# LISP
# programozási segédlet

NOVOTRADE RT.

# TARTALOM



IS-LISP az Enterprise 1. verziójára



Progamming: M. Richer

Manual Writing R. Hurley , J. Hurley,
R. D. Virgo

# ELŐSZÓ

Bár a LISP (LIST PROCESSING) már kb. húsz éve van forgalomban, még mindig nagyon népszerű a "Mesterséges Intelligencia" és a "Szakértői rendszerek" területén, ahol ideálisnak tekinthető rugalmassága és kiterjeszthetősége miatt.

Ennek a kézikönyvnek nem célja, hogy a nyelvről minden információt tartalmazó enciklopédia legyen, de leírja az IS-LISP implementációját az Enterprise számítógépen és közelebbi betekintést ad arról, hogyan történik a kommunikáció a felhasználóval és az EXOS operációs rendszerrel. A kezdőnek rövid bevezetést ad a nyelvbe, de feltételezi, hogy az olvasó ismeri a gép működését és vannak - legalábbis korlátozott - ismeretei a Basic nyelvről.

Mivel a LISP interaktív nyelv, amely minden függvényt úgy és akkor kiértékel, ahogy az a számítógépbe került, könnyű rajta kísérletezni a különböző függvényekkel. Tanácsoljuk az olvasónak, hogy próbálja ki a kézikönyvben található példákat, amikor csak lehetséges.

**Könyvek a LISP-ről**

Sok jó könyv íródott a LISP-ben való programozásról. Különösen ajánljuk a következőket:

"LISP on the BBC microcomputer" (A. Norman, G. Cattell) jó bevezetést ad a LISP nyelvű programozáshoz. A BBC-LISP-et írja le, amelyik igen közel áll az IS-LISP-hez.

"LISP" (P. Winston, B. Horn) a LISP egy másik dialektusát írja le, de ismertet néhány újszerű eszközt is, pl. a makrókat.

# AZ ELINDULÁS

Hogy elkezdjünk programozni IS-LISP-ben, tegyük be a cartridge-et az Enterprise mikroszámítógép bal oldalán levő portba és kapcsoljuk be a gépet. Ezután nyomjuk meg kétszer a Reset gombot. A hidegindítás szokásos folyamata megy végbe, a gép ellenőrzi a memória-szekciókat, és az alább látható üzenet jelenik meg a képernyőn.

Most elkezdhetjük a programozást. Az utasítások ugyanúgy vihetők be a billentyűzetről, mint az IS-BASIC használatakor. Minden kommunikáció a felhasználó és a LISP interpreter között az "editor"-on keresztül történik, ami azt jelenti, hogy a kurzor-billentyűk és a joystick használhatók a kurzor mozgatására a képernyőn, hogy szöveget vigyünk be, javítsunk, vagy töröljünk.

Amikor egy programsort helyesen beírtunk és megnyomtuk a return gombot, ennek hatására az "EVAL" program működésbe lép, és - feltéve, hogy a sor szintaktikusan helyes - kiértékelődik, a kiértékelés eredménye pedig megjelenik a képernyőn.

Például, két szám összeadására beírhatjuk: (PLUS 3 4), és a számítógép kiadja: 7.

Ezen a ponton fontos, hogy emlékeztessünk rá: minden LISP programsor azonnal ellenőrződik és kiértékelődik, rögtön a bevitel és a return gomb megnyomása után. Ez teljesen más, mint amivel BASIC környezetben dolgozva találkozunk, ahol egy szekció vagy akár az egész program bekerül a gépbe, és csak ezután történik kísérlet a végrehajtásra.

Ezt észben tartva elkezdhet kísérletezni és programozni a LISP nyelven. A következő néhány fejezet jó kalauz néhány ismert LISP függvény használatához.

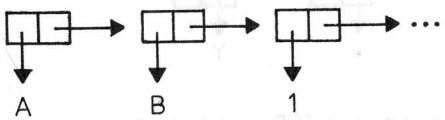
2. Fejezet

# BEVEZETÉS A LISP-BE

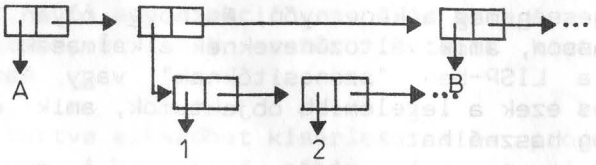
A LISP-ben, mint minden magasszintű nyelvben, a programozó elvárja, hogy szöveges üzeneteket jelentethessen meg a képernyőn, és hogy olyan szavakat használhasson, amik változóneveknek alkalmasak. Az ilyen szavakat a LISP-ben "azonosítóknak" vagy "atomoknak" hívjuk, és ezek a legelemibb objektumok, amik egy LISP programban használhatók.

Az azonosító olyan objektum, amely karakterekből épül fel, (pl. Z, DOG, T.NIL), és a fenti célok bármelyikére használható. Atom minden azonosító és minden szám (pl. 7, -24). Ezen atomok használatával fel tudjuk építeni az adatstruktúrát, ami a LISP-nek a nevét adja, azaz a listákat.

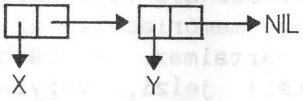
A lista adattételek egy gyüjteménye, ezek mindegyikének van egy rákövetkezője, kivéve az utolsót. Hasznos, ha ezt úgy tekintjük, mint pointerek egy rendszerét, amelyek mentén a felhasználó végighaladhat egy listán, és minden pointer egy memóriacellához van rendelve, egy cella két pointert tartalmaz. A baloldali pointer az atomot (vagy listát) jelzi, vagyis az adatot, a jobboldali pedig oda mutat, ahol a lista további része kezdődik. Az (A B 1...) listára ezt kapjuk:



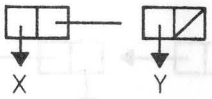
Ahogy az előbb jeleztük, egy listában tárolt adattételek maguk is lehetnek listák, így az (A(1 2...) B...) lista részletesebben:




Már említettük, hogy egy lista minden tételének van egy rákövetkezője, kivéve az utolsót, amelyre a LISP-nek egy speciális, "NIL" nevű azonosítót tart fenn, az üres lista, a () reprezentálására. Így az (X Y) listát reprezentáló teljes diagram:




amelyet megegyezés szerint így ábrázolunk:




Ez mind elmélet és a LISP működésének alapját képezi, de most kezdjünk el ismerkedni ezzel a nyelvvel.

## Aritmetika

A LISP-ben az aritmetika azonosítók használatával működik, nem pedig a hagyományos infix operátorokkal, mint "+" és "x". Számok (atomok) összeadására a PLUS parancsot használjuk.
A 7+5 összeadás elvégzésére (PLUS 7 5)-öt kell írnunk, erre a 12 választ kapjuk, és hogy a 7+5+3+1 értéket kiszámítsuk, a helyes LISP parancs

(PLUS 7 5 3 1),

amelyre a 16 választ kapjuk.

Az összeadás inverz művelete, a kivonás, a DIFFERENCE parancs segítségével történik, amely az első argumentumból levonja a másodikat, azaz 7-5 így irandó be:

(DIFFERENCE 7 5),

amire 2 lesz az output.

Az infix jelölés mellett, a "-" jelnek egy második jelentése is van, a negatív előjel. A LISP egy másik azonosítóval, a MINUS-szal elkerüli ezt a kettősséget. Így 7-5, ami a 7+(-5) formában is írható, a

(PLUS 7(MINUS 5))

alakot kapja, és ennek eredménye is 2.

A szorzás a TIMES parancs használatával történik, így pl. 7x5 kiszámításához a

(TIMES 7 5)

parancs használatos, amire 35 a válasz, és hogy kiszámítsuk a 7x5x3x1 értéket, a

(TIMES 7 5 3 1)

parancs beírása szükséges, amire a 105 válasz érkezik.

A LISP csak egész számok használatát engedi meg, és ez okozhat némi nehézséget, amikor osztani akarunk, pl. 7/5=1.4, vagy tört formában, 1 2/5. Három különböző, az osztással kapcsolatos LISP parancs van, amelyek eredménye is különböző, a DIVIDE, a QUOTIENT és a REMAINDER.

A DIVIDE parancs egy rendezett párt (speciális listatípust) ad vissza, amelynek első eleme a hányados, második eleme a maradék, pl.

(DIVIDE 7 5)--> (1.2)

A QUOTIENT a hányados egészrészét adja vissza és a törtrészt nem veszi figyelembe,

pl. (QUOTIENT 7 5)--> 1

végül a REMAINDER az osztás maradékát adja vissza,

pl. (REMAINDER 7 5)--> 2

Jegyezzük meg, hogy a LISP a -32768 és 32767 közti intervallumon belül minden egész számot megenged, és hogy hibát okoz, ha egy aritmetikai számítás eredménye ezen az intervallumon kívül esik.

A LISP-ben az ALPHA szó reprezentálhat egy változót vagy egy adatrészt, és a két esetet egy szimpla idézőjel segítségével különböztetjük meg. így ALPHA változót jelent, és értéke elérése után azonnal kiíródik, míg 'ALPHA az ALPHA szót reprezentálja, mint szöveg-adatot. Ha egy ALPHA változó nem kapott értéket, LISP értékét UNDEFINED-nak fogja tekinteni, ez a nyelvben speciális azonosító, erre a használatra fenntartva.

Hogy egy változónak értéket adjunk, ezt megtehetjük a SETQ azonosító használatával.
Pl.
(SETQ ALPHA '(A B C D E))

az ALPHA változó értékéül az (A B C D E) listát adja. [Vegyük észre a ' idézőjelet, amely azt eredményezi, hogy (A B C D E) nem kerül kiértékelésre]. Amikor a változó hozzárendelése megtörtént, értéke kijelződik, ha zárójel nélkül szerepel, azaz ALPHA.

Például,
(SETQ TEN 10)
(TIMES TEN TEN)

a TEN változónak a 10 értéket adja, majd kiszámítja TEN négyzetét, ami 100.
A változóhoz rendelendő kifejezés értékét is ki lehet számítani a SETQ paranccsal, így a

(SETQ ANSWER (TIMES TEN TEN))

utasítás a számítás eredményét, azaz a 100-at fogja az ANSWER változóhoz rendelni.

## Listakezelés

Két alapvető függvény van, amivel egy lista bizonyos értékeit lehet megtalálni, mégpedig a CAR és a CDR függvények.
Az IS-LISP a CAR és HEAD parancsot engedi meg adott lista első elemének megjelölésére, a CDR és TAIL parancsokat pedig a további elemekből álló lista megjelölésére.

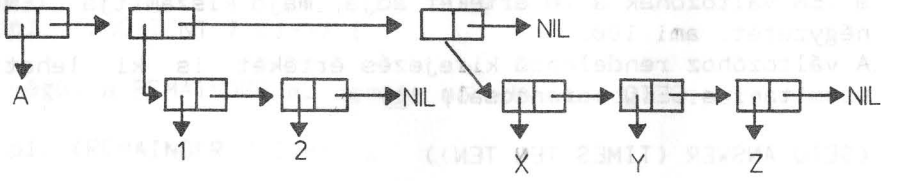
Tehát az (X Y) listára
(CAR'(X Y))--> X
(CDR'(X Y))-->(Y)

Tekintsük a következő SETQ utasítás által definiált listát:

(SETQ EXAMPLE'(A (1 2) (X Y Z)))

A diagram erre a listára:

EXAMPLE



Egyes tételek az EXAMPLE listából a következő módon nyerhetők:

| Utasítás | Eredmény |
|---|---|
| (CAR EXAMPLE) | Az A azonosító |
| (CDR EXAMPLE) | A két tagból álló ((1 2) X Y Z)) lista |
| (CAR(CDR EXAMPLE)) | Az (1 2) lista |
| (CDR(CDR EXAMPLE)) | Az (X Y Z) lista |
| (CAR(CAR(CDR EXAMPLE))) | Az 1 azonosító |

Az IS-LISP lehetővé teszi , hogy a felhasználó az ilyen utasításokat lerövidítse, ezt írva:

| | |
|---|---|
| (CAAR EXAMPLE) | (CAR(CAR EXAMPLE)) helyett |
| (CDDR EXAMPLE) | (CDR(CDR EXAMPLE)) helyett |
| (CADR EXAMPLE) | (CAR(CDR EXAMPLE)) helyett |
| (CAAAR EXAMPLE) | (CAR(CAR(CAR EXAMPLE))) helyett |
| (CADAR EXAMPLE) | (CAR(CDR(CAR EXAMPLE))) helyett, |

összesen legfeljebb három CAR és CDR utasítás mellett, akármilyen kombinációban.

## A T azonosító

A speciális T azonosító az az érték, amit a gép kiad, ha egy Boole-függvény igaznak értékelődött ki.

Például,

```
(NULL NIL)    -->   T, mivel NIL ().
(ATOM 3)      -->   T, mivel 3 valóban atom
(EQ 5 5)      -->   T, mivel 5=5
```

## Döntés

A BASIC-ben a döntéseket általában az IF... THEN... ELSE szerkezetben hozzuk, amit a következő módon alkalmazunk:

```
IF <1. állítás> THEN <1. kifejezés>
ELSE IF <2. állítás> THEN <2. kifejezés>
ELSE IF <3. állítás> THEN <3. kifejezés>
...
ELSE <kifejezés>
```

A LISP-ben ez a következő formát kapja:

```
(COND  (<1. állítás> <1. kifejezés>)
       (<2. állítás> <2. kifejezés>)
       ...
       (T <kifejezés>)),
```

ahol T a speciális azonosító az igaz állítások reprezentálására. Így a COND függvény úgy dolgozik, hogy minden állítást kiértékel, míg csak nem talál egy igazat. Ekkor kiértékeli a megfelelő kifejezést és annak értékét adja vissza.

Példa

```
(COND ((ATOM'(X Y)) (TIMES 4 9))
      ((ATOM 4) (TIMES 5 8))
      (T (TIMES 6 7)))
```

Itt, mivel '(X Y) lista és nem atom, ezért az első állítás NIL-t ad és nem veszi figyelembe a program. A második állítás viszont igaz, hiszen a 4 atom, és így az utasítás értékeként a gép (TIMES 5 8)-at, azaz a 40-et adja vissza.

**Ciklus**

A LISP-ben egy kifejezés-csoport ismétlése a LOOP paranccsal történik. Így pl. a következő utasítás

```
(LOOP(PRINT(PLUS 7 5))
   (PRINT(DIFFERENCE 7 5)))
```

ciklust eredményez, amelyben felváltva a 12, majd a 2, majd a 12..., stb. jelződik ki, amíg meg nem nyomjuk a STOP gombot. Jobban használható a LOOP olymódon, hogy események egy sorozatát idézi elő, mindaddig, amíg egy bizonyos terminális (befejező) feltétel nem teljesül, és ezt a WHILE vagy az UNTIL függvényekkel éri el.

Amikor egy ciklus tartalmaz WHILE vagy UNTIL utasítást, addig folytatja a végrehajtást, amíg a WHILE állítás igaz, illetve, amíg az UNTIL állítás hamis marad.

Adjuk a C változónak a 7-es értéket a (SETQ C 7) paranccsal, A-t állítsuk 0-ra a (SETQ A 0) paranccsal.

Ekkor alkalmazzuk ezt a ciklust:

```
(LOOP (WHILE(GREATERP C 0) A)
      (SETQ A(PLUS A C))
      (SETQ C(SUB1 C)))
```

Ez a 7+6+5+4+3+2+1=28 értéket fogja adni, mivel a ciklus mindig megismétlődik, amikor C>0.

Egy másik megoldás: Adjuk C-nek a 0 értéket a (SETQ C 0) paranccsal és A-t is 0-zzuk a (SETQ A 0) paranccsal.

Ekkor alkalmazzuk ezt a ciklust:

```
(LOOP (UNTIL (EQ C 8) A)
      (SETQ C (PLUS A C))
      (SETQ C (ADD1 C)))
```

ami az 1+2+3+4+5+6+7=28 választ adja, mivel addig ismétlődik a ciklus, mígnem C=8.

3. Fejezet

# INPUT/OUTPUT

A legfontosabb parancsok bármelyik nyelvben az input/output függvények, mivel ezek nélkül semmilyen értéket nem vihetnénk be a gépbe és számításaink eredményét sem nyerhetnénk vissza.

## Input szintaxis

A legszokásosabb módja, hogy adatokat/programokat vigyünk a LISP-be, a READ függvény használata. Ez lehetővé teszi, hogy rendezett párokat, listákat, változókat és -32768 és 32767 közti számokat vigyünk be az aktuális input csatornán keresztül. Amikor először kapcsoljuk be a rendszert, az input csatorna száma "0", a billentyűzethez kapcsolva, bár ez megváltoztatható az OPEN, illetve az RDS parancsokkal, amint ez a 6. fejezetben szerepel.

Jegyezzük meg, hogy ahol % jel van olvasás közben, onnan kezdve egészen az első sorvég jelig mindent commentnek tekint a gép és nem veszi figyelembe.

Az azonosítók inputjának szintaxisa olyan, hogy a gép elfogadja az A....Z, a....z, 0....9 karaktereket, valamint a következő speciális karaktereket:

| | | |
|---|---|---|
| - mínusz | _ aláhúzás | = egyenlőség |
| , vessző | ; pontosvessző | : kettőspont |
| [ bal szögletes zárójel | * csillag | & et-jel |
| $ dollár | ~ hullám | # hashmark |
| @ | / per-jel | ? kérdőjel |
| + plusz | \| | { kapcsos zárójel |
| ↑ felfelé nyíl | < kisebb-jel | > nagyobb-jel |
| \ backslash | | |

Például, A?BC, ::><:<-40:, + legális azonosítók.

Minden más karaktert az escape ('!') karakternek kell megelőznie, ha egy azonosítóban szerepel. Általában minden "(" balzárójelet közvetlenül le kell zárni egy ")" jobbzárójellel, de a "szuperzárójel", "]" használható az ö s s z e s megnyitott balzárójel lezárására. Például:

(DEFUN SQUARE (X) (TIMES X X]

Általában egy azonosító idézése úgy történik, hogy a ' karaktert tesszük eléje, pl. 'ABC. Ehelyett dupla

idézőjelek közé is tehetjük, pl. "ABC". Ez könnyű megoldás arra, ha különleges karaktereket, pl. space-eket akarunk az azonosítókba rakni.

Pl. (PRINTC"A B")

**Output formátum**

Az outputot általában a négy print utasítás egyikével visszük ki: PRIN, PRINC, PRINT, és PRINTC.

Két különböző formátum van: a PRIN típusú és a PRINC típusú. A PRIN-nel készült output visszaolvasható lesz a READ utasítással, és, ahol kell az azonosítók escape ('!') jelet kapnak. A PRINC utasítás viszont nem látja el az azonosítókat ezzel a jellel, hanem változatlanul viszi ki.

Az output képzésének egy másik módja a SPRINT függvény használata, ahol az egyes tételek egy sorba kerülnek, ahol ez lehetséges, egyébként pedig ebben a formában:

```
(<függvény-név>
 <1. arg.>
 <2. arg.>
 <3. arg.>
...
 <n. arg.>)
```

Amikor a definíció elkészült, SPRINT két üres sort visz ki, és a NIL értéket adja vissza.

4. Fejezet

# FÜGGVÉNYEK DEFINIÁLÁSA

Mint már említettük, a LISP nagyon rugalmas nyelv, amely kiterjeszthető úgy, hogy az egyedi felhasználók igényeinek eleget tegyen.
Amikor éppen beléptünk a LISP környezetbe, már egy egész sor függvény áll rendelkezésünkre, amely definiált és hívható egyszerűen a megfelelő függvény-néven, pl.

```
(PLUS 4 7)
```

A "PLUS" függvény a LISP interpreter részeként van definiálva és hatására a "kiértékelő" megkeresi a 4 és a 7 összegét. Nagyon valószínű, hogy néha használni akarunk valamilyen függvényt, amely még nincs definiálva és ilyenkor magunknak kell definiálni a "DEFUN" parancs segítségével.

A parancs szintaxisa:

```
(DEFUN NAME(PARAMETERS) (BODY))
```

ahol NAME az a név, amin a függvényt ezentúl hívjuk;
PARAMETERS a függvény input adatai; végül
BODY egy vagy több, előzőleg definiált LISP függvényből áll.

1. PÉLDA

Tekintsük azt a helyzetet, amikor 2x+3y értékét kell meghatározni több ízben egy program futása során. Beírhatnánk a következő programrészt minden alkalommal:

```
(PLUS (TIMES 2 X) (TIMES 3 Y))
```

Ezt nyilvánvalóan sokáig tart begépelni és jobb lenne, ha volna egy függvényünk, amivel ezt mindig végrehajtjuk. Ezt megtehetjük a következő utasítással:

```
(DEFUN OPERATE (P Q) (PLUS (TIMES 2 P) (TIMES 3 Q)))
```

A LISP interpreter válaszol:

```
(LAMBDA (P Q) (PLUS (TIMES 2 P) (TIMES 3 Q)))
```

jelezve, hogy a függvényt sikeresen definiáltuk.

Ha most a 2x+3y függvényt ki akarnánk értékelni, egyszerűen beírhatnánk:

```
(OPERATE x y)
```

"x" és "y" értéke adódik a függvényben "p" és "q" értékéül, s a gép kiszámítja a függvényt, a kiértékelő segítségével.

```
Pl. (OPERATE 4 5)--> 23
```

A függvénynek ez a használata igen struktúrált programozási módhoz vezet, amit "TOP-DOWN" megközelítésnek nevezünk. Ebben definiálódik egy függvény, amely használható egy második függvény definíciója részeként, amely ismét használható egy harmadik függvény definíciója részeként, stb., s a legtöbb LISP program így íródik.

2. PÉLDA

Tekintsük azt a feladatot, ahol egy második függvényt, "OPERATE2"-t kell definiálni, amely 3(2x+3y) alakú. Ez a második függvény nagyon könnyen definiálható az elsőt, "OPERATE"-et felhasználva:

```
(DEFUN OPERATE2 (P Q) (TIMES 3 (OPERATE P Q)))
```

A 3(2X+3Y) érték most már kiszámítható egyetlen függvényhívással:

```
(OPERATE2 X Y)
```

**Rekurzió**

A rekurzió olyan eszköz a LISP programozó kezében, amely igen nagy rugalmasságot tesz lehetővé programok felépítésekor. Rekurzió segítségével lehet egy függvényt definiálni, amely ismételten hívja önmagát, míg egy bizonyos feltétel nem teljesül. Működésében nagyon hasonlít a standard LOOP-hoz, de általában rövidebb és hatékonyabb programot eredményez.

PÉLDA

Tekintsük azt a feladatot, hogy miképp definiáljuk a következő LEN rekurzív függvényt: LEN vesz egy LIST(A B C D E) listát és megszámolja, hány atom van a listában. Ezt a következőképp tehetjük:

```
(DEFUN LEN(X)
  (COND
     ((ATOM X) 0)
     (T (ADD1 (LEN (CDR X]
```

## Opcionális változók

Definiálhatunk egy függvényt, amelyik hívható (pl.) egy, de két argumentummal (változóval) is:

```
(DEFUN ADDON (A (B.2))
  (PLUS A B))
```

Itt az első változót mindig meg kell adni, míg a második opcionális--ha nem adjuk meg, a 2 értéket veszi fel.

```
Pl. (ADDON 4 8)-->12
    (ADDON 5)-->7
```

Egy függvénynek akárhány közönséges és opcionális változója lehet, azzal a megszorítással, hogy az opconális változók mind a közönségesek után következnek.

Jegyezzük meg, hogy a default érték (alapértelmezés, a fenti példában a 2) kiértékelődik, ha nem adtuk meg, tehát a fenti példa így is írható:

```
(DEFUN ADDON (A (B.(PLUS 1 1)))
   (PLUS A B))
```

## Lokális változók

Ha egy függvényt pl. egy közönséges és két opcionális argumentummal definiáltunk:

```
pl. (DEFUN A-FUN (A (B.1) (C.NIL)) ... )
```

és mindig egy argumentummal hívjuk, akkor a B és C változók valójában lokális változók: használhatók "ideiglenes társnak" az A-FUN függvényen belül és eltünnek, amikor megtörtént a visszatérés a függvényből.

## Függvények editálása

Amikor írunk egy LISP programot, gyakran fogunk benne hibát találni és szükségünk lesz arra, hogy megváltoztassuk egy függvény definícióját. Például tegyük fel, hogy begépeljük:

```
(DEFUN SQUARE (X) (TIMES X Y))
```

mert véletlenül az egyik X helyett Y-t ütöttünk be.

A SQUARE függvény editálásához írjuk be:

```
(FEDIT SQUARE)
```

A képernyő kivilágosodik és ezt fogjuk látni:

```
(LAMBDA (X) (TIMES X Y))
```

így reprezentálja a gép a függvényt: A DEFUN szót és a függvény nevét a LAMBDA szóval helyettesítette.

Hogy a függvényt editáljuk, vigyük a kurzort az Y-hoz, ahogy szokás és írjuk felül X-re. Amikor befejeztük a javítást, nyomjuk meg az ESCAPE gombot. A SQUARE függvényt ezzel újradefiniáltuk, a helyes formára.

Ha az új definíció olvasásakor hiba derült ki (pl. egy fölösleges "." vagy ")", vagy file-vég, ami kevés jobbzárójelet jelent), a hibaüzenet megjelenik a képernyő tetején néhány másodpercre. Ez eltűnik, és a kurzor visszatér, hogy a hibát ki lehessen javítani.

5. Fejezet

# KIMENTÉS ÉS BETÖLTÉS

A SAVE és a LOAD parancsok lehetővé teszik, hogy a rendszer állapotát kazettára vigyük és az később használható legyen. Mielőtt megkísérli, hogy a rendszert kimentse, győződjön meg róla, hogy bent van-e egy üres kazetta az adatátvitelre, és hogy az Enterprise mikroszámítógép output tokjából jövő vezetékek a magnó "MIC" és "REM" tokjához csatlakoznak-e.

Kell a rendszer aktuális állapotának egy nevet adni, ha ez pl. TUESDAY, akkor beütve a

(SAVE "TUESDAY")

parancsot, ez az állapot kimentődik a kazettára, készen arra, hogy későbbi időpontban betöltsük. Az újra-betöltés egyszerűen a

(LOAD "TUESDAY")

parancs beütésével történik, és így az összes, felhasználó által definiált kifejezés újra betöltődik a számítógép memóriájába. Ne felejtsük el csatlakoztatni a magnó "EAR" és "REM" tokjait az ENTERPRISE input tokjaihoz.

MEGJEGYZÉS

Amikor a kazettáról új file-t töltünk be, minden, éppen a számítógépben levő információ felülíródik. Ha ez az információ fontos, először mentsük ki a SAVE függvénnyel, ahogy fent leírtuk.

# AZ EXOS

Az EXOS az Enterprise mikroszámítógép kiterjeszthető operációs rendszere. Interface-t szolgáltat az IS-LISP és a gép hardware-e közt. Az EXOS fő alkotórészei egy csatorna alapú input-output (I/O) rendszer és bonyolult memóriakezelő eszközök. Az I/O rendszer lehetővé teszi a berendezés-független kommunikációt egy egész sor beépített berendezéssel, valamint további, a géphez csatolható berendezés-meghajtó egységgel.

A beépített berendezések :

1. Video-meghajtó szöveg- és grafikakezelésre.

2. Billentyűzet-kezelő, a joystick-kel, automatikus ismétlővel és programozható funkciós gombokkal.

3. Képernyő-editor, szövegkezelési lehetőségekkel.

4. Könnyen kezelhető sztereó hang-generátor.

5. Kazettás magnó.

6. Centronics-kompatibilis párhuzamos interface.

7. RS232 típusú soros interface.

8. Hálózati interface

A rendszernek sok alkotórészét néhány egybyte-os változó, az úgynevezett "EXOS-változók" irányítják. Ezeknek a változóknak egy teljes listája az 1. függelékben található.

A kezdő LISP programozónak nem sok dolga akad az EXOS-beli kommunikációval. Előbb-utóbb azonban szüksége lesz erre a kommunikációra. A fejezet további része ennek mikéntjéről szól.

## Képernyő - editor

Amikor a gépet éppen bekapcsoltuk, és az IS-LISP cartridge a helyén van, a felhasználó a "képernyő-editor"-ral kommunikál, amelyik pedig a LISP interpreterrel tartja a kapcsolatot. Ez lehetővé teszi, hogy a felhasználó mozgassa a kurzort a képernyőn, illetve a képernyőt mozgassa fel és le.

## Csatornák

Ahogy a fejezet elején említettük, minden az I/O csatornákon keresztül történik. A LISP-et úgy tervezték meg, hogy a legegyszerűbb műveletek ennek a ténynek az ismerete nélkül is elvégezhetők. Ugyanakkor, a rendszer teljes kihasználásához szükséges, hogy tudjuk, hogyan kezelhetők ezek a csatornák és a megfelelő EXOS változók.

Közvetlenül a gép bekapcsolása után a következő csatornák lesznek automatikusan nyitottak:

0. CSATORNA EDITOR

1. CSATORNA VIDEO (csak a grafikus rész) -- a GRAPHICS paranccsal nyitható meg.

2. CSATORNA VIDEO (csak a szöveges rész)

3. CSATORNA hang

5. CSATORNA billentyűzet

A kommunikáció a különböző berendezésekkel mindig ezeken a csatornákon keresztül történik, hacsak nincs az egyik valamilyen okból lezárva.

A 0., 2., 3. és 5. csatornát lehetőleg nyitva kell hagyni. Ha nincs rá nyomós ok, ne zárjuk le ezeket.

**Új csatornák**

Néha szükségünk lehet arra, hogy új csatornát nyissunk egy berendezés-meghajtó egység számára, hogy a kommunikáció ne a megadott alapcsatornák valamelyikén történjen. Erre két függvény szolgál: az OPEN és a CREATE. A legtöbb esetben a kettő ekvivalens, de mikor szalagon vagy lemezen file-t nyitunk, a CREATE új file-t létesít, míg az OPEN feltételezi, hogy a file már létezik.

PÉLDA

Tekintsük azt a feladatot, hogy egy függvényt kell definiálnunk, amely bizonyos függvényeket nyomtat párhuzamos nyomtatón SPRINT utasítással.

Ez a következő programmal oldható meg:

```
(DEFUN SPRINTER (EXP)
  (OPEN 10 "PRINTER:")
  (WRS 10)
  (SPRINT EXP)
  (WRS 0)
  (CLOSE 10))
```

A program megnyitja a 10. csatornát a párhuzamos nyomtató számára, majd a WRS 10 utasítással az aktuális outputot erre a csatornára irányítja. Az "EXP" paraméterrel definiált függvény a SPRINT utasítással kikerül a nyomtatóra, majd az aktuális outputot újra az alapcsatornához rendeli a program.

MEGJEGYZÉS

A csatornák használata közben könnyen elfeledkezünk arról, hogy az input és az output automatikusan az alapcsatornákon halad, ahogy ezt meg fogjuk alább mutatni, és az átirányításhoz az SNDS, RDS, WRS, és GRS parancsokra van szükség.

| Berendezés | Alapcsatorna | Parancs |
|---|---|---|
| HANG | 3 | SNDS |
| SZÖVEG(INPUT) | 0 | RDS |
| SZÖVEG(OUTPUT) | 0 | WRS |
| GRAFIKA | 1 | GRS |

PÉLDA

A 12. csatornán menő inputhoz szükséges:

```
(OPEN 12 "DEVICE:")
(RDS 12)
```

Amikor csatornát nyitunk, ilyen alakú parancsot használunk:

(OPEN 12 "<berend.>:<filenév>.<kiterj.>")

ahol <berend.>, <filenév>, <kiterj.> egyike sem kötelező.

A használható berendezések listáját lásd alább, ha nincs megadva, az alapértelmezés a szalag.

| | |
|---|---|
| KEYBOARD: | A billentyűzet |
| VIDEO: | Video képernyő berendezés |
| EDITOR: | Szövegszerkesztő berendezés |
| PRINTER: | Centronics port |
| SOUND: | Hang-berendezés |
| SERIAL: | RS232 port |
| NET: | Hálózat-berendezés |
| TAPE: | Kazetta berendezés |

SZALAG-FILE KEZELÉS

Néha szükségünk lehet arra, hogy információt vagy függvényeket mentsünk ki szalagra, anélkül, hogy a teljes környezetet kimentenénk. Ezt a következő függvény definiálásával tehetjük:

```
(DEFUN FILE EXP)
  (CREATE 11 "TAPE:RH")
  (WRS 11)
  (PRINT EXP)
  (WRS 0)
  (CLOSE 11))
```

Amikor a "FILE" függvényt hívjuk, az"EXP" paraméter tartalma kimentődik egy "RH" nevű file-ba. Ez később használható lesz egy második függvény segítségével, amely a kifejezést a szalag-file-ból előveszi.

PÉLDA

```
(DEFUN EXTRACT ((TEMP))
(OPEN 11 "TAPE:RH")
(RDS 11)
(SETQ TEMP (READ))
(RDS 0)
(CLOSE 11)
TEMP)
```

(EXTRACT) begépelésével, az "RH" file-ban levő kifejezést fogjuk megkapni.

## EXOS parancsok

Három LISP függvény van, az EXOS-READ, az EXOS-WRITE és az EXOS-TOGGLE, amelyek arra szolgálnak, hogy egy tapasztalt programozó lekérdezzen vagy megváltoztasson bizonyos rendszerváltozókat. Az 1. függelék tartalmazza ezeknek a változóknak a listáját, elhelyezkedésükkel együtt, és rövid leírást ad meg arról, mire használhatók.

Az "EXOS-változók" használatára és fontosságára álljon itt példaként egy függvény kinyomtatásának feladata, SPRINT utasítással, RS232 soros porttal összekötött printerre, amely 1200 BAUD sebességgel működik. A LISP parancsokat végigvizsgálva látjuk, hogy nincs függvény a sebesség csökkentésére, de az 1. Függelékből látjuk, hogy van egy megfelelő EXOS változó, amely így használható:

```
(DEFUN SERPRINT (EXP)
  (EXOS-WRITE 16 8) % Set BAUD rate to 1200
  (OPEN 11 "SERIAL:")
  (WRS 11)
  (SPRINT EXP)
  (WRS 0)
  (CLOSE 11))
```

# AZ INTERRUPTOK KEZELÉSE

Az IS-LISP Enterprise változatában van egy speciális azonosító, ami lehetővé teszi, hogy a LISP programozó függvényeket írjon, amelyek kihasználják az interrupt rendszert.

Mikor éppen bekapcsoltuk a mikroszámítógépet, a HANDLER változó UNDEFINED, és akármilyen software interrupt érkezik, ez SOFTWARE INTERRUPT(ERROR 4) kijelzést eredményez. Viszont, a felhasználónak lehetősége van, hogy definiájon egy függvényt--nevezzük FRED-nek--egy változóval, ami maga az interrupt HANDLER, ezt írva:

```
(SETQ HANDLER "FRED)
```

Amikor software interrupt érkezik, FRED fog hívódni, az interrupt típus, mint egyetlen változója szerint (ld. alább).

PÉLDA

Ha így definiáljuk FRED-et:

```
(DEFUN FRED (TYPE)
  (COND
     ((EQ TYPE 24) (PRINTC "UNDEFINED FUNCTION")))
     ((EQ TYPE 48) (PRINTC "NETWORK INTERRUPT"))
     ((EQ TYPE 64) (PRINTC "TIME-OUT"))
     (T NIL)))
```

Látjuk, hogy egy ilyen egyszerű függvény is, mint ez, számos hasznos eszközt ad a felhasználó kezébe:

1. Idő-kezelés

2. Információ arról, ha a hálózat szervízelést igényel.

3. Lehetőség, hogy a programot megszakítsuk, pl. a shift-F1 gombbal, és ezekután dolgozzunk tovább, mintha nem történt volna semmi.

A következő táblázat mutatja a HANDLER-nek adott értékeket, amikor a software interrupt beérkezik:

| Kód | Interrupt | Kód | Interrupt |
|---|---|---|---|
| 16 | 1. funkciós gomb | 26 | 11. funkciós gomb |
| 17 | 2. funkciós gomb | 27 | 12. funkciós gomb |
| 18 | 3. funkciós gomb | 28 | 13. funkciós gomb |
| 19 | 4. funkciós gomb | 29 | 14. funkciós gomb |
| 20 | 5. funkciós gomb | 30 | 15. funkciós gomb |
| 21 | 6. funkciós gomb | 31 | 16. funkciós gomb |
| 22 | 7. funkciós gomb | 32 | STOP |
| 23 | 8. funkciós gomb | 33 | akármelyik gomb |
| 24 | 9. funkciós gomb | 48 | HÁLÓZAT |
| 25 | 10. funkciós gomb | 64 | ÓRA |

(A 9-16-os gombok shifteltek)

### Funkciós gomb-interruptok

Ha egy funkciós gomb az üres stringgel van beprogramozva, megnyomásakor software interruptot generál. A LISP felügyelete alatt bizonyos gombok előre vannak programozva hasznos parancsokra, pl. FEDIT, a többiek az üres stringre.

## Stop-interrupt

Ha a (8-as számú) STOP_IRQ EXOS-változó nem-0 értékű, akkor a stop-gomb olyan kódot fog visszaadni, mint bármely más gomb, ami valójában a stop-gombot hatástalanítja. Egyébként, ha a stop-gombot nyomjuk meg, 32-es software interrupt érkezik.

## "Bármely gomb-interrupt"

Ha a (9-es számú) KEY_IRQ EXOS változó 0 értéket kap (alapértelmezése 255!), akkor, akármelyik gombot nyomjuk meg, 33-as software interrupt érkezik, a kódot is visszaadva.

## Hálózati interrupt

Amikor üzenet érkezik a hálózathoz, általában software interrupt generálódik. (kivéve, ha a (19-es számú) NET_IRQ EXOS-változó 0). Ez azért hasznos, mert így a felhasználó , elolvasva az üzenetet, reagálni tud rá.

Megjegyzés A (18-as számú) ADDR_NET EXOS-változó fogja ekkor tartalmazni annak a hálózati csatornának a számát, amelyről az adat beolvasható.

Az (5-ös számú) TIMER EXOS-változó általában 0 értékű. Ha viszont más értékre állítjuk, az EXOS-WRITE paranccsal, akkor ettől kezdve másodpercenként eggyel csökken. Amikor 0-vá válik, ezzel egy 64-es software interruptot generál. Ez hasznos lehet képernyőn működő óraként, ahogy azt alább láthatjuk.

PÉLDA

Az itt következő programrész hibakereséshez (debug) ad segítséget. Ha megnyomjuk a shift-F1 gombot, a LISP a DEBUG függvényt fogja hívni, amely lehetővé teszi, hogy a felhasználó begépeljen tetszőleges LISP kifejezést, amely kiértékelődik, és az eredmény kinyomtatásra kerül. Például, a változók értékei megtudhatók, egyszerűen a nevük beírásával. Befejezéskor a felhasználó beírja: "BYE", és a program folytatja addigi tevékenységét.

```
(SETQ HANDLER 'FRED)

(DEFUN FRED (TYPE)
  (AND (EQ TYPE24) (DEBUG)))

(DEFUN DEBUG((INPUT))
  (PRINTC"Lisp debugger - type BYE to leave")
  (LOOP
        (PRINTC"DEBUG>")
        (LOOP(WHILE(ATOM(SETQ INPUT (ERRORSET
        (READ))))))
        (SETQ INPUT(CAR INPUT))
        (UNTIL (EQ INPUT'BYE)
              (PRINTC"***Leaving Debug"))
        (ERRORSET(PRINT(EVAL INPUT)))))
```

Az interruptkezelés hatékony eszköz egy tapasztaltabb programozó kezében és lehetséges vele bonyolult függvényeket alkotni, amelyek akkor lépnek működésbe, amikor egy speciális software interrupt érkezik.

Megjegyzés Sajnos a LISP nem válaszol az interruptokra, amíg a felhasználóra vár, hogy az valamit begépeljen (vagyis, amíg a kurzor villog).

8. Fejezet

# AZ IS-LISP FÜGGVÉNYEI

Ez a fejezet a LISP függvények alfabetikus felsorolását tartalmazza , leírja struktúrájukat és működésüket, és néhány példát ad használatukra. Mindegyik függvény az itt felsorolt kategóriák egyikébe tartozik:

Subr - közönséges függvény, amely összes argumentumát kiértékeli.

Fsubr - függvény, amely nem szükségszerűen összes argumentumát értékeli ki.

Id - azonosító, a LISP-ben speciális jelentéssel.

Var - a rendszer szolgáltatta változó.

A LISP függvények argumentumait "<>" zárójelek közé tesszük, vagy "[]" zárójelek közé, az utóbbi opcionális argumentumot jelez.

A függvény eredménye is meg van adva, és ez egy lista (azaz (a b c ... z)), egy egész szám a -32768 és a 32767 közti intervallumban, vagy a "tet" szó, ami azt jelenti, hogy az erdemény típusa tetszőleges LISP típus lehet.

**(ABS <szám>)--> szám** Subr

Az argumentum abszolút értékét adja.

```
pl. (ABS 23)--> 23
    (ABS-19)--> 19
```

**(ADD1 <szám>)--> szám** Subr

Az argumentumnál 1-gyel nagyobb számot ad.

```
pl. (ADD1 23)--> 24
```

**(AND <1. kifejezés> <2. kif.>...)--> NIL vagy tet** Fsubr

Sorra kiértékeli argumentumait. Ha az egyik NIL, a függvény mindenképp NIL-t ad; egyébként az utolsó argumentum értékét adja.

```
pl. (AND(NUMBERP 10 (ATOM'A B)))--> NIL
                (AND (LISTP'(A B))'XYZ)--> XYZ
```

**(APPEND <X> <y>)--> list** Subr

Ha <x> és <y> listák, akkor ez a függvény azt a listát fogja adni, amelyik először <x>, majd <y> elemeit sorolja fel.

```
(DEFUN APPEND (X Y)
  (COND
    ((ATOM X) Y)
     (T (CONS (CAR X) (APPEND (CDR X) Y ]

pl. (APPEND'(A B) 'C D))--> (A B C D)
```

**(APPLY <fg> <arg>)--> tet** Subr

Az <fg> függvény értékét adja az <arg> argumentumok mellett.

pl. (APPLY NUMBERP'(10))--> T

**(ASSOC <kód> <a lista>)--> NIL vagy (<kód>.érték)** Subr

Megkeresi a rendezett párok megfelelő listáját az adott kódra. Ha a keresés sikerrel járt, egy (<kód>.érték) párt ad; egyébként NIL-t.

```
(DEFUN ASSOC (U ALIST)
        (COND
          ((ATOM ALIST) NIL)
          ((ATOM(CAR ALIST) ERROR "BAD
            ASSOCIATED LIST"))
          ((EQUAL U(CAAR ALIST)) (CAR ALIST))
          (T (ASSOC U (CDR ALIST]
```

pl. (ASSOC'A'((B.2) (A.-3)))--> (A.-3)

**(AT <sor> <oszlop>)--> NIL** Subr

A kurzort az aktuális output csatornán a(<sor>,<oszlop>) pozícióra viszi.

pl. (AT 10 20)--> NIL

**(ATOM <x>)--> T vagy NIL** Subr

T-t ad (igaz), ha <x> nem rendezett pár.

pl. (ATOM'(A.B))--> NIL
(ATOM'A)--> T
(ATOM 10)--> T
(ATOM CAR)--> T

**AUTOLOAD** **Id**

Amikor az EVAL program megkísérli egy

(<azonosító> <arg>)

alakú kifejezés kiértékelését, és azt találja, hogy <azonosító> UNDEFINED, ez általában hibát eredményez. A felhasználó viszont írhat egy autoloadert, amely LISP függvény, és mindig kiértékelődik, amikor a felhasználó definiálatlan értékű függvényt hív.

Az autoloader beállítható a

(SETQ AUTOLOAD 'FRED)

paranccsal. így a FRED függvény mindig hívódik, amikor definiálatlan függvény szerepel a programban, ennek a függvénynek a nevével, mint egyetlen argumentummal.
Ez a top-down struktúrájú programozásban a leghasznosabb, mert így a felső szinten levő függvények tesztelhetők anélkül, hogy az alsóbb szintek függvényeit definiálnánk.

**(BAND2 <x> <y>)--> szám** **Subr**

Az <x> és <y> számok bitenként képzett AND-jét fogja adni, amikor bináris formában szerepelnek.

pl., mivel 7 bináris fomája 111, 3-é 11, ezért
(BAND2 7 3)--> 3

**(BEAM <kifejezés>--> NIL** Subr

Ha (kif.> nem NIL, akkor az aktuális grafikus csatornán a sugárzás megkezdődik, különben befejeződik.

pl. (BEAM NIL)--> kikapcsolás

(BEAM T)--> bekapcsolás

**BLANK** Var

Ez reprezentálja a space karaktert (" ").

pl. (PRINC BLANK) egy space karaktert ad ki a képernyőre.

**(BNOT <szám>)--> szám** Subr

A szám bitenkénti NOT-ját képezi, azaz x-ből (-x)-1 lesz.

pl. (BNOT 7)--> -8
(BNOT -3)--> 2

**(BORDER <szám>)--> NIL** Subr

Ha 0 < (szám) < 255, akkor így a keret szine a <szám> által reprezentált színre változik.

pl. (BORDER 42)

**(BOR2 <x> <y>)--> szám** Subr

A két szám, <x> és <y>, bitenkénti OR-ját képezi.

pl., mivel 7 binárisan 111, 3 pedig 11, ezért
(BOR2 7 3)--> 7

**(BXOR2 <x> <y>)--> szám** **Subr**

Az <x> és <y> számok bitenkénti exkluzív -OR-ját képezi.

pl., mivel 7 binárisan 111, 3 pedig 11, ezért
(BXOR2 7 3)--> 4

**(CAPTURE <régi> <új>)--> NIL** **Subr**

Minden olvasási műveletet átirányít a <régi> csatornáról az <új> csatornára.

pl. (CAPTURE 34 102)--> NIL

**(CAR <x>)--> tet** **Subr**

Az <x> pontozott pár első mezejét adja eredményül. A 25-ös hiba lép fel, ha <x> nem rendezett pár. A CAR név helyett HEAD is írható. A CAR függvény kiterjeszthető a CAAR és a CAAAR rövidítéseket használva, a CAR CAR, ill. a CAR CAR CAR függvények reprezentálására.

pl. (CAR '(A.B))--> A
(CAR '((A.B).C))--> (A.B)
(CAAR '((A.B).C))--> A

**(CDR <x>)--> tet** Subr

Az <x> rendezett pár második mezejét adja. A 25-ös hiba lép fel, ha <x> nem rendezett pár. A CDR név helyett TAIL is írható. A CDR függvény kiterjeszthető a CDDR és a CDDDR rövidítéseket használva, a CDR CDR, ill. a CDR CDR CDR függvények reprezentálására.

```
pl. (CDR '(A.B))--> B
    (CDR '(A.(B.C)))--> (B.C.)
    (CDDR '(A.(B.C)))--> C
```

**(CHARACTER <szám>)--> id** Subr

A <szám>, mint ASCII kód által definiált azonosítót adja. A BASIC-beli CHR$ LISP-ekvivalense.

```
pl. (CHARACTER 65)--> A
```

**(CHARP <x>)--> T vagy NIL** Subr

T-t ad, ha <x> azonosító, különben NIL-t.

```
pl. (CHARP 'A)--> T
    (CHARP 10)--> NIL
```

**(CHARS <kif.>--> szám** Subr

Azoknak a karaktereknek a számát adja ki, amelyeket egy atom generálna, ha kinyomtatnák. Az érték függ az argumentum típusától.

```
lista--> 0
azonosító--> a kinyomtatott név karaktereinek száma
             (ekvivalens a BASIC-beli LEN-nel)
szám--> 6
kódpointer--> 10
```

pl. (CHARS 'A)--> 1
(CHARS 10)--> 6
(CHARS CAR)--> 10

**(CLEAR)--> NIL** **Subr**

Megvilágítja az aktuális grafikus csatorna video page-ét.

**(CLOSE <szám>)--> <szám>** **Subr**

Ha <szám> érvényes csatornaszáma egy nyitott file-nak, akkor azt lezárja.

pl. (CLOSE 1) lezárja az 1-es csatornához tartozó file-t.

**(CODEP <x>)--> T vagy NIL** **Subr**

T-t ad, ha <x> kódpointer, különben NIL-t.

pl. (CODEP (A.B.))--> NIL
(CODEP CAR)--> T

**(COMMENT <tet1> <tet2>...)--> NIL** **Fsubr**

Arra használjuk, hogy szöveges megjegyzéseket helyezzünk el egy kifejezésben.

pl. (COMMENT EZ NEM ÉRTELMEZŐDIK)-->NIL

**(COND (állítás)(állítás)...)--> TET** Subr

Ez egy módszer a struktúra irányítására, és hasonló a BASIC-beli IF THEN ELSE parancshoz.

```
IF <áll1> THEN <kif1>
ELSE IF <áll2> THEN <kif2>
...
ELSE <kif.>
```

a LISP-ben így írható:

```
(COND (<áll1> <kif1>)
      (<áll2> <kif2>)
   ... (T<kif.>))
```

```
pl. (COND ((EQ A 3) (PRINT "A=3"))
          ((EQ A 4) (PRINT "A=4"))
          (T NIL))
```

**(CONS <car rész><cdr rész>)--> lista** Subr

Új listát alkot a két megadott kifejezésből.

```
pl. (CONS'A'B)--> (A.B)
    (CONS'A(CONS'B NIL))--> (A B)
```

**(CONSTANTP <x>)--> T vagy NIL** Subr

T-t ad, ha <x> szám vagy kódpointer, különben NIL-t.

```
(DEFUN CONSTANTP (OR (NUMBERP X) (CODEP X]
```

```
pl. (CONSTANTP 10)--> T
    (CONSTANTP 'A)--> NIL
```

**(COPY <x>)--> <x>** **Subr**

<x> egy példányát adja.

```
(DEFUN COPY (X)
  (COND
    ((ATOM X)X)
    (T(CONS(COPY(CAR X)) ((COPY (CDR X]
```

pl. (COPY"(ABCDE))--> (ABCDE)

**(CREATE <szám> <filenév>)--> <szám>** **Subr**

Ez a függvény megnyitja a <filenév> file-t a <szám> csatornaszámon. Ekvivalens az "Access output"-tal specifikált BASIC OPEN parancshoz -- ld. a BASIC kézikönyvet.

pl. (CREATE 15 "NAME")

**CRLF** **Var**

CRLF értéke egy kocsi vissza/soremelés

**(CURSOR <kif>)--> NIL** **Subr**

Ha <kif> nem NIL, akkor a kurzor az aktuális output csatornán bekapcsolódik; különben kikapcsolódik.

pl. (CURSOR NIL)--> NIL kurzor ki
(CURSOR T)--> NIL kurzor be

**(DEFLIST <dlista> <ind>)--> lista** Subr

A <dlista> argumentum olyan lista, amelyiknek minden eleme kételemű lista; <dlista>-ban minden azonosítónak megvan az <ind> indikátorral jelzett tulajdonság-listán elhelyezett tulajdonsága. Az indikátorok egy listáját kapjuk vissza.

```
(DEFIN DEFLIST (U IND)
  (COND
    ((ATOM U) NIL)
    (T(PUT(CAAR U) IND (CADAR U))
      (CONS(CAAR U) (DEFLIST(CDR U) IND)))))
```

pl. (DEFLIST '((V DD) (H RG)) 'NAME)--> (V H)

ami ugyanaz, mint:

```
(PUT 'V 'NAME 'DD)--> DD
(PUT 'H 'NAME 'RG)--> RG
```

és visszakapható, így:

```
(GET 'V 'NAME)--> DD
```

**(DEFMAC <név> <paraméter> <törzs>... )--> <név>** Fsubr

Ez a szokásos mód makrók definiálására.

```
pl. (DEFMAC IF X
       (LIST
          "COND"
          (LIST (CADR X) (CADDR X))
          (LIST T (CAR (CDDDR X)))))
```

Ez az IF makrót definiálja, amelyet 3 argumentummal hívunk:

```
pl. (IF A B C)
```

"A" kiértékelődik,: ha igaz, B lesz az eredmény; különben C.

pl. (IF (ZEROP X) 4 5)--> 4, ha X=0
(IF (ZEROP X) 4 5)--> 5, ha X<>0

**(DEFUN <név> <paraméterek> <törzs>...)--> <név>** **Fsubr**

Ez a szokásos mód függvények definiálására.

pl. (DEFUN SQUARE (X) (TIMES2 X X))--> SQUARE

**(DEFVIDEO <+mód> <g-mód> <g-szín>--> NIL** **Subr**

Ez a függvény definiálja azokat a paramétereket, amelyeket csak a TEXT és a GRAPHICS függvények használnak.
<+mód> 40 vagy 80 és a képernyő oszlopainak számát jelöli.

<g-mód> a következők egyike:

| | |
|---|---|
| 1 | nagyfelbontású grafika |
| 5 | kisfelbontású grafika |
| 15 | attribútum mód |

<g-szín> a következők egyike:

| | |
|---|---|
| 0 | két-szín mód |
| 1 | négy-szín mód |
| 2 | 16-szín mód |
| 3 | 250-szín mód |

pl. (DEFVIDEO 40 1 0)--> NIL

**(DEL <ch>)--> <ch>** Subr

Lezárja a <ch> EXOS csatornát. A legtöbb berendezés esetében ekvivalens a CLOSE függvénnyel.

pl. (DEL 1)--> 1

**(DELETE <x> <y>)--> lista** Subr

Az <y> listát adja vissza, amelyből kitörölte <x> első előfordulását.

```
(DEFUN DELETE (A L)
 (COND
  ((ATOM L) L)
  ((EQUAL A (CAR L)) (CDR L))
  (T (CONS(CAR L)(DELETE A (CDR L))))))
```

pl. (DELETE 'A '(B A C A)--> (B C A)

**(DIFFERENCE <x> <y>)--> szám** Subr

Az <y> számot kivonja az <x> számból és kiadja az eredményt.

pl. (DIFFERENCE 7 3)--> 4

**(DIGIT <x>)--> T vagy NIL** Subr

T-t ad, ha <x> azonosító, amelynek kinyomtatott neve számjeggyel kezdődik (0-9)

pl. (DIGIT 'A)--> NIL
(DIGIT '!0A)--> T

(DISPLAY <csat><tól><sor><nál>)--> NIL Subr

<sor> db. sort jelenít meg a képernyőn, a <csat> csatorna page-énak <tól> sorától kezdve, az első sort a képernyő <nál> sorára teszi. Minden argumentumnak egésznek kell lennie.

pl. (DISPLAY 4 10 7 1)--> NIL
a 4-es csatorna page-énak 10.-16. sorait jeleníti meg, a képernyő legfelső sorában kezdve.

(DIVIDE <x> <y>)--> <hányados.mar> Subr

Az <x> számot osztja el az <y> számmal és a (hányados.maradék) párt adja vissza.
pl. (DIVIDE 7 3)--> (2.1)

DOLLAR Var

Ez a $ karakter.

(EDIT <kif>)--> tet Subr

Egy szöveges page-et készít, és a SPRINT használatával megjeleníti <kif>-et. Ekkor editálni lehet <kif>-et, ehhez használható az összes szövegszerkesztési lehetőség, a befejezés az "ESCAPE" gomb megnyomásával történik. A módosult forma beolvasásra kerül és megjelenik a képernyőn. Függvények editálására viszont a szokásos mód az FEDIT függvény hívása.

(ELLIPSE <x> <y>)--> NIL Subr

Ellipszist rajzol, az aktuális képernyőpozícióval, mint középponttal, az aktuális grafikus csatornán, az

aktuális előtér-színnel. <x>, ill. <y> rendre az x-sugarat, ill. az y-sugarat jelöli.

pl. (ELLIPSE 200 100)--> NIL

**(ENVELOPE <en><er>[<cp><cl><cr><pd>])--> NIL** **Fsubr**

A hangburkolónak egy teljes leírása található a BASIC kézikönyvben.
<en> a hangburkoló szám (0-250)
<er> a fázisok száma a hangkibocsátás előtt (255, ha nincs kibocsátás
A szögletes zárójelben levő paraméterek egy fázist határoznak meg és legfeljebb 12-szer ismételhetők.

<cp> hangmagasság-változás félhangokban
<cl> a baloldali hangszóró hangereje (-63...63)
<cr> a jobboldali hangszóró hangereje (-63...63)
<pd> a fázis időtartama, 1/50 s-okban.

pl. (ENVELOPE 1 1 10 30 20 10)--> NIL

**(EOF)--> T vagy NIL** **Subr**

Megvizsgálja, hogy az aktuális input csatornán filevégnél tartunk-e. Értéke T, ha igen, különben NIL.

**(EQ <kif1> <kif2>)--> T vagy NIL** **Subr**

A függvény T-t ad, ha argumentumaira a következők egyike igaz:

1. Mindkettő ugyanaz az azonosító.
2. Mindkettő ugyanaz a szám.
3. Mindkettő ugyanaz a lista a LISP memóriában.

```
pl. (EQ 1 'A)--> NIL
    (EQ 10 10)--> T
    (EQ '(A B) '(A B))--> NIL
```

**(EQUAL <kif1> <kif2>)--> T vagy NIL** **Subr**

Megvizsgálja, hogy két adott általános kifejezés azonos-e.

```
(DEFUN EQUAL (X Y)
  (COND
    ((EQ X Y) T)
    ((OR (ATOM X) (ATOM Y)) NIL)
    ((EQUAL(CAR X) (CAR Y)) (EQUAL (CDR X) (CDR Y)))
    (T NIL)))

  pl. (EQUAL 1 'A)--> NIL
      (EQUAL '(A B) '(A B))--> T
```

**(ERROR <szám>)--> nincs értéke** **Subr**

Feloldja a <szám> LISP hibakódot

**(ERRORSET <tet>)--> szám vagy tet** **Fsubr**

Kiértékeli az argumentumot. Ha eközben nem derül ki hiba, akkor visszaadja <tet> kiértékelésének eredményét, mint egy rendezett párt, a NIL-lel, különben a hibakódot.

```
pl. (ERRORSET (ATOM 10))--> (T)
    (ERRORSET (A.B))--> 17
```

**(EVAL <tet>)--> <tet>** **Subr**

Argumentumának második kiértékelését hajtja végre.

pl. (EVAL '(CAR '(A B)))--> A

**(EVLIS <lista>)--> lista** **Subr**

Kiértékeli a lista összes elemét és az eredmények listáját adja.

```
(DEFUN EVLIS (ARGS)
  (COND
    ((ATOM ARGS) NIL)
    (T(CONS(EVAL(CAR ARGS))
           (EVLIS(CDR ARGS))))))
```

pl. (EVLIS'((CHARP'Z) (EQ'A'B)))--> (T NIL)

**(EXOS-READ <var>)--> <szám>** **Subr**

A <var> EXOS változó aktuális értékét adja ki.

pl. (EXOS-READ 10)--} 234

**(EXOS-TOGGLE <var>)--> <szám>** **Subr**

Kapcsolóként működik és "váltja" a megfelelő EXOS változót. Az eredmény a változó aktuális státuszának komplemense.

pl. (EXOS-TOGGLE 8)--> 0

**(EXOS-WRITE <var> <val>)--> <val>** **Subr**

A <var> EXOS változót a <val> értékre állítja be.

pl. (EXOS-WRITE 10 234)--> 234

**(EXPAND <lista> <függvény>)--> lista** **Subr**

<függvény>-nek két argumentumúnak kell lennie. Ha a listában n elem van, L(1), L(2), ... ,L(n), akkor a következő listát kapjuk eredményül:

```
(<függvény> L(1) (<függvény> L(2) (...
(<függvény> L(n-1) L(n))...)))
```

```
(DEFUN EXPAND (L FN)
  (COND
    ((ATOM(CDR L)) (CAR L))
    (T(CONS FN
      (CONS(CAR L)
      (CONS(EXPAND (CDR L) FN) NIL))))))
```

pl. (EXPAND '(A B C D) 'PLUS2)--> (PLUS2 A (PLUS2 B (PLUS2 C D)))

**(EXPANDMACRO <macro függvény> <teszt> --> tet** **Subr**

A debug-hoz hasznos eszköz, akkor alkalmazható, amikor makrókkal dolgozunk.

pl. (EXPANDMACRO IF '(IF(EQ X 3)4 5))
lásd. DEFMAC-nál IF definícióját)

**(EXPLODE <azon>)--> list** **Subr**

Egy listát ad, <azon> kinyomtatott nevének karaktereivel.

pl. (EXPLODE 'ABCDE)--> (A B C D E)
(EXPLODE'''')--> NIL

**(FEDIT <fgv>)--> NIL** **Fsubr**

Hasonló EDIT-hez, de lehetővé teszi, hogy az <fgv> függvényt megváltoztassuk. Nyomjuk meg az "ESCAPE" gombot az editálás végeztével, hogy a függvény újradefiniálódjon.

**(FKEY <kn> <str>)--> NIL** **Subr**

A <kn> funkciós gombhoz a <str> stringet rendeli, amely legfeljebb 23 karakter lehet.
<kn> az 1-16 intervallumba esik, 9-16 az 1-8-ból shifttel állnak elő.

pl. (FKEY 1 "Most be vagyok programozva!")--> NIL

**(FLAG <azlis> <ind>)--> NIL** **Subr**

A listában minden azonosítót az <ind> jelzővel lát el.

pl. (FLAG '(A B C D) 'FINE)--> NIL

**(FLAGP <azon> <ind>--> T vagy NIL** **Subr**

Arra használatos, hogy megvizsgáljuk, az <azon> azonosító az <ind> jelzővel van-e ellátva.

pl. (FLAGP 'A 'FINE)--> T vagy NIL

**(FLATTEN <x>)--> lista** **Subr**

<x> egész részfa-struktúráját megszünteti.

```
(DEFUN FLATTEN (X)
  (COND
    ((NUL X) NIL)
    ((ATOM X) (CONS X NIL))
    (T (NCONC (FLATTEN(CAR X))) (FLATTEN(CDR X))))))
```

pl. (FLATTEN '(A((B)) NIL (C.D) E))--> (A B C D E)

**(FLUSH <ch>)--> NIL** **Subr**

Kiüríti a <ch> hálózati csatornát.

pl. (FLUSH 17)--> NIL

**(FSUBRP <x>)--> T vagy NIL** **Fsubr**

T-t ad, ha <x> egy Fsubr típusú függvény kódpointere, NIL különben.

pl. (FSUBRP COND)--> T
(FSUBRP 'A)--> NIL

**FUNARG** **Id**

FUNARG értéke határozatlan. Az interpreteren belül van speciális jelentése, ún. FUNARG lezárásoknál.

**(FUNCTION <fn>)--> FUNARG <fn> környezet** **Fsubr**

Éppen úgy működik, mint QUOTE <qu>, de csak függvényekre alkalmazható. Amikor az eredményül kapott formába (ún.

FUNARG lezárás) bizonyos argumentumokat helyettesítünk, minden változónak ugyanaz lesz az értéke, mint a lezárási operáció elvégzése előtt.

**(GENSYM)--> azon** **Subr**

Egyetlen azonosítót ad vissza, ennek formája G0000, G0001, stb.

pl. (GENSYM)--> G0042

**(GET <azon> <ind>)--> tet** **Subr**

Az <azon> azonosító listáján levő tulajdonságot adja eredményül, az <ind> jelző mellett, vagy NIL-t, ha ilyen tulajdonság nincs.

pl. (GET 'V 'NAME)--> DD

**(GETCHAR)** **Subr**

Egyetlen karakter olvasódik be az aktuális input áramból, ez lesz az eredmény. Jegyezzük meg, hogy az a függvény nem ugyanaz, mint a BASIC-beli INKEY$. INKEY$ LISP ekvivalense:

```
(DEFUN INKEY ()
  (RDS 5)
(PROG1 (AND (ZEROP (READSTATUS)) (GETCHAR)) (RDS 0)))
```

pl. (GETCHAR)--> J

**(GRAPHICS)--> NIL** **Subr**

Ha van grafikus csatorna, akkor az a képernyőn megjelenik; különben standard grafikus page kerül

megnyitásra, és megjelenik a képernyő felső húsz sorában. A grafikus mód és a szinek a DEFVIDEO paranccsal definiálódnak.

**(GREATERP <x> <y>)--> T vagy NIL** **Subr**

T-t ad, ha az <x> szám nagyobb, mint az <y> szám; különben NIL-t.

pl. (GREATERP 7 3)--> T

**(GRS <ch>)--> <szám>** **Subr**

Hatására <ch> lesz az aktuális grafikus csatorna, az előző száma pedig az eredmény.

**HANDLER** **Id**

A LISP működése közben "software interruptok" érkezhetnek az operációs rendszertől. A HANDLER változó beállítható úgy, hogy az iterrupt érkezésekor a rendszer egy bizonyos műveletet végezzen el. Lásd a 7. fejezetet.

**(HEAD <x>)--> tet** **Subr**

Ez a függvény a CAR függvénnyel azonos.

pl. (HEAD 'A.B))--> A

**(IMPLODE <idlist>)--> id** Subr

összeláncolja az <idlist>-ben levő atomokat, s az így kapott azonosítót adja vissza.

pl. (IMPLODE '(A B CD EF G))--> ABCDEFG

**(IN <ioport>)--> szám** Subr

A Z80 <ioport> input/output portjától kapott értéket adja vissza.

pl. (IN 129)--> 3

**(INK <col>)--> NIL** Subr

A palettáról választott "logikai" színre cseréli az aktuális grafikus csatorna előtér-szinét.

pl. (INK 10)--> NIL

**(INTERN <id>)--> <id>** Subr

Keresi <id>-et az oblist listán. Ha ott nem találja, kiegészíti vele a listát és eredményül is <id>-et ad.

pl. (INTERN 'WRITE)--> WRITE

**(JOY <szám1>)--> szám2** Subr

A <num> joystick szám státuszát adja eredményül--a definíciót lásd a BASIC kézikönyvben.

pl. (JOY 1)--> 17

**LAMBDA** **Id**

Speciális jelentése van az interpreteren belül--egy ún. lambda kifejezés.

**(LAST <x>)--> tet** **Subr**

Az <x> lista utolsó tagját adja vissza.

```
(DEFUN LAST(X)
  (COND
    ((ATOM X) X)
    ((NULL(CDR X)) (CAR X))
    (T (LAST (CDR X)))))
```

pl. (LAST'A B C))--> C

**(LENGTH <x>)--> szám** **Subr**

Az <x> lista hosszát adja-- a legfelső szinten mérve.

```
(DEFUN LENGTH(X)
  (COND
    ((ATOM X) 0)
    (T (ADD1 (LENGTH(CDR X))))))
```

pl. (LENGTH 'A)--> 0
(LENGTH '(A B (C.D) E))--> 4

**(LESSP <x> <y>)--> T vagy NIL** **Subr**

T-t ad, ha az <x> szám szigorúan kisebb, mint az <y> szám; különben NIL-t.

pl. (LESSP 7 3)--> NIL

**(LINELENGTH <tet>)--> szám** Subr

Egy sor hosszát adja, ahogy azt a SPRINT utasítás használja.

pl. (LINELENGTH 60)--> 64 (az előző érték)

**(LIST <arg1> <arg2>...)--> lista** Subr

Az (<arg1> <arg2> ... ) listát adja.

pl. (LIST 'A 'B '-9 'C)-->(A B -9 C)

**(LISTP <x>)--> T vagy NIL** Subr

T-t ad, ha <x> rendezett pár; különben NIL-t

pl. (LISTP '(A B))--> T

**(LITER <x>)--> T vagy NIL** Subr

T-t ad, ha <x> olyan azonosító, aminek kinyomtatott neve betűvel kezdődik; különben NIL-t.

pl. (LITER 'A)--> T
(LITER '!0A)--> NIL

**(LOAD <filenév>)** Subr

Betölt egy SAVE függvénnyel kimentett memóriarészt, vagy egy LISP kifejezésekből álló ASCII file-t.

pl. (LOAD "NAME")

(LOOP <tev1> <tev2> ...)--> tet Subr

Addig végzi a <tev1>, <tev2>, ... tevékenységeket, amíg a WHILE vagy UNTIL részben levő állítás nem válik hamissá/igazzá.

pl. (LOOP (PRIN'*) (SETQ CT(SUB1 CT))(UNTIL (ZEROP CT)))
egy sornyi *-ot nyomtat ki, ha előtte (SETQ CT <szám>) utasítás szerepelt.

LPAR Var

LPAR értéke a "(" karakter.

MACRO Id

Speciális jelentése van az interpreteren belül--egy ún. Macro kifejezés.

```
(DEFUN MAP (X FN)
  (LOOP
    (UNTIL (ATOM X) NIL)
    (FN X)
    (SETQ X (CDR X))))
```

pl. (MAP'ABC D) '(LAMBDA (x) (PRIN x)))--> NIL
az (A B C D) (B C D) (C D) (D) NIL
outputot fogja adni.

**(MAPC <x> <fn>)--> NIL** Subr

Az <fn> függvénybe helyettesíti sorra az <x> lista elemeit, s ezenkívül még egy NIL-t ad eredményül.

```
(DEFUN MAPC(X FN)
  (LOOP
    (UNTIL (ATOM X) NIL)
    (FN (CAR X))
    (SETQ X (CDR X))))
```

pl. (MAPC'(A B C D) '(LAMBDA (x) PRIN (x)))--> NIL
az ABCD NIL outputot adja.

**(MAPCAN <x> <fn>)--> lista** Subr

Az a követelmény, hogy az <fn>-be való minden helyettesítés listát ad eredményül. A MAPCAN függvény az így kapott listák összeláncoltját adja.

```
(DEFUN MAPCAN (X FN)
  (COND
    ((ATOM X) NIL)
    (T (NCONC (FN (CAR X)) (MAPCAN(CDR X) FN)))))
```

pl. (MAPCAN'(A B C D)'(LAMBDA(x) (LIST xx)))
-->
(A A B B C C D D)

**(MAPCAR <x> <fn>)--> lista** Subr

Azt a listát adja eredményül, amely az <x> lista elemeinek sorra <fn>-be való helyettesítésekor jön létre.

```
(DEFUN MAPCAR (X FN)
  (COND
    ((ATOM X) NIL)
    (T(CONS(FN(CAR X))(MAPCAR(CDR X) FN)))))

  pl. (MAPCAR'A B C D)'(LAMBDA(x) (CONCx x)))
      -->
      ((A.A) (B.B) (C.C) (D.D))
```

**(MAPCON <x> <fn>)--> lista** **Subr**

Követelmény, hogy az <fn>-be való minden helyettesítés listát adjon. MAPCON <fn>-be helyettesíti rendre <x>-et, (CDR <x>)-et, (CDDR <x>)-et, amíg a lista el nem fogy és a kapott listák összeláncoltja lesz az eredmény.

```
(DEFUN MAPCON (X FN)
  (COND
    ((ATOM X) NIL)
    (T (NCONC (FN X) (MAPCON (CDR X) FN)))))

  pl. (MAPCON'(A B C D)'(LAMBDA(x) (LIST
      (CONS 1 x)))--> ((1 A B C D)(1 B C D)(1 C D)
      (1 D))
```

**(MAPLIST <x> <fn>)--> lista** **Subr**

Azt a listát adja ki, amely az <x>-nek, (CDR<x>)-nek, (CDDR<x>)-nek,..., stb. <fn>-be való helyettesítésekor adódik, míg a lista ki nem merül.

```
(DEFUN MAPLIST (X FN)
  (COND
    ((ATOM X) NIL)
    (T(CONS(FN X) (MAPLIST(CDR X)FN)))))
```

pl. (MAPLIST'(A B C D)'(LAMBDA (x)
(LENGTH x)))--> (4 3 2 1)

**(MAX2 <x> <y>)--> szám** Subr

az <x> és az <y> számok közül a nagyobbikat adja.

pl. (MAX2 7 3)--> 7

**(MEMBER <x> <y>)--> NIL vagy lista** Subr

NIL-t ad, ha <x> nem eleme az <y> listának. Különben <y>-nak az x-szel kezdődő szeletét adja.

```
(DEFUN MEMBER (X Y)
  (COND
    ((ATOM Y) NIL)
    ((EQUAL X (CAR Y)) Y)
    (T(MEMBER X (CDR Y)))))
```

pl. (MEMBER'A'(B C A D E))--> (A D E)

**(MEMQ <x> <y>)--> NIL vagy lista** Subr

Ugyanaz, mint MEMBER, azzal a különbséggel, hogy az összehasonlításhoz EQ-t alkalmaz, nem pedig EQUAL-t.

```
(DEFUN MEMQ(X Y)
  (COND
  ((ATOM Y) NIL)
  ((EQ X(CAR Y)) Y)
  (T(MEMQ X(CDR Y)))))
```

pl. (MEMQ'A B) '(A B (A B) C))--> NIL

**(MESSOFF <szám>)--> szám** Subr

A MESSOFF és a MESSON függvények bizonyos rendszerüzenetek kiirásának vezérlésére alkalmasak. Egy vezérlő byte-ban levő biteket vezérel, az alábbiak szerint:

| | | |
|---|---|---|
| 1 | - Összegyűjtött kontrollszummák | OFF |
| 2 | - Kontrollszumma | OFF |
| 4 | - Hibakód (és üzenet, ha van) | ON |
| 8 | - Hibakereső | ON |
| 64 | - A (QUOTE<id>) "<id>"-dé való konverzióját vezérli | ON |

Például, (MESSON 2) a kontrollszummát írja ki, míg (MESSOFF 8) leállítja a hibakeresést.

pl. (MESSOFF 10)--> 255

**(MESSON <szám>)--> szám** Subr

ld. MESSOFF

**(MIN2 <x> <y>)--> szám** Subr

Az <x> és <y> számok közül a kisebbiket adja ki.

pl. (MIN2 7 3)--> 3

**(MINUS <szám>)--> szám** Subr

A <szám> negáltját adja.

pl. (MINUS 7)--> -7
(MINUS -3)--> 3

**(MINUSP <x>)--> T vagy NIL** Subr

T-t ad, ha <x> szám és x<0; különben NIL-t.

```
pl. (MINUSP -3)--> T
    (MINUSP 4)--> NIL
```

**(MKQUOTE <x>)--> lista** Subr

A (QUOTE<x>) listát adja.

```
pl. (MKQUOTE'(A B C)--> (QUOTE(A B C))
```

**(NCONC <x> <y>)--> lista** Subr

Összeláncolja <x>-et és <y>-t, anélkül, hogy <x>-et átmásolná. Úgy működik, mint APPEND, de nem olyan megbízható. Kétséges esetben használjuk APPEND-et!

```
(DEFUN NCONC(A B (W))
  (COND
    ((ATOM A) B)
    (T(SETQ W A)
     (LOOP
       (UNTIL(ATOM (CDR W)))
       (SETQ W(CDR W)))
          (RPLACD W B)
          A)))

  pl. (NCONC'(A B C)'(D E))--> (A B C D E)
```

**NIL** Id

A "hamis" igazságértéket adja.

**(NOT <x>)--> T vagy NIL** Subr

Ha <x> NIL, akkor T-t ad; ha <x> T, akkor NIL-t ad.

pl. (NOT NIL)--> T

**(NULL <x>)--> T vagy NIL** Subr

Ha <x> NIL, T-t ad; NIL-t különben. Teljesen ekvivalens a NOT függvénnyel (ld. fent).

pl. (NULL 1)--> NIL

**(NUMBERP <x>)--> T vagy NIL** Subr

T-t ad, ha <x> szám; különben NIL-t.

pl. (NUMBERP 7)--> T

**(OBLIST)--> lista** Subr

Listát ad az összes, a rendszer számára ismert azonosítóról. (FLATTEN (OBLIST)) egy egyszerűbb listát ad.

**(ONEP <x>)--> T vagy NIL** Subr

T-t ad, ha <x> az 1-es szám; különben NIL-t.

pl. (ONEP 1)--> T

**(OPEN <szám> <filenév>)--> <szám>** Subr

Megnyitja a <filenév> nevű file-t, a <szám>

csatornaszámon. Hasonlít a Basic-beli OPEN parancshoz.

pl. (OPEN 15 "NAME")

**(OR <kif1><kif2><kif3>...)--> tet** **Fsubr**

Az első nem-NIL argumentumot adja; ha ilyen nincs, akkor NIL-t.

pl. (OR (NUMBERP'A) (CONS A B)(ZEROP 'T))--> (A B)

**(ORDERP <id1> <id2>)--> T vagy NIL** **Subr**

T-t ad, ha <id1> kinyomtatott nevének ASCII kódja nagyobb, mint <id2>-é; különben NIL-t ad.

pl. (ORDERP 'A 'B)--> NIL
(ORDERP 'B 'A)--> T
(ORDERP 'AB 'AA)--> T

**(ORDINAL <id>)--> szám** **Subr**

<id> kinyomtatott nevének első karakterét adja, ASCII kódban.

pl. (ORDINAL 'APPLE)--> 65

**(OUT <érték> <ioport>)--> <érték>** **Subr**

Elküldi <érték>-et a Z80 <ioport>-tal specifikált input-output portjához.

pl. (OUT 10 254)--> 10

**(PAINT)--> NIL** Subr

Kitölti az összes olyan vonalat, amely az aktuális képernyőpozíciótól olyan határig tart, amely az aktuális képernyőpozíciótól különböző színű.

**(PAIR <x> <y>)--> egy lista** Subr

<x> és <y> azonos elemszámú listák kell, hogy legyenek. PAIR rendezett párok egy listáját adja, amelyeknek CAR-ja <x>-ből, CDR-je <y>-ból van.

pl. (PAIR"(A B)"(1 2))--> ((A.1)(B.2))

**(PALETTE <c0> <c1> <c2> <c3> <c4> <c5> <c6> <c7>)--> NIL** Subr

A palettát specifikálja az aktuális grafikus csatornára, a <c0>,...,<c7> egész számokkal a 0-255 intervallumban, mindegyikük egy-egy színt specifikál.

Megjegyzés 2-szín-módban c2,...,c7 fölösleges, 4-szín-módban c4,...,c7 fölösleges, 16-szín-módban a többi 8 szín c0,...,c7-ből a szokásos módon származtatható.

pl. (PALETTE 10 20 40 5 73 122 5 0)--> NIL

**(PAPER <col>)--> NIL** Subr

Átváltoztatja az aktuális grafikus csatorna háttérszínét <col>-ra. Ennek hatása akkor látszik, amikor a csatornára legközelebb CLEAR utasítást adunk ki.

pl. (PAPER 29)--> NIL

**(PEEK <cím>)--> szám** Subr

<cím> tartalmát adja ki.

pl. (PEEK 10)--> 6

**PERIOD** Var

Értéke a "." karakter.

**(PLIST <id>)--> lista** Subr

Az <id> azonosító tulajdonság-listáját adja.

pl. (PLIST'A)--> ((tul1. ért1)(tul2. ért2)....)

**(PLOT <x> <y>)--> NIL** Subr

Megvilágítja a képernyő (<x>,<y>) pozícióját, s ha ez megtörtént, ott egy vonalat rajzol.

pl. (PLOT 200 100)--> NIL

**(PLOTR <x> <y>)--> NIL** Subr

Az aktuális képernyőpozíciótól x irányba <x>, y irányba <y> elmozdulást tesz, majd vonalat rajzol.

pl. (PLOTR 100 200)--> NIL

**(PLOTMODE <szám>)--> NIL** **Subr**

A rajzmódot állítja be az aktuális grafikus csatornán, a következőképp:

0. PUT rajz (alapértelmezés)
1. OR rajz
2. AND rajz
3. XOR rajz

pl. (PLOTMODE 2)--> NIL

**(PLOTSTYLE <szám>)--> NIL** **Subr**

A vonal típusát adja meg az aktuális grafikus csatornán. <szám> egész az 1-14 intervallumban, 1-es folytonos vonalat jelent, a többi szám különböző pontozott vonalakat.

pl. (PLOTSTYLE 5)--> NIL

**(PLUS <arglista>)--> szám** **Fsubr**

Az <arglista>-ban megadott egész számok összegét szolgáltatja.

pl. (PLUS 10 3 16)--> 29

**(PLUS2 <x> <y>)--> szám** **Subr**

Ha pontosan két számot kell összeadni, akkor ez a hatékonyabb módja.

pl. (PLUS2 7 3)--> 10

**(POKE <cím> <érték>)--> <érték>** Subr

A memória <cím> által megadott helyére <érték>-et ír.

pl. (POKE 10 23)--> 23

**(PRIN [<arglista>])--> tet** Subr

Az <arglista>-ban levő argumentumok kiértékelődnek, és kiíródnak, közbenső blank jelek nélkül. A speciális karakterek elé escape jel kerül.

pl. (PRIN'A BLANK'B)--> B hatására A! B íródik ki.

**(PRINC [<arglista>])--> tet** Subr

Ugyanolyan, mint PRIN, azzal a különbséggel, hogy a speciális karakterek elé nem kerül escape jel.

pl. (PRINC'A BLANK'B)--> B hatására
A B jelenik meg.

**(PRINT [<arglist>])--> NIL** Subr

Ugyanaz, mint PRIN, azzal a különbséggel, hogy a CR/LF-et is magában foglalja, és NIL-t ad.

pl. (PRINT'A BLANK'B)--> NIL hatására
A! B jelenik meg.

**(PRINTC [<arglista>])--> NIL** Subr

Ugyanolyan, mint PRINT, de a speciális karakterek elé nem kerül escape jel.

pl. (PRINTC'A BLANK'B)--> NIL és
A B jelenik meg.

**(PROG1 <kif1> <kif2>)--> <kif1>** **Subr**

Az első argumentumot adja vissza.

pl. (PROG1 'A'B)--> A

**(PROG2 <kif1> <kif2>)--> <kif2>** **Subr**

A második argumentumot adja vissza.

pl. (PROG2'A'B)--> B

**(PROGN <kif1><kif2>...<kifn>)--> tet** **Fsubr**

Kiértékeli rendre <kif1>-et, <kif2>-t,...,<kifn>-et, és <kifn>-et adja vissza.

pl. (PROGN'A 3 4'B)--> B

**(PUT <id><ind><tul>)--> <tul>** **Subr**

A <tul> tulajdonságot <id> tulajdonságlistájára teszi az <ind> indikátor alatt.

pl. (PUT'V'NAME'DD)

**(QUOTE <tet>)--> kiértékeletlen<tet>** **Fsubr**

Leállítja a kiértékelést, és LISP-en keresztül kiírja: '<tet>

pl. (QUOTE A)--> A
vagy
'A--> A

**(QUOTEP <x>)--> T vagy NIL** Subr

T-t ad, ha <x> idézett kifejezés, különben NIL-t.

pl. (QUOTEP'(QUOTE Z))--> T

**(QUOTIENT <x> <y>)--> szám** Subr

Elosztja az <x> számot az <y> számmal, és - a maradékot figyelmen kívül hagyva - kiadja a "hányadost".

pl. (QUOTIENT 7 3)--> 2

**(RANDOM <szám>)--> szám** Subr

Véletlen számot ad a 0-(<szám>-1) intervallumban, hacsak a <szám> nem 0; ekkor az intervallum 0-32767. <szám> maximális lehetséges értéke 2000.

pl.(RANDOM 1967)--> 1510

(RANDOMISE <mag>)--> <mag> Subr

Alkalmazható a véletlen számok ellenőrzésére. Ha <mag> 0, akkor a sorozat nem jósolható, ha viszont nem 0, egy konkrét ismételt sorozatot nyertünk.

pl. (RANDOMISE 43)--> 764

(RDS <ch>)--> <cr> Subr

<ch>-t aktuális input áramnak véve, a megelőző input áramot adja.

pl. (RDS 1)--> 0

(READ)--> tet Fsubr

A függvény az aktuális input csatorna soronkövetkező s-kifejezéséből kiolvasott eredményt adja ki.

(READLINE)--> tet Fsubr

Az aktuális input csatornából a következő sor első karakteréig olvas, ebből egyetlen azonosítót képez, ezt adja eredményül.

(READ-STATUS)--> <szám> Subr

Az aktuális input csatorna státuszát adja ki. Értékei:

-1 - file-vég
0 - a karakter olvasásra kész
1 - különben

**(RECLAIM)--> <szám>** Subr

Tömörítést végez. Megadja a szabad LISP cellák számát. Ennek a számnak kb. az ötszöröse lesz a szabad byte-ok száma.

**(REDIRECT <régi> <új>)--> NIL** Subr

Az összes output operációt átirányítja a <régi> csatornáról az <új> csatornára.

pl. (REDIRECT 42 104)--> NIL

**(REMAINDER <x> <y>)--> szám** Subr

Ha x>0, az eredmény x MOD(ABS y). Ha x<0, az eredmény y-(ABS x) MOD (ABS y).

pl. (REMAINDER 7 3)--> 1
(REMAINDER -7 3)--> 2

**(REMFLAG <azon. lista><ind>)--> NIL** Subr

Az <azon. lista> azonosító listát megfosztja a flag-ektől.

pl. (REMFLAG '(A B)'FINE)--> NIL

**(REMOB <id>)--> <id>** Subr

Megkeresi az <id> azonosító oblist-jét, és ha van, eltávolitja.

pl. (REMOB'A)--> A

**(REMPROP <azon> <ind>)--> tet** Subr

Az <azon> azonosító tulajdonság-listájáról eltávolítja az <ind> tulajdonságot. NIL-t ad, ha ezt a tulajdonságot nem találja.

pl. (REMPROP'V'NAME)--> DD

**(REPEAT <szám><kif>)--> NIL** Fsubr

Kiértékeli a <kif> kifejezést, egymásután <szám>-szor.

pl. (REPEAT 5 (PRINC'AB))--> NIL,
ABABABABAB jelenik meg.

**(REVERSE <x>)--> lista** Subr

Az <x> lista megfordítottját adja vissza.

pl. (REVERSE'(A B(C D) E))
-->(E (C D) B A)

**(REVERSEIP <x>)--> lista** Subr

Ugyanazt teszi, mint REVERSE, csak sokkal gyorsabban, viszont kevésbé megbízható.

pl. (REVERSEIP'(A B C D))--> (D C B A)

**RPAR** Var

Értéke a ")" karakter.

**(RPLACA <mod><kif>)--> tet** Subr

A <mod> CAR mezejét helyettesíti <kif>-fel.

pl. (RPLACA'(A B) 1)--> (1 B)

**(RPLACD <mod><kif>)--> tet** Subr

A <mod> CDR mezejét helyettesíti <kif>-fel.

pl. (RPLACD'(A B) 1)--> (A.1)

**(SASSOC<kulcs><alista>)--> (kulcs><érték>) vagy tet** Subr

Egy adott <kulcs> kulcsot keres <alista>-ban, és ha megtalálta, a (kulcs,érték) párt adja vissza. Különben a függvény argumentumok nélkül értékelődik ki.

pl.(SASSOC'A'((B.27)(A.-3))FN)--> (A.-3)
(SASSOC'A'((B.27)"(LAMBDA NIL 5))--> 5

**(SAVE <filenév>)--> <filenév>** Subr

Kimenti e rendszer aktuális állapotát (amely később LOAD-dal visszanyerhető)

pl. (SAVE "NAME")

**(SET <azon><kif>)--> <kif>** Subr

<azon> értékét <kif>-re változtatja.

pl. (SET 'X 42)--> 42

(SETATTRIBUTES <szám>)--> NIL Subr

A grafikus attribútum-flag-byte-ot változtatja <szám>-ra (az aktuális grafikus csatornán). Alapvetően ez a flag-byte határozza meg, hogyan történik a rajzolás attribútom-módban. A teljes leírást lásd az EXOS specifikációknál. A függvény NIL-t ad eredményül.

pl. (SETATTRIBUTES 3)--> NIL

(SETCOLOUR <szám><szín>)--> NIL Subr

A <szám> "logikai" színt teszi a palettán a <szín> szín helyére.

pl. (SETCOLOUR 3 24)--> NIL

(SETQ <azon><kif>)--> <kif> Fsubr

Ugyanolyan, mint SET, csak az első argumentum automatikusan idézve szerepel.

pl. (SETQ X 42)--> 42

(SET-TIME <azon>)--> NIL Subr

Lehetővé teszi, hogy a rendszer óráját átállítsuk (ld. TIME). Az <azon> argumentumnak nyolc karakter hosszú azonosítónak kell lennie, amely az időt a következő formában ábrázolja:

hh:mm:ss

pl. (SET-TIME "01:30:42")--> NIL

**(SETVIDEO <ind><col><x><y>)--> NIL** Subr

Egy video oldalt definiál, közvetlenül az oldal megnyitása vagy kreálása előtt kell hívni.

<ind> értékei lehetnek:

0 - 128 kisfelbontású karakter (soronként 42 karakter)
1 - nagyfelbontású pixel grafika (872 pixel)
2 - 128 nagyfelbontású karakter (soronként 84 karakter)
5 - kisfelbontású pixel grafika (436 pixel)
15 - attribútum mód

A zárójelben megadott felbontások a teljes képernyőre vonatkoznak, két-szín-módban. A függőleges felbontás teljes képernyőre 27 karakter, azaz 243 pixel.

<col> értékei lehetnek:

0 - 2 szín
1 - 4 szín
2 - 16 szín
3 - 256 szín

<x> és <y> a video oldal méretét határozza meg.

1<x<42
0<y<255

pl. (SETVIDEO 1 2 40 20)--> NIL

**(SNDS <ch>)--> <szám>** Subr

Hatására <ch> lesz az aktuális grafikus csatorna, az előzőt pedig eredményül adja.

pl. (SNDS 56)--> 34

(SOUND <env><p><vl><vr><sty><ch><d><f>)--> NIL Subr

Egy hangot eredményez (feltételezve, hogy az aktuális hangcsatorna nyitva van a SOUND eszközre). A paraméterek jelentése:

<env> - A hanghoz használt hangburkoló (ld. ENVELOPE). 255-ös hangburkoló konstans erejű és magasságú "bip"-et ad a hangadás teljes időtartamára.

<p> - A hang kezdő magassága félhangokban

<vl> - A baloldali hangszóró maximális hangereje

<vr> - A jobboldali hangszóró maximális hangereje

<sty> - A hangmoduláció-byte. Ennek a byte-nak a hatását legjobban kísérletileg lehet meghatározni! A nulla tiszta hangot ad a hangcsatornákra és fehérzajt a zajcsatornára.

<ch> - A hangforrás - 0, 1, vagy 2 a megfelelő hangcsatornára, és 3 a zajcsatornára.

<d> - A hangadás időtartama (1/50 mp-ben mérve)

<f> - Flag-byte

0 - a hang a várakozó sorba kerül

128 - a hang minden, erre a csatornára várakozó hangot megelőz.

A rutin NIL-t ad eredményül.

pl. (SOUND 255 20 3 10 30 40 0 0)--> NIL

**(SPRINT <kif>)--> NIL** Subr

Ez a program formattáló. A <kif> kifejezést szebb formában jeleníti meg.

**(SUB1 <szám>)--> szám** Subr

<szám>-1-et ad vissza.

pl. (SUB1 23)--> 22

**(SUBLIS <alista> <kif>)--> tet** Subr

Az eredmény úgy alakul ki, hogy <kif> CAR részének minden előfordulásába <alista> CDR részét helyettesítjük.

```
pl. (SUBLIS'((A.10)(B.C))'(H A (B) A))
     --> (H 10 (C) 10)
```

**(SUBRP <x>)--> T vagy NIL** Subr

T-t ad, ha <x> egy szubrutinra mutató kódpointer, különben NIL-t.

pl. (SUBRP CAR)--> T

**(SUBST <x> <y> <kif>)--> lista** Subr

<kif>-ben mindenhol <y> helyett <x>-et helyettesít.

```
pl. (SUBST'A 'B (C B (A) (B A)))
     -->(C A (A) (A))
```

T Id

A TRUE igazságértéket jelenti.

(TAIL <x>)--> tet Subr

Ez a függvény a CDR-rel azonos.

pl. (TAIL '(A.B))--> B

(TERPRI)--> NIL Subr

A képernyőre egy kocsi-visszát tesz ki.

(TEXT)--> NIL Subr

Új szöveges oldalt nyit, az oldal oszlopainak számát a legutóbbi DEFVIDEO hívás határozza meg.

(TIME)--> azonosító Subr

A pillanatnyi időt adja, egy nyolckarakteres azonosítóban, ebben a formában:

hh:mm:ss

Az óra bekapcsoláskor indul, 00:00:00-ról, és a SET-TIME függvénnyel állítható át (ld. ott).

pl. (TIME)--> 00:10:38

**(TIMES <arglista>)--> szám** Fsubr

Kiértékeli <arglista> elemeit, és összeszorozza őket, a szorzatot adja eredményül.

pl. (TIMES 2 5 -3)--> -30

**(TIMES2 <x><y>)--> szám** Subr

Hatékonyabb mód pontosan két szám, <x> és <y> összeszorzására .

pl. (TIMES2 3 7)--> 21

**UNDEFINED** Id

Amikor egy azonosítót először használunk, "UNDEFINED" értéket kap.

**(UNTIL <kif>)** Fsubr

A LOOP ciklusutasításban szerepel.

pl. (UNTIL (EQ A 3))

**VERSION** Id

String, amely leírja, milyen LISP verzió működik éppen.

**(WHILE <kif>)** Fsubr

A LOOP ciklusutasításban szerepel.

pl. (WHILE (EQ A 3))

**(WRS <handle>)--> <handle>** **Subr**

Hatására a <handle> csatorna lesz az aktuális output áram.

pl. (WRS 2)--> 3

**(ZEROP <x>)--> T vagy NIL** **Subr**

T-t ad, ha <x> a nulla szám; különben NIL-t ad.

pl. (ZEROP 0)--> T

# EXOS VÁLTOZÓK

Az alábbi lista megadja azokat az EXOS változókat, amelyek az EXOS-READ, EXOS-WRITE és az EXOS-TOGGLE parancsokkal kezelhetők.

Minden változó beállítható a 0-tól 255-ig terjedő értékek bármelyikére. Ugyanakkor, sokan közülük kapcsolóként működnek, be- vagy kikapcsolnak valamit, ahol 0 felel meg a "be" és 255 a "ki" iránynak.

**0 IRQ_ENABLE_STATE**

0. bit- hang megszakítás
2. bit- 1 Hz megszakítás
4. bit- Video megszakítás
6. bit- Külső megszakítás

Az 1., 3., 5., 7. biteknek 0-nak kell lenni. Ezt a változót normál használatban ne módosítsuk.

**1 FLAG_SOFT_IRQ**

Egy eszköz nem-0-ra állítja, software megszakítás céljából.

**2 CODE_SOFT_IRQ**

Egy software megszakítás rutin vizsgálja, a megszakítás okának kiderítése céljából.

**3 DEF_TYPE**

Az alapértelmezésként szereplő eszköz típusa

0 szalag,
1 lemez.

**4 DE_CHAN**

A default csatorna száma.

| | |
|---|---|
| 5 TIMER | 1 Hz-es visszaszámláló, a nulla elérésekor software megszakítást okoz, és meg is áll. |
| 6 LOCK_KEY | A rögzített billentyűk aktuális helyzete. |
| 7 CLICK_KEY | 0 A billentyűzet hangja bekapcsolva,<br>1 kikapcsolva. |
| 8 STOP_IRQ | 0 A STOP billentyű megszakítást okoz,<br>1 A STOP billentyű a kódját adja vissza. |
| 9 KEY_IRQ | 0 Bármely billentyű megnyomása software megszakítást okoz, és a kódját is visszaadja. |
| 10 RATE_KEY | A leütött billentyű 1/50 s-onként ismétli magát. |
| 11 DELAY_KEY | Késleltetés az önismétlés kezdetéig<br>0 önismétlés letiltva. |
| 12 TAPE_SND | 0 magnó kontroll hang engedélyezve. |
| 13 WAIT_SND | 0 Ha a SOUND DRIVER pufferje megtelik, vár.<br><>0 SQFUL hibakódot ad. |

| | |
|---|---|
| 14 MUTE_SND | 0 Aktivizálja a belső hangszórót,<br><>0 letiltja a belső hangszórót. |
| 15 BUF_SND | A hangburkoló pufferjének mérete fázisokban. |
| 16 BAUD_SER | A soros csatorna sebessége<br>6 300 baud<br>8 1200 baud<br>10 2400 baud<br>12 4800 baud<br>14 9600 baud |
| 17 FORM_SER | A soros csatorna formátumát adja meg.<br>0. bit adatbitek száma (0=8 bit, 1=7 bit)<br>1. bit paritás vezérlés (0=kikapcs.)<br>2. bit paritás választás (0=páros, 1=páratlan)<br>3. bit stop, bitek száma (0=2, 1=1) |
| 18 ADDR_NET | A gép hálózati száma. |
| 19 NET_IRQ | 0 Adat érkezése a hálózaton megszakítást okoz. |
| 20 CHAN_NET | A hálózaton fogadott adatblokk csatornaszáma. |
| 21 MACH_NET | Az adó gép hálózati száma. |
| 22 MODE_VID | Video mód. |
| 23 COLR_VID | Szín-mód. |
| 24 X_SIZ_VID | Az oldal X mérete. |

| | |
|---|---|
| 25 Y_SIZ_VID | Az oldal Y mérete. |
| 26 ST_FLAG | 0 megjeleníti a státusz sort. |
| 27 BORD_VID | Keret-szín. |
| 28 BIAS_VID | A paletta 8...16 színeit meghatározó érték. |
| 29 VID_EDIT | Az editorhoz rendelt video page csatornaszáma. |
| 30 KEY_EDIT | Az editorhoz rendelt billentyűzet csatornaszáma. |
| 31 BUF_EDIT | Az editor pufferének mérete (256 byte-os lapokban). |
| 32 FLG_EDIT | Az editor flag-byte-ja. |
| 33 SP_TAPE | A nem-0 érték lassú magnókezelést eredményez. |
| 34 PROTECT | Nem-0 érték esetén védett file-t kreál. |
| 35 LV_TAPE | A magnó kimenet szintje. |
| 36 REM.1 | 0 REMOTE 1 bekapcs.<br><>0 REMOTE 1 kikapcs. |
| 37 REM.2 | 0 REMOTE 2 bekapcs.<br><>0 REMOTE 2 kikapcs. |

A következő EXOS változó számokat a felhasználó ne változtassa: 0,1,2.

# HIBAÜZENETEK

Az alábbi lista az IS-LISP interpreter által adott hibajelzéseket sorolja fel.

1. Memória-túllépés
2. A végrehajtás megszakítva (a stop billentyűt számítás közben megnyomták)
3. Megszakítás nyomtatás közben
4. Software megszakítás (meghatározatlan handler)
5. Aritmetikai túlcsordulás
6. Osztás nullával
7. Argumentumként várt szám
8. Elvárt: azonosító
9. Elvárt: byte (szám a 0-255 intervallumban)
10. Elvárt: byte vagy negatív szám
11. Elvárt: csatorna (szám a 0-255 intervallumban)
12. Elvárt: indikátor
13. Fölösleges "." vagy ")" olvasáskor
14. Illegális pont-jelölés
15. Túl nagy szám olvasáskor
16. Túl hosszú string (minden string legfeljebb 255 karakter hosszú lehet)
17. Definiálatlan függvény
18. Adott Fsubr alkalmazása függvényként
19. Adott szám alkalmazása függvényként
20. Rossz LAMBDA kifejezés
21. Rossz FUNARG kifejezés
22. Túl sok argumentum egy LAMBDA-kifejezéshez
23. Túl kevés argumentum egy LAMBDA-kifejezéshez.
24. Az opcionális argumentumoknak követnie kéne az egyszerű argumentumokat.
25. Egy atom CAR-jának vagy CDR-jének képzése
26. Rossz COND kifejezés

27. Rossz asszociációs lista
28. Rossz tulajdonság-lista
29. Primitív függvény argumentumszáma nem helyes
30. A rendszerváltozók módosítása lehetetetlen
31. Rendszer-azonosító egy LAMBDA/MACRO paraméter-listában
32. MACRO forma egy nulla paraméterrel
33. A MACRO paraméternek atomnak kéne lennie
34. Rossz MACRO kifejezés
35. Különböző hosszúságú listák a PAIR függvény argumentumaiként
36. Rossz argumentum egy véletlen függvényhez (0-2000--ig terjedő egésznek kell lennie)
37. Ismétlési faktorként számot vár
38. Rossz idézendő argumentum
39. RPLACA/RPLACD számára listát vár
40. Nincs jó azonosító lista IMPLODE számára
41. Túl sok karakter IMPLODE számára
42. SET, ill. SETQ azonosítót vár
43. Nincs elég memória a file betöltésére: próbáljuk meg a nem szükséges csatornákat lezárni és kísérelj ük meg újra
44. A betöltött kiterjesztésekkel együtt nem menthető ki
45. Rossz argumentum a SET-TIME számára.

# A FUNKCIÓS BILLENTYÛK

Az Enterprise mikroszámítógépnek nyolc funkciós billentyűje van, f1-től f8-ig cimkézve. Önállóan vagy shifttel együtt használva 16 funkciót látnak el összesen, amelyeket a felhasználó definiálhat az FKEY függvény segítségével.

Pl. (FKEY 5 "SZIA. 5. GOMB MEGNYOMVA")

beprogramozza KEY5-öt, hogy a fenti üzenet íródjon ki a megnyomásakor.

Bekapcsoláskor vagy reset után, az f1,...f8 gombok alapértelmezése az alábbi:

| GOMB | FUNKCIÓ |
|---|---|
| 1 | (FLATTEN (OBLIST)) |
| 2 | (DEFUN |
| 3 | (FEDIT |
| 4 | (EXOS-TOGGLE 36) váltja a remote 1 kazettát. |
| 5 | (TEXT) |
| 6 | (GRAPHICS) |
| 7 | (EXOS-TOGGLE 7) váltja a billentyű-hangot |
| 8 | (RECLAIM) |
| 9-16 | a null string, "". |

Megjegyzés Alapértelmezésben software interruptot okoznak megnyomáskor.

Készült a Texgraf Ipari Szövetkezet nyomdájában
Felelős vezető: dr. Bernáth Tibor – 87.1713 Texgraf, Dunaharaszti