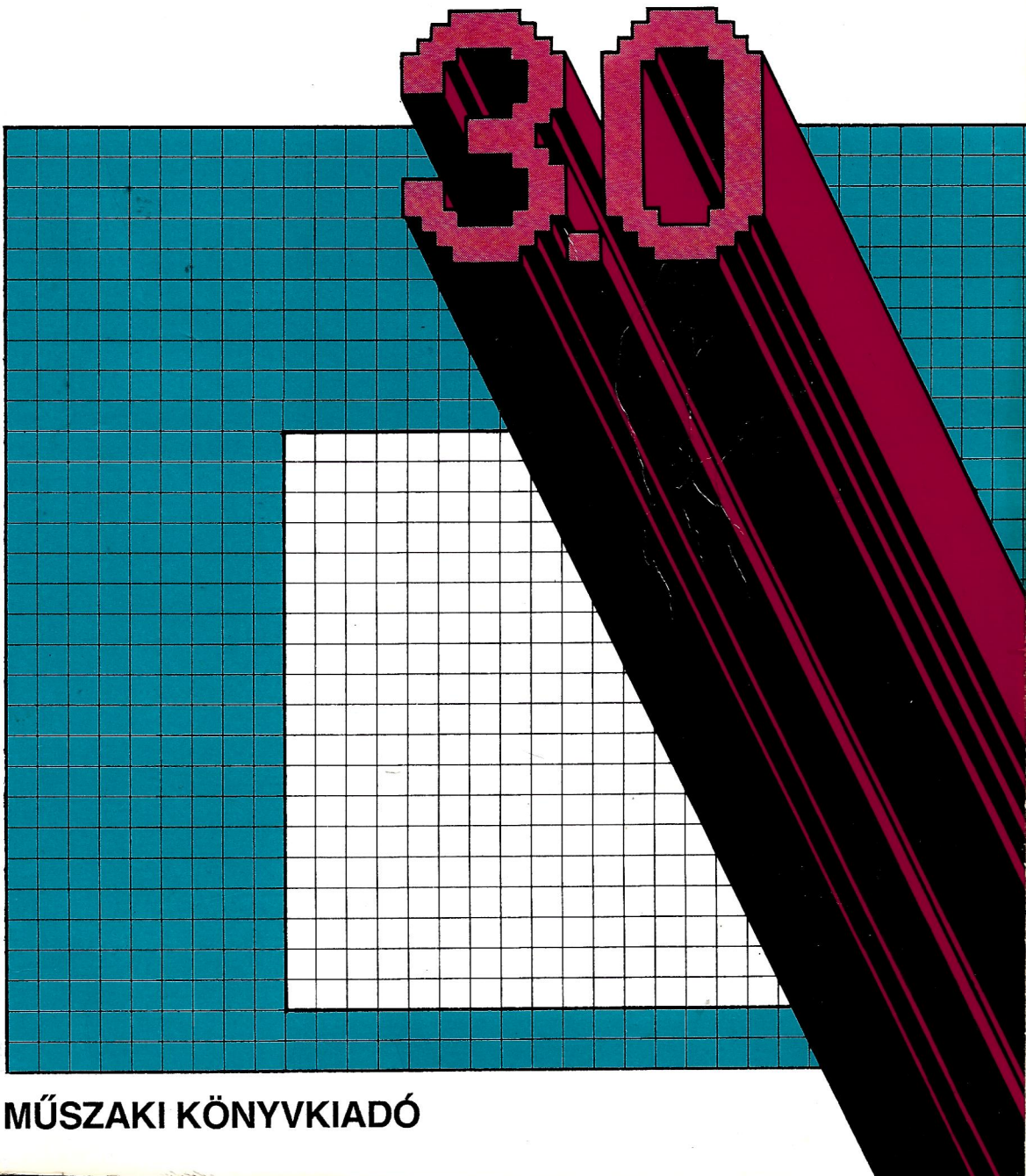
Jan Bielecki

# TURBO PASCAL 3.0



MŰSZAKI KÖNYVKIADÓ

Jan Bielecki

# Turbo Pascal 3.0

Jan Bielecki

# Turbo Pascal 3.0

Műszaki Könyvkiadó, Budapest, 1990

Eredeti mű: Jan Bielecki: **Turbo Pascal. Wersja 3.0**
originally published by WNT, 1987.

Lektorálta: **Marx Ferenc** okl. villamosmérnök



ETO: 681.3.06
ISBN 963 10 8586 4

Kiadja a Műszaki Könyvkiadó
Felelős kiadó: Szűcs Péter igazgató

BM KEP Nyomdaüzem – 90.153
Felelős vezető: Jónás György

Felelős szerkesztő: Györke Tiborné okl. villamosmérnök
A szedés a Műszaki Könyvkiadóban készült
Műszaki vezető: Bereczki Gábor
Műszaki szerkesztő: Vigh László
A fedelet tervezte: Kováts Tibor
A könyv formátuma: B/5
Ívterjedelme: 14 (A/5)
Azonossági szám: 80 030
MŰ: 4442-i-9093

# Tartalom

# 1. Bevezetés

A Pascal nyelv története 1972-ben kezdődik, amikor Niklaus Wirth közreadta a nyelv leírását tartalmazó javított könyvet.

Kezdetben a Pascal nyelvet a strukturált programozás tanítására alkalmas eszközként használták. Az olcsó és jó fordítóprogramok kidolgozása eredményeképpen a nyelv rendkívül gyorsan elterjedt, és jelenleg a modern programozási nyelvek egyik legfontosabb képviselője — különösen a rendszerprogramozás területén.

A szabványos Pascal nyelv egyik fontos implementációja a vele 90%-ban kompatíbilis Turbo Pascal nyelv, amelyet elsősorban mikroszámítógépes alkalmazásokra fejlesztett ki a Borland cég, és amely piaci megjelenése után azonnal világsiker lett.

A Borland cég által megvalósított implementáció legfontosabb előnyei közé tartoznak: a fordítóprogram kis mérete, az igen gyors fordító algoritmus, a fordítóprogram és az interaktív képernyős szövegszerkesztő összekapcsolása, a forrásszintű hibajelzések, a nagy szubrutinkönyvtár, valamint azok a bővítések, amelyek a szabványos nyelvhez képest nagymértékben megkönnyítik a rendszerprogramozást.

A Turbo Pascal interaktív programozási rendszer, amely a Turbo Pascal nyelv fordítóprogramjából és a vele összekapcsolt képernyős szövegszerkesztőből áll. A Turbo Pascal szövegszerkesztőjét a közismert WordStar szerkesztőprogramról mintázták. A rendszer nemcsak azért interaktív, mert kényelmes szövegszerkesztést tesz lehetővé, hanem azért is, mert a programban feltárt hibákat a forrásnyelvi szövegben jelöli meg.

A szövegszerkesztés, fordítás és futtatás közötti üzemmódváltás természetes és kevés manipulációt igényel. Emiatt a Turbo Pascal rendszerben a programok „belövése" szinte kizárólag forrásszintű. A jelentős fordítási sebesség miatt a forrás- és a futtatható tárgyprogram közötti váltás szinte azonnali, ez számottevően lerövidíti a javítási és újrafuttatási ciklust, meggyorsítva a végleges programváltozat kifejlesztését.

A Turbo Pascal rendszer használatát egy egyszerű program mutatja be, amely az adott sugarú gömb térfogatát számítja ki:

```
program Terfogat:
var Sugar, Terf: real:
begin
  ClrScr;
  GotoXY(10,11);
  Write('sugar=');
  Readln(Sugar);
  Terf:=4/3*Pi*Sugar*Sugar*Sugar*;
```

```
    GotoXY(10,13);
    Writeln('terfogat=',Terf);
repeat until KeyPressed
end.
```

A programban szerepel az előre definiált *Pi* konstansnév* (a π szám közelítése), valamint a *Sugar* (a gömb sugara) és a *Terf* (a gömb térfogata) szimbolikus név. A *ClrScr* eljárás végrehajtása képernyőtörlést eredményez, a *GotoXY* eljárás pedig pl. a *GotoXY(10,13)* esetén a képernyő 10. oszlopát és 13. sorát jelenti. A sor- és oszlopkoordináták értéke 1-től a max. sorszámig változhat. A **repeat until** ciklus leállítja a program további futását mindaddig, amíg a billentyűzetről nem viszünk be egy tetszőleges karaktert. Ez a karakter megmarad a bemeneti pufferben, és a Turbo Pascal rendszer később értelmezheti.

A program lefordításához és a futtatáshoz először el kell indítani — a TURBO név begépelésével — a Turbo Pascal rendszert. A program indítása után a képernyőn a következő rendszerazonosító üzenet jelenik meg:

```
TURBO Pascal system
Copyright (C) ... Borland Inc.
Include error messages (Y/N)?
```

A megjelenő szövegben szereplő kérdés arra kíváncsi, akarjuk-e a hibaüzenetek szövegét a rendszerhez csatolni. Mivel a hibaüzenetek helyigénye csak 1,5 kbájt, s használatuk jelentősen megkönnyíti a hibakeresést, ajánlatos az Y (igen) válasz megadása.

A választ követően a képernyőn megjelenik a rendszermenü, amelyben az egyes szavak kiválasztandó betűit nagyobb fényerő jelzi.

```
Logged drive:
Work file:
Main file:
Edit Compile Run Save
Dir Quit compiler Options
Text:
Free:
```

A kiemelt betűnek megfelelő billentyű leütésével interaktív módon adhatjuk ki a rendszerparancsokat.

* A konstansnév olyan azonosító, amely egy konstans értéket képvisel. E konstans olyan objektum, amely a program végrehajtása során nem változik. Több szerző más fogalmakat használ, és a konstansneveket „állandók"-nak, a változóneveket pedig „változók"-nak nevezi.

### *Az L parancs*

Az L billentyű megnyomása lehetővé teszi az alapértelmezés szerinti mágneslemezes meghajtóegység megváltoztatását. Ha pl. a B meghajtót szeretnénk kijelölni, akkor a feltett

New drive:

kérdésre be kell gépelnünk a B: karakterpárt, és a parancs megadását a CR (8 bites mikroszámítógép) vagy az Enter (IBM PC) billentyűvel kell befejeznünk.

### *A W parancs*

A W billentyű megnyomása után megadhatjuk a forrásprogramot tartalmazó munkaállomány nevét. A megadott név egy létező vagy egy új, létrehozandó állomány neve. A feltett

Work file name:

kérdésre be kell gépelnünk az állomány nevét meghatározó szöveget, amelyet a CR (8 bites mikroszámítógép) vagy az Enter (IBM PC) billentyűvel zárunk le.

A CP/M-80 és a PC DOS operációs rendszerben az állománynév két tagból állhat. Az első tag legfeljebb 8 karakter hosszúságú, a második tag — melyet kiterjesztésnek nevezünk — max. 3 karakter hosszúságú lehet. A név két tagját a . (pont) karakter választja szét. Ha az előző kérdésre csak az állománynév első tagját adjuk meg, akkor az alapértelmezés szerint a név automatikusan kiegészül egy ponttal és a PAS kiterjesztéssel. Ha le akarjuk tiltani az automatikus kiterjesztés-hozzáadást, elegendő begépelni az állománynév első tagját és a pontot.

*Megjegyzés:* ha egy új munkaállományt jelölünk ki — mielőtt az eddigit kivittük volna a mágneslemezre —, akkor a rendszer előbb megkérdezi, hogy az eddigi forrásszöveget szándékozzuk-e megőrizni. A lehetséges válasz e kérdésre (szokásos módon) Y (igen) vagy N (nem).

### *Az M parancs*

Az M billentyű megnyomása után megadhatjuk a lefordítandó forrásprogramot tartalmazó állomány nevét. Ha ezt a nevet nem határozzuk meg, akkor a lefordítandó forrásszöveget az aktuális munkaállomány tartalmazza.

A lefordítandó forrásprogramot tartalmazó állomány (a továbbiakban főállomány) nevére vonatkozó szabályok megegyeznek a munkaállomány nevére vonatkozó előírásokkal. A főállomány használata akkor kényelmes, ha a forrásprogram tartalmaz olyan direktívákat, amelyek — az előfordulási helyükön — beiktatnak más állományokban szereplő programrészeket. Ha ekkor valamely „beiktatott" programrészben a fordító hibát jelez, az adott programrészt tartalmazó állomány automatikusan munkaállománnyá válik, amely azonnal szerkeszthető, javítható. A javítások befejezése után egyszerűen újraindíthatjuk a főállományban szereplő forrásprogram fordítását.

### *Az E parancs*

Az E billentyű leütésével behívjuk a képernyős szövegszerkesztőt, ezután elkezdhetjük a munkaállományban levő szöveg szerkesztését. Ha még nem határoztuk meg a munkaállomány nevét, akkor először a W parancsnál ismertetett módon a rendszer megkérdezi az állománynevet, s csak ezután kezdődhet el a szerkesztés.

### *A C parancs*

A C billentyű leütésével elkezdhetjük a főállományban levő forrásszöveg fordítását. Ha eddig nem határoztuk meg a főállomány nevét, akkor a munkaállományban szereplő szöveget fordítja le a rendszer. Ha megadtuk ugyan a főállomány nevét, de a munkaállományt szerkesztettük, akkor a rendszer előbb elmenti a munkaállományt a háttértárolóra, s csak ezután kezdi el a fordítást.

A fordítás eredményeképpen keletkező tárgyprogram az operatív tárba vagy .COM, ill. .CHN kiterjesztésű mágneslemezes állományba kerül. A fordítás jellemzői az O paranccsal (fordítási opciók) állíthatók be. Az alapértelmezés szerint a tárgyprogram az operatív tárban kap helyet.

Ideiglenesen bármelyik billentyűvel megszakíthatjuk a fordítást. A feltett

Abort compilation (Y/N)?

kérdésre válaszolva, a fordítás véglegesen megszakad (Y), ill. folytatódik (N).

### *Az R parancs*

Az R billentyűvel elindítjuk az operatív tárban levő, lefordított program végrehajtását. Ha az O paranccsal előzetesen beállítottuk a COM-fájl opciót, akkor a rendszer a mágneslemezről előbb betölti a programot. Ha az R parancs előtt nem használtuk a C parancsot, akkor a program fordítása automatikusan megelőzi futtatását.

### *Az S parancs*

Az S billentyű megnyomása után a rendszer mágneslemezre menti az eddig szerkesztett forrásszöveget. A munkaállomány előző, eredeti példányának névkiterjesztése .BAK-ra változik.

### *Az X parancs (csak a CP/M-80 alatt)*

Az X billentyűvel tetszőleges bináris programot futtathatunk le. A feltett

Program:

kérdésre a végrehajtandó — gépi kódú — programot tartalmazó, .COM kiterjesztésű mágneslemezes állomány nevét kell megadnunk.

### *A D parancs*

A D billentyű leütésével egy tetszőleges mágneslemezes alkönyvtár tartalomjegyzéke jeleníthető meg a képernyőn. A feltett

Dir mask:

kérdésre megadhatjuk a konkrét állománynevet vagy egy állománycsoport nevét, pl. B: *.JB? (a * tetszőleges karaktersorozatot, a ? pedig tetszőleges karaktert jelent). Ha a kérdésre nem adjuk meg a meghajtó vagy az alkönyvtár nevét, akkor a főmenü szerinti alapértelmezés a mérvadó.

### *A Q parancs*

A Q billentyű megnyomásával befejeződik a Turbo Pascal rendszer futása, a vezérlés visszakerül az operációs rendszerhez. Ha a Q parancs megadása előtt szerkesztettük a munkaállományt, akkor a rendszer előbb megkérdezi, hogy elmentjük-e mágneslemezre a szerkesztett szöveget.

### *Az O parancs*

Az O billentyűvel fordítási opciókat adhatunk meg. Az alapértelmezés szerint a lefordított program az operatív tárban foglal helyet. Ha az O parancs után a C alparancsot adtuk ki, akkor a tárgykód egy .COM kiterjesztésű mágneslemezes állományba kerül. Az előző alapértelmezés az M alparanccsal visszaállítható.

A H alparancs hatására a tárgykód egy .CHN kiterjesztésű mágneslemezes állományba kerül. A .COM és a .CHN kiterjesztésű állományok közötti különbség az, hogy a .CHN kiterjesztésű állomány nem tartalmazza a standard szubrutinkönyvtárat, emiatt kizárólag egy másik programból hívható a Chain eljárással.

A P alparanccsal paramétereket adhatunk meg az operatív tárban levő program számára. E paramétereket ugyanúgy adja át a rendszer, mint a mágneslemezről betöltött programokban.

Az F paranccsal a hiba helyét határozhatjuk meg azokban a programokban, amelyek a .COM és a .CHN kiterjesztésű állományokban kaptak helyet. Az operatív tárban levő programokban a rendszer a forrásprogramban megjelöli a hiba helyét; a mágneslemezen tárolt programokban viszont a feltárt futásidejű hibák helye kódoltan (a hiba sorszáma és az utasításszámláló értéke) áll rendelkezésre. Ha az F alparancs hívásával megadjuk az utasításszámláló értékét, akkor a rendszer a forrásprogramban megjelöli a futásidejű hiba helyét.

A Q alparancs az O parancs utolsó opciója. Hatására befejeződik az O parancs végrehajtása, és a vezérlés a Turbo Pascal főmenüjéhez kerül vissza.

A fenti leírásból látható, hogy a Turbo Pascal rendszer parancsainak és alparancsainak használata nem túlságosan bonyolult. Egyszerű a forrásprogram írása és javítása is (az E paranccsal vagy automatikusan, a fordítási hibák fellépésekor).

A Turbo Pascal rendszer szövegszerkesztője nagymértékben hasonlít az ismert WordStar programhoz. A C függelékben megtalálható a szövegszerkesztő teljes leírása; itt csak a forrásszövegek alapvető feldolgozására alkalmas, minimális parancskészletet mutatjuk be.

*Megjegyzés:* ha nem követünk el hibákat, akkor közvetlenül a szövegszerkesztő behívása után (E parancs) begépelhetjük a program szövegét karaktersorok formájában, amelyeket a CR (8 bites mikroszámítógép) vagy az Enter (IBM PC) billentyűvel zárunk le. A szöveg begépelése után a ^KD paranccsal befejezzük a szerkesztést, és visszatérünk a főmenühöz.

A ^KD parancs a K és a D betűből áll, s — a Turbo Pascal rendszer szövegszerkesztőjének többi parancsához hasonlóan — szükség van még a Ctrl billentyűre, amelyet lenyomva kell tartanunk a K billentyű leütésekor (rövidítve: Ctrl-K-D vagy ^KD). A képernyőn megjelenített szövegrészen belül fontos szerepe van a kurzor feletti karakternek. A szövegmanipulációk mindig e karakterre (vagy az őt tartalmazó szóra, ill. sorra) vonatkoznak. A kurzor mozgatása egy pozícióval balra, jobbra, felfelé vagy lefelé a Ctrl-S, Ctrl-D, Ctrl-E vagy Ctrl-X parancsokkal lehetséges. E parancsok S, D, E és X billentyűi négyzetet alkotnak:

E

S D

X

Így könnyű megjegyezni, hogy a Ctrl-E parancs egy pozícióval felfelé, a Ctrl-X egy pozícióval lefelé, a Ctrl-S egy pozícióval balra, s végül a Ctrl-D egy pozícióval jobbra mozgatja a kurzort. A kurzormozgató parancsok ilyen hozzárendelése nem az egyes betűkhöz kapcsolódik, hanem a négy billentyű síkbeli elrendezésén alapul.

Karaktertörlésre a ← billentyű (Backspace) használható, amely törli a kurzortól balra eső karaktert. Új karakterek beszúrásához (akár szavakon belül, akár szavak közt) nincs szükség semmilyen különleges műveletre; a begépelt karakterek a kurzor fölötti karakter elé szúródnak be, ugyanakkor a sor többi karaktere egy pozícióval jobbra tolódik. Egész sorok törlésére a Ctrl-Y parancs, szavak törlésére a Ctrl-T parancs, új sorok beszúrására a Ctrl-N parancs alkalmas. A szövegszerkesztő számos más parancsát és lehetőségét a C függelék írja le. E fejezet tartalmát összegezve elmondhatjuk, hogy a fejezet elején bemutatott 1. program futtatható változatának előállítása a következő műveletek elvégzését igényli:

- Indítsuk el a Turbo Pascal rendszert!
  TURBO
- Csatoljuk a rendszerhez a hibaüzenetek szövegét!
  Y
- Adjuk meg a munkaállomány nevét!
  W
- Kezdjük el a forrásprogram gépelését!
  E
- Fordítsuk le a forrásprogramot!
  C
- Futtassuk le a tárgyprogramot!
  R

E tevékenységek befejezése után a Q paranccsal visszatérhetünk az operációs rendszerhez.

Ha a tárgyprogramot szeretnénk megőrizni egy mágneslemezes állományban (.COM kiterjesztéssel), akkor a fordítás előtt adjuk ki az O parancsot, utána pedig a C és a Q alparancsokat. A fordítás és az operációs rendszerhez való visszatérés után a példaprogram már a Turbo Pascal rendszer felügyelete nélkül, önállóan is futtatható.

# 2. Lexikális elemek, elválasztók, megjegyzések

A Turbo Pascal programozási nyelv részletes leírását lexikális (szótári) elemeinek felsorolásával kezdjük. A többi programozási nyelvhez hasonlóan ezek a következők: kulcsszavak, azonosítók, konstansok és határolók. A szóközök, tabulátorjelek, sorvégjelek és megjegyzések (kommentárok) nem lexikális elemek; e karakterek és a megjegyzések tetszőleges sorozata (a továbbiakban elválasztó szintaktikai elem) — a fordító szempontjából — egy szóköznek tekinthető. Az elválasztó elem használata csak akkor nélkülözhetetlen, ha a forrásprogramban szomszédos azonosítók vagy kulcsszavak szerepelnek.

Az osztályozásnak megfelelően egy Turbo Pascal nyelvű program lexikális és elválasztó elemekből tevődik össze. A program lexikális elemzésekor különíthetjük el az egyes elemeket. Az elemzés természetes sorrendben megy végbe, vagyis balról jobbra, ill. felülről lefelé. A fordító az adott karaktersorozatot akkor tekinti lexikális elemnek, ha nem tartalmaz elválasztót.

A lexikális elemek és megjegyzések a következő alapszimbólumokból épülnek fel:

- az angol ábécé kis- és nagybetűi, valamint az aláhúzásjel
  _ , a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z
- decimális számjegyek
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- különleges szimbólumok
  + — * / = { }
  ( ) [ ] < > ^
  . , : ; ' ' # $
- összetett szimbólumok
  értékadás: :=
  relációk: <> <= >=
  intervallum: . .
  zárójelek: (* *) (. .)

(A „csillagos" zárójelek kapcsos zárójelet, a „pontos" zárójelek pedig szögletes zárójelet helyettesítenek.)

A nagy- és kisbetűket az azonosítókban és a kulcsszavakban azonosnak tekinti a nyelvi definíció. Emiatt pl. a filename és a FileName azonosító nem különbözik egymástól.

## 2.1. Kulcsszavak

A felsorolt, kötött jelentésű, összefüggő betűsorozatok a kulcsszavak:

| | | | |
|---|---|---|---|
| **absolute** | **external** | **nil** | **shl** |
| **and** | **file** | **not** | **shr** |
| **array** | **for** | **of** | **string** |
| **begin** | **forward** | **or** | **then** |
| **case** | **function** | **overlay** | **to** |
| **const** | **goto** | **packed** | **type** |
| **div** | **if** | **procedure** | **until** |
| **do** | **in** | **program** | **var** |
| **downto** | **inline** | **record** | **while** |
| **else** | **label** | **repeat** | **with** |
| **end** | **mod** | **set** | **xor** |

**Példa**

A Turbo Pascal nyelven megírt legrövidebb program a következő:

```
begin
end.
```

- A program két kulcsszóból és egy határolóból áll.
- A program lefuttatása semmilyen következménnyel sem jár.

## 2.2. Azonosítók

A betűvel kezdődő, betűkből és/vagy számjegyekből álló karaktersorozatot (feltéve, hogy nem azonos egyetlen kulcsszóval sem) azonosítónak nevezzük. A „betű" alatt az angol ábécé tetszőleges kis- vagy nagybetűje, ill. az aláhúzásjel értendő. Az azonosító karaktereinek száma nem lépheti túl a 127-et; minden karakter szignifikáns.

Számos azonosító előre definiált; ezek konstansokat, függvényeket és eljárásokat képviselnek. E csoportba a következő azonosítók tartoznak:

| | | | |
|---|---|---|---|
| **Abs** | **Delete** | **Keypressed** | **Read** |
| **Addr** | **Dispose** | **Length** | **Readln** |
| **ArcTan** | **Eof** | **Ln** | **Real** |
| **Assign** | **Eoln** | **Lo** | **Release** |
| **Aux** | **Erase** | **LowVideo** | **Rename** |
| **Bdos** | **Execute** | **Lst** | **Reset** |
| **Bios** | **Exit** | **Mark** | **Rewrite** |
| **BiosHL** | **Exp** | **MaxAvail** | **Round** |
| **BlockRead** | **False** | **MaxInt** | **Seek** |
| **BlockWrite** | **FilePos** | **Mem** | **Sin** |

| | | | |
|---|---|---|---|
| **Boolean** | **FileSize** | **MemAvail** | **SizeOf** |
| **BufLen** | **FillChar** | **Move** | **SeekEof** |
| **Chain** | **Frac** | **NormVideo** | **Sqr** |
| **Char** | **FreeMem** | **Odd** | **Sqrt** |
| **Chr** | **GetMem** | **Ord** | **Str** |
| **Close** | **GotoXY** | **Output** | **Succ** |
| **ClrEol** | **Halt** | **OvrDrive** | **Swap** |
| **ClrScr** | **HeapPtr** | **ParamCount** | **Text** |
| **Con** | **Hi** | **ParamStr** | **Trm** |
| **Concat** | **Input** | **Pi** | **True** |
| **Copy** | **Insert** | **Port** | **Trunc** |
| **Cos** | **InsLine** | **Pos** | **UpCase** |
| **CrtExit** | **Int** | **Pred** | **Usr** |
| **CrtInit** | **Integer** | **Ptr** | **Val** |
| **DelLine** | **IOresult** | **Random** | **Write** |
| **Delay** | **Kbd** | **Randomize** | **Writeln** |

Ha a listán szereplő azonosítók valamelyikét deklaráljuk, akkor ezzel a deklaráció érvényességi körén belül „eltakarjuk" az adott azonosító eredeti jelentését.

**Példa**

Az azonosító jelentésének „eltakarása":

```
program jb;
const
  false = true;
begin
  Writeln(false)
end.
```

- A programfuttatás eredményeképpen a TRUE felirat jelenik meg a képernyőn.
- Ha a programból kitörölnénk az első pontosvessző és a **begin** közötti szövegrészt, akkor futási eredményként a FALSE felirat jelenne meg.

## 2.3. Konstansok

A konstansok aritmetikai, karakter, karakterlánc és logikai konstansok lehetnek. Az aritmetikai konstansok tovább csoportosíthatók egész és valós típusú konstansokra. Az aritmetikai konstansokat a továbbiaknak számoknak fogjuk nevezni.

### 2.3.1. Egész típusú konstansok

Az egész típusú konstans decimális számjegyek sorozata, melyet a + vagy a — jel előzhet meg. Az egész típusú konstans számértékének a [—32768, 32767] zárt inter-

vallumba kell tartoznia. Azon egész típusú konstanst, amelynek számértéke a [0,255] zárt intervallumba tartozik, *byte* típusú konstansnak nevezzük.

A Turbo Pascal nyelvben az egész és a byte típusú konstansok hexadecimális (tizenhatos számrendszerbeli) számokkal is megadhatók. Ez esetben a konstans a $ jelből és az azt követő hexadecimális számjegyek sorozatából áll. A $ jel előtt szerepelhet a + vagy a — jel.

**Példa**

Néhány — helyes és hibás — egész típusú konstans:

Egész típusú konstansok: 23
—23
002
$ABC
—$2B

Olyan karaktersorozatok, amelyek nem képviselnek egész típusú konstansokat:

2B6 a B betű nem decimális számjegy;
$3G a G betű nem hexadecimális számjegy;
2.6 a . karakter nem számjegy.

## 2.3.2. Valós típusú konstansok

A valós típusú konstans egymás utáni összetevői a következők: egészrész, tizedespont, törtrész, kis vagy nagy E betű, kitevő. Az egészrész, a törtrész és a kitevő decimális számjegyek sorozata; az egészrészt és a kitevőt megelőzheti a + vagy a — jel. A törtrész (a tizedesponttal együtt) és a kitevő (az E betűvel együtt) elhagyható. Megengedhető továbbá, hogy a konstans törtrésze üres legyen.

**Példa**

Néhány — helyes és hibás — valós típusú konstans:

Valós típusú konstansok: —2.3
2.5e2
4.E—3
$ABC
2E3

Olyan karaktersorozatok, amelyek nem képviselnek valós típusú konstansokat:

3e2.0 a kitevőben nem lehet tizedespont;
.25 hiányzik az egészrész;
2.3D2 hibás kitevő.

### 2.3.3. Karakter, illetve karakterlánc típusú konstansok

A karakterlánc típusú konstansok karaktersorozatokat képviselnek. Ha a karakterlánc egyetlenegy karakterből áll, akkor karaktertípust képviselt. Általános esetben egy karakterlánc típusú konstans a következőkből állhat: két aposztróf közötti karaktersorozatból, vezérlő karakterekből, valamint decimálisan vagy hexadecimálisan ábrázolt karakterekből. A két aposztróf között levő karaktersorozatban tetszőleges látható karakterek (a szóközt is beleértve) fordulhatnak elő; az ' (aposztróf) karaktert pedig egy aposztrófpár képviseli. A vezérlő karaktereket egy karakterpár ábrázolja, ahol az első karakter egy ^, a második pedig egy látható karakter. Az így ábrázolt vezérlő karakter kódja azonos azzal a kóddal, amely a Ctrl billentyű és az adott látható karakter billentyűjének együttes lenyomásával keletkezik. A decimálisan ábrázolt karakterek a # karakterből és az utána következő decimális karakterkódból állnak, a hexadecimálisan ábrázolt karakterek pedig a # karakterből, a $ karakterből és az utána következő hexadecimális karakterkódból.

**Példa**

Néhány — helyes és hibás — karakterlánc típusú konstans:

Karakterlánc típusú konstansok:

| | |
|---|---|
| 'jb' | (két karakter) |
| '''' | (egy karakter) |
| ^G | (egy karakter) |
| #65 | (egy karakter) |
| #$41 | (egy karakter) |
| ^G'jb' | (három karakter) |
| ' ' | (két szóköz) |
| ^G^G^G'hello'#13#10'jan' | (13 karakter) |

Olyan karaktersorozatok, amelyek nem képviselnek karakterlánc típusú konstansokat:

| | |
|---|---|
| ''' | hiányzik az aposztróf; |
| #256 | 255-nél nagyobb kód; |
| #$4G | a G nem hexadecimális számjegy. |

### 2.3.4. Logikai konstansok

A logikai konstansokat a *true* és a *false* karaktersorozat képviseli. Közülük az első a logikai „igaz" értéket, a második pedig a logikai „hamis" értéket jelenti.

**Példa**

Néhány — helyes és hibás — logikai konstans:

Logikai konstansok: false
FALSE
True

Olyan karaktersorozatok, amelyek nem képviselnek logikai konstansokat:

igaz hibás konstans;
hamis hibás konstans.

## 2.4. Megjegyzések

A megjegyzés nyitó kapcsos zárójellel kezdődő és záró kapcsos zárójellel végződő karaktersorozat. A nyitó és záró kapcsos zárójelek helyett használhatók a ( * és *) összetett szimbólumok is. A { és } karakterekkel határolt megjegyzések beágyazhatók a ( * és *) szimbólumokkal határolt megjegyzésekbe; hasonlóképpen a ( * és *) szimbólumokkal határolt megjegyzések beágyazhatók a { és } karakterekkel határolt megjegyzésekbe.

Ha a megjegyzést megnyitó karakter vagy szimbólum után közvetlenül a $ karakter következik, akkor ezt a megjegyzést a fordítóprogram direktívának tekinti. A direktíva nem szerepelhet beágyazott megjegyzésben.

**Példa**

Néhány — helyes és hibás — megjegyzés és direktíva:

Megjegyzések: (*data*)
{data}
(*out (file)*)
Direktívák: (*$i include.set*)
{$v,b+}

Olyan karaktersorozatok, amelyek nem képviselnek megjegyzést:

(*inside} hibás határoló;
{{inside}} hibás beágyazás.

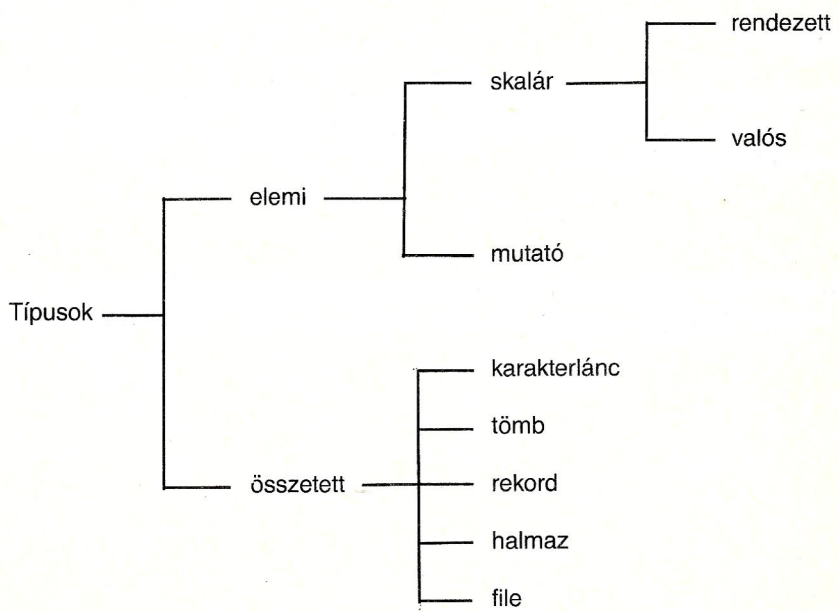
Olyan karaktersorozatok, amelyek nem képviselnek direktívát:

{$ v—} szóköz a { és a $ között;
{$ v—} szóköz a $ és a v— között.

# 3. Standard skalártípusok

A típus fogalma egy adathalmazhoz kapcsolódik. Ilyen értelemben egy változó bizonyos típusú, ha e típust leíró halmaz elemeit értékéül kaphatja. A Turbo Pascal nyelv megköveteli a program összes változójának nyílt típusmeghatározását.

A változó nevét és a kiválasztott típust a változó deklarációjában kapcsolhatjuk össze. A változótípusok az 1. ábrán látható elemi és összetett típusokra oszthatók fel. Elemi típus a skalár- és a mutatótípus; az összetett típusokhoz a karakterlánc-, a tömb-, a rekord-, a halmaz- és a file-típus tartozik. A skalártípusok között megkülönböztethetünk standard skalártípusokat, mégpedig *integer* (egész-), *byte, char* (karakter-), *boolean* (logikai) és *real* (valós) típusokat. Az első négyet rendezett típusnak is nevezzük. Jellemző tulajdonságuk, hogy mindegyikük egy megszámlálható halmazhoz kapcsolódik.




*1. ábra. A változótípusok osztályozása*

A standard skalártípusok előre definiáltak. „Rejtett" definíciójuk érvényességi tartománya az egész programra kiterjed, kivéve azokat a részeket, amelyekben nyílt eltakaró deklarációk vannak.

### *Az integer típus*

Az *integer* típusú adat az egész számok részhalmazának az eleme. E számok értékei a [-32768,32767] zárt intervallumhoz tartoznak. Az *integer* típusú adatok mindegyike 2 bájt tárhelyet foglal le.

Az *integer* típusú adatokon végzett aritmetikai műveletekben nincs túlcsordulásellenőrzés. Emiatt az eredmény helyességének az a feltétele, hogy a művelet argumentumai és az összes közbenső eredmény az előbb meghatározott intervallumhoz tartozzék.

### *A byte típus*

A *byte* típusú adat egész számok részhalmazának az eleme, értékei a [0,255] zárt intervallumhoz tartoznak. Ahol a programban egy *byte* típusú adatra hivatkozunk, majdnem mindenhol hivatkozhatunk egy *integer* típusú adatra (és fordítva). Fontos kivétel e szabály alól a paraméterek és argumentumok társítása az eljárásokban és a függvényekben; ekkor az adattípusok teljes megfeleltetése a követelmény. A *byte* típusú adatok mindegyike 1 bájt tárhelyet foglal le.

### *A char típus*

A *char* típusú adat az ASCII kódkészlet eleme. A *char* típusú adatok mindegyike 1 bájt tárhelyet foglal le.

### *A boolean típus*

A *boolean* típusú adat a kételemű logikai adathalmaz eleme. Ezeket az adatokat a *false* és a *true* szabványos azonosítókkal jelölhetjük. A *boolean* típusú adatok mindegyike 1 bájt tárhelyet foglal le.

### *A real típus*

A *real* típusú adat valós számhalmaz eleme. E halmazhoz tartozik a 0 szám, ill. azok a pozitív és negatív számok, melyek abszolút értékei a $[10^{-38}, 10^{38}]$ intervallumhoz tartoznak.

# 4. Programszerkezet

A Turbo Pascal nyelvben megírt program programfejlécből, blokkból és . (pont) karakterből áll. A fejlécben a program neve és egy kerek zárójelekkel határolt azonosítólista van, amely leírja a program és környezete közötti adatátvitel módját. A blokk egy deklarációs és egy végrehajtó részből áll. A fejléc és a deklarációs rész el is hagyható.

## 4.1. Programfejléc

Ha a program fejléccel kezdődik, akkor szerepel benne legalább a program neve. A programnév után megadható — kerek zárójelekkel határolva — a program paraméterlistája. Ez a lista felsorolja azon adatállományok azonosítóit, melyekkel a program kapcsolatot tart környezetével. A fordítóprogram nem értelmezi a lista tartalmát, ezért figyelmen kívül hagyja (a határoló zárójelekkel együtt).

*Szintaxis*

programfejléc:
  **program** programnév (állománynévlista)
programnév:
  azonosító
állománynév:
  azonosító

**Példák**

```
program Volume
program Lister(Output);
program Copier(Input,Output);
```

## 4.2. Blokk

A blokk egy deklarációs és egy végrehajtó részből áll. A deklarációs részben vannak a típusdefiníciók, a konstansnév-definíciók, az alprogramok (eljárások és függvények) definíciói és deklarációi, valamint a címkék és a változók deklarációi. Mindezek a de-

finíciók és deklarációk tetszőleges sorrendben következhetnek; az egyes definíciók és deklarációk csoportjai keveredhetnek egymással. Ezt úgy kell érteni, hogy pl. a típusdefiníciók után írhatjuk a változók deklarációit, utánuk újból a típusdefiníciókat és a változódeklarációkat stb. A végrehajtó rész egy csoportosító utasításból áll, amely a program utasításait tartalmazza.

*Szintaxis*

blokk:
  deklarációs_ rész végrehajtó_ rész
deklarációs_ rész:
  a_ deklarációk_ és_ a_ definíciók_ felsorolása
végrehajtó_ rész:
  csoportosító_ utasítás
deklarációk_ és_ definíciók:
  címkedeklarációk
  konstansnév-definíciók
  típusdefiníciók
  változódeklarációk
alprogram_definíciók_ és_ -deklarációk
  inicializálások

### 4.2.1. Címkedeklarációk

A program minden utasítását címkével láthatjuk el, amely vezérlésátadást tesz lehetővé a **goto** utasítás használatakor. A címke egy azonosítóból vagy egy számjegysorozatból áll. A közvetlenül a címke után következő : karakter a címkét elválasztja egy másik címkétől vagy utasítástól. Mielőtt címkét használnánk a programban, deklarálnunk kell a nevét. A címkedeklarációk a **label** kulcsszóból és az utána következő — a címkenevekkel azonos — karaktersorozatok listájából állnak, végül a ; következik.

*Szintaxis*

címkedeklarációk:
  **label** címkelista;

**Példák**

**label** 10, abort, quit;
**label** 9999;

### 4.2.2. Konstansnév-definíciók

A program áttekinthetősége szempontjából gyakran hasznos, ha a konstansokat az őket képviselő, megfelelően megválasztott azonosítókkal helyettesítjük. Ilyen esetekben az azonosító azonos a vele összekapcsolt konstanssal. A Turbo Pascal nyelvben

az azonosítókat és a konstansokat a konstansnév-definíciók értelmezésekor kapcsolhatjuk össze. A konstansnév-definíció a **const** kulcsszóból és az utána következő hozzárendelések sorozatából áll. A hozzárendelés az azonosítóból, az = karakterből, a konstanskifejezésből és a ; karakterből áll. A konstanskifejezés lehet szám, karakterlánc, konstansnév vagy — (mínusz) karakterrel megelőzött konstansnév. Az előre definiált konstansnevek a következők:

*Pi* *real* típusú, a 3.1415926536 konstanst képviseli;
*MaxInt* *integer* típusú, a 32767 konstanst képviseli.

*Szintaxis*

konstansnév-definíciók
  **const** hozzárendelések_ sorozata
hozzárendelés:
  azonosító = szám;
  azonosító = karakterlánc;
  azonosító = konstansnév;
  azonosító = mínuszjel konstansnév;

**Példák**

```
const e = 2.718282;
      g = 9.81;
const TwoChar = '2'
      Two = 2;
      Minus2 = —Two;
const First Name = 'janek'
```

### 4.2.3. Típusdefiníciók

A program változóinak deklarálásakor meghatározzuk a változók típusait. A változó típusa lehet előre definiált vagy standard típus (pl. *boolean* vagy *real*), de külön is meghatározható egy típusdefinícióban, amely általános esetben a **type** kulcsszóból és a típusdefiníciók sorozatából áll. A típusdefiníciók mindegyike a definiált típus azonosítójából, az = karakterből, a típusleírásból és a ; karakterből áll.

*Szintaxis*

típusdefiníciók:
  **type** típusdefiníciók_ sorozata
típusdefiníció:
  definiált_ típus = típusleírás;
definiált_ típus:
  azonosító
típusleírás:
  a_ standard_ típus_ leírása
  a_ definiált_ típus_ leírása

a_ standard_ típus_ leírása:
standardtípus-azonosító
a_ definiált_ típus_ leírása:
felsorolásitípus-leírás
intervallumtípus-leírás
karakterlánctípus-leírás
tömbtípus-leírás
rekordtípus-leírás
halmaztípus-leírás
a file-típus_ leírása
mutatótípus-leírás

**Példák**

```
type
  Number = integer;
  Color = (Red,Green,Blue,Orange);
  RGB   = Red..Blue;
  Name = string[20];
  Matrix = array[1..5,1..5] of real;
  Person = record
             FirstName : string[6];
             LastName : Name;
             Salary : real;
           end;
  Digits = set of 0..9;
  DataFile = file of Person;
  Ref = Matrix;
```

### 4.2.4 Változódeklarációk

A program — egyszerű és összetett — változóját használat előtt deklarálni kell. A változódeklarációk a **var** kulcsszóból és a deklarációk azt követő felsorolásából állnak. A deklarációfelsorolás az azonosítók listájából, a : karakterből, a típusmeghatározásból és a ; karakterből tevődik össze. A deklaráció értelmezése a megadott típusú változó(k) létrehozását eredményezi.

*Szintaxis*

változódeklarációk:
**var** deklarációk_ sorozata
deklaráció:
azonosítók_ listája : típusmeghatározás;

**Példák**

```
var
  Length, Area, Volume : real;
  Count : integer;
  Buffer : array[0..255] of byte;
  Rejected : boolean;
```

Egy változó láthatósági tartománya (hatásköre) az a blokk, amelyikben a változót deklaráltuk. Ha ez a blokk más blokkokat is tartalmaz, melyekben ugyanilyen azonosítóval deklarált változók szerepelnek, akkor a deklaráció érvényességi tartománya kisebb, mint a láthatósági tartománya, és nem vonatkozik azokra a belső blokkokra, ahol az eltakaró deklarációk előfordultak. A változó láthatósági tartománya nem a blokk elején, hanem azonosítójának deklarálási helyén kezdődik.

**Példa**

A változó láthatósági és érvényességi tartománya:

```
                                  Láthatóság      Érvényesség
program Tricky;                   1               1
var                               1               1
  Toggle : false[1]..true;        1               1
  false[2] : integer;             1  2               2
begin                             1  2               2
  false:=5;                       1  2               2
  Writeln(false)                  1  2               2
end.
```

- Az előre definiált *false* azonosító érvényességi tartománya kisebb, mint a láthatósági tartománya (a nyíltan deklarált — *integer* típusú — *false* azonosító hatáskörével).
- A nyíltan deklarált *false* azonosító láthatósági és érvényességi tartománya azonos.
- Mivel a *Writeln* eljárást a nyíltan deklarált *false* azonosító érvényességi tartományán belül hívjuk, a program végrehajtásakor az 5 íródik ki.

### 4.2.5. Alprogram-deklarációk

Az alprogramok eljárásokra és függvényekre oszthatók. Az alprogram deklarációja azokat a tevékenységeket határozza meg, amelyek az alprogram hívása során végrehajtódnak. A függvénydeklaráció meghatározza a hívás helyén átadott eredmény típusát is. Az alprogram törzsét tartalmazó deklarációkat definícióknak nevezzük; azokat a deklarációkat, amelyekben a fejléc után a **forward** kulcsszó következik, előzetes deklarációknak hívjuk.

Az előzetes deklarációt csak akkor kell használni, ha a deklarációkat nem lehet úgy elrendezni, hogy az alprogramokat deklarációjuk hatáskörében hívjuk.

**Példa**

Eljárások kereszthívása:

```
program jb;
  ...
  procedure second; forward;
  procedure first;
  begin
    ...
    second;
  end;
  procedure second;
  begin
    ...
    first;
  end;
begin
  ...
end.
```

- Mivel a first eljárás deklarációjában

```
procedure first;
begin
  ...
  second;
end;
```

benne van a *second* eljárás hívása, a *second* eljárás deklarációjában pedig a *first* eljárásé, a *second* eljárást előzetesen deklarálni kell:

**procedure** *second;* **forward;**

- A definiáló deklarációval ellentétben az előzetes deklaráció csupán „bejelenti" az alprogram azonosítóját (és paramétereit, ha vannak).

# 5. Kifejezések

Azokat a karaktersorozatokat, amelyek az adat kiszámításához (ill. meghatározásához) végrehajtandó tevékenységeket határozzák meg, kifejezéseknek nevezzük. E tevékenységek végrehajtását a kifejezés kiértékelésének hívjuk. Mivel a kifejezés kiértékelésének az eredménye egy adat, magát a kifejezést az adatot képviselő karaktersorozatnak tekinthetjük. E fejezetben tárgyaljuk azon kifejezések kiértékelési szabályait, amelyek összetevői *integer*, *real*, *char* és *boolean* típusú adatokat képviselnek. A definiált típusú adatokra hivatkozó kifejezésekkel a megfelelő típusok leírásánál foglalkozunk majd.

## 5.1. Operátorok

Az operátorok egy- vagy kétargumentumosak lehetnek, de elsőbbségük (precedenciájuk) szerint is osztályozhatók. Csökkenő precedencia szerint a következőképpen osztályozhatjuk őket:

- Egyargumentumú előjelváltó operátor:
  —
- Egyargumentumú tagadó operátor (negáció):
  **not**
- Kétargumentumú, szorzás jellegű operátor:
  * / **div mod and shl shr**
- Kétargumentumú, összeadás jellegű operátor:
  \+ — **or xor**
- Kétargumentumú relációs operátor:
  = <> < > <= >= **in**

Ha egy alkifejezésben két, azonos elsőbbségű operátor van, akkor a megfelelő műveletek balról jobbra hajtódnak végre. Legelőször a kerek zárójelekkel körülvett alkifejezések értékelődnek ki. Ilyen értelemben a kerek zárójelek egyargumentumú, legnagyobb precedenciájú operátornak tekinthetők.

Ha a szorzás, ill. az összeadás jellegű műveletek argumentumai *integer* vagy *real* típusú adatokat képviselő kifejezések, akkor a művelet eredménye csak akkor *integer* típusú, ha mindkét argumentum *integer* típusú és a művelet nem osztás. Egyébként a művelet eredménye *real* típusú.

**Példa**

A 2 + 3 / 4 kifejezés által képviselt adat kiértékelése:

- Mivel az osztás elsőbbsége nagyobb az összeadásénál, a kifejezés azonos a
  2 + (3 / 4) kifejezéssel, de nem azonos a
  (2 + 3) / 4 kifejezéssel.
- A 3/4 kifejezés kiértékelésének eredménye egy *real* típusú adat, amelynek értéke — közelítőleg — azonos a 0.75 konstans értékével.
- A teljes kifejezés egy *real* típusú adatot képvisel, amelynek értéke — közelítőleg — azonos a 2.75 konstans értékével.

## 5.2. Előjelváltó operátor

Az egyargumentumú — (mínusz) művelet végrehajtása ellenkezőjére változtatja meg az adat előjelét. Ha az adat értéke 0, akkor a művelet eredménye szintén 0 értékű adat. A — (mínusz) művelet argumentumai csak *real* és *integer* típusú adatok lehetnek.

**Példák**

| *Kifejezés* | *Eredmény* |
|---|---|
| —(—4) | 4 |
| —3/2 | —1.5 |

## 5.3. Negációs operátor

Az egyargumentumú **not** művelet végrehajtása kizárólag a *boolean* és az *integer* típusú adatokra vonatkozik. Az első esetben egy *false* értékű adat *true* eredményt ad és fordítva; a második esetben az adatreprezentáció minden bitje ellenkező értékűre változik.

**Példák** (vö. a 20. fejezettel)

| *Kifejezés* | *Eredmény* |
|---|---|
| **not** false | true |
| **not** true | false |
| **not** 0 | —1 |
| **not** $FFFF | 0 |
| **not** —2 | 1 |
| **not** —32768 | 32767 |

## 5.4. Szorzás jellegű operátorok

A kétargumentumú *(szorzás) és / (osztás) operátor a *real* és az *integer* típusú adatokra vonatkozik, a kétargumentumú **and** operátor a *boolean* és az *integer* típusú adatokra, a többi, szorzás jellegű operátor pedig kizárólag az integer típusú adatokra. E követelményeket és az eredmény típusát az *1. táblázat* foglalja össze.
A *és a / műveletek végrehajtását nem szükséges magyarázni. Jegyezzük meg, hogy az osztás eredménye mindig real típusú adat.

*1. táblázat. Szorzás jellegű operátorok*

| Operátor | 1. argumentum | 2. argumentum | Eredmény |
|---|---|---|---|
| * | real | real | real |
| | real | integer | real |
| | integer | real | real |
| | integer | integer | integer |
| / | real | real | real |
| | real | integer | real |
| | integer | real | real |
| | integer | integer | real |
| **div** | integer | integer | integer |
| **mod** | integer | integer | integer |
| **and** | integer | integer | integer |
| | boolean | boolean | boolean |
| **shl** | integer | integer | integer |
| **shr** | integer | integer | integer |

**Példák**

| *Kifejezés* | *Eredmény* |
|---|---|
| 25 *40 | 1000 |
| 25 *40.0 | 1000.0 |
| 0.25 *40.0 | 10.0 |
| 24 / 3 | 8.0 |
| 24.0 / 3.0 | 8.0 |

A **div** művelet eredménye *integer* típusú adat, melynek értéke egyenlő a két argumentum osztási eredményének egészrészével. A **mod** művelet definíciója a következő:

a **mod** b = a — ( ( a **div** b ) *b )

**Példák**

| *Kifejezés* | *Eredmény* |
|---|---|
| 25 **div** 7 | 3 |
| −25 **div** 7 | −3 |
| 25 **mod** 7 | 4 |
| −25 **mod** 7 | −4 |

Az **and** művelet eredménye két *boolean* típusú adaton az argumentumok logikai szorzata:

| *a argumentum* | *b argumentum* | *a and b* |
|---|---|---|
| *false* | *false* | *false* |
| *true* | *false* | *false* |
| *false* | *true* | *false* |
| *true* | *true* | *true* |

Az **and** művelet eredménye két *integer* típusú adaton olyan *integer* típusú adat, amelynek bitjei az argumentumok megfelelő bitpárjának logikai szorzatával egyenlők. Egy bitpár logikai szorzata csak akkor 1, ha mindkét bit 1, különben 0.

**Példák**

| *Kifejezés* | *Eredmény* |
|---|---|
| false **and** true | false |
| 2 **and** 2 | 2 |
| 12 **and** 22 | 4 |
| −5 **and** 0 | 0 |

A kétargumentumú **shl** és **shr** művelet kizárólag az *integer* típusú adatokra vonatkozik. A műveletek eredménye is *integer* típusú. Az eredmény bitképe úgy keletkezik, hogy az első argumentum bitjeit annyi pozícióval balra (**shl**) vagy jobbra (**shr**) toljuk el, amennyi a második argumentum értéke. Az eltolás logikai jellegű, vagyis az előjelbit nem sokszorozódik, ugyanakkor az üresedő bitpozíciók 0 értékű bitekkel töltődnek fel.

**Példák**

| *Kifejezés* | *Eredmény* |
|---|---|
| 2 **shl** 1 | 4 |
| 2 **shl** 3 | 16 |
| 3 **shr** 1 | 1 |
| −3 **shr** 1 | 32766 |

## 5.5. Összeadás jellegű operátorok

A kétargumentumú + és — operátor a *real* és az *integer* típusú adatokra vonatkozik, a többi, összeadás jellegű operátor pedig az azonos típusú argumentumpárokra. E követelményeket és az eredmény típusát a *2. táblázat* foglalja össze.

*2. táblázat. Összeadás jellegű operátorok*

| Operátor | 1. argumentum | 2. argumentum | Eredmény |
|---|---|---|---|
| + | real | real | real |
| | real | integer | real |
| | integer | real | real |
| | integer | integer | integer |
| — | real | real | real |
| | real | integer | real |
| | integer | real | real |
| | integer | integer | integer |
| **or** | integer | integer | integer |
| | boolean | boolean | boolean |
| **xor** | integer | integer | integer |
| | boolean | boolean | boolean |

A + és a — műveletek végrehajtását nem szükséges magyarázni. Jegyezzük meg, hogy a művelet eredménye csak akkor *integer* típusú, ha mindkét argumentum *integer* típusú. Egyébként a művelet eredménye *real* típusú.

**Példák**

| *Kifejezés* | *Eredmény* |
|---|---|
| 20+40 | 60 |
| 20.0+40 | 60.0 |
| 60.0—40.0 | 20.0 |

Az **or** művelet eredménye két *boolean* típusú adaton az argumentumok logikai összege:

| *a argumentum* | *b argumentum* | *a or b* |
|---|---|---|
| *false* | *false* | *false* |
| *true* | *false* | *true* |
| *false* | *true* | *true* |
| *true* | *true* | *true* |

Az **or** művelet eredménye két *integer* típusú adaton olyan *integer* típusú adat, amelynek bitjei az argumentumok megfelelő bitpárjainak logikai összegével egyenlők. Egy bitpár logikai összege csak akkor 0, ha mindkét bit 0, különben 1.

**Példák**

| *Kifejezés* | *Eredmény* |
|---|---|
| false **or** true | true |
| 2 **or** 3 | 3 |
| 12 **or** 22 | 30 |
| $8000 **or** 1 | −32767 |

A **xor** művelet eredménye két *boolean* típusú adaton az argumentumok logikai „kizáró VAGY" kapcsolata:

| *a argumentum* | *b argumentum* | *a xor b* |
|---|---|---|
| *false* | *fal* | *false* |
| *true* | *false* | *true* |
| *false* | *true* | *true* |
| *true* | *true* | *false* |

A **xor** művelet eredménye két *integer* típusú adaton olyan *integer* típusú adat, amelynek bitjei az argumentumok megfelelő bitpárjainak modulo 2 összegével egyenlők. Egy bitpár modulo 2 összege csak akkor 0, ha mindkét bit azonos, különben 1.

**Példa**

| *Kifejezés* | *Eredmény* |
|---|---|
| true **xor** true | false |
| $ 8000 **xor** 2 | 32766 |
| 12 **xor** 22 | 26 |
| −1 **xor** −1 | 0 |

## 5.6. Relációk

A relációk operátorai tetszőleges *integer*, *real*, *char* és *boolean* típusú adatokra vonatkozhatnak. A műveletek eredménye mindig egy *boolean* típusú adat, melynek értéke *false* vagy *true.* Ha a reláció egyik argumentuma *boolean* vagy *char* típusú, akkor a másik argumentumnak ezzel azonos típusúnak kell lennie. Aritmetikai argumentumok esetén nem követelmény a típusazonosság. Ez lehetővé teszi pl. a *real* és az *integer* típusú adatok összehasonlítását.

A relációk operátorai a következők: = (egyenlő), <> (nem egyenlő), < (kisebb), > (nagyobb), <= (kisebb vagy egyenlő), >= (nagyobb vagy egyenlő).

**Példák**

| *Kifejezés* | *Eredmény* |
|---|---|
| 20 = 20 | true |
| 30.0 > 20 | true |
| 'J' < 'B' | false |
| false < true | true |

## 5.7. Függvényhívások

A függvényhívás a függvény azonosítójából és az azt követő, kerek zárójelekkel körülvett hívási argumentumok listájából áll. A paraméter nélküli függvény hívása csak a függvény azonosítójából áll.

**Példák**

*Eof(Input)*
*KeyPressed*

# 6. Utasítások

Az utasítás azon tevékenységek leírása, amelyeket végre kell hajtani az algoritmus megvalósításakor. A Turbo Pascal nyelvben két utasításfajta van: egyszerű és strukturált. Egyszerű utasítás az értékadás, az eljáráshívás, az ugró utasítás és az üres utasítás; strukturált utasítás pedig a csoportosító utasítás, a feltételes utasítás, a kiválasztó utasítás és három iterációs utasítás. A programban szereplő minden szomszédos utasításpárt a ; karakter választja el.

## 6.1. Értékadó utasítás

Az értékadó utasítás a változónévből, az értékadó szimbólumból és a kifejezésből áll. Egy függvénydefiníción belül szerepelnie kell legalább egy értékadó utasításnak, amelyikben a változót azonosító karaktersorozat a függvény azonosítója.

Az értékadó utasítás végrehajtása során kiértékelődik a kifejezés képviselte adat, és ennek értékét megkapja az értékadó szimbólum bal oldalán álló azonosítóval meghatározott változó. A függvénydefiníción belüli értékadás a függvény értékét adja meg (vagyis azt az adatot, amely hozzáférhetővé válik a függvényhívás helyén).

Az értékadáskor a kifejezés típusának a változó típusával azonosnak kell lennie. Ez a típusazonosság akkor is fennáll, ha a változó *real*, a kifejezés pedig *integer* típusú.

*Megjegyzés:* a tömb, a rekord és a halmaz típusú adataggregátumban megengedett az értékadás, a file típusú változóban nem.

*Szintaxis*

értékadó_ utasítás:
változónév := kifejezés
függvényazonosító := kifejezés

**Példák**

```
i:=i+2
x1:=(-b+sqrt(b*b - 4*a*c)) / (2*a)
arr[2,3]:= m = n
fun:=false
```

## 6.2. Eljáráshívás

Az eljáráshívás az eljárás azonosítójából és az azt követő, kerek zárójelekkel körülvett hívási argumentumok listájából áll, a paraméter nélküli eljárás hívása pedig csak az eljárás azonosítójából.

Az eljárás hívásakor végrehajtódnak az eljárás definíciójában meghatározott tevékenységek. Előtte azonban lezajlik a hívott eljárás argumentumainak meghatározása és a hívásban szereplő kifejezések kiértékelése.

*Szintaxis*

eljáráshívás:
  eljárásnév (argumentumlista)
  eljárásnév
eljárásnév:
  azonosító

**Példák**

Rewrite(OutFile)
Move(source,target)
Exit

## 6.3. Ugró utasítás

Az ugró utasítás (vezérlésátadás) a **goto** kulcsszóból és az azt követő címkenévből áll. Az ugró utasítás végrehajtása eredményeképpen a program a címkével ellátott utasítással (és az utána következő utasítássorozattal) folytatódik. Ennek feltétele, hogy egy bizonyos címkenevet tartalmazó ugró utasítás e címkedeklaráció hatáskörén belül legyen. Nem megengedett, hogy az ugró utasítás végrehajtása a vezérlést egy strukturált utasítás vagy egy alprogram belsejébe, ill. az alprogramon kívülre adja át.

*Szintaxis*

ugró_ utasítás:
  **goto** címkenév
címkenév:
  azonosító
  számjegysorozat

**Példák**

**goto** 345
**goto** finish

## 6.4. Üres utasítás

Az üres utasítás a nyelv egyetlen olyan szerkezete, amely nem igényli a nyelvi szimbólumok használatát. Akkor alkalmazzuk, amikor a programon belül szükség van egy utasításra, de semmilyen tevékenységet nem akarunk végeztetni vele.

Az üres utasítás végrehajtásának nincs következménye.

*Szintaxis*

üres_ utasítás:
  üres
üres:

**Példák**

Az üres utasítás helyét egy felkiáltójelet tartalmazó megjegyzés jelöli:

```
begin {!}end
while false do {!};
repeat {!} until true
begin {!} ; {!} end
case true of false: {!} ; true {!} end
```

## 6.5. Csoportosító utasítás

A csoportosító utasítás (röviden csoportos utasítás)* a **begin** kulcsszóból, az utasítások sorozatából és az **end** kulcsszóból áll. A csoportosító utasítást ott használjuk, ahol a nyelv szintaxisa csak egy utasítást enged meg, itt viszont utasítássorozatra van szükség.

A csoportosító utasítás végrehajtásakor végrehajtódik a benne foglalt utasítássorozat.

*Szintaxis*

csoportosító_ utasítás:
  **begin** utasítások_ sorozata **end**

**Példák**

```
begin
  i:=2; j:=2
end
```

* A csoportosító utasítást számos könyvben összetett utasításnak hívják, holott ez a nyelv egyik legegyszerűbb szerkezete.

```
begin
   Writeln(OutFile);
   Close(OutFile)*
end
```

## 6.6. Feltételes utasítás

A feltételes utasítás az **if** kulcsszóból, az azt követő logikai kifejezésből, a **then** kulcsszóból és egy utasításból áll. Az utasítás után az **else** kulcsszó és egy újabb utasítás is következhet.

A feltételes utasítás végrehajtása a logikai kifejezés kiértékelésével kezdődik; ha az eredmény *true,* akkor a **then** kulcsszó utáni utasítás hajtódik végre. Ha az eredmény *false* és az utasítás nem tartalmazza az **else** kulcsszót, akkor az utasítás végrehajtását befejezettnek tekintjük; ellenkező esetben az **else** kulcsszó utáni utasítás hajtódik végre.

Ha a feltételes utasításban szereplő utasítás is feltételes, akkor csak az **else** kulcsszó előtt állhat (feltéve, hogy ez az utasítás is tartalmazza az **else** kulcsszót).

*Szintaxis*

feltételes_utasítás:
**if** logikai_kifejezés **then** utasítás
**if** logikai_kifejezés **then** utasítás
**else** utasítás

**Példák**

```
if a = b then a := 5 else b := 5
if a = b then begin
                 a := 6;
                 b := 6
              end
if a = b then
   if a = 5 then
      b:=2
   else
      a:= 3
else
   a:= 13
```

* A lektor megjegyzése: az end-et megelőző utasítást nem kell pontosvesszővel zárni, de a fordító nem jelez hibát, ha kitesszük.

## 6.7. Kiválasztó utasítás

A kiválasztó utasítás a **case** kulcsszóból, az azt követő kiválasztó (szelektor-) kifejezésből, az **of** kulcsszóból, a kiválasztási címkékkel ellátott utasítások sorozatából és az **end** kulcsszóból áll.

A kiválasztási címkék listája konstansokból épül fel; a listát egy : karakter választja el az utasítástól. A listában szereplő mindegyik konstans típusának azonosnak kell lennie a kiválasztó kifejezés típusával. Ha a konstanslistán belül rendezett típusú, egymás utáni elemek sorozata jelenik meg, akkor a sorozat az

első..utolsó

szerkezettel helyettesíthető, mely meghatározza a sorozat első és utolsó elemét. A kiválasztó utasítás utolsó ágában (az **end** kulcsszó előtt) szerepelhet az **else** kulcsszó is (kettőspont nélkül).

*Szintaxis*

kiválasztó_utasítás:
**case** kiválasztó_kifejezés **of**
kiválasztó_utasítások_sorozata **end**

A kiválasztó utasítás végrehajtása a kiválasztó kifejezés kiértékelésével kezdődik; utána az az utasítás hajtódik végre, amelyiknek a kiválasztási címkéje egyenlő a kiválasztó kifejezés értékével. Ha nincs ilyen utasítás, akkor az **else** kulcsszó utáni utasítások hajtódnak végre; ha hiányzik az **else** ág, akkor egy üres utasítás hajtódik végre.

**Példák**

```
case a = b of
  false: i := 5;
  true: j : = 2;
end
case Year of
  1945..1955 : Writeln('hard work');
  1956..1969 : Writeln('hopes');
  1970..1979 : Writeln('prosperity');
  1980..1981 : Writeln('euphoria');
  else         Writeln('no comment')
end
```

## 6.8. Iterációs utasítások

Az iterációs utasításokkal programozott ciklusok szervezhetők. Három fajtájuk van: a *for*, a *while* és a *repeat.*

### *A for utasítás*

A *for* utasítás a **for** kulcsszóból, az azt követő ciklusváltozó azonosítójából, az értékadás-szimbólumból, a ciklusváltozó kezdeti értékét meghatározó kifejezésből, a **to** vagy a **downto** kulcsszóból, a ciklusváltozó végértékét meghatározó kifejezésből, a **do** kulcsszóból és egy tetszőleges utasításból áll.

A ciklusváltozónak és mindkét kifejezésnek rendezett típusúnak kell lennie.

*Szintaxis*

for_ utasítás:
**for** változónév := kifejezés$_a$ **to** kifejezés$_b$ **do** utasítás
**for** változónév := kifejezés$_a$ **downto** kifejezés$_b$ **do** utasítás

A *for* utasítás végrehajtásakor a ciklusban szereplő utasítások a ciklusváltozó összes olyan értékére végrehajtódnak, melyek a „kifejezés$_a$" és a „kifejezés$_b$" által határolt zárt intervallumhoz tartoznak.

Ha a *for* utasításban a **to** kulcsszó szerepel, akkor a ciklusváltozó növekvő sorrendben vesz fel értékeket (a legkisebbtől a legnagyobbig). Ha a **downto** kulcsszó jelenik meg, akkor a ciklusváltozó csökkenő sorrendben vesz fel értékeket (a legnagyobbtól a legkisebbig). Ha az első esetben igaz a

kifejezés$_a$ > kifejezés$_b$

reláció, vagy a második esetben igaz a

kifejezés$_a$ < kifejezés$_b$

reláció, akkor a *for* utasítással meghatározott ciklus egyáltalán nem hajtódik végre. Ilyen esetben a *for* utasítás végrehajtása csupán a két kifejezés kiértékelését jelenti.

Egyéb esetben a ciklus legalább egyszer végrehajtódik; befejezés után a ciklusváltozó értéke egyenlő a „kifejezés$_b$" értékével.

*Megjegyzés:* az említett kifejezések csak egyszer értékelődnek ki (a ciklusba való belépéskor). A ciklusváltozónak nyíltan nem lehet értéket adni.

**Példák**

```
for i:=2 to 8 do Writeln(i)
for j:=8 downto 2 do Writeln(j)
for toggle:=false to true do
  for letter:='i' to 'n' do
    arr[toggle,letter] := 2
```

### *A while utasítás*

A *while* utasítás a **while** kulcsszóból, az azt követő *boolean* típusú változóból, a **do** kulcsszóból és egy tetszőleges utasításból áll.

*Szintaxis*

while_ utasítás:
**while** kifejezés **do** utasítás

A *while* utasítás a következő algoritmus szerint hajtódik végre:

1. Először a kifejezés értékelődik ki.
2. Ha a kifejezés értéke *false,* akkor az utasítás végrehajtása befejeződött.
3. Ha a kifejezés értéke *true,* akkor a **do** kulcsszó utáni utasítás hajtódik végre; ezután a leírt tevékenységek az *1.* ponttól kezdve ismétlődnek.

**Példák**

```
while i <>0 do begin
  i := i—1;
  Writeln(i)
end

while Flag do Fun(Flag)
```

### *A repeat utasítás*

A *repeat* utasítás a **repeat** kulcsszóból, az azt követő tetszőleges utasítások sorozatából, az **until** kulcsszóból és egy *boolean* típusú kifejezésből áll.

*Szintaxis*

repeat_ utasítás:
**repeat** utasítások **until** kifejezés

A *repeat* utasítás a következő algoritmus szerint hajtódik végre:

1. Először a **repeat** kulcsszó utáni utasítások hajtódnak végre.
2. Kiértékelődik az **until** kulcsszó utáni kifejezés.
3. Ha a kifejezés értéke *true,* akkor az utasítás végrehajtása befejeződött.
4. Ha a kifejezés értéke *false,* akkor a leírt tevékenységek az 1. ponttól kezdve ismétlődnek.

**Példák**

```
repeat
  Writeln(i);
  i := i — 1
until i = 0

repeat until KeyPressed
```

# 7. Felsorolási és intervallumtípusok

A Turbo Pascal elemi adattípusai a skalártípusok, ezen belül pedig a rendezett típusok. A rendezett típusok mindegyikének fontos tulajdonsága, hogy az őt definiáló halmaz megszámlálható.

*Megjegyzés:* a *real* típus — annak ellenére, hogy skalártípus — nem rendezett típus. Ez az oka annak, hogy a *case* utasítás szelektorkifejezése és címkéi, valamint a vezérlőkifejezések, ill. a *for* utasításban szereplő kifejezések nem lehetnek *real* típusúak.

## 7.1. Felsorolási típusok

A rendezett típusok közül az egyik legfontosabb a felsorolási típus. A felsorolási típusok mindegyike kis elemszámú halmazhoz kapcsolódik, amelynek elemein nem hajtunk végre aritmetikai műveleteket. E halmazok egyes elemeit egyedi azonosítókkal jelöljük, rendezettségük a felsorolási típus leírásában szereplő azonosítók sorrendjén alapul.

A felsorolási típus leírása az adott típus elemazonosítóinak listájából és az azt körülvevő kerek zárójelekből áll. Az azonosítók mindegyike egyben az adott típus konstansa is. Lényeges a lista azonosítóinak felsorolási sorrendje.

*Szintaxis*

felsorolásitípus_ leírás:
(az_ azonosítók_ listája)

**Példák**

```
type
  Day = (Mon,Tue,Wed,Thu,Fri,Sat,Sun);
  Color = (Heart,Diamond,Spade,Club);
  RGB = (Red,Green,Blue);
```

- A *Day* típusú adatokat a hét napját jelölő hárombetűs nevek;
- a *Color* típusú adatokat a kártyaszín szimbolikus nevei (*Heart*, *Diamond*, *Spade* és *Club);*
- az *RGB* típusú adatokat pedig az alapszínek szimbolikus nevei (*Red*, *Green*, *Blue*) azonosítják.

A felsorolási típus érvényességi tartományán belül nem megengedett egy másik felsorolási típus használata, amely az első típusban szereplő szimbólumok (konstansok) bármelyikére hivatkozik. Ez a korlátozás nem vonatkozik az előre definiált *boolean* felsorolási típusra; tehát megengedett pl. a (*false, true*) típusleírás.

**Példa**

```
type
  Day = (Mon,Tue,Wed,Thu,Fri,Sat,Sun);
  WeekEnd = (Sat,Sun);
```

• Ez a típusdefiníció hibás, mert a *Day* és a *WeekEnd* szimbólumlista metszete nem üres halmaz.

Mivel a felsorolási típusok megszámlálható halmazokhoz kapcsolódnak, az ilyen halmaz minden eleméhez egy nem negatív egész szám rendelhető hozzá. Ez a szám meghatározza a felsorolási típus azonosítólistáján az elem helyét. Ilyen értelemben a

```
type
  Direction = (North,South,East,West)
```

definícióban a *North* azonosítóhoz a 0-t, a *South*-hoz az 1-et, az *East*-hez a 2-t, ill. a *West*-hez a 3-at rendelhetjük hozzá. A hozzárendelt számok értékeit az előre definiált *ord* függvény adja meg. E függvény argumentuma a felsorolási típusú adatot képviselő kifejezés, értéke pedig az az *integer* típusú nem negatív szám, amely megadja az argumentum 0-tól számított pozícióját (sorszámát) az adatot tartalmazó, rendezett halmazon belül.

A felsorolási típusú adatokon értelmezett két további függvény a *pred* és a *succ*. A *pred* függvény argumentuma egy tetszőleges, rendezett típusú kifejezés, amely a rendezett halmaz egyik elemét képviseli. A függvény értéke a halmaz azon eleme, amely közvetlenül megelőzi az argumentum képviselte elemet.

A *pred* függvény argumentuma nem képviselheti a halmaz első elemét, mivel ezt az elemet nem előzi meg más elem.

A *pred* függvényhez hasonló a *succ* függvény. A függvény argumentuma egy tetszőleges, rendezett típusú kifejezés, amely a rendezett halmaz egyik elemét képviseli. A függvény értéke a halmaz azon eleme, amely közvetlenül követi az argumentum képviselte elemet. A *succ* függvény argumentuma nem képviselheti a halmaz utolsó elemét, mivel ezt az elemet nem követi más elem.

Az *ord*, a *pred* és a *succ* függvény argumentuma lehet *integer,* ill. *byte* típusú adatokat képviselő kifejezés. Ilyen esetben, ha a függvény használata szabályos, érvényesek a következő összefüggések:

```
ord(n) = n
pred(n) = n - 1
succ(n) = n + 1
```

**Példák**

Az előre definiált *boolean* típus és a nyíltan definiált *Color* típus

```
type
  Color = (Heart,Diamond,Spade,Club);
```

érvényességi tartományán belül érvényesek a következő állítások:

| *Kifejezés* | *Eredmény* |
|---|---|
| ord(false) | 0 |
| pred(true) | false |
| ord(Club) | 3 |
| succ(Diamond) | Spade |

## 7.2. Intervallumtípusok

Az intervallumtípus a rendezett típusok másik fajtája. Minden ilyen típus egy tetszőleges rendezett halmaz olyan részhalmaza, amelynek elemeit az *ord* függvény egy 1-es különbségű számtani sorozatra képezi le.

Az intervallumtípus leírása az intervallum alsó határát jelölő konstansból, a .. szimbólumból (pontpárból) és a felső határt jelölő konstansból áll. Az itt felsorolt konstansok által meghatározott intervallum nem lehet üres halmaz.

*Szintaxis*

```
intervallumtípus-leírás:
  alsó_határ..felső_határ
határ:
  konstans
  konstansnév
```

**Példák**

```
type
  Quadrant = 0..90;
  Upper = 'A'..'Z';
  Logical = false..true;
  Colour = (Red,Blue,Green,Yellow,Orange);
  Hue = Blue..Yellow;
```

- A *Quadrant* típusú adatok a 0 és 90 közötti (beleértve a két határértéket is) *integer* típusú (egész) számok.
- Az *Upper* típusú adatok az *'A'* és *'Z'* közötti *char* típusú (karakteres) konstansok.
- A *Logical* típusú adatok a *false* és a *true* konstans (mindkettő *boolean* típusú).
- A *Hue* típusú adatok a *Blue,* a *Green* és a *Yellow* felsorolási konstansok (mindegyik *Colour* típusú).

*Megjegyzés:* az előre definiált *byte* adattípus — bizonyos korlátozásokkal — az *integer* alaptípusból származtatott intervallumtípus, vagyis érvényes rá a következő „rejtett" definíció:

```
type
  byte = 0..255;
```

Az intervallum típusú adatokra is alkalmazható az *ord*, a *pred* és a *succ* függvény. E függvények értékképzése az alaptípuson belül megy végbe.

**Példák**

A *Hue* típusdefiníció

```
type
  Colour = (Red,Blue,Green,Yellow,Orange);
  Hue = Blue..Yellow;
```

érvényességi tartományán belül érvényesek a következő összefüggések:

| *Kifejezés* | *Eredmény* |
|---|---|
| ord(Blue) | 1 |
| succ(Yellow) | Orange |
| pred(Yellow) | Green |

Hangsúlyoznunk kell, hogy a felsorolási intervallumtípusok használata megnöveli a programok áttekinthetőségét és megkönnyíti „belövésüket", mert lehetővé teszi az adatok értéktartományának automatikus vizsgálatát. Ezekkel az adattípusokkal operatívtár-helyeket is megtakarítunk, ugyanis ha az alaptípus halmazelemeinek száma nem haladja meg a 255-öt, akkor a Turbo Pascal az adat tárolására csak egy bájtot használ fel. Hasonlóan takarékos az adatábrázolás, ha *integer* az alaptípus, és a két tartományhatár a [0,255] zárt intervallumba tartozik.

## 7.3. Típuskonverzió

A Turbo Pascal nyelvben bővítették az *ord* függvény koncepcióját (ez a függvény egy tetszőleges, rendezett típusú adatot *integer* típusú adattá konvertál). Bevezették ui. az egyparaméteres operátorok családját (nevük megegyezik a rendezett típusokéval), ezért adatkonverzióra nyílik lehetőség tetszőleges, rendezett típusok között.

A konverziós operátor *opr(arg)* alakú, ahol az *opr* egy rendezett típus azonosítója, az *arg* pedig bármilyen rendezett típusú kifejezés. A konverzió eredménye egy olyan *opr* típusú adat, amelyre igaz az *ord(a) = ord(opr(a))* reláció.

**Példák**

Az előre definiált, rendezett típusok és a *Color* nyílt típusdefiníció

```
type
  Color = (Heart,Diamond,Spade,Club);
```

érvényességi tartományán belül érvényesek a következő összefüggések:

| *Kifejezés* | *Eredmény* |
|---|---|
| integer(Club) | 3 |
| Color(2) | Spade |
| char(65) | 'A' |
| integer('A') | 65 |
| boolean(Heart) | false |
| byte(true) | 1 |

## 7.4. Értéktartomány-ellenőrzés

A Turbo Pascal nyelv fordítóprogramja lehetővé teszi, hogy a lefordított forrásprogram végrehajtása közben megtörténjék a skalár típusú adatok értéktartomány-ellenőrzése. Mivel ezek az ellenőrzések lelassítják a program futását, az alapértelmezés szerint ki vannak kapcsolva, de a {$R+} direktívával bekapcsolhatók. A forrásprogram azon részeiben, ahol ezek az ellenőrzések feleslegesek, a {$R−} direktívával kikapcsolhatók.

**Példa**

Az adatok értéktartomány-ellenőrzésének be- és kikapcsolása:

```
program Check;
type
  Figure = (Square,Circle,Diamond,Triangle);
var
  Shape : Figure;
begin
  Shape := Circle;
  Shape := Figure(4);
  {$R+}
  Shape := succ(Triangle);
  {$R−}
  Shape := pred(Square)
end.
```

- Az értéktartomány-ellenőrzés csak a fordítóprogramnak szóló direktívák között álló értékadó utasítás idejére marad bekapcsolva.
- Ha a direktívákat eltávolítanánk a programból, akkor az hibajelzés nélkül futna le.
- Ez a program egy „értékhatár-túllépés" hibajelzéssel áll le, mégpedig a *succ* függvény kiértékelésekor.

# 8. Karakterlánctípusok

A karakterlánctípusok mindegyike összetett típus. Egy karakterlánctípus definiálásakor meghatározzuk az e típushoz tartozó változó maximális karakterszámát. Minden karakterlánctípushoz karaktersorozatok halmaza kapcsolódik. A halmaz elemei az üres karakterlánc és mindazok a karaktersorozatok, amelyek hosszúsága nem lépi túl a (definícióban meghatározott) maximális karakterszámot.

A karakterlánctípus leírása a **string** kulcsszóból és az azt követő, szögletes zárójelekbe foglalt maximális karakterszámból áll. E szám kifejezhető egy *integer* típusú konstanssal vagy egy ilyen konstanssal egyenértékű konstansnévvel.

*Szintaxis*

karakterlánctípus-leírás:
  **string** [méret]
méret:
  konstans
  konstansnév

**Példák**

```
type
  Cities = string [20];
  Names = string [12];
```

A karakterlánc típusú adatok az adott típus maximális karakterszámánál eggyel több bájtot foglalnak le a tárban. Ez a plusz bájt tartalmazza a karakterlánc pillanatnyi hosszát. Ebből következik, hogy egy karakterlánc 255 karakternél hosszabb nem lehet.

## 8.1. Karakterláncok összefűzése

Két karakterlánc a konkatenációoperátorral egy lánccá fűzhető össze. Az operátor jele a + karakter, amely tetszőleges típusú karakterláncpárokat fűzhet össze. A művelet eredménye olyan karakterlánc, amely az első karakterlánc összes karakteréből és az utána következő második lánc összes karakteréből áll. Követelmény, hogy az összefűzés eredményeképpen keletkező karakterlánc ne legyen hosszabb 255 karakternél.

**Példák**

| *Kifejezés* | *Eredmény* |
|---|---|
| 'jan' + 'ewa' | 'janewa' |
| '5' + '.' + '4' | '5.4' |
| 'j' + '' + 'b' | 'jb' |

(Az utolsó példában egy üres karaktersorozatot képviselő karakterlánc is van.)

## 8.2. Relációk

Két karakterlánc a relációs operátorokkal hasonlítható össze. A relációs operátorok a következők: = (egyenlő), <> (nem egyenlő), > (nagyobb), < (kisebb), >= (nagyobb vagy egyenlő), <= (kisebb vagy egyenlő). A relációs operátorok elsőbbsége kisebb az összefűző operátor elsőbbségénél. Ez teszi lehetővé olyan kifejezések szerkesztését, mint pl. *'jan'= 'j'+ 'an'* — zárójelek használata nélkül.

Két karakterlánc akkor egyenlő, ha azonos a hosszúságuk és azonos karakterekből állnak. Ha nem egyenlőek, akkor a karakterlánc-összehasonlítás helyett az első egymásnak megfelelő, de eltérő karaktereket hasonlítjuk össze. Ha nincs ilyen karakterpár, mert az egyik karakterlánc a másiknak a kezdőrésze, akkor az a karakterlánc kisebb, amelyik rövidebb.

**Példák**

| *Kifejezés* | *Eredmény* |
|---|---|
| 'A' >'B' | false |
| 'jan' > 'Jan' | true |
| '' < char(0) | true |
| '' = '' | true |
| 'Jan' <= 'Jane' | true |
| '2599' < '270' | true |

## 8.3. Értékadás

Egy karakterlánc típusú változó az értékadó utasítással kaphat értéket. Az értékadó szimbólum jobb oldalán tetszőleges, karakterlánc típusú kifejezés áll, a szimbólum bal oldalán pedig egy karakterlánc típusú változó neve. Ha a kifejezés által képviselt karakterlánc hosszabb, mint a változóban ábrázolható lánc, akkor az értékadás balról jobbra megy végbe, a többletkarakterek elhagyásával.

**Példák**

```
type
  Name = string[7]
var
  FirstName, LastName : Name;
...
  FirstName := 'Jan';
  LastName := 'Bielecki';
```

- A *FirstName* változó új értéke *'Jan'*.
- A *LastName* változó új értéke *'Bielecki'*.

## 8.4. Karakterláncfüggvények és -eljárások

A karakterláncokon végezhető műveleteket bővítik a karakterláncfüggvények és -eljárások. Használatuk nagyban megkönnyíti a karakterlánc típusú adatok kezelését.

### *A Length függvény*

Hívása: *Length*(*Str*)

A *Str* egy karaktersorozatot képviselő karakterlánc típusú kifejezés. A *Length* függvény eredménye e karaktersorozat hosszát (karaktereinek számát) jelentő, *integer* típusú adat.

**Példák**

| *Kifejezés* | *Eredmény* |
|---|---|
| Length('jb') | 2 |
| Length('''') | 1 |
| Length('') | 0 |

### *A Str eljárás*

Hívása: *Str* (*Val, Var*)

A *Val* aritmetikai kifejezés, a *Var* pedig egy karakterlánc típusú változó neve. Az *Str* eljárás végrehajtásakor a *Var* változó új értéket kap, mégpedig a *Val* kifejezés által képviselt adat számértékét, karakterlánc formájában. Ha a *Val* kifejezés *integer* típusú, akkor a keletkező karaktersorozat olyan egész számot ábrázol, amely a legkevesebb számjegyet tartalmazza.

Ha közvetlenül a *Val* kifejezés után *:m* alakú módosító tag is van, ahol az *m integer* típusú kifejezés, akkor a *Val* képviselte adat átalakításából kapott karakterlánc *m* hosszúságú lesz. A keletkező számjegysorozat jobbra igazítottan jelenik meg a karakterláncban. Ha a számérték nem ábrázolható *m* számú karakterben, akkor az *m* értéke automatikusan annyival nő, amennyi a szám ábrázolásához kell.

**Példák**

(Feltételezzük, hogy a *StrVar* típusa **string**[4].)

| *Hívás* | *Karakterlánc* | *StrVar* |
|---|---|---|
| Str(45,StrVar) | 45 | '45' |
| Str(−250,StrVar) | −250 | '−250' |
| Str(45:3,StrVar) | b45 | '45' |
| Str(45:5,StrVar) | bbb45 | ' 4' |
| Str(−2:5,StrVar) | bbb−2 | ' −' |

Ha a *Val* kifejezés *real* típusú, akkor a mintasorozat a következő alakú:

*bsd.ddddddddddEsdd*

ahol *b* egy szóköz (space karakter), *s* a mantissza vagy az exponens (kitevő) előjele és *d* a számjegy. A mantissza előjelét egy szóköz vagy egy — (mínusz) karakter, az exponens előjelét pedig egy + vagy — (mínusz) karakter ábrázolja.

Ha közvetlenül a *Val* kifejezés után *:m* alakú módosító tag is van, ahol az *m integer* típusú kifejezés, akkor a *Val* képviselte adat átalakításából kapott karakterlánc *m* hosszúságú lesz. A keletkező számjegysorozat jobbra igazítottan jelenik meg a karakterláncban. Ha a számérték nem ábrázolható *m* számú karakterben, akkor a bemutatott mintasorozatból elmaradnak a vezető szóközök; ha ez nem elegendő, akkor csökken a mantissza számjegyeinek száma (legfeljebb két számjegyig). Ekkor az eredmény kerekítve jelenik meg.

**Példák**

(Feltételezzük, hogy az *Exp* értéke 45.678E1, a *StrVar* típusa pedig **string**[10].)

| *Hívás* | *Karakterlánc* | *StrVar* |
|---|---|---|
| Str(Exp:9,StrVar) | 4.568E+02 | '4.568E+02' |
| Str(Exp:8,StrVar) | 4.57E+02 | '4.57E+02' |
| Str(Exp:7,StrVar) | 4.6E+02 | '4.6E+02' |
| Str(Exp:6,StrVar) | 4.6E+02 | '4.6E+02' |
| Str(Exp:10,StrVar) | 4.5678E+02 | '4.5678E+02' |
| Str(Exp:11,StrVar) | 4.56780E+02 | '4.56780E+0' |

Ha közvetlenül a *Val* kifejezés után *:m:n* alakú módosító tag is van, ahol *m* és *n integer* típusú kifejezés, akkor a keletkező karakterlánc — jobbra igazítva — *m* hosszúságú lesz, és az *m* értékétől függetlenül egy *n* hosszúságú törtrésszel rendelkező, fixpontos, kerekített számkonstansként jelenik meg. Ha a számérték nem ábrázolható *m* számú karakterben, akkor az *m* értéke automatikusan annyival nő, amennyi a szám ábrázolásához kell.

**Példák**

(Feltételezzük, hogy az *Exp* értéke 45.678E1, a *StrVar* típusa pedig **string**[10]).

| *Hívás* | *Karakterlánc* | *StrVar* |
|---|---|---|
| Str(Exp:6:2,StrVar) | 456.78 | '456.78' |
| Str(—Exp:6:2,StrVar) | —456.78 | '—456.78' |
| Str(—Exp:8:2,StrVar) | b—456.78 | ' —456.78' |
| Str(—Exp:12:2,StrVar) | bbbbb—456.78 | ' —456.' |
| Str(—Exp:0:2,StrVar) | —456.78 | '—456.78' |

## A *Val* eljárás

Hívása: *Val* (*Str,Var,Rep*)

Az *Str* karakterlánc típusú kifejezés, a *Var integer* vagy *real* típusú változó, a *Rep* pedig *integer* típusú változó. A *Val* eljárás végrehajtásakor a *Var* változó új értéket kap, mégpedig a *Str* kifejezés által képviselt, karakterlánc típusú adat számértékét. Ez az adat nem kezdődhet és nem végződhet szóközökkel. Ha a *Var* változó *integer* típusú, akkor a konvertálandó számnak is *integer* típusúnak kell lennie.

Ha a konverzió elvégezhető, akkor a *Rep* változó 0 értéket kap, különben az új értéke olyan pozitív szám, amely sorszáma annak a karakternek (a konvertálandó karakterláncban), amelyik leállította a konverziót. Ekkor a *Var* változó nem kap új értéket.

**Példák**

(Feltételezzük, hogy az *IntVar* és az *IntRep integer* típusú változó.)

| *Hívás* | *IntVar* | *IntRep* |
|---|---|---|
| Val('23,IntVar,IntRep) | 23 | 0 |
| Val('23.0',IntVar,IntRep) | változatlan | 3 |
| Val(—200 ',IntVar,IntRep) | változatlan | 5 |

## *A Copy függvény*

Hívása: *Copy* (*Str,Pos,Num*)

A *Str* karakterlánc típusú kifejezés, a *Pos* és a *Num* pedig *integer* típusú változó. A *Copy* függvény eredménye olyan karakterlánc típusú adat, amely az *Str* karakterlánc *Pos* pozíciójától kezdve *Num* számú karakterből áll. Ha *Pos* > *Length* (*Str*), akkor a függvény eredménye egy üres karakterlánc. Ha (*Pos*+*Num*) > *Length* (*Str*), akkor az eredmény úgy alakul, mintha a *Num* értéke csak *Length*(*Str*)—*Pos*+1 lenne. *Pos* értéke nem lehet 1-nél kisebb és 255-nél nagyobb.

**Példák**

| *Hívás* | *Eredmény* |
|---|---|
| Copy('Pascal',3,4) | 'scal' |
| Copy('Pascal',5,3) | 'al' |
| Copy('Pascal',7,2) | '' |
| Copy('Pascal'2,1) | 'a' |

### *A Concat függvény*

Hívása: *Concat* ($St_2, \ldots, St_k$)

A $St, St_2, \ldots, St_k$ karakterlánc típusú kifejezés. A *Concat* függvény eredménye a $St_1+St_2+\ldots+St_k$ karakterlánc típusú adat. A függvény argumentumainak teljes hossza nem lépheti túl a 255-öt.

**Példák**

| *Hívás* | *Eredmény* |
|---|---|
| Concat('j','a','n') | 'jan' |
| Concat('j','','b') | 'jb' |
| Concat('jan',' ','b') | 'jan b' |

### *A Pos függvény*

Hívása: *Pos* (*Src, Trg*)

Az *Src* és a *Trg* karakterlánc típusú kifejezés. A *Pos* függvény eredménye egy *integer* típusú adat, amely a *Trg* karakterláncban azt a karakterpozíciót határozza meg, amelytől kezdve az *Src* tartalmával azonos karakterek következnek. Ha az *Src* karakterlánc nem fordul elő a *Trg*-ben, akkor a függvény eredménye 0.

**Példák**

| *Hívás* | *Eredmény* |
|---|---|
| Pos('23','123123') | 2 |
| Pos('21','123123') | 0 |
| Pos('c','abcabc') | 3 |
| Pos('','Pascal') | 0 |

### *Az Insert eljárás*

Hívása: *Insert* (*Src,Trg,Pos*)

Az *Src* karakterlánc típusú kifejezés, a *Trg* karakterlánc típusú változó, a *Pos* pedig egy *integer* típusú kifejezés. Az *Insert* eljárás beszúrja a *Trg* képviselte karaktersorozatba — a *Pos* pozíciójú karakter elé — az *Src* karakterlánc tartalmát. Ha *Pos* > Length(*Trg*), akkor feltételezzük, hogy *Pos*:= *Length* (*Trg*)+1. *Pos* értéke nem lehet 1-nél kisebb és 255-nél nagyobb.

Ha a karakterek beszúrása a *Trg* változó maximális méretének túllépését okozná, akkor a jobb oldali többletkarakterek elvesznek.

**Példák**

(Feltételezzük, hogy a *Trg* változó típusa **string**[5], értéke pedig — az *Insert* eljárás mindenkori hívása előtt — *'jb'*.)

| *Hívás* | *Trg* |
|---|---|
| Insert('an',Trg,2) | 'janb' |
| Insert('jb',Trg,4) | 'jbjb' |
| Insert('Janek',Trg,1) | 'Janek' |

### *A Delete eljárás*

Hívása: *Delete* (*Str,Pos,Num*)

A *Str* karakterlánc típusú változó, a *Pos* és a *Num* pedig *integer* típusú változó. A *Delete* eljárás a *Pos* pozíciójú karaktertől kezdve *Num* számú karaktert töröl a *Str* képviselte karaktersorozatból. Ha *Pos* > Length(*Str*), akkor nem történik semmi. Ha *Pos*+*Num*—1 > *Length* (*Str*), akkor az eredmény úgy alakul, mintha a *Num* értéke *Length* (*Str*)—*Pos*+1 lenne. *Pos* értéke nem lehet 1-nél kisebb és 255-nél nagyobb.

**Példák**

(Feltételezzük, hogy a *Trg* változó típusa **string**[5], értéke pedig — az *Insert* eljárás mindenkori hívása előtt — *'janek'*.)

| *Hívás* | *Trg* |
|---|---|
| Delete(Trg,4,2) | 'jan' |
| Delete(Trg,6,1) | 'janek' |
| Delete(Trg,5,3) | 'jane' |

## 8.5. Karakterlánc- és karaktertípusok

Valamennyi karakterlánctípus megfelel a rendezett *char* típusnak. Ez azt jelenti, hogy a program mindazon helyein, ahol megengedett a karakterlánc típusú kifejezés használata, használható a *char* típusú adat, és fordítva. Csak az a követelmény, hogy amikor egy *char* típusú változónak értéket adunk, az értékadó szimbólum jobb oldalán álló karakterlánc típusú adat 1 karakter hosszúságú legyen.

A karakterlánc típusú változó egyes karaktereihez nemcsak az előre definiált *Copy* függvénnyel férhetünk hozzá, hanem — sokkal egyszerűbben — indexeléssel is. Az indexelés definíciójának elfogadhatjuk a következő azonosságot:

*St[i]* = *Copy* (*St,i,*1)

Ez az azonosság csak akkor igaz, ha teljesül az 1 < = i < *Length* (*St*) feltétel. Ezen az intervallumon kívül megengedett az operatív tár szomszédos bájtjainak elérése, különösen az *ord* (*St* [0]) kifejezés esetén, amely megadja az *St* által képviselt karakterlánc hosszúságát.

*Megjegyzés:* a {$R+} direktíva használata csak kismértékben teszi lehetővé a helytelen indexeléssel kapcsolatos hibák kiderítését. Ez abból ered, hogy a direktíva bekapcsolja ugyan a — deklarációból ismert — maximális index túllépésének vizsgálatát, de nem vezeti be a pillanatnyi mérettúllépés ellenőrzését.

**Példa**

```
var
    Name : string[5];
...
Name := 'Jan';
{$R+}
Name[0] :=chr[5];
Name[4] :='e';
Name[5] :='k';
```

- Az értékadások végrehajtása után a *Name* változó értéke *'Janek'.*
- A {$R+} direktíva használata felesleges, mert nem befolyásolja az értékadások végrehajtását.
- Ha a direktíva hatáskörén belül olyan értékadás fordulna elő, mint pl. a *Name*[6] := *'B'*, akkor a program „a maximális index túllépése" hibajelzéssel állna le.

# 9. Tömbtípusok

A tömbtípusok mindegyike összetett típus. A tömb meghatározott számú, azonos típusú komponensből (elemből) áll. A komponensek akár elemi, akár összetett típusúak lehetnek. A tömbelemek indexeléssel érhetők el. Az index szögletes zárójelekkel körülvett, tetszőleges skalárkifejezés, melynek megengedett értéktartományát a tömbtípus leírása határozza meg.

A tömbtípus leírása az **array** kulcsszóból, az azt követő, szögeletes zárójelekbe foglalt indexleírásból, az **of** kulcsszóból és a komponensek típusleírásából áll.

*Szintaxis*

tömbtípus_ leírás:
  **array** [indextípus] **of** elemtípus
indextípus:
  rendezett-típus-leírás
elemtípus:
  típusleírás

**Példák**

```
type
  Color = (Red,Green,Blue);
  TruthTable = array[boolean] of boolean;
  Height = 0..200;
var
  Square : array [Color] of Height;
  Matrix : array [2..8] of array [2..8] of integer;
  Table : TruthTable;
```

- A *TruthTable* típus a *boolean* típusú indexekkel és elemekkel rendelkező tömbök halmazához kapcsolódik.
- A *Square* változó olyan tömb, amelynek indexei *Color,* elemei pedig *Height* típusúak.
- A *Matrix* változó olyan tömb, amelynek indexei 2..8 típusúak és elemei is tömbök, 2..8 típusú indexekkel, ill. *integer* típusú elemekkel.
- A *Table változó TruthTable* típusú tömb, *boolean* típusú indexekkel.

A tömbelemeket a programokban tömbelemnevek képviselik. A tömbelem neve a tömbnévből és az azt követő, szögletes zárójelekkel körülvett indexkifejezésből áll. Az indexkifejezés tetszőleges rendezett típusú kifejezés, típusa a tömbtípus leírásában szereplő „indextípus"-sal egyezik meg.

A {$R+} direktíva használata olyan kódgenerálást eredményez, mely lehetővé teszi az indexkifejezések helyességének vizsgálatát. A vizsgálat során a rendszer ellenőrzi, hogy az indexkifejezés értéke a tömbtípus leírásában szereplő indextípus értéktartományán belül van-e.

**Példa**

```
type
  Color = (Red,Green,Blue);
var
  Matrix : array [Color] of boolean;
```

- A *Matrix* változó olyan tömb, amelynek indexei *Color,* elemei pedig *boolean* típusúak.
- A *Matrix* tömb helyes nevű elemei a következők:

*Matrix* [*Green*]
*Matrix* [*pred* (*Blue*)]
*Matrix* [*Color* (*ord* (*true*) ) ]

- A {$R+} direktíva hatáskörén belül pl. a *Matrix* [*succ* (*Blue*)] tömbelemnév hibásnak minősül.

## 9.1. Többdimenziós tömbök

Mivel egy tömbelem tömb is lehet, könnyű létrehozni két- és többdimenziós tömböket. A szintaxisnak megfelelően, egy kétdimenziós tömböt leíró típus definíciója a következő alakú:

```
type
  TwoDim = array [2..5] of array [boolean] of real:
```

Ezt a definíciót egyszerűsítve is felírhatjuk:

```
type
  TwoDim = array[2..5,boolean] of real;
```

vagyis az indextípusok leírása szögletes zárójellel körülvett listát alkothat. Hasonló egyszerűsítéssel élhetünk a tömbelemnevek esetén; pl. a

```
var
  Matrix : TwoDim;
```

deklaráció érvényességi körén belül helyesek és egyenértékűek a következő nevek (amelyek a *Matrix* tömb elemei): *Matrix*[3][*true*] és *Matrix*[3,*true*].

**Példa**

```
type
  Pupils = string[20];
  Class = array[1..30] of Pupils;
var
  School : array [1..20] of Class;
```

- A *Class* típus olyan halmazhoz kapcsolódik, amelynek elemei egy osztály tanulóinak neveiből álló tömbök.
- A *School* változó olyan kétdimenziós tömb, amelynek típusa

**array** [1..20] **of array**[1...30] **of string** [20]

és tartalmazza az iskola 20 osztályában a tanulók névsorát.

- A *School* [3][25] név a 3. osztály névsorában a 25. pozícióban szereplő tanulóra vonatkozik. E név egyenértékű az egyszerűsített *School* [3,25] névvel.

## 9.2. Karaktertömbök

A karaktertömb olyan egydimenziós tömb, amelynek indexe *integer* típusú, elemei pedig *standard char* típusúak. A karaktertömbök olyan karakterlánc típusú változónak tekinthetők, amelyek állandó hosszúságú karaktersorozatokat képviselnek. Ennek az értelmezésnek köszönhetően a karaktertömbök és az elemek neve a karakterlánc típusú kifejezésekben mindazon helyeken előfordulhat, ahol egyébként karakterlánc típusú változók állnak. Ez különösen a relációkra érvényes.

**Példa**

```
type
  Number = 2..5;
  StrTyp = string[5];
  ArrTyp = array[Number] of char;
var
  StrVar: StrTyp;
  ArrVar: ArrTyp;
  i: byte;
begin
  for i:=2 to 5 do
    ArrVar[i]:=chr(63+i);
  StrVar:='98765';
  StrVar:=Copy(StrVar,2,2) + ArrVar + '/' + ArrVar[4];
end;
```

- A második értékadás után a *StrVar* tartalma '87ABC'.
- Ugyanakkor az *ArrVar* > *StrVar* reláció értéke *true.*

## 9.3. Értékadások

Összhangban azzal a szabállyal, miszerint tetszőleges típusú változó egyetlen utasítással megkaphatja az azonos típusú adat értékét, egyetlen utasítással egy teljes tömb értékét is átadhatjuk a tömbváltozónak. Megengedett egy karakterkonstans értékének átadása is, de csak akkor, ha a konstans karaktereinek száma egyenlő a tömb méretével. A tömböknek más karakter típusú kifejezés értéke nem adható át. A karakterlánc típusú változók viszont értéket kaphatnak karakterlánc típusú kifejezésektől és karaktertömböktől egyaránt.

**Példa**

```
type
  Line = string[60];
  Page = array[1..30] of Line;
  Chapter = array [1..50] of Page;
  Word = array[1..5] of char;
var
  Book1,Book2 : array[1..6] of Chapter;
  Arr:Word;
  Brr:array[1..5] of char;
  Str:string[5];
```

- A példában bemutatott definíciók és deklarációk érvényességi körén belül helyesek a következő értékadások:

```
Book1:=Book2
Arr:='Janek'
Str:=Arr
```

- Hibásak viszont a következők:

| | |
|---|---|
| Arr:='Jan' | mert a karakterlánc túl rövid; |
| Arr:=Str | mert a *Str* egy karakterlánc típusú kifejezés; |
| Arr:=Brr | mert az *Arr Word* típusú, a *Brr* pedig nem az. |

## 9.4. Előre definiált tömbök

A Turbo Pascal nyelvben a 8 bites mikroszámítógépek számára előre definiáltak két tömböt, amelyek *byte* típusú elemekből állnak. E tömbök segítségével hozzáférhetünk az operatív tárhoz és a bemeneti, ill. kimeneti portokhoz.

### *A Mem tömb*

A *Mem* tömbbel hozzáférhetünk az operatív tár bájtjaihoz. A tömbhivatkozásban szereplő index *integer* típusú kifejezés, amely meghatározza a bájt tárcímét.

**Példa**

```
ByteVar:=Mem[200]
Mem[300]:=$FF
```

- Az első utasítás végrehajtása után a *ByteVar* nevű változó új értéke a 200-as tárcímen levő bájt értéke lesz.
- A második értékadás végrehajtása után a 300-as tárcímre a $FF értékű bájt kerül.

*A Port tömb*

A *Port* tömbbel hozzáférhetünk a bemeneti, ill. kimeneti portokhoz. A tömbhivatkozásban szereplő index *integer* típusú kifejezés, amely meghatározza a port címét. A *Port* tömb elemei nem lehetnek alprogram-paraméterek; de előfordulhatnak kifejezésekben és az értékadó szimbólum bal oldalán. A kifejezésben a megadott portról származó adatokat jelentik, az értékadó szimbólum bal oldalán pedig azt eredményezik, hogy az adat a megadott portra íródik.

**Példa**

```
Port[34]:=$5F
PortStatus:=Port[35]
```

- Az első utasítás a 34-es című portra a $5F konstans által képviselt adatot írja.
- A második utasítás adatot olvas be a 35-ös című portról, és elhelyezi azt a *PortStatus* nevű változóban.

A Turbo Pascal nyelvben az IBM PC számára hasonló előre definiált tömbök vannak, azzal a különbséggel, hogy a *Mem* tömb indexe:

*seg:=offs*

alakú. A *seg* a szegmens sorszáma, az *offs* pedig a szegmensen belüli eltolást jelenti. Ezenkívül előre definiálták még a *MemW* és a *PortW* tömböt, amelyek memóriaszavak és szavas portok elérését teszik lehetővé.

**Példa**

```
Mem[$0020:$0030]:='E'
```

- Az utasítás az *'E'* konstans által képviselt adatot beírja az operatív tár $20-as számú szegmensen belüli, $30 relatív című bájtjába.

# 10. Rekordtípusok

A rekordtípusok mindegyike összetett típus. A rekord meghatározott számú, nem feltétlenül azonos típusú komponensből áll. A komponensek akár elemi, akár összetett típusúak lehetnek. A rekordelemek, melyeket mezőknek is nevezünk, szelekcióval (kiválasztással) érhetők el. Kiválasztáskor megadjuk a rekord nevét, a . (pont) karaktert és a rekordmező azonosítóját.

A rekordtípus leírása a **record** kulcsszóból, a rekordmezők listájából és az **end** kulcsszóból áll. A lista minden eleme egy-egy meződeklaráció. A meződeklarációkat pontosvessző választja el egymástól; a lista utolsó eleme lehet a változó rekordrész. A változó rekordrész a **case** kulcsszóból, az azt követő vezérlőmező-deklarációból (vagy vezérlőtípus-leírásból), az **of** kulcsszóból és a változatlistából áll. A változatlista minden eleme a kiválasztási címkék listájából (melyet kettőspont követ), valamint a kerek zárójelekkel körülvett változatmezők listájából áll. A változatlista egyes elemeit pontosvessző választja el egymástól.

*Szintaxis*

rekodtípus_ leírás:
  **record** rekordmező-lista **end**
rekordmező-lista:
  meződeklaráció
  változó_ rekordrész
meződeklaráció:
  mezőnévlista : típusleírás
mezőnév:
  azonosító
változó_ rekordrész:
  **case** vezérlőmező-deklaráció **of** változatlista
  **case** vezérlőtípus-leírás **of** változatlista
vezérlőmező-deklaráció:
  vezérlőmező-név : vezérlőtípus-leírás
vezérlőmező-név:
  azonosító
vezérlőtípus-leírás:
  rendezett-típus-azonosító
változatlista:
  kiválasztásicímke-lista : (változatmezők_ listája)
kiválasztási_ címke:
  konstans
  konstansnév

**Példák**

```
type
  Days = 1..31;
  Date = record
           Day : Days;
           Month : 1..12;
           Year : 1900..1999
         end;
Person = record
           Name : string[20];
           BirthPlace : string[30];
           case Sex : (male,female,alien) of
           male,female : (BirthDate : Date);
           alien : (BirthDate : real;
             BodyData : record
                          H : integer;
                          W : real
                        end)
         end;
var
  MyBirthDay : Date;
  EarthMan,Galaxian : Person;
```

- A *Date* típus olyan rekordok halmazához kapcsolódik, amelyek mezői *Days*, 1..12 és1900..1999 típusúak.
- A *Person* típus változó résszel rendelkező rekordok halmazához kapcsolódik. A vezérlőmező neve *Sex,* típusa (*male, female, alien*). Ha a *Sex* vezérlőmező értéke *male* vagy *female,* akkor a *Person* típusú rekord mezőinek típusa rendre: **string**[20], **string**[30], (*male,female,alien*), 1..31, 1..12 és 1900..1999. Ha a *Sex* vezérlőmező értéke *alien,* akkor a *Person* típusú rekord mezőinek típusa rendre: **string**[20],**string**[30], (*male,female,alien*), *real, integer* és *real.*
- A *MyBirthDay* változó olyan rekord, amelynek *Day, Month* és *Year* mezői vannak.
- Az *EarthMan* és a *Galaxian* változó *Person* típusú rekord. A változó rekordrész kiválasztásakor a vezérlőmezőnek a (*male, female, alien*) típusú adattal értéket adunk.
- Az *EarthMan* változó vezérlőmezője *EarthMan.Sex,* a *Galaxian* változóé pedig *Galaxian.Sex* alakú.
- A *Galaxian.Sex*:=*alien* értékadás végrehajtása után hozzáférhetünk az *alien* változat komponenseihez. A legjobb a

*Galaxian.BodyData.H*:=20

értékadás, amelyben a kiválasztás operátorát (a pontot) először a *Galaxian* rekord komponenseire alkalmazzuk, később pedig a *Galaxian.BodyData* rekord komponenseire is.

## 10.1. A with utasítás

A rekordkomponensekre hivatkozás nemcsak a mezőnév megadását igényli, hanem a rekordnévét is. Mivel ez bizonyos esetekben számottevően meghosszabbítja a program szövegét, kényelmes a *with* összekötő utasítás, amely mintegy „kitakarja" a rekordtípus-definíció belsejét.

Általános esetben a *with* utasítás a **with** kulcsszóból, az azt követő rekordnévlistából, a **do** kulcsszóból és egy tetszőleges utasításból áll. Az utasításon belül a listában szereplő rekord mezőinek kiválasztása a mezőnévvel helyettesíthető. Definíciószerűen elfogadjuk, hogy a

```
with a,b,...,z do i
```

alakú utasítás egyenértékű a

```
with a do
  with b do
    ...
      with z do i
```

utasítással.

*Szintaxis*

with_ utasítás:
**with** rekordnévlista **do** utasítás
rekordnév:
azonosító
tömbelemnév
mezőnév

**Példa**

```
var
  Alfa : record
           Beta : record
                    Gamma : char;
                    Delta : real
                  end;
           Gamma : integer
         end;
  Beta : array[boolean]of record
                            Chi,Tau : real
                          end;
```

- A példában bemutatott deklaráció érvényességi körén belül helyes a

```
with Alfa.Beta,Beta[true] do
  Delta:=Tau
```

utasítás, amely ugyanúgy hajtódik végre, mint az

```
Alfa.Beta.Delta:=Beta[true].Tau
```

értékadás.

- Helyes a

```
with Alfa.Beta,Alfa do begin
  Delta:=23.4;
  Writeln(Gamma)
end
```

utasítás is, amely ugyanúgy hajtódik végre, mint a

```
begin
  Alfa.Beta.Delta:=23.4;
  Writeln(Alfa.Gamma)
end
```

utasítás.

- Vegyük figyelembe, hogy a **with** utasítás rekordnévlistáján lényeges a sorrend, mert az olyan utasítás, mint pl. a

```
with Alfa,Alfa.Beta do begin
  Delta:=23.4;
  Writeln(Gamma)
end
```

ugyanúgy hajtódik végre, mint a

```
begin
  Alfa.Beta.Delta:=23.4;
  Writeln(Alfa.Beta.Gamma)
end
```

és nem úgy, mint a

```
begin
  Alfa.Beta.Delta:=23.4;
  Writeln(Alfa.Gamma)
end
```

## 10.2. Változatok (unionok)

Ha a rekordban változatok vannak, akkor a Turbo Pascalban elfogadott belső ábrázolás szerint minden változat első mezője ugyanazon a memóriacímen kezdődik. Mivel a vezérlőmező deklarációja típusleírássá egyszerűsíthető, a rekordleírást könnyű átalakítani olyan szerkezet leírásává, amelyet a C nyelvben unionnak hívnak. Tudnunk kell azonban, hogy a rekordokon végzett műveletektől eltérően az unionokon végrehajtott műveletek megkövetelik a komponensek ábrázolásának — vagy legalább tárolásának és méretének — ismeretét.

**Példa**

```
var
  Word : record
           case boolean of
             false : (LowByte,HighByte : byte);
             true : (FullWord : integer)
         end
```

• A példában bemutatott deklaráció érvényességi körén belül megengedett, hogy a

```
with Word do
  FullWord:=FullWord and $FF00
```

utasítás helyett a

```
with Word do
  LowByte:= 0
```

utasítást használjuk.

• A példa helyessége azon a tényen alapszik, hogy az *integer* típusú adat alacsonyabb helyi értékű bájtja kisebb tárcímen található, mint a magasabb helyi értékű bájt, ill. hogy a szerkezet mezői ugyanolyan sorrendben helyezkednek el a tárban, mint amilyen sorrendben deklaráltuk őket (vagyis a *LowByte* megelőzi *HighByte*-ot).

## 10.3. Rekordértékadás

A tömbváltozókhoz hasonlóan megengedett, hogy egy rekord típusú változó az egész rekord értékét megkapja egy utasítással. Egyetlen követelmény, hogy a rekordváltozó típusa azonos legyen az értékadó rekord típusával.

**Példa**

```
type
  Name = record
           LastName : string[32];
           Names : array[(fst,snd,trd)] of string[8]
         end;
var
  JohnName,EwaName : Name;
  ...
EwaName:=JohnName;
```

# 11. Halmaztípusok

A halmaztípusok mindegyike összetett típus. A halmazváltozó olyan változó, amely egy hatványhalmaz részhalmazától kaphat értéket. A hatványhalmaz alaphalmaza rendezett típusú elemek tetszőleges halmaza. A halmazváltozó elemeinek mindegyike ilyen elem.

Két halmaz típusú adat akkor és csak akkor egyenlő, ha azonos elemekből áll. Ha az egyik halmazváltozó minden eleme egyben eleme a másik halmazváltozónak, akkor azt mondjuk, hogy az első változó részváltozója a másiknak.

A halmaztípus leírása a **set** kulcsszóból, az azt követő **of** kulcsszóból és az alaptípus leírásából áll. Az alaptípusként felfogható rendezett típus nem tartalmazhat 256 elemnél többet, és az elemek mindegyikére alkalmazott *ord* függvény értékének a [0,255] zárt intervallumhoz kell tartoznia.

*Szintaxis*

halmaztípus-leírás:
  **set of** alaptípus
alaptípus:
  rendezett_ típus

**Példák**

```
type
  DaysOfWeek = set of (Mon,Tue,Wed,Thu,Fri,Sat,Sun);
  Chars = set of char;
  SmallLetters = set of 'a'..'z';
  DayNumbers = set of 1..31;
```

- A *DaysOfWeek* típusú adat értéke a hét napjainak neveiből alkotott halmaz tetszőleges részhalmaza (beleértve az üres halmazt is).

- A *Chars* típusú adat értéke az ASCII kódkészlet karaktereiből alkotott halmaz tetszőleges részhalmaza.

- A *SmallLetters* típusú adat értéke az ASCII kódkészlethez tartozó, kisbetűkből alkotott halmaz tetszőleges részhalmaza.

- A *DayNumbers* típusú adat értéke az [1,31] zárt intervallumhoz tartozó egész számokból alkotott halmaz tetszőleges részhalmaza.

## 11.1. Halmazkonstansok

Egy halmazkonstans szögletes zárójelekkel körülvett, azonos (rendezett) típusú konstansok listájából áll. A lista lehet üres, tartalmazhat ismétlődő elemeket, ill. *min*..*max* alakú szerkezeteket, ahol a *min* és a *max* azonos típusú konstans. Ekkor a min..*max* egyenértékű a *min*-től a *max*-ig terjedő összes konstanssal, a határértékeket is beleértve. Ha *min*> *max*, akkor üres részhalmazról van szó.

*Megjegyzés:* a halmazkonstans értéke nem függ az őt alkotó alap típusú konstansok sorrendjétől; többszörös fellépésük nem befolyásolja a halmazkonstans értékét.

**Példa**

Néhány — helyes és hibás — halmazkonstans:

| | | |
|---|---|---|
| Helyes halmazkonstansok: | 'P','a','s','c','a','1'] | |
| | [4,9,5,0,8,6] | |
| | [true,false] | |
| | [] | |
| | [20..30,50,70..80] | |
| Hibás halmazkonstans: | [true,2] | eltérőek az alaptípusok; |
| | [200..300] | ord(300)> 255 |
| | [4.0,5.0] | az alaptípus nem rendezett típus. |

## 11.2. Kifejezések

A halmazkifejezések a következő alkotóelemekből tevődnek össze: halmazváltozókra, -konstansokra, -szerkezetekre való hivatkozások, valamint az unió, a különbség és a metszet operátorai.

A halmazszerkezetek a halmazkonstansokhoz hasonló alakúak, azzal a különbséggel, hogy a listájukon szereplő elemek nemcsak az alaptípus konstansai, hanem ilyen típusú tetszőleges kifejezések is lehetnek.

A halmazunió-operátor jelölésére a + karaktert, a különbségoperátor jelölésére a — (mínusz) karaktert, a metszetoperátor jelölésére pedig a * karaktert használjuk.

A halmazok uniója olyan halmaz, amely mindkét összetevő összes elemét tartalmazza. A halmazok különbsége olyan halmaz, amely a kisebbítendő halmaz azon elemeiből áll, melyek nem elemei a kivonandónak. A halmazok metszete olyan halmaz, amely mindkét tényező közös elemeiből áll.

A halmazműveleteken kívül a következő relációkban fordulhatnak elő halmaz típusú adatok:

| | |
|---|---|
| *a* = *b* | igaz, ha *a* és *b* azonos elemek halmazai; |
| *a* <> *b* | igaz, ha *a* és *b* különböző elemek halmazai; |
| *a* <= *b* | igaz, ha *a* üres halmaz, vagy az *a* minden eleme egyben *b* eleme is; |
| *a* >= *b* | igaz, ha *b* üres halmaz, vagy a *b* minden eleme egyben *a* eleme is; |
| *e* **in** *a* | igaz, ha az *e* alap típusú elem az *a* halmaz eleme. |

**Példák**

A *SetVar* változódeklaráció érvényességi körén belül (feltételezve, hogy a változó jelenlegi értéke a [*Red,Green*] konstans által képviselt halmaz típusú adat)

```
type
  Colors = (Red,Green,Blue);
  Paint = set of Colors;
var
  SetVar : Paint;
...
SetVar:=[Red,Green];
```

igazak a következők:

| *Kifejezés* | *Eredmény* |
|---|---|
| SetVar = [Red..Blue] — [Blue] | true |
| SetVar <>[] | true |
| SetVar <=[Red..succ(Green)] | true |
| SetVar >= [Blue..Green] | true |
| SetVar*[Red] = [Red] | true |
| Green **in** SetVar | true |

## 11.3. Halmazértékadás

A tömbváltozókhoz és a rekordokhoz hasonlóan megengedett, hogy egy halmaz típusú változó egy utasítással az egész halmaz értékét megkapja, de a halmazváltozó típusának azonosnak kell lennie az értékadó halmaz típusával.

**Példa**

```
var
  SetVar : array[2..5, boolean] of set of 20..30;
  SetVar[3,false] := [5*5...pred(29)];
```

• Az értékadó utasítás jobb oldalán egy halmazszerkezet áll, amely egyenértékű a [25,26,27,28] halmazkonstanssal.

# 12. Fájltípusok

A fájltípusok mindegyike összetett típus. A fájl komponensekből áll, melyek azonos típusúak. A fájl komponenseit elemeknek nevezzük. A tömbtípusoktól eltérően egy fájl elemeinek számát nem határozzuk meg a fájldeklarációban, hanem a program futásának eredményétől tesszük függővé (elsősorban attól, hogy a fájlt milyen adatállományhoz rendeljük hozzá).

Hangsúlyozni kell, hogy a fájlnak nevezett objektum csupán elvont logikai modellje annak a fizikai adatállománynak, amely leggyakrabban a programon kívül létezik. A fizikai adatállományok — itt röviden állományok — helyet foglalhatnak a számítógép háttértárában vagy operatív memóriájában, de azonosíthatók a külső berendezések segítségével be- és kivihető adatfolyamokkal is.

A programozás szempontjából nagy egyszerűsítést jelent, hogy különböző állományok ugyanazzal a fájllal dolgozhatók fel, ha hozzárendeltük őket egymáshoz. Így a programozónak nem kell ismernie a konkrét operációs rendszer tulajdonságait, sem a háttértárak adatszervezési és -tárolási megoldásait. Ezért a fájl szintű programozás többnyire a fájl megnyitását, a fájl elemein végrehajtandó műveleteket és a fájl lezárását jelenti.

A fájltípus leírása a **file** kulcsszóból, az azt követő **of** kulcsszóból és az elemtípus leírásából áll. A fájlelemek tetszőleges elemi típusúak, ill. összetett típusúak lehetnek (kivéve a fájltípust).

*Szintaxis*

fájltípus-leírás:
  **file of** elemtípus:
elemtípus
  típusleírás

**Példák**

```
type
  Measurements = file of real;
  Persons = file of
              record
                Name : string[20];
                Sex : (male,female);
                Age : 18..65
              end;
```

```
var
  Experiment : Measurements;
  Population : Persons;
  Complexes : file of record
                Re,Im : real
              end;
  Arrays : file of array[2..6,4..8,boolean] of byte;
```

- A *Measurements* típus a *real* típusú adatok sorozatához kapcsolódik.
- A *Persons* típus olyan rekordok sorozatát írja le, amelyeknek összetevői rendre: **string**[20], (*male,female*) és 18..65.
- Az *Experiment* változó egy *Measurements* típusú fájlváltozó.
- A *Complexes* változó olyan fájlváltozó, amely *real* típusú adatpárok sorozatához kapcsolódik.
- Az *Arrays* változó olyan tömbök sorozatát írja le, amelyek típusa **array** [2..6,4..8,*boolean*] **of** *byte*.

## 12.1. Alprogramok

A fájlokon végezhető műveletek kizárólag függvényekkel és eljárásokkal valósíthatók meg. E műveletek végrehajtását meg kell előznie a fájl és az adatállomány egymáshoz rendelésének, ha a művelet a fájl elemeire vonatkozik, akkor előzőleg a fájlt meg kell nyitni. A fájlt és az adatállományt az *Assign* eljárás rendeli egymáshoz; a fájlt a *Reset* és a *Rewrite* eljárás nyitja meg. A *Reset* eljárás használata nem jelenti azt, hogy a fájlt csak olvasásra nyitjuk meg; a *Rewrite* eljárás sem csak írásra nyitja meg a fájlt. A program befejezése előtt minden fájlt le kell zárni: erre való a *Close* eljárás. A programfutás befejezése nem jelenti a nyitott fájlok lezárását!

Közvetlenül a megnyitás után a fájl kezdeti pozícióban (az elején) áll. A fájlpozíciót a *Seek* eljárással változtathatjuk. A fájl aktuális mérete a *FileSize* függvénnyel, pozíciója pedig a *FilePos* függvénnyel határozható meg. Ha a fájl közbenső pozícióban van (azaz a kezdeti és a végpozíció között), akkor minden *Write* eljáráshívás a legközelebbi fájlelemet írja felül. Ez a felülírás nincs hatással a fájl többi elemére, ezért az adott típusú elemekből álló fájlok szekvenciális szervezésű adatállományokat képviselhetnek.

### *Az Assign eljárás*

Hívása: *Assign* (*FileVar,StrExp*)

A *FileVar* egy fájlváltozó neve, a *StrExp* pedig egy karakterlánc típusú kifejezés. A *FileVar* névvel azonosított fájl nem lehet nyitva. Az *Assign* eljárás végrehajtása hozzárendeli a *FileVar* névvel azonosított fájlt a *StrExp* kifejezéssel meghatározott adatállományhoz.

**Példa**

```
var
  Results : file of real;
...
Assign(Results,'A:TEST,DAT');
```

- A *Results* fájlváltozó egy *real* típusú adatokból álló fájlt képvisel.
- Az *Assign* eljárás végrehajtása hozzárendeli a *Results* változóval azonosított fájlt a TEST.DAT nevű adatállományhoz, amely az A: meghajtóban levő hajlékony mágneslemezen található.
- Az *Assign* eljárás végrehajtása nem nyitja meg a *Results* változóval azonosított fájlt.

### *A Reset eljárás*

Hívása: *Reset* (*File Var*)

A *FileVar* egy fájlváltozó neve. Az eljárás hívása előtt a *FileVar* névvel azonosított fájlt egy létező adatállományhoz kell hozzárendelni.

A *Reset* eljárás megnyitja a *FileVar* névvel azonosított fájlt. A megnyitás kezdeti pozícióba (az első elem elé) állítja a fájlt.

**Példa**

```
var
  InpFile : file of record
                      Name : string[30];
                      Income : real
                    end;
...
Assign(InpFile,'INCOME.DOC');
Reset(InpFile);
```

- Az *Assign* eljárás végrehajtása hozzárendeli az *InpFile* változóval azonosított fájlt az INCOME.DOC nevű adatállományhoz, amely az aktuális meghajtó aktuális könyvtárában található.
- A *Reset* eljárás végrehajtása megnyitja az *InpFile* változóval azonosított fájlt. Ha a fájl nem létezik, hibajelzést kapunk.
- A fájl *Reset* eljárással való megnyitása nem zárja ki a rajta végezhető későbbi *Seek* és *Write* műveleteket.

### *A Rewrite eljárás*

Hívása: *Rewrite* (*File Var*)

A *FileVar* egy fájlváltozó neve. Az eljárás hívása előtt a *FileVar* által képviselt fájlt egy adatállományhoz kell hozzárendelni.

A *Rewrite* eljárás megnyitja a *FileVar* által képviselt fájlt. A megnyitás kezdeti pozícióba (az első elem elé) állítja a fájlt.

Ha a *Rewrite* hívása előtt a fájlhoz hozzárendelt állomány nem létezett, akkor az eljárás végrehajtásakor létrejön. Ha pedig volt már ilyen állomány, akkor törlődik és újra létrejön. Mindkét esetben egy üres adatállomány keletkezik.

**Példa**

```
var
  OutFile : file of array[1..20]of byte;
...
Assign(OutFile,'B:TESTS.OUT');
Rewrite(OutFile);
```

- Az *Assign* eljárás végrehajtása hozzárendeli az *OutFile* változóval azonosított fájlt a TESTS.OUT nevű adatállományhoz, amely a B: meghajtóban levő hajlékony mágneslemezen található.
- A *Rewrite* eljárás végrehajtása megnyitja az *OutFile* változóval azonosított fájlt. A megnyitás után a fájl üres lesz.
- A fájl *Rewrite* eljárással való megnyitása nem zárja ki a rajta végezhető későbbi *Seek* és *Read* műveleteket.

### *A Read eljárás*

Hívása: *Read (FileVar,VarList)*

A *FileVar* egy fájlváltozó neve, a *VarList* pedig egy változónév vagy változónév-lista. A *FileVar* névvel azonosított fájlnak nyitottnak kell lennie. A *VarList* változó(k) csak a fájltípussal azonos típusú(ak) lehet(nek).

A változónévlistát tartalmazó *Read* eljárás egyenértékű a változóneveket tartalmazó *Read* eljárások sorozatával. Egy változónévvel hívott *Read* eljárás beolvas egy elemet a fájlból, és ennek értékét a megadott nevű változónak adja át.

**Példa**

```
type
  ElmType = record
              Re,Im : real
            end;
var
  InpFile : file of ElmType;
  ArrVar : array[boolean,2..4] of ElmType;
...
Assign(InpFile,'COMPLEX.DAT');
Reset(InpFile);
Read(InpFile,Arr,Var[true,3]);
```

- Az *Assign* eljárás végrehajtása hozzárendeli az *InpFile* változóval azonosított fájlt a COMPLEX.DAT nevű adatállományhoz.
- A *Reset* eljárás végrehajtása megnyitja az *InpFile* változóval azonosított fájlt.
- A *Read* eljárás beolvas egy elemet az *InpFile* változóval azonosított fájlból, és ennek értékét átadja az *ArrVar* tömb *ArrVar*[*true*,3] elemének.

### *A Write eljárás*

Hívása: *Write* (*FileVar*,*VarList*)

*A FileVar* egy fájlváltozó neve, a *VarList* pedig egy változónév vagy változónévlista. A *FileVar* névvel azonosított fájlnak nyitottnak kell lennie. A *VarList* változó(k) csak a fájltípussal azonos típusú(ak) lehet(nek).

A változónévlistát tartalmazó *Write* eljárás egyenértékű a változóneveket tartalmazó *Write* eljárások sorozatával. Egy változónévvel hívott *Write* eljárás a megadott nevű változó értékét beírja a *FileVar* változóval azonosított fájlba.

**Példa**

```
type
  ElmType = record
              Re,Im : real
            end;
var
  OutFile : file of ElmType;
  ArrVar : array[boolean] of ElmType;
...
Assign(OutFile,'COMPLEX.RES');
Rewrite(OutFile);
Write(OutFile,ArrVar[false],ArrVar[true]);
```

- Az *Assign* eljárás végrehajtása hozzárendeli az *OutFile* változóval azonosított fájlt a COMPLEX.RES nevű adatállományhoz.
- A *Rewrite* eljárás végrehajtása megnyitja az *OutFile* változóval azonosított fájlt.
- A *Write* eljárás beírja az *OutFile* változóval azonosított fájlba az *ArrVar* tömb *ArrVar*[*false*] és *ArrVar*[*true*] elemeinek értékét.
- A fenti *Write* utasítás helyettesítése a

*Write*(*OutFile*,*ArrVar*);

utasítással hibás lenne, mert a tömb neve nem jelenti a tömb elemeinek felsorolását.

### *A Seek eljárás*

Hívása: *Seek* (*FileVar*,*PostExp*)

A *FileVar* egy fájlváltozó neve, a *PosExp* pedig egy *integer* típusú kifejezés. A *FileVar* névvel azonosított fájlnak nyitottnak kell lennie. A *PosExp* kifejezés értékének a

[0,*FileSize*(*FileVar*)] zárt intervallumon belül kell lennie. (A *FileSize* függvény értelmezését l. később, ugyanebben a szakaszban.)

A *Seek* eljárás a *PosExp* kifejezés értékének megfelelő sorszámú elem elé állítja a fájlt. Ha a kifejezés értéke 0, akkor a *Seek* eljárás a fájlt a kezdeti pozícióba állítja, ha ez az érték *FileSize*(*FileVar*), akkor pedig a végpozícióba.

**Példa**

```
type
  Name = string[5];
var
  FileVar : file of Name;
  ElmVar1,ElmVar2 : Name;
  Len,Pos : integer;
begin
  Assign(FileVar,'FAMILY');
  Reset(FileVar);
  Len := FileSize(FileVar) — 1;
    for Pos := 0 to Len shr 1 do begin
    Seek(FileVar,Pos);
    Read(FileVar,ElmVar1);
    Seek(FileVar,Len — Pos);
    Read(FileVar,ElmVar2);
    Seek(FileVar,Pos);
    Write(File,ElmVar2);
    Seek(FileVar,Len — Pos);
    Write(FileVar,ElmVar1)
  end;
  Close(FileVar)
end.
```

- A program végrehajtása megfordítja a FAMILY adatállomány elemeinek sorrendjét.
- A *FileVar* névvel azonosított fájlt a *Reset* eljárással nyitottuk meg, mivel a *Rewrite* eljárás hívása következtében elveszne a FAMILY adatállomány tartalma.

### *A Close eljárás*

Hívása: *Close* (*FileVar*)

A *FileVar* egy fájlváltozó neve. A *FileVar* névvel azonosított fájlnak nem kötelező nyitva lennie.

A *Close* eljárás lezárja a *FileVar* névvel azonosított fájlt. Ha az eljárás hívása előtt a fájl nem volt nyitva, akkor állapota nem változik.

*Megjegyzés:* a programfutás befejezése nem jelenti a *Close* eljárás(ok) automatikus hívását.

**Példa**

```
var
  NewFile : file of boolean;
begin
  Assign(NewFile,'EMPTY');
  Rewrite(NewFile);
  Close(NewFile)
end.
```

- A program végrehajtása után az aktuális lemezmeghajtóban levő mágneslemezen egy EMPTY nevű, üres állomány keletkezik.
- A későbbiekben ezt az állományt *boolean* típusú elemekkel tölthetjük fel.

### *Az Erase eljárás*

Hívása: *Erase* (*FileVar*)

A *FileVar* egy fájlváltozó neve. Ajánlott, hogy a *FileVar* névvel azonosított fájl ne legyen nyitva.

Az *Erase* eljárás törli azt az állományt, amelyet hozzárendeltünk a *FileVar* névvel azonosított fájlhoz.

**Példa**

```
var
  KillFile : file of byte;
begin
  Assign(KillFile,'SECRET.DOC');
  Erase(KillFile)
end.
```

- A program végrehajtása az aktuális meghajtóban levő mágneslemezről törli a SECRET.DOC nevű állományt.
- Ez a művelet olyan fájlra vonatkozik, amelyet hozzárendeltünk az adatállományhoz, de nem nyitottunk meg.
- A *KillFile* változóval azonosított fájl elemeinek típusa közömbös.

### *A Rename eljárás*

Hívása: *Rename* (*FileVar,StrEpx*)

A *FileVar* egy fájlváltozó neve, a *StrExp* pedig egy karakterlánc típusú kifejezés. A *FileVar* névvel azonosított fájl nem lehet nyitva.

A *Rename* eljárás átnevezi azt az állományt, amelyet hozzárendeltünk a *FileVar* névvel azonosított fájlhoz. Az állomány nevét a *StrExp* karakterlánc-kifejezés határozza meg. Az új név nem tartalmazhatja a lemezmeghajtó kijelölését, és ilyen nevű állomány nem létezhet.

**Példa**

```
var
  RenFile : file of byte
begin
  Assign(RenFile,'SECRET.DOC');
  Rename(RenFile,'SECRET.BAK')
end.
```

- A program a SECRET.DOC állomány nevét SECRET.BAK-ra változtatja.
- A *RenFile* változóval azonosított fájl elemeinek típusa közömbös.

### *Az Eof függvény*

Hívása: *Eof* (*FileVar*)

A *FileVar* egy fájlváltozó neve. A *FileVar* névvel azonosított fájlnak nyitottnak kell lennie.

Az *Eof* függvény értéke *true*, ha a *FileVar* névvel azonosított fájl végpozícióban van, egyébként *false.*

**Példa**

```
var
  FileVar : file of byte;
begin
  Assign(FileVar,'USELESS.DOC');
  Rewrite(FileVar);
  Writeln(Eof(FileVar))
end.
```

- Közvetlenül a *Rewrite* eljárás végrehajtása után a fájl egyszerre kezdeti és végpozícióba kerül.
- A *Writeln*(*Eof*(*FileVar*)) utasítás vágrehajtása a TRUE felirat kiírását eredményezi.

### *A FileSize függvény*

Hívása: *FileSize* (*FileVar*)

A *FileVar* egy fájlváltozó neve. A *FileVar* névvel azonosított fájlnak nyitottnak kell lennie.

A *FileSize* függvény értéke *integer* típusú adat, amely a *FileVar* változóval azonosított fájl pozícióját adja meg. A kezdeti fájlpozíció sorszáma 0, a végpozícióé pedig *FileSize* (*FileVar*).

**Példa**

```
    FileVar : file of (Red,Green,Blue);
begin
    Assign(FileVar,'COLORS');
    Reset(FileVar);
    Writeln(FilePos(FileVar) = 0)
end.
```

- Ha a COLORS állomány üres, akkor a program végrehajtásakor TRUE, egyébként pedig FALSE íródik ki.

## 12.2. Szövegfájlok

A többi fájltípustól eltérően a szövegfájlok nem azonos típusú elemekből, hanem karakterekből összetett sorokból állnak. A szövegfájl minden sora CR/LF (carriage return/line feed, „kocsi vissza"/soremelés) karakterpárral zárul. A fájl utolsó sorát egy Ctrl-Z karakter követi.

Mivel a fájl sorai különböző hosszúságúak lehetnek, feldolgozni csak szekvenciálisan tudjuk; a fájl csak olvasásra (*Reset*), ill. írásra (*Rewrite*) nyitható meg.

A szövegfájltípust az előre definiált *text* azonosítóval írhatjuk le.

*Szintaxis*

szövegfájltípus-leírás:
    text

**Példa**

```
type
    TextType =text;
var
    OutFile : TextType;
    InpFile : text;
```

- A *TextType* típusú adatokat karakterekből álló sorok alkotják; a sorok CR/LF karakterpárral zárulnak.
- Az *OutFile* és az *InpFile* fájlváltozó szövegfájlokat azonosít.

A Turbo Pascal nyelvben külső berendezéseken — konzolon, terminálon, nyomtatón és modemen — keresztül szövegfájlokkal lehet adatokat átvinni. Ezek a szövegfájlok tehát az e berendezéseken át elérhető fizikai állományok modelljei.

A külső berendezéseket három betűből és egy kettőspontból álló szimbolikus névvel (logikai eszköznévvel) jelöljük.

### *CON: konzol*

A konzol beviteli, ill. kiviteli berendezés, amelynek beviteli eleme a billentyűzet, kiviteli eleme pedig a képernyő. A konzolról bevitt adat pufferelt, azaz teljes sorok formájában olvashatjuk be. Mivel minden sor CR („kocsi vissza") karakterrel végződik, bevihetünk a konzolról egy karaktersorozatot, s mielőtt lezárnánk CR karakterrel, átszerkeszthetjük. A következő karakterekkel szerkeszthetünk:

Ctrl-H, Backspace:

Törli a kurzor bal oldalán álló karaktert, és a kurzor egy pozícióval balra ugrik.

Ctrl-X:

Törli a kurzor bal oldalán álló összes karaktert, és a kurzor a sor első pozíciójára ugrik.

Ctrl-D:

Bemásolja a kurzor pozíciójába az előző sor megfelelő karaktereit, és a kurzor egy pozícióval jobbra ugrik.

Ctrl-R:

A kurzor pozíciójától kezdve bemásolja az előző sor megfelelő karaktereit, és a kurzor jobbra ugrik, az utolsó átmásolt karakter mögé.

CR, Ctrl-M:

Befejezi a sorbeírást, és a beviteli pufferba elhelyezi a CR/LF karakterpárt.

Ctrl-Z:

Befejezi a sorbeírást, és a beviteli pufferba elhelyezi a Ctrl-Z karaktert.

*Megjegyzés:* a konzol beviteli pufferének méretét az előre definiált *BufLen* változó határozza meg. E változó max. és alapértelmezés szerinti (default) értéke 127. Ha a *BufLen* változónak értéket adunk, akkor az csak a legközelebbi beviteli utasításig lesz érvényben; a bevitel után a *BufLen* újból a 127-es értéket veszi fel.

### *TRM: terminál*

A terminál beviteli, ill. kiviteli berendezés, amelynek beviteli eleme a billentyűzet, kiviteli eleme pedig a képernyő. A konzoltól eltérően az adatbevitel nem pufferelt, ami azt jelenti, hogy minden bevitt karaktert azonnal feldolgoz, és ki is ír a képernyőre. A vezérlőkarakterek közül ez csak a CR karakterre vonatkozik, amely CR/LF karakterpárként íródik ki.

### *KBD: billentyűzet*

A billentyűzet beviteli berendezés. A bevitt karakterek a konzol beviteli eleméből származnak, nem pufferoltak, és nem íródnak ki a képernyőre.

*LST: nyomtató*

A nyomtató kiviteli berendezés. A kivitt karaktereket a rendszer nem pufferolja, ami azonban megoldható a nyomtatóban.

A szövegfájlt az *Assign* eljárással rendelhetjük hozzá egy logikai eszközhöz. Hívásakor meg kell adni a fájlváltozó nevét, és a logikai eszköz nevét meghatározó karakterlánc-kifejezést. A fájl-adatállomány jellegű hozzárendeléstől eltérően a szövegfájl hozzárendelése egy logikai eszközhöz automatikusan meg is nyitja a szövegfájlt. Ekkor felesleges a *Reset* és a *Rewrite* eljárás használata; végrehajtásuk — a *Close* eljáráshoz hasonlóan — nem jár semmilyen következménnyel. Hibás azonban pl. az *Erase* vagy a *Rename* eljárás használata, mivel ezek csak a mágneslemezes háttértárban található állományokon értelmezhetők.

**Példa**

```
var
  Console : text;
...
Assign(Console,'CON:');
```

- Az *Assign* eljárás végrehajtása hozzárendeli a *Console* fájlváltozóval azonosított szövegfájlt a CON: logikai eszközhöz, azaz a konzolhoz, ugyanakkor meg is nyitja a fájlt.

- Mivel egy logikai eszközhöz hozzárendelt fájl lezárása nem jár következménnyel, a példában bemutatott szövegfájlt csak egy újabb *Assign* eljárással — pl. *Assign (Console,*'KBD:') — lehet lezárni.

A logikai eszközhöz hozzárendelt szövegfájlok esetén — az egyszerűsítésre törekedve — a Turbo Pascal nyelvben több előre definiált fájlváltozót vezettek be, amelyek szövegfájlokat azonosítanak.

*3. táblázat. Szövegfájlok és logikai eszközök egymáshoz rendelése*

| *Fájlváltozó* | *Logikai eszköz* |
|---|---|
| *Input* | CON: vagy TRM: |
| *Output* | CON: vagy TRM: |
| *Con* | CON: |
| *Trm* | TRM: |
| *Kbd* | KBD: |
| *Lst* | LST: |

Mint a *3. táblázatból* kiderül, a konkrét logikai eszközök hozzákapcsolódnak az előre definiált fájlváltozók nevéhez. E szabály alól kivétel az *Input* és az *Output* fájlváltozó: mindegyikük akár a CON:, akár a TRM: logikai eszközhöz hozzárendelt fájlt azonosíthatja.

A CON: és a TRM: közötti választást a fordítóprogram {$B+} és {$B—} direktívái befolyásolják. Alapértelmezés szerint a {$B+} van érvényben; ekkor az *Input* és *Output* fájlváltozóval azonosított fájlok a CON: logikai eszközhöz kapcsolódnak. A {$B—} direktíva viszont a TRM: eszközt választja a CON: helyett.

A felsorolt fájlváltozókkal *(Input, Output, Con, Trm, Kbd* és *Lst)* azonosított szövegfájlok mindig nyitottak, és a rajtuk végezhető műveletek a konkrét logikai eszközökre vonatkoznak (CON:, TRM:, KBD: és LST:).

**Példa**

```
begin
  Writeln(Con,'-Hello world-')
end.
```

- A *Con* előre definiált fájlváltozó a konzolhoz hozzárendelt fájlt jelent.
- A program kiírja a konzolra a „-Hello World-" feliratot.

## 12.3. Szövegfájlokon értelmezett műveletek

A szövegfájlokat karakterekből álló sorok alkotják. A szövegfájl minden sora CR/LF karakterpárral zárul, a fájl utolsó karaktere pedig Ctrl—Z. Ez a karakter a fájl lezárásakor kerül az írásra vagy bővítésre megnyitott fájlba.

A szövegfájlok adatállományokhoz vagy logikai eszközökhöz rendelhetők hozzá. Az első esetben a fájlban tárolt adatok feldolgozását meg kell előznie az *Assign* és a *Reset* vagy a *Rewrite* eljárás hívásának. A feldolgozás végét a *Close* eljárás jelenti. A második esetben (vagyis fájl és eszköz társításakor) használhatjuk az előre definiált fájlváltozót. Ekkor nem megengedett az ilyen változóra vonatkozó *Assign, Reset, Rewrite* és *Close* eljárás hívása. Jó megoldás az is, hogy az eszközt egy szimbolikus nevet viselő adatállománynak tekintjük, és az *Assign* eljárással társítjuk a szövegfájllal (utána már kezelhetjük).

### *Az Assign eljárás*

Hívása: *Assign (TextVar,StrExp)*

A *TextVar* egy *text* típusú fájlváltozó neve, a *StrExp* pedig karakterlánc típusú kifejezés. A *TextVar* névvel azonosított fájl nem lehet nyitva. A *TextVar* nem lehet egy előre definiált fájlváltozó.

Az *Assign* eljárás végrehajtása hozzárendeli a *TextVar* névvel azonosított fájlt a *StrExp* kifejezéssel meghatározott adatállományhoz vagy logikai eszközhöz.

**Példa**

```
var
  Device : text;
...
Assign(Device,'CON:');
```

- A *Device* fájlváltozó egy szövegfájlt jelöl.
- Az *Assign* eljárás végrehajtása hozzárendeli a *Device* változóval azonosított fájlt a konzolhoz.
- Az *Assign* eljárás végrehajtása megnyitja a *Device* változóval azonosított fájlt.

### *A Reset eljárás*

Hívása: *Reset (TextVar)*

A *TextVar* egy *text* típusú fájlváltozó neve. Az eljárás hívása előtt a *TextVar* névvel azonosított fájlt egy létező adatállományhoz vagy logikai eszközhöz hozzá kell rendelni.

A *Reset* eljárás megnyitja a *TextVar* névvel azonosított fájlt. Ha ez a fájl egy logikai eszközhöz kapcsolódik, akkor már megnyitottnak tekinthető, és a *Reset* eljárás hívása nem jár következménnyel.

**Példa**

```
var
  InpFile : text ;
...
Assign(InpFile,'OLDBOOK');
Reset(InpFile);
```

- Az *Assign* eljárás végrehajtása hozzárendeli az *InpFile* változóval azonosított fájlt az OLDBOOK nevű adatállományhoz.
- A *Reset* eljárás végrehajtása olvasásra nyitja meg az *InpFile* változóval azonosított szövegfájlt.

### *A Rewrite eljárás*

Hívása: *Rewrite (TextVar)*

*A TextVar* egy *text* típusú fájlváltozó neve. Az eljárás hívása előtt a *TextVar* által képviselt fájlt egy adatállományhoz vagy egy logikai eszközhöz kell hozzárendelni.

A *Rewrite* eljárás megnyitja a *TextVar* által képviselt fájlt. Ha ez a fájl egy logikai eszközhöz kapcsolódik, akkor már megnyitottnak tekinthető, és a *Rewrite* eljárás hívása nem jár következménnyel. Ha a *Rewrite* hívása előtt a fájlhoz hozzárendelt állomány nem létezett, akkor az eljárás végrehajtásakor létrejön. Ha pedig volt már ilyen állomány, akkor törlődik és újra létrejön. Mindkét esetben egy üres adatállomány keletkezik.

**Példa**

```
var
  OutFile : text;
...
Assign(OutFile,'NEWBOOK');
Rewrite(OutFile);
```

- Az *Assign* eljárás végrehajtása hozzárendeli az *OutFile* változóval azonosított szövegfájlt a NEWBOOK nevű adatállományhoz.
- A *Rewrite* eljárás végrehajtása írásra nyitja meg az *OutFile* változóval azonosított fájlt. A megnyitás után ez a fájl üres lesz.

### *A Read eljárás*

Hívása: *Read (TextVar,VarList)*

A *TextVar* egy *text* típusú fájlváltozó neve, a *VarList* pedig egy változónév, ill. *char, string, integer* vagy *real* típusú változónevek listája. Ha a *TextVar* értéke *Input,* akkor a hívás *Read(VarList)* alakra egyszerűsíthető. Ha a hívás explicit vagy „rejtett" *Input* paramétert tartalmaz, és a {$B+} direktíva van érvényben, akkor a *TextVar* nevet a fordítóprogram *Con* névnek tekinti; ha a {$B—} érvényes, akkor pedig *Trm* névnek. A *TextVar* névvel azonosított fájlnak mindig nyitottnak kell lennie.

A *Read* eljárás a *TextVar* változóval azonosított fájlból karakterláncot olvas be, melyet — megállapodás szerinti — adat(ok)nak értelmez; értékét a *VarList*-ben szereplő változó(k)nak adja át.

*Megjegyzés:* ha a bevitel a CON: logikai eszközről megy végbe, akkor a *Read* mindenkori hívása egy új sor beolvasását jelenti, még akkor is, ha a konzol pufferében maradt(ak) még be nem olvasott karakter(ek). Ekkor a pufferben található karakterek csak a CR karakteres sorlezárás után értelmezhetők; az esetleg korábban bevitt Ctrl—Z karakter figyelmen kívül marad. Ez a karakter egyébként mindig a puffer végére kerül, de csak a CR bevitele után.

A *TextVar* változóval azonosított fájl karaktereinek értelmezése függ a *VarList*-beli változó(k) típusától.

*1.* A *char* típusú változók

A rendszer beolvas egy karaktert, és átadja a változónak.

*2.* A *string* típusú változók

A rendszer beolvassa a lehetséges leghosszabb karakterláncot, és átadja a változónak. A bevitt karakterek száma nem lépheti túl a karakterlánc-változó max. hosszát, és nem lehet több az aktuális sor karakterszámánál.

*3.* Az *integer,* ill. a *real* típusú változók

A rendszer beolvas egy egész, ill. valós számot ábrázoló karakterláncot, amely után a következő karakterek valamelyike következik: szóköz, tabulátor, CR vagy Ctrl—Z. A beolvasandó számot megelőző szóközök, tabulátorok, Cr és LF karakterek

figyelmen kívül maradnak. A soron következő *VarList*-beli változó a maradó karakterekkel ábrázolt következő egész, ill. valós számot kapja értékül stb. Ha a beolvasott karakterlánc a leírástól eltérő alakú, akkor beviteli vagy kiviteli hiba keletkezik.

**Példa**

```
var
  IntVar : integer;
  FixVar : real;
  TextVar : text;
  Ter1,Ter2 : char;
  Str : string[8];
begin
  Assign(TextVar,'ONELINE.DAT');
  Reset(TextVar);
  Read(TextVar,IntVar,Ter1,FixVar,Ter2,Str);
...
end.
```

- Tegyük fel, hogy az ONELINE.DAT állomány első sora a következő alakú:

12 —13.8 Izabela

Ekkor az *IntVar* változó új értéke 12, a *FixVar* változóé —13.8, a *Str* változóé pedig 'Izabela' lesz.

- A *Ter1* és a *Ter2* változó egy szóközt kap értékül. Ez azt jelenti, hogy a számot határoló karaktert a rendszer kétszer értelmezi: a számot ábrázoló karakterlánc utolsó (záró) karaktereként, ill. a szám után következő karakterlánc első karaktereként is.

Ha a *Read* eljárás végrehajtásakor a fájl végpozícióba kerül, és a *VarList* változói közül maradt még olyan, amely nem kapott értéket, akkor minden *char* típusú változó Ctrl—Z karaktert kap értékként, a *string* típusú változó üres karakterláncot, az *integer* és a *real* típusú változók pedig nem kapnak új értéket.

**Példa**

```
var
  IntVar : integer;
  FixVar ; real;
  TextVar : text;
  Ter1,Ter2 : char;
  Str : string[4];
begin
  IntVar:=—44;
  FixVar:=—44;
  Ter1:='x';
  Ter2:='y';
  Str:='janb';
```

```
Assign(TextVar,'SHORT.DAT');
Reset(TextVar);
Read(TextVar,IntVar,Ter1,FixVar,Ter2,Str);
...
end.
```

• Tegyük fel, hogy a SHORT.DAT állomány a következő karakterekből áll: 1, 3, szóköz, CR, LF, Ctrl—Z. Ekkor a *Read* utasítás végrehajtása után az *IntVar* változó új értéke 13, a *FixVar* változóé —44.0, a *Ter1* változóé #32 (szóköz), a *Ter2* változóé #26 (Ctrl—Z), a *Str* változóé pedig '' lesz.

Ha a fájlt a konzolhoz rendeljük hozzá, akkor minden beolvasott sor végén Ctrl—Z karakter lesz. Ennek az a hatása, mintha a fájl minden sor végén végpozícióba kerülne.

**Példa**

```
var
  IntVar : integer;
  StrVar : string[3];
  ChrVar : char;
begin
  IntVar:=—2;
  StrVar:='jan';
  ChrVar:='b';
  Read(IntVar,StrVar,ChrVar);
...
end.
```

• A *Read* eljárás hívását a rendszer a következő hívásként értelmezi:

*Read (Input,IntVar,StrVar,ChrVar);*

• Mivel ez a hívás az alapértelmezés szerinti {$B+} direktíva hatáskörében van, egyenértékű a

*Read (Con,IntVar,StrVar,ChrVar);*

hívással.

• Ha a *Read* eljárás végrehajtásakor a konzolról egyetlen CR karakter érkezik, akkor ezzel befejeződik az utasítás végrehajtása. E pillanatban az *IntVar* változó értéke —2, a *StrVar* változóé '', a *ChrVar* változóé pedig #26 (Ctrl-Z) lesz.

### *A Readln eljárás*

Hívása: *Readln (TextVar)*
*Readln (TextVar,VarList)*

A *TextVar* egy *text* típusú fájlváltozó neve, a *VarList* pedig egy változónév, ill. *char, string, integer* vagy *real* típusú változónevek listája. Ha a *TextVar* értéke *Input,* akkor a hívás egyszerűsíthető: az első esetben *Readln,* a másodikban *Readln(VarList)* alakra.

A *Readln (TextVar)* típusú hívás karaktereket olvas a fájlból, CR/LF-ig (bezárólag); a beolvasott karakterláncot figyelmen kívül hagyja. Ha a fájl egy logikai eszközhöz kapcsolódik, akkor csak CR-ig (bezárólag) olvassa be a karaktereket, és a beolvasott karakterláncot figyelmen kívül hagyja. Ha a *TextVar* konzolt jelent, akkor a képernyőre CR/LF-et ír ki. A

*Readln (TextVar,VarList)*

hívás egyenértékű a

*Read (TextVar,VarList); Readln(TextVar)*

alakkal, ezért nem igényel további magyarázatot.

**Példa**

```
var
  Count : integer;
  Source : text;
  Line : string[128];
begin
  Assign(Source,'VOLUME.DAT');
  Reset(Source);
  Count:=0;
  while not Eof(Source) do begin
    Readln(Source,Line);
    Count := Count+1
  end;
  Writeln(Count)
end.
```

- A program kiírja a VOLUME.DAT adatállomány sorainak számát.

### *A Write eljárás*

Hívása: *Write (TextVar,ExpList)*

*A TextVar* egy *text* típusú fájlváltozó neve, az *ExpList* pedig *char, string, integer, real* vagy *boolean* típusú változó neve (vagy ilyen nevek listája). Közvetlenül a felsorolt kifejezések után következhet a *:m* alakú minősítő tag; *real* típusú kifejezés után megengedett még a *:m:n* alakú minősítő is, ahol az *m* és az *n* integer típusú kifejezés. Ha a *TextVar* értéke *Output,* akkor a hívás *Write(ExpList)* alakra egyszerűsíthető. Ha a hívás explicit vagy „rejtett" *Output* paramétert tartalmaz, akkor a *TextVar* nevet a fordítóprogram *Con* (vagy a megfelelő programkörnyezetben ezzel egyenértékű *Trm*) névnek tekinti. A *TextVar* névvel azonosított fájlnak mindig nyitottnak kell lennie.

A *Write* eljárás a *TextVar* változóval azonosított fájlba karakterláncot ír be, amely az *ExpList*-ben szereplő kifejezések értékét képviselő adatokat ábrázolja.

Az említett karakterlánc alakjának értelmezése az *ExpList*-beli változó(k) típusától és az *m, n* minősítők értékétől függ.

*1.* A *char* típusú kifejezések

A rendszer kiviszi a kifejezés által képviselt karaktert. Ha van *m* minősítő, és $m > 1$, akkor a karaktert $m-1$ szóköz előzi meg.

*2.* A *string* típusú kifejezések

A rendszer kiviszi a kifejezés által képviselt karakterláncot. Ha van *m* minősítő, és *m* nagyobb a karakterlánc *l* hosszánál, akkor a karakterláncot $m-l$ szóköz előzi meg.

*3.* Az *integer* típusú kifejezések

A rendszer kiviszi a kifejezés értékét ábrázoló legrövidebb karakterláncot. Ha van *m* minősítő, és *m* nagyobb a karakterlánc *l* hosszánál, akkor a karakterláncot $m-l$ szóköz előzi meg.

*4.* A *real* típusú kifejezések

A rendszer kiviszi a kifejezés valós értékét ábrázoló *bsd.ddddddddddEsdd* alakú karakterláncot, amelyben a *b* szóközt jelent; az első *s* a szám előjelét ábrázolja, mégpedig szóközként (nemnegatív érték esetén) vagy — (mínusz) karakterként (negatív érték esetén). Minden *d* egy decimális számjegyet jelent; a második *s* pedig egy + vagy — (mínusz) karaktert. Ha van *m* minősítő, és $m > 18$, akkor a karakterláncot $m-18$ szóköz előzi meg. Ha $m = 18$, akkor a 4. pont elején megadott minta szerint keletkezik a karakterlánc. Ha $m < 18$, akkor pontosan *m* karaktert visz ki a rendszer, mégpedig úgy, hogy először a kezdő szóközöket, utána pedig a mantissza utolsó számjegyeit hagyja el, kerekítve. Ha az *m* értéke 2-nél kevesebb számjegyet hagyna a mantisszából, akkor a rendszer csak a harmadik és az utána következő számjegyeket vágja le, kerekítve. Ha a minősítő tag *:m:n* alakú, akkor a rendszer olyan valós számformátumot képez, amely exponens nélküli, *n* darab tizedesjegyű, jobbra igazított, *m* hosszúságú karakterlánc. Ha a szám nem ábrázolható ilyen formában, akkor az *m* értéke automatikusan annyival nő, hogy a számot ábrázolni lehessen. Ha $n = 0$, akkor a rendszer a számot tizedespont és tizedesrész nélkül viszi ki; a [0,24] zárt intervallumon kívülre eső *n* értékek esetén a *:m:n* minősítőt a rendszer — az előzőekben leírt — *:m* tagként értelmezi.

*5.* A *boolean* típusú kifejezések

A rendszer kiviszi a kifejezés értékét képviselő TRUE vagy FALSE karakterláncot. Ha van *m* minősítő, és *m* nagyobb a képzett karakterlánc *l* hosszánál, akkor a karakterláncot $m-l$ szóköz előzi meg.

**Példák**

(A b szimbólum szóközt jelent.)

| *Kifejezés* | *Kivitt karakterlánc* |
|---|---|
| 'j' | j |
| 'j':2 | bj |
| 'jan' | jan |
| 'jan':4 | bjan |
| 44 | 44 |

| | |
|---|---|
| −44 | −44 |
| 44:4 | bb44 |
| 23.5 | bb2.3500000000E+01 |
| 23.456789:10 | 2.3457E+01 |
| 23.456789:6 | 2.3E+01 |
| 23.456789:6:2 | b23.46 |
| −23.456789:6:0 | bbb−23 |
| 23.456789:6:−2 | 2.3E+01 |

- Láthatjuk, hogy az első elhagyott számjegy alapján lehet kerekíteni.

### *A Writeln eljárás*

Hívása: *Writeln (TextVar)*
*Writeln (TextVar,ExpList)*

*A TextVar* egy *text* típusú fájlváltozó neve, az *ExpList* pedig *char, string, integer, real* vagy *boolean* típusú változó neve (vagy ilyen nevek listája). Közvetlenül a felsorolt kifejezések után következhet a *:m* alakú minősítő tag; egy *real* típusú kifejezés után megengedett még a *:m:n* alakú minősítő is, ahol az *m* és az *n integer* típusú kifejezés. Ha a *TextVar* értéke *Output,* akkor a hívás egyszerűsíthető: az első esetben *Writeln,* a másodikban *Writeln(ExpList)* alakra.

A *Writeln(TextVar)* típusú hívás a *TextVar* változóval azonosított fájlba CR/LF karakterpárt visz ki. A

*Writeln(TextVar,ExpList)*

hívás egyenértékű a

*Write(TextVar,ExpList); Writeln(TextVar)* alakkal, ezért nem igényel további magyarázatot.

**Példa**

```
begin
  Writeln(false);
  Writeln;
  Writeln(true)
end.
```

- A program három sort ír ki a konzolra.
- A második kivitt sor üres.
- Összesen 15 kivitt karakter van: a FALSE és a TRUE felirat, valamint három CR/LF karakterpár.

### *A Close eljárás*

Hívása: *Close (TextVar)*

A *TextVar* egy *text* típusú fájlváltozó neve. A *TextVar* nem lehet előre definiált fájlváltozó.

A *Close* eljárás lezárja a *TextVar* névvel azonosított fájlt.

**Példa**

```
var
  Output : text;
begin
  Assign(Output,'JB.DOC');
  Rewrite(Output);
  Writeln(Output,'jasio');
  Close(Output)
end.
```

- A nyílt deklaráció miatt az *Output* nem egy előre definiált fájlváltozó neve, ezért az *Assign* eljárás hívása helyes és szükség is van rá.
- A program összesen nyolc karaktert tartalmazó állományt hoz létre.
- A *Close* eljárás hívása kivisz egy Ctrl—Z karaktert és lezárja a JB.DOC állományt, amely a következő karakterláncot tartalmazza: jasio CR LF Ctrl—Z.

### *Az Eof függvény*

Hívása: *Eof (TextVar)*

A *TextVar* egy *text* típusú fájlváltozó neve. Ha a *TextVar* értéke *Input*, akkor a hívás *Eof* alakúra egyszerűsíthető.

Az *Eof* függvény adatállományhoz és logikai eszközhöz hozzárendelt fájlra is alkalmazható. A függvény mindkét esetben *boolean* típusú, és *false* vagy *true* értéket vehet fel.

Ha a fájl egy állományhoz kapcsolódik, és a Ctrl—Z karakter előtt vagy a végpozícióban állt, akkor az *Eof* függvény értéke *true,* egyébként *false.*

Ha a fájl egy logikai eszközhöz kapcsolódik, és az utolsó értelmezett karakter Ctrl—Z volt, akkor az *Eof* függvény értéke *true,* egyébként *false.*

**Példák**

```
var
  FileVar : text;
begin
  Assign(FileVar,'EMPTY');
  Rewrite(FileVar);
  Writeln(Eof(FileVar));
  Close(FileVar)
end.
```

● A program FALSE feliratot ír ki, mert a *FileVar* változóval azonosított fájl kezdőpozícióban van.

### *Az Eoln függvény*

Hívása: *Eoln (TextVar)*

A *TextVar* egy *text* típusú fájlváltozó neve. Ha a *TextVar* értéke *Input,* akkor a hívás *Eoln* alakúra egyszerűsíthető.

Az *Eoln* függvény adatállományhoz és logikai eszközhöz hozzárendelt fájlra is alkalmazható. A függvény mindkét esetben *boolean* típusú, és *false* vagy *true* értéket vehet fel.

Ha a fájl egy állományhoz kapcsolódik, és a CR karakter előtt áll, vagy az *Eof* függvény értéke szintén *true* lenne, akkor az *Eoln* függvény értéke *true,* egyébként *false.*

Ha a fájl egy logikai eszközhöz kapcsolódik és az utolsó értelmezett karakter CR volt, vagy az *Eof* függvény értéke szintén *true* lenne, akkor az *Eoln* függvény értéke *true,* egyébként *false.*

**Példa**

```
var
  ChrVar : char;
  TxtVar : text;
begin
  Assign(Txt,Var,'STRANGE.DOC');
  Reset(TxtVar);
  while not Eof(TxtVar) do begin
    Writeln(Eoln(TxtVar));
    Read(TxtVar,ChrVar)
  end
end.
```

● Feltételezzük, hogy a STRANGE.DOC állomány egyetlen üres sorból (azaz CR, LF, Ctrl—Z karakterekből) áll. Ekkor a program

```
TRUE
FALSE
```

feliratot ír ki, mert az LF karakter előtti pozícióban az *Eoln* függvény *false* értékű.

### *A SeekEof függvény*

Hívása: *SeekEof (TextVar)*

A *TextVar* egy *text* típusú fájlváltozó neve. Ha a *TextVar* értéke *Input,* akkor a hívás *SeekEof* alakúra egyszerűsíthető.

A *SeekEof* függvény működése hasonlít az *Eof* függvényre: a legközelebbi szóközök, tabulátorok, CR és LF karakterek elhagyása után az eredmény az *Eof* függvényével azonos.

**Példa**

```
var
  TxtVar : text;
begin
  Assign(TxtVar,'STRANGE.DOC');
  Reset(TxtVar);
    Writeln(SeekEof(TxtVar))
end.
```

• Feltételezzük, hogy a STRANGE.DOC állomány egyetlen üres sorból áll. Ekkor a program TRUE feliratot ír ki.

### *A SeekEoln függvény*

Hívása: *SeekEoln (TextVar)*

A *TextVar* egy *text* típusú fájlváltozó neve. Ha a *TextVar* értéke *Input,* akkor a hívás *SeekEoln* alakúra egyszerűsíthető.

A *SeekEoln* függvény működése hasonlít az *Eoln* függvényre: a legközelebbi szóközök és tabulátorok elhagyása után az eredmény az *Eoln* függvényével azonos.

**Példa**

```
var
  TxtVar : text;
  Tally : integer;
  ChrVar : char;
begin
  Tally:=0;
  Assign(TxtVar,'TEXT.DOC');
  Reset(TxtVar);
  while not SeekEof(TxtVar) do begin
    while not SeekEoln(TxtVar) do begin
    Tally:=Tally + 1;
    Read(TxtVar,ChrVar)
    end
  end;
```

```
    Writeln(Tally)
end.
```

• A program kiírja a TEXT.DOC nevű állomány karaktereinek számát; a sorzáró szóközöket, tabulátorokat, a CR, LF és Ctrl—Z karaktereket nem veszi figyelembe.

## 12.4. Elemtípus nélküli fájlok

Az elemtípus nélküli fájlok lehetőséget adnak a nem pufferolt beviteli, ill. kiviteli műveletek végrehajtására, közvetlenül a program változói és a külső mágneslemezes háttértár között. Az elemek — megállapodás szerint — olyan objektumok, amelyek 128 bájtnyi memóriát foglalnak le. Az elemtípus nélküli fájl tetszőleges mágneslemezes adatállományt képviselhet; ezért pl. az *Erase* vagy a *Rename* jellegű műveletek is végrehajthatók.

Az elemtípus nélküli fájl leírása a **file** kulcsszóból áll.

*Szintaxis*

```
elemtípus_ nélküli_ fájltípus
  file
```

**Példa**

```
var
  DiskFile : file;
  FileName : string[40];
begin
  Write('FileName:—');
  Readl(FileName);
  Assign(DiskFile,FileName);
  Erase(DiskFile)
end.
```

• A program a konzol billentyűzetéről beolvassa egy tetszőleges mágneslemezes állomány nevét, s utána törli ezt az állományt.

• Az állomány elemtípusa közömbös.

Az elemtípus nélküli fájlokon végrehajtható beviteli, ill. kiviteli műveleteket a *BlockRead* és a *BlockWrite* eljárás valósítja meg; ezek helyettesítik a megfelelő elemtípusos eljárásokat. Más műveletek *(Assign, Reset, Rewrite, Close, Seek* és *Eof)* értelmezése azonos az elemtípusos fájlokéval.

*A BlockRead eljárás*

Hívása: *BlockRead (FileVar,Buffer,Count,Reply)*
*BlockRead (FileVar,Buffer,Count)*

A *FileVar* egy elemtípus nélküli fájlváltozó neve, a *Buffer* tetszőleges változónév, a *Count integer* típusú kifejezés, a *Reply* pedig *integer* típusú változó neve.

A *BlockRead* eljárás a *FileVar* változóval azonosított fájlból *Count* számú, 128 bájt méretű objektumot olvas be, a *Buffer* változó által lefoglalt memóriaterületre. Ha a hívás tartalmazza a *Reply* paramétert, akkor a *Reply* változó új értéke a ténylegesen beolvasott objektumok számát adja meg. Ha ez a szám *Count*-nál kisebb, akkor a fájl a végpozícióban van.

*A BlockWrite eljárás*

Hívása: *BlockWrite (FileVar,Buffer,Count,Reply)*
*BlockWrite (FileVar,Buffer,Count)*

A *FileVar* egy elemtípus nélküli fájlváltozó neve, a *Buffer* tetszőleges változónév, a *Count integer* típusú kifejezés, a *Reply* pedig egy *integer* típusú változó neve.

A *BlockWrite* eljárás a *FileVar* változóval azonosított fájlba *Count* számú, 128 bájt méretű objektumot ír be, a *Buffer* változó által lefoglalt memóriaterületről. Ha a hívás tartalmazza a *Reply* paramétert, akkor a *Reply* változó új értéke a ténylegesen kivitt objektumok számát adja meg. Ha ez a szám *Count*-nál kisebb, akkor a kivitel sikertelen volt.

**Példa**

```
program Copy;
var
  Src,Trg : file;
  Buffer : array[0..255,boolean] of byte;
  Source,Target : string[40];
  Reply : integer;
begin
  Write('Source:—');
  Readln(Source);
  Write('Target:—');
  Readln(Target);
  Assign(Src,Source);
  Assign(Trg,Target);
  Reset(Src);
  Rewrite(Trg);
  repeat
    BlockRead(Src,Buffer,4,Reply);
    BlockWrite(Trg,Buffer,Reply)
```

```
    until Reply < 4;
    Close(Src);
    Close(Trg)
end.
```

- A program tetszőleges elemtípusú adatállományt másol.

## 12.5. Beviteli és kiviteli műveletek ellenőrzése

A beviteli és kiviteli műveletek ellenőrzése a program fordításakor kérhető. Az alapértelmezés szerinti {$I+} direktíva érvényességi körén belül a rendszer minden beviteli és kiviteli műveletet ellenőriz. Hiba esetén a program leáll, és hibaüzenet íródik ki a konzolra. A {$I—} direktíva hatáskörében a hibaállapot nem állítja le a program futását, csak felfüggeszti a további beviteli, ill. kiviteli műveleteket. Ez az állapot a beépített *IOresult* függvény hívásáig áll fenn. A függvény *integer* típusú értéke megadja a hiba jellegét. Ha ez az érték 0, akkor az eddigi beviteli, ill. kiviteli műveletek hibátlanul végződtek; ha a függvény 0-tól különböző értékkel tér vissza, akkor ez az érték — a B Függelékben felsoroltak szerint — a hiba kódját jelenti.

**Példa**

```
program Delete;
var
   FileVar : file;
   FileName : string[40];
   Flag : Boolean;
begin
   Write('FileName:—');
   Readln(FileName);
   Assign(FileVar,FileName);
   {SI—} Erase(FileVar); {$I+}
   if not (IOresult = 0) then
      Writelnt('File'+FileName+' did not exist')
end.
```

- A program a konzolról beolvassa egy állomány nevét, s utána törli ezt az állományt.
- Ha egy nem létező állomány nevét adtuk meg, akkor erről hibaüzenetet kapunk.
- Ha egy programból kihagynánk a direktívákat, és a törlendő állomány nem létezik, akkor a program az *Erase* műveletnél „rendszerbeli hiba" üzenettel leállna.

# 13. Mutatótípusok

A mutatótípusok mindegyike elemi típus. A mutató típusú adat értéke a *mutató* (pointer). A mutatótípus leírása a ^ karakterből és az azt követő típusnévből áll. E típusnevet nem kötelező előbb definiálni; ez az egyetlen kivétel azon szabály alól, miszerint csak az előzőleg definiált objektumokra hivatkozhatunk.

*Szintaxis*

mutatótípus-leírás:
^ típusnév
típusnév:
azonosító

**Példa**

```
type
  EmployeePtrs = ^EmployeeData;
  EmployeeData = record
                   Name : string[30];
                   Salary : real
                 end;
var
  EmployeeRef : EmployeePtr;
  IntRef :^integer;
```

- Az *EmployeePtr* típus az *EmployeeData* típusú adatokra mutat.
- Az *EmployeeRef* egy mutató típusú változó neve. Ez a változó *EmployeeData* típusú változókra mutató adatokat kaphat értékül.
- Az *IntRef* szintén egy mutató típusú változó neve. Ez a változó *integer* típusú változókra mutató adatokat kaphat értékül.

A mutatótípusokon és -változókon kívül van egy „üres" mutató is a Turbo Pascalban, melyet a **nil** kulcsszó képvisel. E mutató típusa egyenértékű bármilyen más mutatótípussal. A mutató típusú változók neveit és az üres mutatót az = és a <> (nem egyenlő) relációkban használhatjuk. Az értékadó utasításban kötelező a bal és a jobb oldal típusazonossága. Ez csak akkor teljesül, ha az értékadó szimbólum két oldalán álló nevek azonos típusú változókat képviselnek.

**Példa**

```
type
  Days = (Mon,Tue,Wed,Thu,Fri,Sat,Sun);
  Week = Mon..Sun;
var
  Day1 :^Days;
  Day2,Day3 :^Week;
```

- A példában bemutatott definíciók és deklarációk érvényességi körén belül helyesek pl. a

```
Day1 :=nil;
Day2:=Day3
```

értékadások, de hibás a

```
Day2:=Day1
```

utasítás, mert a *Day2* és a *Day1* változó különböző típusú.

A mutató típusú változók felhasználhatók pl. az operatív tár dinamikus szervezésében (helyfoglalásban). Ehhez nyújtanak segítséget a tárfoglaló és -felszabadító eljárások.

*A New eljárás*

Hívása: *New (PtrVar)*

A *PtrVar* egy mutató típusú változó neve. A *New* eljárás végrehajtása következtében olyan típusú változó jön létre, mint amilyenre a *PtrVar* mutat. A művelet elvégzése után a *PtrVar* éppen a létrehozott változóra fog mutatni. Az új változó a *halom*-nak (heap) nevezett tárterületen foglal helyet, és ugyanúgy kezelhető, mint tetszőleges más — azonos típusú — változó.

**Példa**

```
type
  Matrix = array[boolean] of array[boolean] of real;
var
  MatrixPtr :^Matrix;
begin
  New(MatrixPtr);
  MatrixPtr^[true][true]:=23.5;
  ...
end.
```

- A *New* eljárás a halom területén létrehoz egy *Matrix* típusú változót; a *MatrixPtr* mutató típusú változó éppen erre az új változóra fog mutatni.
- Az értékadáskor a *MatrixPtr*^ tömb egyik sarokeleme a 23.5 értéket kapja.

• Jegyezzük meg, hogy a *MatrixPtr* mutató típusú változó (rövidebben: mutató) a *MatrixPtr*^ pedig azon objektum neve, amelyre a *MatrixPtr* mutat.

### *A Dispose eljárás*

Hívása: *Dispose (PtrVar)*

A *PtrVar* egy mutató típusú változó neve. A *Dispose* eljárás végrehajtása következtében megszűnik az a változó, amelyre a *PtrVar* mutat. A megszüntetendő változót előzőleg a *New* eljárással kell létrehozni. A változó által eddig lefoglalt tárterület visszakerül a halomba, és más változók létrehozására újra felhasználható.

**Példa**

```
var
  IntRef : ^integer;
begin
  New(IntRef);
  IntRef ^ := 20;
  Dispose(IntRef);
  New(IntRef);
  IntRef ^ := 30;
    Writeln(Intref^)
end.
```

• A program 30-at ír ki.

• Ha a programból töröljük a második értékadó utasítást, akkor logikailag hibás lesz, mert a *Writeln* eljárás végrehajtásakor az *IntRef* változónak határozatlan az értéke.

### *A Mark eljárás*

Hívása: *Mark (PtrVar)*

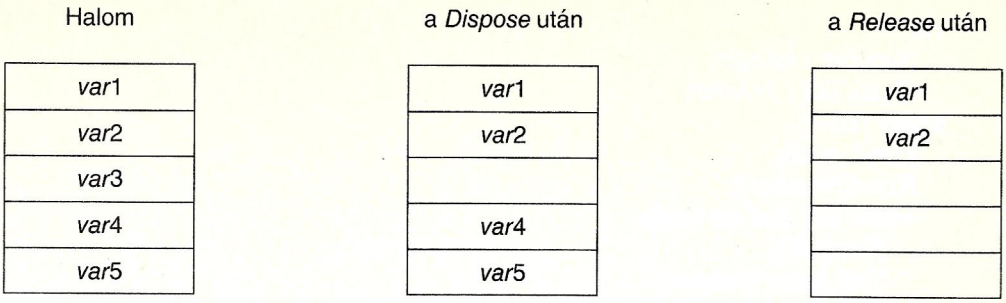
A *PtrVar* egy tetszőleges típusra mutató változó neve. A *Mark* eljárás végrehajtása következtében a *PtrVar* változó új értéke a halom pillanatnyi teteje lesz.

### *A Release eljárás*

Hívása: *Release (PtrVar)*

A *PtrVar* egy tetszőleges típusra mutató változó neve, amely a *Mark* eljárással kapott értéket. A *Release* eljárás végrehajtása következtében megszűnik az a változó, amelyre a *PtrVar* mutat, és a halmon utána következő összes változó is.

A *New-Dispose* és a *Mark-Release* eljárás két különböző halomszervezési módszert ad a programozónak; egy programban lehetőleg csak az egyiket használjuk! A két módszer közötti lényeges különbséget a *2. ábra* mutatja, ahol a *Ptr* változó a *var*3-ra mutat.




*2. ábra. A halomszervező módszerek összehasonlítása*

### *A GetMem eljárás*

Hívása: *GetMem (PtrVar,IntExp)*

*A PtrVar* egy mutató típusú változó neve, az *IntExp* pedig *integer* típusú kifejezés. A *GetMem* eljárás lefoglalja a halmon az *IntExp* értékkel meghatározott (bájtban értendő) tárterületet; a *PtrVar* mutató a lefoglalt területre fog mutatni.

### *A FreeMem eljárás*

Hívása: *FreeMem (PtrVar,IntExp)*

A *PtrVar* egy mutató típusú változó neve, az *IntExp* pedig *integer* típusú kifejezés. A *FreeMem* eljárás visszaadja a halomnak a *PtrVar* mutatóval megcímzett *IntExp* méretű (bájtban értendő) tárterületet. A *FreeMem* eljárással visszaadott tárterületnek pontosan akkorának kell lennie, mint a *GetMem* eljárással lefoglaltnak.

### *A MaxAvail függvény*

Hívása: *MaxAvail*

A *MaxAvail* függvény értéke egy *integer* típusú adat, amely megadja a halmon hozzáférhető legnagyobb összefüggő tárterület méretét (a 8 bites mikroszámítógépekét bájtban, az IBM PC méretét pedig paragrafusban, ahol 1 paragrafus = 16 bájt). Ha az eredmény negatív, akkor a valódi méretet úgy kapjuk meg, hogy az eredményhez a 65 536 valós számot adjuk hozzá.

**Példa**

```
type
  Pointer =^Pair;
  Pair = record
           Int : integer;
           Ref : Pointer;
         end;
```

```
var
  Number : integer;
  Head,Tail : Pointer;
begin {$B-}
  Head:=nil;
  Read(Number);
  while not Eof do begin
    New(tail);
    Tail^.Int:=Number;
    Tail^.Ref:=Head;
    Head:=Tail;
    Read(Number);
  end;
  Tail:=Head;
  while Tail <> nil do begin
    Writeln(Tail^.Int);
    Tail:=Tail^.Ref
  end
end.
```

• A program egész számok sorozatát olvassa be a konzolról, utána pedig fordított sorrendben kiírja azokat.

# 14. Kezdeti értékadás

A Turbo Pascal nyelv lehetővé teszi, hogy a változóknak kezdeti értéket adhassunk. Ha a program végrehajtásakor ezek a változók megőrzik kezdeti értéküket, akkor konstansoknak tekinthetők. Ha viszont a program futtatásakor más, a kezdeti értéküktől eltérő értéket is kapnak, akkor a program ismételt futtatásakor (ha az az operatív tárban marad) ezek a változók az előző futás befejezése utáni értékkel indulnak. Mivel egy ilyen program futtatásának ismételhetősége elvész, csak a konstans jellegű változók használata ajánlott. Éppen ezért azon változódeklarációk, amelyekben kezdeti értéket adunk, a programban a **const** kulcsszó (és nem a **var** kulcsszó) után szerepelnek.

**Példa**

```
program Increment;
const
  Number : integer = 1;
begin
  Writeln('Number = ',Number);
  Number:=Number + 1
end.
```

- A programban az *integer* típusú *Number* változó kezdeti értéke 1.
- A program ismételt futtatása (az R direktívával) következtében egymás utáni természetes számok íródnak ki.

A kezdeti értékadással bővített változódeklaráció tehát a következő: változónév, kettőspont, típusleírás, egyenlőségjel, kezdeti érték, pontosvessző. A kezdeti érték alakja a változó típusától függ. Elemi változókban konstans, tömbváltozókban kerek zárójelekkel körülvett kezdeti értékek listája, rekordváltozókban pedig — szintén kerek zárójelekkel körülvett — kezdeti értékek felsorolása.

*Szintaxis*

```
deklaráció_ kezdeti_ értékadással:
  változónév : típusleírás = kezdeti_ érték ;
változónév:
  azonosító
```

kezdeti_ érték:
konstans
konstansnév
(a_ kezdeti_ értékek_ listája)
(a_ kezdeti_ értékek_ felsorolása)

**Példák**

```
const
  LineLength : byte = 132;
  Radius : real = 13.64;
  FullName : string[12] = 'Jan Bielecki';
  CtrlM : char =^M;
  Separator : set of char = ['/',';',','.'];
  SexIsMale : boolean = true;
  ShortName : string[3] = 'Izabela';
  WholeReal : real = 44;
```

- A *LineLength* változó (*byte* típusú) kezdeti értéke 132.64.
- A *Radius* változó (*real* típusú) kezdeti értéke 13.64.
- A *FullName* változó (*string*[12] típusú) kezdeti értéke 'Jan Bielecki'.
- A *CtrlM* változó (*char* típusú) kezdeti értéke^M.
- A *Separator* változó (**set of** *char* típusú) kezdeti értéke ['/',';',','.'].
- A SexIsMale változó (*boolean* típusú) kezdeti értéke *true.*
- A *ShortName* változó (**string**[3] típusú) kezdeti értéke 'Iza'.
- A *WholeReal* változó (*real* típusú) kezdeti értéke 44.0.

Ha a kezdeti értékkel inicializált változó egy tömb, akkor kezdeti értéke kerek zárójelekkel körülvett lista, amely felsorolja a tömb összes elemének kezdeti értékét. Ha a kezdeti értékkel inicializált változó egy rekord, akkor kezdeti értéke kerek zárójelekkel körülvett lista, amely felsorolja a rekord összes (vagy néhány) elemének kezdeti értékét. A kezdeti értékadások felsorolási sorrendjének azonosnak kell lennie a rekordkomponensek deklarációbeli sorrendjével. A lista elemeit pontosvesszők választják el; egy listaelem a rekordmező nevéből, a kettőspontból és a rekordmező kezdeti értékéből áll.

**Példák**

```
type
  Color = (Red,Black,Fair);
  Person = record
             FirstName : string[3];
             LastName : string[12];
             Hair : Color;
             Age : byte
           end;
```

```
  Sex = (male,female);
const
  Vector : array[1..3] of Sex =
           (male,male,female);
  Matrix : array[Color,2..3] of byte =
           ((0,0),(1,1),(2,5));
  ThreeD : array[boolean,boolean,boolean] of Color =
           (((Red,Red),(Red,Fair)),
            ((Black,Black),(Red,Black)));
  JanB : Person =
         (FirstName : 'Jan';
          LastName : 'Bielecki';
          Hair : Black;
          Age : 44);
  Family : array[1..3] of Person =
           (FirstName : 'Jan';
            LastName : 'Bielecki';
            Hair : Black;
            Age : 44),
           (Firstname : 'Ewa';
            LastName : 'Bielecka';
            Hair : Black;
            Age : 38),
           (FirstName : 'Iza';
            LastName : 'Bielecka';
            Hair : Black;
            Age : 3));
```

- A *Vector*[2] kezdeti értéke *male.*
- A *Matrix*[*Fair*,3] kezdeti értéke 5.
- A *ThreeD*[*false,true,true*] kezdeti értéke *Fair.*
- A *JanB.Age* komponens kezdeti értéke 44.
- A *Family*[2].*Age* komponens kezdeti értéke 38.
- A *Vector* változó

*Vector:* **array**[1..3] **of** *Sex = (male,female);* alakú deklarációja hibás lenne, mert e változó egy háromelemű tömb, a kezdeti értékek listája pedig csak 2 elemből áll.

- A *JanB* változó

*JanB : Person* = (*FirstName :* 'Janek');
alakú deklarációja hibátlan lenne és egyenértékű a

*JanB : Person* = (*FirstName* : 'Jan');
deklarációjával.

- A *JanB* változó

*JanB : Person* = (*LastName* : 'Bielecki');
alakú deklarációja hibás lenne.

A változó rekordrészt tartalmazó rekordokban a kezdeti értékadásnak csak akkor van értelme, ha azokra a változó rekordrészmezőkre vonatkoznak, amelyek az adott változathoz tartoznak. Ezt a követelményt a fordítóprogram nem ellenőrzi.

**Példa**

```
type
  Sex = (male,female);
  Color = (Black,Red,Blond);
  Person = record
             Name : string[20];
             case Gender : Sex of
               male : (Height : byte);
               female : (Hair : Color)
           end;
const
  Trio : array[1..3] of Person =
         ((Name : 'Ewa'),
          (Name : 'Jan'; Gender : male),
          (Name : 'Iza'; Gender : female;
           Hair : Black));
```

- A *Trio*[2].*Gender* rekordtömb-komponens kezdeti értéke *male.*
- A következő kezdeti értékadás

(*Name* : 'Iza'; *Gender* : *male; Hair* : *Black*)

értelmetlen ugyan, mégis helyes.

# 15. Függvények és eljárások

A függvények és az eljárások olyan objektumok, amelyek a programmal megvalósított algoritmus jól definiált részeit írják le. Emiatt a továbbiakban alprogramoknak nevezzük őket.

Egyéb strukturált utasításoktól (pl. a kiválasztó utasítástól) eltérően az alprogram végrehajtása igényli az alprogram hívását, azaz — a helyzetnek megfelelően — az eljáráshívó utasításnak vagy a függvény hívásának az alkalmazását. Az eljáráshívó utasítás a program mindazon helyein szerepelhet, ahol tetszőleges utasítás állhat; ezzel szemben a függvény hívása csak egy kifejezésben megengedett.

A változókhoz hasonlóan az alprogramokat is deklarálni kell. Az alprogramok deklarációi a blokk deklarációs részében vannak. Az alprogram deklarációja a fejlécből és az alprogram törzsét jelentő blokkból áll. Az eljárás fejléce a következő: a **procedure** kulcsszó, az eljárásparaméterek felsorolása (kerek zárójelekkel körülvéve) és a pontosvessző. A függvény fejléce a következő: a **function** kulcsszó, a függvényparaméterek felsorolása (kerek zárójelekkel körülvéve), a kettőspont, a függvény eredményének típusmeghatározása és a pontosvessző. Ha az alprogramnak nincs paramétere, akkor a paraméterfelsorolás a körülvevő zárójelekkel együtt elhagyható.

A függvény végrehajtásakor olyan értékadó utasításnak is lennie kell, amelyben az értékadó szimbólum bal oldalán a függvény azonosítója, jobb oldalán pedig a függvény eredményének típusával azonos típusú kifejezés áll. Ebben az értékadásban a kifejezés értéke a függvény eredménye. A függvény végrehajtása nem korlátozódik az eredmény átadására; a hívás argumentumainak megváltoztatását is jelentheti.

*Szintaxis*

alprogram-deklaráció:
  eljárásdeklaráció
  függvénydeklaráció
eljárásdeklaráció:
  **procedure** eljárásnév (paraméterfelsorolás);
  **procedure** eljárásnév ;
függvénydeklaráció:
  **function** függvénynév (paraméterfelsorolás)
                : eredménytípus ;
  **function** függvénynév : eredménytípus ;
eljárásnév:
  azonosító

függvénynév:
azonosító
eredménytípus:
elemitípus-azonosító
elemitípus-azonosító:
azonosító

Az alprogram paramétereinek felsorolása pontosvesszővel elválasztott elemekből áll. A felsorolás mindegyik eleme a paraméterazonosítók listája, amelyet kettőspont és a listában szereplő paraméterek típusazonosítója követ.

Az alprogram hívásának pillanatában megtörténik az alprogram paramétereinek és a hívás argumentumainak egymáshoz rendelése. Az argumentumok számának egyenlőnek kell lennie a paraméterek számával; a paraméterek és az argumentumok egymáshoz rendelése az alprogram fejlécében és hívásában meghatározott sorrendű.

A paramétert és az argumentumot kétféleképpen lehet egymáshoz rendelni: érték vagy név szerint. Az érték szerinti hivatkozásban a paraméter az alprogram egyik belső változójának tekinthető, amely az alprogram végrehajtásának pillanatában (az adott hívásra értelmezve) megkapja az argumentum híváskori értékét. A név szerinti hivatkozásban a paraméter magát az argumentumot képviseli. Ennek az az eredménye, hogy a paraméteren értelmezett műveletek úgy hajtódnak végre, mintha az argumentumra vonatkoznának. A név szerinti hivatkozást a paraméterazonosítók listája előtt álló **var** kulcsszóval jelölhetjük ki.

A név szerinti hivatkozás különleges esetében az argumentum egy alprogram neve. A Pascal nyelv szabványával ellentétben a Turbo Pascal nyelvből hiányzik az ilyen hivatkozás lehetősége.

*Szintaxis*

paraméterfelsorolás_ eleme:
paraméternév_ listája : típusmeghatározás
**var** paraméternév_ listája : típusmeghatározás
**var** paraméternév_ listája
típusmeghatározás:
típusazonosító
típusazonosító:
azonosító
paraméternév:
azonosító

**Példák**

*1.* Egyparaméteres eljárás, érték szerinti hivatkozással:

```
program FiveMessages;
var
  Tally : byte;
procedure SlowDown(Count : integer);
```

```
begin
  for Count:=Count downto 1 do
end;
begin
  for Tally:=1 to 5 do begin
    Writeln('Wake up!');
    SlowDown(400)
  end
end.
```

- A *FiveMessages* program két deklarációt tartalmaz: a *Tally* változóét és a *SlowDown* eljárását.
- A paraméterfelsorolás eleme (és egyben maga a felsorolás is) a

*Count : integer*

- A *Count* paraméter és a 400 argumentum egymáshoz rendelése érték szerinti.
- A *Count* paraméteren végrehajtott műveletek valójában a *SlowDown* eljárás lokális változójára vonatkoznak; e változó kezdeti értéke 400.

2. Kétparaméteres eljárás, név szerinti hivatkozással:

```
program ConvertAndSwap;
var
  theFloat : real;
  theFixed : integer;
procedure Swap(var Float : real;
               var Fixed : integer);
var
  Temp : real;
begin
  Temp:=Float;
  Float:=Fixed;
  Fixed:=trunc(Temp + 0.5)
end;
begin
  theFloat:=12.8;
  theFixed:=10;
  Writeln(theFloat,theFixed:3);
  Swap(theFloat,theFixed);
  Writeln(theFloat,theFixed:3)
end.
```

- A *ConvertAndSwap* program négy deklarációt tartalmaz: a *theFloat* és a *theFixed* változókét, a *Swap* eljárását és a *Temp* változóét.
- A *Swap* eljárás paraméterfelsorolása két elemből áll, melyeket pontosvessző választ el egymástól.
- A paraméterek és az argumentumok egymáshoz rendelése név szerinti.

- A *Float* paraméteren végrehajtott műveletek valójában a *theFloat* argumentumra, a *Fixed* paraméteren végrehajtottak pedig a *theFixed* argumentumra vonatkoznak.
- A program két sort ír ki; az elsőben 12.8 és 10, a másodikban 10 és 13 jelenik meg.
- Ha a *Swap* eljárás fejléce

```
procedure Swap(Float : real;
                    var Fixed : integer);
```

alakú lenne, akkor eredményként a 12.8 és 10, valamint a 12.8 és 13 számok jelennének meg. Ennek az az oka, hogy a megváltozott fejlécben a *Float* paraméter és a *theFloat* argumentum egymáshoz rendelése érték szerinti.

*3.* A legrövidebb, egy paraméter nélküli eljárás deklarációját és hívását tartalmazó program:

```
procedure P;
begin
end;
begin
  P
end.
```

*4.* Háromparaméteres függvény:

```
program MultiplyDivide;
var
  Product : integer;
function Divide(SourceOne,SourceTwo : byte;
                    var Target : integer) : real;
begin
  Target:=SourceOne * SourceTwo;
  Divide:=SourceOne / SourceTwo
end;
begin
  Writeln(Divide(12,4,Product),Product:3)
end.
```

- A *SourceOne* és a *SourceTwo* paraméterérték szerint van hozzárendelve a 12 és a 4 argumentumhoz; a *Target* paraméter pedig név szerint a *Product* argumentumhoz; a függvény értéke ekkor *real* típusú adat.
- A program egy sort ír ki, amelyben 3.0 és 48 jelenik meg.
- Ha a *Divide* függvény fejléce

```
function Divide(SourceOne,SourceTwo : byte;
                    Target : integer) : real;
```

alakú, akkor a program hibás, mert a *Product* változó nem kap semmilyen értéket.

- Ha a *Divide* függvény fejléce

```
function Divide (var SourceOne,SourceTwo : byte;
                 var Target : integer) : real;
```

alakú, akkor a program szintén hibás, mert a név szerinti hivatkozásban olyan argumentumok szerepelnek, amelyek nem változónevek, hanem kifejezések (itt 12 és 14).

*5.* A legrövidebb, egy paraméter nélküli függvény deklarációját és hívását tartalmazó program:

```
function F : byte;
begin
  F:=0
end;
begin
  Write(F)
end.
```

*6.* Az alprogramokat képviselő paramétereket tartalmazó alprogram:

```
program VeryComplicated;
procedure Ext(procedure Par; Flag : boolean);
var
  Fix : byte;
procedure Int;
begin
  Fix:=Fix+1;
end;
begin {Ext}
  Fix:=0;
  if Flag then Ext(Int,false)
          else Par;
  Writeln(Fix)
end;
begin {VeryComplicated}
  Ext(Ext,true)
end.
```

- A program olyan szabványos nyelvi szerkezetet tartalmaz, amelyet nem valósítottak meg a Turbo Pascal nyelvben, ezért hibás.

A fordítóprogram ellenőrzi a paramétertípus és az argumentumtípus azonosságát, kivéve az ún. típus nélküli paramétert, amelyről a fordítóprogram feltételezi, hogy típusa azonos a hozzárendelt (tetszőleges) argumentum típusával. A nyelv e bővítése a 19.1. szakaszban leírt **absolute** kulcsszó használatával együtt lehetővé teszi a korábbinál jóval hatékonyabb programok írását.

**Példa**

```
program MoveArray;
var
  SourceArray : array[1..3,1..4] of integer;
  TargetArray : array[1..2,1..6] of integer;
...
procedure MoveBytes(var Source,Target; Count: integer);
var Index : integer;
type
  ByteField = array[1..MaxInt] of byte;
var
  Src : ByteField absolute Source;
  Trg : ByteField absolute Target;
begin
  for Index:=1 to Count do
    Trg[Index]:=Src[Index]
end;
...
begin
  ...
  MoveBytes(SourceArray,TargetArray,24);
  ...
end.
```

- A *SourceArray* és a *TargetArray* tömb különböző típusú, ezért nem megengedett a

*TargetArray* := *SourceArray*

utasítás használata. A célt mégis elérhetjük a *MoveBytes* eljárás használatával.

- Az eljárásban a *Source* és a *Target* típus nélküli paraméter. Az *Src* és a *Trg ByteField* típusú lokális változó, és ugyanazon a tárterületen helyezkednek el, mint a hozzájuk rendelt argumentumtömbök.

Az a program, amelyben meghatározott típusú paraméterhez eltérő típusú argumentumot rendelünk, nyilvánvalóan hibás. Kivétel a *real* típusú paraméter, mert ehhez *integer* típusú argumentum is hozzárendelhető.

Kizárólag a karakterlánc típusú paraméterek és argumentumok esetén mellőzhető a típusazonosság követelménye. A {$V—} direktíva hatáskörén belül minden karakterlánc típusú paraméter azonos a hozzárendelt karakterlánc típusú argumentummal. Ezáltal tetszőleges típusú karakterláncok feldolgozására nyílik lehetőség.

**Példa**

```
program CountSpaces;
type
  OneLine = string[80];
```

```
function SpaceCount(SourceText : OneLine ) : integer;
    var
  Count, Index : integer;
begin
  Count:=0;
  for Index:=1 to Length(SourceText) do
    if SourceText[Index] = '' then
      Count:=Count + 1;
  SpaceCount:=Count
end;
begin
  ...
  Writeln(SpaceCount(
          {$V—1} 'To be or not to be' {$V+}
                          ));
  ....
end.
```

- A *SourceText* paraméter **string**[80] típusú, a hozzárendelt argumentum típusa viszont **string**[18].
- A *SpaceCount* függvény hívása helyes, mert a paraméter és az argumentum egymáshoz rendelése a {$V—} direktíva hatáskörén belüli.
- A *Writeln* utasítás az 5-ös számot írja ki.

A többi Algol-szerű nyelvhez hasonlóan a Turbo Pascal nyelvben megírt alprogramok rekurziós változatban is megfogalmazhatók, ez sokszor egyszerűsíti az algoritmust.

**Példa**

```
function Fibonacci(Index : byte) : integer;
begin
  if Index < 3 then
    Fibonacci:=1
  else
    Fibonacci:=Fibonacci(Index — 1) +
               Fibonacci(Index — 2)
end
```

- A függvény kiszámítja a Fibonacci-sor elemeit. E sorban az első két elem értéke 1, az összes utána következő elemé pedig egyenlő az őt közvetlenül megelőző két elem összegével.

Ha a rekurziós algoritmus két olyan eljárást használ, amelyben az első hívja a másodikat és a második is hívja az elsőt, lehetetlen úgy átrendezni a deklarációkat, hogy minden hívás a már deklarált objektumra vonatkozzék. Ekkor szükség van az ún. előzetes deklarációra, amely az alprogram fejlécéből, a pontosvesszőből és a **forward** kulcsszóból áll.

Az előzetes deklaráció után a megfelelő helyen szerepelnie kell az alprogram tulajdonképpeni deklarációjának. Ennek fejléce már nem tartalmazza a zárójelekkel körülvett paraméterfelsorolást, függvény esetén pedig a kettőspontot és a típusmeghatározást sem.

**Példa**

```
function FunOne(var Source : byte;
                    Target : byte) : char; forward;
function FunTwo(Count : byte) : real;
begin
  ...
  Write(FunOne(Count,2*Count));
  ...
end;
function FunOne;
begin
  ...
  Write(FunTwo(Source + Target));
  ...
end;
```

- Mivel a *FunTwo* függvény hivatkozik a *FunOne* függvényre, és a *FunOne* függvény is hívja a *FunTwo* függvényt, szükség volt olyan előzetes deklarációra, amelyben a **forward** kulcsszó helyettesíti a *FunOne* függvény törzsét.
- A példában bemutatott függvények deklarációi természetesen fordított sorrendben is szerepelhetnek. Ekkor az előzetes deklaráció a *FunTwo* függvényre vonatkozik és

```
function FunTwo(Count : byte) : real;
forward;
```

alakú.

# 16. Standard alprogramok

A *standard alprogram* megnevezés azokra a beépített alprogramokra vonatkozik, amelyek nyílt deklarálásuk nélkül használhatók. Számos ilyen alprogramot leírtunk már az előzőekben; ide tartoznak azok az alprogramok, amelyek karakterláncokon végeznek műveleteket (pl. a dinamikus társszervező alprogramok, valamint a beviteli, ill. kiviteli műveleteket megvalósító alprogramok).

E terjedelmes csoport kiegészítői a képernyős és a különleges eljárások, az aritmetikai, a skalár és a konverziós függvények, valamint más segédfüggvények.

## 16.1. Képernyős eljárások

A képernyős eljárások azokban az implementációkban használhatók, amelyekben installálták őket.

### *A ClrEol eljárás*

Hívása: *ClrEol*

A *ClrEol* eljárásnak nincs paramétere. Az eljárás szóközökkel felülírja a kurzor alatti és a tőle jobbra álló karaktereket a sor végéig. A kurzor pozíciója nem változik.

### *A ClrScr eljárás*

Hívása: *ClrScr*

A *ClrScr* eljárásnak nincs paramétere. Az eljárás „törli" a képernyőt, azaz szóközökkel tölti fel. A kurzor a képernyő bal felső sarkába kerül. Az installációtól függ, hogy a — később leírt — *LowVideo* és *NormVideo* eljárással beállított jellemzők változnak-e.

### *A CrtInit eljárás*

Hívása: *CrtInit*

A *CrtInit* eljárásnak nincs paramétere. Az eljárás a Turbo Pascal rendszer installálásakor meghatározott inicializáló karaktersorozatot küld a képernyőre.

### A CrtExit eljárás

Hívása: *CrtExit*

A *CrtExit* eljárásnak nincs paramétere. Az eljárás a Turbo Pascal rendszer installálásakor meghatározott lezáró karaktersorozatot küld a képernyőre.

### A DelLine eljárás

Hívása: *DelLine*

A *DelLine* eljárásnak nincs paramétere. Az eljárás törli azt a sort, ahol a kurzor áll. Az alatta levő sorok eggyel feljebb kerülnek, így a képernyő legalsó sora üres lesz. A kurzor pozíciója nem változik.

### Az InsLine eljárás

Hívása: *InsLine*

Az *InsLine* eljárásnak nincs paramétere. Az eljárás hatására az a sor, ahol a kurzor áll (és az alatta levő sorok) eggyel lejjebb kerül. Így a képernyőn üres lesz a sor, ahol a kurzor áll, a legalsó sor pedig törlődik. A kurzor pozíciója nem változik.

### A GotoXY eljárás

Hívása: *GotoXY (xPos,yPos)*

Az *xPos* és az *yPos integer* típusú kifejezés. Az eljárás az *(xPos,yPos)* koordinátájú karakterpozícióba állítja a kurzort. A képernyő bal felső sarkának koordinátája (1,1); a vízszintes pozíciók balról jobbra, a függőlegesek pedig felülről lefelé növekednek. Ha az *xPos* vagy az *yPos* kifejezés értéke nagyobb a sor- vagy oszlopméretnél, akkor a rendszer azt a moduloméret szerint veszi figyelembe.

### A LowVideo eljárás

Hívása: *LowVideo*

A *LowVideo* eljárásnak nincs paramétere. Az eljárás hatására a legközelebbi *NormVideo* eljárás hívásáig a képernyőre kivitt karakterek inverzben jelennek meg.

### A NormVideo eljárás

Hívása: *NormVideo*

A *NormVideo* eljárásnak nincs paramétere. Az eljárás a szokásosra (vö. a *LowVideo* eljárással) állítja vissza a karaktermegjelenítését.

## 16.2. Különleges eljárások

### *A Delay eljárás*

Hívása: *Delay (Time)*

A *Time integer* típusú kifejezés. Az eljárás a program végrehajtását kb. *Time* ms időtartamra felfüggeszti. Ha a *Time* kifejezés értéke negatív, akkor 0-nak tekinti a rendszer.

### *A FillChar eljárás*

Hívása: *FillChar (Var,Cnt,Val)*

A *Var* egy tetszőleges típusú változó neve, a *Cnt integer* típusú kifejezés, a *Val* pedig *char* vagy *byte* típusú kifejezés. Az eljárás a *Var* változó által lefoglalt tárterület elejétől *Cnt* számú bájtot feltölt a *Val* által képviselt bájtos adat értékével.

### *Az Exit eljárás*

Hívása: *Exit*

*Az Exit* eljárásnak nincs paramétere. Az eljárás hatására a vezérlés a blokk — amelyben a hívás szerepel — **end** kulcsszava elé adódik át. Ezzel befejeződik a blokk végrehajtása, ha pedig a programblokkban szerepelt az *Exit* hívása, akkor ez a program végét jelenti.

### *A Halt eljárás*

Hívása: *Halt*

A *Halt* eljárásnak nincs paramétere. Az eljárás hatására befejeződik a program futása.

### *A Move eljárás*

Hívása: *Move (Src,Trg,Cnt)*

Az *Src* és a *Trg* tetszőleges típusú változók neve, a *Cnt* pedig *integer* típusú kifejezés. Az eljárás az *Src* változó által lefoglalt terület elejéről *Cnt* számú bájtot másol át a *Trg* változó által lefoglalt terület elejére. Az eljárás akkor is helyesen másol, ha e területek részben vagy egészben fedik egymást.

### *A Randomize eljárás*

Hívása: *Randomize*

A *Randomize* eljárásnak nincs paramétere. Az eljárás véletlen értékű adattal inicializálja az álvéletlenszám-generátort.

## 16.3. Aritmetikai függvények

### *Az Abs függvény*

Hívása: *Abs (Num)*

A *Num real* vagy *integer* típusú kifejezés. Az *Abs* függvény értéke a *Num* képviselte adat abszolút értéke, típusa azonos az argumentum típusával.

Abs (—2) = 2
Abs(Pi) = 3.1415926536

### *Az ArcTan függvény*

Hívása: *ArcTan (Num)*

A *Num real* vagy *integer* típusú kifejezés. Az *ArcTan* függvény értéke a *Num* képviselte adat arkusz tangense, típusa *real.*

ArcTan(0) = 0.0.
ArcTan(1,0) = 0,7853981634

### *A Cos függvény*

Hívása: *Cos (Num)*

A *Num real* vagy *integer* típusú kifejezés. A *Cos* függvény értéke a *Num* képviselte adat koszinusza, típusa *real.*

Cos(0) = 1.0
Cos(Pi) = —1.0

### *Az Exp függvény*

Hívása: *Exp (Num)*

A *Num real* vagy *integer* típusú kifejezés. Az *Exp* függvény típusa *real,* értéke az $e^{Num}$ hatványozás eredménye, ahol *e* a természetes logaritmus alapja: *e* = 2.7182818285.

Exp(0) = 1.0
Exp(—1.0) = 0.36787944117

### *A Frac függvény*

Hívása: *Frac (Num)*

A *Num real* vagy *integer* típusú kifejezés. A *Frac* függvény értéke a *Num* képviselte adat törtrésze, típusa *real.*

Frac(3) = 0.0.
Frac(pi) = 0.1415926536

### *Az Int függvény*

Hívása: *Int (Num)*

A *Num real* vagy *integer* típusú kifejezés. Az *Int* függvény értéke a *Num* képviselte adat egészrésze, típusa *real.*

Int(2) = 2.0
Int(—Pi) = —3.0

### *Az Ln függvény*

Hívása: *Ln (Num)*

A *Num real* vagy *integer* típusú kifejezés. Az *Ln* függvény értéke a *Num* képviselte adat természetes alapú logaritmusa, típusa *real.*

Ln(1) = 0.0
Ln(3.0) = 1.0986122887

### *A Sin függvény*

Hívása: *Sin (Num)*

A *Num real* vagy *integer* típusú kifejezés. A *Sin* függvény értéke a *Num* képviselte adat szinusza, típusa *real.*

Sin(0) = 0.0
Sin(Pi/2) = 1.0

### *Az Sqr függvény*

Hívása: *Sqr (Num)*

A *Num real* vagy *integer* típusú kifejezés. Az *Sqr* függvény értéke a *Num* képviselte adat négyzete, típusa azonos az argumentum típusával.

Sqr(2) = 4
Sqr(2.0) = 4.0

### *Az Sqrt függvény*

Hívása: *Sqrt (Num)*

A *Num real* vagy *integer* típusú kifejezés. Az *Sqrt* függvény értéke a *Num* képviselte adat négyzetgyöke, típusa *real.*

Sqrt(4) = 2.0
Sqrt(4.0) = 2.0

## 16.4. Skalárfüggvények

*Az Odd függvény*

Hívása: *Odd (Num)*

A *Num integer* típusú kifejezés. Az *Odd* függvény „a *Num* kifejezés értéke páratlan szám" mondat logikai értékét adja meg, típusa *boolean.*

Odd(3) = true
Odd(4) = false

*A Pred függvény*

Hívása: *Pred (Num)*

A *Num* tetszőleges, rendezett típusú kifejezés. A *Pred* függvény értéke a rendezett típus azon eleme, amely közvetlenül az argumentum képviselte elem előtt áll, típusa azonos az argumentum típusával.

Pred('B') = 'A'
Pred(—20) = —21

*A Succ függvény*

Hívása: *Succ (Num)*

A *Num* tetszőleges, rendezett típusú kifejezés. A *Succ* függvény értéke a rendezett típus azon eleme, amely közvetlenül az argumentum képviselte elem után áll, típusa azonos az argumentum típusával.

Succ(false) = true
Succ(0) = 1

## 16.5. Konverziós függvények

*A Chr függvény*

Hívása: *Chr (Num)*

A *Num integer* típusú kifejezés. A *Chr* függvény típusa *char,* értéke a *Num* kódú karakter.

Chr(65) = 'A'
Chr(27) = ^[

*Az Ord függvény*

Hívása: *Ord (Num)*

A *Num* tetszőleges, rendezett típusú kifejezés. Az *Ord* függvény típusa *integer,* értéke megadja a *Num* képviselte adat sorszámát a rendezett típuson belül.

Ord(true) = 1
Ord(—2) = —2

*A Round függvény*

Hívása: *Round (Num)*

A *Num real* típusú kifejezés. A *Round* függvény típusa *integer,* értéke a *Num* képviselte adathoz legközelebb álló szám.

Round(5.5) = 6
Round(—1.5) = —2

*A Trunc függvény*

Hívása: *Trunc (Num)*

A *Num* real típusú kifejezés. A *Trunc* függvény típusa *integer,* értéke a *Num* képviselte adat egészrésze.

Trunc(3.14) = 3
Trunc(—2.9) = —2

## 16.6. Segédfüggvények

*A Hi függvény*

Hívása: *Hi (Num)*

A *Num integer* típusú kifejezés. A *Hi* függvény típusa *integer,* értéke a *Num* képviselte adat magasabb helyi értékű bájtja.

Hi(256) = 1
Hi(—1) = 255

*A KeyPressed függvény*

Hívása: *KeyPressed*

A *KeyPressed* függvénynek nincs paramétere. A függvény *boolean* típusú, értéke „a konzolpufferban van még be nem olvasott karakter" mondat logikai értékét adja meg.

### *A Lo függvény*

Hívása: *Lo (Num)*

A *Num integer* típusú kifejezés. A *Lo* függvény típusa *integer,* értéke a *Num* képviselte adat alacsonyabb helyi értékű bájtja.

```
Lo(256) = 0
Lo(—1) = 255
```

### *A Random függvény*

Hívása: *Random*
*Random (Num)*

*A Num integer* típusú kifejezés. A *Random* függvény értéke paraméter nélküli változatban *real* típusú véletlen szám, amelynek értéke 0.0 vagy annál nagyobb, de 1.0-nél kisebb. Ha a *Random* függvénynek van paramétere, akkor az eredmény *integer* típusú véletlen szám, értéke 0.0 vagy annál nagyobb, de *Num*-nál kisebb.

### *A ParamCount függvény*

Hívása: *ParamCount*

A *ParamCount* függvénynek nincs paramétere. A függvény *integer* típusú, értéke megadja a program hívásakor átadott paraméterek számát. Feltételezzük, hogy a paramétereket szóközök vagy tabulátorok választják el egymástól.

### *A ParamStr függvény*

Hívása: *ParamStr (Num)*

A *Num integer* típusú kifejezés. A *ParamStr* függvény karakterlánc típusú adat, amely a program hívásakor átadott *Num* sorszámú paramétert jelenti. Feltételezzük, hogy az első paraméter sorszáma 1.

### *A SizeOf függvény*

Hívása: *SizeOf (Nam)*

A *Nam* változónév vagy típusnév. A *SizeOf* függvény *integer* típusú, értéke megadja a *Nam* változónak kiosztott vagy a *Nam* típusú adat ábrázolásához szükséges bájtok számát.

```
SizeOf(Boolean) = 1
SizeOf(integer) = 2
```

### *A Swap függvény*

Hívása: *Swap (Num)*

A *Num integer* típusú kifejezés. A *Swap* függvény *integer* típusú, értékének magasabb helyi értékű bájtja *Lo(Num),* alacsonyabb helyi értékű bájtja pedig *Hi(Num).*

Swap(256) = 1
Swap(1) = 256

### *Az UpCase függvény*

Hívása: *UpCase (Chr)*

A *Chr char* típusú kifejezés. Az *UpCase* függvény *char* típusú, értéke — ha a *Chr* betű — nagybetű, különben *Char.*

UpCase('a') = 'A'
UpCase('.') = '.'

**Példa**

```
program PrintArguments;
var
  Count : byte;
begin
  Write('Arguments are: ');
  for Count:=1 to ParamCount do
  Write(' ',ParamStr(Count))
end.
```

- Ha a *PrintArguments* programot a

*Jan Ewa Iza*

argumentumokkal hívjuk meg, akkor futásának eredményeként megjelenik az

Arguments are: Jan Ewa Iza

felirat.

- Ha argumentum nélkül hívjuk meg, akkor az eredmény az

Arguments are:

felirat.

# 17. Állományok beillesztése

A fordítóprogram fontosabb direktívái közé tartozik az állományt beillesztő direktíva (include). Alakja a következő:

{$I *Name*}

ahol *Name* az állomány neve. E direktívát új sorba kell írni. Végrehajtásakor a fordítóprogram helyettesíti a *Name* nevű állomány tartalmával. Beilleszthetjük vele a programba az előzőleg megírt deklarációs, ill. alprogramkönyvtárakat, forrásnyelvi programrészeket, a beillesztett állományban azonban nem szerepelhet egy újabb beillesztési direktíva.

Figyelembe kell még venni, hogy ha a *Name* állománynév nem tartalmaz hárombetűs kiterjesztést, akkor a direktívát lezáró kapcsos zárójel és a *Name* között legalább egy szóköz álljon, mert különben a fordítóprogram az állománynév részének tekinti majd a zárójelet.

**Példa**

Legyen a FACTOR.DOC állomány tartalma a következő:

```
function Factorial(Num : byte ) : integer;
begin
  if Num < 2 then Factorial:=1
             else Factorial:=Num *Factorial(Num - 1)
end;
```

A program:

```
program Demo;
{$I Factor.doc}
begin
  Writeln(Factorial(5))
end.
```

E program fordítása ugyanolyan eredménnyel jár, mintha a következő programot fordítottuk volna le:

```
program Demo;
function Factorial(Num : byte ) : integer;
begin
  if Num < 2 then Factorial:=1
             else Factorial:=Num*Factorial(Num - 1)
end;
begin
  Writeln(Factorial(5))
end.
```

# 18. Alprogramok átlapolása

A Turbo Pascal rendszer lényeges korlátozása, hogy a tárgyprogram és adatai nem lehetnek nagyobbak 64 kbájtnál. E korlátozás megkerülhető az átlapolt alprogramok (overlay módszer) alkalmazásával.

Eljárások és függvények is átlapolhatók; az átlapolt alprogramok azonban nem hívhatók rekurzív módon, és nem szerepelhetnek előzetes deklarációkban sem. Ez nem jelent akadályt, hiszen deklarálható egy nem átlapolt alprogram is, amely a másik — eredetileg rekurzív módon hívandó — alprogramot hívja.

Hibakereséskor viszont számíthatunk bizonyos nehézségekre, ezért csak a kipróbált, „belőtt" alprogramokat alakítsuk át átlapolttá.

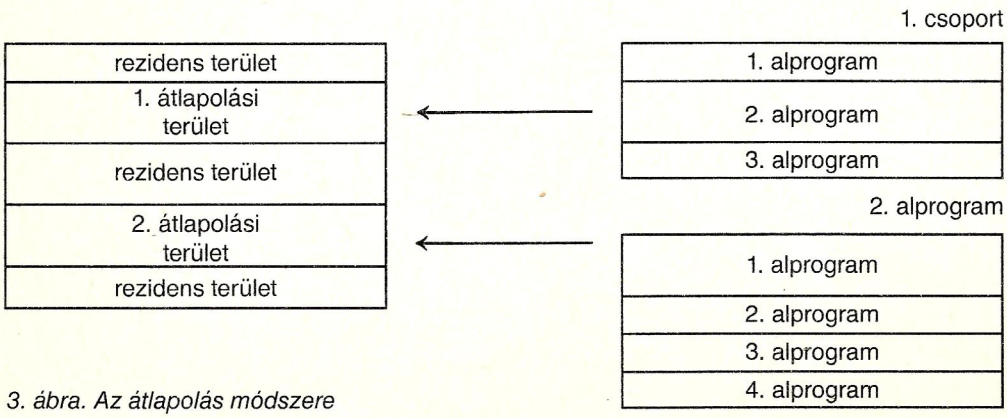
Az átlapolás módszerét a *3. ábra* mutatja. A tárgyprogram számára kiosztott tárterület két részre bontható: tárrezidens részre (amelyet a program állandó része foglal le) és átlapolási (overlay) területre (ahova az egyes átlapolt programrészek — alprogramcsoportok töltődnek be).

Minden átlapolt alprogramcsoport több alprogramból áll; a csoporthoz egy átlapolási terület tartozik. A program végrehajtásakor az átlapolási területen a csoport egyetlen tagja tartózkodhat. E terület mérete akkora, hogy a csoport minden alprogramja belefér. Ez azt is jelenti, hogy egy átlapolt csoport használatakor a megtakarított tárterület a csoportméret és a csoporton belüli legnagyobb alprogram méretének különbségével egyenlő.

Az operatív tár megtakarításának ára:

*1.* A csoport egyes alprogramjai nem hívhatják egymást;

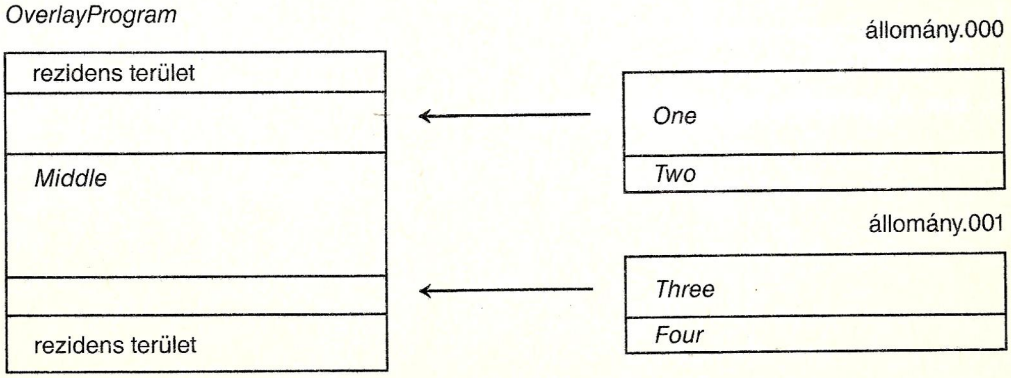
*2.* Mágneslemezes átvitel (alporgramok betöltése) szükséges azokban az átlapolt alprogramokban, amelyek hívásukkor nincsenek a tárban.




*3. ábra. Az átlapolás módszere*

Az átlapolt alprogramcsoportok létrehozási szabálya igen egyszerű: egy vagy több egymás utáni alprogram-deklaráció elé az **overlay** kulcsszót kell írni. Az így kijelölt alprogramok csoportot alkotnak, melyet a rendszer mágneslemezes állományban tárol. Az állomány neve a főprogram nevével azonos (l. a rendszermenü M parancsát), kiterjesztése háromjegyű. Az első csoport kiterjesztése .000, a másodiké .001 stb. Ha az átlapolt alprogramok között rezidens alprogramok is szerepelnek, akkor az átlapoltak különálló csoportokba kerülnek. Ezt mutatja a *4. ábra*, amelyen az *Overlay Program* lefordításából keletkezett tárgykód szerkezete látható.

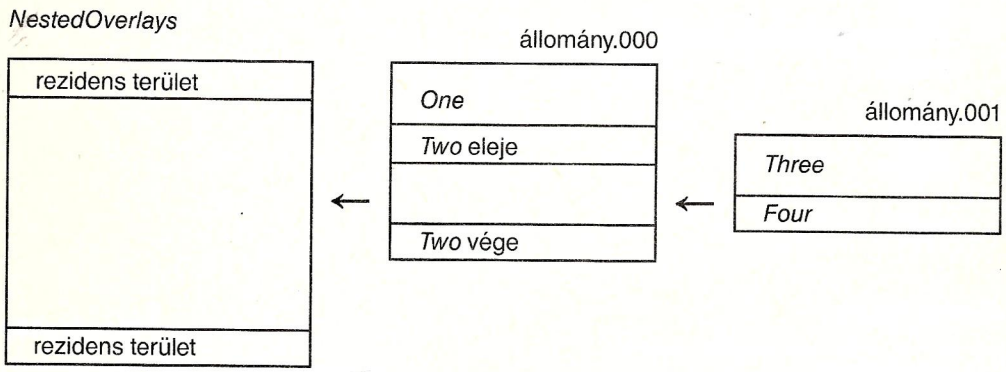
```
program overlayProgram;
overlay procedure One;
begin
  ...
end;
overlay procedure Two;
begin
  ...
end;
procedure Middle;
begin
  ...
end;
overlay procedure Three;
begin
  ...
end;
overlay procedure Four;
begin
  ...
end;
begin
  ...
end.
```




*4. ábra. Az OverlayProgram szerkezete*

Az átlapolt csoportok létrehozási szabálya nem zár ki újbóli átlapolást (az átlapolt alprogramon belül). Az *5. ábrán* a *NestedOverlays Program* lefordításából keletkezett tárgykód szerkezete látható.

```
program NestedOverlays;
overlay procedure One;
begin
  ...
end;
overlay function Two(Num : byte) : byte;
  overlay function Three : real;
  begin
  ...
  end;
  overlay procedure Four;
  begin
  ...
  end;
begin {Two}
  ...
end;
begin {NestedOverlays}
  ...
end.
```




*5. ábra. A NestedOverlay program szerkezete*

# 19. Néhány kiválasztott implementációs bővítés

A Turbo Pascal rendszer — mint beépített képernyős szövegszerkesztővel rendelkező programozási nyelv — különböző operációs rendszerek környezetében installálható. Mindegyikben biztosítja az adott rendszerre és fordítóprogramra jellemző járulékos szolgáltatásokat:

- a nyílt helyfoglalást a programváltozók számára;
- a különleges rendszerfüggvényeket és -eljárásokat;
- a rendszerrutinok közvetlen hívását;
- a gépi szintű programozási lehetőséget és
- a külső programok indítását.

E bővítéseket a CP/M operációs rendszerben érvényes alakban mutatjuk be.

## 19.1. Nyílt helyfoglalás programváltozók számára

A változódeklaráció **absolute** kulcsszavával helyet foglalhatunk a változó számára egy másik változóhoz hozzárendelt címen vagy egy előírt című tárhelyen.

Az ilyen deklaráció a deklarált változó nevéből, a kettőspontból, az **absolute** kulcsszóból, a tárhely-meghatározásból és a pontosvesszőből áll. A tárhely meghatározása egy előzőleg deklarált név vagy egy abszolút cím (a 8 bites mikroszámítógépen egy *integer* típusú konstans, az IBM PC esetén pedig kettősponttal elválasztott szegmens- és eltolásérték).

*Szintaxis*

nyílt_ helyfoglaló_ deklaráció:
  deklarált_ név : típusmeghatározás;
    **absolute** tárhely-meghatározás;
deklarált_ név:
  azonosító
tárhely-meghatározás:
  változónév
  konstans
  konstansnév
változónév:
  azonosító

**Példa**

8 bites mikroszámítógépen:

```
var
  CharStr : string[6];
  StrLen : byte absolute CharStr;
  Buffer : array[0..15] of byte absolute $2000;
function Fun(var IntVar : integer) : byte;
var
  LSB : byte absolute IntVar;
begin
  if LSB = Lo(IntVar) then ...
  ...
end;
```

- A *CharStr* változónak kijelölt tárhely első bájtja azonos a *Strlen* változó első bájtjával. Eszerint a *StrLen*-re hivatkozás ugyanazt az adatot képviseli, mint a *Length(CharStr)* és az *ord(CharStr[0])*.
- A *Buffer* változó a memória $2000-es címén kezdődő tárterületet kapja meg.
- Az *LSB* változó ugyanazt a memóriabájtot kapja helyéül, mint az *IntVar* paraméterhez hozzárendelt változó alacsonyabb helyi értékű bájtja, ezért a *Fun* függvényben szereplő reláció mindig igaz.

## 19.2. Különleges rendszerfüggvények és eljárások

### *Az Addr függvény*

Hívása: *Addr (Nam)*

A *Nam* egy változó vagy egy alprogram neve. Az *Addr* függvény értéke *integer* típusú adat, a *Nam* változó (vagy alprogram) kijelölt tárcímét adja meg.

*Addr(Mem*[2])

### *Az OvrDrive eljárás*

Hívása: *OvrDrive (Drv)*

A *Drv integer* típusú kifejezés. Az *OvrDrive* eljárás hatására az átlapolt programrészek a *Drv* által meghatározott mágneslemezes egységről töltődnek be (0 = alapértelmezés szerinti egység, 1 = A, 2 = B stb.).

*OvrDrive*(2)

## 19.3. Rendszerrutinok hívása (8 bites mikroszámítógép)

A rendszerrutinok (alprogramok) a *Bdos* és *Bios* eljárással, valamint a *Bdos, BdosHL, Bios* és *BiosHL* függvénnyel közvetlenül hívhatók.

### *A Bdos eljárás*

Hívása: *Bdos (Fun,Par)*
*Bdos (Fun)*

A *Fun* és az (elhagyható) *Par integer* típusú kifejezés. A *Bdos* eljárás végrehajtásakor a processzor C regiszterébe betöltődik a *Fun* képviselte adat alacsonyabb helyi értékű bájtja, ezután a program a 0005-ös címen folytatódik (vagyis a *Fun* sorszámú BDOS rendszerfunkciót hajtja végre). Ha a hívás kétparaméteres, akkor az említett ugrás előtt a *Par* képviselte adat a DE regiszterbe kerül.

### *A Bdos függvény*

Hívása: *Bdos (Fun,Par)*
*Bdos (Fun)*

A *Fun* és az (elhagyható) *Par integer* típusú kifejezés. A *Bdos* függvényhívás hatása azonos a *Bdos* eljárásával; a függvény értéke *byte* típusú adat, amelyet a BDOS az A regiszterben ad át.

### *A BdosHL függvény*

Hívása: *BdosHL (Fun,Par)*
*BdosHL (Fun)*

A *Fun* és az (elhagyható) *Par integer* típusú kifejezés. A *BdosHL* függvényhívás hatása azonos a *Bdos* függvényével; a függvény értéke *integer* típusú adat, amelyet a BDOS a HL regiszterben ad át.

### *A Bios eljárás*

Hívása: *Bios (Fun,Par)*
*Bios (Fun)*

A *Fun* és az (elhagyható) *Par integer* típusú kifejezés. A *Bios* eljárás a *Fun* sorszámú BIOS alrendszerfunkciót hajtja végre. Ha a hívásban szerepel a *Par* paraméter, akkor előtte a *Par* képviselte adat a processzor BC regiszterébe kerül.

*A Bios függvény*

Hívása: *Bios (Fun,Par)*
*Bios (Fun)*

*A Fun* és az (elhagyható) *Par integer* típusú kifejezés. A *Bios* függvényhívás hatása azonos a *Bios* eljárásával; a függvény értéke *byte* típusú adat, amelyet a BIOS az A regiszterben ad át.

*A BiosHL függvény*

Hívása: *BiosHL (Fun,Par)*
*BiosHL (Fun)*

A *Fun* és az (elhagyható) *Par integer* típusú kifejezés. A *BiosHL* függvényhívás hatása azonos a *Bios* függvényével; a függvény értéke *integer* típusú adat, amelyet a BDOS a HL regiszterben ad át.

## 19.4. Gépi szintű programozás

Néha szükség van egy gépi kódú alprogram beillesztésére a Turbo Pascal nyelvben megírt programba (vagy legalább egy gépi szintű „betét"-re). E célra az **external** és az **inline** kulcsszót használhatjuk.

Ha egy alprogram törzsét képező blokk helyett az

**external** *integer_* típusú_ konstans

szerkezet fordul elő, akkor ez azt jelenti, hogy a gépi kódú alprogram e konstanssal meghatározott tárcímen található. Fontos, hogy az alprogram végrehajtásának eredményeként az SP regiszter tartalma (a veremméret) ne változzék!

**Példa**

```
function Factorial(Arg : byte) : integer;
external $6000;
```

- A *Factorial* függvény deklarációs és végrehajtó részét az **external** kulcsszót tartalmazó szerkezet helyettesíti.
- A *Factorial* függvényt gépi kódban írták, és a memória $6000-es címén található.

Az **external** szerkezet csupán a gépi kódú alprogram helyét mutatja meg, az **inline** utasítással azonban gépi kódot is generálhatunk. Az **inline** utasítás az **inline** kulcsszóból és az azt követő, kerek zárójelekkel körülvett kódelemsorozatból áll. A kód elemeit perjelek választják el egymástól; minden kódelem + vagy — (mínusz) jellel elválasztott adatelemekből áll. Az adatelem lehet *integer* típusú konstans, változó vagy egy alprogram azonosítója és a *-gal jelölt programszámláló (PC) pillanatnyi értéke.

**Példa**

```
inline(20/$40/Fun-2/*+3)
```

Minden elem 1 bájtot vagy 1 szót (2 bájtot) hoz létre. Az előállított adat az adatelemeken végrehajtott műveletek eredménye. A változó vagy az alprogram azonosítója e változó (vagy alprogram) címét képviseli, a programszámláló szimbóluma pedig azt a címet, ahova a generált kód kerül.

Ha egy kódelem kizárólag konstansokból és szeparátorokból áll, értéke pedig 0..255 között van, akkor a fordítóprogram 1 bájtot generál. Ha az érték ennél nagyobb, ill. a kódelem változónevet vagy alprogramnevet, esetleg programszámláló-hivatkozást tartalmaz, akkor a fordítóprogram 1 szót generál. E szóban az alacsonyabb helyi értékű bájt megelőzi a magasabb helyi értékűt.

Ez a szabály felülbírálható a kódelem elé helyezett < vagy > jellel. Az első esetben a szó alacsonyabb helyi értékű bájtja generálódik, a másodikban pedig 2 bájt még akkor is, ha a második (magasabb helyi értékű) bájt tartalma 0.

**Példa**

```
inline(>$12/<$3456)
```

- A kód első eleme 1 bájtnyi adatot képvisel, de a > jel miatt 2 bájtot generál: $12-t és $00-t.
- A kód második eleme egyszavas adatot képvisel, de a < jel miatt csak 1 bájtot generál: $56-ot.
- Az **inline** utasítás 3 bájtot generál, sorrendjük $12, $00, $56.
- Ha az utasítás alakja

```
inline($12/$3456)
```

lenne, akkor a generált kód: $12, $56, $34.

**Példa**

(A *Turbo Pascal Reference Manual, Version 3.0* szerint, a Borland International engedélyével.)

```
type
  AnyString = string[255];
```

```
procedure ToUpper(var Str : AnyString);
begin
  inline(
        $2a/Str/
        $46/
        $04/
        $05/
        $ca/*+20/
        $23/
        $7e/
        /$fe/$61/
        $da/*-9/
        $fe/$7b/
        $d2/*-14/
        $d6/$20/
        $77/
        $c3/*-20/
end;
```

- Az eljárás a *Str* paraméterhez rendelt karakterlánc típusú változóban az összes kisbetűt nagybetűre cseréli.
- Az **inline** utasításban szereplő kódelemek a következő assembly nyelvű programot alkotják:

```
      LD   HL,(Str)
      LD   B,(HL)
      INC  B
L1:   DEC  B
      JP   Z,L2
      INC  HL
      LD   A,(HL)
      CP   'a'
      JP   C,L1
      CP   'z'+1
      JP   NC,L1
      SUB  20H
      LD   (HL),A
      JP   L1
L2:
```

## 19.5. Külső programok indítása

A Turbo Pascal nyelvben írt programokban két segédeljárást használhatunk, ezek külső programok indítását (programok láncolását) teszik lehetővé. E két eljárás az

*Execute(FileVar)* és a *Chain(FileVar),*

ahol *FileVar* egy fájlváltozó neve. E változó azonosítja a betöltendő (külső) programot tartalmazó állományt. Az *Execute* eljárás .COM kiterjesztésű programot indít, a *Chain* pedig .CHN kiterjesztésűt.

A *Chain* eljárással hívott program adatokat kaphat a hívó programból akár globális változók révén, akár fix cím(ek)en levő változó(ko)n keresztül. A globális változókat — ha használjuk őket — mindkét programban az első helyen és azonos sorrendben kell deklarálni. E szabály betartásával a hívó és a hívott programban szereplő (globális) változóhivatkozások ugyanazokra az adatokra vonatkoznak.

**Példa**

Két program, amelyből az első átad a másodiknak egy karakterlánc típusú és egy számadatot.

```
program First;
var
  LastName : string[8];
  Age : byte;
  NameFile : file;
begin
  Write('Jan ');
  LastName:='Bielecki';
  Age:=44;
  Assign(NameFile,'Second.chn');
  Chain(NameFile)
end.
{*********************************}
program Second;
var
  Surname : string[8];
  Age : byte;
begin
  Writeln(SurName,' is',Age:3,' now')
end.
```

- Feltételezzük, hogy a lefordított *Second* program a *Second.chn* állományba került.
- A *'Bielecki'* tartalmú karakterlánc-változót a *First* programban a *LastName* név képviseli, a *Second* programban pedig a *SurName* név.
- A két program végrehajtásának eredménye a

Jan Bielecki is 44 now

kiírás.

# 20. Az adatok ábrázolása (8 bites mikroszámítógépeken)

## 20.1. Rendezett típusú adatok

A rendezett típusú adatok az operatív tár 1 vagy 2 bájtját foglalják le.

Egybájtos ábrázolásúak a *char* típusú adatok, valamint azok a felsorolási típusú adatok, amelyek 256 elemnél kisebb elemszámú halmazhoz kapcsolódnak, ill. azok a *min..max* intervallum típusú adatok, amelyekben az *ord(min)* és az *ord(max)* a 0..255 intervallumhoz tartozik. A *boolean* és a *byte* típusú adatok is 1 bájt tárhelyet foglalnak le.

Kétbájtos ábrázolásúak az *integer* típusú adatok, az 1 bájtban nem ábrázolható intervallum típusú adatok, valamint a 256 elemnél nagyobb elemszámú felsorolási típusú adatok. A 2 bájtot elfoglaló adatokban az alacsonyabb helyi értékű bájt mindig megelőzi a magasabb helyi értékűt.

**Példa**

```
type
  Color = (Red,Green,Blue,Yellow,Orange);
  Hue = Green..Yellow;
  Selector = (enum,card,bool);
  union = record
            case Selector of
              enum: (Rng : Hue);
              card: (Int : integer);
              bool: (Log : boolean)
          end;
const
  HueVar : union = (Rng : Green);
begin
  wirth HueVar do
    Write(Int,Log:5)
end.
```

- Mivel az *ord(Green)*<=255 és az *ord(Yellow)*<=255, a *Hue* típusú, *Rng* azonosítójú mező ábrázolásához elegendő 1 bájt is.
- Az *Int* mező alacsonyabb helyi értékű bájtja fedi az *Rng* és a *Log* mező bájtját.
- A *Write* utasítás végrehajtása következtében az 1 TRUE kiírás jelenik meg.

## 20.2. Valós típusú adatok

A *real* típusú adatok az operatív tár 6 bájtját foglalják le. Ezek az adatok egybájtos kitevőből és ötbájtos mantisszából tevődnek össze; a legkisebb címen a szám kitevője van, ezt a mantissza bájtjai követik (a legalacsonyabb helyi értékű bájttól a legmagasabbig). A kitevő értéke $80-nal eltolt; a mantissza normalizált alakú, és a legmagasabb helyi értékű bitje nélkül tárolódik. Ez az ún. implicit bit megadja a mantissza előjelét. Egy *real* típusú, 0.0 értékű adat kitevője csupa 0-s bitből áll.

**Példa**

```
type
  union = record
            case boolean of
              false: (Float : real);
              true: (Arr : array[0..50] of byte)
          end;
const
  RealVar : union = (Arr : ($83,0,0,0,0,$80));
begin
  with RealVar do
    Write(Float)
end.
```

- A *RealVar* változó első bájtja $83 értékű adat, a kitevő értéke tehát 3.
- A $80 értékű bájt a mantissza legmagasabb helyi értékű bájtja. Jelentése: a mantissza negatív előjelű, értéke pedig

10000000 00000000 00000000 00000000 00000000

(A mantissza legmagasabb helyi értékű bitje itt a negatív előjelet képviseli.)

Mivel a mantissza értéke —0.5, a kitevőé pedig 3, a *RealVar* kezdeti értéke $-0.5 * 2^3 = -4.0$.

- A *Write* utasítás a —4.0 számot írja ki.

## 20.3. Karakterlánc típusú adatok

A **string**[*n*] típusú adatok ábrázolása *n*+1 bájtot igényel. A karakterlánc típusú adat első bájtja megadja ezen adat karaktereinek számát (a karakterlánc hosszát).

**Példa**

```
const
  StrVar : string[5] = 'Janek';
procedure CutOff(var Par);
var
  Len : byte absolute Par;
```

```
begin
  Len :=Len −2
end;
begin
  Writeln(StrVar);
  CutOff(StrVar);
  Write(StrVar)
end.
```

- A *Len* változón végrehajtott művelet valójában a *StrVar* változó legmagasabb helyi értékű bájtjára vonatkozik.
- A program a

Janek
Jan

szavakat írja ki.

## 20.4. Halmaz típusú adatok

A *min* legkisebb és a *max* legnagyobb elemű rendezett típuson alapuló halmaz típusú adatok a memória

ord(max) **div** 8 — ord(min) **div** 8 + 1

bájtját foglalják le. A halmaz típusú adatok úgy ábrázolhatók, mintha a rendezett alaptípus 256 elemből állna. Így egy változónak 32 bájtra* lenne szüksége, amelyek közül azonban az esetek többségében csak keveset használunk**, ezért a 32 bájtból a rendszer elhagyja azokat a „szélső" bájtokat, amelyek értéke sosem változik.

A **set of** 10..16 típusú adat ábrázolásához pl. nem 32 bájtra van szükség (az x-szel az adat „aktív" pozícióit jelöltük).

00000000 ... 00000000 0000000x xxxxxx00 00000000,

hanem csak 2 bájt a helyigénye :

0000000x xxxxxx00

(a legalacsonyabb helyi értékű bájttól a legmagasabbig).

**Példa**

```
type
  union = record
            case boolean of
              false: (aSet : set of 10..16);
            true: (anInt : integer)
          end;
```

* A lektor megjegyzése: minden elem 1 bitet foglal le, így 256 elem 256/8=32 bájtot.
** Hiszen a halmaz típusú változó általában 256-nál kevesebb elemből áll.

```
const
  SetVar : union = (aSet : [16]);
begin
  with SetVar do
    Write(anInt)
end.
```

- Az *aSet* mező 2 bájtot foglal le, tárbeli alakja:

00000000 00000001.

- A *Write* utasítás 256-ot ír ki.

## 20.5. Mutató típusú adatok

A mutató típusú adatok 2 bájtot foglalnak le. Ábrázolásuk azonos az *integer* típusú adatokéval, tartalmuk viszont más: memóriacímet jelentenek. A **nil** mutató által képviselt adat csupa 0-s bitből áll.

**Példa**

```
type
  union = record
            case boolean of
              false: (Ref :^byte);
              true: (Int : integer)
          end;
const
  PtrVar : union = (Int : 0);
begin
  Write(PtrVar^.Ref = nil)
end.
```

- A *PtrVar* változó 2 bájtot foglal le.
- A *Write* utasítás a *TRUE* feliratot írja ki.

## 20.6. Tömb típusú adatok

A tömbök elemei soronként helyezkednek el az operatív tárban.

**Példa**

```
type
  union = record
            case boolean of
              false: (Arr : array[boolean,
```

```
                    boolean] of byte);
                  true: (Vec : array [1..4] of byte)
            end;
const
  ArrVar : union = (Arr : ((11,12),(21,22)));
var
  Index : 1..4;
begin
  with ArrVar do
    for Index:=1 to 4 do
      Write(Vec[Index]:3)
end.
```

- A *Write* utasítás a következő számokat írja ki: 11, 12, 21, 22.

## 20.7. Rekordtípusok

A rekordmezők deklarációjuk sorrendjében helyezkednek el a memóriában. Ha a rekord nem tartalmaz változórészt (variánst), akkor mérete azonos a mezőméretek összegével, különben a fix rész és a legnagyobb helyigényű változat méretének összege adja meg a rekord ábrázolásához szükséges tárhelyet. Akár tartalmaznak változórészt, akár nem — a rekordok fix méretűek. Ez a tény lényeges a beviteli, ill. kiviteli műveletek szempontjából.

**Példa**

```
var
  RecVar : record
                 case Selector : integer of
                 6: (Arr : array[1..3,1..2] of byte);
                 8: (aSet : set of  A'..'z')
           end;
begin
  with RecVar do
    Write(SizeOf(RecVar),
          SizeOf(Arr):2,
          SizeOf(aSet):2)
end.
```

- A *RecVar* fix részének mérete 2 bájt.
- Az első variáns hossza 6 bájt, a másodiké 8.
- A rekord összmérete tehát 10 bájt.
- A *Write* utasítás a következő számokat írja ki: 10, 6 és 8.

# Irodalom


1. *Bielecki J.:* Turbo Pascal z grafika dla IBM PC, Varsó (előkészületben). WNT
2. *Cherry G.:* Pascal Programming Structures. Reston, Virginia, Restone Publishing Company, 1980.
3. *Iglewski M.—Madey J.—Matwin S.:* Pascal. Varsó, WNT, 1986.
4. *Jensen K.—Wirth N.:* Pascal — User Manual and *Report.* Berlin Springer Verlag, 1974.
5. *Schauer H.:* Pascal für Anfänger. Wien, R. Oldenbourg Verlag, 1976.
6. *Turbo Graphix Toolbox.* California, Borland International, Scotts Valley, 1985.
7. *Turbo Pascal version* 3.0. California, Borland International, Scotts Valley, 1985.
8. *Turbo Toolbox.* California, Borland International, Scotts Valley, 1985.
9. *Turbo Tutor.* California, Borland International, Scotts Valley, 1985.
10. *Wirth N.:* The programming language Pascal (Revised Report). N° 6. Zürich, Bericht der Fachgruppe Computer-Wissenschaften, Eidgenössische Technische Hochschule, 1972.

# Függelék

## A Függelék: ASCII kódtáblázat

| Dec | Hex | Char | Dec | Hex | Char |
|---|---|---|---|---|---|
| 0 | 00 | ^@NUL | 32 | 20 | |
| 1 | 01 | ^A SOH | 33 | 21 | ! |
| 2 | 02 | ^B STX | 34 | 22 | " |
| 3 | 03 | ^C ETX | 35 | 23 | # |
| 4 | 04 | ^D EOT | 36 | 24 | $ |
| 5 | 05 | ^E ENQ | 37 | 25 | % |
| 6 | 06 | ^F ACK | 38 | 26 | & |
| 7 | 07 | ^G BEL | 39 | 27 | ' |
| 8 | 08 | ^H BS | 40 | 28 | ( |
| 9 | 09 | ^I HT | 41 | 29 | ) |
| 10 | 0A | ^J LF | 42 | 2A | * |
| 11 | 0B | ^K VT | 43 | 2B | + |
| 12 | 0C | ^L FF | 44 | 2C | , |
| 13 | 0D | ^M CR | 45 | 2D | — |
| 14 | 0E | ^N SO | 46 | 2E | . |
| 15 | 0F | ^O SI | 47 | 2F | / |
| 16 | 10 | ^P DLE | 48 | 30 | 0 |
| 17 | 11 | ^Q DC1 | 49 | 31 | 1 |
| 18 | 12 | ^R DC2 | 50 | 32 | 2 |
| 19 | 13 | ^S DC3 | 51 | 33 | 3 |
| 20 | 14 | ^T DC4 | 52 | 34 | 4 |
| 21 | 15 | ^U NAK | 53 | 35 | 5 |
| 22 | 16 | ^V SYN | 54 | 36 | 6 |
| 23 | 17 | ^W ETB | 55 | 37 | 7 |
| 24 | 18 | ^X CAN | 56 | 38 | 8 |
| 25 | 19 | ^Y EM | 57 | 39 | 9 |
| 26 | 1A | ^Z SUB | 58 | 3A | : |
| 27 | 1B | ^[ ESC | 59 | 3B | ; |
| 28 | 1C | ^\FS | 60 | 3C | < |
| 29 | 1D | ^] GS | 61 | 3D | = |
| 30 | 1E | ^^ RS | 62 | 3E | > |
| 31 | 1F | ^_ US | 63 | 3F | ? |

| Dec | Hex | Char | Dec | Hex | Char |
|---|---|---|---|---|---|
| 64 | 40 | @ | 96 | 60 | ‘ |
| 65 | 41 | A | 97 | 61 | a |
| 66 | 42 | B | 98 | 62 | b |
| 67 | 43 | C | 99 | 63 | c |
| 68 | 44 | D | 100 | 64 | d |
| 69 | 45 | E | 101 | 65 | e |
| 70 | 46 | F | 102 | 66 | f |
| 71 | 47 | G | 103 | 67 | g |
| 72 | 48 | H | 104 | 68 | h |
| 73 | 49 | I | 105 | 69 | i |
| 74 | 4A | J | 106 | 6A | j |
| 75 | 4B | K | 107 | 6B | k |
| 76 | 4C | L | 108 | 6C | l |
| 77 | 4D | M | 109 | 6D | m |
| 78 | 4E | N | 110 | 6E | n |
| 79 | 4F | O | 111 | 6F | o |
| 80 | 50 | P | 112 | 70 | p |
| 81 | 51 | Q | 113 | 71 | q |
| 82 | 52 | R | 114 | 72 | r |
| 83 | 53 | S | 115 | 73 | s |
| 84 | 54 | T | 116 | 74 | t |
| 85 | 55 | U | 117 | 75 | u |
| 86 | 56 | V | 118 | 76 | v |
| 87 | 57 | W | 119 | 77 | w |
| 88 | 58 | X | 120 | 78 | x |
| 89 | 59 | Y | 121 | 79 | y |
| 90 | 5A | Z | 122 | 7A | z |
| 91 | 5B | [ | 123 | 7B | { |
| 92 | 5C | \ | 124 | 7C | : |
| 93 | 5D | ] | 125 | 7D | } |
| 94 | 5E | ^ | 126 | 7E | ~ |
| 95 | 5F | — | 127 | 7F | DEL |

# B Függelék: Hibakódok

## Fordítás alatti hibák

1 Hiányzó ';'
2 Hiányzó ':'
3 Hiányzó ','
4 Hiányzó '('
5 Hiányzó ')'
6 Hiányzó '='
7 Hiányzó ':='
8 Hiányzó '['
9 Hiányzó ']'
10 Hiányzó '.'
11 Hiányzó '..'
12 Hiányzó 'BEGIN'
13 Hiányzó 'DO'
14 Hiányzó 'END'
15 Hiányzó 'OF'
17 Hiányzó 'THEN'
18 Hiányzó 'TO' vagy 'DOWNTO'
20 Hiányzó *boolean* típusú kifejezés
21 Hiányzó fájlváltozó
22 Hiányzó egész típusú konstans
23 Hiányzó egész típusú kifejezés
24 Hiányzó egész típusú változó
25 Hiányzó egész vagy valós típusú konstans
26 Hiányzó egész vagy valós típusú kifejezés
27 Hiányzó egész vagy valós típusú változó
28 Hiányzó mutató típusú változó
29 Hiányzó rekord típusú változó
30 Hiányzó elemi típus
31 Hiányzó elemi kifejezés
32 Hiányzó karakterlánc típusú konstans
33 Hiányzó karakterlánc típusú kifejezés
34 Hiányzó karakterlánc típusú változó
35 Hiányzó szövegfájl-azonosító
36 Hiányzó típusazonosító
37 Hiányzó elemtípus nélküli fájlazonosító
40 Definiálatlan címke
41 Ismeretlen azonosító vagy szintaxishiba

42 Ismeretlen mutatótípus
43 Újradefiniált címke vagy azonosító
44 Hibás típus
45 Intervallumon kívüli konstansérték
46 Hibás konstanstípus vagy CASE-kiválasztó
47 Hibás argumentum vagy operátor
48 Hibás eredménytípus
49 Hibás karakterlánchossz
50 Hibás a karakterlánckonstans mérete
51 Hibás az intervallumtípus alaptípusa
52 A felső határ kisebb az alsónál
53 Fenntartott szó
54 Hibás értékadás
55 A karakterlánckonstans egy sornál hosszabb
56 Hibás egész típusú konstans
57 Hibás valós típusú konstans
58 Hibás karakter az azonosítóban
60 Konstans itt nem használható
61 Fájl és mutató típusú változók itt nem használhatók
62 Strukturált változók itt nem használhatók
63 Szövegfájlok itt nem használhatók
64 Szövegfájlok és elemtípus nélküli fájlok itt nem használhatók
65 Elemtípus nélküli fájlok itt nem használhatók
66 Bemeneti, ill. kimeneti műveletek itt nem használhatók
67 A fájlváltozók igénylik a VAR deklarációt
68 A fájlok nem lehetnek fájlelemek
69 Hibás mező-hozzárendelés
70 Túl nagy halmaz
71 Hibás GOTO
72 Az aktuális blokkon kívüli címke
73 Definiálatlan FORWARD eljárás
74 Hibás INLINE utasítás
75 Hibás az ABSOLUTE használata
90 Ismeretlen fájlváltozó
91 Váratlan programvég
97 Túl sok egymásba ágyazott WITH
98 Memóriatúlcsordulás
99 A fordítóprogram túlcsordult

## Futásidejű hibák

Egy program végrehajtásakor kétféle hiba fordulhat elő: fatális és beviteli, ill. kiviteli hiba.

Fatális hiba esetén a következő felirat íródik ki:

```
Run-time error NN, PC=addr
Program aborted
```

ahol *NN* a hiba kódja, az *addr* pedig a hibát előidéző utasítás memóriacíme.

Beviteli, ill. kiviteli hiba esetén az üzenet alakja:

I/O error NN, PC=addr
Program aborted

ahol *NN* és *addr* jelentése azonos az előbbivel.

### *Fatális hibák*

01 Fixpontos túlcsordulás
02 Nullával való osztás
03 Hibás az *Sqrt* argumentuma
04 Hibás az *Ln* függvény argumentuma
10 255 karakternél hosszabb karakterlánc létrehozása, ill. 1 karakternél hosszabb karakterlánc *char* típusú adattá konvertálása
90 Hibás tömbindex
91 Hibás értékadás: az adat értéke a megengedett intervallumon kívülre esik
92 A *Trunc* vagy a *Round* függvény argumentuma a megengedett intervallumon kívülre esik
FF Elfogyott a memóriahely

### *Beviteli, illetve kiviteli hibák*

01 A fájl nem létezik
02 A fájl nincs nyitva olvasásra
03 A fájl nincs nyitva írásra
04 A fájl nincs nyitva
10 Számábrázolási hiba
20 Tiltott művelet
21 Tiltott művelet
22 Az *Assign* eljárás használata nem megengedett
90 Összeférhetetlenség a rekordméreteknél
91 Fájlon kívüli hozzáférés kísérlete
99 Váratlan fájlvég
F0 A mágneslemez írási hibája
F1 Katalógustúlcsordulás
F2 Fájltúlcsordulás
F3 Túl sok fájl van már nyitva
FF Hiányzó fájl

# C Függelék: Képernyős szövegszerkesztő

## Alapfogalmak

A *WordStar* képernyős szövegszerkesztőt a MikroPro cég terjesztette el a mikroszámítógépes piacon, 1977-től kezdve. A program fő előnye (és elterjedésének oka) a valódi „full screen" (teljes képernyős) szerkesztési elv, és a program gépfüggetlen installálásának lehetősége. Számos szövegszerkesztő vetélytárs megjelenése után a *WordStar* megőrizte vezető pozícióját, és néhány „mutációja" a 8 és 16 bites számítógépeken is használható. A következőkben a Turbo Pascal rendszerbeli implementációját mutatjuk be.

Több más szövegszerkesztőtől eltérően a *WordStar* program főleg a szövegbevitelt támogatja, és kevésbé alkalmas a szövegfeldolgozó parancsok végrehajtására. Ennek következményeként a szövegen végrehajtandó műveletek olyan karakterekkel végezhetők, amelyek láthatóan közvetlenül nem ábrázolhatók. E karaktereket vezérlőkaraktereknek nevezzük; bevitelükhöz a CTRL billentyűt egy vagy két betűjeles billentyűvel együtt kell lenyomni. Mivel a továbbiakban gyakran hivatkozunk a vezérlőkarakterekre, jelképes jelölésükben a ^ jel a CTRL billentyű lenyomását jelenti. Így pl. a CTRL A vezérlőkarakter jelölése ^A, bevitele pedig a CTRL billentyű lenyomott állapotában leütött A billentyűvel lehetséges. Azokban az esetekben, amikor a szövegszerkesztőnek kiadandó parancs két vezérlőkarakterből áll (pl. ^KB), lenyomjuk a CTRL billentyűt, és ezután ütjük le a K és a B billentyűt (a második billentyű leütésekor már nem kell lenyomva tartani a CTRL billentyűt).

A szövegszerkesztő szokásos használata a főmenüből kiadott E paranccsal kezdődik. Ennek hatására elindul a szövegszerkesztő program, amely „kiteríti" a képernyőre az előzőleg W paranccsal azonosított munkaállomány tartalmát (kezdőrészét). A szövegbevitel vagy -módosítás befejeztével a ^KD paranccsal visszatérhetünk a főmenühöz, ahonnan kiadhatjuk az S parancsot, amely mágneslemezre menti ki a munkaállományt.

A szövegszerkesztő program indítása után a számítógép többfunkciós írógépként használható. Az írógép alapvető tulajdonsága, hogy minden bevitt karakter azonnal megjelenik a képernyőn. A hibásan bevitt karaktereket a ← billentyűvel (Backspace) törölhetjük. Ha a javítandó hibák a már korábban bevitt szavakban találhatók, akkor a kurzormozgató billentyűkkel a kurzort elvihetjük a javítás helyére, és a módosítást azonnal végre is hajthatjuk. A módosítás szabályait a következő szakasz tartalmazza.

## Szövegbevitel, -módosítás és -törlés

Mint már említettük, a szövegszerkesztő program indítása után szöveget gépelünk be vagy módosítunk. E tevékenységek lényegében új szavak bevitelét vagy a már bevitt szavakon végzett műveleteket jelentik. Itt a szó fogalmán tetszőleges karaktersorozatot értünk, amelyet egy szóköz vagy más elválasztókarakter (pl. vessző, pont, kettőspont, aposztróf) zár le. E definíció értelmében példaprogramunk

```
program JanB;
begin
    Write('I am JanB')
end.
```

a következő szavakból áll:

```
program
JanB;
begin
Write('
I
am
JanB'
)
end.
```

A képernyőn megjelenített szövegrészen belül kitüntetett a kurzor feletti karakter szerepe. A szövegmanipulációk mindig e karakterre (vagy az őt tartalmazó szóra, ill. sorra) vonatkoznak. A kurzort karakterenként, szavanként, soronként és laponként mozgathatjuk (a *lap* több sorból álló szövegrész, mérete egyenlő a képernyő kapacitásával).

A kurzort egy pozícióval balra, jobbra, felfelé vagy lefelé a ^S, ^D, ^E és ^X parancsokkal mozgathatjuk. E parancsok betűjeles billentyűi négyzetet alkotnak:

```
      E
   S     D
      X
```

Így könnyű megjegyezni, hogy az ^E parancs egy pozícióval felfelé, a ^X egy pozícióval lefelé, az ^S és a ^D egy pozícióval balra, ill. jobbra mozgatja a kurzort. Látható, hogy a kurzormozgató parancsok ilyen hozzárendelése nem az egyes betűkhöz kapcsolódik, hanem a négy billentyű síkbeli elrendezésén alapul.

A szavankénti kurzormozgatást az ^A és az ^F parancs valósítja meg:

```
         E
A     S        D     F
         X
```

Az ^A parancs egy szópozícióval balra (az előző szóra), az ^F pedig egy szópozícióval jobbra (a következő szóra) állítja a kurzort.

Laponkénti kurzormozgatásra (előre- és hátralapozásra) az ^R és a ^C parancs való. Billentyűik az E—X billentyűk által kijelölt tengely jobb oldalán helyezkednek el. E tengely bal oldalán viszont azok a billentyűk kaptak helyet, amelyek a képernyő tartalmát egy sorral lefelé (^W) vagy felfelé (^Z) mozgatják. A kurzor- és szövegmozgató billentyűk végleges elrendezése tehát a következő:

```
W        E        R
A  S           D  F
Z        X        C
```

Ez az elrendezés könnyen megtanulható, ezért nincs szükség külön billentyűfeliratokra vagy magyarázó táblázatra.

A fenti billentyűk jobb oldalán három újabb parancsbillentyűt találunk: ^T, ^Y, ^G:

T Y

G

A ^G parancs a kurzor feletti karaktert törli, a ^T parancs a kurzor feletti és a tőle jobbra álló karaktereket a szó végéig, a ^Y parancs pedig az egész sort. Végül megemlítjük a ← (Backspace) billentyűt, amely a kurzortól balra eső karaktert törli.

## Összefoglalás

^E A kurzor mozgatása egy sorral felfelé; az oszloppozíció nem változik.
^X A kurzor mozgatása egy sorral lefelé; az oszloppozíció nem változik.
^S A kurzor mozgatása egy pozícióval balra; ha a kurzor a sor elején áll (bal szélső pozíció), akkor a parancs hatástalan.
^D A kurzor mozgatása egy pozícióval jobbra; ha a kurzor a sor végén áll (jobb szélső pozíció), akkor a parancs hatástalan.
^A A kurzor mozgatása az előző szó elejére.
^F A kurzor mozgatása a következő szó elejére.
^R Lapozás felfelé.
^C Lapozás lefelé.
^W A képernyőtartalom mozgatása egy sorral lefelé ("roll down").
^Z A képernyőtartalom mozgatása egy sorral felfelé ("roll up").
^G Karaktertörlés.
^T Karaktertörlés a szó végéig.
^Y Sortörlés.
← A kurzortól balra eső karakter törlése.

Ezeket a parancsokat egészíti ki az ^N parancs, amely egy üres sort hoz létre, és az előbbi kurzormozgató billentyűkhöz hasonló hatású, de nagyobb „lépéssel" értelmezendő parancsok: a ^Q vezérlőkarakter után leütött

E R

S D

X C

billentyűk valamelyike. A ^QE parancs a kurzort a képernyő első sorába viszi, a ^QX parancs az utolsó sorba, a ^QS a sor elejére, a ^QD pedig a sor végére. A ^QR és a ^QC parancs a szöveg elejére, ill. végére viszi a kurzort.

Az itt felsorolt kurzormozgató parancsok lehetővé teszik a szöveghelyesség ellenőrzését és a szöveg módosítását. Módosításkor előfordulhat a már begépelt szöveg kiegészítése vagy a szöveg helyettesítése más karaktersorozatokkal.

Kiegészítéskor (beszúrás, insert) elegendő begépelni az új szöveget; a szövegszerkesztő program automatikusan beszúrja (a kurzor pillanatnyi pozíciójától kezdve) úgy, hogy az utána álló karaktereket jobbra tolja.

Módosításkor (felülírás, overwrite) a ^V paranccsal kikapcsolhatjuk ezt a szolgáltatást, így a begépelt karakterek felülírják a régieket. A ^V parancs újbóli kiadása visszaállítja a beszúrás üzemmódot.

## Összetett szövegműveletek

Az eddig leírt parancsok nemcsak szöveges állományok létrehozására alkalmasak, hanem a szövegfeldolgozást is lehetővé teszik. Mégis gyakran szükség van olyan műveletekre, amelyek a szöveg egy részét egyszerre kezelik (pl. szövegmásolás vagy -áthelyezés), vagy pedig képesek mintakeresésre és szöveghelyettesítésre. Erre használhatjuk a következő parancsokat.

### *A ^QF parancs*

Először a képernyőn megjelenő

FIND:

kérdésre adjuk meg a keresendő mondatot (mintát). A *mondat* tetszőleges karaktersorozat, melyet a CR billentyű (8 bites mikroszámítógép) vagy az Enter billentyű (IBM PC) (a továbbiakban RETURN) zár le. A következő

OPTIONS:

kérdésre adott válasz meghatározza a keresés módját. A leggyakoribb esetben (keresés a kurzor pozíciójától a szöveg végéig) elegendő a RETURN billentyű leütése. Ha a keresett mondat szerepel a szövegben, akkor a kurzor megáll a megtalált mondat utolsó karaktere mögött. Más keresési opciók pl. a következők:

B visszafelé keres;
W csak teljes szavakat keres;
U a kis- és nagybetűket azonosnak tekinti.

Ha egy *n* számot adunk meg opcióként, akkor a program a megadott mondat *n*-edik előfordulását keresi meg.

### *A ^QA parancs*

A ^QA parancs a ^QF bővítése: a keresendő (helyettesítendő) mondat megadása után megjelenik a

REPLACE WITH:

kérdés, amelyre válaszként írjuk be a helyettesítő mondatot. A keresés, ill. helyettesítés módját meghatározó opció lehet még:

G a művelet az egész szövegállományra vonatkozik, a kurzorpozíció közömbös;
N feltétel nélkül keres és helyettesít.

Ha az N opciót adtuk meg, akkor minden megtalált mondat helyére a helyettesítő mondat kerül. Ha nem adtuk meg az N opciót, akkor a program minden megtalált mondatnál megkérdezi, hogy kérjük-e a helyettesítést. Az Y billentyű leütése jelenti az igenlő választ; minden más billentyű hatására folytatódik a keresés. A keresési, ill. helyettesítési folyamat az ^U paranccsal bármikor megszakítható.

### *A ^KB parancs*

A ^KB parancs egy blokk kezdetének jelöli ki a pillanatnyi kurzorpozíciót; a teljes szövegblokk meghatározásához szükség van még a ^KK blokkzáró parancsra.

### *A ^KK parancs*

A ^KK parancs egy blokk végének jelöli ki a pillanatnyi kurzorpozícióit. A blokk kijelölésének (a ^KB és a ^KK parancsokkal) pillanatában a blokk összes karaktere inverz ábrázolásban vagy kisebb fényerővel jelenik meg. A következő ^Kx parancsok az így kijelölt szövegrészre vonatkoznak.

### *A ^KC parancs*

A ^KC parancs a ^KB és a ^KK paranccsal kijelölt blokkot átmásolja a pillanatnyi kurzorpozícióba. E művelet után a kurzor az átmásolt blokk első karakterére áll; a blokkmásolat kijelölése érvényben marad.

### *A ^KV parancs*

A ^KV parancs a ^KB és a ^KK paranccsal kijelölt blokkot áthelyezi a pillanatnyi kurzorpozícióba. E művelet után a kurzor az áthelyezett blokk első karakterére áll; a blokk kijelölése érvényben marad.

### *A ^KY parancs*

A ^KY parancs törli a ^KB és a ^KK paranccsal kijelölt blokkot. Megjegyzés: a ^KB, ^KK, ^KY parancssorozat egyetlen hatása a blokk-kijelölés megszüntetése (l. még ^KH parancsot).

### *A ^KR parancs*

A ^KR parancs a mágneslemezes állományban található szöveget beszúrja a pillanatnyi kurzorpozícióba. A feltett

Read block from file:

kérdésre válaszként a beillesztendő szövegállomány nevét kell megadni.

### *A ^KW parancs*

A ^KW parancs mágneslemezes állományba menti ki a ^KB és a ^KK paranccsal kijelölt blokkot. A feltett

Write block to file:

kérdésre válaszként annak a létrehozandó szövegállománynak a nevét kell megadni, amelybe a kijelölt blokk kerül. Ha ilyen nevű állomány már létezik, akkor a kimentés előtt törlődik és újból létrejön.

### *A ^KH parancs*

A ^KH parancs megszünteti a ^KB és a ^KK parancs hatását (érvényteleníti a blokk kijelölését).

## Segédparancsok

### *Az ^N parancs*

Az ^N parancs a kurzor pillanatnyi pozíciójába beszúrja a CR/LF karakterpárt, azaz új sort hoz létre.

### *Az ^I parancs*

Az ^I parancs a legközelebbi tabulátorpozícióba viszi a kurzort; a tabulátorpozíciókat itt a kurzor feletti sor szavainak kezdőkarakterei jelentik.

### *Az ^L parancs*

Az ^L parancs megismétli az utolsó ^QF parancsot.

### *Az ^U parancs*

Az ^U parancs bármelyik parancs végrehajtását megszakítja.

### *A ^P parancs*

A ^P parancs hatására az utána közvetlenül leütött vezérlőkarakter kódja bekerül a szövegbe. Így pl. a ^P ^M ^P ^J karaktersorozat a sorokat lezáró CR/LF karakterpárokat jelenti.

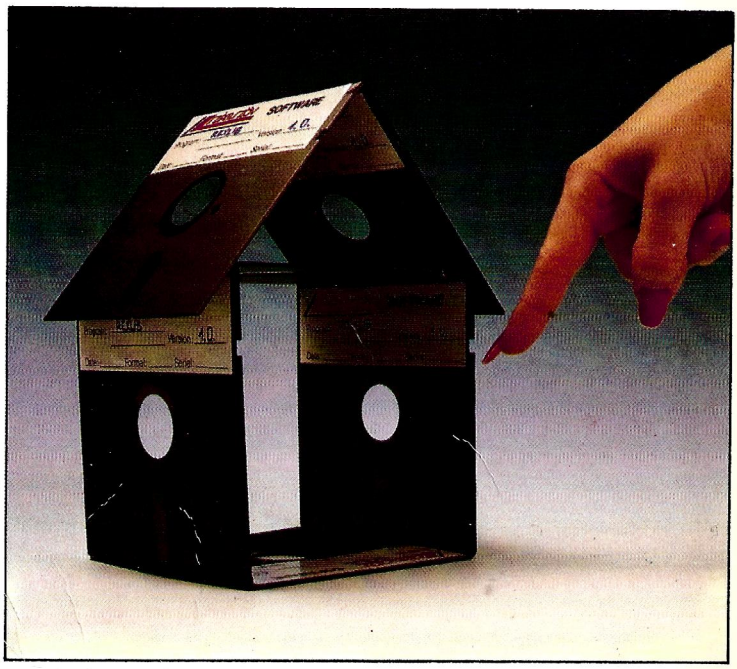
**Példa**

A következő karaktersorozat begépelésével (a RETURN billentyű jele *r*, a szóközé *s*) az egész szöveg 5 pozícióval jobbra tolható el:

^QR*sssss*
^QA ^P ^M ^P ^J*r* ^P ^M ^P ^J*ssssss*r NG*r*

# Tárgymutató

# REXLIB-PLUS



**képernyőkezelő**
**közös file-kezelő**
**residens rutin- és utility-csomag**

---

**TURBO PASCAL**
**programnyelven fejlesztők számára**

---

**MS-DOS és NOVELL hálózati**
**környezetben**

SZÁMSZÖV®

SZÁMÍTÁSTECHNIKAI KISSZÖVETKEZET

1116 Budapest XI., Hunyadi János u. 162.
Postacím: 1519 Bp. Pf. 353.
Telefon: 166-5322 ● Telex: 22-3600 szszv h