TECH MEMO

a working paper

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

TM- 4310/100/00

AUTHOR John F. Burger

TECHNICAL Robert E. Long

RELEASE Harold Sackman

for Clark Weissman

DATE 12-1-69 PAGE 1 OF 19 PAGES

DRAFT UPDATE 3/1/72
(Author Delivered) R.E.L.

LISP EDIT Program, LISPED Users Guide

ABSTRACT

This document describes the LISPED program, which is a context editor for LISP data and exists as a separate program in the programmer's package of the time-sharing system.

TABLE OF CONTENTS

TABLES

FIGURES

1. INTRODUCTION

LISPED is a context editor for LISP 1.5 data and programs. It exists in the programmer's package of TSS as a separate version of LISP, and is made up of approximately fifty LISP functions used by the three modes of operation: LISPEDIT, EDIT, and EVALQUOTE.

This paper is adapted from TM-2337/100/00, a similar paper on Q-32 LISPED, written by Lowell Hawkinson, who programmed the initial versions of LISPED for the Q-32 Time-Sharing System. LISPED for the 360 systems was adapted from this earlier program by Robert E. Long and John F. Burger.

An understanding of this document and of the use of the LISPED program presupposes an understanding of LISP 1.5.

2. LISPED MODES

LISPED operates in three modes. LISPED and EVALQUOTE modes are described in the remainder of Section 1. EDIT is described in Section 2. Within the descriptions, the following notations will be utilized.

f means LISP library file name. A file name is any LISP literal atom. It serves no other purpose than to identify a particular file.

ℓ means either a single file name or a list of file names enclosed in parentheses.

LISPEDIT

This is the normal mode in which LISPED is entered. Any error encountered within LISPEDIT returns to LISPEDIT mode.

EVALQUOTE

The EVALQUOTE mode is similar to the normal mode of operation of 360 LISP, operating successive pairs of S-expressions and printing out the results. EVALQUOTE is entered from LISPEDIT by the command Sure, and continues until either EXIT or LISPEDIT is given as the first S-expression to EVALQUOTE, in which case LISPEDIT mode resumes. An error occurring in EVALQUOTE mode returns to EVALQUOTE mode.

EDIT

The EDIT mode is entered from LISPEDIT by means of the commands INPUT, EDIT or STRINGED. The functions available in EDIT mode are described in Section 2. EDIT mode continues until the EXIT command is accepted by EDIT (either the FILE command followed by EXIT, or else EXIT EXIT must be used). An error occurring within EDIT returns to EDIT mode.

The system is relatively foolproof in that library files are protected from damage in the event of LISP unwind. After an unwind, the system always returns to the same state in LISPEDIT, EVALQUOTE, or EDIT which it was in before the error occurred.

LISPED is, in general, a talkative system and most of the messages printed out have a relatively simple interpretation.

2.1 LISPEDIT

LISPEDIT is the normal mode in which LISPED is entered. If the system is clear, the entrance into LISPEDIT is signified only by the statement

LISPEDIT MODE //

If the system is re-entered following an error, the statement

LISPEDIT RECOVERY // FILES SAVED

is given additionally to indicate that LISPEDIT is ready for input.

The LISPEDIT mode may be used for inputting and editing LISP data, reading LISP data from tape and disc, performing general file maintenance operations, and running and testing of LISP programs.

The LISPEDIT commands are listed in Table 1 and described in Section 2.2.

2.2 LISPEDIT COMMANDS

LISPEDIT accepts the 16 commands given in Table 1. Commands other than OPEN and SHUT are followed by their arguments without parentheses. A final space must always be used except in the case of a list type argument.

OPEN (fdescr)

OPEN works in LISPEDIT mode exactly the same way it works in 360 LISP Release 30 or in EVALQUOTE mode of LISPED.

Files opened in either EVALQUOTE mode or LISPEDIT mode can be read in either mode.

In LISPEDIT mode, the response to OPEN is

IO FILE filename OPENED // CONTINUE

SHUT (filename)

SHUT works in LISPEDIT mode exactly the way it works in 360 LISP Release 30 or in EVALQUOTE of LISPED.

Files opened in either LISPEDIT mode or EVALQUOTE mode can be shut in either mode.

In LISPEDIT mode, the response to SHUT is

I/O FILE filename SHUT //CONTINUF

For both OPEN and SHUT see also TM-4310/200/00, the standardized LISP 360 I/O.

READ fdescr

Reads the file fdescr and adds the contents to the list of current library files. If fdescr is an atom it refers to a tape or disc file already opened with the OPEN command in either LISPEDIT or EVALQUOTE mode. Otherwise fdescr is assumed to be a standard LISP file descriptor list, in which case the file is opened, read from, and shut, with appropriate messages printed on the user's terminal at each stage.

During the actual READ stage, in either case, LISPEDIT prints the name of each library file as it is read, both to keep the user abreast of the progress of the read function, and to make the system run slightly faster.

WRITE fdescr l

If l is any atom (e.g., "ALL") then all current library files are written onto fdescr.

If l = $(f_1\ f_2\ \ldots\ f_n)$, where f_1, f_2, ... f_n are names of library files, then the library files f_1, f_2, ... f_n are written, in that order, onto the output file fdescr.

As each library file is written, its name is printed on the user's terminal, and after writing the output file, WRITE writes an end-of-file on the output device.

If fdescr is an atom ~~other than "DLO"~~ it refers to a tape or disc file already opened with the OPEN command in either LISPEDIT or EVALQUOTE mode. If fdescr is not an atom it is assumed to be a standard LISP file descriptor list, in which case the file is opened, written and shut, with appropriate messages printed at the user's console.

* See page 19

Table 1. LISPEDIT Commands

Name	Arguments
COMBINE	f f f
DELETE	ℓ
EDIT	f
~~EHQ~~ SUPV	-
FILES	-
INPUT	-
LIST	f
OPEN	fdescr
READ	fdescr
REORDER	ℓ
RUN	f
RUNSPEAK	f
SHUT	(filename)
STRINGED	f
WRITE	fdescr ℓ
MOOZ	fdescr
SET	

← (See page 49)

~~If fdeser is the special atom DLO, it operates exactly as if it were the file descriptor list.~~

~~(DLO V7XDLO DIT WRITE DLO SYM)~~

INPUT

INPUT causes LISPED to go into the EDIT mode with an initially empty string. The only acceptable EDIT command which can reasonably be used at this point is INSERT to enter new LISP data into a system. This data will be of use only if eventually a FILE command is given to supply it with a file name.

EDIT f

This is the same as INPUT except that the EDIT is entered with the string equivalent to the file named f.

STRINGED f

This is identical to the command EDIT f.

RUNSPEAK f

The file named f, which must consist of EVALQUOTE pairs, is operated. Each successive pair of S-expressions in the file f is passed to EVALQUOTE and the results printed.

RUN f

This is the same as RUNSPEAK except that the EVALQUOTE output is not printed.

FILES

This command results in printout of a list of names of all current files.

DELETE ℓ

ℓ can be either a single library file name or a list of library file names. After checking appropriately to see that these names are all names of files in the system and that the user really wanted to delete these files, this command will cause the library file or files to be deleted. In response to question from DELETE, a user should respond with either YES or NO.

REORDER ℓ

If ℓ is a single file name, the named file is placed at the end of the list of current files. If ℓ is a list of file names, these files are removed from the current list of active files and placed in the order named at the end of the list of files.

MOOZ fdeser *

opens a disc file and reads contents. The system enters EDIT mode and the file is ready to be edited and FILEd. (The file can be put back onto disc using the ZOOM command.)

* see page 19

COMBINE f_1 f_2 f_3

The contents of f_1 and f_2 are concatenated and inserted as a new file named f_3. If a file named f_3 is already in the system, the new file named f_3 will replace it.

LIST f

The command LIST f causes the contents of file f to be printed, one-S-expression per line.

SUPV
EVQ

SUPV
The EVQ command causes LISPED to enter the EVALQUOTE mode in which successive pairs of S-expressions input from the teletype are passed to EVALQUOTE and the results printed on the teletype. The EVALQUOTE mode is similar to the normal operating mode of LISP 1.5.

If the LISPEDIT system is saved using the /SAVE function, then when the saved version is loaded the LISPED system will print out

LISPEDIT RECOVERY RECOVERY // FILES SAVED

followed by

mode MODE //

where mode is the mode in which the system was saved.

EVQ mode continues until EXIT or LISPEDIT is given as a command to EVALQUOTE.

3. EDIT

EDIT is a context editing program which is used to update an existing LISP file or to generate a new one.

When EDIT is entered from its parent program LISPEDIT, the file to be edited (a series of S-expressions, possibly empty) is converted into the equivalent token string. This string may then be examined and operated on through successive commands input on the teletype and interpretively executed by EDIT. Whenever a desired transformation of the string has been achieved, a command to file the series of S-expressions equivalent to it may be given. Finally, after all desired editing and filing to the string has been completed, EDIT may be exited and control returned to LISPEDIT.

3.1 TOKEN STRING EQUIVALENT OF AN S-EXPRESSION

The rules for converting an S-expression to its equivalent token string (from which may also be inferred an inverse transformation) are expressed in the table below:

where x is an S-expression
LP is the atom "left parenthesis"
RP is the atom "right parenthesis"
DT is the atom "dot"

S-expression x	Token string equivalent
NIL	LP RP
non-NIL atom	x
$(a_1\ a_2 \cdots a_n)$	LP $a_1\ a_2 \cdots a_n$ RP
$(a_1\ a_2 \cdots a_{n-1} \cdot a_n)$	LP $a_1\ a_2 \cdots a_{n-1}$ DT a_n RP
e.g., (A (B NIL) . C)	LP A LP B LP RP RP DT C RP

3.2 VISUALIZATION OF THE TOKEN STRING

At any moment during the editing of the token string, there is a particular portion thereof whose boundaries serve as reference points for whatever action may be called for by the next EDIT command. This portion of the string is termed the object fragment (or simply the fragment, abbreviated as FR). The object fragment may be empty, or it may include the entire string (which also could be empty).

The token string with its object fragment sub-string should be visualized as in Figure 1. The symbol "t" denotes here an arbitrary token (parenthesis, dot, or other LISP atom). Definitions for left boundary (LB), right boundary (RB), head, and tail are self-evident from the diagram. The directions backwards and forwards mean "towards the head" and "towards the tail," respectively. These six terms will be used extensively in the descriptions of EDIT commands (Section 3.5).

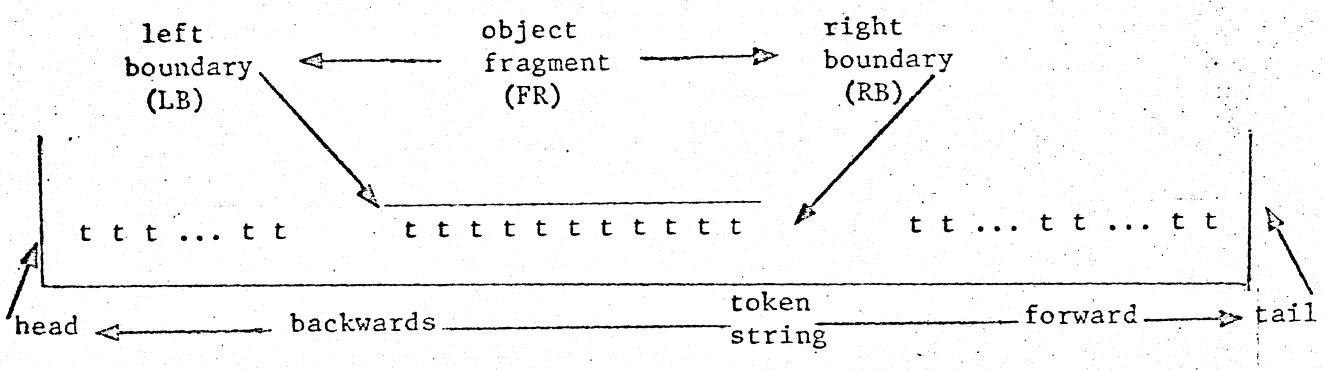

Fig. 1. Token String, Object Fragment

3.3 FORMAT OF COMMANDS

Commands to EDIT are input, one after another, on the teletype. As many commands as fit may be entered on a single line; conversely, any command which would overflow a single line may be continued onto as many additional lines as are necessary.

Each command consists of a name followed by a specific number (zero, one, or two) of arguments. When input, names and arguments must always be followed by one or more blanks, even when they occur as final items in a line. (The algorithm for making the input line-to-line transition, to be followed reflexively, is: type blank and carriage return; allow any output responses to be printed; await asterisk signal.)

The name of an EDIT command is a LISP identifier. All but three of the commands (FILE, REFILE and EXIT) are a one- (or two-) character abbreviation.

Each argument of a command is of one of four types:

- **n** integer
- **ℓ** label
- **s** fragment
- **c** commands

An integer argument is simply a LISP fixed-point number. A label argument is any LISP atom other than / or $. A fragment argument has several possible formats, which are discussed in detail in the next section. A commands argument, used only in the M(MULT) command, is any string of legal EDIT mode commands except another M command. It is bounded by the integer argument on the left and the carriage return on the right. In other words, it consists of the entire rest of the input line following the integer argument. Should a particular command be given an argument of the wrong type, an appropriate diagnostic will be printed and the remainder of the input line will be ignored.

3.4 FRAGMENT ARGUMENTS

A fragment argument of an EDIT command has four distinct formats, each of which specifies some particular fragment (sequence of tokens) as its value. These formats, paired with their corresponding values, are listed in the table below. Symbol t denotes an arbitrary token (parenthesis, dot, or other LISP atom). Symbol f denotes the name of a LISP file.

format of s	value of s
/ /	empty
/ $ f	copy of file f
/ ℓ	saved fragment under label ℓ
t t ... t t /	non empty input fragment t t ... t t

In the input of a fragment argument, two syntatic rules involving the delimiter blank must be observed.

(1) / and $ must always be separated by one or more blanks.

(2) Two adjacent tokens (t t) must be separated by one or more blanks when neither is a parenthesis.

A saved fragment (third format above) is one which, by some previous command, had been copied from the then current FR and saved for future use under a label ℓ. The saving of FR may be called for explicitly by means of the LABEL command (see its description in Section 3.5).

Should a fragment be saved under a label which is already in use, the new value will supersede (and in fact replace) the old. Thus, only the most recently input and deleted fragments are available at any given moment under a given label.

There are three particular tokens (namely /, $/, and ***) which are used as delimiters within an input fragment (fourth format above). All other tokens (parenthesis, dot, or other legitimate LISP atom) are taken literally.

(1) / is the terminal delimiter of an input fragment and must be preceded by at least one literal token.

(2) $/ is an escape delimiter. If it is not the first token in an input fragment, and is not immediately preceded by ***, the command preceding $/ will not be executed, a diagnostic will be printed and the remainder of the input line will be ignored.

(3) *** indicates that the token immediately following it is to be taken literally, even if it would normally be interpreted as a delimiter, e.g., the token / is written *** /.

3.5 EDIT COMMANDS

The commands available within STRINGED are listed in Table 2, and are described below.

The command descriptions given below are presented in a standard format. The heading displays, in order, the name of the command, an abbreviated specification of the arguments (where n, &, s, c, and a dash denote integer, label, string, commands, and no arguments, respectively) and in most cases a mnemonic for the name. The first paragraph describes the action of the command when conditions are such that it may be properly carried out. The second paragraph (if any) indicates the conditions required for execution of the command, and what will happen if they are not fulfilled. The assumption is made in the descriptions that all arguments have been correctly entered; where this is not the case a diagnostic will be printed, execution of the command will not be attempted, and the remainder of the input line will be ignored (see Sections 3.3 and 3.4).

The reader is referred to the diagram and definitions in Section 3.2, since all commands are described in terms of them.

EXIT -

EDIT is exited and control returned to the supervisory program LISPEDIT. A message confirming successful exit is printed.

If, however, this command was not immediately preceded by a successful FILE command or unsuccessful EXIT command, EDIT will not be exited, a diagnostic message will be printed, and FR will remain unchanged. This interlock feature prevents a premature, accidental exit from EDIT. To leave EDIT without having filed the token string, two EXIT commands must be issued.

FILE f

The complete token string is first converted into the particular series of S-expressions equivalent to it and then is filed under library file name f for future reference by LISPEDIT commands. If a LISP library file named f already exists, it will be replaced by the newly generated one. A message confirming successful filing is printed, and FR remains unchanged. (See Section 2.2 for descriptions of LISPEDIT commands and Section 4 for a description of LISP library files.)

If a syntax error is detected during the conversion from token string to S-expressions (i.e., if the string contains unmatched parentheses or dots out of context), no filing will occur, a diagnostic will be printed, LB will be set to the point at which the error was detected, RB will be set to the tail of the string, and the remainder of the current input line will be ignored.

ZOOM fdesc*

Opens a disc file and writes the complete token string into disc. This works even when there is a syntax error. The file can be read back in from disc by using the MOOZ command.

Table 2. EDIT Commands

NAME	MNEMONIC	ARGUMENTS
A	ADVANCE	n
B	BOUND	s
C	ECHO	-
D	DELETE	-
E	ELLIPSE	n
F	FIND	s
G	GROUP	-
H	STASH	s
HK	STASHKEEP	s
I	INSERT	s
IK	INSERTKEEP	s
L	LABEL	ℓ
M	MULT	n c
N	NEXT	-
O	ONE-EXPRESSION	-
P	PRINT	-
Q	QUANTIFY	-
R	REPLACE	s
S	SUBSTITUTE	n s
T	TOP	-
U	OUT-EXPRESSION	-
W	WHOLE	-
X	EXTEND	n
Z	POSITION	-
EXIT	-	-
FILE	-	f
REFILE	-	f
ZOOM	-	fdesc *

*See p19

REFILE f

Identical to file except that if an old copy of f exists in the library files, it will be deleted before conversion starts. This is sometimes necessary for very large files in order to regain LISP storage space. It can also shorten conversion time over using FILE.

However, if a syntax error is detected during conversion, no filing will occur, as in FILE, and library file f will be gone.

C - ECHO

The "echo switch" (initially off) is flipped from off to on, or vice versa, depending upon its current state. While the echo switch is on, a PRINT command is implied after every executed command which does not call for the printing of FR. This feature allows the editing process to be monitored step-by-step.

P - PRINT

FR is printed in full or elliptically, depending upon its length (number of tokens within it). If the length is less than 17, FR is printed in its entirety (PRINT in this case is identical to the command WHOLE). If, however, FR includes 17 or more tokens, only the first eight and last eight of these, separated by the ellipsis symbol "...", are printed. Thus, whatever the length of FR, PRINT will, in general, produce only one line to represent it. FR remains unchanged.

W - WHOLE

FR is printed in full, whatever its length may be. Should FR be empty, a blank line will appear. FR is unchanged.

Z - POSITION

A message is printed which indicates the location of LB relative to the string head, the length of FR, and the total length of the token string. The specific format of this message is:

POSITION a LENGTH b TOTAL c

where a = number of tokens between string head and LB
b = length of FR (number of tokens therein)
c = length of the token string

FR remains unchanged.

L ℓ LABEL

A copy of FR is saved under label ℓ for future reference as a saved fragment argument. If a saved fragment labeled ℓ already exists, it will be replaced by the current FR. FR remains unchanged. (See Section 3.4 for a full description of saved fragments.)

T - TOP

LB and RB are set to the string head and tail, respectively. Thus, after execution of this command, FR will include the entire token string.

N - NEXT

LB is set to the previous RB; RB is reset to the tail of the string. Thus, after execution of a NEXT command, FR becomes that part of the token string which was forward of (to the right of) the old FR. NEXT in combination with the ONE-EXPRESSION command is particularly useful for skipping over expressions.

O - ONE-EXPRESSION

RB is moved, either forward or backward, such that there will be exactly one S-expression between LB and RB; LB remains unchanged. After execution of this command, therefore, FR will contain a single S-expression.

If it is syntactically impossible to construct an S-expression beginning at LB and reading forward (i.e., if LB is at the tail of the string, if the first token forward of LB is a right parenthesis, if an out-of-context dot is encountered, etc.), a diagnostic message will be printed, LB will be set to the point at which the syntax error was detected, RN will be set to the tail of the string, and the new FR will be elliptically printed. The user is cautioned to be prepared for such a contingency.

U - OUT-EXPRESSION

OUT-EXPRESSION is identical to ONE-EXPRESSION, except for the fact that, if a legitimate S-expression is found, it will be printed in its entirety (using the LISP "PRETTYPRINT" routines).

A n ADVANCE

LB is moved forward (backward) n(-n) tokens, where n is a positive (negative) integer, except that if such movement would take LB past the tail (head) of the string, then LB is set at the tail (head). RB remains unchanged unless LB is moved past (i.e., to the right of) it, in which case it is set to the tail of the string. In short, LB is moved n tokens (n positive for forward direction) with appropriate resolutions being made in impossible situations.

X n EXTEND

RB is moved forward (backward) n (-n) tokens, where n is a positive (negative) integer, except that if such movement would carry RB past the string tail (LB), then RB is set at the tail (LB); LB remains unmoved. Thus, EXTEND does to RB what ADVANCE does for LB, except that, whereas LB can be advanced past RB, RB cannot be extended back through LB. In short, RB is moved n tokens (n positive for forward direction), but not past LB or the tail of the string.

D – DELETE

FR is deleted from the string, LB remains at the point of the deletion, and RB is reset to the same point. Thus, after execution of a DELETE command, FR is empty.

H s STASH

String argument s is inserted into the string at LB; FR is set to the inserted fragment. Thus, the new FR is a fragment input to the left of the old FR.

HK s STASHKEEP

This command is identical to STASH except that, after its execution, FR will contain the old FR as well as the newly inserted fragment.

I s INSERT

String argument s is inserted into the string at RB; FR is set to the inserted fragment. Thus, the new FR is a fragment input to the right of (forward of) the old FR. INSERT is equivalent to a NEXT followed by a STASH.

IK s INSERTKEEP

This is identical to INSERT except that, after its execution, FR will contain the old FR as well as the newly inserted fragment.

R s REPLACE

The old FR is replaced by string argument s, which becomes the new FR. Thus, the new FR is a fragment input to replace the old FR. REPLACE is equivalent to a DELETE followed by INSERT or STASH.

F s FIND

Beginning at LB, a search is made in the forward direction (to the right) until a portion of the string is encountered which is identical to the string argument s. FR is then set to that portion. Two numerical tokens are deemed identical if and only if they are equal both in value and in type (LISP predicate EQN is used here). FIND is probably the most useful of the stringed commands for purposes of editing.

If a portion of the string identical to s is not found forward of LB, a diagnostic message will be printed, and FR will ramain unchanged. The user is cautioned to be prepared for this contingency.

B s BOUND

BOUND is identical to FIND except that, where the search has been successful, instead of setting FR to merely the found fragment, FR is set to that portion of the string which begins at the original LB and extends to the right boundary of the found fragment. This may be stated even more simply--BOUND is identical to FIND except that under no circumstances is LB allowed to be moved. A particular use of BOUND in conjunction with FIND is in setting FR to a long portion of the string. FIND is first employed to locate the left boundary thereof, then a BOUND command is given to set the right boundary without having to change the left one.

S n s SUBSTITUTE

String argument S replaces FR and the n-1 string portions identical to FR, searching to the right with FIND. Algorithmically, execution of the command involves the following loop: REPLACE current FR by a copy of s, use NEXT to move past the replacement, FIND portion identical to the original FR, repeat if FIND command was successful and decremented n is non-zero, otherwise terminate with FR set to the last replacement made. SUBSTITUTE is used generally for three purposes: multiple substitution, multiple deletion and multiple replication. For multiple substitution, the first instance of the fragment to be replaced must be located (using FIND), and then a SUBSTITUTE command given with the replacement fragment and number of replacements to be made as arguments. Multiple deletion is accomplished similarly with an empty fragment being entered as the string argument of SUBSTITUTE. For multiple replication, the point at which the copies are to be placed is located, a FIND command with an empty argument is issued (FR becomes an empty fragment situated at the prior LB of FR), and finally a SUBSTITUTE command is given with n as the number of copies of the string argument to be produced (upon termination, FR will be set to the rightmost of the copies).

If argument n is not an integer greater than zero, a diagnostic message is printed and FR remains unchanged. If a search fails at any time before n reaches zero, the action of SUBSTITUTE is effectively terminated by the action of the FIND.

E n ELLIPSE

FR is first set to a single S-expression using ONE-EXPRESSION. It is then printed, using a depth-controlled elliptical printer in which n determined the depth to which parenthetical nesting will be displayed. Any sub S-expression of FR whose depth is greater than n is replaced in its entirety by the ellipsis symbol "&". This type of display of FR is sometimes easier to read than that generated by the P (PRINT) command. However the E command calls ONE-EXPRESSION, which may cause RB to move.

Q - QUANTIFY

QUANTIFY works for the most part like the W (WHOLE) command. The entire fragment FR is printed without making any adjustment for syntax. LISP PRETTYPRINT is not used. The difference between QUANTIFY and WHOLE is that QUANTIFY numbers every parenthesis encountered. The numbering starts with 1 and counts up one for each left parenthesis encountered and down one for each right parenthesis. This command can be very useful for matching left- and right-parentheses.

G - GROUP

This command is similar to Q (QUANTIFY) with two exceptions: GROUP calls ONE-EXPRESSION which may move RB, and QUANTIFY does not; also, GROUP uses the LISP PRETTYPRINT which QUANTIFY does not.

Thus G (GROUP) is to O (OUT-EXPRESSION) as Q (QUANTIFY) is to W (WHOLE).

M n c MULT

In this command, the argument c refers to the entire remainder of the string of EDIT mode commands following the argument n and before the next following carriage return. C may contain any legal EDIT mode commands except another MULT.

MULT causes the string of commands in c to be processed end to end, n times.

An error at any time causes the remainder of the MULT operation to be aborted.

4. LIBRARY FILES AND DATA STRUCTURES

The data files maintained by LISPEDIT consist of a list of S-expressions (library file) stored on the VARIABLE (GLST . 120). Each library file consists of a single S-expression whose CAR is the file name and whose CDR is a list of the contents of the file. To be compatible with RUN, RUNSPEAK and LOADEXP, the contents of a file should be a series of EVALQUOTE pairs.

The corresponding tape and disc format consists of one S-expression containing each library file. For example, suppose that there are two files LIB1 and LIB2 containing

```
CSET(AA 2) EVAL (AA)
```

and

```
DEFINE (((NILF (LAMBDA () ( ))) (ONEP (LAMBDA (J) (EQUAL J 1)))))
```

respectively. The LISP library files would appear on tape or disc as

```
(LIB1 CSET (AA 2) EVAL1 (AA))
(LIB2 DEFINE (((NILF (LAMBDA () ( ))) (ONEP (LAMBDA (J) (EQUAL J 1))))))
End-of-file
```

After the LISPED READ command is performed, the VARIABLE (GLST . 120) contains the files.

Two functions exist within LISPEDIT to provide access to the contents of files from within EVALQUOTE mode.

> TEDSEEKER (f) yields the library file &f as the entire S-expression whose CAR is name of the file, f.
>
> TEDFILER (&f) enters the library file &f into the list of files under the name f given as the CAR of &f.

A note about file-descriptor-lists (fdescr):

When a command requires a file-descriptor-list as an argument it may be a list or an atom.

If it is a list the CAR is the filename and the CDR contains other standard LISP file descriptor information, e.g.:

(NEWSTUFF V49001 N [illegible] UGLY PACKED SYM RTM6 255)

If it is an ATOM, it is the filename. If the file is already open this is a sufficient description. If the file is not open, this filename will be CONSed onto the value of a file-descriptor-variable, and this list will be used by OPEN. For input the file-descriptor-variable is (IFDL . 120) for output it is (OFDL . 120). In LISPEDIT mode these variables may be set by using the SET command, e.g.

SET IFDL (V49001 E)

or

SET OFDL (V49004 E UGLY PACKED SYM RTM6 255)

D . . .

A sample LISPEDIT conversation:

INPUTTING A FILE BY HAND.

```
/LOGIN LISP
JOB LOGGED IN . . . etc
/LISP
  Program Started
'LISP 2/24/72'
* <-some LISP's type * to signal that they're ready for input. Others ring the bell, etc.
LISPEDIT ¢
   LISPEDIT MODE
*
   INPUT ¢
*
 I   DEFINE(((FOO(LAMBDA (X Y) (PROG
 *
       (A B) A(SETQ . . . . . ))))   / P O P N O P ¢
 $ DEFINE(((FOO(LAMBDA( . . . . . ))))))
   DEFINE
   (((FOO(LAMBDA(X . . . . . ))))))
*
 N P ¢
*
 ZOOM  ABFILE1 ¢
     FILE ABFILE1 OPENED ON - - - - -
     FILE ZOOMED AS ABFILE1 //CONTINUE
*
FILE  ABFILE1  EXIT ¢
    FILED AS ABFILE1  //CONTINUE
   LISPEDIT MODE
*LIST  ABFILE1
     LISTING OF  ABFILE1
   DEFINE
          .
          .
          .
     END OF FILE ABFILE1
*
```

If it complains here that there's a syntax error, you edit it here to fix the error and then file it.

Choose this name as follows: 8 letters total, 1st 2 are your initials, last 6 may contain only letters or digits, no other characters. Be sure no one else in the class picks the same 2 initials. Make up as many names as you like.

2

Editing A File that has been prestored

Your card deck should look like:

1st card: (ABFILE1 ← use a name made up as described on page 1 of this memo.

your deck for file ← if you make a bad syntax error the file will have to be poked with another editor called OLE before you can get it into LISPEDIT.

last card: STOP)))))))) lots of extra right parens to avoid hitting the eof while reading

Typed by you

```
e    / LOGIN LISP
       job logged in etc
me   / LISP
       program started
       'LISP 2/24/72'
     *
me     LISPEDIT
         LISPEDIT MODE
     *
me     MODE ABFILE1 %
         EDIT MODE
     *
me     T F STOP / $ X 999 P
       STOP )))))))   ← The extra jungle at the end
me     D T P %
       DEFINE((( FOO(LAMBDA(.....  )))))  your file
     *
me     FILE ABFILE1 %
         FILED AS ABFILE1 //CONTINUE   ← If it complains here that there is a syntax error, you edit it here to fix the error and then file it.
me   * ENT
         LISPEDIT MODE
me   *   LIST ABFILE1
           LISTING etc
```