TM 4310/000/00
AUTHOR R. E. Long
TECHNICAL
RELEASE H. Sackman/DD
H. Sackman
for
C. Weissman
DATE 5-27-69 PAGE 1 OF 16 PA

TECH MEMO

a working paper

System Development Corporation/2500 Colorado Ave./Santa Monica, California 90406

(Page 2 blank)

A Table of LISP 1.5/360 Functions & Variables

ABSTRACT

This document lists approximately 200 functions and 30 variables that are available in the SDC LISP 1.5/360 System. LISP 1.5/360 runs under SDC's time-sharing systems on the IBM 360/65 and 360/50. The table of functions and variables specifies section numbers, information about the values of functions and variables, and information about the arguments. For each there is an explanation, comment, and/or reference to a document containing the definition and further information. Since all names in section Ø, the LISP section, are included in the table, the programmer may use this table to avoid a conflict of names. The table also lists some useful functions in other sections, but it does not include all of the system functions which are not in section Ø.

(Page 4 blank)

INDEX OF SYMBOLS FOR VALUE & ARGS

A	An *atom*
CHID	A *character identifier*
FL	A *floating point number*
FN	A *function*
FNAME	The *name* of a *function* (*name* or *name* dotted with *section*)
I	An *integer* (Is means a *LISP SMALL INTEGER*)
ID	An *identifier*
L	A *list*
N	A *number* (any type)
P	*special format list*, CAR points to start of *list*, CDR points to last member.
S	An *s expression*
T/()	T or NIL
VNAME	The name of a *variable* (*name* or *name* dotted with *section*)

When an argument is listed in the table in the form (N...), the function has an indefinite number of arguments.

INDEX OF SYMBOLS FOR KIND

I	An INSTRUCTION
M	A MACRO
S	A *special variable*
number	A FUNCTION of *number* arguments

INDEX OF SYMBOLS FOR REFERENCES

W	LISP 1.5 Primer, by Clark Weissman, Dickenson, 1967.
Q	LISP 1.5 Reference Manual for Q32, by S.L. Kameny, SDC document TM-2337/101/00.
B	The BBN-LISP System, Reference Manual April 1969, by D. G. Bobrow, et.al.
L	LISP Edit Program LISPED, by Kameny & Hawkinson. SDC document TM-2337/100/01.
M	LISP 1.5 Programmer's Manual by McCarthy et.al, MIT Press 1966.
S	The SDC LISP 1.5 System for IBM 360 Computers, by J.A. Barnett and R. E. Long, SDC document SP-3043.
I	The I/O document for SDC LISP 1.5/360, not yet written.
*	This document.
	[Comments are in square brackets]

Name	Section	Kind	Type of Value	Argument type after evaluation of arguments: 1st Arg	2nd Arg	3rd Arg	References
ABSVAL	0	1	N	N			W85
ADDSMALL	0	I	Is	Is	Is'		*13
ADD1	0	1	N	N			W83
AND	0	I	T/()	S	S	(S...)	W78
APPEND	0	2	L	L	L		W97
ARRAYP	0	1	T/()	S	[T for arrays]		
ATOM	0	1	T/()	S			W74
ATTACH	0	2	L	S	L		B6.2
BLANKS	0	1	()	I	[like PRIN of I blanks]		
BLOCK	0	I					S7
CAR CAAR thru CDDR CAAAR thru CDDDR CAAAAR thru CDDDDR CDR	0	1	S	L			W31-7
CHIDP	0	1	T/()	S	[T only for character-identifiers]		
CODE	0	I	S	(S....)			*13
COMPRESS	0	1	A	L			Q67, *14
COND	0	I					W70
CONS	0	2	S	S	S		W30
CONSCOUNT	0	0	N	[number of conses since system start]			
COPY	0	1	S	S	[(COPY(LAMBDA(L)(COND((ATOM L)L) (T(CONS(COPY(CAR L))(COPY(CDR L))))))]		
CSET	0	2	VNAME	VNAME	S		W118
CSETQ	0	M	VNAME	VNAME	S		W118
CAPITALIZE	124	S	T/()	[When T, small letters are capitalized on input]			
COMPRESS	124	2	A	L	T/()/Ø		
DDELETE	0	2	L	A	L	[Destructive delete-in-place, changes L]	
DEFINE	0	1					W66
DEFLIST	0	2					Q69
DELETE	0	2	L	A	L	[Deletes members of L EQUAL to A]	W104,112
DELETEL	0	2	L	L_1	L_2	[Deletes members of L_2 which are members of L_1]	Q69

Name	Section	Kind	Type of Value	Argument type after evaluation of arguments: 1st Arg	2nd Arg	3rd Arg	References
DIFFERENCE	0	2	N	N	N		W83
DIVIDE	0	2	L	N	N		W84
DREVERSE	0	1	L	L [Destructive reverse-in-place, changes L]			
DGET	122	2	L	L	A [Uses EQ, similar to (CAR(FINDN L))]		
DGETQ	122	M	[Quotes 2nd argument for (DGET .122)]				
ENTIER	0	1	I	N			W85
EQ	0	I	T/()	A	A		W75
EQARRAY	0	2	T/()	[Initial value of (EQAF . 122)]			
EQUAL	0	2	T/()	S	S		W75
EQUALN	0	2	T/()	S	S		W70
EVAL	0	1	S	FORM			W61
EVENP	0	1	T/()	N [T only for even integers]			W77
EXPLODE	0	1	L	ID			Q70
EQAF	122	S	[Functional applied by EQUAL to compare arrays]				
EQNF	122	S	[Functional applied by EQUAL to compare numbers]				
ERR	122	1		S			
ERR2	122	2		S_1	S_2 [((ERR . 122)(CONS S_1 S_2))]		
EXCISE	122	1	()	FN			
ESCAPECH	124	S	% [Character which causes error unwind from RATOM]				
FASTCALL	0	I					*14
FIND	0	2	L	S	L [Returns first member of L which has its CAR EQUAL to S, else NIL]		
FINDN	0	2	L	A	L [Returns first member of L which has its CAR EQ to A, else NIL)		
FIRST	0	1	A	S [Recurrs down CAR for first ATOM]			
FIXP	0	1	T/()	S [T only for fixed point numbers]			W76
FLOAT	0	1	FL	N			W85
FLOATP	0	1	T/()	S [T only for floating pt #'s]			W76
FOR	0	M					S9
FOURTH	0	I					*15
FREESPACE	0	0	I [Causes garbage collect, value is the number of free words in node-bps]				
FUNCTION	0	I					W137

Name	Section	Kind	Value	Argument type after evaluation of arguments: 1st Arg	2nd Arg	3rd Arg	References
FUNCTIONP	0	1	T/()	S	[T only for functions]		
FUNCP	0	1	T/()	S	[Used by garbage collector]		
FINDN	122	2	L	A	[L must be a LIST ending in NIL]		
FIXD	122	2	L	L	L		
FILES.	124	S	[List of file descriptor lists for all open files]				
GENSYM	0	0	GENID				W124
GET	0	2	PROPERTY	L or ID	A [Uses EQ, advances with CDR]		W150
GETP	0	2	PROPERTY	L or ID	A [Uses EQ,advances with CDDR]		
GETPROP	0	1	L	ID	[Value is property list of ID]		
GO	0	I					W108
GREATERP	0	2	T/()	N_1	N_2 [T when $N_1>N_2$]		W77
GETA	122	2	S	SYMBARRAY	Is	[Value is the Ith element of the symbolic array]	
GETARRAY	122	3	ARRAY	TYPE	NENT	PRESET	*15
GETC	122	2	CHID	CHARARRAY	Is	[Value is the Ith element of the character array]	
GROWIDS	123	1	NIL	Is	[Grows ARRAY-ID space while shrinking LIST-Binary program space; grows by 256 x Is words]		
GILETTER	124	S	G	[First letter of GENID print names]			
GINUMBER	124	S	[starts at Ø]	[Current number for GENID print names]			
HEXSWITCH	124	S	T/()	[When T, binary #'s print in hex; when (), in octal]			
IDP	0	1	T/()	S	[T for ID's and CHID's]		
INTERSECTION	0	2	L	L	L		
ISDD	124	S	T/()	[Set by RATOM, T for $$ artifacts]			
JUST	0	1	I	N			Q82
LABEL	0	I	[Different from LABEL in LISP 1.5]				S7
LAST	0	1	S	L	[Last member of list]		
LCONC	0	2	P	L	P		B6.2
LEFT	0	I					*15
LENGTH	0	1	I	L			W103,111
LESSP	0	2	T/()	N_1	N_2 [T if $N_1<N_2$]		W77

Name	Section	Kind	Value	Argument type after evaluation of arguments: 1st Arg	2nd Arg	3rd Arg	References
LISPEDIT	0	0					L18
LIST	0	I	L	S	S	(S...)	W58
LOADEXP	0	1	FILENAME				I
LOGAND	0	I	I	N	N	(N...)	W86
LOGOR	0	I	I	N	N	(N...)	W86
LOGXOR	0	I	I	N	N	(N...)	W86
MAP	0	2	L	L	FN		W138
MAPC	0	2	L	L	FN		W140
MAPCAR	0	2	L	L	FN		W139
MAPLIST	0	2	L	L	FN		W138
MAX	0	I	N	N	N	(N...)	W84
MEMB	0	2	L/()	A [Uses EQ		Returns portion of L	B25
MEMBER	0	2	L/()	S [Uses EQUAL		starting with A]	W77
MIN	0	I	N	N	N	(N...)	W84
MINUS	0	1	N	N			W83
MINUSP	0	1	T/()	S			W77
MAPCHECK	122	1	T/()	FN [T if FN is a function of 1 arg]			
MESSAGE	122	1	S	S [Prints S on teletype, if S is an identifier the message is written without using PRINT]			
NCONC	0	2	L	L	L [Changes L]		W147,156
NODEP	0	1	T/()	S [Equivalent to (NOT(ATOM S))]			
NOT	0	I	T/()	S			W78
NULL	0	I	T/()	S			W77
NUMBERP	0	1	T/()	S			W76
ONEP	0	1	T/()	S [T for any type # exactly equal to 1]			W77
OR	0	I	T/()	S	S	(S...)	W78
OPEN	124	2	L	FILENAME	L		I
PLUS	0	I	N	N	N	(N...)	W83
POSITION	0	2	STATUS	FILENAME	ACTION		I
PRINT	0	1	S	S			W125
PRINTCH	0	1	CHID	CHID			Q76
PRINØ	0	1	S	S			Q75

				Argument type after evaluation of arguments			
Name	Section	Kind	Value	1st Arg	2nd Arg	3rd Arg	References
PRIN1	0	1	A	A			W127,Q75
PROG	0	M					W106
PROGN	0	I					W160
PROP	0	3	L	L or ID	A	FN [Uses EQ]	W151
PUT	0	3	ID	ID	A	PROPERTY	W150
PRETTYSWITCH	124	S	T/()	[When T, PRINT will PRETTYPRINT]			
QUOTE	0	I		S			W59
QUOTIENT	0	2	N	N	N		W84
QTIME	122	0	I	[CPU time, since load, in sec/60]			
RATOM	0	0	A				W126
RDS	0	1	OLDFILE	NEWFILE			I
READ	0	0	S				W125
READCH	0	0	CHID				W127
RECIP	0	1	I	N			W85
REMAINDER	0	2	I	N	N		W84
REMPROP	0	2	ID	ID	A [Uses EQ]		W151
REPLACA	0	2	NODE	NODE	S		W146
REPLACD	0	2	NODE	NODE	S		W146
RETURN	0	I					W108
REVERSE	0	1	L	L			W104,112
RIGHT	0	I					*15
ROUND	0	1	I	FL			
RTMG	124	S	I	[The right margin for the currently write-selected file]			
SASSOC	0	3	S	A	L	FN	Q63
SELECT	0	I					W72
SELECTQ	0	I					S8
SETPROP	0	2	ID	ID	L [L becomes the property-list of ID]		
SETQ	0	I	S	VARNAME	S		W107
SPECIAL	0	1	L	L			W119
STRINGED	0	2					L18
STRINGP	0	1	T/()	S [T only for character arrays]			
SUBLIS	0	2	S	L	S		B6.4;M12,61
SUBSMALL	0	I					*13

Name	Section	Kind	Value	Argument type after evaluation of arguments: 1st Arg	2nd Arg	3rd Arg	4th Arg	References
SUBST	0	3	S	S	S	S		Q78;M11,61
SUB1	0	1	N	N				W83
SALN	122	1	I	ARRAY [Length in half-words, excluding header]				
SARP	122	1	T/()	ARRAY [T if ARRAY is of type symbol]				
SETA	122	3	SYMBARRAY	SYMBARRAY	Is	S	[Sets the Ith element of the symbolic array]	
SETC	122	3	CHARARRAY	CHARARRAY	I	CHID	[Sets the Ith element of the character array]	
SIGNON	122	S	S	[Printed by LOOPSUPV after each start]				
STLN	122	1	I	ARRAY [Length in bytes, excluding header]				
SUPF	122	S	FN	[Called by LOOPSUPV after each start]				
SUPV	122	0	[LISP supervisor, is initial value of (SUPF . 122)]					
SWITCHFN	122	4	()	NAME1	SECTION1	NAME2	SECTION2	[Interchanges binary programs]
SHUT	124	1	L	FILENAME				I
SYMSWITCH	124	S	T/()	[When T, PRINT will sym-print]				
TABIN	0	1	Is	Is				I
TABOUT	0	1	Is	Is				I
TCONC	0	2	P	S	P			B6.2
TEDFILER	0	1						L18
TEDSEEKER	0	1						L18
TEREAD	0	0	()					W127
TERPRI	0	0	()					W127
THIRD	0	I						*15
TIMES	0	I	N	N	N	(N...)		W83
TRAPPEDP	0	1	[Used by garbage collector]					
UNION	0	2	L	L	L			W103
UNSPECIAL	0	1	L	L				W124
WRS	0	1	OLDFILE	NEWFILE				I
ZEROP	0	1	T/()	S [T if S is Ø, any type#]				W76

Name	Section	Kind	Value	Argument type after evaluation of arguments: 1st Arg	2nd Arg	3rd Arg	References
DOLLAR	0	S	$				
DOT	0	S	.				
EMPTY	0	S	[The identifier with Ø characters]				
*EQN	0	2	T/()	N	N		Q82
*EQP	0	2	T/()	N	N		Q83
*EXPAND	0	2	FORM	FORM	FNAME		W153 Q31
FUNARG	[Don't use this name in any section!]						
LBRAC	0	S	<				
*LOGAND	0	2	I	N	N		Q80
*LOGOR	0	2	I	N	N		Q80
*LOGXOR	0	2	I	N	N		Q80
LPAR	0	S	(				
*MAX	0	2	N	N	N		Q80
*MIN	0	2	N	N	N		Q80
MINUS	0	S	-				
PERCENT	0	S	%				
*PLUS	0	2	N	N	N		Q80
PLUS	0	S	+				
PRIME	0	S	'				
RBRAC	0	S	>				
RPAR	0	S	)				
SLASH	0	S	/				
SPACE	0	S	0				
*TIMES	0	2	N	N	N		Q80

The sections are assigned as follows:

0	General LISP functions.
1 : : 119	Unused by the system and available for the user (optional).
120	LISPEDIT functions and variables (present in LISPED, absent in LISP).
121	Functions of indefinite number of arguments.
122	LISP system.
123	Garbage Collector.
124	I/O
125	LAP op-codes
126	LAP
127	Compiler

The following paragraphs describe some of the functions and compiler instructions which are perhaps unique to this system.

ADDSMALL and SUBSMALL are special forms. The value is the sum or difference of the two arguments. The calculation is performed by efficient in-line code with no error checking. The first argument may be any form which evaluates to a small integer. The second argument must be a small integer or a variable which evaluates to a small integer. Note that the second argument may not be a function. The value must also be a small integer. Small integers are: $-12{,}288 \leq \text{Is} < 12{,}288$.

Some examples:

```
        LISP                    →              LAP

(ADDSMALL   A      B)                  (L     AC      A)
                                       (A     AC      B)
                                       (S     AC      (QUOTE Ø))

(ADDSMALL   C      2)                  (L     AC      C)
                                       (LA    AC      (8 AC))

(SUBSMALL (FOO)    D)                  (ARGS)
                                       (CALL FOO)
                                       (S     AC      D)
                                       (A     AC      (QUOTE Ø))
```

CODE is a special form for inserting LAP code into LISP functions.

For example:

```
         LISP                   →              LAP

(UNBOX    Y)                           (ARGS)
(SETQ Y                                (L     AC      X)
   (CODE                               (CALL  UNBOX)
      (L   AC  (QUOTE A))              (L     AC      (QUOTE A))
      (AR  AC  ACØ)))                  (AR    AC      ACØ)
(SETQ  Z    Y)                         (ST    AC      Y)
                                       (ST    AC      Z)

(SETQ  A    X)                         (L     AC      X)
(CODE (SR   4   4))                    (ST    AC      A)
(SETQ  B    A)                         (SR     4      4)
                                       (L     AC      A)
                                       (ST    AC      B)
```

Note that in the second example the compiler does not assume that the value of A and B are still in AC after the CODE statement.

(COMPRESS . 124) is the primitive function used for compressing a list of characters into an identifier. This function has two arguments:

((COMPRESS . 124) L B)
L is a list of character-identifiers

If B is
- Ø, the spelling will be checked using (SPELLP . 124);
- NIL, then spelling is assumed incorrect, e.g., 1A;
- otherwise, spelling is assumed to be correct (LISP syntax).

VALUE:
- if L was an atom, value is L;
- if L was a list of character-ids, value is the identifier with a print-name formed by concatenation of the list L;
- if some members of L are not character-ids, then some of the characters in the id will be garbage.

The standard function compress is then defined thus:

```
(COMPRESS(LAMBDA(L)((COMPRESS . 124) L Ø)))
```

When programming, if you know that the spelling is correct, the program will run faster if you use ((COMPRESS . 124) L T) .

FASTCALL is a special form which allows the program to execute a functional without error checking. If X is a variable and the program is:

```
(SETQ X Y)
(X   L)
```

then at execution time, the system will check to be sure that the value of X is a function. An error unwind will occur if X is not a function.

In the program

```
(SETQ X   (FUNCTION CAR))
(FASTCALL X   L)
```

the value of X will be applied as a function, without error checking. (In this case the programmer knows that there can be no error since X was just set to a function.) If an error is made in a FASTCALL, a program interrupt may occur.

(GETARRAY . 122) is the primitive function for creating arrays. It is a function of three arguments:

((GETARRAY . 122) TYPE NENT PRESET)

Type is SYMBOL, STRING, ID, INTEGER, BIT, FLOAT, or TABLE.

NENT is number of entries in the array.

PRESET is the preset value for all entries in the array.
for INTEGER, BIT, and FLOAT the PRESET must be a number.
for SYMBOL and TABLE the preset may be any S-expression.
for STRING and ID the PRESET must be a character-id.
If PRESET is NIL, all bits in the array are preset to Ø, regardless of TYPE.

The value of (GETARRAY . 122) is the newly created array.

Arrays of type SYMBOL and TABLE have symbolic entries, which are marked-from and updated by the garbage collector. The entries are 2 bytes long. The contents of the entries may be obtained and set by (GETA . 122) and (SETA . 122), respectively. The function ((SALN . 122)a) will return the number of entries (halfwords) in the array a, for type SYMBOL or TABLE.

Arrays of type STRING and ID have entries containing EBCDIC characters. Properly spelled identifiers have a print name of type ID; if they are improperly spelled (e.g., $$ artifacts) the print name is of type STRING. The entries are 1 byte long. The contents of the entries may be obtained and set by (GETC . 122) and (SETC . 122), respectively. The function ((STLN . 122)a) will return the number of entries (bytes) in the array a, for type STRING and ID.

Arrays of type INTEGER, BIT, and FLOAT have entries containing numbers (not LISP pointers). The entries for arrays of type INTEGER and BIT are 1 word long. The entries for arrays of type FLOAT are 2 words long. The system presently contains no primitaves for accessing or setting entries in arrays of type INTEGER, BIT, or FLOAT. This may be done by code written in LAP or by writing the primitives in LAP.

LEFT, RIGHT, THIRD, and FOURTH are special forms which produce in-line code for unpacking the contents from the location pointed to by a LISP pointer. LEFT unpacks the left half-word of the location, RIGHT unpacks the right half-word. THIRD unpacks the following half-word, and FOURTH unpacks the half-word after that. There is no error checking. If the argument is incorrect, a program interrupt may occur. When the argument is a node, LEFT and RIGHT are equivalent to CAR and CDR, respectively, but in-line with no error checking.

Some examples:

LISP →	LAP
(LEFT X)	(L AC X) (LH AC (Ø AC SORG)) (NR AC MASK) (SLL AC (2))
(RIGHT (CAR X))	(ARGS) (L AC X) (CALL CAR) (L AC (Ø AC SORG)) (NR AC MASK) (SLL AC (2))
(THIRD X)	(L AC X) (LH AC (4 AC SORG)) (NR AC MASK) (SLL AC (2))
(FOURTH X)	(L AC X) (L AC (4 AC SORG)) (NR AC MASK) (SLL AC (2))

```
(COND ((ATOM X) (FOO X))
      (T (FOO (LEFT  X))))
```

It is faster to use LEFT instead of CAR in the above example. Since the call to ATOM was made, (LEFT X) will only be executed when X is a node.

```
(COND ((EQ X (CAR L)) (GO L1))
      (T(SETQ  L  (RIGHT L)))
```

Here it is safe to use RIGHT instead of CDR, since CAR has already performed a check to see that L is a node. If L were not a node, CAR would cause an error-unwind.