MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Project MAC

Artificial Intelligence Project
Memo 67 (Revised)

Memorandum MAC-M-153 (Revised)

REVISED USER'S VERSION

Time Sharing LISP

by William Martin and Timothy Hart

1. Introduction

This memo describes changes to the LISP system by several people. The changes reduce printout and give the user more control over it. They also make it possible for LISP to communicate with the teletype and the disk. The last sections describe programs available in the public files which are useful for printing, editing, and debugging LISP functions.

This system and the supporting programs allow the user to quickly assemble that combination of functions and data needed for a particular job. This makes it unnecessary to keep duplicate copies of functions in different combinations. The writing of functions to handle rare cases can be postponed by intercepting calls to these functions with breakpoints. Breakpoints are also useful for restricting recursion during debugging and to avoid a long trace printout before an error condition, which is known but not understood, is reached. Certain errors in syntax will be found by the print program and most conceptual errors are readily corrected with the edit program.

2. CTSS Command

LISP X — This command will cause the LISP system to read and evaluate file x DATA. If x is not supplied, the system will read LISP DATA. A convenient way to operate is to let file LISP DATA contain the single pair LISTEN NIL followed by STOP. LISTEN is described in the next section.

3. LISP Functions for Communication with the Teletype

rdflx[ ] — The value of RDFLX is one S-expression read from the teletype.

listen[ ]	LISTEN accepts doublets for evalquote from the teletype. It prints out their value. If an error occurs, LISTEN waits for a new doublet. When it receives STOP it returns NIL as its value.
backtrace[ ]	Prints the backtrace at the time of the last error, unless there has been a garbage collection after the error.
gcgag[x]	If x is NIL, there will be no garbage collector printout. The g.c. is initially silenced.
comprint[x]	If x is NIL, the compiler printout is reduced. The compiler is initially silenced.
setbk[ ]	After this function has been executed, an interrupt signal will cause error[$$$ $] to be executed and the break to be reset. SETBK can only be used to a depth of 3. One should combine this with ERSETQ. An initial setbk has been performed.
savbk[ ]	Negates SETBK.
ersetq[form]	Returns value NIL if eval [form;alist] causes an error. Returns list [eval[form;alist]] otherwise. It is used with SETBK. The alist used is the alist current when ERSETQ is called.
eα	Returns eval[α;alist]. This is useful on the evalquote level where it allows one to type the doublet E(CONS X Y) instead of EVAL((CONS X Y) NIL).
print[x]	Prints the S-expression x on the teletype.

4. LISP Functions for Communication with the Disk

a. Reading

fileseek[x;y]	FILESEEK opens disk file x y for reading. Its value is NIL.
read[ ]	The value of READ is the next S-expression read from the currently open file.

fileendrd[x;y]	Closes file x y, which was opened by FILESEEK.
evalread[x;y;z]	Reads doublets for evalquote from file x y until STOP is read. The entire file is read before any doublets are evaluated. If z is SPEAK, the values of the doublets are printed out, otherwise not. This may be used recursively, i.e., the file being evalread may use EVALREAD.
load[$\alpha_1\alpha_2$...]	Performs evalread[α_i;DATA;NIL] for each i.
run[x]	evalread [x; DATA; SPEAK]
execute[x;y]	EXECUTE first undefines the functions of the INLIST described in section 5. It then performs evalread [x;DATA;NIL]. It next evaluates the function x with argument list y. It returns this value after reading the files on the INLIST back in.

b. Writing

filewrite[x;y;z]	Creates a file x y containing an S-expression z.
fileapnd[x;y;z]	Appends S-expression z to file [x y].
filedelete[x;y]	Deletes file x y.

5. Internal Changes to LISP

1. ERROR does not print the index registers. Use a PM for index 4.
2. ERROR does not print a backtrace.
3. The arguments of EVALQUOTE are not printed.
4. The cons counter is turned off.
5. Every file brought in by LOAD or EVALREAD is examined for DEFINE statements. A list of the file's name and the functions defined in it is attached to the constant INLIST.

6. LISP Functions for Debugging

break1[x;y;z]	The third argument is optional. If x evaluates to true, then either BREAK or, if a third argument is given, the evaluation of z is printed. BREAK1 then listens to the teletype for S-expressions. If STOP is received, it returns the evalu-

ation of y as its value. If RETURN α is received it returns the evaluation of α. Any other S-expression received is evaluated and printed; BREAK1 then listens for another. Note that expressions can be executed for their effect.

break[x;y;z] — Redefines EXPR x = (LAMBDA (X_1X_2...) FORM) to be (LAMBDA (X_1X_2...) (BREAK1 y FORM z)).

unbreak[X] — Redefines X to negate BREAK.

7. Printing

There is a file PRINT DATA for printing function definitions in a nice format. In this file there is a function prettyprint[e] which will print out the definitions of the EXPR or FEXPR defined functions mentioned in the list e. The function printd[file] will print the definitions in the file with first name file if it has been read in.

8. A LISP Program to Edit LISP Functions in Time-Sharing

Introduction

This program makes it possible to debug LISP functions without returning to the system monitor. An updated version of the define statements of files which contain these functions and which have been read into the LISP system is automatically kept on the disk.

The program consists of several functions which are described below. The main function, EDIT, tears apart and rebuilds the EXPR of the function to be edited. EDIT is directed by a series of one word commands. In the explanation, we let u be the S-expression which EDIT holds. This editor is contained in a file called EDIT DATA.

Functions

edit[α] — atom[α] $\rightarrow$ u := get[α;EXPR]
T $\rightarrow$ u := get[car[α]; cadr[α]]
Edit then accepts the following commands.

A	u := car[u]
D	u := cdr[u]
PRINT	u is printed
MARK	u is marked for return from within u.

RETURN	u is reconstructed until a mark is reached.
CHAIN	The car-cdr chain leading to u is printed.
an integer n	n con's are performed toward reconstructing u.
APPEND α	u:= append[α;u]
DELETE n	The n'th sublist in the top level of u is deleted.
LIST n	The first n elements of u are listed.
DROP	The cons above u is deleted.
SUBST α β	u:= subst [α;β;u]
USE α	u:= subst [u;EXPRESSION;α]
CONS α	u:=cons [α;u]
MATCH α	Edit finds the expression α within u, keeping track of its car-cdr address. u:=α. An expression (A (B C D) E F) may be abbreviated (A (B CUT) CUT).
STOP	Edit returns FINISH and does not redefine the function.
RESTORE	Edit totally reconstructs u and redefines the function. It also creates or updates disk files. For every file α B which was read in and which defined the function being edited, a file α EDIT with all function defined in α B as they exist in the LISP System at that time is created or updated. Edit returns FINISH.
DEFINE	DEFINE is the same as RESTORE except that it does not update the disk.
OK	If edit receives an incorrect command, or it cannot carry out a MATCH or DELETE command it reports this and then ignores all further commands until it receives OK.
output[α;B;γ;β]	Creates disk file α B which contains the functionc defined in $\gamma\beta$, if $\gamma\beta$ has been read in.
filelistadd[$\alpha_1 \ldots \alpha_m$);b;γ]	Adds functions α_i to those defines in b γ and creates file b EDIT. It does this by altering the constant INLIST.

Comment

Since EDIT works with S-expressions it is poor for correcting parenthesis errors. Most other corrections can be expressed by typing a short string of commands on one line and then proceding to the next example. At first some of the strings are conceptually difficult to produce, but as the user learns the grammar they become much easier. Thus, EDIT is best for those who make extensive use of LISP.

Example

The following output illustrates some of the edit commands.

```
resume lispc ty
WAIT,
 VALUE
NIL
1.  load ((edit trnk))
    NIL
2.  edit (trnk1) print
    (LAMBDA (X Y) (COND ((NULL X Y) (T (CONS (EPLS
    (CAR X) (CAR Y)) (TRNK1 (CDR X) (CDR Y))))))
3.  match epls 1 print
    (EPLS (CAR X) (CAR Y))
4.  mark d print
    ((CAR X) (CAR Y))
5.  match car 1 subst (car expression) return print
    (EPLS (CAR (CAR X)) (CAR Y))
6.  restore
    FINISH
7.  edit(trnk1) match t 1 print
    (T (CONS (EPLS (CAR (CAR X)) (CAR Y)) (TRNK1 (CDR X) (CDR Y))))
8.  cons (x y) print
    ((X Y) T CONS (EPLS (CAR (CAR X)) (CAR Y)) (TRNK1  (CDR X) (CDR Y))))
9.  d drop 1 print
    ((T (CONS (EPLS (CAR (CAR X)) (CAR Y)) (TRNK1 (CDR X) (CDR Y)))))
10. stop
    FINISH
```

1. We read in the files edit data and trnk data.
2. We want to edit the function TRNK1 defined in trnk data. Edit grasps its definition.
3. We find a car-cdr chain down to EPLS, then return one cons and print.
4. We mark our place and take cdr of the expression.
5. We find the first CAR, go back one to get (CAR X) and then substitute to get (CAR (CAR X)).
6. We are finished with TRNK1.
7. We match T and return 1.
8. (X Y) is cons'd to this expression.
9. We take cdr and then drop the (X Y). Going up a level and printing, we see that the (X Y) is gone.
10. We do not wish to redefine the function.

Massachusetts Institute of Technology
Cambridge, Massachusetts
Project MAC

Artificial Intelligence Project
Memo 67

Memorandum MAC-M-153
April 15, 1964

A D D E N D U M

TEMPORARY USER'S VERSION

Time Sharing LISP

by William Martin and Timothy Hart

The LISP System described in this memorandum is being tested and debugged. This is a list of differences which are currently noted between the system described in the memorandum and the system currently available.

1. The CTSS command is CTEST1 not LISP.

2. The input pairs for evalquote are taken from the file CTEST1 DATA. If a "saved" file is made from CTEST1 and named X SAVED, then the input for this file will be the file X DATA.

3. No initial setbk has been performed.

CS-TR Scanning Project
Document Control Form

Date : 11/30/95

Report # AIM-67

Each of the following should be identified by a checkmark:
Originating Department:

- [x] Artificial Intellegence Laboratory (AI)
- [ ] Laboratory for Computer Science (LCS)

Document Type:

- [ ] Technical Report (TR)
- [x] Technical Memo (TM)
- [ ] Other:_____

Document Information

Number of pages: 8 (12-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:

- [x] Single-sided or
- [ ] Double-sided

Intended to be printed as :

- [x] Single-sided or
- [ ] Double-sided

Print type:

- [ ] Typewriter
- [ ] Offset Press
- [ ] Laser Print
- [ ] InkJet Printer
- [ ] Unknown
- [x] Other: MIMEOGRAPH

Check each if included with document:

- [ ] DOD Form
- [ ] Funding Agent Form
- [ ] Cover Page
- [ ] Spine
- [ ] Printers Notes
- [ ] Photo negatives
- [x] Other: ADDENDUM

Page Data:

Blank Pages(by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description :	Page Number:
IMAGE MAP: (1-8) PAGES #'ED 1-7, ADDENDUM	
(9-12) SCANCONTROL, TARGETS (3)	

Scanning Agent Signoff:

Date Received: 11/30/95 Date Scanned: 12/6/95 Date Returned: 12/7/95

Scanning Agent Signature: Michael W. Cook

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned

Date: 12/6/1995

M.I.T. Libraries
Document Services