MASSACHUSETTS INSTITUTE OF TECHNOLOGY
PROJECT MAC

ARTIFICIAL INTELLIGENCE
MEMO 93

MEMORANDUM MAC-M-296

A New Version of CTSS LISP

(This System Will Be Temporarily Available)
(As CTEST 8)

by Robert R. Fenichel and Joel Moses

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
PROJECT MAC

ARTIFICIAL INTELLIGENCE
MEMO 93

MEMORANDUM MAC-M-296
February 2, 1966

A New Version of CTSS LISP

by Robert R. Fenichel and Joel Moses

SUMMARY

A new version of CTSS LISP is now available. The new system provides additional data storage and several new functions and constants. The I/O capabilities, EXCISE, the error comments, and several routines have been improved. Much irrelevant code and many bugs have all been removed.

FAP source decks and BCD listings are available. The decks are organized so as to ease the job of assembling private LISP systems in which unneeded features are absent.

Without reassembling, the user can create a private LISP system in which the data storage space has been arbitrarily allocated among binary program space, the push-down list, full word space, and free storage.

It is assumed that the reader is familiar with the old version of LISP as described in AI Memos 67 and 74 (MAC M-153 and M-206)

Incompatibilities between the Old and the New Versions

1) The EVALQUOTE and the LOAD routines read files with secondary name LISP instead of DATA.

2) The PUNCH routine writes files with secondary name LSPOUT instead of OUTPUT.

The above changes were made in order to obtain special treatment from other CTSS programs such as ROUEST PRINT.

3) The character (14)8 (dash) is no longer equivalent to the minus sign, although the apval for the minus sign is still DASH. This equivalence was originally introduced as a convenience for users of the card punch equipment.

4) The character-reading functions ADVANCE, STARTREAD, and ENDREAD access the console instead of the disk.

5) Names of the new system subroutines may conflict with names of functions previously defined by the user.

6) Routines which modify absolute locations of the old LISP system will not have the same effect on the new one.

Features of the New System

The LISP Command

Command	Procedure
LISP	The program will execute LISTEN after typing LITHP IS LITHTENING
LISP 'NOCOMT'	The system will execute LISTEN without typing the message above.
LISP '$$' ...	The system reads the remainder of the command arguments and gives them to EVALQUOTE to be executed. The command should end with a STOP. This feature is intended to be used in conjunction with the CHAIN program.
LISP FOO	The file FOO LISP is read and evaluated. Note that the old version read the file FOO DATA.

I/O Modifications

1) If a disk error occurs the new version will generally print the following line:

```
DISK ERROR. SAVE AND PRNTER, OR START GIVEUP
```

If the user executes

```
START 'GIVEUP'
```

then the message below will be written.

```
*** ERROR CALLED
-
```

and the system will revert to the top level as with any other error.

However, if the user executes

```
SAVE FOO T
PRNTER
```

then the comments printed by PRNTER will enable him to decide whether he can change his files in such a way that the error will not occur a second time. If such a change can be made then

```
RESUME FOO
```

will retry the offending I/O call and continue the program from that point. Otherwise
```
RESUME FOO 'GIVEUP'
```
will have the same effect as the START 'GIVEUP' above.

Exceptions to the I/O error procedure given above:

FILESEEK(A B)	File A B is closed and then reopened. Any errors occurring during the closing of the file are ignored.
FILEDELETE(A B)	All errors occurring while deleting the file are ignored.
FILEAPND(A B)	If file A B is nonexistent this acts as if it were FILEWRITE(A B)

2) The disk reading routines will automatically place one blank followed by EOF))))) following the last word of a file. EOF is recognized by the EVALREAD, LOAD and EVALQUOTE programs as having the same effect as a STOP. The advantage of EOF over STOP is that it is not physically present in the file and thus allows files to be FILEAPNDed or COMBINed easily.

3) Squashed files can be read in the new system. Their advantage over card-image files is that they require less track space and are therefore read faster by the system. Squashed files may be freely intermixed with card-image files and console I/O except that a file formed by COMBINing card-image and squashed files may not be read. A card-image file with atom names split across column 72 should not be squashed.

4) The filewriting routines produce files in squashed form. Line length varies from 2 to 19 words and averages about 12 words. Atoms are not split over line boundaries.

5) The files written by PUNCH have secondary name LSPOUT instead of OUTPUT.

6) Files are read a record (432 words) at a time. This causes an approximate three-fold increase in speed.

7) The character reading functions ADVANCE, STARTREAD, and ENDREAD access the console instead of the disk.

8) The character (11)8 (dash) was formerly equivalent to the minus sign. This equivalence has been removed since most users employ a console which does not contain a dash character. The apval DASH still has as value a minus sign.

9) The alphabetic apvals PRIME('), TYTAB(tab), CR(carriage return), and COLON(:) have been added. ' and : are class A characters. Tab and carriage return are class C characters

like comma and blank (See Appendix F of the LISP manual).

Added Functions

CLOCK(X)	CLOCK(NIL) resets an interval timer and has value 0. CLOCK(*T*) has as value the number of tenths of seconds the program has spent in execution since the last time CLOCK(NIL) was called. If the program goes into the dormant state, by quitting, say, then the timer will stop incrementing; it will not restart until CLOCK(NIL) is explicitly given again.
EXPLODE(X)	Results in a list of the characters which would be printed by PRINT(X).
FILEGONE(A B)	has value NIL if file A B exists, *T* otherwise.
PUNCHP()	causes all further disk output to be in permanent mode. This is the default mode.
PUNCHT()	causes further disk output to be in temporary mode.
RPLACW(A B)	is equivalent to RPLACD(A CDR(B)) followed by RPLACA(A CAR(B)).
RPSREMAINDER()	yields the number of words remaining in binary program space as a LISP fixed point number.
FSLEFT	is an apval whose value is the number of words of free storage collected during the last call to the garbage collector.
FWLEFT	is an apval whose value is the number of full words collected during the last call to the garbage collector.
EXPT(A B)	EXPT has been completely rewritten. It is faster, shorter and more accurate than before. If A is negative and B is a floating point number, then the absolute

value of A is used.

EXPSUB(A) — This routine is an entry to the EXPT routine. It evaluates e-to-the-A. It does not handle LISP numbers. Its inputs and outputs are floating-point numbers in the AC.

LOGSUB(A) — Computes log(A) to base e. It does not handle LISP numbers. Its inputs and outputs are floating-point numbers in the AC.

DIVIDE CHECKS — System has been changed to ignore divide checks.

FLOATING POINT TRAP — The system has been changed to give no error when a floating point trap occurs. Instead the standard FORTRAN routine is used to restore the AC and MQ.

NUMVAL(X) — If X is a LISP number then NUMVAL yields pointers to the value of X in the AC and $FIX or $FLOAT in the MQ. Otherwise it yields *** ERROR NUMVAL.

Changes to System Functions

SUBLIS(X Y) — A new SUBLIS (see AI Memo 74) is in the system. It is an EXPR, and if compiled will be faster than the old SUBLIS. Bugs in the Hart-McCarthy version have been removed. The routine uses the subroutine SUBLIS1.

LITER,OPCHAR — These routines are now EXPRs.

The routines above can be removed by the user.

UNSATISFIED CONDITIONAL — The old error *A 3* has been removed. The value of an unsatisfied conditional is NIL. Error *C 1* - unsatisfied conditional in compiled functions - has also been removed.

Compiler Bugs

The following bugs are known to exist in the compiler and have not been corrected at present.

1) The following form compiles incorrectly
(COND(FOO(SETQ X (COND ...))))

2) (COND((NOT(OR(EQ...)(EQ ...))) ...) ...) also compiles incorrectly.

EXCISE

A new EXCISE has been put into the system. This EXCISE eliminates some of the bugs present in the old version of LISP and it effectively REMOBs many objects of value only to the compiler. Free storage gained by EXCISE(*T*) has been increased by 1900 words.

On excising the compiler the following functions will become undefined: COMPILE, COMMON, UNCOMMON, SPECIAL and UNSPECIAL. Upon excision of LAP all the instructions commonly used are REMOBed along with LAP, SALIST and OPDEFINE.

EXCISE is now an EXPR. IT SHOULD NOT BE COMPILED.

Modification of Storage Parameters

There are four storage parameters in LISP. These are the lengths of free storage, full word space, push down list and binary program space. In both the old and new versions the first three have the values 24000, 4220 and 5000 (octal) respectively. Binary program space in the new system is approximately 15500 (octal) locations long which includes an effective increase of 900 (decimal) locations.

In order to modify these parameters execute the following:

```
LINK PARAM BCD T302 2517
RUNCOM PARAM N1 N2 N3 SYSTEM
```

where

N1= number of words on the push down list in octal
N2= number of words in full word space in octal
N3= number of words in free storage in octal
SYSTEM= name of the new LISP system

The new system will be created with these storage parameters. Length of binary program space will be the difference between the total space available and the sum of the three parameters given.

Creating a New System

It is possible to assemble a private version of the LISP system which has more storage available due to the deletion of certain routines. The authors should be consulted before this is attempted. One can gain about 500 (decimal) words by deleting the punch routines, about 200 by deleting the ARRAY feature and approximately 325 words by deleting the floating point routines.

ERROR

The error routine has been rewritten to yield error comments with more mnemonic content. The error message is of the form

*** ERROR FOO

where FOO is to be found below. Refer to pages 32-34 of the manual.

OLD	NEW	DESCRIPTION
A 1	CALLED	APPLIED FUNCTION CALLED ERROR
A 2	UNDFFN	UNDEFINED FUNCTION- APPLY
A 3	removed	CONDITIONAL UNSATISFIED (see above)
A 4	SETONV	SETQ GIVEN ON NONEXISTENT PROGRAM VARIABLE
A 5	SETUVR	SET GIVEN ON NONEXISTENT PROGRAM VARIABLE
A 6	UNDEGO	UNDEFINED GO
A 7	TMNARG	TOO MANY ARGUMENTS
A 8	URNDVR	UNBOUND VARIABLE - EVAL
A 9	UNDFCN	UNDEFINED FUNCTION - EVAL
C 1	removed	UNSATISFIED CONDITIONAL IN COMPILED FUNCTION
CH 1	PKG120	PRINT NAME TOO LONG - PACK
CH 2	NUMOB	FLOATING POINT NUMBER OUT OF RANGE- NUMOB
CH 3	removed	TAPE READING ERROR
F 1	CNSCTR	CONS COUNTER TRAP
F 2	A1LSA2	FIRST ARGUMENT LIST SHORTER THAN SECOND
F 3	A2LSA1	SECOND ARGUMENT LIST SHORTER THAN FIRST
F 5	STRTRP	STR TRAP
G 1	removed	FLOATING POINT TRAP OR DIVIDE CHECK
G 2	PDLOUT	OUT OF PUSH DOWN LIST
GC 1	removed	FATAL ERROR - RECLAIMER
GC 2	NOROOM	NOT ENOUGH WORDS COLLECTED - RECLAIMER
I 1	NARYRM	NOT ENOUGH ROOM FOR ARRAY
I 2	removed	FIRST ARGUMENT NEGATIVE - EXPT
I 3	NUMVAL	BAD ARGUMENT - NUMVAL
I 4	FIXVAL	BAD ARGUMENT - FIXVAL
L 1	LAPORG	UNABLE TO DETERMINE ORIGIN - LAP
L 2	BPSOUT	OUT OF BINARY PROGRAM SPACE
L 3	LAPSMB	UNDEFINED SYMBOL - LAP
L 4	LAPSUB	FIELD CONTAINS SUB-SUBFIELDS - LAP
O 1-7	removed	OVERLORD ERRORS
P 1	PINATM	PRIN1 ASKED TO PRINT NON-OBJECT
R 1	READER	CONTEXT ERROR IN READING
R 2	READER	CONTEXT ERROR IN READING
R 3	ILCHAR	ILLEGAL CHARACTER
R 4	removed	END OF FILE ON READING
R 5	NCHG30	PRINT NAME GREATER THAN 30 CHARACTERS
R 6	NMTBIG	NUMBER TOO BIG IN CONVERSION

TRAC(L) a Tracing Function for LISP (by A. Guzman)

TRAC provides a printout of the values of the arguments and the functions contained in the list L, each time these functions are called.

Features

1) We may omit the printing of certain or all of the arguments of the traced function, and/or of its value. We may ask to have certain arguments punched on the disk, some other printed on the console, some on both, and still omit some others. The same options exist for the value of the function. This is a convenience when some arguments are very long, or their values are already known or are irrelevant.

2) We may print the value of any other variable which we know is bound at the time the traced function is entered.

3) The depth of recursion is also printed.

The ability to send output to the disk permits us to have a rather large number of S-expressions stored (by TRAC) on the disk, which may be printed later, and at the same time it permits us to have a reasonable amount of output on-line, on the console.

Usage

TRAC ((NAME1 NAME2 ... NAMEn)) will produce normal tracing of the functions NAME1, NAME2,..., that is, the arguments and the value will be printed on the console only.

TRAC ((NAME1(I V1 I1 V2 I2,...,Vn In) NAME2 ...)) indicates that we also want the values of the additional variables V1, V2,... Vn; the Is to their right are atoms which specify the destination of these values; Ik may be P(print), D(punch on disk),PD(both print and punch) or N(neither). I, the first indicator, tells us what to do with the normal tracing of the function; that is, this first atom decides to which output device(s) we will send the arguments and the value of the function. For example, TRAC((FOO(PD X P Y D) FIE (D))) indicates that the arguments and the value of FOO will be sent to the disk and will also appear on the console; in addition, we want to print (P) the variable X, and we want to send to the disk (D) the variable Y. We also want the arguments and the value of FIE to be punched onto the disk(D), and we do not specify any extra variables for it.

TRAC((NAME1 ((A1 A2 ...)I V1 I1 V2 I2 ...)NAME2 ...))
This example is handled as before, but now the additional list to the left of I tells TRAC to consider

NAME1 as a function only of the arguments A1, A2, ... ; again, I will indicate to which output device(s) these arguments and the value of NAME1 are sent.

If we are interested only in the value of a function: TRAC((NAME1 (() P)))

The output on the disk appears in the file TRAC OUTPUT, which is created in temporary mode; it may be printed on-line using the PRINT command.

UNTRAC((NAME1 NAME2,...,NAMEn)) stops the tracing of the functions named. One may now trace them again, perhaps with different combinations.

Description

TRAC alters the definition of the functions being traced, adding print statements in suitable places. UNTRAC suppresses these modifications to the definitions.

If a function is a SUBR, TRAC asks for a list of its arguments. Then it defines in place of this function a dummy one, and proceeds to trace the latter.

TRAC uses the property list of the atom TRACE1, under which it stores the numbers which indicate the depth of recursion.

TRAC is obtained by linking to TRACE2 LISP T316 4170, and then executing LOAD((TRACE2)).

Acknowledgements

The features of this system are a product of the suggestions of several people. Lewis Norton and Warren Teitelman deserve special mention because they greatly aided in the debugging of the system. Dan Edwards designed some of the I/O modifications, Tim Hart rewrote the EVALQUOTE operator and William Martin wrote the EXPLODE routine.