupt (nr)
INT	CC	Interrupt 3
INTO	CE	Interrupt on overflow
IRET	CF	Interrupt return
JA	77 *dl*	Jump on above
JAE	73 *dl*	Jump on above or equal

JB	72 *dl*	Jump on below
JBE	76 *dl*	Jump on below or equal
JC	72 *dl*	Jump on carry
JCXZ	E3 *dl*	Jump on CX zero
JE	74 *dl*	Jump on equal
JG	7F *dl*	Jump on greater
JGE	7D *dl*	Jump on greater or equal
JL	7C *dl*	Jump on less than
JLE	7E *dl*	Jump on less than or equal
JMP	E9 *dj dh*	JMP disp 16
JMP	EA *kk jj hh gg*	Jump ⟨addr⟩
JMP	EB *dl*	Jump ⟨disp⟩
JMP	FF *m101a dl dh*	Jump *mr*
JMP	FF *m100a dl dh*	Jump *mr*
JNA	76 *dl*	Jump on not above or
JNAE	72 *dl*	Jump on not above or equal
JNB	73 *dl*	Jump on not below
JNBE	77 *dl*	Jump on not below or equal
JNC	73 *dl*	Jump on no carry
JNE	75 *dl*	Jump on not equal
JNG	7E *dl*	Jump on not greater
JNGE	7C *dl*	Jump on not greater or equal
JNL	7D *dl*	Jump on not less than
JNLE	7F *dl*	Jump on not less than or equal
JNO	71 *dl*	Jump on not overflow
JNP	7B *dl*	Jump on not parity
JNS	79 *dl*	Jump on not sign
JNZ	75 *dl*	Jump on not zero
JO	70 *dl*	Jump on overflow
JP	7A *dl*	Jump on parity
JPE	7A *dl*	Jump on parity even
JPO	7B *dl*	Jump on parity odd
JS	78 *dl*	Jump on sign
JZ	74 *dl*	Jump on zero
LAHF	9F	Load AH with flags
LDS	C5	Load pointer into DS
LEA	8D *mra dl dh*	Load effective addressreg, addr
LES	C4	Load pointer into ES
LOCK	F0	LOCK bus
LODSB	AC	Load byte (string)
LODSW	AD	Load word (string)
LOOP	E2 *dl*	LOOP
LOOPE	E1 *dl*	LOOP while equal
LOOPNE	E0 *dl*	LOOP while not equal
LOOPNZ	E0 *dl*	LOOP while not zero
LOOPZ	E1 *dl*	LOOP while zero
MOV	A2 *qq pp*	Move addr,AL
MOV	A3 *qq pp*	Move addr,AX
MOV	B4 *kk*	Move AH,*kk*
MOV	A0 *qq pp*	Move AL,addr

MOV	B0 *kk*	Move AL,*kk*
MOV	A1 *qq pp*	Move AX,addr
MOV	B8 *kk jj*	Move AX,*jjkk*
MOV	B7 *kk*	Move BH,*kk*
MOV	B3 *kk*	Move BL,*kk*
MOV	BD *kk jj*	Move BP,*jjkk*
MOV	BB *kk jj*	Move BX,*jjkk*
MOV	B5 *kk*	Move CH,*kk*
MOV	B1 *kk*	Move CL,*kk*
MOV	B9 *kk jj*	Move CX,*jjkk*
MOV	B6 *kk*	Move DH,*kk*
MOV	BF *kk jj*	Move DI,*jjkk*
MOV	B2 *kk*	Move DL,*kk*
MOV	BA *kk jj*	Move DX,*jjkk*
MOV	C7 *m000a*	Move m,*jjkk*
MOV	C6 *m000a*	Move m,*kk*
MOV	88 *mra dl dh*	Move *mr,r* byte
MOV	89 *mra dl dh*	Move *mr,r* word
MOV	8C *m0SSa dl dh*	Move *mr,* segred
MOV	8A *mra dl dh*	Move *r,mr* byte
MOV	8B *mra dl dh*	Move *r,mr* word
MOV	BE *kk jj*	Move SI,*jjkk*
MOV	BC *kk jj*	Move SP,*jjkk*
MOVS	A4	Move byte string
MOVS	A5	Move word string
MUL	F7 *m100a dl dh kk jj*	Multiply *mr, jjkk*
MUL	F6 *m100a dl dh kk*	Multiply *mr,kk*
NEG	F7 *m011a dl dh kk jj*	NEG *mr, jjkk*
NEG	F6 *m011a dl dh kk*	NEG *mr,kk*
NOP	90	No Operation
NOT	F7 *m010a dl dh kk jj*	NOT *mr, jjkk*
NOT	F6 *m010a dl dh kk*	NOT *mr,kk*
OR	0C *kk*	Or AL,*kk*
OR	0D *kk jj*	Or AX,*jjkk*
OR	81 *m010a dl dh jj kk*	Or *mr, jjkk*
OR	80 *m010a dl dh kk*	Or *mr,kk*
OR	08 *mra dl dh*	Or *mr,r* byte
OR	09 *mra dl dh*	Or *mr,r* word
OR	0A *mra dl dh*	Or *r,mr* byte
OR	0B *mra dl dh*	Or *r,mr* word
OUT	E6 *kk*	OUT AL,*kk*
OUT	E7 *kk*	OUT AL,*kk*
OUT	EE	OUT DX,AL
OUT	EF	OUT DX,AX
POP	58	Pop AX
POP	5D	Pop BP
POP	5B	Pop BX
POP	59	Pop CX
POP	5F	Pop DI
POP	1F	Pop DS
POP	5A	Pop DX
POP	07	Pop ES

POP	8F	*m000a dl dh*	Pop *mr*
POP	5E		Pop SI
POP	5C		Pop SP
POP	17		Pop SS
POPF	9D		Pop Flags
PUSH	50		Push AX
PUSH	55		Push BP
PUSH	53		Push BX
PUSH	OE		Push CS
PUSH	51		Push CX
PUSH	57		Push DI
PUSH	1E		Push DS
PUSH	52		Push DX
PUSH	06		Push ES
PUSH	FF	*m110a dl dh*	Push *m*
PUSH	56		Push SI
PUSH	54		Push SP
PUSH	16		Push SS
PUSHF	9C		Push Flags
RCL	D0	*m010a dl dh*	Rotate thru Carry Left *mr*, 1 byte
RCL	D1	*m010a dl dh*	Rotate thru Carry Left *mr*, 1 word
RCL	D2	*m010a dl dh*	Rotate thru Carry Left *mr*, CL byte
RCL	D3	*m010a dl dh*	Rotate thru Carry Left *mr*, CL word
RCR	D0	*m011a dl dh*	Rotate thru Carry Right *mr*, 1 byte
RCR	D1	*m011a dl dh*	Rotate thru Carry Right *mr*, 1 word
RCR	D2	*m011a dl dh*	Rotate thru Carry Right *mr*, CL byte
RCR	D3	*m011a dl dh*	Rotate thru Carry Right *mr*, CL word
REP	F3		Repeat
REPE	F3		Repeat on equal
REPNE	F2		Repeat on not equal
REPNZ	F2		Repeat on not zero
REPZ	F3		Repeat on zero
RET	CB		Return
RET	C3		Return
RET	C2	*kk jj*	Return *jjkk*
RET	CA	*kk jj*	Return *jjkk*
ROL	D0	*m000a dl dh*	Rotate Left *mr*, 1 byte
ROL	D1	*m000a dl dh*	Rotate Left *mr*, 1 word
ROL	D2	*m000a dl dh*	Rotate Left *mr*, CL byte
ROL	D3	*m000a dl dh*	Rotate Left *mr*, CL word
ROR	D0	*m001a dl dh*	Rotate Right *mr*, 1 byte
ROR	D1	*m001a dl dh*	Rotate Right *mr*, 1 word
ROR	D2	*m001a dl dh*	Rotate Right *mr*, CL byte
ROR	D3	*m001a dl dh*	Rotate Right *mr*, CL word
SAHF	9E		Store AH into flags
SAL	D0	*m100a dl dh*	Shift Arithmetic Left/SHL *mr*, 1 byte
SAL	D1	*m100a dl dh*	Shift Arithmetic Left/SHL *mr*, 1 word
SAL	D2	*m100a dl dh*	Shift Arithmetic Left/SHL *mr*, CL byte
SAL	D3	*m100a dl dh*	Shift Arithmetic Left/SHL *mr*, CL word
SAR	D0	*m111a dl dh*	Shift Arithmetic Right *mr*, 1 byte
SAR	D1	*m111a dl dh*	Shift Arithmetic Right *mr*, 1 word

SAR	D2 *m111a dl dh*	Shift Arithmetic Right *mr*,CL byte
SAR	D3 *m111a dl dh*	Shift Arithmetic Right *mr*,CL word
SBB	IC *kk*	Subtract with borrow AL,*kk*
SBB	ID *kk jj*	Subtract with borrow AX,*jjkk*
SBB	81 *m011a dl dh jj kk*	Subtract with borrow *mr,jjkk*
SBB	83 *m011a dl dh jj kk*	Subtract with borrow *mr,jjkk* word
SBB	80 *m011a dl dh kk*	Subtract with borrow *mr,kk*
SBB	82 *m011a dl dh kk*	Subtract with borrow *mr,kk* byte
SBB	18 *mra dl dh*	Subtract with borrow *mr,r* byte
SBB	19 *mra dl dh*	Subtract with borrow *mr,r* word
SBB	1A *mra dl dh*	Subtract with borrow *r,mr* byte
SBB	1B *mra dl dh*	Subtract with borrow *r,mr* word
SCASB	AE	Scan byte (string)
SCASW	AF	Scan word (string)
SEG CS	2E	
SEG DS	3E	
SEG ES	26	
SEG SS	36	
SHR	D0 *m101a dl dh*	Shift Right *mr*,1 byte
SHR	D1 *m101a dl dh*	Shift Right *mr*,1 word
SHR	D2 *m101a dl dh*	Shift Right *mr*,CL byte
SHR	D3 *m101a dl dh*	Shift Right *mr*,CL word
STC	F9	Set carry flag
STD	FD	Set direction flag
STI	FB	Set interrupt flag
STOSB	AA	Store byte (string)
STOSW	AB	Store word (string)
SUB	2C *kk*	Subtract AL,*kk*
SUB	2D *kk jj*	Subtract AX,*jjkk*
SUB	29 *mra dl dh*	Subtract borrow *mr,r* word
SUB	81 *m101a dl dh jj kk*	Subtract *mr,jjkk*
SUB	83 *m101a dl dh jjj kk*	Subtract *mr,jjkk* word
SUB	80 *m101a dl dh kk*	Subtract *mr,kk*
SUB	82 *m101a dl dh kk*	Subtract *mr,kk* byte
SUB	28 *mra dl dh*	Subtract *mr,r* byte
SUB	2A *mra dl dh*	Subtract *r,mr* byte
SUB	2B *mra dl dh*	Subtract *r,mr* word
TEST	A8 *kk*	TEST AL,*kk*
TEST	A9 *kk jj*	TEST AX,*jjkk*
TEST	F7 *m000a dl dh jj kk*	Test *mr,jjkk*
TEST	F6 *m000a dl dh kk*	Test *mr,kk*
TEST	84 *mra dl dh*	TEST *mr,r* byte
TEST	85 *mra dl dh*	TEST *mr,r,* word
WAIT	9B	Wait
XCHG	95	Exchange ax,bp
XCHG	93	Exchange ax,bx
XCHG	91	Exchange ax,cx
XCHG	97	Exchange ax,di
XCHG	92	Exchange ax,dx
XCHG	96	Exchange ax,si
XCHG	94	Exchange ax,sp
XCHG	86 *mra dl dh*	Exchange *r,rm* byte

XCHG	87 *mra dl dh*	Exchange *r,rm* word
XLAT	D7	Translate
XOR	81 *m110a dl dh jj kk*	Exclusive Or *mr, jjkk*
XOR	80 *m110a dl dh kk*	Exclusive Or *mr,kk*
XOR	34 *kk*	Exclusive Or AL,*kk*
XOR	35 *kk jj*	Exclusive Or AX,*jjkk*
XOR	30 *mra dl dh*	Exclusive Or *mr,r* byte
XOR	31 *mra dl dh*	Exclusive Or *mr,r* word
XOR	32 *mra dl dh*	Exclusive Or *r,mr* byte
XOR	33 *mra dl dh*	Exclusive Or *r,mr* word

2. A Turbo–Pascal 1.0, 2.0 és 3.0 változata

2.1. A FENNTARTOTT SZAVAK

A fenntartott szavak változatlanok a *Turbo–Pascal* különböző verzióiban.

absolute
and
array
begin
boolean
byte
case
char
const
dispose
div
do
downto
else
end
external
false
file
for
forward
function
goto
if
in
inline
integer
label
mod
new
not
nil
of
or
overlay
packed
procedure
program
real
record
repeat
set
shl
shr
string
text
then
to
true
type
until
var
while
with
word
xor

2.2. ELŐRE DEFINIÁLT VÁLTOZÓK

Az előre definiált változók is azonosan a különböző verziókban

Változó	Jelentés
Aux	választható eszköz (AUX:)
AuxInPtr	az AuxIn függvény címe
AuxOutPtr	az AuxOut függvény címe
BufLen	a beviteli puffer hossza
Con	konzol (CON:)
ConInPtr	a ConIn függvény címe
ConOutPtr	a ConOut eljárás címe
ConStPtr	a ConSt függvény címe
Cseg	a Cs regiszter tartalma
Dseg	a DS regiszter tartalma
HeapPtr	halommutató (heap–)
Input	input file (alapértelmezésben)
Kbd	billentyűzet
Lst	listázó (nyomtató)
LstOutPtr	az LstOut eljárás címe
MaxInt	a legnagyobb egész szám
Mem[s:o]	a tár s:o címének tartalma byte-ban
MemW[s:o]	a tár s:o címén levő szó tartalma
Ofs(v)	a v változó ofszetje
Output	Output file (alapértelmezésben)
Pi	π értéke
Port [p]	Array of Byte a portra vonatkoztatva
PortW [p]	Array of Integer a portra vonatkoztatva
Seg(v)	a v változó szegmens értéke
Sseg	az SS regiszter tartalma
Trm	terminál
Usr	felhasználói egység
UsrInPtr	az UsrIn függvény címe
UsrOutPtr	az UsrOut eljárás címe

2.3. ELŐRE DEFINIÁLT FÜGGVÉNYEK

Függvény	Típus	Verzió-számtól	Leírás
Abs(*v*)	real/integer	1	abszolút érték
Addr(*v*)	pointer	1	cím
ArcTan(*v*)	real	1	árkusz tangens
Chr(*v*)	char	1	ASCII-jel *v* értékhez
Concat(*s*, ..)	string	1	karakterláncok összefűzése
Copy(*s*, *p*, *l*)	string	1	készlánc : *s*-ből, *p*-től, *l* db karakter
Cos(*v*)	real	1	koszinusz
Eof(*v*)	boolean	1	file vége?
Eoln(*v*)	boolean	1	sor vége?
Exp(*v*)	real	1	exponenciális érték
FilePos(*v*)	integer	1	az aktuális rekordsor száma
Filesize(*v*)	integer	1	a file rekordjainak a száma
Frac(*v*)	real	1	a szám tizedes része
Hi(*v*)	byte	1	felső byte
Int(*v*)	integer	1	a szám egész része
IOresult	integer	1	a file-kezelés hibakódja
Key Pressed	boolean	1	billentyű lenyomva?
Length(*v*)	integer	1	a karakterlánc hossza
Ln(*v*)	real	1	természetes logaritmus
Lo(*v*)	byte	1	alsó byte
LongFilePosition	real	1	az aktuális rekordszám
LongFileSize	real	1	a file rekordjainak száma
MaxAvail	integer	1	a maximális halomméret
MemAvail	integer	1	a rendelkezésre álló halomméret
Odd(*v*)	boolean	1	a szám páratlan?
Ord(*v*)	integer	1	az elemhez rendelt sorszám
Pos(*t*,*s*)	integer	1	a részlánc pozíciója a karakterláncban
Ptr(*s*,*o*)	pointer	1	mutató *s*-ből és *o*-ból
Pred(*v*)	scalar	1	a *v* előtti elem
ParamCount	integer	3	a paraméterek száma
ParamStr(*v*)	string	3	paraméterérték
Random	real	1	véletlen szám
Random(*v*)	integer	1	véletlen szám
Round(*v*)	integer	1	kerekített érték
SeekEof(*v*)	boolean	1	szövegfile vége?
SeekEoln(*v*)	boolean	1	szövegsor vége?
Sin(*v*)	real	1	színusz
Sqr(*v*)	real/integer	1	négyzet
Sqrt(*v*)	real	1	négyzetgyök
Succ(*v*)	scalar	1	a *v* mögötti elem
Swap(*v*)	integer	1	a byte alsó és felső értékének cseréje
Trunc *v*	integer	1	a megelőző legnagyob egész szám
UpperCase	char	1	nagybetű

2.4. ELŐRE DEFINIÁLT ELJÁRÁSOK

Eljárás	Verzió-számtól	Leírás
Append(*f,s*)	3	hozzáfűzés a file-hoz
Assing(*f,s*)	1	file-változó és file-név hozzárendelése
BlockRead(*f,s,r*)	1	típus nélküli file olvasása
BlockWrite(*f,s,r*)	1	típus nélküli file írása
Chain(*f*)	1	az új főprogram behívása
ChDir(*s*)	3	az aktuális könyvtár megváltoztatása
Close(*f*)	1	a file lezárása
ClrEol	1	törlés a képernyősor végéig
ClrScr	1	a képernyő törlése
CrtExit	1	a Terminal Reset String kiküldése
CrtInit	1	Terminal Initialization String kiküldése
Delay(*i*)	1	késleltetés
Delete(*s,p,l*)	1	az *s* láncból *p*-től *l*-jel törlése
DelLine	1	sortörlés (a többi felzárkózik)
Erase(*f*)	1	a file törlése
Execute(*f*)	1	az új főprogram behívása
Exit	1	elhagyja az aktuális blokkot (alprogramot)
FillChar(*v,n,w*)	1	*v*-től kezdve *n* byte-ot *w*-vel tölt fel
Flush(*f*)	1	a file-puffer kiürítése
FreeMem(*p,i*)	1	a halomtároló felszabadítása
GetDir(*i,s*)	3	az aktuális könyvtár megjelenítése
GotoXY(*s,z*)	1	a kursor pozicionálása sorra, helyre
Halt	1	a programvégrehajtás befejezése
Halt(*i*)	3	befejezés és Return-kód visszaadása
Insert(*s1, s2,p*)	1	az s1 lánc beszúrása az *s2*-be a *p* helyen
InsLine	1	üres sor beszúrása a képernyőn
Intr(*i,r*)	1	az *i* DOS–Interrupt hívása
LongSeek(*f,r*)	1	a file-mutató mozgatása az *r* rekordra
Mark(*p*)	1	a halommutató tárolása *p*-be
MkDir(*s*)	3	alkönyvtár készítése
Move(*v1,v2,n*)	1	*v1*-ből *v2*-be *n* byte másolása
MsDos(*r*)	1	$21 DOS-megszakítás hívása
OvrPath(*s*)	3	az overlay file-hoz út megadása
Randomize	1	a véletlenszám-generátor indítása
Read(*f,v,*)	1	olvasás a file-ból
Readin(*f,v*)	1	olvasás a file-ból
Release(*p*)	1	a halom felszabadítása a *p* mutatótól
Rename(*f,s*)	1	a file átnevezése
Reset(*f*)	1	meglevő file megnyitása
Rewrite(*f*)	1	új file megnyitása
RmDir(*s*)	3	a könyvtár törlése
Seek(*f,i*)	1	a file-mutató pozicionálása az *i* sorszámú rekordra
Str(*n,s*)	1	az *n* szám átalakítása *s* láncba
Truncate(*f*)	3	a file levágása a file-mutatónál
Val(*s,n,c*)	1	az *s* lánc átalakítása az *n* számba, a *c*-ben a hiba van
Write(*f,v*)	1	írás a file-ba
Writeln(*f,v*)	1	írás a file-ba

3. ASCII-táblázat

Az *ASCII* (American Standard Code for Information Interchange) szabványosított kódrendszer elsősorban a megjelenített szöveghez kapcsolódik. Más kódok mellett ezt alkalmazzuk az adatok tárolására, valamint az adatátvitelhez a számítógép és a perifériás egységei között.

Az *ASCII* 7 bites kód, ezért 128 különböző karakter ábrázolására alkalmas. Elegendő tehát, az összes betűhöz és számjegyhez és még marad hely a speciális jelek számára is. Bár a szokásnak megfelelően jelenként mindig 8 bitet tartunk fenn (a tár byte-onként címezhető), néhány gyártó bővített ASCII kódrendszert definiált, amely 256 karakter kódjait tartalmazza. Ez a rész (a 127 és a 255 közötti karakterek) nem szabványosított, ezért itt nem szerepeltetjük.

Az alsó 32 *ASCII* karakter nem nyomtatható, az adatcseréhez használatosak vezérlőjelként. Ezeket az első táblázatba írtuk.
A második táblázat a teljes *ASCII*-tartományt mutatja és négy csoportra tagozódik. Ezek a csoportok az egyes karakterek összetartozását ábrázolják, ebből kitűnik, hogy például a vezérlőjel és kisbetű csak egy bitben különbözik a nagybetűtől.

Az alábbi rövidítéseket alkalmaztuk

SP . . . jelentése Space
DEL . . . jelentése Delete-karakter

Kód hex.	dec.	Karakter	Megnevezés
00	00	^ @NUL	Null
01	01	^ A SOH	Start of Heading
02	02	^ B STX	Start of Text
03	03	^ C ETX	End of Text
04	04	^ D EOT	End of Transmit
05	05	^ E ENQ	Enquiry
06	06	^ F ACK	Acknowledge
07	07	^ G BEL	Bell
08	08	^ H BS	Backspace
09	09	^ I HT	Horizontal Tab
0A	10	^ J LF	Line Feed
0B	11	^ K VT	Vertical Tab
0C	12	^ L FF	Form Feed
0D	13	^ M CR	Carriage Return
0E	14	^ N SO	Shift Out
0F	15	^ O SI	Shift In
10	16	^ P DLE	Data Line Escape
11	17	^ Q DC1	Device Control 1
12	18	^ R DC2	Device Control 2
13	19	^ S DC3	Device Control 3
14	20	^ T DC4	Device Control 4
15	21	^ U NAK	Negative Acknowledge
16	22	^ V SYN	Synchronous idle

17	23	^W ETB	End of Transmit Block
18	24	^X CAN	Cancel
19	25	^Y EM	End of Medium
1A	26	^Z SUB	Substitute
1B	27	^% ESC	Escape
1C	28	^\ FS	File Seperator
1D	29	^& GS	Group Seperator
1E	30	^ RS	Record Seperator
1F	31	^_US	Unit Seperator

Kód	hex.	dec.	Karakter	Kód hex:	dec.	Karakter
	00	00	^@	10	16	^P
	01	01	^A	11	17	^Q
	02	02	^B	12	18	^R
	03	03	^C	13	19	^S
	04	04	^D	14	20	^T
	05	05	^E	15	21	^U
	06	06	^F	16	22	^V
	07	07	^G	17	23	^W
	08	08	^H	18	24	^X
	09	09	^I	19	25	^Y
	0A	10	^J	1A	26	^Z
	0B	11	^K	1B	27	^[
	0C	12	^L	1C	28	^\
	0D	13	^M	1D	29	^]
	0E	14	^N	1E	30	
	0F	15	^O	1F	31	^_
	20	32	SP	30	48	0
	21	33	!	31	49	1
	22	34	”	32	50	2
	23	35	#	33	51	3
	24	36	$	34	52	4
	25	37	%	35	53	5
	26	38	&	36	54	6
	27	39	’	37	55	7
	28	40	(	38	56	8
	29	41	)	39	57	9
	2A	42	*	3A	58	:
	2B	43	+	3B	59	;
	2C	44	,	3C	60	<
	2D	45	–	3D	61	=
	2E	46	.	3E	62	>
	2F	47	/	3F	63	?

Kód	hex.	dec.	Karakter	Kód	hex.	dec.	Karakter
	40	64	@		50	80	P
	41	65	A		51	81	Q
	42	66	B		52	82	R
	43	67	C		53	83	S
	44	68	D		54	84	T
	45	69	E		55	85	U
	46	70	F		56	86	V
	47	71	G		57	87	W
	48	72	H		58	88	X
	49	73	I		59	89	Y
	4A	74	J		5A	90	Z
	4B	75	K		5B	91	[
	4C	76	L		5C	92	\
	4D	77	M		5D	93	]
	4E	78	N		5E	94	^
	4F	79	O		5F	95	—
	60	96	,		70	112	p
	61	97	a		71	113	q
	62	98	b		72	114	r
	63	99	c		73	115	s
	64	100	d		74	116	t
	65	101	e		75	117	u
	66	102	f		76	118	v
	67	103	g		77	119	w
	68	104	h		78	120	x
	69	105	i		79	121	y
	6A	106	j		7A	122	z
	6B	107	k		7B	123	{
	6C	108	l		7C	124	|
	6D	109	m		7D	125	}
	6E	110	n		7E	126	~
	6F	111	o		7F	127	DEL

4. Programpéldák jegyzéke

Csak a teljes, futtatható programok, amelyek lemezen is beszerezhetők, kerültek felsorolásra.

I. Elmélet

II. Gyakorlat

Leírás (Turbo–Pascal forrásprogramok)		Név
1.1.	Keretprogram rendezéshez	Rendhivo
1.2.	Beszúrásos rendezés (lassabb)	Rendez1
1.3.	Beszúrásos rendezés (gyorsabb)	Rendez2
1.4.	Rendezés Shell-módszerrel	Rendez3
1.5.	Gyorsrendező	Rendez4
1.6.	Gyorsrendező (20 elemnél kevesebbre)	Rendez5
1.7.	Fastruktúrás rendezés	Rendez7
1.8.	Lineáris keresés	LinKer
1.9.	Bináris keresés	Kerhivo
1.10.	Beszúrásos rendezés, bináris kereséssel	Rendez6
1.11.	Keresztreferencia generátor	Nyomtat
2.1.	LOLA-elemző	Elemzo
2.2.	LOLA-fordító	Fordito
2.3.	LOLA-értelmező	Run
3.1.	Nagyobb pontosságú számolás	Szamolas
4.1.	File-kezelés	FileDir
4.2.	File tartalmának hex. megjelenítése	FileDump
4.3.	I/O egységek állapotának vizsgálata	EgysEll

Leírás (LOLA forrásprogramok)		
1.	Hatványozás	Negyzet
2.	Ciklusszervezés	Ciklus
3.	Feltételes elágazás	Szamlal
4.	Számok összegzése	Összeg
5.	Átlagfogyasztás számítása	BenzFogy

5. Védjegyek (trademarks)

Az alábbi védett márkaneveket használtuk a könyvünkben.

Turbo–Pascal **TURBO–Toolbox**	a **Borland International Inc.** (USA) termékei. A német nyelvű kézikönyvet a **Heimsoeth Software** (NSZK) készítette.
MS–DOS, **XENIX**	A **Microsoft Corp.** (USA) védett termékjelei.
CP/M	A **Digital Research Inc.** (USA) védett termékjele.
Intel 8088, **Intel 8086,** **Intel 80186,** **Intel 80286,** **Intel 80386,**	Az **Intel Corp.** (USA) mikroprocesszorai.
IBM–PC, **PC–DOS**	Az **IBM Corp.** (USA) védett termékjelei.
VICTOR, **VICKI,** **SIRIUS 1**	A **VICTOR Technologies Inc.** (USA) védett termékjelei.

Irodalom

[1] *Aho, Hopcroft, Ullman:* The Design and Analysis of Computer Algorithms, Addison–Wesley, Reading, Mass., 1974.

[2] *P. Bachmann:* Grundlagen der Compilertechnik, Oldenburg, 1975.

[3] *A. C. Hartmann:* A Concurrent Pascal Compiler for Minicomputers. Springer Verlag, 1977.

[4] *Jähnichen, Oeters, Willis:* Übersetzerbau. Vieweg, 1978.

[5] *D. E. Knuth:* The Art of Computer Programming Vol III: Sorting and Searching. Addison–Wesley, Reading, Mass., 1973.

[6] *R. L. Kruse:* Data Structures and Program Design. Prentice–Hall, 1984.

[7] *P. Norton:* Die verborgenen Möglichkeiten des IBM PC. Carl Hanser Verlag, 1985.

[8] *Rheingold, Nievergelt:* Deo, Combinatorial Algorithms. Prentice–Hall, 1977.

[9] *Sargent III, Shoemaker:* The IBM Personal Computer from the Inside Out. Addison–Wesley Publishing Company, 1984.

[10] *L. J. Scanlon:* Die Assemblersprache des IBM–PC & XT, Markt und Technik Verlag, 1985.

[11] *Waite, Goos:* Compiler Construction. Springer Verlag, 1984.

[12] *N. Wirth:* Algorithmen und Datenstrukturen. Teubner Studienbücher, 1975.

[13] *N. Wirth:* Compilerbau. Teubner Studienbücher, 1977.

290,– Ft