KISS GYÖRGY
2500 ESZTERGOM
ARANY J.U.4.

KISS GYÖRGY
2500 ESZTERGOM
ARANY JÁNOS U.4.
TEL: (33)314 603

PRIMO PASCAL

FELHASZNALÓI LEIRAS

TARTALOM

.PA

1. BEVEZETÉS

Az Ön által vásárolt PRIMO PASCAL lehetővé teszi, hogy PRIMO A-64-es személyi számitógépén a "PASCAL" magasszintű programnyelven irjon és futtasson programokat.
Ezen mikrogépes PASCAL verzió rendelkezik a standard PASCAL minden lényeges lehetőségével, kiegészitve néhány, kifejezetten a személyi számitógépes alkalmazást és a gép lehetőségeinek teljes kiaknázását segitő művelettel. (A "standard PASCAL" terminológia itt és a továbbiakban is a Kathleen Jensen-Niklaus Wirth: A PASCAL programozási nyelv cimű könyvben ismertetett PASCAL verzióra vonatkozik. [1]

A főbb eltérések a következők:
- FILE tipus nem értelmezett;
- variálható RECORD tipus nem értelmezett;
- eljárások és függvények nem lehetnek más eljárások és függvények paraméterei;
- adatok tárolása -- hasonlóan a BASIC rendszerhez -- a géphez csatlakozó magnetofonon lehetséges; az adatok ("tárképszerű": core image) kiirása és beolvasása közvetlenül programból történik; egy ilyen művelet egy file egészére vonatkozik (beleértve a file megnyitását és lezárását is).
- halmaz /SET/ tipus számossága max. 256 /!/, de egyéb megszoritások érvényesek, amelyeket a megfelelő helyen ismertetünk;
- RECORD tipus mezőelnevezése semelyik egyedi változónévvel nem egyezhet meg. Ha mégis igy lenne, pl. figyelmetlenségből, a hibajelzést a forditás során akkor kapjuk, ha a hibás mezőre /!/ hivatkozunk. Talán nem érdektelen azt sem megjegyezni, hogy az 55-ös kódú hibaüzenet ellenére a forditás eredménye kivételesen helyes lesz!

Ez a dokumentáció elsősorban olyanok számára készült, akik már többé kevésbé ismerik a standard PASCAL programozási nyelvet, de remélhetőleg azok is haszonnal forgatják, akik még csak más, - például BASIC - programozási nyelven szerzett gyakorlattal rendelkeznek.

1. A FORDITÓPROGRAM ELINDITÁSA

A PASCAL fordítóprogram betöltése a kazettáról éppúgy történik, mint bármely más PRIMO programé: helyezzük a kazettát magnónkba, irjuk be a LOAD parancsot és inditsuk el a magnót. A RETURN billentyü lenyomása után rövidesen megjelenik a "FOUND: PASCAL" felirat, jelezvén a PASCAL betöltésének megkezdését. Abban az esetben, ha a hibaszámláló értéke nem nulla, a program betöltését a RESET gombbal szakitsuk félbe, és a müveletet ismételjük meg, esetleg módositva magnónk hangerő - és hangszinszabályozó gombjainak állását. Szükség esetén a lejátszófej alkohollal történő megtisztitására, illetőleg demagnetizálására is sor kerülhet.

A program betöltés után automatikusan elindul. Ennek első jele a képernyő tetején megjelenő "End of Memory?" kérdés, és a villogó kurzor, továbbá hangjelzést is kapunk. Ilyenkor kétféle lehetőségünk van:

1. a RETURN billentyü megnyomása esetén a program alapértelmezésként a BASIC string memória végcimét tekinti a számára elérhető memória végének;

2. egy 65535-nél kisebb decimális szám beirása esetén a PASCAL e cim fölötti memóriaterületet érintetlenül hagyja. (Ne feledkezzünk meg arról, hogy a PRIMO képernyő-memóriája az 59392-es cimtől fölfelé helyezkedik el!)

E két lehetséges válasz valamelyike után a PASCAL újabb kérdést tesz fel:
"Length of Table ?"
Ilyenkor lehetőségünk van a fordító által készitett szimbólumtábla hosszának beállitására a generált kód javára, illetve rovására; tehát ha olyan PASCAL programot kivánunk irni, amely igen sok változót, tipust stb. kezel, e kérdésre érdemes egy nagy számot beirni válaszul; ez esetben ennyi byte-ot foglalunk a szimbólumtábla számára. Forditott esetben tehát ha hosszú, de kevés változót használó programot tervezünk irni, itt kisebb számot érdemes megadnunk.

(Egy szimbólum -- változó, tipus, eljárás stb. -- a szimbólumtáblában 10-15 byte-ot foglal el a nevén kivül, tehát például egy "VALTOZO" nevü változóra kb. 7 + 12 = 19 byte helyet számolhatunk.))
Ha nem irunk be semmilyen számot, csak a RETURN billentyüt nyomjuk le, a program alapértelmezésben átlagos PASCAL programok számára foglal helyet.

Előfordulhat olyan eset, hogy a két kérdésre adott válasz nem megfelelő tárkiosztást eredményez (pl. túl rövid vagy túl hosszú szimbólumtábla), ilyenkor a kérdéseket megismétli a program.

Figyelem: az itt beállitott memória-felosztáson csak a PASCAL újbóli betöltésével változtathatunk!

Ezek után a PASCAL bejelentkezik; a sor első poziciójában megjelenő

">" jel, az ún. prompt jelzi, hogy a PRIMO PASCAL szövegszerkesztője (editor) parancsot vár.

A PASCAL elindulása pillanatától gyakorlatilag teljesen független a PRIMO BASIC rendszerétől; igy nem használhatjuk a megszokott editort sem, helyette a PASCAL saját, kellően hatékony sor-editorral rendelkezik.

Ezen editor használat mellett két jelentős eltérést figyelhetünk meg a BASIC editorhoz viszonyitva:

1. a PASCAL-ban alapértelmezésben csak nagybetűket tudunk használni;
   Ez alól az "á", "é", és "ó", valamint az "ú" betűk kivételek; ezek nagybetűs alakjának előállitásához az "UPPER" billentyű előzetes, vagy a SHIFT egyidejű lenyomása szükséges.
   (kisbetűket programból a megfelelő CHR(nn) függvény használatával irathatunk ki.)
   A PRIMO billentyűzetén található ékezetes betűk szimbólumokban való használata tilos!
2. néhány billentyű új jelentést kapott:
   "ö" helyett "]",
   "ő" helyett "[",
   "ü" helyett "(",
   "ű" helyett ")" jelenik meg.
   Ezen átdefiniált karakterek eredeti formájukban sehogyan sem irathatók ki a PRIMO PASCAL-ból.

A PASCAL elindulása után a PRIMO hátlapján található RESET gomb az editor melegindítására szolgál; hatására tehát bármilyen éppen futó program vagy funkció végrehajtása megszakad, a képernyő törlődik és az addig beirt szöveg törlése nélkül visszakerülünk az alapállapotba: megjelenik a prompt, és az editor új parancsot vár. Ez az állitás nem igaz, ha előzetesen letiltottuk az NMI-t, pl. az OUT(0,CHR(0)) vagy hasonló utasitással.

3. AZ EDITOR

A sor szélén megjelenő ">" jel jelzi, hogy a PASCAL editora parancsot vár. Ennek általános formája a következő:

sn1,n2,t1,t2

ahol

s: a végrehajtandó parancs;
n1 és n2: egész szám 1 és 32767 között;
erre a - zárt - sorszámintervallumra vonatkozik a parancs
t1 és t2: maximálisan 20 karakterből álló szöveg.

A parancson belül az argumentumok elválasztására vesszők szolgálnak; ez a K paranccsal más határoló karakterre cserélhető ki. A parancs előtt és az egyes tagok között tetszés szerinti számú szóköz-karakter állhat, ezeket az editor figyelmen kívül hagyja. Jóllehet nem minden parancsnál szükséges az összes argumentum jelenléte, de vannak parancsok, melyek azok nélkül nem működnek.

Előfordulnak olyan esetek, ahol egy parancs több argumentuma közül pl. elhagyjuk az első kettőt, de megtartjuk a harmadikat (pl. G parancs).

Ilyen esetben az editornak jeleznünk kell, hogy a megtalált egyetlen argumentum nem az első kíván lenni, ezért előtte annyi határoló karaktert (vesszőt) kell elhelyeznünk, amennyi az argumentumok elhagyása nélkül ott lenne. (pl. a G parancs sorszámok megadása nélkül helyesen a következő: G,,t1,t2 - értelmét lásd később.)

Az editor mindig eltárolja az utoljára kiadott parancs argumentumait: sorszámokat, szövegeket. /Ha az utolsó parancsnak valamelyik argumentuma hiányzott, az az előtti argumentuma marad tárolva, és i.t./ Ez akkor is így történik, ha a parancs valamilyen oknál fogva nem végrehajtható /hibás/, vagy ha a parancs nem használ eltárolt értéket. Amennyiben a következő parancs/ok/ban valamely argumentum hiányzik /de helyét jelzi a vessző/, annak helyére - feltéve, hogy a parancs működik eltárolt argumentummal /pl. az E, D, N parancsok nem!/ - az eltárolt érték kerül. Az alapértelmezéseket a V paranccsal irathatjuk ki.

A PASCAL elindításakor az érvényes tárolt argumentum-értékek a következők:

n1 = n2 = 10
t1 = t2 = üres string

Hibás parancs beirása esetén az editor a "Pardon ?" kérdéssel utasitja vissza azt. /Egykarakteres hibás parancsoknál ez nincs igy; ott az editor egyszerűen nem hajtja végre azt. A tárolt alapértelmezés azonban ilyenkor is megváltozhat!/

3.1 Az editor parancsai

Az editor parancsok ismertetésénél azokat az argumentumokat, melyek elhagyhatók, zárójelbe tettük.

3.1.1. Program beirása

PASCAL program szövegének beirására kétféle lehetőségünk van: vagy sorszám és egy szóköz után közvetlenül, vagy az A parancs által nyújtott automatikus sorszámozási lehetőség kihasználásával.
A PASCAL nyelv ezen változatánál sincs semmi szerepe a sorok számozásának, erre csak az editornak van szüksége egy adott sor kiválasztásához.

A parancsok a következők:

A /n1,n2/ /AUTO/

A parancs kiadása után az automatikus sorszámozó üzemmódba lépünk: az n1-edik sortól kezdve - és azt beleértve - n2 növekménnyel iródik ki a sor szélére a következő sorszám. Utána közvetlenül irható a szöveg, az üzemmód a sorszám után szükséges szóközt is kiirja. A programsor beirását a RETURN gombbal kell befejezni, ekkor a következő sor elején megjelenik az új sorszám. A beirható programsor maximális hossza 80 karakter, ezt azonban az editor nem is engedi túllépni.
Ebből az üzemmódból a "lefelé nyil" billentyű lenyomásával léphetünk ki.
Ha egy már korábban létező sorszámú sort irunk be, és ezt RETURN-nel lezárjuk, az eddigi sort ezzel - minden figyelmeztetés nélkül! - felülirjuk. / Ellentétben a BASIC editorral, ahol ilyenkor a sorszámot "*" követi./
Ha az automatikus beirás közben a sorszám eléri a 32767-et, a "le-nyil" billentyű nélkül visszalépünk az editorba.

3.1.2. Program listázása

L/n1,n2/ /LIST/

A beirt szöveget listázza az n1-edik sortól az n2-edikig. Alapértelmezésben n1 mindig 1, n2 pedig a szöveg utolsó sora, tehát nem függ a korábbi eltárolt értékektől! Igy az argumentumok nélkül megadott L parancs az egész szöveget listázza. A parancs operandusai - ha vannak - módositják a tárolt alapértelmezéseket. Erre ügyeljünk!

A képernyőn a kényelmes olvashatóság érdekében egyszerre csak bizonyos számú sor jelenik meg.
/E sorok számát a W paranccsal módosithatjuk, alapértelmezésben 15 sor./
A listázás megszakadása esetén a "le-nyil" billentyű hatására visszakerülünk az editorba, bármely más gomb lenyomása esetén a listázás a soron következő számú sor kiirásával folytatódik.

W /n1/ /WAIT/

E paranccsal állithatjuk be, hogy a listázás hány sor kiirása után álljon le; n1 értéke 1 és 255 között lehet; az alapértelmezés érvényes!

3.1.3. Szöveg szerkesztése

D n1,n2 /DELETE/

Hatására n1-től n2-ig az editor törli az összes meglévő sort. Ha csak n1-et vagy n2-t adunk meg, a funkció nem fog működni. Egy sort csak D n1, n1-gyel törölhetünk! /Vagy egyszerűbben n1, RETURN - nel./

M /n1,n2/ /MOVE/

E parancs hatására az n1-edik sor felülirja az n2-ediket, illetve ha az nem létezik, ilyen sorszámmal az editor bemásolja a szövegbe. A parancs végrehajtása az eredeti /n1-edik/ sort változatlanul hagyja.

N n1,n2 /NUMBER/

A parancs a proram átszámozására szolgál, n1 kezdettel, n2 növekménnyel. Ha az újraszámozás hatására bármelyik, vagy a soron következő sorszám elérné a 32767-et, a régi sorszámozás marad érvényben.

G/n1,n2,t1,t2/ /GLOBAL SEARCH/

t1 szöveg keresése n1 és n2 sorok közötti szövegrészben. Amelyik sorban a t1 szöveg először előfordul, az megjelenik a képernyőn, és a vezérlés Edit módba kerül /ld. később/. t2 az a szöveg, amelyre t1 kicserélhető, hiánya esetén t1 törölhető. A parancs végrehajtása minden olyan sorban megáll, ahol t1 szöveg előfordul, és egy sorszerkesztő al-utasitást vár. Ha ez S , t1-et kicseréli t2-re, ill. törli; RETURN esetén pedig továbblép t1 következő előfordulási helyére, a kijelzett sort változatlanul hagyva.

/Ha n1-et és n2-t nem adunk meg, itt is az aktuális tárolt sortartományok lesznek érvényesek a parancs vérehajtásánál./

E n1 /EDIT/

A parancs hatására az n1-edik sor Edit üzemmódba kerül. A sor számával együtt megjelenik a képernyőn, és az őt követő sor elején ismét megjelenik a kivánt sorszám, közvetlenül utána pedig a villogó kurzor. E sort javithatjuk a következőkben felsorolt utasitások segitségével. Az editor a módositásokat a tárolt

programsoron nem hajtja valóságosan végre mindaddig, mig a RETURN-t meg nem nyomtuk, igy az L, R vagy Q billentyűk segitségével visszatérhetünk az eredeti állapothoz.

A sor szerkesztésére a következő "alparancsok" szolgálnak:

SPACE billentyű hatására a szerkesztett sor következő karaktere jelenik meg. A sor vége után semmi hatása nincs.

← gomb hatására a kurzor egy karaktert visszalép, az aktuális karakter törlése nélkül.

RETURN hatására kilépünk Edit üzemmódból, végrehajtva az összes változtatást.

Q : a szerkesztés lezárása anélkül, hogy a változtatásokat végrehajttatnánk a programsoron. /QUIT/

R : a sor szerkesztésének újrakezdése; a kijelölt programsor eredeti formájában kerül ismét elénk. /REPEAT/

L : hatására a teljes sor megjelenik, és a kurzor az első pozicióba kerül; utána a szerkesztés folytatható. /LIST/

K : a kurzor által mutatott karakter törlése. /KILL/
E paranccsal több karaktert is törölhetünk, de a 2 ... karakterek törlése "vakon" történik, ezért tisztában kell lennünk a törlendő karakterek pontos számával /SPACE-ok is számitanak!/

E a kurzor által mutatott karakterrel kezdődően törlés a sor végéig. (ERASE)

G a G/n1,n2,t1,t2/ editor parancsban megadott t1 szöveg következő előfordulását keresi a szerkesztett soron belül. Ha talál ilyet a sorban, a kurzort annak első karakterére állitja, ha nem, automatikusan kilép az Edit módból /megtartván a változtatásokat/. (GET)

S a G/n1,n2,t1,t2/ parancsban megadott t2 szövegre cseréli ki t1 aktuális előfordulását, majd végrehajt egy G alparancsot, azaz vagy t1 következő előfordulási helyére áll, vagy kilép Edit módból. (SUBSTITUTE)

I áttérés "beszúrás" üzemmódba: ezt a sor bal szélén megjelenő I karakter jelzi. Mig RETURN billentyűvel ki nem lépünk ezen módból, az editor minden beirt karaktert beszúr a kurzor által mutatott pozicióba. Ez a pozició az "I" módból való kilépésig nem változik, azaz a kurzor kilépéskori pozicióját követő karakterek eltolódnak. /INSERT/
Figyelem! Ha a beszúrás folyamán a 80 karakter hosszú sorpuffer betelik, a beszúrás megáll és az adott számú sor szerkeszthetetlenné válik. Ezért ilyenkor ezt törölni kell!

A a kurzor a sor legvégére ugrik, és áttér "I" módba. /APPEND/

C áttérés "csere" üzemmódba, ezt a beszúráshoz hasonlóan a sor első poziciójában megjelenő C karakter jelzi. A RETURN-nel történő kilépésig a beirt karakter felülirja az ott lévőt, és a kurzor eggyel továbblép. /CHANGE/

A sor végére érve "C" üzemmódban nem tudunk több karaktert beirni, át kell térni "I" módba.

3.1.4. Program elmentésének lehetőségei

S n1,n2,név /SAVE/

Ezzel a paranccsal a PASCAL forrásnyelvű programot kiirhatjuk a csatlakoztatott kazettás magnóra. A parancs az n1 és n2 sorszámok közötti szöveget irja ki, /beleértve n1 és n2 sort is,/ "név" pedig a kiirandó szövegfile neve lesz, maximum 8 karakter hosszúságban. Kapcsoljuk magnónkat felvétel állásba, majd a RETURN billentyű lenyomása után a kiirás megkezdődik.

Mivel a PRIMO PASCAL külön ellenőrzés /TEST/ funkcióval nem rendelkezik, érdemes elmentés után visszaolvasni programunkat - az eredeti kitörlése nélkül. /I parancs./

F n1,n2,név /FILE/

A parancs a fordító F opciója számára ment kazettára szöveget. A parancs argumentumainak értelmezése és hatása megegyezik az S parancsnál látottakkal.

Figyelem! Az S paranccsal kivitt szövegfile nem használható az F opcióhoz, továbbá az F paranccsal kiirt file sem olvasható be az I paranccsal. /F opció magyarázatát lásd az 5. fejezetben./

I/,,név/ /INPUT/

E paranccsal olvashatunk be S paranccsal elmentett szöveget. Itt "név" a keresendő file neve, ha elmarad, az editor az első megfelelő formátumú azaz S paranccsal létrehozott file-t olvassa be.

Ha már van szöveg beirva, az editor az ily módon betöltött file-t a tárban levőhöz hozzászerkeszti, mögékapcsolja és az egészet átszámozza 1 kezdősorral és 1 növekménnyel.

/A parancsban a két vessző az itt szükségtelen n1,n2 argumentumok helyét jelöli - ld. a fejezet bevetőjét./

3.1.5. PASCAL forrás program fordítása és futtatása

C/n1/ /COMPILE/

Ezzel a paranccsal fordithatjuk PASCAL forrásnyelvű programunkat. A fordítás az n1-edik sornál kezdődik, ha pedig n1-et nem adunk meg, az első létező sornál.
A fordításról - ha azt az L compiler direktivával nem tiltjuk le - lista készül.
Ha a fordító a fordítás során szintaktikus hibát talál, * ERROR * hibajelzéssel leáll, és kiir egy nyilat, mely a hibát okozó szimbólumra mutat. A nyil mellett kiirja a hiba kódját is, mely jelentése az 1. Függelékben megtalálható. A fordítás megszakadásakor az E billentyű hatására az aktuális sor Edit üzemmódba kerül, és közvetlenül javitható, a P hatására pedig az azt megelőző sorral történik ugyanez. Mindkét gomb használatával a fordítás befejeződik, mig ha a hibajelzés után bármely más billentyűt nyomjuk meg, a fordítás tovább folytatódik /esetleg újabb hibaüzenetek kijelzésével/. Ha a "le-nyil" billentyűt nyomjuk be véletlenül, ennek hatására inverz képernyő jelenik meg, és a fordítás folytatódik. Megszakitása RESET-tel történhet. Ha a fordítás során azt észleljük, hogy a hiba nem a kurrens, vagy az azt megelőző sorban van, legcélszerűbben az E, majd a RETURN billentyűt egymást követő lenyomásával térhetünk vissza az editorhoz. Előfordulhat az is, hogy a hiba ellenére a fordítást folytatni kivánjuk /lásd a Bevezetőt/. Ez történik, ha bármely gombot benyomjuk hibajelzés esetén /kivéve az E, P, RESET és "le-nyil" gombokat/.
Sikeres fordítás után a fordító a "Run ?" kérdést szegezi nekünk, melyre ha az "Y" választ adjuk, a leforditott program azonnal elindul. Minden más válasz esetén visszakerülünk az editorba.

R /RUN/

Hatására a korábban leforditott program lefut - de csak abban az esetben, ha a fordítás óta a forrásszöveg nem lett hosszabb !
A program lefutása után visszakerülünk az editorba.
Ha a program pl. végtelen ciklusba kerül, a PASCAL rendszer elrontása nélkül kisérelhető meg megszakitása RESET-tel. Ha ez nem sikerül - pl. a RESET gombot letiltottuk - akkor sajnos csak a tápegység kikapcsolása vezet célra, de ilyenkor értelemszerűen újra kell tölteni a PASCAL - t, és a forrásprogram is elvész.

T/n1/ /TRANSLATE/

E parancs révén generálhatunk olyan kazetta file-t, amelyet később betöltve önállóan futó, működőképes programot kapunk.
A parancs kiadása után elindul a program fordítása pontosan a C parancsnál látott módon, az n1-edik sortól, ill. a legelejétől. A sikeres fordítás után a gép az "O.k. ?" kérdést teszi fel.Ha erre nem "Y"-nal válaszolunk, a vezérlés visszakerül az editorhoz. Ha azonban a kérdésre az "Y" billentyűt nyomjuk le, az S parancsnál

látott módon kezdódik a kiirás kazettára. Az ily módon keletkezett file a PRIMO LOAD parancsával olvastatható be. A program beolvasása után azonnal futni kezd.

Mivel a T parancs alkalmazása tönkreteszi a compiler gépben lévő másolatát, azt használni ilyen formában utána már nem lehet, a file kimentése után a PASCAL befejezi működését, és visszatér a BASIC rendszerbe.

Vigyázzunk tehát arra, hogy a T paranccsal csak hibátlan és többszörösen is ellenőrzött programot mentsünk el.

Rendkivül fontos, hogy az igy leforditott program használata során kerüljük a "le-nyil" billentyű használatát, mert ez a program törlődését eredményezi!

3.1.6. Egyéb parancsok

Q /QUIT/

E parancs az editor hideginditására szolgál. Hatására az editor törli az eddig beirt forrás, és tárgyprogramot.

K,,t1 /KEY/

A parancs segitségével új argumentum-elválasztó karaktert jelölhetünk ki. /Alapértelmezésben ez vesszö./A parancs kiadása után minden további utasitásban az új elválasztó karaktert kell alkalmazni, mely a t1 string első karaktere lesz. /Space nem lehet!/

V /VALIDITY/

Kiirja az aktuális sortartományt, /melyet a legutoljára kiadott parancsban megadtunk/ és a G editor-parancsban megadott t1 helyettesitendő és t2 helyettesitő szöveget, illetve ha ezek nem léteznek akkor két üres sort ir ki.

B /BOTTOM/

E parancs hatására az editor kiirja a fordító végcimét a képernyőre hexadecimálisan.

4. A PRIMO PASCAL ISMERTETÉSE

A PRIMO PASCAL - és vele a PASCAL programnyelv - ismertetésénél olyan strukturát fogunk követni, amely lehetővé teszi e leirás kézikönyvként történő felhasználását, a nyelv valamely jellemzőjének gyors visszakeresését, ezért e fejezetben a nyelv lehetőségeit kulcsszavakban és példákon át mutatjuk be. Részletes szintaktikai diagramot az olvasó a 3. Függelékben találhat.

4.1. A PASCAL program felépitése

```
PROGRAM  név;
```

A program neve az angol ABC-ben szereplő karakterekből álló tetszőleges szöveg lehet.

Deklarációk : A PASCAL nyelv egyik legszembeötlőbb sajátossága, hogy benne minden azonositót /legyen az konstans, tipus, változó, eljárás, cimke stb./ az első felhasználása előtt deklarálni kell.

A deklarációk sorrendje a következő:

```
LABEL  cimke;
```

Cimke előjel nélküli egész szám lehet, melyekre GOTO utasitásokban hivatkozhatunk. E cimkedeklarációban több cimke is felsorolható vesszővel elválasztva.

```
CONST   azonositó = érték; [ azonositó = érték; ] ...
```

A PASCAL programnyelv lehetőséget ad arra, hogy programunkban tetszőleges konstansokat definiáljunk. Ezekre a programon belül bármikor hivatkozhatunk, de új értéket többé nem kaphatnak.

```
TYPE   tipusnév  =  leirás; [ tipusnév = leirás; ....]
```

A PASCAL nyelv egyik legnagyobb előnye, hogy az előre definiált tipusok /ld.később/ segitségével tetszés szerinti új adattipusok hozhatók létre.

```
VAR   változóazonositó : tipus; [ azonositó : tipus; ... ]
```

Itt kell deklarálni a programunkban előforduló valamennyi változót. Az itt megadott tipus a valtozóra mindvégig jellemző lesz, azt nem lehet megváltoztatni.

A változók deklarációja után következik a programban hsznált eljárások és függvények leirása. /Ezeket együtt szegmenseknek szokás nevezni./ Ezek strukturájukban megegyeznek a teljes Pascal program felépitésével: van "fejlécük", melyet a PROCEDURE ill. FUNCTION szóval kell kezdeni, ezt nevük, valamint az esetleges paraméterlista követ, majd következnek az eljárás belső /lokális/ cimkéinek, konstansainak, tipusainak, változóinak és belső eljárásainak, függvényeinek deklarációi. Ezen lokális szimbólumok csak azon a szegmensen belül érvényesek, ahol deklarálva vannak, valamint az ezen szegmensen belül definiált további szegmensekben, melyekre nézve ők már "globális" szimbólumok.

Ily módon a PASCAL programban tetszőleges mélységig egymásba ágyazhatók eljárások és függvények. Mindegyikben deklarálhatók helyi tipusok, változók stb., melyek érvényessége az adott és az ennél mélyebb szintek mindegyikére kiterjed, de följebb nem. Igy a főprogramban definiált szimbólumok érvénye az egész programra, minden szegmensre kiterjed. Ha egy szegmensen belül olyan néven deklarálunk valamit, ami felsőbb szinten már szerepelt, a továbbiakban e névre való hivatkozásoknál mindig az utolsó definició lesz a mérvadó, de a szegmensből való kilépés után az eredeti változónk változás nélkül használható.

Pl.:

```
PROGRAM  PELDA;
 VAR    EGYIK, MASIK  :  REAL;       {REAL=egész szám}

 PROCEDURE ELJARAS;
   VAR    A,B   : INTEGER;          {INTEGER=egész szám}
          EGYIK : CHAR;             {CHAR= karakter}

   PROCEDURE BELSO;
    VAR  CSAKITT : BOOLEAN;         {BOOLEAN=logikai}
```

/A "{" és "}" között levő szövegek megjegyzések, ld. 4.4. fejezet./

E példában a főprogramban hivatkozhatunk az EGYIK és a MASIK REAL tipusú változókra, valamint az ELJARAS eljárásra;
az ELJARAS eljárásban hivatkozhatunk A és B INTEGER, MASIK REAL és EGYIK CHAR /!/ tipusú változókra, valamint BELSO eljárásra;
a BELSO eljárásban pedig használhatjuk a CSAKITT logikai, az EGYIK karakter, az A és B egész, valamint a MASIK valós tipusú változókat.

E deklarációk után következik a program maga, BEGIN és END kulcsszavak közé ékelve. /Kulcsszónak nevezzük a PASCAL-ban az eleve meghatározott jelentéssel biró szavakat, pl.: 'PROGRAM', 'VAR' stb. Teljes listájuk a 2. Függelékben található./
A programon belül többször szerepelhet tetszőleges mélységig további BEGIN ... END pár, melyek között további utasitások lehetnek. Logikailag és szintaktikailag az ilyen "utasitás-zárójelbe" tett utasitások egy utasitásnak számitanak.

A programblokkok és az utasitás-zárójelek csaknem korlátozás nélküli felhasználhatósága biztositja a PASCAL program nagymértékű szegmentálhatóságát, áttekinthetőségét, és azt, hogy GOTO utasitásra csak a legritkább esetben van szükség.

A PASCAL programnyelvben - eltérően pl. a BASIC-tól - a sornak semmiféle kitüntetett szerepe nincsen, itt a sor nem képvisel

információegységet a forrásprogramban. Igy egy soron belül akárhány utasitás irható, ill. egy utasitás több sorra bontható. Fontos szerepe van azonban a szóköz-karakternek annyiban, hogy egy kulcsszót nem választhatunk szét szóközökkel, és két szó között mindig kell állnia legalább egy szóköznek.

A PASCAL-ban az utasitások között a ";" karakternek kell állnia. Ez alól kivétel az END és az UNTIL /ld.később/ kulcsszavak előtt álló utasitás /ld.még az IF ... THEN ... ELSE utasitást!/

A program végét az "END." jelzi, igy ez a programon belül sehol máshol nem fordulhat elő.

A következő részben sor kerül a PRIMO PASCAL részletes ismertetésére. Ebben már fel fogjuk használni a fentiekben ismertetetteket, ezért annak az Olvasónak, aki a PASCAL nyelvvel csak most ismerkedik, ajánljuk, hogy mig azokat még nem értette, ne haladjon tovább.

/Az alapok elsajátitásához ajánljuk még az Irodalomjegyzékben felsorolt müveket./

4.2. A PRIMO PASCAL által értelmezett szimbólumok

Ebben a fejezetben ismertetjük a PRIMO PASCAL-ban használható előre definiált konstansokat, tipusokat, függvényeket és eljárásokat, valamint a programban használható utasitásokat.

Szintaxisukról részletes összefoglaló a 3. Függelékben található.

4.2.1. Konstansok

A konstansok deklarálásának módját az előző fejezetben láttuk. Konstansként definiálhatunk tetszőleges tipusú adatot.

```
pl.:    CONST  SZAM  = 12;
               BETU  = 'Q';
               SZO   = 'BETUK';
               IGAZ  = TRUE;
stb.
```

Előre definiált konstansok:

MAXINT = 32767;	A PRIMO PASCAL-ban tárolható legnagyobb egész tipusú pozitiv szám.
TRUE, FALSE	BOOLEAN tipus; TRUE = logikai igaz FALSE = logikai hamis
NIL	pointer tipus; NIL az a pointer, amely semmelyik elemre sem mutat.

4.2.2. Tipusok

A PRIMO PASCAL a következő tipusokat értelmezi:

INTEGER :

egész szám, melynek értéke -MAXINT és MAXINT közé eshet. Ha valamely számitás során ennél kisebb, ill. nagyobb értéket kapna egy egész tipusú változó, a program futása "Overflow" / =túlcsordulás / hibaüzenettel megszakad.

REAL :

valós szám. A legnagyobb valós szám 3.4E38 /=3.4 * 10 ** 38/. A valós számok kitevőjét a PRIMO PASCAL előjel + abszolutérték formában ábrázolja. Ezért az ábrázolható legkisebb abszolutértékű valós szám 1 * 10 * * -38, ami - sok más rendszertől eltérően - nem egyenlő nullával!

Figyeljünk arra is, hogy a gép számábrázolási módja miatt a valós számok között az egyenlőség csak igen ritkán teljesül. Ezért, ha két valós számot hasonlitunk össze, a 0.001-nél kisebb eltérések esetén már érdemes egyenlőnek tekinteni őket.

BOOLEAN :

logikai tipus, értéke TRUE vagy FALSE lehet. értelmezettek logikai műveletek is, melyek eredménye szintén vagy TRUE vagy FALSE lehet.

Pl.: 1 = 2 eredménye FALSE
5.1 > = 2.009 eredménye TRUE.

CHAR :

karakter tipus. Az ilyen tipusú változók értéke egyetlen ASCII karakter lehet. Karakterkonstansokat a PRIMO PASCAL-ban 'aposztrofok között' adhatunk meg. Az ilyen konstanson belüli aposztrofot két, közvetlen egymás utáni aposztroffal adhatunk meg. Üres stringet CHR(0) - vel irhatunk le.

Felsorolt tipus :

a PASCAL nyelvben egy tipus definiálása azt jelenti, hogy megadjuk azon elemek halmazát, melyeket az adott tipusba tartozó változók felvehetnek. Az eddig megismert tipusoknál a tipus neve eleve meghatározta ezt a halmazt.

A PASCAL nyelv lehetőséget ad arra, hogy a programozó maga megadja a tipusba tartozó változók által felvehető értékek halmazát, igy definiálva egy új adattipust, amit aztán tetszés szerint alkalmazhat a további deklarációkban.

Egy felsorolt tipus definiálása a tipust alkotó elemek megadásával történik. Ezen konstansok felsorolásszerű megadása miatt nevezzük az ilyen adattipusokat /t.i. ahol a tipus elemeinek halmaza egyértelműen felsorolható/ felsorolt tipusnak.

Vegyük észre, hogy előre definiált felsorolt tipus az INTEGER, CHAR és a BOOLEAN tipus is!

Tegyük fel, hogy valamely feladatunkban szükség van a hét napjainak feldolgozására. Ekkor célszerű a következő adattipus definiálása:

```
TYPE  NAPOK =  HETFO,KEDD,SZERDA,CSUTORTOK,PENTEK,SZOMBAT,VASARNAP;
```

A példából kiolvasható a felsorolt tipus deklarálásának szintaktikája is.

A tipust alkotó szimbólumok felsorolásának sorrendje igen lényeges tényező: ennek alapján értelmezhetők köztük az összehasonlitási műveletek. Minden egyes szimbólumhoz a felsorolás sorrendjében hozzárendelhető a 0,1,2... számsorozat. Valamely konstanshoz az ily módon hozzárendelt szám az adott elem sorszáma lesz. Ennek alapján nevezhetjük e tipusokat sorszámozott tipusnak.

/Sorszámozott tipussal eddig háromfélével találkoztunk: ilyen az INTEGER, a CHAR és a BOOLEAN tipus. Ezeknél a felsorolható elemek sorrend szerint:
-MAXINT, -MAXINT+1, ... ,-1,0,1,2 ... MAXINT; CHAR tipusnál az ASCII karakterkészlet sorrendben; BOOLEAN tipusnál pedig a sorrend: FALSE, TRUE ./
Igy a példánkban látott szimbólumokra pl.
HETFO < KEDD < SZERDA < VASARNAP .

Intervallum tipus :
Az előzőekben látott felsorolt tipushoz hasonlóan az intervallum tipus elemeit is a programozó maga definiálhatja. Ezen elemek azonban egy már definiált vagy eleve definiált felsorolt tipus elemeinek részhalmazaként adódnak.

Pl. ha programunkban már szerepelt a hét napjainak fentiekben látott módon történő deklarációja, a továbbiakban egy változónak a következő tipust adhatjuk:

```
VAR    MUNKANAP: HETFO .. PENTEK;
```

Igy a MUNKANAP nevű változó a következő értékek valamelyikét veheti fel: HETFO,KEDD,SZERDA,CSUTORTOK,PENTEK.

Természetesen az igy deklarált intervallumtipus is sorszámozott lesz, a szó fent leirt értelmében.

További példák intervallumtipusokra:

```
TYPE     SZAMOK = 1..23;
         BETUK  = 'A' .. 'T';
 VAR     BB : BETUK;
         CC : 'C' .. 'H';  stb.
```


Tömbök deklarálásának a PASCAL-ban bő lehetőségei vannak. Megadhatók tetszőleges tipusok tömbjei, az indexek bármely sorszámozott tipusú változók közül kerülhetnek ki; nincs korlátozás a tömbök dimenziószámára sem.

Egy tömb deklarálásának módja a következő:

```
ARRAY   index   OF tipus;
```

ahol "index" valamely sorszámozott tipus lehet, ebből több is megadható vesszővel elválasztva - igy adhatunk meg több dimenziós tömböket; "tipus" pedig bármely egyszerű vagy összetett tipus lehet. /Igy lehet pl. "tömbök tömbjét" deklarálni./

/"összetett tipusnak" nevezzük a tömb, RECORD és SET tipusokat /értelmezésük pár sorral később/, ugyanis ezek egy egész adatcsoport milyenségét definiálják szemben a többi, "egyszerű tipussal", melyek csak egyetlen adatra vonatkoznak./

A tömb valamely elemére ezek után a tömb neve után szögletes zárójelek közé irt index/ekk/el hivatkozhatunk.
/Megjegyzés: más PASCAL változatoknál találkozhattunk a PACKED ARRAY tipussal, mely tömöritett módon tárolja a tömb elemeit - ily módon memóriaterület megtakaritást érhetünk el. A PRIMO PASCAL-ban - bár a PACKED kulcsszó értelmezve van - jelentősége nincs, mert a tömöríthető adatokat eleve igy tárolja./

RECORD:
Ez a PASCAL egyik leghatékonyabb adatstrukturája. A RECORD egy különböző tipusokból összeállitott "adatcsomagot" jelent, melyre egy névvel együtt, vagy elemenként külön-külön hivatkozhatunk. Mivel tehát egy rekord is több más adatot foglal magába, hasonlóságot fedezhetünk fel közte és a tömbök között; lényeges különbség azonban egyrészt az, hogy míg a tömb elemei mindig azonos tipusúak, addig a rekord tetszőleges tipusokat foglalhat össze, másrészt pedig az, hogy a rekord elemeire azonositóval hivatkozhatunk, míg egy tömbnél ez az index alapján történik.

Például, ha valamilyen csoport tagjairól kivánunk nyilvántartást készíteni, melyben egy személynek tároljuk a nevét, születési idejét és helyét, valamint pl. egy általa befizetett pénzösszeget, akkor egy lehetséges adatstruktura a következő:
/E példában szerepel néhány olyan elem is, melynek részletes ismertetésére csak a továbbiakban kerül sor, ez azonban nem jelenthet nagyobb nehézséget a példa megértésénél./

```
    . . .
TYPE        SZUL = RECORD
                     EV, NAP     : INTEGER;
                     HONAP, HELY : ARRAY  [1..10]  OF CHAR;
                   END;
              ALAK = RECORD
                     NEVE        : ARRAY  [1..20]  OF CHAR;
                     SZULETETT   : SZUL;
                     PENZE       : INTEGER;
                   END;

VAR         EMBEREK : ARRAY  [1..100]  OF ALAK;
    . . .
```

E példából kiolvashatjuk a rekord deklarálásának szintaktikáját is:

```
rekordnév = RECORD
              rekord_elemek: tipus;
                    . . .
            END;
```

A PRIMO PASCAL-ban ha egy azonositót felhasználunk egy rekordon belül, ugyanolyan nevü szimbólum használata később nem megengedett. /Ez természetesen csak az azonos szintü deklarációkra vonatkozik - egy szinttel lejjebb, például egy eljárásban bátran használhatjuk már az ilyen neveket is!/

Az ily módon definiált rekordra a programon belül kétféleképpen hivatkozhatunk: vagy a teljes rekordra,vagy egy meghatározott elemére. /A 3. hivatkozási lehetőséget - a WITH utasitást - a 4.3. fejezetben tárgyaljuk./
Az első eset értékadó utasitásokban, vagy paraméterátadásként /ld.később/ szerepelhet, mig a második esetben az ily módon kijelölt változó éppúgy kezelhető, mint bármely "közönséges" változó.

```
Pl. EMBEREK[ 2 ]  := EMBEREK[ 4 ]  ;
    EMBEREK[ 3 ].PENZE := 123;
```

/Mint itt látható, a rekord egy elemére a rekordnév után tett pont segitségével hivatkozhatunk: rekordnév.rekord_elem /

SET:
Halmaz tipus, mely a matematikából ismert halmaz fogalom PASCAL programbeli megfelelője. Megadásának módja a következő:
azonositó = SET OF alaptipus;
ahol "alaptipus" valamely sorszámozott tipus lehet. Egy halmaznak a PRIMO PASCAL-ban maximum 256 eleme lehet. Igy létezhet pl. SET OF CHAR; mely a teljes karakterkészletet tartalmazza.

Halmazokat szögletes zárójelek közé kell zárni. Halmaz tipusú konstansok következőképpen adhatók meg: alsó-határ.. felső-határ
ahol a két határ a halmaz tipusát alkotó sorszámozott tipus két eleme.

Példák halmazok megadására:

```
[ ]              üres halmaz
[ 1,2,3 ]
[ 'A'..'Z' ]
[ 'C'..'Q','1'..'7' ]
```

```
TYPE    SZAMOK = SET OF 2.. 9;
VAR     H:SET OF 2,4,6,8,10;
```

E legutóbbi halmaz elemei lehetnek például:

```
[ 2 ]  , [ 6 ] , [ 8 ]
[  2,4 ]  , [ 4,10 ]
[  2,6,8 ] , [ 6,8,10 ]    stb.
```

A halmaznak nem lehet az elemeire külön-külön hivatkozni.

Köztük a következő müveletek értelmezhetők:

```
  =     : halmazok egyenlőségének vizsgálata;
 <>     : halmazok különbözőségének vizsgálata;
 <=     :  vizsgálat,  hogy  valamely  halmaz  részhalmaza-e
          halmazunknak;
 IN     : vizsgálat, hogy adott elem eleme-e halmazunknak;
  +     : halmazok egyesitése;
  -     : halmazok különbségének képzése;
  *     : halmazok közös részének képzése.
```

Az első négy művelet eredménye BOOLEAN, mig az utolsó háromé az adott tipusú halmazváltozó.

Pl. az előző példa H halmazára: 2 IN H eredménye TRUE
[2,4] <= H eredménye TRUE
lesz.

Pointer /mutató/ tipus:

Az eddig ismertetett tipusok nem teszik lehetővé, hogy olyan azonos tipusú adatokkal dolgozzunk programunkban, melyek mennyiségét előre nem ismerjük. Ilyen jellegű feladatot eddig csak tömbök létrehozásával oldhatunk meg, melynek nagy hátránya, hogy méretüket a deklarálásnál meg kell adnunk, igy előfordulhat, hogy a "biztonság kedvéért" jó nagyra definiált tömbünk fölöslegesen foglal el sok helyet a memóriából, vagy hogy egy újabb adat beirásával elérjük tömbünk legfelső határát, melyen változtatni nem áll módunkban.

Az ilyen problémák megoldására létezik a PASCAL nyelvben a dinamikus adatszerkezetek létrehozásának lehetősége.

A dinamikus változók legfontosabb jellemzője, hogy részükre csak akkor foglal helyet a gép a memóriában, ha arra közvetlen programutasitást adunk, /NEW eljárás./ tehát igy futás közben "születhetnek" változók, szemben a különféle tipusokba deklarált változókkal, melyek részére a PASCAL már a fordításkor helyet foglal a memóriában. Ezért ők a program elején a VAR deklarációkban nem szerepelnek. Igy a rájuk való hivatkozás nem közvetlenül,hanem egy mutató /pointer/ változón keresztül, közvetetten lehetséges. Ez a mutató egy speciális pointer tipusú változó, melynek értéke a dinamikusan létrehozott változó memóriabeli cime. E pointerváltozót kell deklarálnunk a program /vagy szegmens/ elején. A deklaráció annak leszögezése, hogy a pointer milyen tipusú adatra mutathat.

Dinamikus adatstrukturákra leggyakoribb példa a "dinamikus rekord". Az ilyen rekord elemei között általában szerepel egy /vagy több/ olyan ponter, amely e rekord tipusával azonos rekordra mutat. Igy ilyen rekordokból láncokat, fákat stb. épithetünk fel, melyek futás közben képesek méretüket változtatni. Egy ilyen struktura tárolásához két különállóan deklarált rekord-pointerre van szükség: az egyik a lánc vagy fa legelső elemére fog mutatni, a másik pedig munkaváltozóként az éppen felhasználtra. A lánc, fa vagy más dinamikus adatstrukturák létrehozásánál szükségünk van egy olyan pointer-konstransra, melynek értéke a "sehova-sem-mutat". A PASCAL nyelvben ezt a szerepet a NIL konstans tölti be.

Pointer tipusú változók között az értékadás művelet és az egyenlőség /ill. egyenlőtlenség/ - vizsgálat reláció értelmezett.

A PRIMO PASCAL sok más PASCAL verziótól eltérően nem engedi meg, hogy egy bizonyos tipusra mutató pointer deklarálása megelőzze a tipus deklarálását, igy például egy lánc megvalósitása a következő adatstrukturán keresztül történhet:

```
TYPE      LANC = RECORD
                   SZO: ARRAY  [1..10]  OF CHAR;
                   SZAM: INTEGER;
                   KOVETKEZO:   LANC;
                  {  ily módon deklarálhatunk a LANC
                       tipusú adatra mutató pointert   }
                END;
          PTR = ^LANC;
VAR       ELSO, MUT: PTR;
```

^ ASCII kódja: 94

Programunkban például a lánc második elemének számértékére a következő módon hivatkozhatunk :

```
MUT := ELSO^.KOVETKEZO;        { MUT igy a rekordra mutat }
A := MUT^.SZAM;                { ahol A  INTEGER  tipusú. }
```

"↑" jel jelenik meg, de a gépen lévő ↑ nem az!

A példákból a dinamikus változók deklarálásának és használatának szintaktikája kiolvasható.

Az eddigiekből az Olvasó láthatja, hogy a PASCAL-ban a tipusdeklaráció néha "csak" az eleganciát, a jobb áttekinthetőséget szolgálja, használata gyakran igen leegyszerüsiti, kényelmessé teszi a dolgunkat, de vannak esetek /pl. a dinamikus változók esetében/, mikor tipusok deklarálása elkerülhetetlen.

4.2.3. Változók

A PASCAL programban minden változó első felhasználását meg kell előznie a változó deklarációjának, melyben egyértelműen megadjuk annak tipusát. E tipuson a program folyamán változtatni nem lehet, és ha egy változó olyan értéket kapna, mely nem egyezik meg tipusával, a program futása hibaüzenettel megszakad. Ez alól kivételt csak az INTEGER --- REAL tipusok közti automatikus konverzió jelent, mely megenged olyan értékadást, melyben egy valós tipusú változónak egész tipusú értéket adunk. Ez forditva természetesen nem áll.

A változók deklarációját a VAR kulcsszó vezeti be. Ez után sorolandók fel a változók, és kettőspont után a tipusuk. Egy sorban több azonos tipusú változót is deklarálhatunk, tehát az alábbi részlet teljesen helyes:

```
VAR A,B,C : INTEGER;
    D     : INTEGER;
    X,Y   : REAL;
    E     : INTEGER;
```

Ez alól kivételt jelent - a standard PASCAL-tól eltérően - a tömbök deklarálása; a két tömböt a PRIMO PASCAL csak abban az esetben tekint azonos tipusúnak, ha azokat egy sorban

deklaráltuk!
Igy a következő esetben ATOMB és BTOMB azonos tipusú lesz:

```
VAR  ATOMB, BTOMB: ARRAY [1..10]  OF REAL;
                   . . .
BEGIN
                   . . .
 ATOMB:= BTOMB;   ( Igy ez az értékadás végrehajtható.)
                   . . .
END;
```

mig a következő példában a két tömb nem lesz azonos tipusú, igy például nem hajtható végre köztük közvetlen értékadás:

```
VAR   ATOMB : ARRAY [1..10]  OF REAL;
      BTOMB : ARRAY [1..10]  OF REAL;
                   . . .
BEGIN
                   . . .

 ATOMB:=BTOMB;     Ez esetben forditáskor * ERROR * 10
                    hibaüzenetet kapunk.
                   . . .
END;
```

Több dimenziós tömb deklarálásának formailag két lehetősége van:

```
C:ARRAY [ 1..8] OF ARRAY [ 1..8]  OF CHAR  megegyezik a
C:ARRAY [ 1..8,1..8 ] OF CHAR              két dimenziós tömbbel.
```

A PRIMO PASCAL lehetővé teszi, hogy - attól függetlenül, hogy a deklarálás milyen módon történt - programon belül mindkét módon hivatkozhatunk a tömb valamely elemére. Igy C[1][1] és C [1,1] ugyanazon mátrix (1,1) indexü elemére utal.

Eltérést jelent a standard PASCAL-tól, hogyha egy rekordon belül deklaráltunk egy változót, ugyanilyen nevü változó újbóli deklarálása - a rekordon kivül is - hibához vezet.

A PASCAL programnyelv olyan nevek használatát engedi meg, melyek az angol ABC betüjével kezdődnek, ezt pedig további ilyen betük vagy számjegyek követhetik.
Ez a név lehet egy változó, konstans, tipus, eljárás, függvény vagy a program neve.

A PRIMO PASCAL a nevek első 10 karakterét veszi figyelembe.

Meg kell emliteni, hogy az alábbi tipus bizonyos szempontból kitüntetett:

```
ARRAY [1..N] OF CHAR;
```

ahol N > 1. Ez az ún. string /karakterlánc/ tipus, bár ezt a szót magát a fordító nem értelmezi. A karakteres konstansok is ilyen string tipusúak, N hosszal, és állhatnak olyan értékadás jobb oldalán, ahol a bal oldalon iyen tipusú karaktertömb található. /Itt olyan konstansokról lehet szó, melyeket a program CONST részében deklaráltunk./ Minden azonos hosszúságú string változó azonos tipusúnak számit, tehát - eltérően egyéb tömböktől - köztük értékadás akkor is végrehajtható, ha nem egy bekezdésben

deklaráltuk őket.

KISS GYÖRGY
2500 ESZTERGOM
ARANY JÁNOS U.4.
TEL: (33)314 603

4.2.4. Eljárások és függvények

A PASCAL program felépitését ismertető fejezetben az olvasó már megismerkedett az eljárások és függvények fogalmával, a globális és lokális változók jelentésével, deklarálásuk módjával. Most a fogalmak pontos ismertetésére, néhány eddig nem emlitett bevezetésére, valamint a PRIMO PASCAL előre definiált eljárásainak és függvényeinek leirására kerül sor.

Összetettebb feladatok megoldásánál elkerülhetetlenné válik programunk "alprogramokra" történő lebontása. Ezen alprogramok a feladat egy-egy részfeladatát oldhatják meg, vagy lehetnek olyan általánosan használt programrészletek, melyekre több helyen is szükség van: ilyenkor alprogramunk szerepe megegyezik a programozásban általánosan használt szubrutinok feladatával.

Egy program alprogramokra /szegmensekre/ bontása abban az esetben is indokolt, ha bizonyos szegmenseket csak egyetlen helyről hivunk meg, tehát nem szubrutin szerepe van; az alprogramok alkalmazása nagymértékben növeli programunk áttekinthetőségét, lehetővé teszi, hogy egyes részfeladatokat önállóan, egészként oldjunk meg, a program könnyen bővithetővé válik stb. Nagyobb PASCAL rendszereknél óriási előnyt jelenthet az, hogy a szegmentált programok lefordítását részletekben is végezhetjük, valamint egyes részek különböző programozóknak önállóan kiadhatók.

A PASCAL-ban az alprogramok kétfélék lehetnek: eljárások és függvények. A kettő között a különbség az, hogy mig egy eljárás neve kizárólag annak meghivására, aktivizálására szolgál, addig a függvény neve emellett értéket is képvisel: a függvény értékét, melynek tipusát a függvény dekarációjában meg kell adnunk, konkrét értéke pedig a függvényen belül, annak meghivásakor kerül kiszámitásra.

Mindkét tipusú alprogramnak szüksége lehet olyan adatokra, melyeket a szegmens meghivásakor a hivő környezet aktuális változóinak értéke fog meghatározni. Ezen adatok felhasználására kétféle lehetőség van. Az egyik a globális változók használata, tehát azon változóké, melyeket a szegmensen belül nem deklarálunk újra - hiszen egy szegmens korlátozás nélkül használhat magasabb szinten deklarált adatokat. Ez a módszer azonban igen gyakran hátrányos lehet: ha pl. egy függvényt sok helyen hivunk meg, körülményes volna a függvény bemeneti paraméterét /vagy paramétereit/ a hivás előtt mindig egy bizonyos változóba beirni. E nehézség kiküszöbölésére hozták létre a PASCAL programozási nyelvben a paraméterek átadásának lehetőségét.

Ez formailag a következőt jelenti: egy eljárás vagy függvény deklarálásakor annak fejlécében az u.n. formális paraméterlistában fel kell tüntetnünk mindazon változókat, tipusuk megjelölésével, melyek a szegmens aktivizálásakor átadásra kerülnek. E paraméterek az alprogramon belül lokális

jelentéssel birnak, ebből a szempontból mindaz érvényes rájuk nézve, amit a lokális változókról korábban elmondtunk. értékük az eljárás vagy függvény meghivásakor azonossá válik a meghivásakor a formális paraméterlistába irt változók aktuális értékével, tehát mintha ezek a változók "átadnák" értéküket a szegmens paramétereinek.

Pl.: . . .

```
PROCEDURE  ALPGM (2  VAR  A,B : INTEGER;  X : REAL;
                     VAR  C : ARRAY  [1.. 8]  OF CHAR ) ;
  BEGIN
    X := A/2;
    A := A*2;
    B := A+2;
    C := 'SZOVEG  ';
  END;

    . . .

BEGIN            (   főprogram  )

   . . .

   Q := 1;  W := 99;  Y := 22.222;
   SZO := 'ABCDEFGH';
(   Q  és  W  egész, Y  REAL, SZO pedig ARRAY  1..8  OF CHAR
                tipusú változó )

   ALPGM ( Q,W,Y,SZO ) ;

        (      és itt     Q = 2
                          W = 3
                          Y = 22.222  maradt !
                          SZO = 'SZOVEG  '  )

END.
```

Ebben a példában az ALPGM eljárás formális paraméterlistájában felsorolt változókat két csoportra bonthatjuk: az A, B és C változókat a VAR kulcsszó előzi meg, mig ez az X előtt nincs. Azok a paraméterek, melyek előtt nincs VAR, az eljárásnak csak bemenő paraméterei lesznek, az eljárásból visszatérve értékük változatlan marad /ez példánkban az Y változó/. Ha formális paraméterlistában egy változó előtt szerepel a VAR kulcsszó, e változó a szegmensnek kimeneti változója is lesz, tehát értéke a szegmensből kilépve az ott kapott érték lesz /Q, V, SZO változók/.

Példánkból is látható, hogy egy szegmens átadott paramétereinek azonositása a formális paraméterlistában elfoglalt helyük alapján történik. Tehát pl. a fenti eljárás ALPGM (Q,Y,W,SZO) ; formában történő hivása szintaktikailag hibás, hiszen az eljárás deklarálásakor a formális paraméterlista második helyére egy egész 'tipusú változót irtunk, itt pedig másodikként egy REAL számot adnánk át.

Már a fentiekből is következik, hogy a szegmens deklarálásakor

irt paraméterekkel megegyező számú és tipusú adatot kell az alprogram minden hivásakor átadni.

Mint már emlitettük, a függvények csak annyiban különböznek az eljárásoktól, hogy ott a függvény neve képvisel egy értéket. Ezért a függvény tipusát deklarációjában meg kell adni, és nevének a függvény belsejében egy értékadás bal oldalán szerepelnie kell. A paraméterátadásokra vonatkozó szabályok természetesen a függvényekre is érvényesek, igy VAR kulcsszóval átadott paraméterek segitségével lehetőség van több kimeneti változóval rendelkező függvények definiálására.

Lássunk egy példát:

```
    . . .

  FUNCTION  KOB (  X:REAL ) :REAL;
  (     a függvény bemeneti  paramétere  és
        értéke egyaránt valós szám lesz  )
    BEGIN
     KOB:= X * X * X;
    END;

        . . .

  BEGIN          (     főprogram    )

        . . .

    Y:=KOB( 2.14 );      (     Y REAL változó  )

        . . .

  END.
```

Vigyázzunk arra, hogy ha a függvényen belül a függvény neve nem csak értékadó utasitás bal oldalán szerepel, hanem pl. jobb oldalán is, vagy egy IF utasitásban stb., az a függvény rekurziv meghivását fogja jelenteni!

Ügyeljünk viszont arra, hogy a függvény nevének - a szegmens jellegéből következően - legalább egy értékadó utasitás bal oldalán szerepelnie kell. A függvény "szabályos" meghivása pedig az jelenti, hogy a függvény neve a szegmensen kivül szerepel valamely kifejezésben, természetesen teljes paraméterlistájával.

Előfordulhat, hogy programunk struktúrája olyan bonyolult, hogy elkerülhetetlen pl. egy eljárás hivása olyan helyen, ami előtt deklarációjára még nem kerülhetett sor.
Erre a megoldást a FORWARD direktiva jelenti.

Pl.:

```
        . . .
    PROCEDURE ELJ1 ( A,B : BOOLEAN) ;  FORWARD;
(   jelzés,  hogy az ELJ1 eljárás definiálására a  későbbiekben
       kerül   sor.  Itt  kell  azonban  megadni  a  formális
       paraméterlistát.  )
        . . .
    PROCEDURE ELJ2;
     BEGIN
        . . .
       ELJ1  TRUE,FALSE  ;
        . . .
     END;
        . . .
    PROCEDURE ELJ1;
    ( itt következik az eljárás tulajdonképpeni leirása.
       Itt már nem szabad kiirni a formális paraméterlistát )
      BEGIN
        . . .
       ELJ2;
        . . .
      END;
                              stb.
```

Lényeges, hogy az ily módon történő "előre deklarálásnak" és a szegmens tényleges deklarálásának ugyanabban a blokkban kell elhelyezkednie.

4.2.4.1. Előre definiált eljárások

a/ Ki- és beviteli eljárások

WRITE ,WRITELN (p1, p2, ..., pn)

Ezen eljárások segitségével irathatunk ki változókat, konstansokat, szövegeket a képernyőre. A WRITE eljárás sorban kiirja az alább részletezett formátukban az adatokat. A WRITELN eljárás ettől csak annyiban különbözik, hogy a kiirás után egy sort emel. /Igy egy WRITELN; utasitás egy soremelést eredményez./

Ezeknek az eljárásoknak a paraméterei nem kötött számuak, és nem kötött tipusúak abban az értelemben, ahogy a programozó által definiálható szegmensek paraméterei kötöttek. /Ebből a szempontból tehát a WRITE eljárás egy igen különleges eljárás./ WRITELN (A,B,C,) tehát azonos hatású a WRITE (A) ; WRITE (B) ;WRITE (C) ; WRITELN; utasitásokkal, ezért a továbbiakban csak a WRITE (p1) utasitással foglalkozunk.

Itt p1 a következő formájú lehet:

```
        e
vagy ,  e : e1
vagy    e : e1 : e2
vagy    e : e1 : H
```

ahol e a kiirandó kifejezés, mely lehet karakter, egész, valós vagy logikai kifejezés, de lehet karakterlánc ill. szöveg

aposztrofok között.
A további tagok nem kötelezőek, segitségükkel a kiirás formátuma határozható meg.
e1 a kiirandó szám vagy szöveg teljes szélessége lesz;
/amennyiben e1 nagyobb, mint a szám jegyeinek száma, az első számjegy előtt a szükséges mennyiségű szóköz karakter jelenik meg. Egész számok kiirásánál, ha e1 kisebb a számjegyek számánál, csak az utolsó e1 számjegyet irja ki a program, REAL számok esetén pedig ha e1 kisebb a szám ábrázolásához minimálisan szükséges helynél, a kiirás az alapértelmezés szerinti, 12 karakteres formában történik./
e2 jelenléte esetén a valós tipusú szám fixpontos formátumban kerül kiirásra, ahol e2 a tizedes jegyek száma lesz;
/e2 = 0 esetben a kiirt szám a REAL érték egész része lesz./
a negyedik tipusú formátumot egész számok kiirásánál használhatjuk. Ilyenkor a szám hexadecimálisan kerül kiirásra. A hexa szám szélessége ez esetben is e1 lesz.

```
Pl.:    WRITE  1490:4:H       ==>   05D2
        WRITE 0.32E2:6:2      ==>    32.00
        WRITE  42.1234:8:3    ==>    42.123
        WRITE  -2.4563E14:9   ==>   -2.45E+14
```

Lehetőség van karakterek, stringek /= ARRAY[1..N]OF CHAR /, valamint BOOLEAN értékek kiirására is; ez esetben is megadható az e1 hosszúság, mely ha túl nagy, bevezető space-ek kerülnek a szöveg elé, ha túl kicsi, a kiirás alapértelmezésben történik.

```
Pl.:    WRITE ( ' SZOVEG')        ==>   SZOVEG
        WRITE ( ' SZOVEG':10 )    ==>       SZOVEG
        WRITE ( 1 < 2 )           ==>  TRUE
```

Vegyük észre, hogy más programozási nyelvekkel ellentétben a PRIMO PASCAL karakteres kiirásnál is jobbra igazitást végez.

Ha a CHR(16) - vagy editorból a CTR P - karaktert iratjuk ki, annak speciális szerepe van: az azután következő minden kiirás nyomtatóra történik a következő ilyen karakter kiirásáig.
Ha a géphez nincs nyomtató csatlakoztatva, a gép végtelen ciklusba esik, melyből csak a RESET gombbal vehetjük ki.
/Ld. még a fordító P opcióját is!/

PAGE

Ezen eljárás a képernyő törlését eredményezi.

READ , READLN (p1,p2,...,pn)

Ezen eljárások segitségével olvashatunk be programunkba adatokat - a PRIMO PASCAL-ban a billentyűzetről.
A READ, READLN eljárások paramétereivel kapcsolatban érvényesek azok a megjegyzések, melyeket a WRITE eljárásokkal kapcsolatban tettünk, tehát itt sincs korlát a paraméterek számára, nincs szigorúan meghatározott tipusuk, igy ők "különleges" eljárásnak tekintendők. Mivel a READLN (p1,p2,...,pn) ; utasitás egyenértékű a

```
BEGIN
   READ ( p1 ) ;
   READ ( p2 ) ;
   . . .
   READ ( pn ) ;
   READLN;
END;
```

utasitásokkal, /ill. a READ (p1.. pn) ; ugyanigy, csak READLN nélkül/, a tárgyalásnál csak a READ (p1) esetet részletezzük.

Minden karakter /betű, szám, karakterlánc/ melyet a program futása közben /egy READLN utasitás kérésére/ bebillentyűzünk, a PASCAL rendszer ún. puffer-területére kerül. A READ eljárás mindig e pufferből olvas ki paraméter/ei/nek megfelelő számú és tipusú adatot. A puffer aktuális karakterére, melyet tehát a legközelebbi READ eljárás be fog olvasni, egy pointer mutat. Ha ez az aktuális karakter egy RETURN /sor vége/, akkor az EOLN előre definiált függvény értéke TRUE, ellenkező esetben FALSE. Mivel a puffer a program elindulásakor egyetlen CHR(13) / RETURN / karaktert tartalmaz, e függvény értéke ilyenkor TRUE, és minden READ eljárás eredménye CHR(0), üres string lesz.

A READLN eljárás "kiüriti" a puffert, a karakter-pointert az első pozicióra állitja, és lehetővé teszi, hogy a felhasználó a billentyűzetről irjon a pufferbe. Hatására az EOLN függvény értéke mindig FALSE lesz. A READLN eljárást követő első READ tehát a READLN után a pufferbe beirt karaktereket fogja beolvasni.

A puffer ily módon "részletekben" is kiolvasható a READ eljárással. Ha pl. egy READLN-ra öt karaktert billentyűztünk be, és a következő READ eljárás paramétere egy CHAR tipusú változó, ez alkalommal a legelső beirt karakter lesz a változó értéke, a pointer pedig a második karakterre fog mutatni. Az ezt követő READ eljárással tehát beolvashatjuk a további négy karaktert, vagy egy READLN-nal újrakezdhetjük a puffer feltöltését.

Ha a READ eljárás közben a pointer egy RETURN karakterre bukkan, az EOLN függvény értéke TRUE lesz, és a további beolvasandó karakterek helyén CHR(0)-t kapunk.

A fentiek értelmében tehát egy olyan program, mely visszairja a bebillentyűzött karaktereket, a következőképpen nézhet ki:

```
PROGRAM  ECHO;
VAR   C : CHAR;
BEGIN
  REPEAT
    READLN; READ(C);
    WRITE(C)
  UNTIL  C = CHR(0);
  WRITELN;
END.
```

Összefoglalva:

- Ha az eljárás paramétere CHAR tipusú, az eljárás egyszerűen beolvassa a a puffer következő karakterét, és ez lesz a paraméter értéke,

- Ha a paraméter tipusa string, az eljárás mindaddig olvas be karaktereket a pufferből, amig a beolvasott karakterek száma meg nem egyezik a tömb deklarációjában megadottal. Ha a beolvasás

közben egy sor-vége karaktert talál, a string minden további eleme CHR(0) karakterrel lesz feltöltve.

- Egész vagy valós tipusú paraméter esetén minden sor-vége és szóköz karaktert figyelmen kivül hagy, és az ezután következő számot beolvassa a pufferből; egész szám esetén az első nem-szám karakterig, valós esetén a számokat követő pontot vagy E /=exponens/ karaktert is beolvassa, de ha ezeket nem követi további számjegy, a futás "Number expected" hibaüzenettel leáll. Ugyanez történik akkor is, ha a beirt szám túl kicsi, vagy túl nagy, vagy ha az első karakter nem szám, vagy '+' vagy '-' előjel.

b/ Egyéb előre definiált eljárások

NEW (p)

Az eljárás paramétere egy mutató tipusú változó. Az eljárás a p pointer tipusának megfelelő dinamikus változót hoz létre a memóriában. P erre a memóriaterületre /erre a változóra/ fog mutatni.
Az eljárás számára közömbös p bemenő értéke.

MARK (p)

Ez az eljárás a dinamikus változók tárolására használt "halom" helyzetét menti el a p pointer változóba. A RELEASE eljárás az ily módon beállitott p értékig fogja felszabaditani a halom helyzetét.

1988.08.24.

RELEASE (P)

Hatására a változóhalom helyzete arra az értékre áll vissza, amit a P az utolsó MARK eljárás során kapott. Igy tehát az azóta NEW eljárással létrehozott dinamikus változók törlődnek!

1988.12.06.

INLINE (p1,p2,...,pn)

Ez az eljárás lehetővé teszi, hogy PASCAL programunkba Z80 gépi kódú utasitásokat illesszünk. P1,p2... a beirandó utasitások kódjai; lehetnek decimális vagy hexadecimális számok - ez utóbbiak megkülönböztetésére, mint mindenütt a PRIMO PASCAL-ban a # karakter szolgál, melyet a szám elé kell irni. A forditóprogram a program lefordításakor az INLINE utasitásban irt Z80 utasitáskódokat a program aktuális helyére szerkeszti be.

Pl. INLINE (#C3, 00, 00);

Vissza a BASIC-hoz

Ezen eljárás a programba a JP 0 Z80 utasitást teszi be, melyet végrehajtva visszakerülünk a PRIMO BASIC rendszerébe.

USER (P)

Ez az eljárás a P cimen kezdődő gép kódú program hivását eredményezi. A rutinnak RET utasitással kell végződnie, és nem ronthatja el az IX regiszter tartalmát. Mivel a PRIMO PASCAL az egész számokat kettes komplemens kódban ábrázolja, a 32767-nél nagyobb cimeket negativ számokkal érhetjük el. E kényelmetlenség kikerülése végett is ajánlatos a P cim hexadecimális értékének használata.

HALT

Ezen eljárás megszakitja a program futását a következő üzenettel: "Halt at PC=nnnn", ahol nnnn az a hexadecimális memóriacim, ahol a futás megszakadt.
Ezt az eljárást elsősorban a programok hibakeresésésnél használhatjuk.

POKE (cim, kifejezés)

Ez az eljárás hasonlit a BASIC azonos nevű utasitására: cim címmel kezdődően leteszi a kifejezés értékét a PRIMO memóriájába. Az eljárásban cim egy egész tipusú kifejezés, kifejezés pedig akármilyen tipusú lehet, SET kivételével. Igy lehetséges ASCII karakterek közvetlen beirása a memóriába, és ha "kifejezés" tipusa pl. REAL, az eltárolt szám annak 4 byte-on ábrázolt értéke lesz.

TOUT (név, kezdőcim, hossz)

Ez az eljárás módot ad adatok kazettán való tárolására. Itt a "név" tipusa ARRAY [1..8] OF CHAR , ez lesz a kiirt file neve. Az eljárás "hossz" darab byte-ot visz ki "kezdőcim"-től számitva. E két paraméternek egész tipusúnak kell lenni.
Az eljárást változók elmentésére az ADDR és a SIZE függvények segitségével használhatjuk: pl. az X változó elmentése "IKSZ" néven a következőképpen történhet:

```
TOUT ( 'IKSZ    ', ADDR( X ), SIZE (X) )  ;
```

TIN (név, kezdőcim)

Ezen eljárás a TOUT eljárással elmentett változók beolvasására szolgál. "név" és "kezdőcim" tipusa megegyezik az ott látottakéval.
Tehát például az előző pontban elmentett változó beolvasása a következőképpen történik:

```
TIN ( 'IKSZ    ', ADDR( X )) ;
```

OUT (P,C)

Ez az utasitás lehetővé teszi a Z80 portok közvetlen használatát. Hatására a C karakter a P egész kifejezés által meghatározott portra kerül. / P < 65536 /

4.2.4.2. Előre definiált függvények

a/ Beolvasási függvények

EOLN

E logikai függvény értéke TRUE, ha a következő beolvasandó karakter egy "sor-vége" karakter / RETURN, = CHR (13) /, minden más esetben FALSE. /Ld. READ, READLN eljárások./

INCH

E függvény hivása a billentyűzet letapogatását eredményezi; ha van lenyomva billentyű, a függvény értéke a beirt karakter, ha nincs, értéke CHR (0) lesz.

b/Adattipus átalakító függvények

TRUNC (X)

A függvény eredménye az egész vagy valós tipusú X paraméter csonkitott egész része, természetesen INTEGER tipusú.

Igy TRUNC (3.345) = 3
TRUNC (-1.234) = -1

ROUND (X)

Itt X szintén REAL vagy INTEGER lehet. A függvény értéke X "kerekitett" értéke: az az egész szám, amely X-hez "legközelebb" áll.

Pl. ROUND (3.4) = 3
, ROUND (3.5) = 4
ROUND (-3.5) = -3
ROUND (-3.6) = -4

ENTIER (X)

Ennek a függvénynek az argumentuma valós vagy egész tipusú lehet, értéke pedig az a legnagyobb egész szám, mely kisebb vagy egyenlő X-szel. E függvény hatása pozitiv számokra megegyezik a TRUNC függvénnyel.

Pl. ENTIER (8.9) = 8
ENTIER (-13.2)= -14

ORD (X)

E függvény paramétere valamely sorszámozott tipusú adat lehet. A függvény X értékének sorszámát adja meg, mely tipusának deklarációjakor adódik ki.
Igy természetesen egy egész szám ORD értéke maga a szám, továbbá például
ORD ('A') = 65 ASCII karakter értéke

CHR (X)

A függvény az egész tipusú X-nek megfelelő ASCII kódértékű karaktert adja vissza. /A függvény az X változó első byte-ját veszi figyelembe /MOD 256 értékét/. Negativ X esetén a figyelembe vett érték a kettes komplemens szám első byte-ja lesz./

c/ Aritmetikai függvények

A következőkben ismertetendő függvények mindegyikének paramétere egész vagy valós lehet, értékük pedig mindenkor REAL tipusú.

ABS (X)

értéke X szám abszolút értéke.

SQR (X)

értéke X szám négyzete lesz.

SQRT (X)

A függvény X négyzetgyökét adja eredményül.

SIN (X)

Eredménye X radiánban vett értékének sinusa.

COS (X)

Eredménye X radiánban vett értékének cosinusa.

TAN (X)

Eredménye X radiánban vett értékének tangense.

ARCTAN (X)

E függvény annak a szögnek radiánban vett értékét adja eredményül, melynek tangense X.

EXP (X)

A függvény eredménye a természetes "e" szám X-edik hatványa.

LN (X)

Ez a függvény X természetes /e/ alapú logaritmusát adja meg.

FRAC (X)

E függvény X törtrészét adja eredményül, pontosabban mindig X-ENTIER(X)-et. Igy pl. FRAC (-1.6) értéke 0.4 lesz!!!

d/ Egyéb függvények

ODD (X)

A logikai függvény értéke TRUE, ha az X egész tipusú kifejezés értéke páratlan, FALSE, ha páros.

RÁNDOM

E függvény egy 0 és 255 közötti álvéletlenszámot ad eredményül.

SUCC (X)

X valamely sorszámozott tipusú lehet. A függvény eredménye az X-et követő sorszámú elem lesz.

Pl. SUCC (1) = 2
SUCC ('A') = 'B'
SUCC (*MAXINT*)= -32768 kettes komplemens kód!

PRED (X)

E függvény az X sorszámozott tipusú adatot megelőző sorszámú elemet adja eredményül.

Pl. PRED (2) = 1
PRED ('B')= 'A'
PRED (-MAXINT)= -32768 !

INP (X)

E függvény az OUT eljárás "párja"; segitségével a PRIMO Z80 portjairól tudunk beolvasni. A függvény értéke az X-edik portról beolvasott byte CHR értéke.

ADDR (X)

A függvény értéke X változó memóriabeli cime lesz.

SIZE (X)

Ez a függvény az X változó által elfoglalt memóriaterület nagyságát adja meg byte-okban.

PEEK (X, tipus)

Ez a függvény a POKE eljárás forditottja; eredménye egy "tipus" tipusú érték, mely az X ewgész számú kifejezés által mutatott memóriacimről kerül kiolvasásra.

Tehát pl. PEEK (nnnn,INTEGER) eredménye egy egész szám, mely nnnn /hexadecimális/ cimü memória-byte-on tárolt érték;
PEEK (mmmm,ARRAY [1..8] OF CHAR) eredménye egy 8 értékü karaktertömb, melyben az egyes karakterek az mmmm cimtől kezdve sorban kiolvasott értékeknek megfelelő ASCII karakterek.

4.3. A programban használható utasitások, műveletek

4.3.1. Operátorok a PASCAL-ban

operátor	jelentése	operandusok tipusa	eredmény tipusa
a/ Aritmetikai operátorok			
+	összeadás	egész, valós	egész, valós
-	kivonás	"	"
*	szorzás	"	"
/	osztás	"	valós
DIV	egész osztás	egész	egész
MOD	maradékképzés	"	"
b/ Logikai operátorok			
NOT	negálás	logikai	logikai
OR	logikai vagy	"	"
AND	logikai és	"	"
c/ Halmaz operátorok			
+	halmaz egyesités	azonos halmaztipusok	
-	halmaz különbség	"	
*	metszet	"	
d/ Relációs operátorok			
=	egyenlőség	bármely elemi, mutató, halmaz, rekord	logikai
<>	egyenlőtlenség	"	"
<	kisebb	elemi, string	"
>	nagyobb	"	"
<=	kisebb vagy egyenlő	elemi, string halmaz	"
>=	nagyobb vagy egyenlő	"	"
IN	halmaz eleme	bal: sorszámozott jobb: ennek halmaza	"

4.3.2. A PASCAL utasitásai

E fejezetben ismertetjük a PASCAL programban használható utasitásokat. Ezek egy részét - vagy hozzájuk igen hasonlókat - megtaláljuk szinte minden magasszintű programnyelvben, de vannak olyan utasitások is, melyek a PASCAL nyelv sajátosságaira épülnek, és ezért a legtöbb programozási nyelvben hozzá hasonlót sem találhatunk.

a/ Az értékadó utasitás

Formája: változó := kifejezés;

Hatására "változó" értéke - függetlenül előző értékétől - a "kifejezés" kiértékeléséből adódó eredmény lesz. Ezen eredmény tipusának meg kell egyeznie "változó" tipusával. Ez alól két kivétel van: ha "változó" REAL tipusú "kifejezés" lehet INTEGER is, ilyenkor a tipuskonverzió automatikusan megtörténik; másrészt az egyik ilyen tipus lehet a másikból származtatott intervallumtipus.

A "kifejezés" kiértékelésének sorrendje a következő:

1. függvények kiértékelése
2. * , /, DIV, MOD
3. +, -
4. AND, OR
5. < , > , <=, >=, =, <>

Azonos precedenciájú műveletek között a kiértékelés balról jobbra történik. Zárójelek tetszőleges mélységig alkalmazhatók, rájuk a matematika szabályai érvényesek.

b/ IF utasitás

Programunkban mindig előfordulnak olyan helyzetek, mikor valamilyen esemény bekövetkezésekor, ill. be nem következésekor más-más utasitást, vagy programrészletet kell végrehajtani.

Az ilyen feltételek vizsgálata az IF utasitáson keresztül történhet. Ennek formája a következő:

IF feltétel THEN utasitás1 ELSE utasitás2;

ahol "feltétel" egy olyan kifejezés, mely logikai tipusú eredményt ad - lehet tehát relációs és logikai operátorokkal felépitett bonyolult kifejezés, de lehet pl. egyetlen BOOLEAN tipusú változó, vagy konstans is. "Utasitás" pedig egyetlen utasitás abban az értelemben, ahogyan az a szintaktikai összefoglalóban látszik; tehát a BEGIN END "utasitás-zárójelek" között tetszőleges számú további utasitás, teljes programrészlet szerepelhet.

Az utasitás végrehajtása a következő: ha a feltétel eredménye

TRUE /igaz/, a THEN /=akkor/ kulcsszó, ha FALSE /hamis/, az ELSE /=különben/ után következő utasitás kerül végrehajtásra.

Az ELSE ág nem kötelező, ha nincs ott, a THEN-t követő utasitás után kell a pontosvesszőt kitenni. Ebben az esetben, ha a feltétel hamis, a program végrehajtása a következő utasitáson folytatódik.
Vigyázzunk arra, hogy az ELSE kulcsszó előtt soha nem állhat pontosvessző!

```
Pl.:    IF  A < B  THEN  A := B   ELSE  B := A;

        IF  ( A  <= B )  AND ( B > 2 )  OR (  A = 0 )   THEN
                IF  A = 1  THEN   BEGIN
                                   WRITE ( ' IGEN ' ) ;
                                   A := 999
                                  END
                           ELSE   BEGIN
                                    WRITE ( ' NEM ' ) ;
                                    A := 888
                                  END
                                     ELSE  WRITE (' HIBA ');
```

E második példában olyan esetet láthatunk, mikor a THEN-t követő utasitás maga is egy IF utasitás. Ez természetesen megengedett, de figyelnünk kell arra, hogy az egyes ELSE ágak melyik IF-hez kapcsolódnak. Ennek eldöntése a "belülről kifelé" tipusú felépités végigkövetése alapján történhet: hasonlóan az "egymásba ágyazott" BEGIN - END párok, vagy ciklusok mintájára itt is meg kell találnunk a héjszerűen egymásra ágyazódó THEN - ELSE párokat. E szabályt vegyük figyelembe programunk irásánál.

Példánkban, ha pl. a második IF utasitásban nem volna szükség ELSE ágra, a probléma csak a következőképpen volna megoldható:

```
IF ( A  <= B )  AND  ( B > 2 )  OR ( A = 0 )  THEN
        BEGIN
          IF  A = 1  THEN  BEGIN
                             WRITE ( ' IGEN ' );
                             A := 999
                           END
        END
                                        ELSE  WRITE ( ' HIBA  ' ) ;
```

Ily módon /a belső IF utasitás "zárójelbe tételével"/, elkerültük, hogy a fordító az ELSE ágat a második IF-hez párositsa, ami logikailag hibát okozna programunkban.

c/ Ciklusutasitások

Egy programban igen gyakran előadódnak olyan részfeladatok, amikor ugyanazt a programrészletet többször egymás után kell végrehajtani. Az ilyen programrészleteket ciklusoknak nevezzük.
Egy ciklusnak szüksége van néhány olyan utasitásra, vagy szimbólumra, amelyek az adott programrészlet, a ciklusmag megfelelő számú megismétlését biztositják.
Ilyen utasitás a PASCAL nyelvben három féle van.

1. FOR ciklusváltozó := kezdőérték TO /vagy DOWNTO/ végérték
DO utasitás ;

Ezt a fajta ciklusszervezést olyan esetekben használjuk, amikor a ciklusmag ismétlésének száma a ciklus megkezdésekor pontosan ismert, vagy kiszámitható.

Az utasitásban szereplő "utasitás" egyetlen PASCAL utasitás - úgy, ahogy azt már láttuk. A "ciklusváltozó" egy sorszámozott tipusú változó lehet, melynek szerepe számlálni a ciklusmag végrehajtásait. "Kezdőérték" és "végérték" két olyan kifejezés lehet, melynek tipusa megegyezik a "ciklusváltozó" tipusával.

Az utasitás végrehajtása a következő: a gép kiértékeli a két kifejezést, majd a "ciklusváltozó" "kezdőérték" értékével végrehajtja a ciklusmagot abban az esetben, ha értéke még nem érte el "végértéket", tehát TO esetén kisebb vagy egyenlő, DOWNTO esetén pedig nagyobb vagy egyenlő annál. Az utasitás végrehajtása után "ciklusváltozó" értéke TO esetén SUCC ciklusváltozó , DOWNTO esetén pedig PRED ciklusváltozó lesz - tehát ha pl. a ciklusváltozó tipusa INTEGER, eggyel megnöveli, ill. lecsökkenti azt. Ezután következik a vizsgálat /a ciklusváltozó elérte-e a végértéket/, majd a ciklusmag újbóli végrehajtása, ill. kilépés a ciklusból.

Másképpen fogalmazva a két határt megadó kifejezések kiértékeléséből egy diszkrét elemekből álló intervallum adódik ki. Ezen az intervallum elemein halad végig növekvő, ill. csökkenő sorrendben "ciklusváltozó" értéke. A ciklusmag annyiszor kerül végrehajtásra, ahány eleme ennek az intervallumnak van, tyehát ha TO esetén a felső határt kisebbnek vagy egyenlőnek adjuk meg az alsó határral, a ciklusmagot egyszer sem hajtja végre.

A ciklusváltozót a ciklusmagon belül természetesen felhasználhatjuk, de új értéket nem kaphat!

Pl.: FOR A := 1 TO 10 DO WRITE (A) ;
kiirja a természetes szamokat 1-től 10-ig

FOR A := 10 DOWNTO 1 DO WRITE (A) ;
ez csökkenő sorrendben teszi ugyanezt

```
FOR KAR := 'A' TO 'Z' DO BEGIN
                           WRITE ( KAR );
                           WRITELN ( ORD( KAR ))
                         END;
```

kiirja az angol ABC betűit, feltüntetve ASCII kódjukat is

2. REPEAT utasitás UNTIL feltétel ;

Ez az utasitás olyan ciklusok szervezésére ad lehetőséget, ahol a ciklusból való kilépés egy olyan feltételhez van kötve, amely a ciklusmag bárhányadik végrehajtása során beállhat.

Végrehajtása a következő: a program végrehajtása a REPEAT

kulcsszó elérésekor tovább folytatódik, rákerül a ciklusmagra, mely pontosvesszőkkel elválasztott tetszőleges számú utasitásból állhat. Igy a ciklusmag legalább egyszer feltétlenül végrehajtódik!

Végrehajtva a ciklusmagot, a gép kiértékeli az UNTIL utasitás után álló logikai eredményt adó "feltételt". Ha ennek eredménye FALSE, a futás a REPEAT kulcsszót követő ciklusmag első utasitásán folytatódik, ha pedig a kifejezés TRUE eredményt ad, kilép a ciklusból.

Pl.:
```
REPEAT
  READLN ( KAR );    ( KAR CHAR tipusú változó )
  WRITE ( KAR )
UNTIL  KAR  = 'Q';
```

E példa addig olvas be a billentyűzetről és ir ki a képernyőre karaktereket, mig Q-t nem irunk be.

Megjegyezzük, hogy a REPEAT ... UNTIL utasitáspár a BEGIN... END-hez hasonlóan utasitás-zárójelként is funkcionál; a teljes ciklus egy utasitásnak számit.

3. WHILE feltétel DO utasitás;

Ez a ciklusutasitás elsősorban abban különbözik a REPEAT ... UNTIL ciklustól, hogy itt a feltétel az utasitás végrehajtása előtt kerül kiértékelésre, igy előállhat olyan eset, amikor a ciklusmagot egyszer sem hajtjuk végre.

Ez utasitás hatására a gép olyan esetben hajtja végre a ciklusmagot, ha a "feltétel" igaz. Tehát ha a WHILE utasitásra történő ráfutáskor a feltétel FALSE eredményt ad, a végrehajtás azonnal a következő utasitáson folytatódik.

Pl.:
```
HIBA := A > 100;          ( HIBA BOOLEAN tipusú )
WHILE  NOT HIBA  DO  BEGIN
                       A := A * 2;
                       WRITE( A );
                       HIBA := A <= MAXINT DIV 2
                     END;
```

érdemes "algoritmikus formában" összefoglalni a PASCAL ciklusutasitásait.

FOR ciklus:
1. ciklusváltozó := kezdőérték
2. IF ciklusváltozó végérték THEN vége
3. utasitás
4. ciklusváltozó := SUCC ciklusváltozó
5. GO TO 2

REPEAT ... UNTIL ciklus:
1. utasitás
2. IF feltétel = TRUE THEN vége
3. GO TO 1

WHILE ciklus:

```
1. IF  feltétel = FALSE  THEN  vége
2. utasitás
3. GO TO 1
```

d/ A CASE utasitás

Ez az utasitás lehetővé teszi, hogy az IF utasitástól eltérően ne csak egy feltétel igaz vagy hamis volta alapján hozzunk döntéseket, hanem egy sorszámozott tipusú változó értékétől függően más és más utasitásokat hajtsunk végre.

Pl. ha SZAM és S INTEGER tipusú változók, a

```
CASE  SZAM MOD 12  OF
 1    : S := 1;
 2    : S := -2;
 3..5, 7..9 : S :=2;
 6,10,11 : S := 3
END;
```

utasitás SZAM értékétől függetlenül ad S-nek különböző értékeket.

E példáról az utasitás használatával kapcsolatos legtöbb szabály leolvasható; az OF kulcsszó után sorolandók fel a CASE után álló kifejezés lehetséges értékei, majd kettőspont után az azon érték esetén elvégzendő utasitás, mely természetesen lehet BEGIN ... END, vagy REPEAT ... UNTIL jellegű utasitás is.

Látható az is, hogy a konstansok felsorolásánál egy sorban /egy utasitás előtt/ megadhatunk több értéket is vesszővel elválasztva, valamint megadható intervallum is.

A fenti példánk azonban hibás. Ugyanis nem soroltuk fel a kifejezés által felvehető valamennyi értéket, hisz SZAM MOD 12 adhat 0 eredményt is. Ebben az esetben programunk itt hibaüzenettel leállna. Egy CASE utasitásban fel kell készülni minden eshetőségre.

Részben ennek megkönnyitésére szolgál az a lehetőség, hogy a CASE utasitás utolsó "konstansaként" irható az ELSE kulcsszó. Ha ezt tettük, és a kifejezés olyan értéket vesz fel, mely a felsorolt utasitások között nem szerepel, akkor az ELSE után következő utasitás kerül végrehajtásra, mely természetesen lehet egyetlen pontosvesszős, azaz üres utasitás is.

Lássunk egy példát egy ilyen CASE utasitásra:

```
CASE  KAR  OF          (   KAR  CHAR tipusú  )
  'A'     : S := 1;
  'B','C' : S := 2;
  'D'     : BEGIN
              S := 3;
              M := 12
            END;
  'E'     : S := 4
  ELSE  S := 0
  END;
```

A más PASCAL verziókhoz szokott Olvasónak feltűnhet az itt látható utasitás szokatlan szintaktikája: ha CASE utasitásunkban ELSE ág szerepel, az ELSE-t megelőző utasitás végére nem szabad pontosvesszőt tenni, /ELSE előtt ; nem állhat !/, az ELSE szó és az utasitás között nem áll kettőspont, valamint ebben az esetben a CASE utasitást nem szabad END; -el lezárni, ami pedig ELSE hiánya esetén kötelező.

e/ WITH utasitás

Ez az utasitás a RECORD tipusú adatok kezelését könnyiti meg. Mint tudjuk, egy rekord valamelyik elemére a következőképpen hivatkozhatunk: REKORDNéV.ELEM . Abban az esetben, ha egy programrészleten belül egy adott rekord elemeire sokszor hivatkozunk, a rekord nevének állandó ismételgetése kényelmetlenné válik.

Ennek egyszerüsitését szolgálja ez a parancs, melynek formája a következő:

```
WITH változó DO utasitás;
```

ahol "változó" egy rekord tipusú változó neve, "utasitás" pedig a már sok helyen látott egyetlen utasitás, mely természetesen ez esetben is lehet BEGIN és END közötti programrészlet. Ezen az utasitáson belül pedig a "változó" rekord elemeire úgy hivatkozhatunk, mint önálló változókra, tehát nincs szükség a ponttal való minősitésre.

Pl.:

```
      . . .
      VAR   REK : RECORD
                    EGYIK : INTEGER;
                    MASIK : 1..100;
                    ALFA  : ARRAY [ 1..8 ] OF CHAR
                  END;
            HIBA : BOOLEAN;
      BEGIN
            . . .
        WITH REK DO IF  NOT HIBA  THEN
                                  BEGIN
                                    EGYIK := EGYIK+1;
                                    MASIK := EGYIK MOD 50;
                                    ALFA:='O.K.     ';
                                    HIBA:=TRUE
                                  END;
            . . .
      END.
```

A WITH utasitás utáni "utasitásban" természetesen nem csak olyan változók szerepelhetnek, amelyek a rekord elemei: lehetnek ott egyéb "normális" változók, de akár egy másik rekord is - ez esetben az arra való hivatkozásnál a teljes szintaktika szerint kell eljárni.

Megjegyzések:

1. Más PASCAL verzióknál ügyelni kell arra, hogy amennyiben létezik olyan nevü "egyszerü"változónk is, amilyen névvel rekord egyik elemét is illettük, akkor a WITH utasitáson belül az "egyszerü" változó elérhetetlen - hisz nevével ez esetben a rekord ilyen elemét jelöltük ki. Mivel a PRIMO PASCAL ilyen névegyezést nem enged meg, ez a probléma nem jelentkezik.

2. Szintén eltérést jelent a standard PASCAL-tól, hogy a PRIMO PASCAL-nál két WITH utasitás "egymásba ágyazása" hibához vezet.

f/ A GOTO utasitás

Mint már a bevezetőben emlitettük, a PASCAL programozási nyelv segitségével a problémák általában megoldhatók GOTO utasitások nélkül, ami szép, elegáns és hatékony programfelépitést eredményez. Általánosságban is ajánlhatjuk, hogy a programozó kerülje PASCAL programjaiban GOTO utasitásokat, mert azok nagymértékben rontják a program áttekinthetőségét, és gyakran nehezen felderithető hibákhoz vezethetnek.

Ennek ellenére előfordulhat olyan eset, amikor feltétlen vezérlésátadás /GOTO utasitás/ nélkül nem, vagy csak nagyon körülményesen oldható meg valamely probléma.

Az ilyen esetekben használhatjuk a GOTO cimke formájú utasitást. Ez feltétlen vezérlésátadást eredményez a "cimkét" követő utasitásra. Az utasitásban szereplő "cimkét" a program legelején, LABEL kulcsszóval kezdődően deklarálni kell. /Természetesen itt kell deklarálni a programban előforduló valamennyi cimkét. A cimke csak előjel nélküli egész szám lehet; ily módon mindig elkülönithető más szimbólumoktól./

A cimke a programban - deklarálásán kivül - egyetlen egyszer szerepelhet, itt egy kettőspont kell, hogy kövesse. Az ilyen cimkére vonatkozó GOTO utasitás végrehajtása hatására e cimke /és kettőspont/ utáni első utasitással fog folytatódni a program végrehajtása. Egy cimkére, természetesen, több GOTO utasitás is hivatkozhat.

GOTO utasitással lehetőség van egy szegmenssel belőli ugrásra, de lehet szegmensből ugrani főprogramba, sőt egyik szegmensből egy másikba is. Ez utóbbi esetet azonban csak roppantul indokolt esetben használjuk; nehezen felderithető bonyodalmak származhatnak belőle.

Pl.:

```
        . . .
LABEL   999;
VAR     ADAT : ARRAY [ 1..20 ] OF INTEGER;
        HIBA : BOOLEAN;
    . . .
BEGIN
    . . .
  FOR  I := 1  TO 20 DO  BEGIN
                           READLN (ADAT[I]);
                           HIBA := ADAT[ I ] > 100;
                           IF  HIBA  THEN  GOTO 999
                         END;
    . . .
 999:
END.
```

E példánkban hibás adat beirása esetén a program azonnal leáll.

Összefoglalásul azt ajánlhatjuk, hogy a Programozó többször is gondolja át a feladatot, és GOTO utasitást csak akkor alkalmazzon, ha ezt ez után is elkerülhetetlennek tartja.

4.4. Megjegyzések /commentek/

Programunk irása közben hasznos segitséget jelenthet szóbeli megjegyzések elhelyezése az utasitások között. Erre a PRIMO PASCAL a következő lehetőséget biztositja: a programnak bármely olyan helyén elhelyezhető megjegyzés, ahol szóköz szerepelhet /tehát bármely két szimbólum között!/. A comment szövegét vagy kapcsos zárójelek /"ů",itt."ü" billentyűk/, vagy "(*" és "*)" karakterpárok közé kell tenni. A fordító az ilyen karakterek között levő szöveget teljes egészében figyelmen kivül hagyja, a programra semmi hatása nincsen. /Ez alól kivételt jelent az az eset, ha a comment legelső karaktere egy "$" jel: ilyenkor a következő pozicióban opciók adhatók meg a fordítónak - ld. a következő fejezet./

KISS GYÖRGY
2500 ESZTERGOM
ARANY JÁNOS U.4.
TEL: (33)314 603

5. A FORDITO OPCIOI

A PRIMO PASCAL alapértelmezésben a C editor parancs hatására a beirt /vagy előzőleg kazettáról betöltött/ szöveget leforditja gépi kódra oly módon, hogy a leforditott szövegről a képernyőre folyamatos listát készit. Az egyes PASCAL sorok előtt feltünteti /hexadecimálisan/ azt a memóriacimet, ahol a sornak megfelelő gépi kódú utasitások kezdődnek.
Ettől a formától, valamint néhány forditási szolgáltatás eredeti formájától eltérhetünk: ezt forditási opciókkal érhetjük el.

A forditási opciókat comment elején adhatjuk meg; ha a commentet nyitó kapcsos zárójelet /vagy (* jelet/ közvetlenül egy $ karakter követ, az ez után álló szöveget a fordító opciónak tekint.
Ez a következők valamelyike lehet:

L alapértelmezés: L+

Ezzel az opcióval engedélyezhetjük, ill. tilthatjuk le a programszöveg listázását forditáskor.

L+ esetén a programról folyamatosan lista készül;
L- esetén e helytől kezdve csak a szintaktikailag hibás sorokról készül lista ;

O alapértelmezés: O+

O+ esetén a program egész számok összeadása és kivonásakor is végez túlcsordulás-vizsgálatot.
O- megadása esetén ez nem történik meg.

/Szorzásnál, osztásnál, valamint valós számokkal végzett bármely művelet esetén a PASCAL mindenképpen végez ilyen vizsgálatot!/

S alapértelmezés: S+

S+ esetén a PASCAL minden eljárás és függvény hivása előtt megvizsgálja, hogy ebben a szegmensben előállhat-e túlcsordulás a stackben /egymásba ér a visszatérési cimewk tárolására szolgáló "verem" /stack/ és a dinamikus változókat tároló "halom" /heap// pl. a dinamikus változók nagy tömege miatt. Ha ez előfordul, a futás "Out of RAM at PC=nnnn" hibajelzéssel megáll.
S- esetén ez a vizsgálat nem történik meg.

A alapértelmezés: A+

A+ megadása esetén a PASCAL tömbök használatánál

megvizsgálja, hogy az aktuális index beleesik-e a deklarált határok közé, és ha nem, a futás hibaüzenettel leáll.

A- esetén letiltjuk ezt a vizsgálatot.

I alapértelmezés: I-

A PRIMO PASCAL által használt számábrázolási mód mellett olyan számok összehasonlitásakor, melyek 32767-nél nagyobb értékkel különböznek egymástól, hibás eredmény keletkezhet.

Az I+ opció megadása biztositja, hogy az eredmény ilyen esetekben is helyes lesz.

I- esetén az összehasonlitások eredményének vizsgálata elmarad.

O-, S-, A- vagy I- opció esetén a program futása gyorsabb lesz, de egyes esetekben előfordulhat a program, rosszabb esetben az egész PASCAL rendszer "elszállása". Az alapértelmezések az "átlagos" esetekben biztositják a program biztonságos, kielégitő gyorsaságú futását.

F

Ez az opció lehetőséget ad arra, hogy úgy forditsunk le PASCAL programot, hogy csak egészen minimális helyet foglaljon el a memóriából.

Az opció betűjele után egy 8 karakterből álló file névnek kell következnie /nem aposztrofok között!/. Ha a név ennél rövidebb lenne, szóközökkel ki kell azt egészitenünk 8 karakterre! Ez a file név lesz a forditáshoz felhasznált file neve. Ezt a file-t korábban az editor F parancsával kellett kiiratnunk a magnóra.

Ezen opció megadása esetén a fordító elkezdi keresni a szalagon a megadott nevű file-t. Ha megtalálta beolvas abból néhány sort, azokat leforditja, majd újabbakat keres. Ezt teszi a file végéig, majd folytatja a forditást a begépelt szöveggel. Itt a program folytatása, befejezése, vagy akár további F opció következhet. Igy lehetőség van előre megirt programrészletek, eljárások több programban történő felhasználására anélkül, hogy azokat újra és újra be kellene gépelni, vagy beszerkeszteni a forrásnyelvű programba.

A működés meggyorsitása érdekében szalagról forditás közben a PASCAL nem listáz.

1. FÜGGELÉK: HIBAÜZENETEK

A fordító által jelzett hibakódok jelentései a következők:

1. Túl nagy szám
2. ; hiányzik
3. Deklarálatlan ayonositó
4. Hiányzó azonositó
5. Konstans deklarációban = és nem := jelet kell használni
6. = hiányzik
7. Kifejezés nem kezdődhet ezzel az azonositóval
8. := hiányzik
9.) hiányzik
10. Helytelen tipus
11. . hiányzik
12. Hiányzó tényező
13. Hiányzó konstans
14. Ez az azonositó nem konstans
15. THEN hiányzik
16. DO hiányzik
17. TO vagy DOWNTO hiányzik
18. (hiányzik
19. Ez a kifejezés nem létezik
20. OF hiányzik
21. , hiányzik
22. ; hiányzik
23. PROGRAM hiányzik
24. Egy változó hiányzik
25. BEGIN hiányzik
26. A READ eljárás hivásánál egy változó hiányzik
27. Ilyen tipusú kifejezéseket nem lehet összehasonlitani
28. Az összes tipusnak INTEGER-nek vagy REAL-nek kell lenni
29. Ilyen tipusú változót nem lehet beolvasni
30. Ez az azonositó nem tipusnév
31. A valós számban hiányzik a kitevő
32. Skalár kifejezés /nem numerikus/ szükséges
33. Üres string nem megengedett, helyette használjunk CHR (0) -t
34. [hiányzik
35.] hiányzik
36. Tömbindex csak sorszámozott tipus lehet
37. .. hiányzik
38.] vagy , hiányzik a tömbdeklarációból
39. Az alsó határ kisebb a felsőnél
40. A halmaz több mint 256 elemből áll
41. A függvény eredményének azonositó tipusnak kell lennie
42. , vagy] hiányzik a halmazban
43. .. vagy , vagy] hiányzik a halmazban
44. A paraméter tipusa csak azonositó lehet
45. Egy nem értékadó utasitás első tényezője nem lehet üres halmaz
46. Skalár tipus - REAL -t is beleértve - hiányzik
47. Skalár tipus - REAL -t nem beleértve - hiányzik
48. Nem kompatibilis halmazok
49. < és > nem használható halmazok összehasonlitásánál
50. FORWARD , LABEL , CONST , VAR , TYPE vagy BEGIN

hiányzik
51. Hexadecimális szám hiányzik
52. Halmazokat nem lehet POKE eljárásban használni
53. Túl nagy tömb
54. END vagy ; hiányzik a RECORD definició végén
55. Mezőlista azonositója hiányzik
56. WITH után nem áll változó
57. WITH utasitásban a változónak RECORD tipusúnak kell lennie
58. A mezőlista-változónak nincs kapcsolata a WITH utasitással
59. Előjel nélküli egész szám hiányzik LABEL után
60. Előjel nélküli egész szám hiányzik GOTO után
61. Ez a cimke rossz szinten van
62. Nem deklarált cimke
63. SIZE paramétere csak változó lehet
64. Egyenlőségvizsgálat csak pointer tipusok között végezhető
67. A két kettőspontot és egy egész tipusú számot használó WRITE eljárás paramétereit si:s2:H formában kell megadni
68. String nem tartalmazhat sorvége karaktert
69. NEW , MARK és RELEASE eljárások paramétere csak pointer tipusó változó lehet
70. Az ADDR függvény paramétere csak változó lehet

Futás közben előadódó hibajelzések:

Halt	= Megállitás
Overflow	= Túlcsordulás
Out of RAM	= Ehhez kicsi a memória
/ by zero	= Nullával osztás
Index too low	= Túl alacsony index
Index too high	= Túl magas index
Math op. error	= Matematikai hiba
Number too large	= Túl nagy szám
Number expected	= Szám hiányzik
Line too long	= Túl hosszú sor
Exponent expected	= Kitevő hiányzik
ROM Call error	= Rendszer hivásakor bekövetkező hiba

KISS GYÖRGY
2500 ESZTERGOM
ARANY JÁNOS U.4.
TEL: (33)314 603

2. FÜGGELÉK : LEFOGLALT SZAVAK, ELŐRE DEFINIÁLT AZONOSITÓK

Foglalt szavak:

AND	ARRAY	BEGIN	CASE	CONST	DIV
DO	DOWNTO	ELSE	END	FORWARD	FUNCTION
GOTO	IF	IN	LABEL	MOD	NOT
OF	OR	PROCEDURE	PROGRAM	RECORD	REPEAT
SET	THEN	TO	TYPE	UNTIL	VAR
WHILE	WITH				

Előre definiált azonositók:

CONST	MAXINT			
	NIL			
TYPE	BOOLEAN	CHAR	INTEGER	REAL
PROCEDURE	WRITE	WRITELN	READ	READLN
	PAGE	HALT	USER	OUT
	NEW	MARK	RELEASE	TIN
	TOUT	INLINE	POKE	
FUNCTION	ABS	SQR	SQRT	ODD
	RANDOM	ORD	SUCC	PRED
	INCH	EOLN	PEEK	CHR
	ENTIER	ROUND	TRUNC	FRAC
	SIN	COS	TAN	ARCTAN
	EXP	LN	ADDR	SIZE
	INP			

SET, RECORD

4. FÜGGELÉK: PASCAL PÉLDAPROGRAMOK

Az itt közölt programok kifejezetten demonstrációs céllal készültek. Ezért természetesen gyakran találhat az Olvasó olyan részleteket, melyeket egyszerűbben, vagy elegánsabban tudna megoldani - kérjük, tegye ezt.

Különösen a PASCAL nyelvvel most ismerkedők számára ajánljuk e programok végigböngészését - rajtuk keresztül sok homályos pont érthetővé válhat.

a/ Másodfokú egyenlet gyökeit számító program

```
PROGRAM  GYOKOK;

VAR   M     : REAL;          (  munkaváltozó )
      D     : REAL;          (  a gyök második tagja lesz )
      C     : CHAR;
      KOMPL : BOOLEAN;       (  TRUE,  ha  az  egyenlet  gyökei
                                   komplexek )
      EHATO : ARRAY [ 'A'..'C' ] OF REAL;

 PROCEDURE  DISZKR ( A, B, C : REAL );
   VAR   M   : REAL;
   BEGIN
     M := SQR ( B ) - 4 * A * C;
     KOMPL := M < 0;
     IF  KOMPL  THEN  M := -M;
     D := SQRT ( M   / ( 2 * A ));
   END;

 BEGIN          (  a főprogram kezdődik )
   REPEAT
     PAGE;
     FOR  C := 'A'  TO  'C'  DO  BEGIN
                                   WRITE ( C, '=' );
    ( együtthatók beolvasása )     READLN ( EHATO[ C ]);
                                   WRITELN
                                 END;
    IF  EHATO ['A'] = 0  THEN  WRITELN('X=',-EHATO['C']
                                          / EHATO ['B'] )
                               (  nem másodfokú egyenlet )
                          ELSE
          BEGIN
            DISZKR ( EHATO['A'], EHATO['B'], EHATO['C']);
            M := -EHATO['B'] / ( 2 * EHATO['A']);
            IF  KOMPL  THEN  BEGIN
                       WRITELN( 'X1 =', M, ' + I  ', D );
                       WRITELN( 'X2 =', M, ' - I  ', D )
                           END
                     ELSE  BEGIN
                       WRITELN( 'X1 =', M + D );
                       WRITELN( 'X2 =', M - D )
                           END
          END;
```

```
    C := CHR( 0 );
    WHILE  C = CHR( 0 )  DO  C := INCH
          (  vár, mig egy billentyût lenyomunk )
  UNTIL  C = 'Q'
( 'Q' billentyû hatására vége, egyébként új adatokat kér )
 END.
```

b/ A következő példa ízelítőt ad a dinamikus változók használatából, bemutatva egy példát a rekurzív eljáráshivásra is.

```
PROGRAM  LANC;

 TYPE   SZO     = ARRAY [ 1..8 ] OF CHAR;
        ADAT    = RECORD
                    SZAM : INTEGER;
                    NEV  : SZÓ;
                    KOV  : ^ADAT;
                  END;
         MUTATO = ^ADAT;

 VAR    ELSO    : MUTATO;
        C       : CHAR;


 PROCEDURE  KARBE;     {  egy karakter beolvasása }
   BEGIN
     REPEAT
       C := INCH;
     UNTIL  C <> CHR ( 0 );
   END;


 PROCEDURE  BEIR ( EL : BOOLEAN ) ;
                {  EL = TRUE, ha az első adat
                      beírása  következik       }
    VAR  P,P1 : MUTATO;

    FUNCTION  UTOLSO : MUTATO;
      VAR  P : MUTATO;
      BEGIN
        P := ELSO;
        WHILE  P^.KOV <> NIL  DO  P := P^.KOV;
        UTOLSO := P;
      END;

    BEGIN                  { BEIR eljárás kezdődik }
      IF  EL  THEN  BEGIN
                     NEW ( ELSO ) ;
                     P1 := ELSO;
                    END
              ELSE  BEGIN
                      P := UTOLSO;
                      NEW ( P1 );
                      P^.KOV := P1;
                    END;
      WITH  P1^ DO  BEGIN
                      KOV := NIL;
                      PAGE;
           WRITE (' SZAM ? '); READLN; READ ( SZAM );
           WRITE (' NEV  ? '); READLN; READ ( NEV );
                    END;
    END;                   { BEIR eljárás vége }
```

```
  PROCEDURE  KIIR1;   (  eljárás az adatok beirási
                         sorrendben történő kiirására )
    VAR  P : MUTATO;
    BEGIN
      PAGE;
      P := ELSO;
      WHILE  P <> NIL  DO  BEGIN
                        WRITELN ( P^.SZAM,'    ',P^.NEV );
                        P := P^.KOV
                             END;
      KARBE;          ( várakozás, hogy a kiirtakat el
                              lehessen olvasni  )
    END;


  PROCEDURE  KIIR2;   (  ez az eljárás irja ki a láncot
                         forditott sorrendben, a REK névü
                         rekurzivan hivogatott eljáráson
                                  keresztül )
    PROCEDURE  REK ( P : MUTATO );
      BEGIN
        IF  P^.KOV <> NIL  THEN  REK ( P^.KOV );
        WRITELN ( P^.SZAM,'    ',P^.NEV )
      END;

    BEGIN            (  KIIR2 eljárás )
      PAGE;
      REK ( ELSO );
    END;


BEGIN      (   főprogram   )
  BEIR  TRUE  ;        (  a legelső elemet be kell irni )
  REPEAT
    PAGE;

    WRITELN; WRITELN;
    WRITELN ('     B = UJ ELEM BEIRASA ');
    WRITELN ('     E = ADATOK KIIRASA BEIRAS SORRENDJÉBEN ');
    WRITELN ('     F = ADATOK KIIRASA FORDITOTT SORRENDBEN ');
    WRITELN ('     V = VÉGE ');
    KARBE;

    CASE  C  OF
      'B' : BEIR ( FALSE );  ( további elemek beirása )
      'E' : KIIR1;
      'F' : KIIR2
      ELSE ;           ( más billentyű esetén nem csinál
                                  semmit   )
    UNTIL  C = 'V';
  END.
```