

ALPHA MICROSYSTEMS AM-100

# ALPHALISP LANGUAGE PROGRAMMING SYSTEM

USER'S REFERENCE MANUAL



17875-N Sky Park North  
Irvine, CA 92714

May 1980

## IMPORTANT NOTICE FOR LISP USERS

Several new functions and enhancements have been added to LISP in AMOS Release 4.4. These features include improved error reporting and the addition of functions to the Extended Library to handle breakpoints.

### 1.0 ERROR HANDLING

When LISP reports an error, it now displays the user function in which the error occurred. For example:

```
* (DE DOUBLE (X) (PLUS XX))
DOUBLE
*(DOUBLE 2)
UNBOUND VARIABLE - EVAL IN DOUBLE
=====
XX
*
_
```

### 2.0 NEW FUNCTIONS

Three new functions have been added to LISP: RETFROM, BREAK, and UNBREAK. In addition, we have added the variable BREAKFNS (which is maintained by BREAK and UNBREAK).

#### 2.1 RETFROM

The call (RETFROM fn val) causes the most recent call of function fn to return with value val. If the specified function is not active, LISP generates an error message. For example:

```
* (DE F1 (X) (PROGN (F2) X))
F1
*(DE F2 () (RETFROM @F1 5))
F2
*(F1 7)
5
_
```

NOTE: The call (RETFROM PROG val) behaves exactly the same as (RETURN val).

## 2.2 BREAK (added to the Extended Library)

The call (BREAK fn1 fn2 ...) causes execution of a program to be interrupted if an attempt is made to call any of the specified functions. You may then single-step execution of the interrupted function by typing a line-feed, or resume execution by typing (RESUME). NOTE: fn1, fn2, ... are not evaluated.

## 2.3 UNBREAK (added to the Extended Library)

The call (UNBREAK fn1 fn2 ...) restores the specified functions so that they no longer interrupt program execution when called. (That is, this function clears breakpoints set via the BREAK function.) NOTE: fn1, fn2, ... are not evaluated.

## 2.4 BREAKFNS (added to the Extended Library)

BREAKFNS is a variable which contains a list of all functions which will interrupt program execution when called. BREAKFNS is maintained by BREAK and UNBREAK; therefore, you should not directly modify this variable. (See BREAK and UNBREAK, above.)

ALPHA MICROSYSTEMS AM-100

ALPHALISP LANGUAGE PROGRAMMING SYSTEM

USER'S REFERENCE MANUAL

ALPHA MICROSYSTEMS  
17875 Sky Park North  
Irvine, CA 92714

'AMOS', 'AlphaBasic', and 'AM-100'

are trademarks of products  
and software of

ALPHA MICROSYSTEMS  
Irvine, CA 92714

© 1977 - ALPHA MICROSYSTEMS

ALPHA MICROSYSTEMS  
17875 Sky Park North  
Irvine, CA 92714

## Table of Contents

PREFACE			
CHAPTER	1	INTRODUCTION	
	1.1	OPERATING INSTRUCTIONS .....	1-1
	1.1.1	System Files .....	1-1
	1.1.2	Operation .....	1-2
CHAPTER	2	DATA TYPES	
	2.1	NUMBERS .....	2-1
	2.1.1	Fixed Point Small Integers .....	2-1
	2.1.2	Floating Point Numbers .....	2-2
	2.2	CHARACTER STRINGS .....	2-2
	2.3	LITERAL ATOMS .....	2-2
	2.4	LISTS .....	2-2
CHAPTER	3	DEMONSTRATION LIBRARY	
	3.1	DOCTOR .....	3-1
	3.2	DIFF .....	3-2
	3.3	ILISP .....	3-3
	3.4	METEOR .....	3-3
CHAPTER	4	SUMMARY OF PERMANENT SYMBOLS	
BIBLIOGRAPHY			



## Preface

Version 1.1 of AlphaLisp corrects the following problems found in Version 1.0:

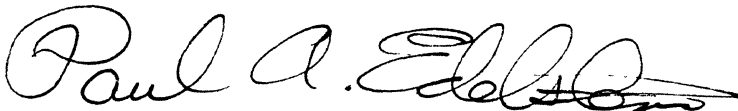
1. The function PUTPROP (and hence DE, DF, and DM) did not properly detect an attempt to redefine a permanent function. Redefining a permanent function could result in system failure.
2. The predicate MINUSP (and hence LESS, GREAT, etc) did not properly test the sign of a Floating Point number.
3. The function VCONC, which returned a list built by TCONC or LCONC, caused system failure if used improperly. This has been corrected at a slight expense in efficiency of the function.
4. The function TTYECHO, which was disabled in the change over to monitor 2.0, has been re-enabled.

Version 1.1 contains the following minor enhancements:

1. The automatic conversion of input from lower to upper case letters has been turned off. This allows the use of lower case in messages and input. **\*\* NOTE \*\*** Built in identifiers must be entered in UPPER CASE in order to be recognized!
2. The OUTPUT function previously returned the error 'FILE ALREADY EXISTS' if an attempt was made to create an output file which was already present. OUTPUT now supersedes (ie. pre-deletes) any existing file.

Finally, a table of contents and bibliography have been added to this manual to improve its usefulness.

Sincerely,



Paul Allen Edelstein  
Alpha Microsystems





## CHAPTER 1

### INTRODUCTION

This manual summarizes the features and operation of AlphaLisp, a version of the programming language LISP. The reader is assumed to be already familiar with LISP, or to have available an introductory text such as the LISP 1.5 PRIMER by Clark Weissman (Dickenson Publishing Co., 1967) or PROGRAMMING LANGUAGE LISP: Its Operation and Applications, edited by Edmund C. Berkeley and Daniel G. Bobrow (The M.I.T. Press, 1966).

AlphaLisp is based on UCI LISP, which in turn is based on Stanford Artificial Intelligence Project's LISP 1.6 (Stanford's LISP is documented in STANFORD ARTIFICIAL INTELLIGENCE LABORATORY OPERATING NOTE 28.6, STANFORD LISP 1.6 MANUAL, by Lynn H. Quam and Whitfield Diffie).

#### 1.1 OPERATING INSTRUCTIONS

##### 1.1.1 System Files

In order to use AlphaLisp, the following files must be present:

1. DSK0:LISP.PRG[1,4]
2. DSK0:LISP.LSP[7,4]

The AlphaLisp language processor, LISP.PRG is re-entrant and can be made part of the system by including it in a SYSTEM command in the file DSK0:SYSTEM.INI[1,4] (See AMOS Operator's Manual).

## 1.1.2 Operation

AlphaLisp is entered by typing:

```
.LISP
```

to which the system responds:

```
.LISP
LOAD EXTENDED LIBRARY?
```

at this point, the user enters either Y for YES or N for NO and presses the return key. If the user responds yes, several extended features will be loaded at the expense of memory space.

If the user has a file INIT.LSP, this file will be automatically loaded at this time. AlphaLisp will then print the following:

```
.LISP
LOAD EXTENDED LIBRARY?Y
"AlphaLisp 1.6 Version 1.0"
*
```

The star (\*) indicates that AlphaLisp is awaiting a command or data, and is referred to as a PROMPT.

The user may now enter any commands he wishes, for example:

```
*5
5
*(ADD 2 3)
5
*(MUL (ADD 3 4) 2)
16      (note: numbers are in octal)
*(DE FACTORIAL (N)
*      (COND ((ZEROP N) 1)
*            (T (MUL N (FACTORIAL (SUB1 N))))))
FACTORIAL
*(FACTORIAL 3)
6
*(SETQ VAR @(A B C))
(A B C)
*(CAR VAR)
A
*(CDR VAR)
(B C)
*
```

When the user wishes to return to the monitor, he enters (EXIT):

```
*(EXIT)
```

```
.
```

If the user wishes to abort output, cancel input, or interrupt program execution, he enters CTRL/C (^C):

```
*(SETQ X 1)
```

```
1
```

```
*(PROG () LOOP ((ONEP X) (GO LOOP)))
```

```
^C
```

```
INTERRUPT
```

```
(GO LOOP)
```

```
*
```

The program can then be single stepped by typing line-feed (Lf) or resumed by entering (RESUME):

```
* (Lf)
```

```
INTERRUPT
```

```
((ONEP X) (GO LOOP))
```

```
* (Lf)
```

```
INTERRUPT
```

```
(ONEP X)
```

```
* (Lf)
```

```
INTERRUPT
```

```
X
```

```
* (Lf)
```

```
INTERRUPT
```

```
(GO LOOP)
```

```
*(SETQ X 0)
```

```
0
```

```
*(RESUME)
```

```
NIL
```

```
*
```



## CHAPTER 2

### DATA TYPES

There are five types of data in AlphaLisp:

1. FIXEd Point small integers ( $-2^{13}$  thru  $2^{13}$ ).
2. FLOATing Point exponential reals (11 digit accuracy).
3. Character STRINGS.
4. Literal Atoms.
5. Lists.

#### 2.1 NUMBERS

There are two types of numbers, small integers and floating point numbers. AlphaLisp automatically performs conversion between these two types whenever possible to minimize storage requirements.

##### 2.1.1 Fixed Point Small Integers

A small integer is a number between  $-2^{13}$  and  $2^{13}$ . When entered, it is interpreted as a number in the radix specified by IBASE, which is initially set to 8. If an integer is entered followed by a decimal point, it is interpreted as a decimal (radix 10) integer. Integers can optionally be preceded by a plus (+) or minus (-).

Small integers are represented internally as special illegal addresses, and hence require no additional storage space when contained in a list.

### 2.1.2 Floating Point Numbers

Floating Point numbers are maintained to an accuracy of 11 digits. They may be entered with a decimal point (such as 3.2, .1, -.09, etc.) or in scientific notation (1E3=1000, 23.45E1=234.5, 23.45E-1=2.345, etc.).

## 2.2 CHARACTER STRINGS

Character Strings are any sequence of characters including special characters such as carriage-return or line-feed, enclosed between a pair of double quotes ("). Double quotes may appear within a string by entering them twice (e.g., "HE SAID ""WHAT"" AND TURNED").

When character strings are read in, they are not INTERNED on the OBLIST (entered into the symbol table), and are initialized to their own value, unlike identifiers which are initially unassigned.

## 2.3 LITERAL ATOMS

Literal atoms (or Identifiers) are a sequence of alphanumeric characters beginning with a non-numeric character, such as A, ALPHA, COUNT, VERYBIGIDENTIFIER, V1, etc. Literal atoms may include special characters such as blank, dot, parentheses, etc., if these characters are preceded by a slash (/). When this is done, these characters are referred to as slashified (e.g., THIS/ IS/ ONE/ IDENTIFIER, /123, A/.B, etc.).

## 2.4 LISTS

Lists are composed of a sequence of data elements including other lists, enclosed between a pair of parentheses (e.g., (A B C), (A (B C) D), (1 2 1.7 "HELLO" ID), etc.).

A special kind of two element list referred to as a dotted pair, has the form (left part . right part). Note that the two elements are separated by a dot (.). The CAR of a dotted pair is the left part, the CDR of a dotted pair is the right part.

On entry, brackets "[" can be used instead of parentheses "()". A close bracket "]" will automatically match an open bracket "[" at any time, producing implicit close parentheses ")" if necessary.

On entry, a double ALTMODE or ESC can be used to terminate an object being entered at any time, producing implicit close parentheses ")" if necessary.

## CHAPTER 3

### DEMONSTRATION LIBRARY

Included with AlphaLisp is a demonstration library composed of the following files:

1. DOCTOR
2. DIFF
3. ILISP
4. METEOR

#### 3.1 DOCTOR

DOCTOR is a program which immitates a psychiatrist who is holding a session with the user. To load doctor, enter:

```
.LISP
LOAD EXTENDED LIBRARY?N
"AlphaLisp 1.6 Version 1.0"
*(DSKIN DOCTOR)
  (Wait about one minute for DOCTOR to be loaded and processed)
NIL
*(WORKER)
*HELLO.
*
```

```
HOW DO YOU DO, PLEASE STATE YOUR PROBLEM
```

```
*MY NAME IS PAUL.
*
```



I AM NOT INTERESTED IN NAMES

\*

DOCTOR is an extremely large program and requires a machine with 56K of memory. Be sure your job has all of the system's memory when you try to load doctor, and that no systems such as BASIC have been loaded into the resident monitor by SYSTEM.INI with the possible exception of LISP itself. If insufficient space exists, you will receive the error message:

WORKSPACE FULL

\*

Whenever you tell the DOCTOR something, be sure to terminate your text with a blank line so that the DOCTOR knows you are finished speaking.

To leave DOCTOR, type a CTRL/C (^C) and then enter (EXIT):

\*^C

INTERRUPT

(TYI)

\*(EXIT)

.

### 3.2 DIFF

DIFF is a program which performs symbolic differentiation (That's calculus in case you've forgotten). To load DIFF, enter:

.LISP

LOAD EXTENDED LIBRARY?Y

"AlphaLisp 1.6 Version 1.0"

\*(DSKIN DIFF)

(Wait about 30 seconds)

NIL

\*(DIFF)

THE DERIVATIVE OF-

\*3X^3

WITH RESPECT TO-

\*X

IS-

9 X^2

THE DERIVATIVE OF-

\*^C

INTERRUPT

(READCH)

\*(EXIT)

.

DIFF is documented in the LISP 1.5 PRIMER by Clark Weissman.

### 3.3 ILISP

ILISP is the incremental language processor presented by L.A. Lombardi and Bertram Raphael in the paper "LISP as the Language for an Incremental Computer" in the book PROGRAMMING LANGUAGE LISP, edited by E. Berkeley. To load ILISP, enter:

```
.LISP
LOAD EXTENDED LIBRARY?Y
"AlphaLisp 1.6 Version 1.0"
*(DSKIN ILISP)
  (Wait a while)
NIL
*(GRINDEF SUBST1)
(DEFPROP SUBST1
 (LAMBDA (X Y Z)
  (COND ((ATOM Z) (COND ((EQ Z Y) X) (T Z)))
        (T (CONS (SUBST1 X Y (CAR Z))
                  (SUBST1 X Y (CDR Z)))))))
EXPR)
*(EVALQUOTE1 @SUBST1 @((A B) C NIL*))
(LAMBDA (G3)
 (COND ((ATOM G3) (COND ((EQ G3 @C) @(A B)) (T G3)))
       (T (CONS (SUBST1 @(A B) @C (CAR G3))
                 (SUBST1 @(A B) @C (CDR G3))))))
*(EXIT)
.
```

### 3.4 METEOR

METEOR is a programming language written in LISP by Daniel G. Bobrow and documented in the article "METEOR: A LISP Interpreter for String Transformations" which appears in the book PROGRAMMING LANGUAGE LISP, edited by E. Berkeley. To load METEOR, enter:

```

.LISP
LOAD EXTENDED LIBRARY?Y
"AlphaLisp 1.6 Version 1.0"
*(DSKIN METEOR)
    (Wait a while)
NIL
*WS123
(A ROSE IS A ROSE IS A ROSE)
*TEST1
(((* (ROSE) (FLOWER) * (SIMPLE REPLACEMENT))
  (* ((*P THE WORKSPACE IS)) * (DEBUG PRINTOUT))
  (* (IS A ROSE) 0 * (DELETION))
  (* (A FLOWER IS) (3 1 2) * (REARRANGEMENT))
  (* ((*P WS2)) *)
  (* (FLOWER) (1 OF RED) * (INSERTION))
  (* (A FLOWER) (THE 2) * (REPLACEMENT IN CONTEXT))
  (* ((*P WS3)) *)
  (* (FLOWER) * (NO OPERATION))
  (* (RED) (1 1) * (DUPLICATION))
  (* ((*P WS4)) *)
  (* (OF ($ 1)) (1) * (SINGLE UNKNOWN CONSTITUENT))
  (* (($ 1)) (QUESTION 1) * (FIRST CONSTITUENT))
  (* ((*P WS5)) *)
  (* (($ 2) FLOWER ($ 3))
    (3 2 1)
  *)
  (N CONSECUTIVE CONSTITUENTS))
  (* ((*P WS6)) *)
  (* (FLOWER $ ROSE) (1 3) * (UNKNOWN NUM OF CONSTITUENTS))
  (* ((*P WS7)) *)
  (* ($) (START C A B D) * (REPLACING ENTIRE WORKSPACE))
  (* (START ($ 1) $ D) (1 3 2 4) *)
  (* ((*P WS8)) *)
  (* ($) END))
*(METEOR TEST1 WS123)
(THE WORKSPACE IS)
(A FLOWER IS A ROSE IS A ROSE)
(WS2)
(IS A FLOWER A ROSE)
(WS3)
(IS THE FLOWER OF RED A ROSE)
(WS4)
(IS THE FLOWER OF RED RED A ROSE)
(WS5)
(QUESTION IS THE FLOWER OF RED A ROSE)
(WS6)
(QUESTION OF RED A FLOWER IS THE ROSE)
(WS7)
(QUESTION OF RED A FLOWER ROSE)
(WS8)

```

```
(START A B C D)
  (START A B C D)
*(EXIT)
.
```

Note that due to differences between AlphaLisp and LISP 1.5, the following changes were made to METEOR:

1. The form ( $\$.n$ ) has been change to ( $\$ n$ ).
2. Slashes (/) have been changed to backslashes (\).



## CHAPTER 4

### SUMMARY OF PERMANENT SYMBOLS

In the following summary, calling sequences to LISP functions are presented in S-expression form, with the CAR of the S-expression being the name of the function. An argument to a function is evaluated unless that argument is surrounded by quotes (") in the calling sequence definition presented. Quotes here mean that the function implicitly QUOTES that argument.

Several definitions refer to file specs. A file spec here refers to a standard AMOS file specification (Refer to AMOS OPERATORS MANUAL, Standard File Specification Format). If the file spec includes special characters, it should be enclosed in quotes.

#### 4.1 ABS

The call (ABS X) returns the absolute value of X, i.e., if  $X \geq 0$  then (ABS X) is X; otherwise, if  $X < 0$  then (ABS X) is (MINUS X).

#### 4.2 ADD

The call (ADD X Y) returns X+Y.

#### 4.3 ADD1

The call (ADD1 X) returns X+1.

#### 4.4 AND

The call (AND X[1] X[2] ... X[n]) returns the value of X[n] if all X[i] are non-NIL, otherwise it returns NIL. AND evaluates its arguments from left to right until either NIL is found in which case the remaining arguments are not evaluated, or until the last argument is evaluated. Note that (AND) returns T.

#### 4.5 ANDP

The call (ANDP X) returns the last element of X if all the elements of X are non-NIL, otherwise it returns NIL. If X is the empty list NIL, ANDP returns T.

#### 4.6 APPEND

The call (APPEND X[1] X[2] ... X[n]) returns a list containing the elements of the lists X[i]. For example, (APPEND @(A B C) @(D E) @((F)) @(G)) returns (A B C D E (F) G). Note: X[n] is an EQ-tail of the result of APPEND.

#### 4.7 APPLY

The call (APPLY FN ARGS) calls the function FN with the elements of the list ARGS as arguments and returns the result of FN. Example: (APPLY @ADD @(1 1)) returns 2.

#### 4.8 ASCII

The call (ASCII N) creates a single character identifier whose ASCII print name equals N. Note: The identifier is not INTERNED on the OBLIST. Example: (ASCII 101) returns an identifier with print name 'A'.

#### 4.9 ASSOC

The call (ASSOC X L FN) searches the list of dotted pairs L for a pair whose CAR is EQ to X. If such a pair is found, it is returned as the value of ASSOC. Otherwise, the value of FN, a function of no arguments, is returned. If FN is omitted or NIL, ASSOC simply returns NIL if a dotted pair is not found.

#### 4.10 ATOM

The call (ATOM X) is T if X is not a list, i.e., if X is either an identifier or a number. Otherwise the value of ATOM is NIL.

#### 4.11 BASE

The variable BASE controls the output radix for integers, and is initially set to 8. If BASE = 10, then integers will print with a trailing ".", unless the variable \*NOPOINT = T. If BASE > 10, digits higher than 9 will appear as the letters A - Z.

#### 4.12 BKT

The call (BKT) displays a backtrace of all expressions suspended either by an error or in order to evaluate imbedded subexpressions. BKT is used primarily to determine the context of an error. BKT returns NIL.

#### 4.13 CAR

The call (CAR X) returns the left half of the dotted pair X. If X is a list, this amounts to returning the first element of X. The CAR of an atom is illegal with one exception: The CAR of NIL is NIL.

#### 4.14 CDR

The call (CDR X) returns the right half of the dotted pair X. If X is a list, this amounts to returning the list X with its first element removed.

The CDR of an identifier returns the property list of that identifier.

#### 4.15 CAAR, CADR, ..., CDDDDR

All of the composite CAR-CDR functions with up to four A's and D's are available.

Examples:           (CADR X) =           (CAR (CDR X))  
                  (CAADDR X) =       (CAR (CAR (CDR (CDR X))))



## 4.16 CHRCT

The call (CHRCT) returns the number of character positions remaining on the output line of the selected output channel. When characters are output, if CHRCT is made negative, a (TERPRI) is executed.

## 4.17 CHRVAL

The call (CHRVAL X) returns the ASCII representation of the first character of the print name of X. If X is numeric, an ASCII '#' is returned. If X is a list, an ASCII '(' is returned.

## 4.18 CLRBF1

The call (CLRBF1) clears the terminal input buffer.

## 4.19 COND

A conditional expression has the following form:

```
(COND (e[1,1] e[1,2] ... e[1,n[1]])
      (e[2,1] e[2,2] ... e[2,n[2]])
      ...
      (e[m,1] e[m,2] ... e[m,n[m]]))
```

where the e[i,j]'s are any S-expression.

The e[i,1]'s are considered to be predicates, i.e., evaluate to a truth value. The e[i,1]'s are evaluated starting with e[1,1], e[2,1], etc., until the first e[k,1] is found whose value is not NIL. Then the corresponding e[k,2] e[k,3] ... e[k,n[k]] are evaluated respectively and the value of e[k,n[k]] is returned as the value of COND. It is permissible for n[k] = 1, in which case the value of e[k,1] is the value of COND. If all the e[i,1] evaluate to NIL, then NIL is the value of COND.

## 4.20 CONS

The call (CONS X Y) creates a dotted pair with left half X and right half Y. If Y is a list, this amounts to returning the list Y with X inserted as the first element. **\*\*NOTE\*\*** A special restriction of AlphaLisp is that Y may not be numeric!

## 4.21 CONSP

The call (CONSP X) returns X if X is not an atom, NIL otherwise.

## 4.22 COPY

The call (COPY X) returns a copy of X. All non-atomic portions of X are duplicated in storage. COPY is equivalent to (SUBST NIL NIL X).

## 4.23 DE

The call (DE "ID" "ARGS" "BODY") places the form (LAMBDA ARGS BODY) on the property list of ID under property EXPR. If ID previously had any of the properties EXPR, FEXPR, SUBR, FSUBR, or MACRO, then DE will return the list (ID REDEFINED). Otherwise DE returns ID.

## 4.24 DEFPROP

The call (DEFPROP "I" "V" "P") enters the property name P with property value V into the property list of the identifier I. If the property name P is already in the property list, the old value is replaced by the new one; otherwise the new property name P and its value V are placed on the beginning of the property list. DEFPROP returns I.

## 4.25 DF

Same as DE except defines a function with FEXPR property.

## 4.26 DIV

The call (DIV X Y) returns X/Y.

## 4.27 DM

Same as DE except defines a MACRO.

#### 4.28 DREMOVE

The call (DREMOVE X L) is similar to the call (REMOVE X L), but DREMOVE uses EQ instead of EQUAL, and actually modifies the list L when removing X, and thus does not use any additional storage. More efficient than REMOVE.

**\*\*NOTE\*\*** If L = (X ... X) (i.e., a list of any length all of whose top level elements are EQ to L) then the value returned by (DREMOVE X L) is NIL, but even after then destructive changes to X there is still one CONS cell left in the modified list which cannot be deleted. Thus if X is a variable and it is possible that the result of (DREMOVE X L) might be NIL the user must set the value of the variable given to DREMOVE to the value returned by the function.

#### 4.29 DREVERSE

The value of (DREVERSE L A) is EQUAL to (REVERSE L A), but DREVERSE destroys the original list L, and thus does not use any additional storage. More efficient than REVERSE.

#### 4.30 DRM [EXTENDED LIBRARY]

The call (DRM "CHARACTER" "FUNCTION") defines CHARACTER as a Normal Read Macro with "FUNCTION" being a function name or LAMBDA expression of no arguments which will be evaluated each time CHARACTER is detected as a macro during input. FUNCTION is put on the property list of CHARACTER under the property READMACRO. The value of DRM is CHARACTER.

#### 4.31 DSKIN

The call (DSKIN FILESPEC) opens the file specified by FILESPEC for input on channel T, and then READ-EVALS the contents of the file. This is the function to use to read files created by DSKOUT. The value of DSKIN is NIL.

#### 4.32 DSKOUT [EXTENDED LIBRARY]

The call (DSKOUT "FILESPEC" "EXPRSLIST") as used to create an entire output file specified by "FILESPEC". It evaluates all of the expressions in "EXPRSLIST". If an expression in "EXPRSLIST" is atomic, then that atom is given to GRINL instead of being evaluated directly.

For example, if FNLIST is a list of your functions, they can be saved on a disk file FUNCS.LSP by:

(DSKOUT "FUNCS" FNLIST)

#### 4.33 DSM [EXTENDED LIBRARY]

DSM is exactly like DRM except that CHARACTER is defined as a Splice Macro.

#### 4.34 DSUBST

The call (DSUBST X Y Z) is similar to the call (SUBST X Y Z), except DSUBST uses EQ and not copy Z, but changes the list structure z itself. DSUBST substitutes with a COPY of X. More efficient than SUBST.

#### 4.35 EQ

The value of (EQ X Y) is T if X and Y are the same pointer, i.e., the same internal address. Identifiers on the OBLIST have unique addresses and therefore EQ will be T if X and Y are the same identifier. EQ will also return T for equivalent small integers, since they are represented as addresses. However, EQ will not compare equivalent floating point numbers. For non-atomic S-expressions, EQ is T if X and Y are the same pointer.

#### 4.36 EQUAL

The value of (EQUAL X Y) is T if X and Y are equivalent S-expressions, NIL otherwise.

#### 4.37 ERR

The call (ERR E) returns the value of E to the most recent ERRSET, or to the LISP Supervisor if there is no ERRSET. If the value of E is ERRORX, then ERR will return to the most recent ERRSET of the form (ERRSET --- ERRORX). Ordinarily, this will be the most recent top level supervisor call.

## 4.38 ERROR

The call (ERROR E) generates a real LISP error. E is evaluated, and if it is non-NIL, it is PRINCEd (unless error messages are suppressed) and then a break occurs on the expression which invoked the error call.

## 4.39 ERRORX

Special argument to ERR.

## 4.40 ERRSET

The call (ERRSET E "F") evaluates the S-expression E and if no error occurs during its evaluation, ERRSET returns (LIST E). If an error occurs and F = NIL, the error message is suppressed and ERRSET returns NIL, instead of a break occurring. If the function ERR is called during evaluation, then no message is printed and ERRSET returns the value returned by ERR. If F is omitted, it is assumed to be T.

## 4.41 EVAL

The call (EVAL E) evaluates the value of the S-expression E.

## 4.42 EXIT

The call (EXIT) causes termination of LISP and a return to the AMOS Monitor.

## 4.43 EXPLODE [EXTENDED LIBRARY]

The call (EXPLODE L) transforms an S-expression L into a list of single character identifiers identical to the sequence of characters which would be produced by PRIN1.

## 4.44 EXPLODEC [EXTENDED LIBRARY]

The call (EXPLODEC L) transforms an S-expression L into a list of single character identifiers identical to the sequence of characters which would be produced by PRINC.

## 4.45 EXPR

An EXPR is an identifier which has a LAMBDA expression on its property list with property name EXPR. EXPRs are evaluated by binding the values of the actual arguments to their corresponding dummy variables. If there are more actual arguments than dummy variables, the excess arguments are evaluated but ignored. If there are more dummy variables than actual arguments, the excess dummy variables are bound to NIL.

## 4.46 FEXPR

An FEXPR is an identifier which has a LAMBDA expression on its property list with property name FEXPR. FEXPRs are evaluated by binding the actual argument list to the first dummy variable without evaluating any arguments. Any remaining dummy variables are bound to NIL.

## 4.47 FIXP

The call (FIXP N) returns T if N is a small integer, i.e., in the range  $-2^{13}$  thru  $2^{13}$ . Otherwise, FIXP returns NIL.

## 4.48 FLATLE

The call (FLATLE X N) returns (FLATSIZE X) if (FLATSIZE X)  $\leq$  N. Otherwise, FLATLE returns NIL.

## 4.49 FLATSIZE

The call (FLATSIZE X) is equivalent to (LENGTH (EXPLODE X)).

## 4.50 FLATSIZEC

The call (FLATSIZEC X) is equivalent to (LENGTH (EXPLODEC X)).

## 4.51 FLOATP

The call (FLOATP N) returns T if N is a floating point number, NIL otherwise.

## 4.52 FSUBR

A FSUBR is an identifier which has a dispatch number on its property list with property name FSUBR. FSUBRs are evaluated by passing the unevaluated actual argument list to the internal subroutine.

## 4.53 FUNCTION

The call (FUNCTION X) is currently equivalent to (QUOTE X) in AlphaLisp. It is used to prevent the evaluation of a functional argument. In some implementations of LISP, FUNCTION invokes a special mechanism to maintain the proper bindings of free variables in the functional argument.

## 4.54 GET

The call (GET I P) searches the property list of the identifier I looking for the property name P. If such a property name is found, the value associated with it is returned as the value of GET, otherwise NIL is returned. Note that confusion exists if the property is found but its value is NIL.

## 4.55 GETL

The call (GETL I L) searches the property list of the identifier I looking for the first property which is EQ to L or a member (MEMQ) of L. GETL returns the remaining property list, including the property name if any such property was found, NIL otherwise.

## 4.56 GO

The call (GO "ID") causes the sequence of control within a PROG to be transferred to the next statement following the label ID. If ID is non-atomic, it is repeated evaluated until an atomic value is found. GO cannot transfer into or out of a PROG.

## 4.57 GREAT

The call (GREAT X Y) returns T if  $X > Y$ , NIL otherwise.

## 4.58 GRINDEF [EXTENDED LIBRARY]

The call (GRINDEF "F1" "F2" ... "Fn") is used to print the definitions of functions and the values of variables in a format suitable for reading back in to LISP, in what is known as DEFPROP-SETQ format. GRINDEF uses SPRINT to print these S-expressions in a highly readable format, in which the levels of list structure (or parentheses levels) are indicated by indentation. GRINDEF prints all the properties of the identifiers F1 F2 ... Fn which appear on the list GRINPROPS. If Fi is non-atomic, it will be SPRINTed.

## 4.59 GRINL [EXTENDED LIBRARY]

The call (GRINL "F1" "F2" ... "Fn") causes all of the atoms, "F1" "F2" ... "Fn", and all of the atoms on the lists which are the values of the atoms F1 F2 ... Fn to be GRINDEFed. GRINL correctly prints out read macros and is the only function which does. If any of the Fi are non-atomic, they are evaluated but ignored.

## 4.60 GRINPROPS [EXTENDED LIBRARY]

The variable GRINPROPS contains the properties which will be printed by GRINDEF. This variable can be set by the user to print special properties which he has placed on atoms. The initial value of GRINPROPS is (NIL EXPR FEXPR MACRO SUBR FSUBR).



#### 4.61 IASCII

The call (IASCII X) is equivalent to (INTERN (ASCII X))

#### 4.62 IBASE

The variable IBASE specifies the input radix for integers which are not followed by "." Integers followed by "." are decimal integers. IBASE is initially 8.

#### 4.63 INC

The call (INC CHANNEL ACTION) selects the specified channel for input. The channel NIL selects the terminal. If ACTION = NIL, the previously selected input channel is not closed, but only deselected. If ACTION = T, that channel is closed, making it available. INC evaluates its arguments and returns the name of the previously selected channel.

#### 4.64 INITPROMPT

Whenever LISP is forced back to the supervisor, the prompt character is reset. The call (INITPROMPT CHAR) is similar to the call (PROMPT N) except that it sets the supervisor prompt character instead of the current prompt character. The call (INITPROMPT NIL) returns the ASCII value of the supervisor prompt character without changing it.

#### 4.65 INPUT

The call (INPUT "CHANNEL" "FILESPEC") closes any file previously initialized on the input channel, and initializes the file or device specified by FILESPEC for input. INPUT does not evaluate its arguments and returns CHANNEL.

If the file specified by FILESPEC is not found, an attempt is made to find the file on DSK0:[7,4] with extension .LSP with the same file name.

#### 4.66 INTEGER

The call (INTEGER N) returns the greatest integer less than N.

## 4.67 INTERN

The call (INTERN I) puts the identifier I in the appropriate bucket of OBLIST. If the identifier is already in the OBLIST, then INTERN returns a pointer to the identifier already there. Otherwise, INTERN returns I.

## 4.68 LABEL

The form (LABEL "ID" "LAMBDA-EXPR") defines a function with the temporary name "ID". This makes it possible to construct recursive function expressions.

## 4.69 LAMBDA

The form (LAMBDA "ARGUMENT-LIST" "BODY") defines a function by specifying an ARGUMENT-LIST, which is a list of identifiers which are to serve as dummy variables, and a body, which is one or more S-expressions. LAMBDA expressions are evaluated by "binding" actual arguments to the dummy variables, then evaluating BODY with the current dummy variable bindings. If BODY is a single expression, it is simply evaluated. If BODY is several S-expressions, it is interpreted as a PROGRAM (See PROG).

## 4.70 LAST

The call (LAST L) returns a list containing the last element of L.

## 4.71 LCONC

The call (LCONC PTR L) is similar to the call (TCONC PTR X) except that whereas TCONC is used to add elements at the end of a list, LCONC is used for building lists by adding lists at the end. Note that LCONC uses the same pointer conventions as TCONC for eliminating searching to the end of a list, so that the same pointer can be given to TCONC and LCONC interchangeably.

## 4.72 LDIFF [EXTENDED LIBRARY]

In the call (LDIFF X Y), Y must be a tail of X, i.e., EQ to the result of applying some number of CDRs to X. LDIFF gives a list of all elements in X but not in Y, i.e., the List DIFFERENCE of X and Y. Thus (LDIFF X (MEMB FOO Y)) gives all elements in X up to the first FOO.

Note that the value of `LDIFF` is always a new list structure unless `Y = NIL`, in which case `(LDIFF X NIL)` is `X` itself.

If `Y` is not a tail of `X`, `LDIFF` generates an error.

#### 4.73 LENGTH

The call `(LENGTH L)` returns the number of top-level elements of the list `L`.

#### 4.74 LESS

The call `(LESS X Y)` returns `T` if `X < Y`, `NIL` otherwise.

#### 4.75 LEXORDER

The value of `(LEXORDER X Y)` is `T` if `X` is lexically less than or equal to `Y`.  
Note: Both arguments must be atoms, and numeric arguments are lexically less than symbolic arguments. If both `X` and `Y` are numeric, `LEXORDER` returns `NIL`.

#### 4.76 LINELENGTH

The call `(LINELENGTH N)` is used to examine or change the maximum output `linelength`. If `N = NIL`, the current line length is returned unchanged, otherwise the line length is changed to the value of `N`, which is returned as the value of `LINELENGTH`.

#### 4.77 LINEREAD

The call `(LINEREAD)` reads a line, returning it as a list. If the last expression read does not end at the end of the line or is incomplete, `LINEREAD` continues reading.

#### 4.78 LIST

The call `(LIST X[1] X[2] ... X[n])` evaluates the `X[i]` and returns a list of their values.

## 4.79 LITATOM

The call (LITATOM A) returns T if A is an identifier, i.e., a non-numeric atom.

## 4.80 LSUBST

The call (LSUBST X Y Z) is similar to the call (SUBST X Y Z) except that X is substituted as a segment. Note that if X is NIL, LSUBST returns a copy of Z with all Y's deleted.

## 4.81 MACRO

A MACRO is an identifier which has a LAMBDA expression on its property list with property name MACRO. MACROs are evaluated by binding the list containing the macro name and the actual argument list to the first dummy variable. Any remaining dummy variables are bound to NIL. The body in the LAMBDA expression is evaluated and should result in another, "expanded" form, which is then evaluated.

## 4.82 MAKNAM

The call (MAKNAM L) transforms a list of single character identifiers (actually takes the first character of each identifier) and ASCII character codes into an S-expression identical to that which would be produced by READING those characters. MAKNAM however does not INTERN any of the identifiers in the S-expression it produces.

## 4.83 MAP

The call (MAP FN X[1] X[2] ... X[n]) applies the function FN to the argument list X[1] X[2] ... X[n] and to successive CDR's of all the X[i] until one of the X[i] is reduced to an atom or NIL. Note that if any of the X[i] except X[1] are NIL, a circular list of NIL's is substituted for those X[i]. The maximum value of n is 4. The value of MAP is NIL.

## 4.84 MAPC

The call (MAPC FN X[1] X[2] ... X[n]) applies the function FN to successive elements from the X[i] until one of the X[i] become exhausted, i.e., become atomic or NIL. If any of the X[i] except X[1] are NIL, a circular list of NIL's is substituted for those X[i]. The maximum value of n is 4. The value of MAPC is NIL.

## 4.85 MAPCAN

MAPCAN is similar to MAPC except that it returns all the lists produced by FN, NCONCed together.

## 4.86 MAPCAR

MAPCAR is similar to MAPC except that it returns a list of the results produced by FN.

## 4.87 MAPCON

MAPCON is similar to MAP except that it returns all the lists produced by FN, NCONCed together.

## 4.88 MAPLIST

MAPLIST is similar to MAP except that it returns a list of the results produced by FN.

## 4.89 MEMB

The call (MEMB X L) is NIL if X is not EQ to any of the top level elements of L. Otherwise, MEMB returns the tail of L starting at the position where X is found.

## 4.90 MEMBER

The call (MEMBER X L) is similar to MEMB except that it uses EQUAL instead of EQ.

## 4.91 MEMQ

MEMQ is just another name of MEMB.

## 4.92 MINUS

The call (MINUS X) returns -X.

## 4.93 MINUSP

The call (MINUSP X) returns T if  $X < 0$ , NIL otherwise.

## 4.94 MUL

The call (MUL X Y) returns  $X*Y$ , i.e., it multiplies together X and Y.

## 4.95 NCONC

The call (NCONC X Y) is similar in effect to (APPEND X Y) but NCONC does not copy list structures. NCONC modifies list structures by replacing the last element of X with a pointer to Y. The value of NCONC is the modified list X, which is the concatenation of X and Y.

## 4.96 NCONS

The call (NCONS X) is equivalent to (CONS X NIL).

## 4.97 NEQ

The call (NEQ X Y) is equivalent to (NOT (EQ X Y))

## 4.98 NIL

The call (NIL X[1] X[2] ... X[n]) returns NIL without evaluating the X[i]. In general, NIL denotes falsehood, and is equivalent to the empty list "()". NIL is unique in that it is both an atom and a list.

## 4.99 NOT

The call (NOT X) returns T if X is NIL, and NIL if X is non-NIL (e.g., T).

## 4.100 NTH

The call (NTH X N) returns the tail of X beginning with the Nth element, e.g., if N = 2, the value is (CDR X). If N = 0, for consistency, the value is (CONS NIL X).

## 4.101 NTHCHAR

The call (NTHCHAR X N) returns the Nth character in the EXPLODEC representation of X. If N is negative, (NTHCHAR X N) returns the Nth from last character of the EXPLODEC representation of X.

## 4.102 NULL

The call (NULL X) returns T if X is NIL, NIL otherwise.

## 4.103 NUMBERP

The call (NUMBERP X) returns T if X is numeric, NIL otherwise.

## 4.104 OBLIST

In order that occurrences of identifiers with the same print names have the same internal address (and hence value), a special list which is the value of the variable OBLIST is used to remember all identifiers which READ and some other functions have seen. For the sake of searching efficiency, this list has two levels; the first level contains 16 sequentially stored "buckets" which are "hashed" into as a function of the print name of the identifier. Each bucket is a list of all distinct identifiers which have hashed into that bucket. Thus, (CAR OBLIST) is the first bucket, and (CAAR OBLIST) is the first identifier of the first bucket.

## 4.105 ONEP

The call (ONEP X) returns T if X = 1, NIL otherwise.

## 4.106 OR

The call (OR X[1] X[2] ... X[n]) returns the value of the first non-NIL X[i], or NIL if the value of all the X[i]'s is NIL. OR evaluates its arguments from left to right until a non-NIL argument is found, leaving the remaining arguments unevaluated. Note that (OR) returns NIL.

## 4.107 ORP

The call (ORP X) returns the first non-NIL element of X or NIL.

## 4.108 OUTC

The call (OUTC CHANNEL ACTION) selects the specified channel for output. The channel NIL selects the terminal. If ACTION = NIL, then the previously selected output channel is not closed, but only deselected. If ACTION = T, then any buffered output is sent to the previous file, and the file is closed. OUTC evaluates its arguments and returns the name of the previously selected channel.



## 4.109 OUTPUT

The call (OUTPUT "CHANNEL" "FILESPEC") closes any file previously initialized on the output channel, and initializes the file or device specified by FILESPEC for output. OUTPUT does not evaluate its arguments and returns CHANNEL. If a file specified by FILESPEC already exists, it is deleted before the new file is initialized.

## 4.110 PLUS [EXTENDED LIBRARY]

The call (PLUS X[1] X[2] ... X[n]) returns the sum of all the X[i]'s.

## 4.111 PRIN1

The call (PRIN1 S) causes the S-expression S to be printed on the selected output device with no preceding or following spaces. PRIN1 also inserts slashes ("/") before any characters which would otherwise be syntactically incorrect. Double quotes around strings and repeated double quotes within strings are also printed. Two element lists beginning with QUOTE are printed in '@' format (see QUOTE).

## 4.112 PRINC

PRINC is the same as PRIN1 except that no slashes are inserted, double quotes are strings and repeated double quotes within strings are suppressed, and QUOTE appears normally.

## 4.113 PRINT

The call (PRINT S) is equivalent to (PROG2 (TERPRI) (PRIN1 S) (PRINC @/ )).

## 4.114 PROCEED

The call (PROCEED N) RESUMES evaluation of the currently pending expression and causes an interrupt after N expressions are subsequently entered for evaluation.

## 4.115 PROG

The call (PROG "VARLIST" "BODY") specifies a list of program variables, VARLIST, which are initialized to NIL when the PROG is entered, and a body which is a list of labels (which are identifiers) and statements which are non-atomic S-expressions. PROG evaluates its statements in sequence until either a RETURN or GO is evaluated, or the list of statements is exhausted in which case the value of PROG is NIL.

## 4.116 PROGL

The call (PROGL X[1] X[2] ... X[n]) evaluates all the expressions X[1] X[2] ... X[n] (n<6) and returns X[1] as its value.

## 4.117 PROG2

The call (PROG2 X[1] X[2] ... X[n]) evaluates all the expressions X[1] X[2] ... X[n] (n<6) and returns X[2] as its value.

## 4.118 PROGN

The call (PROGN X[1] X[2] ... X[n]) evaluates all the X[i] and returns X[n].

## 4.119 PROMPT

The call (PROMPT N) resets the prompt character displayed by the input routines when a new line is requested, to the character whose ASCII representation is N. If N is NIL, the prompt is left unchanged. The value of PROMPT is the previous value of the prompt character.

## 4.120 PUTPROP

The call (PUTPROP I V P) enters the property name P with property value V into the property list of identifier I. If the property name P is already in the property list, the old value is replaced by the new one; otherwise, the new property name P and its value V are placed on the beginning of the property list. PUTPROP returns V.

## 4.121 QUOTE

The call (QUOTE "E") returns the S-expression E without evaluating it. To improve the clarity of expressions and programs, there exists a concise notation for (QUOTE "E"): @"E". This alternate notation is accepted by the read routines and displayed by all print routines except PRINC.

## 4.122 QUOTIENT [EXTENDED LIBRARY]

The call (QUOTIENT X[1] X[2] ... X[n]) returns  $X[1]/X[2]/\dots/X[n]$ .

## 4.123 RDNAM

The call (RDNAM) is similar to the call (READ) except that identifiers are not INTERNed.

## 4.124 READ

The call (READ) causes the next S-expression to be read from the selected input device, and returns the internal representation of the S-expression. READ uses INTERN to guarantee the references to the same identifier are EQ.

## 4.125 READCH

The call (READCH) causes the next character to be read from the selected input device and returns the corresponding single character identifier. READCH uses INTERN.

## 4.126 READLIST

The call (READLIST L) is identical to (MAKNAM L) except that READLIST INTERNS all identifiers in the S-expression it produces. READLIST is the logical inverse of EXPLODE.

## 4.127 READMACRO

A READMACRO is a single character identifier which has a LAMBDA expression on its property list with property name READMACRO. In order for READ to recognize a character as a READMACRO, the character's internal description must be changed with SETCHR. DRM and DSM are normally used to define READMACROS.

## 4.128 RECIP

The call (RECIP X) returns  $1/X$ .

## 4.129 REMAINDER

The call (REMAINDER X Y) returns the remainder of X divided by Y. X and Y must be small positive integers.

## 4.130 REMOB [EXTENDED LIBRARY]

The call (REMOB "I[1]" "I[2]" ... "I[n]") removes the identifiers I[i] from the OBLIST and returns NIL.

## 4.131 REMOVE

The call (REMOVE X L) removes all top level occurrences of X from the list L, giving a COPY of L with all top level elements EQUAL to X removed.

## 4.132 REMPROP

The call (REMPROP I P) removes the property P from the property list of identifier I. REMPROP returns T if there was such a property, NIL otherwise.

## 4.133 RESET

The call (RESET) clears the suspended evaluation stack and returns control to the top level supervisor.

## 4.134 RESUME

The call (RESUME) resumes evaluation of a suspended expression.

The call (RESUME "E") substitutes E for the suspended expression and resumes execution. If possible, the expression containing the suspended expression is physically modified to substitute E for the suspended expression using DSUBST.

## 4.135 RETURN

The call (RETURN V) causes the most recent PROG to be exited with the value V.

## 4.136 REVERSE

The call (REVERSE L) returns the reverse of the top level elements of the list L.

The call (REVERSE L A) returns (NCONC (REVERSE L) A).

## 4.137 RPLACA

The call (RPLACA X Y) replaces the CAR of X by Y. The value of RPLACA is the modified S-expression X.

## 4.138 RPLACD

The call (RPLACD X Y) replaces the CDR of X by Y. The value of RPLACD is the modified S-expression X.

## 4.139 SELECTQ

The call (SELECTQ X "Y1" "Y2" ... "Yn" Z) is used to select a sequence of instructions based on the value of X (this call is often referred to as a CASE statement in other languages). Each of the Yi is a list of the form (Si E[1,i] E[2,i] ... E[k,i]) where Si is the "selection key".

If Si is an atom the value of X is tested to see if it is EQ to Si (not evaluated). If so, the expressions E[1,i] ... E[k,i] are evaluated in sequence, and the value of SELECTQ is the value of the last expression evaluated, i.e. E[k,i].

If  $S_i$  is a list, and if any element (not evaluated) of  $S_i$  is EQ to the value of  $X$ , then  $E[1,i] \dots E[k,i]$  are evaluated in turn as above.

If  $Y_i$  is not selected in one of the two ways described then  $Y[i+1]$  is tested, etc. until all the  $Y$ 's have been tested. If none is selected, the value of SELECTQ is the value of  $Z$ .  $Z$  must be present.

#### 4.140 SET

The call (SET E V) changes the value of the identifier specified by the expression  $E$  to  $V$  and returns  $V$ . Both arguments are evaluated.

#### 4.141 SETCHR

The call (SETCHR CHAR BITS) allows the internal description for the ASCII character  $CHAR$  to be modified to the value specified by  $BITS$ .  $BITS$  must be the sum of some combination of the following flags:

- 200 Don't slashify character as first character of an identifier.
- 4 Splice option of READMACRO.
- 1 READMACRO.

#### 4.142 SETQ

The call (SETQ "ID" V) changes the value of  $ID$  to  $V$  and returns  $V$ . SETQ evaluates  $V$ , but does not evaluate  $ID$ .

#### 4.143 SPRINT

The call (SPRINT EXPR IND RMARGIN) "pretty prints"  $EXPR$  in a human readable form, with the levels of list structures shown by indentation along the line. The initial indentation is  $IND - 1$  spaces.  $RMARGIN$  is the amount of space to reserve in the right margin for the last line printed in order to print any enclosing tail, and is normally omitted. If  $IND$  is omitted, it defaults to one.

## 4.144 STRINGP

The call (STRINGP X) returns T if X is a string, NIL otherwise.

## 4.145 SUB

The call (SUB X Y) returns X-Y.

## 4.146 SUB1

The call (SUB1 X) returns X-1.

## 4.147 SUBR

A SUBR is an identifier which has a dispatch number on its property list with property name SUBR. SUBRs are evaluated by evaluating up to five actual arguments and passing the results to the internal subroutine. Any additional arguments are ignored, and hence left unevaluated.

## 4.148 SUBST

The call (SUBST X Y S) substitutes S-expression X for all EQUAL occurrences of S-expression Y in S-expression S.

## 4.149 T

The call (T X[1] X[2] ... X[n]) is equivalent to (PROG1 X[1] X[2] ... X[n]). In general, T denotes truth or success.

## 4.150 TAB

The call (TAB N) tabs to position N on the output line doing a TERPRI first if the current position is already past N. Note should be taken that TAB outputs spaces only when necessary and outputs tab characters otherwise.

## 4.151 TAILP

The call (TAILP X Y) returns X if X is a list and a tail of Y (i.e., X is EQ to some number of CDR's of Y), NIL otherwise.

## 4.152 TCONC

The call (TCONC PTR X) is useful for building a list by adding elements one at a time at the end. This could be done with NCONC. However, unlike NCONC, TCONC does not have to search to the end of the list each time it is called. It does this by keeping a pointer to the end of the list being assembled, and updating this pointer after each call. The savings can be considerable for long lists. The cost is the extra cell required for storing both the list being assembled, and the end of the list. PTR is that cell: (CAR PTR) is the list being assembled, (CDR PTR) is (LAST (CAR PTR)). The value of TCONC is PTR, with the appropriate modifications to its CAR and CDR. Note that TCONC is a destructive operation, using RPLACA and RPLACD.

TCONC can be initialized in two ways. If PTR is NIL, TCONC will make up a pointer. In this case, the program must set some variable to the value of the first call to TCONC. After that it is unnecessary to reset since TCONC physically changes PTR.

If PTR is initially (NIL), the value of TCONC is the same as for PTR = NIL, but TCONC changes PTR. This method allows the program to initialize, and then call TCONC without having to perform SETQ on its value.

## 4.153 TERPRI

The call (TERPRI X) prints a carriage-return and line-feed and returns the value of X. X may be omitted if the value of TERPRI is not used.

## 4.154 TIMES [EXTENDED LIBRARY]

The call (TIMES X[1] X[2] ... X[n]) returns the product of all the X[i]'s.

## 4.155 TTYECHO

The call (TTYECHO) complements the terminal echo switch, returning T if echoing is being turned on or NIL if it is being turned off.



## 4.156 TYI

The call (TYI) causes the next character to be read from the selected input device and returns the ASCII code for that character.

## 4.157 TYO

The call (TYO N) prints the character whose ASCII value is N, and returns N.

## 4.158 UNBOUND

The call (UNBOUND) returns the un-INTERNEd atom UNBOUND which the system binds to an identifier to indicate that it currently has no assigned value.

## 4.159 UNTYI

The call (UNTYI CHAR) "unreads" a character (such as a character input by TYI or READCH), so that the next call to READ, TYI, etc., will pick up the UNTYI'ed character as the next character to be read, and returns the ASCII code for that character. NOTE: In the LISP READ routine, an atom may be terminated either by a break character (a character which must be interpreted by READ as well as serving to terminate the atom, such as "(", ")", "[", "]", and ".") or a separator character (a character used only to separate atoms, etc., but not in itself meaningful, such as carriage-return or blank). In order to save a break character for later interpretation, the LISP READ routines use a one-character buffer. UNTYI simply stores its argument in this buffer; thus there are two problems with using TYI. First, if UNTYI is used several times in succession with no intervening READS, TYIs, etc., then only the most recent character is actually "unread" -- all others are lost. Second, if there is a break character in the one-character buffer when an UNTYI is performed, the break character will be lost.

## 4.160 VCONC

The call (VCONC PTR) is used in conjunction with TCONC or LCONC to return the list they have built. VCONC is currently equivalent to CAR. reference by PTR back into the free cell list.

## 4.161 XCONS

The call (XCONS X Y) is equivalent to (CONS Y X)

## 4.162 ZEROP

The call (ZEROP X) returns T if X = 0, NIL otherwise.

## 4.163 #EXPLODE

#EXPLODE is similar to EXPLODE except that it returns a list of ASCII character codes.

## 4.164 #EXPLODEC

#EXPLODEC is similar to EXPLODEC except that it returns a list of ASCII character codes.

## 4.165 #NTHCHAR

#NTHCHAR is similar to NTHCHAR except that the ASCII value of the character is returned.

## 4.166 \$EOF\$

When an end-of-file is detected during input, (ERR \$EOF\$) is executed.

## 4.167 \*EXPAND [EXTENDED LIBRARY]

The call (\*EXPAND L FN) is used within macros to perform macro expansion. \*EXPAND is used by PLUS, TIMES, and QUOTIENT.

Examples:

PLUS can be defined as follows:  
(DM PLUS (L) (\*EXPAND L @ADD))

(PLUS A B C) would be expanded as follows:

```
(*EXPAND @(PLUS A B C) @ADD) = (ADD A (PLUS B C))
(*EXPAND @(PLUS B C) @ADD)   = (ADD B (PLUS C))
(*EXPAND @(PLUS C) @ADD)     = C
```

Thus converting (PLUS A B C) into (ADD A (ADD B C)).

#### 4.168 \*NOPOINT

If BASE = 10, integers will print with a trailing "." unless the variable \*NOPOINT is set to T.

## Bibliography

- Berkeley, E.C., and D.G. Bobrow (eds). "The Programming Language LISP: Its Operation and Applications," The M.I.T. Press, Cambridge, Mass., 1962.
- Friedman, D.P. "The Little LISPer," Science Research Associates, Inc., Palo Alto, Calif., 1974.
- Maurer, W.D. "A Programmer's Introduction to LISP," American Elsevier Publishing Company, Inc., New York, 1973.
- McCarthy, J. "Recursive Functions of Symbolic Expressions and their Computation by Machine," Part I, Comm. ACM,3,4 (1960), 184-195.
- McCarthy, J., P.W. Abrahams, D.J. Edwards, T.P. Hart, and M.I. Levin, "LISP 1.5 Programmer's Manual," The M.I.T. Press, Cambridge, Mass. 1962.
- Quam, L.H. and W. Diffie, "Stanford Artificial Intelligence Laboratory Operating Note 28.6, Stanford LISP 1.6 Manual," Stanford University.
- Ribbens, D. "Programmation non numerique," LISP 1.5, Dunrod, Paris, France, 1970.
- Siklossy, L. "Let's Talk LISP," Prentice-Hall, Inc., Englewood Cliffs, N.J., 1976.
- Weissman, C. "LISP 1.5 Primer," Dickenson Publishing Company, Inc., Encino, Calif., 1966.

