THE DESIGN, IMPLEMENTATION, AND USE OF LEDIT: A REAL-TIME EDITOR FOR LISP

by

JANA TAYLOR GOODMAN

B. S., Kansas State University, 1973
MS Ed., University of Southern California, 1975

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982

Approved by:



Major Professor

SPEC
COLL
LD
2608
.R4
1982
G66
c.2

A11202 303099

THE DESIGN, IMPLEMENTATION, AND USE OF LEDIT: A REAL-TIME EDITOR FOR LISP

Table of Contents

0. Preface

1. Motivation for the Design and Implementation of LEDIT...1

1.1 Disadvantages of the Current LISP System.....1

1.2 Advantages of the Real-Time LISP Editor.....3

1.3 Factors Which Influenced the Editor Design.....4

2. Overview of LEDIT.....8

3. Details of LEDIT Design.....18

4. User Manual for IISP

Appendix I. The LEDIT Program Code

Appendix II. References

ILLEGIBLE DOCUMENT

THE FOLLOWING DOCUMENT(S) IS OF POOR LEGIBILITY IN THE ORIGINAL

THIS IS THE BEST COPY AVAILABLE

PREFACE

The reader need not have expert knowledge of the LISP programming language in order to benefit from reading this document. However, it is assumed that the reader has at least a basic acquaintance with LISP structures, functions, and terminology.

It is particularly important that the reader be familiar with property lists; as LEDIT is especially designed to help the user edit these LISP structures.

An excellent reference the reader may use to learn about or review LISP is Let's Talk LISP , by Laurent Siklossky {1}. Anatomy of LISP, by John Allen {2}, is also highly recommended, especially for the more seasoned LISP programmer.

KEYWORDS: S-expression, atom, property list, editor, interactive, real-time, workspace, FEXPR, EXPR, LISP interpreter, pointer.

THE DESIGN, IMPLEMENTATION, AND USE OF LEDIT: A REAL-TIME EDITOR FOR LISP

Part 1: Motivation for the Design and Implementation of LEDIT

One of the most attractive features of the programming language LISP is that it fosters close and frequent interaction between the user and the computer. Despite LISP's extremely dynamic programming environment, editing programs or data is a considerably less dynamic activity on many LISP systems. Most often, editing function definitions or the data base requires leaving the LISP interpreter to use a conventional text editor. This is somewhat akin to using a draft horse to pull a Ferrari!

During a typical online programming session using a LISP interpreter, the user receives feedback from the machine-- often and immediately. Normal procedure is to create small modules and test them as they are created.

In contrast, sessions employing other widely-used programming languages (especially batch-oriented systems) often result in a minimum of interaction between user and machine. Typically, the programmer codes the program, and the machine compiles it. Then the program is executed, and (eventually) the user receives the output.

Unfortunately, even in many LISP systems, the user is unable to sustain this continuous feedback relationship with the interpreter when the LISP program or database needs editing. Usually the programmer must leave the interpreter and use a system text editor. Leaving LISP is one problem; using a conventional text editor for LISP code presents another problem. Text editors are designed for

the basic structure composed of characters and lines. The basic structures of LISP are S-expressions, which have no relationship to characters and lines. So editing LISP with a text editor is somewhat artificial and inconsistent.

Since the need for editing is bound to arise, this drawback demands attention. The solution presented here is to introduce a LISP editor composed of LISP functions.

Thus, the fundamental purpose of this project was to design and implement such a real-time, workspace LISP editor. The main benefit expected was improvement (in terms of convenience and consistency) in the programming environment for the user of the LISP facilities available on the Itel ASR5 here at KSU.

1.1 DISADVANTAGES OF THE CURRENT LISP SYSTEM

Without a workspace editor, the user has two choices when she wishes to make changes in the function definitions or the data base in her current workspace:

(1) The user can retype the definition or property list with the desired changes. Certainly, this is a poor solution for the user--especially when long portions of code must be reentered.

(2) If the code has been stored in a file, another possibility for the user is to leave the LISP interpreter, use a conventional text editor to make corrections to the LISP file, recall the interpreter, and reload the altered file. In addition, once back in LISP mode, it may be necessary for the user to read the stream from a number of files and repeat some initialization to return to the

state the program was in when the user exited from the interpreter.

1.2 ADVANTAGES OF THE REAL-TIME LISP EDITOR

The in_core LISP editor provides the user with several advantages:

(1) There is no need to exit from the interpreter to correct functions or property lists. The user simply puts the editor into her workspace and uses its functions to make changes in the list structures there. It is possible for the user to do all her work without ever having to leave the LISP environment. This is compatible with the highly interactive nature of LISP.

(2) When combined with a pretty printer, the LISP editor makes it much easier to modify function definitions and property lists.

A pretty printer is a formatting device which makes LISP code display more readable. (A good example is [3].) For example, the following LISP code looks rather confusing:

```
(DEFUN COUNTATOMS(L)(COND((NULL L) 0)
((ATOM L) 1)(T(PLUS(COUNTATOMS(CAR L))
(COUNTATOMS(CDR L))))))).
```

A pretty printer would make the meaning of the expression much clearer by producing the following:

```
(DEFUN COUNTATOMS (L)
       (COND ((NULL L) 0)
             ((ATOM L) 1)
             (T (PLUS (COUNTATOMS (CAR L))
                      (COUNTATOMS (CDR L)))))))).
```

The LISP editor operates on the basic structures of LISP: S-expressions--not on characters and lines as a text editor does. The user can have the editor clearly display the structure she wishes to concentrate on. Then changes which are particularly useful to the

LISP programmer can be made (e.g., group/ungroup S-expressions), as well as textual changes.

(3) Another benefit of this type of editor is particularly appealing to programmers who find the nested parentheses annoying. A very helpful side effect of using LISP functions to edit LISP structures is that the editor will not accept unbalanced parentheses!

1.3 FACTORS WHICH INFLUENCED THE EDITOR DESIGN

Several factors influenced the design of LEDIT:

(1) Since line-by-line terminals are most commonly available here, the editor is designed for that type of line-oriented device. (Examples of editors for screen-oriented display include [4] and EMACS [5].)

(2) The available functions give the programmer the power to do most of the operations available in a conventional text editor. However, the most useful are the LISP-oriented operations.

In addition, LISP editor commands which perform functions equivalent to text editor functions (e.g., QUIT,FILE,CH) have formats similar to the text editor commands. This was done because most users in the student environment will be very accustomed to using conventional text editors; the similarity should make using both types of editors less confusing.

(3) The editor presented here has a relatively uncomplicated design. Readability and ease of implementation were the primary concern; a major goal was for the code to serve as an example for anyone wishing to add a workspace editor to a LISP system. It can

be easily altered, shortened, or extended. In fact, this particular editor is based on a very simple one presented by Charniak-Riesbeck-McDermott in their book Artificial Intelligence Programming {6}. Their basic editor was used as a foundation and extensively modified and enlarged to produce LEDIT.

(4) This editor contains a facility which enables the user to define her own edit commands using combinations of existing commands. This provides an elementary example of how a user may tailor an editor to suit her needs. (EMACS {5} provides an excellent example of this feature.)

(5) It is possible to make the editor a collection of functions and global edit variables. In this type of situation, the edit functions can be called in the same way other LISP functions are invoked. Thus, edit commands may be interspersed with testing.

This is certainly in keeping with the exploratory attitude of LISP, which we are trying to encourage. However, for safety's sake, LEDIT was designed to be a single main function which contains the edit variables within its scope and calls the edit functions as it is driven by the input of edit commands. This design allows the user to abort the editing if necessary.

LEDIT also causes a conceptual shift from LISP to 'edit mode.' Editing is done while in 'edit mode,' and testing is done when the user 'returns' to LISP.

(6) Because this editor is intended primarily for student users, there is extensive error checking in the functions. The error checks include explanatory messages to the user.

(7) One source of potential problems was insoluble on this

system. Commands and arguments are put into the input buffer by the LISP READ function. The interpreter executes the functions when the user enters a carriage return.

Problems can arise if the user does not enter the proper number of arguments with a command, because the editor has no way of distinguishing between a command and an argument. It simply interprets by the order in which atoms are entered. So, there is the possibility that the editor will misinterpret all subsequent input contained in the buffer after an error has occurred.

The best solution to this problem is to have the editor abort all commands in the input buffer after an error condition is encountered. Unfortunately, this is not possible on our system, as there is no way to test for a carriage return.

The only way to check for end of line of input commands and arguments is to have the user place some special delimiter at the end of each line. It was decided that this would be too irritating for the user. In any event, it is unlikely that serious damage will be done if this problem arises; in most cases the editor will merely issue a series of error messages for each misinterpreted command. In addition, the safety valve of aborting the edit version is always available to the user in case of serious trouble. The decision was made to favor user convenience over safety in this particular case.

Conclusion

The primary reason for implementing this real-time LISP editor was to improve the LISP user's programming environment. The purely mechanical aspects of editing should be easier and less time-

consuming when the programmer can use LEDIT instead of a conventional text editor. In addition, the conceptual shift from 'LISP mode' (which requires the user to think in S-expressions) to a text editor (which requires the user to think in characters and lines) will not be necessary when LEDIT is available.

The main factor which influenced the design of this particular implementation of a real-time LISP editor was that it serve as an example for anyone wishing to implement the same type of system. The code was designed to be as self-explanatory as possible. There is no 'clever' programming; only standard LISP functions are used. Thus, the code can be easily transferred to any machine with a basic LISP interpreter.

On the existing system here at Kansas State University, LEDIT is rather expensive to run with the interpreter. A LISP compiler will make the editor operate much more efficiently.

The program is designed to allow easy extensibility. So, it is easy to include more edit functions if the implementer wishes to. The define edit command facility, in particular, is one which can certainly be extended (e.g., allow edit functions which can take arguments). Another desirable feature which may be added with relatively little trouble is the ability to put parts from one property list into another during editing. top The program is also easily modified; it will not be difficult for the implementer to tailor the editor to fit the kinds of editing most frequently needed.

Part 2. Overview of LEDIT

In general, the purpose of the edit functions is

(1) to move the edit pointer (a sort of cursor) to subexpressions of the main list being edited and

(2) to make desired changes in the subexpressions.

Essentially, the editor enables the user to tell the editor where to go and what to do when it gets there. Since the primary structures in LISP are S-expressions, the edit pointer moves from one S-expression to another, and edit commands work on the S-expressions pointed to by the edit pointer.

The user calls the editor by simply calling the function LEDIT (see Figure 2.1) with two arguments:

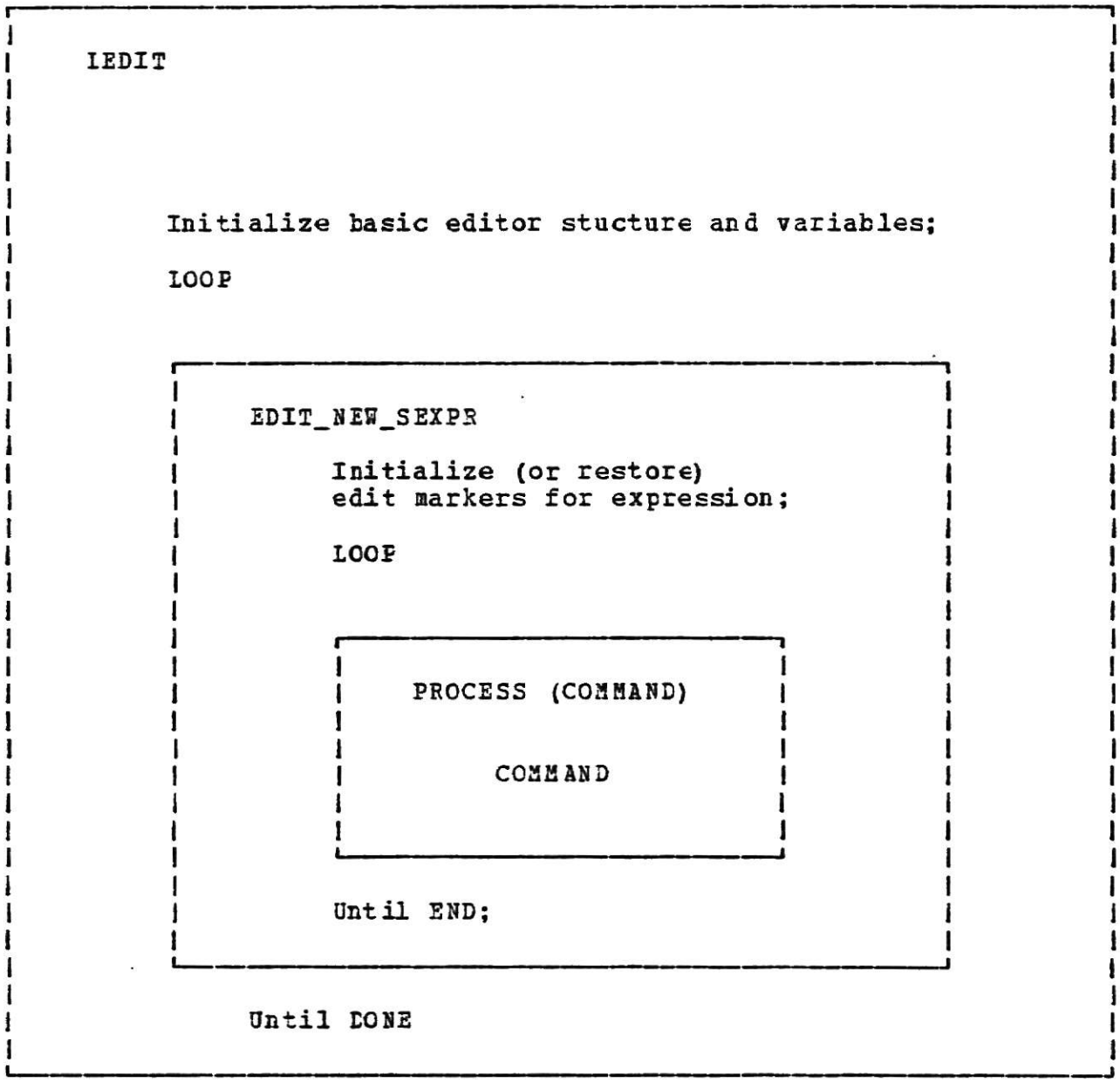
(1) the name of the atom whose property list is to be edited and

(2) the name of the property whose value is to be edited (or nothing if the user wishes to edit the whole property list of the atom). From this point until the user terminates the editing, the user is in edit mode.

The edit function can be called recursively (See Figure 2.1); i.e., while in edit mode, the user may edit another property list and then return to edit the previous list.

Command processing is a continuous loop of <read command--execute command>. The user may enter as many commands as she wishes on a line; the editor will process each one in the order in which they are placed into the input buffer.

Because edit lists may be quite long, nothing except error messages is displayed by the editor-- unless the user requests a printout of the portion she wishes to concentrate on.



If COMMAND = EDIT,

PROCESS calls EDIT_NEW_SEXPR.

Thus S-expressions may be edited recursively.

Figure 2.1

There is only one edit variable which is global. This is the variable which saves names of edit commands defined by the user. This variable is global so that user-defined edit commands may be used throughout the LISP session--not just during execution of the edit function in which the command was defined. All others are local to the edit function and global to the functions it calls. Of course, saved altered property lists are available to the entire workspace.

- LEDIT -

LEDIT is a FEXPR which allows the user to invoke the editor by typing

(LEDIT <atom-name> opt:<property-name>).

{OPT: here denotes an argument which may be omitted.}

The reason LEDIT is a FEXPR is to remove the necessity for the user to type quotes before the parameters. The work of setting editor variables and calling the command processor is done by LEDIT. The major editor variables are all local to LEDIT and global to the editing functions it calls. The body of the LEDIT function is a loop which edits property lists until the *EXPRESSIONS* stack is empty.

- EDIT_NEW_SEXPR -

Initially, this function places the *EXPRESSION* currently being edited (if there is one) onto the *STORED_EXPRESSIONS* list. Then, if PROPERTY_NAME is not nil, EDIT_NEW_SEXPR calls

PROPERTY_LIST_TO_EDIT to locate the given property on the given atom's property-list. If PROPERTY_NAME is not nil, the atom's whole property-list will be the expression to be edited. The rest of the function is a loop which reads a command, processes the command, and stops when the command END is received.

```
--------------------------
- INITIALIZE_EDIT_MARKERS -
--------------------------
```

This function uses FIND_PROPERTY to retrieve a list which contains the marker to the actual property_list being edited. For example:

Atom A has the following property-list:

```
( A     (P1 (M N) P2 (Q R) P3 (X Y)))
  ^atom          ^property-list
```

The command (LEDIT A P2) will store

```
((M N) P2 (Q R))  on the list *ORIGINAL_EXPRESSIONS*.
 ↑ marker to P2's
   place on plist
```

The ccmmand (LEDIT A) will store

```
( A     (P1 (M N) P2 (Q R) P3 (X Y)))
  ^marker
```

This is done so that the first element on the edit list can be modified. The actual value of *EXPRESSION* will be exactly the property list or part of the property list which the user specifies. In the example above, *EXPRESSION* will have the values

(P2 (Q R)) and (P1 (M N) P2 (Q R) P3 (X Y)), respectively.

The edit markers *EDIT|* and *LEVEL* are both now initialized to *EXPRESSION*. A top marker is placed on the *EDIT_POSITIONS* stack. The specified atom and prop-names are stored on the stacks *ATOM_NAMES* and *PROP_NAMES*, so that they can be restored in case of recursive editing.

- PROCESS -

This function recognizes the edit command which is in the input buffer and calls the appropriate edit function to execute the command. The structure of PROCESS is that of a basic CASE statement. Thus, in most cases, adding a command to the editor will involve no more than adding an S-expression of the following form to the branches of the COND statement:

((= '<command> COMMAND) (<command> <possible argument list>)).

2.1 Explanation of How Process Executes Edit Commands

(It will be helpful for the reader to refer to the user manual explanation of edit commands {Part 4} as well as the code and documentation for the edit functions used.)

(1) USER DEFINED EDIT COMMANDS

The function is simply executed. Note that user_defined commands may not have arguments.

(2) HELP

The HELP function requires a single atomic argument which is either the atom 'ALL or the name of an edit command. The appropriate explanatory message is then displayed on the screen.

(3) T(ype)

The Pretty-Printer function SP is called to output the value of *EDIT|* in a meaningful format. THe current edit pointer will be the CAR of this list. Note that the editor does not display to the screen unless there is a user request or an error.

(4) EDIT

This command causes EDIT_NEW_SEXPR to be called to initiate editing of another list; the atom and property are specified by the user. The old edit list and edit markers will be saved. In this way, the user may edit as many lists as she wishes while in edit mode.

(5) DEFCMD (Define Command)

The user may define her own commands using combinations of existing edit commands. To do so, she simply enters the name of the command and a sequence of commands. The user_defined edit function will then be available during any edit session while the interpreter is still executing. Note that these commands may not take arguments.

(6) RPT (Repeat)

RPT allows the user to repeat a sequence of edit commands a given number of times. PROCESS actually generates a new function name, calls DEFCMD to define a function consisting of the sequence of edit commands to be repeated, and uses RPT_USER to execute the sequence the given number of times.

(7) CH_CMD_DELIM and CH_LIST_DELIM

These two commands allow the user to change the delimiters for argument lists. Initially, the character '/' is the *LIST_DELIMITER*, and the atom ENDCMD is the *CMD_DELIMITER*. The character '/' is used to separate argument lists for CH, CH|, CHATOM, CHATOM|, CHSTR, and F(ind). The atom ENDCMD is used with DEFCMD and RPT.

(8) REM

This command allows the user to remove a given number of S-expressions from the level of her current positicn to a buffer which she names. Subsequent REM commands to the same buffer will add those S-expressions onto the end of the buffer. The buffer is really just a list of S-expressions.

(9) REPL

This command allows the user to take a given number of S-expressions from the buffer she specifies beginning at the position in the specified buffer. This list of S-expressions is then inserted after the current edit position. The buffer may be reused.

(10) DISPLAY

The function DISPLAY pretty prints the names of buffers which the user has created.

(11) CONTENTS

This command causes the display the contents of the buffer which the user specifies.

(12) UN and UN<n> and UN*

These commands ungroup the current edit pointer as many times as specified or until the current edit pointer is an atom. UN uses the simple edit function UNGROUP to remove one level of nesting. UN* uses REPEAT and the auxiliary function UN* to ungroup until the current edit pointer is an atom. (The auxiliary function is used so that no error message is returned if the editor tries to ungroup an atom.) If a number of UNGROUPS are requested PROCESS uses REPEAT and the UNGROUP function to do the specified number of ungroups. If the current edit pointer becomes an atom before the given number of ungroups is done, an error message will be printed.

(13) DEL and DEL<n> and DEL*

These functions delete the current edit pointer as many times as specified or until at the end of the current edit level. DEL simply executes the edit function DELETE↑ to delete the current edit pointer. DEL* and DEL<n> use the auxiliary function DELETE_N to do the given number of deletions.

(14) IA and IB

These commands are simply executed with the insertion as the argument list read by the function ARG_LIST.

(15) CH and CH↑

These commands both cause process to call CHANGE_N to make a given number of changes to the edit list. Three arguments are needed: (1) the old part which is to be changed, (2) what it is to be changed to (or CAR *EDIT↑* if command is CH↑), and (3) how many changes are to be made. PROCESS uses the function ARG_LIST to get the first two arguments and the LISP function READ to get the number of changes.

(16) CHATOM and CHATOM↑

These commands cause PROCESS to use CHANGE_ATOM to edit the characters in an atom. The first three arguments required are all atoms. It is possible to get these arguments with simple use of LISP's READ function; however, the function ARG_LIST is used for consistency with CH and CH↑ which require the *LIST_DELIMITER*. ARG_LIST should return a list of the single atom argument. Then the CAR of that list is sent as the parameter to CHANGE_ATOM. READ then reads the final argument which is the number of changes to be made.

(17) CHSTR

PROCESS uses CHANGE_STRING to change a string of characters within atcms to another given string of characters. Again, the CAR of the argument list is passed; and a call to READ reads the number of changes to be made.

(18) F(ind)

The command F cause PROCESS to use FIND to locate the list returned by the function ARG_LIST.

(19) S and TOP

PROCESS calls the corresponding functions which perform these commands: BACK_TO_START and TOP, respectively.

Part 3: Details of LEDIT Design

There is only one global variable in the LEDIT program. This variable is *USER_COMMANDS*, which stores the names of user-defined edit commands. *USER_COMMANDS* is global so that the user will have access to user-defined commands during the entire session with the interpreter, not just during one session in the editor.

The following part of this report is a detailed description of each major function in the LEDIT program. It will also be helpful to refer to the LISP code of LEDIT (APPENDIX I). When a semantic category is absent from the description of a particular function, then the obvious inference is intended; for example, if PARAMETERS is omitted, then the function has no parameters.

LEDIT

Local variables:

EXPRESSION	(1)
EDIT↑	(2)
LEVEL	(3)
EDIT_POSITIONS	(4)
ORIGINAL_EXPRESSIONS	(5)
STORED_EXPRESSIONS	(6)
ATOM_NAMES	(7)
PROP_NAMES	(8)
RPT_CMD	(9)
BUFFERS	(10)
LIST_DELIMITER	(11)
CMD_DELIMITER	(11)

(See section 3.1 for further explanation of these variables.)

Parameters: ATOM_NAME
PROPERTY_NAME

Functions called: EDIT_NEW_SEXPR
ERROR_MESSAGE

Description

```
If no ATOM_NAME then
  ERROR_MESSAGE 'Too few arguments
  RETURN to LISP Mode

Initialize Edit Structures

Loop

  Call EDIT_NEW_SEXPR

Until no edit expr on stack    ; initial
                               ; condition
                               ; restored
```

3.1 Major Variables Used by LEDIT

(1) *EXPRESSION*

This variable is the entire expression being edited; it has the value of either the whole property list of the given atom or the property name and property value of the given atom and property. The value of *EXPRESSION* is initialized by the edit command.

(2) *EDIT↑*

This is the edit pointer; its value is an S-expression contained in *EXPRESSION*. For example, suppose *EXPRESSION* is (A B C) and the user wishes to fix B. She can conceptually move the edit pointer to that place in the list (A B C). The actual value of *EDIT↑* will be the S-expression (B C). So, in this paper 'edit pointer' and 'current edit' position actually refer to the CAR of the variable *EDIT↑*.

(3) *LEVEL*

This variable represents the level which contains the edit pointer. For example,

```
*EXPRESSION*   (A (B C D) E)
*EDIT↑*          (C D)
*LEVEL*        (B C D)
```

Thus, it is used to save the positions to the left of the edit pointer.

(4) *EDIT_POSITIONS*

This variable represents a stack which is used to save edit levels and pointer positions above the current level when the editor is moving down in the expression. When the editor moves down a level, the current value of *EDIT↑* and *LEVEL* are pushed onto the stack of *EDIT_POSITIONS*. When the editor moves up, it resets those variables to the values which are on the front of *EDIT_POSITIONS* (i.e., the CAR and CADR of *EDIT_POSITIONS*). Then those values are popped off the stack; i.e., *EDIT_POSITIONS* is set to the CDDR of *EDIT_POSITIONS*.

(5) *ORIGINAL_EXPRESSIONS*

The editor works only on a copy of the property list to be edited. *ORIGINAL_EXPRESSIONS* saves the original property list. This will be changed to the edited version if the user specifies the FILE option when she leaves the editor. Otherwise, the edited version will be discarded; and the original property list will be unchanged.

(6) *STORED_EXPRESSIONS*

This variable is used to save information about the expression being previously edited when the EDIT command is called recursively inside the editor. The value is saved on the *STORED_EXPRESSIONS* stack and popped as the user finishes editing the expression.

(7) *ATOM_NAMES*

This stack contains the atom names whose property lists were being previously edited when recursive edit calls were made. The atom name for the list being currently edited will be on top of the stack.

(8) *PROP_NAMES*

This stack contains the property names whose values were being previously edited when recursive edit calls were made. The property name for the list being currently edited will be on top of the stack. Names are popped after end of editing.

(9) *RPT_CMD*

This variable is used to save the internally generated name of a command which is composed of the list of edit commands entered by the user when the RPT ccmmand is used.

(10) *BUFFERS*

This variable is a list of buffers named by the user and used to move blocks of S-expressions. (See REMove,REPlace commands.)

(11) *LIST_DELIMITER*, *CMD_DELIMITER*

These variables are delimiters for lists of arguments and commands. (See CH, CH↑, CHATOM, CHATOM↑, CHSTR, F, DEFCMD, RPT commands.)

Originally, they are set to '/' and 'ENDCMD', respectively. They may be changed by the user.

EDIT_NEW_SEXPR

Calling functions: LEDIT

Local variables:

COMMAND: edit command entered by user

Parameters: ATOM_NAME
PROPERTY_NAME

Functions called: PROPERTY_LIST_TO_EDIT
ERROR_MESSAGE
INITIALIZE_EDIT_MARKERS
END
RESTORE_EDIT_MARKERS
PROCESS

Description

```
Store edit expr (or 'Done)

Get new p-list to edit

If p-list NOT FOUND then
  Call ERROR_MESSAGE
  Restore edit expr
  Pop stored_expr list
else initialize edit expr

Loop

  Read edit command
  If command = 'END then
    Discard or save edit expr
    Restore previous edit expr
    RETURN

  Call PROCESS (COMMAND)

End loop
```

PROPERTY_LIST_TO_EDIT

Calling Functions: EDIT_NEW_SEXPR

Parameters: ATOM_NAME
PROPERTY_NAME

Functions called: FIND_PROPERTY

Description

```
If null prop_name then
   (atom_name  entire p-list)
else call FIND_PROPERTY
```

FIND_PROPERTY

Calling Functions: PROPERTY_LIST_TO_EDIT

Parameters: PROPERTY_LIST
PROPERTY

Functions called: FIND_PROPERTY !!!RECURSIVE!!!

Description
If at end of p-list then nil else if prop-name = second elt of p-list then p-list else move over two elts on p-list Call FIND_PROPERTY (new p-list, prop-name)

This function returns the part of the property list along with a leading marker to the place on the property list. The marker is the element to the left of the property to be edited; or it is the atom itself if the entire list is to be edited (i.e, PROPERTY is nil).

FOR EXAMPLE: The property list of ATOM1 is

(P1 (A B) P P2 (C D) P3 (E F))

If PROPERTY Y = P2 then

((A B) P2 (C D) P3 (E F))

will be returned.

If PROPERTY = NIL then

(A (P1 (A B) P2 (C D) P3 (E F)))

will be returned.

INITIALIZE_EDIT_MARKERS

Calling functions: EDIT_NEW_SEXPR

Functions called: COPY (edit expression)

Description
Store original expression If property-name = NIL then   edit expr <-- COPY (CDR atom-name) else edit expr <-- COPY (prop-name prop-value) Push 'TOP marker onto *EDIT_POSITIONS* *LEVEL*,*EDIT#* <-- edit expr Push atom-name onto *ATOM_NAMES* stack Push prop_name onto *PROP_NAMES* stack

COPY

Calling functions: INITIALIZE_EDIT_MARKERS
ARGUMENTS
REMOVE
CHANGE_N

Parameters: L(IST)

Functions called: COPY !!!RECURSIVE!!!

Description

```
If list is an atom (including nil) then
  RETURN List
else (COPY (CAR list)) (COPY (CDR list))
```

END

Calling functions: EDIT_NEW_SEXPR

Local variables:

END_COMMAND: user inputs dispostion of edit expr

Functions called: QUIT
FILE

Description

```
Loop

  Prompt 'QUIT OR FILE?
  Read End Command

Until legal end command read

Call QUIT or FILE
```

QUIT

Calling functions: END

Description
Output 'Aborted No change is made to original expression

FILE

Calling functions: END

Functions called: DELETE_PROPERTY

Description

If a single proerty was edited then
  Call DELETE_PROPERTY to remove the
  prop-name and value from the p-list
  Put the edited version of the prop-name
  and the value on the p-list
else put the edited version of the
  p-list in place of the old one
Prompt 'DO YOU WISH TO PUT INTC A FILE?
  'TYPE YES OR NO
If yes then
  Prompt 'ENTER THE FILENAME:
  Put the new p-list into the given LISP file

DELETE_PROPERTY

Calling functions: FILE

Parameters: ATOM
PROPERTY

Description
Replace CDR of top of *ORIGINAL_EXPRESSIONS* (the original edit expression) with CDDDR of the original edit expression

The effect is to attach the marker on the p-list with the rest of the list past the property and its value.

FOR EXAMPLE:

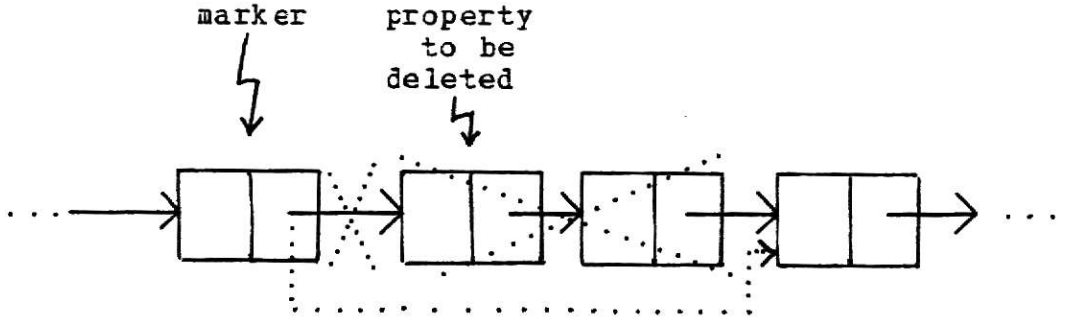
If original expression was

((VAL2 A) PROP3 (VAL3 B) PROP4 (VAL4 C))

DELETE_PROPERTY would produce

((VAL2 A) PROP4 (VAL4 C))

DELETE_PROPERTY uses the function RPLACD which does 'list surgery' on the list, i.e., it actually changes the links on the property list structure (as below).



RESTORE_EDIT_MARKERS

Calling functions: EDIT_NEW_SEXPR

Functions called: RESTORE_*EDIT_POSITIONS*

Description

Pop stack of original expressions
Pop stack of atom-names
Pop stack of prop-names
Call RESTORE_*EDIT_POSITIONS*
Restore previous edit expr
Pop stored expressions
If previous edit-position 'TOP then
  Level and edit↑ <-- edit expr
else
  edit↑ <-- top of edit-position stack
  level <-- next on edit-position stack

RESTORE_*EDIT_POSITIONS*

Calling functions: RESTORE_EDIT_MARKERS

Functions called: RESTORE_*EDIT_POSITIONS*
!!!RECURSIVE!!!

Description

```
If back at top-marker for edit-positions then
  Pop top-marker
else Pop level and edit↑ off edit-positions
     Call RESTORE_*EDIT_POSITIONS*
```

PROCESS

Calling functions: EDIT_NEW_SEXPR

Parameters: COMMAND

Functions called:
HELP
SP
EDIT_NEW_SEXPR !!!RECURSIVE!!!
DEFINE_USER_COMMAND
RPT_USER
REMOVE
REPLACE
DISPLAY
CONTENTS
UNGROUP
UN*
FIRST_N
MAKE_NUMBER
GROUP
NUMBER_OF_REPETITIONS
DELETE↑
DELETE_N
INSERT_AFTER↑
INSERT_BEFORE↑
REPEAT
RIGHT
LEFT
UP
DOWN
CHANGE_N
CHANGE_ATOM
CHANGE_STRING
FIND
BACK_TO_START
TOP
ARG_LIST
ERROR_MESSAGE

Description

(See alsc 'explanation of how process executes commands'--Sectien 2.1)

Case

User-defined command	: execute command
Command is HELP	: Read which command Call HELP to output explanation
Ccmmand is T(ype)	: Pretty print current edit↑
Command is EDIT	: Read atom-name and prop-name Call EDIT_NEW_SEXPR (atom-name prop-name)
Command is DEFCMD	: Read user-command-name Call DEFINE_COMMAND (user-command-name)
Command is RPT	: Generate new atom Call DEFINE_COMMAND (new atom) Read number of repeats Call RPT_USER (new atom n)
CH_CMD_DELIM is command	: Read new command delimiter Echo print *CMD_DELIMITER* <-- new one
CH_LIST_DELIM is command	: Read new list delimiter Echo print *LIST_DELIMITER* <-- new one
Command is REMOVE	: Read buffer-name Read number of expr to remove Call REMOVE (buffer n)
Command is REPLACE	: Read buffer-name Read number of start expr Read number of expr to replace Call REPLACE (buffer s n)
Command is DISPLAY	: Call DISPLAY_BUFFER_NAMES

```
Command is CONTENTS   : Read buffer-name
                        Call DISPLAY_BUFFER_CONTENTS

Command is UN         : Call UNGROUP

Command is UN*        : Call REPEAT (UN* *--global)

First 2 elts of       : Call MAKE_NUMBER to
explode command           convert rest of
are U N                   explode command to
                          number
                      Call REPEAT (UNGROUP n)

G is first elt        : Call NUMBER_OF_REPITITIONS
of explode command        to get arg from
                          explode command
                      Call GROUP (n)

Command is DELETE     : Call DELETE↑

Command is DEL*       : Call DELETE_N (*--globally)

First 3 elts of       : Call MAKE_NUMBER to
explode command           get arg from
are D E L                 explode command
                      Call DELETE_N (n)

Command is IA         : Call INSERT_AFTER↑

Command is IB         : Call INSERT_BEFORE↑

Command is            : Call NUMBER_OF_REPETITIONS
R v L v U v D             to get arg from
                          explode command
                      Call REPEAT
                        (R v L v U v D
                         n)

Command is CH         : Call ARG_LIST for old
                        Call ARG_LIST for change
                        Read number of changes
                        Call CHANGE_N (old change n)

Command is CH↑        : Call ARG_LIST for change
                        Read number of changes
                        Call CHANGE_N
                              (car edit↑ change n)

Command is CHATOM     : Call ARG_LIST for atom
                        Call ARG_LIST for old
                        Call ARG_LIST for change
                        Read number of changes
                        Call CHANGE_ATOM
                               (atom old change n)
```

```
Command is CHATOM↑   : Call ARG_LIST for old
                       Call ARG_LIST for change
                       Read number of changes
                       Call CHANGE_ATOM
                              ((car edit↑) old
                               change n)

Command is CHSTR     : Call ARG_LIST for old
                       Call ARG_LIST for change
                       Read number of changes
                       Call CHANGE_STRING
                              (old change n)

Command is F         : Call ARG_LIST for key
                       Call FIND (key)

Command is S         : Call BACK_TO_START

Command is TOP       : Call TOP

Otherwise            : ERROR_MESSAGE
                       'ILLEGAL COMMAND
```

ARG_LIST

Calling functions: PROCESS

Local variables: I

Parameters: ARG

Description

```
L <-- nil

Loop

  If ARG = *LIST_DELIMITER* then
    If (null I) then RETURN (nil)
    else RETURN L
  else L <-- I app (ARG)
       Read ARG
     Go Loop
```

It is possible tc have null argument for CHATOM and CHSTR. Since PROCESS calls these functions with CAR (ARG_LIST), it is necessary for ARG_LIST to always return a list (otherwise PROCESS will try to take the CAR of an atom). In this case (NIL) is returned. Ctherwise the list of arguments entered by the user is returned.

REPEAT

Calling functions: PROCESS
CHANGE_N

Local variables: COUNT
DO_COMMAND

Parameters: COMMAND
N

Functions called: LEGAL_NUMBER_ARG
ERROR_MESSAGE

Description

```
If n not legal argument or =0 then
  RETURN nil

Count <-- 1

Do first command on list

If command not completed then
  ERROR_MESSAGE 'O COMPLETED
  RETURN

Loop

  If (*--global and command
      not completed) then
    RETURN
  Increment Count
  Do next command on list

Go Loop
```

NUMBER_OF_REPETITIONS

Calling functions: PROCESS

Functions called: MAKE_NUMBER

Description
Calls MAKE_NUMBER to make integer of digit characters in command

MKNUM

Calling functions: NUMBER_OF_REPETITIONS
EXPLODE1

Parameters: N

Description
Makes number atom from character digit

MKDIG

Calling functions: EXPLODEDIG

Parameters: N

Description
Makes char atom of number

MAKE_NUMBER

Calling functions: PROCESS
NUMBER_OF_REPETITIONS

Parameters: STR

Local variables:

COUNT: loop counter
SUM: number to be returned

Functions called: MKNUM

Description

```
If string of digits is null then
  RETURN 1 (default value)

If first elt of string not a digit then
  RETURN first elt

Count <-- string length - 1

Call MKNUM (first digit)
Sum <-- first digit * count ** 10

Loop

  Pop string of digits
  If first elt of string not a digit then
    RETURN first elt
  Decrement count
  Sum <-- Sum + (MKNUM (first digit)) * count ** 10

Until Count = 0

RETURN Sum
```

DEFINE_COMMAND

Calling functions: PROCESS

Local variables:

DEFINITION: user definition
CMD: edit command
DESCRIPTION: list of edit commands

Parameters: NAME of user defined edit function

Functions called: ARGUMENTS
MOVE_OVER

Description

```
Put name on list of user commands
Call USER_DEF for description
DEFINITION <-- (DEFUN name () )
Loop
  Case
    First elt of          CMD <--
    description           (first elt
    is IA IB CH CH↑       arglists)
    CHATOM CHATOM↑
    CHSTR F
    (i.e.,edit function
    needing arg list)

    First elt of              : ERROR_MESSAGE
    description                  first elt
    is HELP                     'NOT ALLOWED
    EDIT DEFCMD                   RETURN
    RPT
DEFINITION <-- DEFINITION app (CMD)
(MOVE_OVER)
Until null DESCRIPTION
```

USER_DEF

Calling functions: DEFINE_COMMAND

Local Variables: ARG
LIS

Description

```
Read argument

List <-- (ARG)

Loop

  Read arg

  If arg is *CMD_DELIMITER* then
    RETURN Lis

  Lis <-- (Lis  Arg)

Go Loop
```

ARGUMENTS

Calling functions: DEFINE_COMMAND

Functions called: GET_ARGS

Description
This function puts the list of a command its arguments from the DEFINE_COMMAND description and puts it in the form (COPY <args>) so that it can be put into the definition of the user_command.

GET_ARGS

Calling functions: ARGUMENTS

Local variables: ARG: ARGUMENT FROM DESCRIPTION
LIS: (ARG1 ... ARGN)

Description

```
Pop command off description
Arg <-- first elt of description
Pop description
List <-- (Arg 1)
Loop
  Arg <-- first elt of description
  If Arg = *LIST_DELIMITER* then
    RETURN List
  Pop description
  List <-- (Arg1 ... Argi Arg)
Go loop
```

RPT_USER

Calling functions: PROCESS

Local variables: COUNT
DO_COMMAND

Parameters: COMMAND N

Description

```
If n = * then
  ERROR_MESSAGE '* not allowed
  RETURN
else if n not number then
  ERROR_MESSAGE 'Illegal arg
  RETURN
else if n = 0 then
  ERROR_MESSAGE '0 Completed
  RETURN
Count <-- 1
Do command
If command not completed (DO_COMMAND = nil)
then
  ERROR_MESSAGE '0 Completed
  RETURN
Loop
  If n = * and command not completed then
    RETURN
  else if n = count then RETURN
       else if command not completed then
              ERROR_MESSAGE (count -1)
                            'completed
              RETURN
  Increment count
  Do Command
Go loop
```

(Additional note on next page)

RPT_CMD is the function used for the edit command RPT. Global performance is not allowed because of the danger of infinite loops (e.g.,RPT G UN ENDCMD *). RPT stops if a repetition is not completed or if the specified number of repetitions is done.

REMOVE

Calling functions: PROCESS

Parameters:
BUFFER_NAME: buffer to place S-expressions in
N: number of S-expressions

Functions called: GET_SEXPRS_FOR_BUFFER

Description

```
If n not legal number then
  RETURN nil
else if n = * (global) then
  n <-- number of S-expr in edit↑
else if n = 0 then
  ERROR_MESSAGE 0 'Illegal Arg
  RETURN
else if new buffer put name on buffer list
Call GET_SEXPRS_FOR BUFFER
Add s-exprs to end of buffer
```

GET_SEXPRS_FOR_BUFFER

Calling functions: REMOVE

Local variables: COUNT: counter for loop
L: list of S-expressions

Parameters: N: number of S-expressions

Functions called: ERROR_MESSAGE

Description

```
Count <-- 1

L <-- (First elt of edit↑ list)

Loop

  Increment ccunt

  If there is no S-expr to delete and
     n < count then
    ERROR_MESSAGE (count -1) 's-exprs removed
    RETURN

  L <-- (L first elt of edit↑ list)

Go loop
```

REPLACE

Calling functions: PROCESS

Parameters: BUFFER_NAME: where to get s-exprs from
N: number of S-expressions
START: number of s-expr to begin with

Local variables: BUFFER: store contents of buffer

Functions called: MARK_START
INSERT_AFTER↑
COPY
ERROR_MESSAGE

Description

Store buffer contents in buffer

```
If empty buffer then
  ERROR_MESSAGE 'Empty Buffer
  RETURN
else if n not legal number then RETURN nil
else if START not legal or <= 0 then
  ERROR_MESSAGE START 'illegal arg
  RETURN
else if N = *(global) then
  n <-- length of buffer - START + 1
else if n = 0 then
  ERROR_MESSAGE
  RETURN 0 'Illegal arg
else if START > length buffer then
  ERROR_MESSAGE START 'Beyond end of buffer
  RETURN

Call INSERT_AFTER↑ to put COPY of
  FIRST_N n (MARK_START START buffer)
  after the first elt on edit↑ list
```

MARK_START

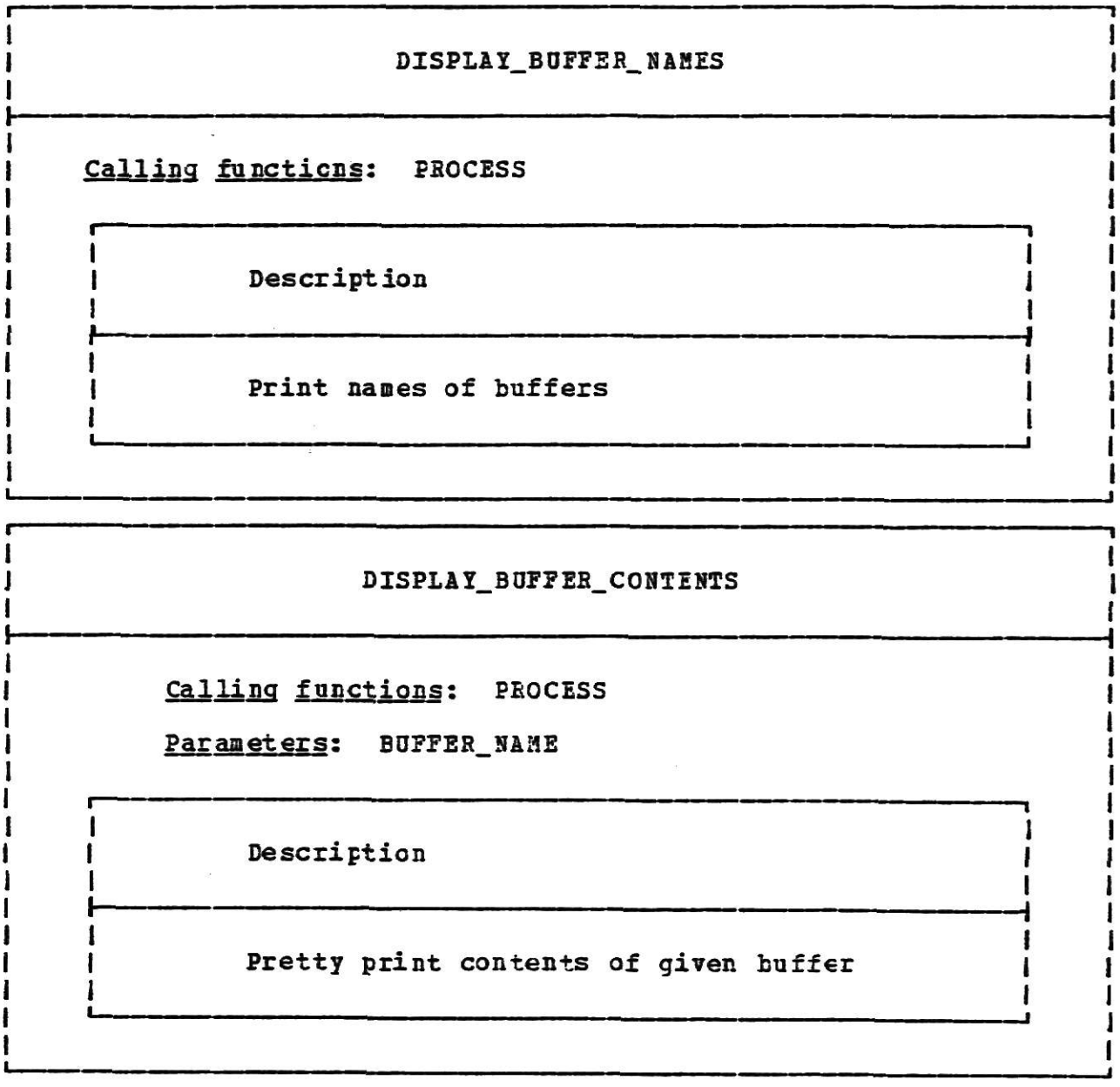
Calling functions: REPLACE

Parameters: N: number of place to mark start
L: list to mark start on

Functions called: MARK_START !!!RECURSIVE!!!

Description

```
If n = 1 then return l
else if at end of l then return l
else  Call MARK_START (n - 1) (cdr l)
```


DISPLAY_BUFFER_NAMES
Calling functions: PROCESS
Description
Print names of buffers
DISPLAY_BUFFER_CONTENTS
Calling functions: PROCESS
Parameters: BUFFER_NAME
Description
Pretty print contents of given buffer

DELETE_N

Calling functions: PROCESS

Parameters: N: number of S-exprs to delete

Local variables: COUNT: counter for loop

Functions called: LEGAL_NUMBER_ARGS
ERROR_MESSAGE
DELETE↑

Delete_n

```
If count not legal arg then RETURN nil
else if n = *(global) then n <-- length edit| list
else if n = 0 then
  ERROR_MESSAGE 0 'zero arg
  RETURN

count <-- 1

Loop

  If n < count then RETURN
  else if there is no s-expr to delete then
    ERROR_MESSAGE count 'Deletions

  Increment count

Go Loop
```

DELETE↑

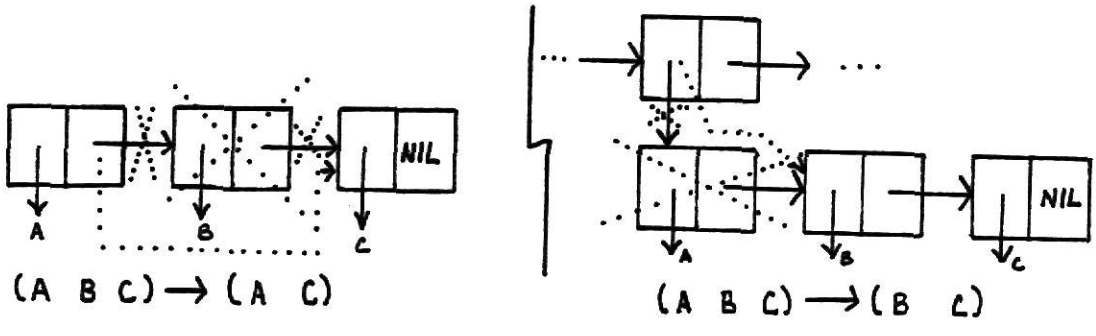
Calling functions: PROCESS
DELETE_N
CHANGE_N

Functions called: R L U D
AT_END_OF_EDIT_LIST
AT_BOTTOM_LEVEL
ERROR_MESSAGE

Description

```
If it is possible to go left (L) then
  go left
  connect current edit↑ to seccnd elt past edit↑
  If at_end_of_edit_level then
    ERROR_MESSAGE 'End of level
  else move pointer to right
else if it is possible to go up (U) then
  go up
  connect current edit↑ to CDAR of edit↑
  if at_bottom_level then
    ERROR_MESSAGE 'end of level
  else move pointer down to next level
otherwise 'at start of edit s-expr'
  edit↑ <-- pop edit↑
  expr <-- edit↑
  level <-- edit↑
```

DELETE↑ does list surgery as shown below:



GROUP

Calling functions: PROCESS

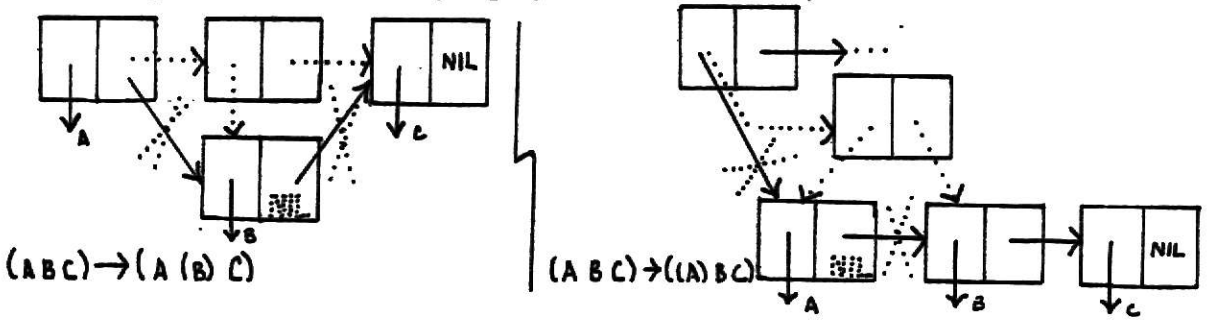
Parameters: N: number of s-exprs to group

Functions called: LEGAL_NUMBER_ARG
AT_BEGINNING
(= EDIT↑ S-EXPR)
ALL_BUT_FIRST

Description

```
If n not legal then nil
else if n = *(global) then
       n <-- length edit↑
       If at_beginning then
         ERROR_MESSAGE 'FIRST S-EXPR OF LIST
       else if n = 0 then
         ERROR_MESSAGE 'zero arg
       else if length edit↑ < n then
         ERROR_MESSAGE 'n > length of edit↑
       otherwise
         edit↑ <-- (first_n of edit↑ list)
```

Group does list surgery (as shown below)



LEGAL_NUMBER_ARG

Calling functions: REMOVE
REPLACE
REMOVE
DELETE_N
GROUP
CHANGE_N

Parameters: N: number to be checked

Functions called: ERROR_MESSAGE

Description
If n is number and n < 0 or n = * then ERROR_MESSAGE 'illegal arg

UNGROUP

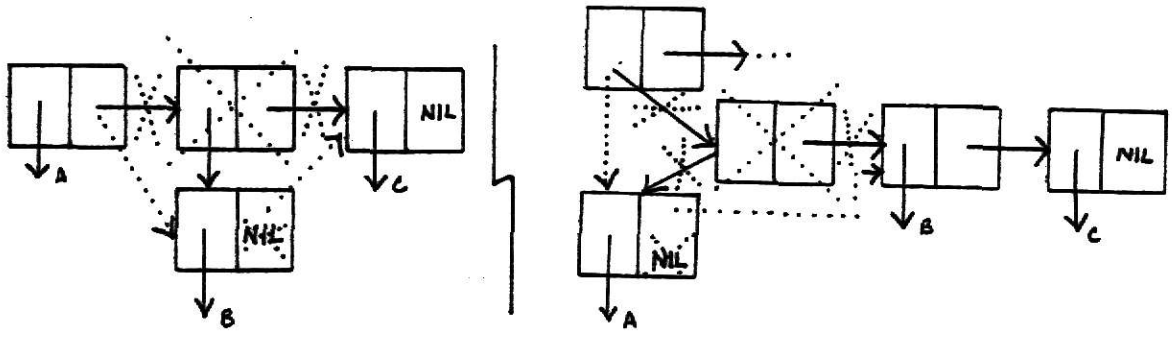
Calling functions: PROCESS

Functions called: ERROR_MESSAGE
AT_BEGINNING

Description

```
If current edit↑ is atom then
  ERROR_MESSAGE 'cannot ungroup atom
else if at_beginning then
  Call CHANGE↑ with current edit↑
  current edit↑ <-- CAR of current edit↑
otherwise
  Nconc current edit↑ and rest of edit↑ list
  If it is possible to go left then
    RPLACD edit↑ with CADR edit↑ list
    go right
  else if it is possible to go up then
    RPLACA edit↑ with CAAR edit↑ list
    go down
```

UNGROUP does list surgery (as below):



(A (B) C) ⟶ (A B C)

((A) B C) ⟶ (A B C)

UN*

Calling functions: PROCESS

Functions called: AT_BEGINNING
CHANGE↑

Description

```
If current edit↑ is atom then RETURN nil
else if at_beginning then
  current edit↑ <-- car current edit↑
ctherwise NCONC current edit↑ and rest of edit↑ list
  If it is possible to go left then
    go left
    RPLACD current edit↑ with CADR edit↑ list
    go right
  go up
  RPLACA current edit↑ with CAAR edit↑ list
  go down
```

CHANGE↑

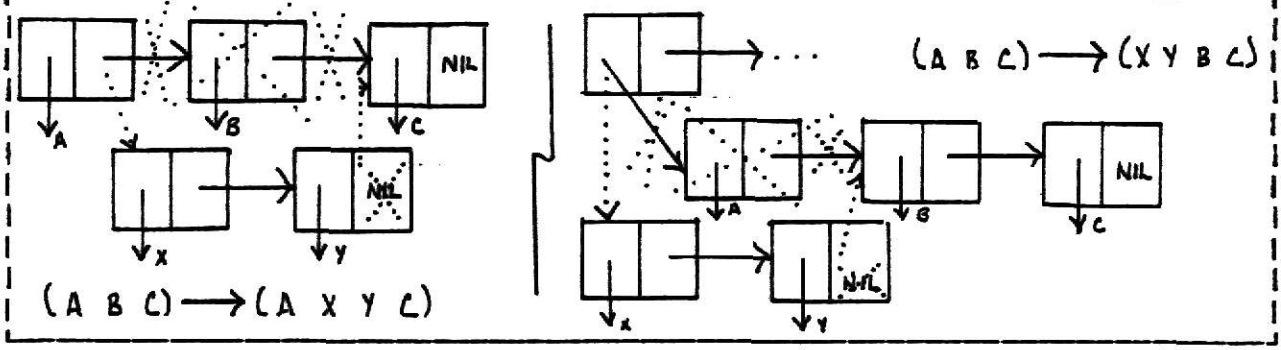
Calling functions: CHANGE_STR
UN*

Parameters: CHANGES

Functions called: ERROR MESSAGE
R L U D

Description

```
If current edit↑ is atom then RETURN nil
else if at_beginning then
  current edit↑ <-- car current edit↑
otherwise connect edit↑ with rest of edit↑ list
  If it is possible to go left then
    go left
    RPLACD current edit↑ with CADR edit↑ list
    go right
  else if it is possible to go up then
    go up
    RPLACA current edit↑ with CAAR edit↑ list
    go down
```



INSERT_AFTER↑

Calling functions: PROCESS
REPL
INSERT_BEFORE↑

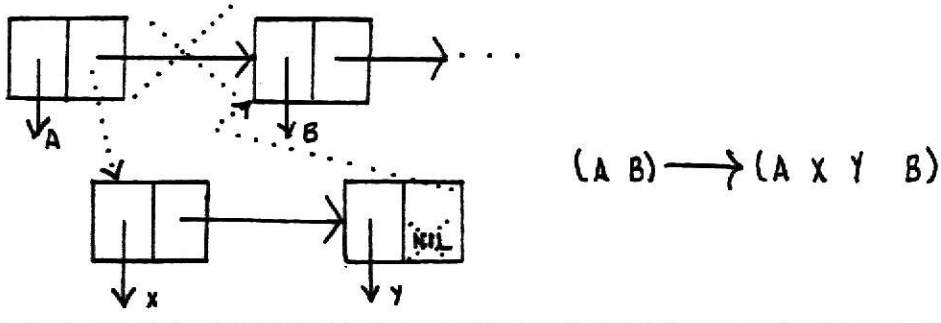
Parameters: INSERTION

Functions called: ERROR_MESSAGE

Description

```
if no insertion then
  Error_message 'no insertions   else  connect
insertion with rest of edit↑ list
  edit↑ <-- insertion
```

INSERT_AFTER↑ does list surgery as follows:



INSERT_BEFORE

Calling functions: PROCESS

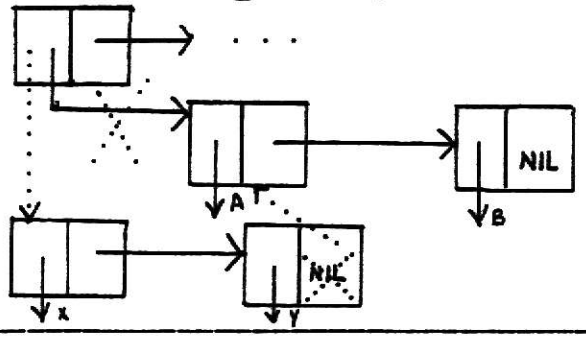
Parameters: INSERTION

Functions called: INSERT_AFTER↑
ERROR_MESSAGE

Description

```
If no insertion then
  ERROR_MESSAGE 'no insertion
else if it is possible to go left then
     go left
     Call INSERT_AFTER↑
     go right
   else if it is possible to go up then
     go up
     connect insertion with rest of edit↑
     edit↑ <-- insertion
     go down
   otherwise
     connect insertion with edit list
     edit↑ <-- insertion
     expr  <-- edit↑
     level <-- edit↑
```

INSERT_BEFORE↑ does list surgery as follows:



(A B) ⟶ (X Y A B)

R
R_MOVE
AT_END_OF_EDIT_LEVEL

Calling functions: PROCESS
INSERT_BEFORE↑
CHANGE_N
NEXT
FIND
LOCATE_NEXT

Description

R(ight)

If AT_END_OF_EDIT_LEVEL
(i.e., null cdr edit list) then
nil
Otherwise edit↑ list <--
R_MOVE (i.e.,cdr edit list)

L
AT_START_OF_EDIT_LEVEL

Calling functions: PROCESS
INSERT_BEFORE

Functions called: L_MOVE

Description

I
If AT_START_OF_EDIT_LEVEL
(i.e.,edit↑ = level) then
nil
otherwise edit↑ list <-- L_MOVE

L_MOVE

Calling functions: L

Functions called: ALL_BUT_FIRST

Description
Call ALL_BUT_FIRST (length level - length edit↑ list - 1) level

FOR EXAMPLE:

If level is (A B C D E) and

edit↑ is (C D E) then

L_MOVE is all but the first (5 - 3 - 1) = 1 of level:

(B C D E)

ALL_BUT_FIRST

Calling functions: L_MOVE

Parameters:
N: number to take off front
L: list

Functions called: ALL_BUT_FIRST !!!RECURSIVE!!!

Description

If end of L then nil

else if N = 0 then L

otherwise Call ALL_BUT_FIRST (N - 1) (rest of L)

U
AT_TOP_LEVEL
RESET↑

Calling functions: PROCESS
INSERT_BEFORE↑
CHANGE_N
NEXT
FIND
LOCATE_NEXT

Description

U

If AT_TOP_LEVEL
(i.e.,'TOP is on top of
edit positions stack) then
nil
else RESET↑
(i.e.,edit↑ <-- top of
edit positions stack)
Pop edit positions
RESET_LEVEL
(i.e.,level <-- top of
edit positions stack)
Pop edit positions

D
AT_BOTTOM_LEVEL
NEW↑
NEW_LEVEL

Calling functions: PROCESS
INSERT_BEFORE↑

Functions called:

Description
D If AT_BOTTOM_LEVEL (i.e.,edit↑ is atom) then nil else push level onto edit positions stack push edit↑ onto edit positions stack Call NEW_LEVEL (i.e.,level <-- car edit↑) Call NEW↑ (i.e.,edit↑ <-- car edit↑)

TOP

Calling functions: BACK_TO_START

Functions called: U TOP !!!RECURSIVE!!!

Description

```
If it is possible to go up then
  Go up
  Call TOP
  Go up until 'TOP is on top of stack
```

BACK_TO_START

Calling functions: PROCESS

Functions called:
TOP L

Description

Call TOP tc get to top level

Loop

go left (L)

Until not possible to go left

CHANGE_N

Calling functions: PROCESS
CHANGE_ATOM
CHANGE_STR

Local variables:
COUNT: loop counter
COPY_CHANGES: store copy of changes

Parameters:
KEY: what is to be changed
CHANGES
N: number of changes to be made

Functions called:
COPY
LEGAL_NUMBER_ARGS
ERROR_MESSAGE

Description

```
If null changes then
  ERROR_MESSAGE 'null changes
  RETURN If null key then
  ERROR_MESSAGE 'null key
  RETURN If n not legal arg the
  RETURN nil else if n = * then
  n <-- length edit↑ list else if n = 0 then
  ERROR_MESSAGE '0 changes
  RETURN

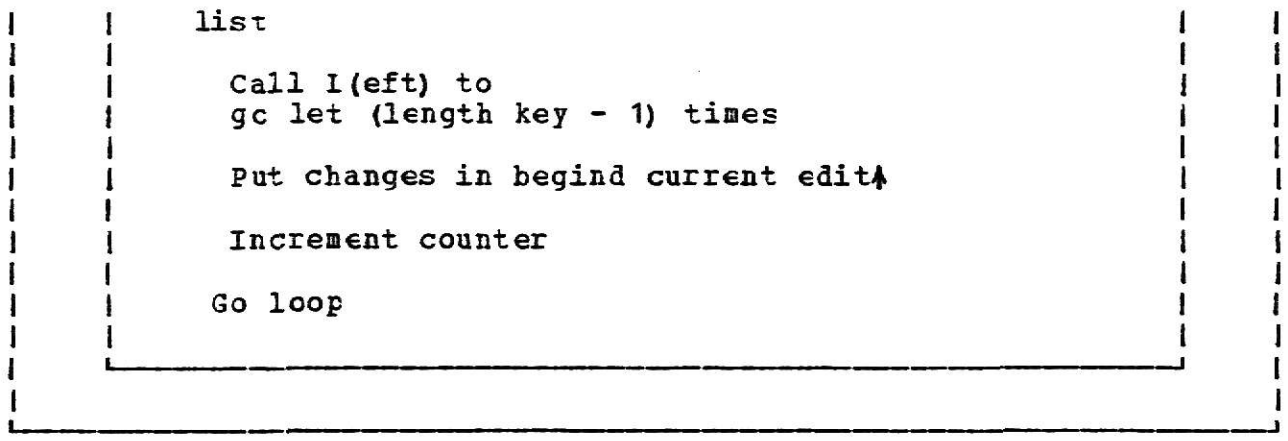
count <-- 1

Loop

  COPY changes

  If either n < count or
            key is not on rest of list then
     ERROR_MESSAGE (count - 1) 'changes
     RETURN
  Call R(ight) to
  go right (length key - 1) times

  Connect  copy of  changes with  rest of  edit↑
```


list
Call I(eft) to
gc let (length key - 1) times
Put changes in begind current edit
Increment counter
Go loop

CHANGE_ATOM

Calling functions: CHANGED_ATOM

local variables:
NEW_ATOM: atcm producd after substring changed
CHK: stores atom made into number

Parameters:
ATM: atom to be changed
OLD: substring to be changed
NEW: substring old to be changed to

Functions called: ERROR_MESSAGE
MAKE_NUMBER
CHANGE_N

Description

```
If old and new and atm are not atoms then
  ERROR_MESSAGE 'argument no atom
  RETURN
else if atm or old is null then
  ERROR_MESSAGE 'null atom/subatom
  RETURN

Call CHANGED_ATOM
NEW_ATOM <-- changed atcm

Call MAKE_NUMBER with new_atom
Store number in CHK

If CHK is a number then new_atm <-- CHK

else if new_atom is same as old atom then
  ERROR_MESSAGE old 'not found
  RETURN

Call CHANGE_N to change atm to new_atm n times
```

NEXT

Calling functions: FIND
LOCATE_NEXT
CHANGE_N

Functions called: R U

Description

```
Loop

  If it is not possible to go up then
    RETURN nil
  else go up
  If it is possible to go right then
    go right
  RETURN t
otherwise go loop
```

FIRST_N

Calling functions: LOCATE_NEXT
FIND
CHLIST

Parameters: N L

Functions called: FIRST_N !!!RECURSIVE!!!

Description

```
If at end of list then nil
else if n <= 0 then nil
otherwise (first elt of list
             (FIRST_N (n - 1)
                rest of list))
```

CHANGED_ATOM

Calling functions: CHANGE_ATOM

Parameters:
ATM: atom to be changed
OLD: substring to be changed
NEW: substring old is to be changed to

Functions called: CHLIST
EXPLODE1

Description

EXPLODE1 old new atm
(EXPLODE1 explode atom and
makes char digits numbers)

EXPLODE atm (leaves char digits)

Call CHLIST to put new in place
of old in exploded atm

Implode this list

Digits within atoms require special care.

FOR EXAMPLE:

If X1 is EXPLODE'd to (X 1) the '1' is not the same as the number 1. EXPLODE1 changes the '1' to the number 1. However, IMPLODE will not work properly on the number 1; so it is necessary to IMPLODE using the '1'. Thus both the usual EXPLODE and EXPLODE1 are necessary.

EXPLODE1

Calling functions: CHANGED_ATOM

Local variables:
L: exploded atm with char digits
NEW_ATOM: exploded atm with char digits converted to numbers

Parameters:
ATM: atom to be exploded

Functions called: MKNUM
EXPL_NUM

Description

```
If atm is a number then
  Call EXPL_NUM to explode atm
  RETURN
else if null atm then RETURN
L <-- EXPLODE atm
new_atom <-- (first elt on L)
Loop
  Pop L
  If at end of L then RETURN new_atom
  new-atom <-- (new_atom
                (first elt of I changed by
                 MKNUM to number if a digit))
Go Loop
```

This function works exactly like the standrd EXPLODE except that it converts char digits to their number value.

EXPL_NUM

Calling functions: EXPLODE1

Parameters:
N: number to be exploded

Functions called: EXPL_NUM !!!RECURSIVE!!!

Description

```
If n = 0 then nil
else ((EXPL_NUM n/10) (mod n 10))
```

CHLIST

Calling functions: CHANGED_ATOM

Parameters:

OLD: list
NEW: list
I1: exploded with number for digits
L2: exploded with char digits

Functions called: CHLIST !!!RECURSIVE!!!
EXPLODEDIG
FIRST_N

Description

If at end of L1 then nil
else if old = (first_n (length old) L1)
(that is, old matches first part
of L1) then
Call EXPLODEDIG to change numbers in new
to char digits
Append new to rest of L2 with old removed
otherwise CONS first-elt-L2 with
CHLIST old new rest of L1 and L2

EXPLODEDIG

Calling functions: CHLIST

Parameters: L: exploded atom with numbers

Functions called: MKDIG
EXPLODEDIG !!!RECURSIVE!!!

Description

```
If at end of L then nil
otherwise
  append first-elt-of-L
         (with number changed to
          char digit if necessary)
    with EXPIODEDIG rest of L
```

USER GUIDE FOR LEDIT:
A REAL-TIME LISP EDITOR

USER GUIDE TO LEDIT

TABLE OF CONTENTS

Introduction

How to Use LEDIT

Entering and Leaving LEDIT....ii
EDIT....iii
QUIT....iv
FILE....v
END....vi
HELP....vii
T(ype)....viii
R(ight)....ix
L(eft)....x
U(p)....xi
D(own)....xii
S(tart)....xiii
TOP....xiv
F(ind)....xv
G(roup)....xvi
UN(group)....xvii
DEL(ete)....xviii
CH(ange)....xix
IA (Insert After)....xx
IB (Insert Before)....xxi
CH(ange) |....xxii
CH(ange) ATOM....xxiii
CH(ange) ATOM |....xxiv
CH(ange) STR(ing)....xxv
RPT (Repeat)....xxvi
DEFCMD (Define Command)....xxvii
CH(ange) _LIST_DELIM(iter)....xxviii
CH(ange) _CMD_DELIM(iter)....xxix
REM(ove)....xxx
REPL(ace)....xxxi
DISPLAY....xxxii
CONTENTS....xxxiii

Sample Edit Session

HOW TO USE LEDIT
A REAL-TIME LISP EDITOR

This manual describes the use of a real_time editor for LISP. The use of this tool will enable you to edit property lists and function definitions without leaving the LISP interpreter. You may also store edited lists in your LISP files, if you wish. This editor provides you with most of the features of a conventional text editor, as well as some commands particularly useful to the LISP programmer.

The following terms are used throughout this document:

1. edit list: the subexpression of the property list you are editing. It contains the edit pointer as it first element.

2. edit pointer or edit↑: the element you wish to edit or examine. It is the first element of the current edit list. In this manual we often use an arrow to indicate the edit pointer.

3. edit level: It is possible to move up and down in levels of nesting in the edit list. The current edit level is the expression which contains the current edit list as a 'tail.'

FOR EXAMPLE:

If the current edit level is (A (B C) D). The current edit list might be (A (B C) D) or ((B C) D) or (D). The current edit↑ could be A or (B C) or D.

You may type as many commands on a line as you wish. The commands will be executed in sequence when you type carriage return. The interpreter reads all commands and delimiters as atoms, so be sure to leave blanks between them.

ENTERING AND LEAVING LEDIT

To enter the editor from the LISP interpreter type

(LEDIT <atom-name> <function-name or NIL>)

The first argument is the name of the atom whose property list you wish to edit. The second argument is the name of the property you wish to edit. If you type NIL for the second argument, the list you will receive to edit will be the entire property list of the atom.

FOR EXAMPLE:

The atom A1 has the following property list:

(P1 (X Y) P2 (Q R) P P3 (S T))

===> (EDIT A1 P2)
results in the edit list

(P2 (Q R))

===> (EDIT A1 NIL)
results in the edit list

(P1 (X Y) P2 (Q R) P3 (S T))

To exit from the editor type the command END.

The response will be 'QUIT or FILE?'.

If you wish to discard the edited version, type QUIT.

If you wish the original property list to be replaced by the edited version, type FILE.

EDIT

EDIT <atom-name> <property-name or NIL>

It is possible to edit another property list while you are already in the editor. Simply use the EDIT command (without parentheses or quotes). When you type END, you will be returned to the edit list you were previously editing. You may EDIT as many lists as you wish while you are in the editor.

FOR EXAMPLE:

The property list of atom A is as follows:

(PROP1 (X Y) PROP2 (Z W) EXPR (LAMBDA (X) (CAR (CADDR X))))

===> EDIT A PROP2
results in the edit list

(PROP2 (Z W))

===> EDIT A NIL
results in the edit list

(PROP1 (X Y) PROP2 (Z W) EXPR (LAMBDA (X) (CAR (CADDR X))))



QUIT

This command simply causes the editor to discard the edited version of the list you have been editing; no changes will be made to the original property list.

The result will be ABORTED.

FILE

FILE

This command replaces the original property list with the edited version of the list.

The terminal will prompt with 'DO YOU WISH TO STORE IN A FILE?
'TYPE YES OR NO

If you type YES, the prompt will be 'TYPE FILE NAME.

The new property list will be stored in the LISP file which you specify.

END

This command causes you to end editing of that property list. The editor will prompt you with 'QUIT OR FILE?'.

QUIT discards the edited property list; the original property list remains unchanged.

FILE replaces the original property list with the new edited version.

HELP

HELP <edit-command or ALL>

HELP ALL displays a list of all edit commands.

HELP <edit-command> displays an explanation of the syntax and function of the command.

T(ype)

T

T(ype) causes the current edit pointer list to be display in 'pretty LISP' format. The edit pointer will be the car of that list.

FOR EXAMPLE:

If the result of T is ((X Y) Z), then the edit pointer is (X Y).

R(ight)

R or R<n> or R*

R(ight) causes the current edit pointer to be moved to the right the given number of S-expressions. If no number is specified, the default is one. R* causes the editor to go right until at the end of that edit level.

FOR EXAMPLE:

The current edit list is (A B (C D) E F).

===> R results in the edit list (E (C D) E F).

===> R2 results in the edit list (E F).

===> R* results in the edit list (F).

If there are fewer S-expressions to the right of the edit pointer than you have specified, the editor will simply move the pointer as far right as it can. A message will be printed to tell you how many moves were completed.

FOR EXAMPLE:

The current edit list is (A (E C) D E).

===> R4 results in the edit list (E)
3 REPETITIONS

L(eft)

L or L<n> or L*

L(eft) causes the current edit pointer to be moved left the given number of S-expressions. If no number is specified, the default is one. L* causes the editor to go to the left until it is at the beginning of the current edit level.

FOR EXAMPLE:

```
      The current edit level is        (M (N) X A B (C D) E F).
      The current edit list is                 ((C D) E F).

  ===> L   results in the edit list              (E (C D) E D).

  ===> L2  results in the edit list          (X A B (C D) E F).

  ===> L*  results in the edit list  (M (N) X A B (C D) E F).
```

If the edit pointer reaches the beginning of the current edit level before it has completed the number of moves specified, it will go as far left as it can. An error message will be displayed to tell you how many moves were completed.

FOR EXAMPLE:

```
      The current edit level is          (A B C (D E) F)
      The current edit list is              ((D E) F)

  ===> L5 results in the edit list     (A B C (D E) F)
                                         3 REPETITIONS
```

U(p)

U or U<n> or U*

U(p) causes the current edit pointer to be moved up the given number of levels. If no number is specified, the default is one. U* causes the editor to move up until it is at the top level.

FOR EXAMPLE:

The current edit expression is (X Y (A (B (C ((D))) E))).

The current edit list is (D).

===> U results in the edit list ((D)).

===> U2 results in the edit list ((C ((D))) E).

===> U* results in the edit list ((A (B (C ((D))) E))).

If there are fewer levels above the edit pointer than you have specified, the editor will simply move as far up as it can. A message will be printed to tell you how many moves were completed.

FOR EXAMPLE:

The current edit expression is (X Y (A (B (C ((D))) E))).

The current edit list is ((C ((D))) E).

===> U5 results in the edit list ((A (B (C ((D))) E)))
2 REPETITIONS.

D(own)

D or D<n> or D*

D(own) causes the current edit pointer to be moved down the given number of levels. If no number is specified the default is one. D* causes the editor to move down until it has reached the bottom level.

FOR EXAMPLE:

```
      The current edit expression is     (X Y (A (B (C ((D))) E)))

      The current edit list              (Y (A (B (C ((D))) E)))

===> D results in the edit list                (B (C ((D))) E).

      The current edit list is                      (((D))).

===> D2  results in the edit list                     (D).

      The current edit list is                      (((D))).

===> D*  results in the edit list                     (D).
```

If there are fewer levels below the edit pointer than you have specified, the editor wil simply move as far down as it can. A message will be printed to tell you how many moves were completed.

FOR EXAMPLE:

```
      The current edit list is        ((B C (D)) E)

===> D5 results in the edit list       (B C (D)).
                                        1 REPETITION
```

S(tart)

S

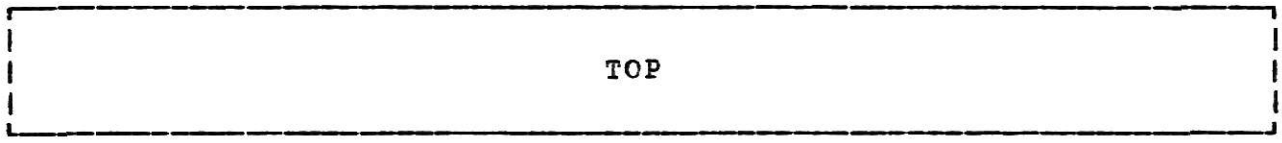
S(tart) moves the edit pointer back to the beginning of the edit expression. Its value will be the entire edit expression in its edited version.

FOR EXAMPLE:

The edit expression is (A (B C (D) E)).

The edit list is (D).

===> S
results in (A (B C (D) E)).



TOP

TOP moves the current edit pointer to the top level. It is equivalent to U*.

F(ind)

F <one or more S-expressions> /

Beginning with the current edit pointer, F(ind) will locate the given argument. The current edit pointer will then point to the argument (or the first expression in the argument if it was a list of S-expressions).

If the argument is not located, the error message 'NOT FOUND' will be displayed. The current edit pointer will then be restored to the position in which the search began.

FOR EXAMPLE:

```
      The current edit list is      (A B (C D E (A (E)) C) D)

===> F A B /
     results in the edit list       (A B (C D E (A (B)) C) D)

===> F D /
     results in the edit list             (D E (A (B)) C) D)

===> F (A (B)) /
     results in the edit list                  ((A (E)) C)

===> F X /
     results in the edit list                  ((A (E)) C)
                                                  NOT FOUND
```

G(roup)

G or G<n> or G*

G(roup) causes the editor to group the given number of S-expressions beginning with the current edit pointer. If no number is specified, the default is one. G* causes the editor to group S-expressions until it reaches the end of the edit level.

FOR EXAMPLE:

```
      The current edit list is        (A B (C) (D E) F G)

===> G
     results in the edit list       ((A) B (C) (D E) F G)

===  G3
     results in the edit list    (((A) B (C)) (D E) F G)

===> G*
     results in the edit list   ((((A) B (C)) (D E) F G))
```

If there are fewer S-expressions to the right of the edit pointer than you have specified, the editor will display an error message. No change will have taken place.

The editor will not group if you are at the beginning of the edit entire property list.

FOR EXAMPLE:

```
      The edit list is           (X Y Z).

===> G
     results in the edit list   (X Y Z).
                                  AT BEGINNING
```

No change will have taken place.

(UN(group)

UN or UN<n> or UN*

UN(group) ungroups the current edit pointer the number of times specified. If no number is specified, the default is one. UN* ungroups the edit pointer until it is an atom.

FOR EXAMPLE:

```
      The current edit list is          (((((A B))) C)).

===> UN
     results in the edit list           ((((A B))) C).

===> UN2
     results in the edit list            (((A B)) C).

===> UN*
     results in the edit list            (A B C).
```

If the current edit pointer becomes an atom before ungroup executes the given number of times, a message will be printed to tell you how many ungroups were done.

FOR EXAMPLE:

```
     The current edit list is          (((A) B) C).

===> UN5
     results in the edit list            (A B C)
                                          2 REPETITIONS
```

DEL(ete)

DEL or DEL<n> or DEL*

Beginning with the current edit pointer, the given number of S-expressions will be deleted by the editor. If no number is specified, the default is one. DEL* will delete everything past and including the current edit pointer. The current edit pointer will now be NIL.

FOR EXAMPLE:

The current edit level is (A B C (D E) (F (G)) H).

The current edit list is (B C (D E) (F (G)) H).

===> DEL
results in the edit list (C (D E) (F (G)) H)

===> DEL2
results in the edit list ((F (G)) H).

===> DEL*
results in the edit list NIL.

If there are fewer S-expressions on the current level than the number you have specified, the editor will delete the rest of the level. The current edit pointer will be NIL. An error message will be displayed to tell you how many deletions occurred.

FOR EXAMPLE:

The current edit list is (A B (C D)).

===> DEL5
result in the edit pointer NIL
AT BOTTOM LEVEL
3 DELETIONS

CH(ange)

```
CH <old list of one or more S-expressions> /
   <new list of one or more S-expressions> /
                    <n or *>
```

CH(ange) finds the old list which is to be changed and changes it to the new list. It begins the search at the current edit pointer. The given number of changes will be made. There must be a number specified; there is no default. CH old / new / * will find and change all occurrences of old which are found between and including the current edit pointer and the end of the edit list. The edit pointer will point at the last change. If CH* is the command, the changes will be made throughout the rest of the edit list beginning with and including the current edit pointer. The current edit pointer will be at the end of the edit list.

FOR EXAMPLE:

```
      The current edit list is        (A (B C) D (A (B) C D) A)

===> CH B / (BBB) / 1
     results in the edit list         (BBB C) D (A (B) C D) A)

===> CH C D / X (Y) / 1
     results in the edit list                    ((X (Y)) A)

===> CH Y / M N / *
     results in the edit list                              (A)
                                  {<=== ((X (M N)) A) }
```

If the editor finds fewer instances of the old list than you specified, it will make the necessary changes until it reaches the end of the edit list. An error message will be displayed to tell you how many changes were made. The edit pointer will be at the end of the edit list.

FOR EXAMPLE:

```
     The current edit list is       (A B (C D) E F).

===> CH D / ((X)) / 3
     results in the edit list                    (F)
                                          1 CHANGE
                                  {<===   ((C ((X))) E F) } 10
```

I(nsert) A(fter)

ia <insertions-one-or-more-S-expressions>

IA inserts the given insertions directly after the current edit pointer.

FOR EXAMPLE:

The current edit list is (A (B C)).

===> IA X Y (Z) /
results in the edit list (A X Y (Z) (B C)).

===> IA (M) /
results in the edit list (A (M) X Y (Z) (B C)).

I(nsert) B(efore)

IB <new-list-of-one-or-more-s-expresssions> /

IB inserts the given insertion directly before the current edit|. The current edit| will then be at the beginning of the insertion.

FOR EXAMPLE:

The current edit list is	(A B (C))
===> IB (M) X /	
results in the edit list	((M) X A B (C))
===> IB (Q R S) /	
results in the edit list	((Q R S) (M) X A B (C))

C(hange)/

CH/ <new-list-of-one-or-more-S-expressions> / <n> or *

CH/ is exactly like CH(ANGE) except that the old list to be changed is the current edit↑.

FOR EXAMPLE:

The current edit list is (A (B C) (A (B (C)) A))

===> CH/ M N / 2
results in the edit list (M N (B (C)) A)
{<== (M N (B C) (M N (B (C)) A)

CH(ange)ATOM

CHATOM <old-atom> / <old-atom-substring> /
<new-atom-substring> / <n> or *

CHATOM changes the old-atom-substring contained in the old-atom to the new-atom-substring the given number of times. A number must be specified; there is no default. The editor will find the given number of the specified old-atom. If * is specified, the editor will look for the atom and make changes until it reaches the end of the edit list. It is possible to use nothing for the new-substring. This has the effect of deleting the old-substring.

FOR EXAMPLE:

```
      The current edit list is        (ABC1 XYZ2 234 P12QRSABCX)
===> CHATOM ABC1 / C1 / YYY / 1
      results in the edit list        (ABYYY XYZ2 234 P12QRSABCX)

===> CHATOM 2 / 2 / 234 / 1
      results in the edit list          (234 234 P12QRSABCX)

===> CHATOM 234 / 23 /AB / 2
      results in the edit list          (AB4 AB4 P12QRSABCX)
```

Care must be taken with numbers. A digit followed by a nondigit is not a legal atom. Thus CHATOM 234 / 4 / X / 1 would be illegal since 23X is not a legal atom.

If the editor finds fewer instances of the atom than you specified, it will continue until it reaches the end of the edit list. An error message will be displayed to tell you how many changes were made.

CH(ange)ATOM↑

CHATOM↑ <old-substring> / <new-substring> / <n> or *

CHATOM↑ performs exactly the same function that CHATOM does except that the atom to be changed is the current edit↑ (and any subsequent identical atoms).

FOR EXAMPLE:

The current edit list is (ABC (XYZ (AEC ABC) Q)

==>CHATOM↑ BC / XXX / 3
results in the edit list (AXXX)
{<==
(AXXX (XYZ (AXXX AXXX) Q))

CH(ANGE) STR(ING)

CHSTR <old-substring> / <new-substring> / <n> or *

CHSTR looks at each atom on the edit list (beginning with and including the current edit|) for the given oldsubstring and changes it to the given new-sub-string the number of times specified. A number must be specified; there is no default. If * is specified, the editor will continue to the end of the edit list. It is possible to use a nil new-atom. This has the effect of deleting old string from the atom.

FOR EXAMPLE:

The current edit list is
(ABC (XYZ (CBA YZW) ABQRS CABRS)

```
===> CHSTR AB / I / 3
     results in  the edit list     (CIRS)
{<==
                  (IC (XYZ (CBA YZW) IQRS CIRS)
```

If the editor finds fewer occurrences of the string than you specified, it will continue until it reaches the end of the edit list. An error message will be displayed to tell you how many changes were made. Be careful not to use or create any illegal (i.e,a leading digit followed by a nondigit --like 12X).

R(e)P(ea)t

RPT <list-of-edit-commands> ENDCMD <n>

RPT allows the user to list a string of edit commands and the editor will repeat the sequence the given number of times. A number must be specified. There is no default: * is illegal with RPT. The editor will stop the repetition if a comand does not execute. An error message will be displayed to tell you how many times the sequence was executed.

FOR EXAMPLE:

```
      The current edit list is    (A B C (D E (F) G) H)

      The current edit list is          (B C (D E (F) G) H)

===> RPT R DEL D P ENDCMD
     results in                         ((D E (F) G)
     current edit list                            (F)

    The current edit list is         (A B (C A B (A (B))))

===> RPT R CH A / AAA / 1 T ENDCMD 2
     results in                        (AAA B (A (B)))
     current edit list                        (AAA (B))

    The current edit list is      (A (B) (C) ((D)) X (Z) V)

===> RP T UN ENDCMD 5
     results in edit list                          (X (Z) V)
                                          3 REPETITIONS
    {<===     (A B C (D) X (Z) V)
```

DEF(ine) C(om) M(an) D

DEFCMD <name-of-user-defined-edit-command>
<list-of-edit-commands>
ENDCMD

DEFCMD allows the user to define his own edit commands using combinations of the existing commands. The following commands are not allowed in DEFCMD lists: HELP, EDIT, DEFCMD, RPT. The user-defined command may then be called at any time during the edit session. It may also be called during subsequent edit sessions as long as you are still in the LISP interpreter. The command has been DEFUN'ed, so it may be saved to a LISP file just as you might save any other functions. User-defined edit commands are not allowed to have arguments.

FOR EXAMPLE:

```
DEFCMD USER1 R2 UN* S CH X / Y / 3 S T ENDCMD
```

When you type USER1 any time you are in edit mode it will perform the command sequene in the definition. User commands may be used within user-command definitions.

CH(ange)_LIST_DELIM(iter)

CH_LIST_DELIM <S-expression>

CH_LIST_DELIM allows you to change the delimiter for the argument lists used with F(ind), CH(ange), CH(ange) CH(ange)ATOM, CH(ange)ATOM, I(nsert)A(fter), I(nsert)B(efore), and CH(ange)STR(ing)

FOR EXAMPLE:

```
===> CH_LIST_DELIM  ***
```

will allow you to type CHSTR /// *** /X/ ***

The changed delimiter will remain in effect until you change it or get out of the editor.

CH(ange)_C(om)M(an)D_DELIM(iter)

CH_CMD_DELIM <S-expression>

CH_CMD_DELIM allows you to change the delimiter for command lists used with RPT and DEFCMD.

FOR EXAMPLE:

```
===>  CH_CMD_DELIM  ***
```

will alow you to type DEFCMD R T TCP D3 ***

The changed delimiter will remain in effect until you change it or get out of the editor.

REM(ove)

REM <buffer-name> <n> or *

REM causes the editor tc remove the given number of S-expressions (beginning with and including the current edit pointer) and place them in the given buffer. A number must be specified; there is no default. If * is specified, the rest of the edit level (beginning with and including the current edit pointer) will be removed. If fewer S-expressions remain than you have specified, the editor will remove as many as there are on that edit level; an error message ewill be displayed to tell you how many S-expressions were removed.

FOR EXAMPLE:

The current edit expression is (A B (C) D (E F))

The current edit list is (B (C) D (E F))

===> REM BUFFER1 3
results in the edit list ((E F))
{<=== (A (E F))

BUFFER1 = (B (C) D)

REPL(ace)

REPL <buffer-name> <starting-number> <n> or *

REPL causes the editor to retrieve the given number of lines from the buffer (or the entire buffer if * is specified) beginning with the starting number given. A starting number and number of S-expressions must be specified; there is no default. The S-expressions will be inserted after the current edit pointer.

FOR EXAMPLE:

The current edit list is (A B)

The contents of BUFFER1 is (S1 S2 S3 S4 S5 S6)

===> REPL BUFFER1 2 4
results in the edit list (A S2 S3 S4 S5 B)

No S-expressions are removed from the buffer.

DISPLAY

DISPLAY

DISPLAY causes the editor to display the names of all buffers that have been created.

CONTENTS

CONTENTS <buffer-name>

CONTENTS causes the editor to display the contents of the given buffer.

SAMPLE EDIT SESSION

```
(EDIT DOUBLELIST)
.T
---------------
(EXPR (LAMBDA (L)
              (CONS ((NULL L))
                    (CDR L)
                    (T (APPEND (IMPLOD (CONS (EXPLODE (CAR L))))
                               (EXPLODE (CAR L))
                               (DOUBLELIST (CDR L)) ) ) ) ) )
---------------
.R D R2 D T
---------------
(CONS ((NULL L))
      (CDR L)
      (T (APPEND (IMPLOD (CONS (EXPLODE (CAR L))))
                 (EXPLODE (CAR L))
                 (DOUBLELIST (CDR L)) ) ) )
---------------
.CHATOM^ S / D / 1 T
     <<< 1 CHANGES >>>
---------------
(COND ((NULL L))
      (CDR L)
      (T (APPEND (IMPLOD (CONS (EXPLODE (CAR L))))
                 (EXPLODE (CAR L))
                 (DOUBLELIST (CDR L)) ) ) )
---------------
.R2 CH^ NIL / 1 T
     <<< 1 CHANGES >>>
---------------
(NIL (T (APPEND (IMPLOD (CONS (EXPLODE (CAR L))))
                (EXPLODE (CAR L))
                (DOUBLELIST (CDR L)) ) ) )
---------------
.L T
---------------
((NULL L)) NIL
           (T (APPEND (IMPLOD (CONS (EXPLODE (CAR L))))
                      (EXPLODE (CAR L))
                      (DOUBLELIST (CDR L)) ) ) )
---------------
.UN T
---------------
((NULL L) NIL
          (T (APPEND (IMPLOD (CONS (EXPLODE (CAR L))))
                     (EXPLODE (CAR L))
                     (DOUBLELIST (CDR L)) ) ) )
---------------
.G2 L T
---------------
(COND ((NULL L) NIL)
      (T (APPEND (IMPLOD (CONS (EXPLODE (CAR L))))
                 (EXPLODE (CAR L))
                 (DOUBLELIST (CDR L)) ) ) )
---------------
.CH IMPLOD / IMPLODE / 1 S T
     <<< 1 CHANGES >>>
---------------
(EXPR (LAMBDA (L)
              (COND ((NULL L) NIL)
                    (T (APPEND (IMPLODE (CONS (EXPLODE (CAR L))))
                               (EXPLODE (CAR L))
                               (DOUBLELIST (CDR L)) ) ) ) ) )
---------------
```

```
.F IMPLODE / F D T
---------------
(CONS (EXPLODE (CAR L)))
---------------
.CH^ APPEND / 1 T
     <<< 1 CHANGES >>>
---------------
(APPEND (EXPLODE (CAR L)))
---------------
.CHATOM DOUBLELIST / E / E_ / 1 T
     <<< 1 CHANGES >>>
---------------
(DOUBLE_LIST (CDR L))
---------------
.S T
---------------
(EXPR (LAMBDA (L)
              (COND ((NULL L) NIL)
                    (T (APPEND (IMPLODE (APPEND (EXPLODE (CAR L))))
                               (EXPLODE (CAR L))
                               (DOUBLE_LIST (CDR L)) ) ) ) ) ) )
---------------
.F APPEND / CH^ CONS / 1 T
     <<< 1 CHANGES >>>
---------------
(CONS (IMPLODE (APPEND (EXPLODE (CAR L))))
      (EXPLODE (CAR L))
      (DOUBLE_LIST (CDR L)) )
---------------
.R T
---------------
((IMPLODE (APPEND (EXPLODE (CAR L))))
 (EXPLODE (CAR L))
 (DOUBLE_LIST (CDR L)) )
---------------
.UN T
---------------
(IMPLODE (APPEND (EXPLODE (CAR L)))
         (EXPLODE (CAR L))
         (DOUBLE_LIST (CDR L)) )
---------------
.R UN T
---------------
(APPEND (EXPLODE (CAR L)) (EXPLODE (CAR L)) (DOUBLE_LIST (CDR L)))
---------------
.G3 L G2 T
---------------
((IMPLODE (APPEND (EXPLODE (CAR L)) (EXPLODE (CAR L))))
 (DOUBLE_LIST (CDR L)) )
---------------
.S T
---------------
(EXPR (LAMBDA (L)
              (COND ((NULL L) NIL)
                    (T (CONS (IMPLODE
                               (APPEND
                                (EXPLODE (CAR L))
                                (EXPLODE (CAR L)) ) )
                             (DOUBLE_LIST (CDR L)) ) ) ) ) )
---------------
.END
     QUIT OR FILE?
.FILE
     DO YOU WISH TO PLACE IN A FILE?
     TYPE YES OR NO
```

```
(CLIENT CUST)
.T
---------------
(NAME ACME
      PHONE
      5321234
      CRED_RTG
      AAA
      ORDERS
      (426 231) )
---------------
.R7 D T
---------------
(426 231)
---------------
.R DEL S T
     <<< END_OF_LEVEL >>>
---------------
(NAME ACME
      PHONE
      5321234
      CRED_RTG
      AAA
      ORDERS
      (426) )
---------------
.F AAA / CH^ BAD / 1 T
$     <<< 1 CHANGES >>>
---------------
(BAD ORDERS (426))
---------------
.P2 D IB 148 / S T
---------------
(NAME ACME
      PHONE
      5321234
      CRED_RTG
      BAD
      ORDERS
      (148 426) )
---------------
.R2 IA 5324321 / R G2 S T
---------------
(NAME ACME
      PHONE
      (5324321 5321234),
      CRED_RTG
      BAD
      ORDERS
      (148 426) )
---------------
.END
     QUIT OR FILE?
.QUIT
     ABORTED
     LISP
.
```

APPENDIX I

The LEDIT Program Code

FILE: EDJANA LISP A1 KANSAS STATE UNIVERSITY VM/SP CMS

```
;  THE LISP PROGRAM FOR LEDIT, A REAL-TIME LISP EDITOR.


(DEFUN = (S1 S2) (EQUAL S1 S2))
(DEFUN GT (X Y) (GREATERP X Y))
(DEFUN LT (ARG1 ARG2)  (LESSP ARG1 ARG2))
(DEFUN LE (ARG1 ARG2)  (OR (LT ARG1 ARG2)  (= ARG1 ARG2)))

(CSETQ LITERALC '@)


(SETQ *USER_COMMAND* NIL)

(DEFUN LEDIT FEXPR  (ARGS)
       (PROG (ATOM_NAME PROPERTY_NAME
              *EXPRESSION*
              *EDIT|*
              *LEVEL*
              *EDIT_POSITIONS*
              *ORIGINAL_EXPRESSIONS*
              *STORED_EXPRESSIONS*
              *ATOM_NAMES*
              *PROP_NAMES*
              *RPT_CMD*
              *BUFFERS*
              *LIST_DELIMETER* *CMD_DELIMETER*)
             (SETQ *ORIGINAL_EXPRESSIONS*  NIL)
             (SETQ *EDIT_POSITIONS*        NIL)
             (SETQ *ATOM_NAMES*            NIL)
             (SETQ *PROP_NAMES*            NIL)
             (SETQ *BUFFERS*               NIL)
             (SETQ *EXPRESSION*            '=DONE=)
             (SETQ *LIST_DELIMETER*        '/)
             (SETQ *CMD_DELIMETER*         'ENDCMD)
             (SETQ *STORED_EXPRESSIONS*    NIL)
             (COND ((= (LENGTH ARGS) 0)
                    (RETURN (ERROR_MESSAGE NIL 'TOO_FEW_ARGUMENTS))))
             (SETQ ATOM_NAME (CAR ARGS))
             (COND ((NULL (CDR ARGS)) (SETQ PROPERTY_NAME NIL))
                   (T                 (SETQ PROPERTY_NAME (CADR ARGS))))
            LOOP
             (EDIT_NEW_SEXPR ATOM_NAME PROPERTY_NAME)
             (COND (( = *EXPRESSION* '=DONE=) (RETURN 'LISP)))
             (GO LOOP) ) )


(DEFUN EDIT_NEW_SEXPR (ATOM_NAME PROPERTY_NAME)
       (SETQ *STORED_EXPRESSIONS* (CONS *EXPRESSION* *STORED_EXPRESSIONS*))
       (SETQ *EXPRESSION* (PROPERTY_LIST_TO_EDIT
                           ATOM_NAME PROPERTY_NAME))
       (COND ((NULL *EXPRESSION*)
              (ERROR_MESSAGE NIL 'NO_PROPERTY_LIST_FOUND)
              (SETQ *EXPRESSION* (CAR *STORED_EXPRESSIONS*))
              (SETQ *STORED_EXPRESSIONS* (CDR *STORED_EXPRESSIONS*)))
             (T (INITIALIZE_EDIT_MARKERS)
              (PROG (COMMAND)
                    LOOP
```

```
                    (SETQ COMMAND (READ))
                    (COND ((= COMMAND 'END)
                           (END)
                           (RETURN (RESTORE_EDIT_MARKERS))))

                    (PROCESS COMMAND)
                    (GO LOOP) ))))

(DEFUN PROPERTY_LIST_TO_EDIT (ATOM_NAME PROPERTY_NAME)
       (COND ((NULL PROPERTY_NAME) ATOM_NAME)
             (T (FIND_PROPERTY ATOM_NAME PROPERTY_NAME))))

(DEFUN FIND_PROPERTY (PROPERTY_LIST PROPERTY)
       (COND ((NULL (CDR PROPERTY_LIST)) NIL)
             ((= (CADR PROPERTY_LIST) PROPERTY) PROPERTY_LIST)
             (T (FIND_PROPERTY (CDDR PROPERTY_LIST) PROPERTY))))

(DEFUN ERROR_MESSAGE (ARG MESSAGE)
       (PRINC '@<<< @)
       (COND (ARG (PRINC ARG) (PRINC '@ @)))
       (PRINC MESSAGE) (PRINC '@ >>>@) (TERPRI) )

(DEFUN INITIALIZE_EDIT_MARKERS ()
       (SETQ *ORIGINAL_EXPRESSIONS* (CONS *EXPRESSION* *ORIGINAL_EXPRESSIONS*))
       (COND ((NULL PROPERTY_NAME)
              (SETQ *EXPRESSION* (COPY (CDR ATOM_NAME))))
             (T (SETQ *EXPRESSION*
                      (COPY (APPEND (LIST (CADR *EXPRESSION*))
                                    (LIST (CADDR *EXPRESSION*))))))))
       (SETQ *EDIT_POSITIONS* (CONS '=TOP= *EDIT_POSITIONS*))
       (SETQ *LEVEL* *EXPRESSION*)
       (SETQ *EDIT|* *EXPRESSION*)
       (SETQ *ATOM_NAMES* (CONS ATOM_NAME *ATOM_NAMES*))
       (SETQ *PROP_NAMES* (CONS PROPERTY_NAME *PROP_NAMES*)))

(DEFUN COPY (L)
       (COND ((ATOM L) L)
             ( T (CONS (COPY (CAR L))
                       (COPY (CDR L)) ))))

(DEFUN END ()
     (PROG (END_COMMAND)
           LOOP
           (PRINC '@QUIT OR FILE?@) (TERPRI)
           (SETQ END_COMMAND (READ))
           (COND ((MEMBER END_COMMAND '(QUIT FILE))
                  (RETURN (EVAL (LIST END_COMMAND)))))
           (GO LOOP)))


(DEFUN QUIT () (PRINT 'ABORTED))

(DEFUN FILE ()
       (PRINC '@DO YOU WISH TO PLACE IN A FILE?@) (TERPRI)
       (PRINC '@TYPE YES OR NO@) (TERPRI)
       (COND ((= (READ) 'YES) (READ) (TERPRI)))
       (SP (COND (PROPERTY_NAME
```

```
                (DELETE_PROPERTY ATOM_NAME PROPERTY_NAME)
                (PUTPROP ATOM_NAME
                         (CADR *EXPRESSION*)
                         (CAR *EXPRESSION*)))
               (T (RPLACD ATOM_NAME *EXPRESSION*)))))

(DEFUN DELETE_PROPERTY (ATOM PROPERTY)
       (RPLACD (CAR *ORIGINAL_EXPRESSIONS*)
               (CDDDR (CAR *ORIGINAL_EXPRESSIONS*))))

(DEFUN RESTORE_EDIT_MARKERS ()
      (SETQ *ORIGINAL_EXPRESSIONS* (CDR *ORIGINAL_EXPRESSIONS*))
       (SETQ  *ATOM_NAMES* (CDR *ATOM_NAMES*))
       (SETQ *PROP_NAMES* (CDR *PROP_NAMES*))
       (RESTORE_*EDIT_POSITIONS*)
       (SETQ *EXPRESSION* (CAR *STORED_EXPRESSIONS*))
       (SETQ *STORED_EXPRESSIONS* (CDR *STORED_EXPRESSIONS*))
       (COND ((= (CAR *EDIT_POSITIONS*) '=TOP=)
              (SETQ *LEVEL* *EXPRESSION*)
              (SETQ *EDIT|* *EXPRESSION*))
             (T (SETQ *LEVEL* (CADR *EDIT_POSITIONS*))
                (SETQ *EDIT|* (CAR *EDIT_POSITIONS*)))))

(DEFUN RESTORE_*EDIT_POSITIONS* ()
       (COND ((= (CAR *EDIT_POSITIONS*) '=TOP=)
              (SETQ *EDIT_POSITIONS* (CDR *EDIT_POSITIONS*)))
             (T (SETQ *EDIT_POSITIONS*
                      (CDDR *EDIT_POSITIONS*))
                (RESTORE_*EDIT_POSITIONS*))))

(DEFUN PROCESS (COMMAND)
       (COND ((MEMBER COMMAND *USER_COMMAND*) (COMMAND))

             ((= 'HELP COMMAND) (HELP (READ)))

             ((= 'T COMMAND) (SP *EDIT|*) T)

             ((= 'EDIT COMMAND)
               (EDIT_NEW_SEXPR  (READ) (READ)))

             ((= 'DEFCMD COMMAND)
               (PRINC '@--- ENTER USER COMMAND NAME: @)
               (TERPRI)
               (DEFCMD (READ)))

             ((= 'RPT COMMAND) (SETQ *RPT_CMD* (GENSYM '$*%!))
                               (DEFCMD *RPT_CMD*)
                               (RPT_USER *RPT_CMD* (READ)))

             ((= 'CH_CMD_DELIM COMMAND)
               (SETQ *CMD_DELIMETER* (READ))
               (PRINC '@--- COMMAND DELIMETER NOW: @)
               (PRINC *CMD_DELIMETER*) (PRINC '@ ---@) (TERPRI))

             ((= 'CH_LIST_DELIM COMMAND)
               (SETQ *LIST_DELIMETER* (READ))
               (PRINC '@--- *LIST_DELIMETER* NOW: @)
```

```
        (PRINC *LIST_DELIMETER*) (PRINC '@ ---@) (TERPRI))

      ((= 'REM COMMAND) (REMOVE (READ) (READ)))

      ((= 'REPL COMMAND) (REPLACE (READ) (READ) (READ)))

      ((= 'DISPLAY COMMAND) (DISPLAY_BUFFER_NAMES))

      ((= 'CONTENTS COMMAND) (DISPLAY_BUFFER_CONTENTS (READ)))

      ((= 'UN COMMAND) (UNGROUP))

      ((= 'UN* COMMAND)
        (REPEAT 'UN* '*))

      ((= (FIRST_N 2 (EXPLODE COMMAND)) '(U N))
        (REPEAT 'UNGROUP (MAKE_NUMBER (CDDR (EXPLODE COMMAND)))))

      ((= 'G (CAR (EXPLODE COMMAND)))
        (GROUP (NUMBER_OF_REPETITIONS)))

      ((= 'DEL COMMAND) (DELETE))

      ((= 'DEL* COMMAND) (DELETE_N '*))

      ((= (FIRST_N 3 (EXPLODE COMMAND)) '(D E L))
        (DELETE_N (MAKE_NUMBER (CDDDR (EXPLODE COMMAND)))))

      ((= 'IA   COMMAND) (INSERT_AFTER (ARG_LIST (READ))))

      ((= 'IB   COMMAND) (INSERT_BEFORE (ARG_LIST (READ))))

      ((MEMBER (CAR (EXPLODE COMMAND)) '(R I U D))
        (REPEAT (CAR (EXPLODE COMMAND))
                (NUMBER_OF_REPETITIONS)))

      ((= 'CH   COMMAND)
        (CHANGE_N (ARG_LIST (READ))
                  (ARG_LIST (READ)) (READ)))

      ((= 'CH!  COMMAND) (CHANGE_N (LIST (CAR *EDIT!*))
                                   (ARG_LIST (READ)) (READ)))

      ((= 'CHATOM COMMAND)
        (CHANGE_ATOM (CAR (ARG_LIST (READ)))
                     (CAR (ARG_LIST (READ)))
                     (CAR (ARG_LIST (READ))) (READ)))

      ((= 'CHATOM! COMMAND) (CHANGE_ATOM (CAR *EDIT!*)
                                         (CAR (ARG_LIST (READ)))
                                         (CAR (ARG_LIST (READ)))
                                         (READ)))

      ((= 'CHSTR COMMAND) (CHANGE_STR (CAR (ARG_LIST (READ)))
                                      (CAR (ARG_LIST (READ)))
                                      (READ)))
```

```
            ((= 'F     COMMAND) (FIND (ARG_LIST (READ))))

            ((= 'S     COMMAND) (BACK_TO_START))

            ((= 'TOP   COMMAND) (TOP))

            (T (ERROR_MESSAGE COMMAND 'ILLEGAL_COMMAND))))

(DEFUN ARG_LIST (ARG)
       (PROG (L)
             (SETQ L NIL)
            LOOP
             (COND ((= ARG *LIST_DELIMETER*)
                    (COND ((NULL L) (RETURN (LIST NIL)))
                          ( T (RETURN L)))))
             (SETQ L (APPEND L (LIST ARG)))
             (SETQ ARG (READ))
             (GO LOOP)))

(DEFUN REPEAT (COMMAND N)
       (PROG (COUNT DOCMD)
             (COND ((OR (NOT (LEGAL_NUMBER_ARG N))
                        (= N 0)) (RETURN NIL)))
             (SETQ COUNT 1)
             (SETQ DOCMD (EVAL (LIST COMMAND)))
             (COND ((NULL DOCMD) (RETURN (ERROR_MESSAGE 0 'COMPLETED))))
            LOOP
             (COND ((AND (= N '*) (NULL DOCMD)) (RETURN T))
                   ( T   (COND ((= N COUNT) (RETURN T))
                               (T (COND ((NULL DOCMD)
                         (RETURN (ERROR_MESSAGE (SUB1 COUNT) 'COMPLETED)))))))))
             (SETQ COUNT (ADD1 COUNT))
             (SETQ DOCMD (EVAL (LIST COMMAND)))
             (GO LOOP)))

(DEFUN NUMBER_OF_REPETITIONS ()
      (MAKE_NUMBER (CDR (EXPLODE COMMAND))) )

(DEFUN MKNUM (N) (RNUMB N) (MKATOM))

(DEFUN MKDIG (N)
       (COND ((= N 1)  1 )
             ((= N 2)  2 )
             ((= N 3)  3 )
             ((= N 4)  4 )
             ((= N 5)  5 )
             ((= N 6)  6 )
             ((= N 7)  7 )
             ((= N 8)  8 )
             ((= N 9)  9 )
             ((= N 0)  0 )))

(DEFUN MAKE_NUMBER (STR)
       (PROG (COUNT SUM)
             (COND ((NULL STR) (RETURN 1)))
             (COND ((NOT (DIGP (CAR STR))) (RETURN (CAR STR))))
             (SETQ COUNT (DIFFERENCE (LENGTH STR) 1))
```

```
            (SETQ SUM (TIMES (MKNUM (CAR STR))
                             (EXPT 10 COUNT)))
            LOOP
            (COND ((= COUNT 0) (RETURN SUM)))
            (SETQ STR (CDR STR))
            (COND ((NOT (DIGP (CAR STR))) (RETURN (CAR STR))))
            (SETQ COUNT (SUB1 COUNT))
            (SETQ SUM (PLUS SUM
                            (TIMES (MKNUM (CAR STR))
                                   (EXPT 10 COUNT))))
            (GO LOOP)))

(DEFUN DEFCMD  (NAME)
       (PROG (PROGRAM CMD DESCRIPTION)
             (SETQ *USER_COMMAND* (CONS NAME *USER_COMMAND*))
             (SETQ DESCRIPTION (USER_DEF))
             (SETQ PROGRAM (LIST 'DEFUN NAME '()))
             LOOP
             (COND ((NULL DESCRIPTION)
                    (RETURN (EVAL PROGRAM))))
             (COND ((= (CAR DESCRIPTION) 'IA)
                    (SETQ CMD (APPENDM
                                '(INSERT_AFTER|)
                                (ARGUMENTS))))
                   ((= (CAR DESCRIPTION) 'IB)
                    (SETQ CMD (APPENDM
                                '(INSERT_BEFORE|)
                                (ARGUMENTS))))
                   ((= (CAR DESCRIPTION) 'CH)
                    (SETQ CMD (APPENDM
                                '(CHANGE_N)
                                (ARGUMENTS)
                                (ARGUMENTS)
                                (MOVE_OVER)
                                (LIST (CAR DESCRIPTION))
                                (MOVE_OVER))))
                  ((= (CAR DESCRIPTION) 'CH|)
                    (SETQ CMD (APPENDM
                               '(CHANGE_N)
                               (LIST '(CAR *EDIT|*))
                               (ARGUMENTS)
                               (MOVE_OVER)
                               (LIST (CAR DESCRIPTION))
                               (MOVE_OVER))))
                  ((= (CAR DESCRIPTION) 'CHATOM|)
                    (SETQ CMD (APPENDM
                               '(CHANGE_ATOM)
                               (LIST '(CAR *EDIT|*))
                               (LIST (LIST 'CAR (CAR (ARGUMENTS))))
                               (LIST (LIST 'CAR (CAR (ARGUMENTS))))
                               (MOVE_OVER)
                               (LIST (CAR DESCRIPTION))
                               (MOVE_OVER))))
                   ((= (CAR DESCRIPTION) 'CHATOM)
                    (SETQ CMD (APPENDM
                               '(CHANGE_ATOM)
                               (LIST (LIST 'CAR (CAR (ARGUMENTS))))
```

```
                                (LIST (LIST 'CAR (CAR (ARGUMENTS))))
                                (LIST (LIST 'CAR (CAR (ARGUMENTS))))
                                (MOVE_OVER)
                                (LIST (CAR DESCRIPTION))
                                (MOVE_OVER))))
                    ((= (CAR DESCRIPTION) 'CHSTR)
                      (SETQ CMD (APPENDM
                                 '(CHANGE_STR)
                                 (LIST (LIST 'CAR (CAR (ARGUMENTS))))
                                 (LIST (LIST 'CAR (CAR (ARGUMENTS))))
                                 (MOVE_OVER)
                                 (LIST (CAR DESCRIPTION)) (MOVE_OVER))))
                   ((= (CAR DESCRIPTION) 'F)
                     (SETQ CMD (APPENDM
                                '(FIND)
                                (ARGUMENTS))))
                  ((MEMBER (CAR DESCRIPTION) '(HELP EDIT DEFCMD RPT ))
                    (RETURN (ERROR_MESSAGE (CAR DESCRIPTION)
                              'NOT_ALLOWED_IN_DEFCMD)))
                  ( T (SETQ CMD (APPEND '(PROCESS)
                                (LIST (LIST 'QUOTE (CAR DESCRIPTION)))))))
                (SETQ PROGRAM (APPEND PROGRAM (LIST CMD)))
                (MOVE_OVER)
                (GO LOOP) ))

(DEFUN USER_DEF () (PRINT '(ENTER NAME DEFN)) (PROG (ARG LIS)
       (SETQ ARG (READ))
       (SETQ LIS (CONS ARG NIL))
       LOOP
       (SETQ ARG (READ))
       (COND ((= ARG *CMD_DELIMETER*) (RETURN LIS)))
       (SETQ LIS (APPEND LIS (LIST ARG)))
       (GO LOOP)))

(DEFUN ARGUMENTS () (LIST (LIST 'COPY (LIST 'QUOTE (GET_ARGS)))))

(DEFUN GET_ARGS ()
       (PROG (ARG LIS)
             (MOVE_OVER)
             (SETQ ARG (CAR DESCRIPTION))
             (MOVE_OVER)
             (SETQ LIS (CONS ARG NIL))
             LOOP
             (SETQ ARG (CAR DESCRIPTION))
             (COND ((= ARG *LIST_DELIMETER*) (RETURN LIS)))
             (MOVE_OVER)
             (SETQ LIS (APPEND LIS (LIST ARG)))
             (GO LOOP)))

(DEFUN MOVE_OVER () (SETQ DESCRIPTION (CDR DESCRIPTION)))

(DEFUN RPT_USER (COMMAND N)
       (PROG (COUNT DOCMD)
             (COND ((= N '*)
                    (RETURN (ERROR_MESSAGE N 'NOT_ALLOWED_WITH_RPT)))
                   ((NOT (NUMBERP N))
                    (RETURN (ERROR_MESSAGE N 'ILLEGAL_ARGUMENT)))
```

```
                ((= N 0) (RETURN 0 (ERROR_MESSAGE 0 'COMPLETED))))
            (SETQ COUNT 1)
            (SETQ DOCMD (EVAL (LIST COMMAND)))
            (COND ((NULL DOCMD) (RETURN (ERROR_MESSAGE 0 'COMPLETED))))
           LOOP
            (COND ((AND (= N '*) (NULL DOCMD)) (RETURN T))
                  ( T   (COND ((= N COUNT) (RETURN T))
                              (T (COND ((NULL DOCMD)
                       (RETURN (ERROR_MESSAGE (SUB1 COUNT) 'COMPLETED)))))))))
            (SETQ COUNT (ADD1 COUNT))
            (SETQ DOCMD (EVAL (LIST COMMAND)))
            (GO LOOP)))

(DEFUN REMOVE (BUFFER_NAME N)
       (PROG ()
             (COND ((NOT (LEGAL_NUMBER_ARG N))
                    (RETURN NIL))
                   ((= N '*) (SETQ N (LENGTH *EDIT|*)))
                   ((= N 0) (RETURN (ERROR_MESSAGE N 'ILLEGAL_ARGUMENT)))
                   ((NOT (MEMBER BUFFER_NAME *BUFFERS*))
                     (SETQ *BUFFERS* (CONS BUFFER_NAME *BUFFERS*))))
              (RETURN (PUTPROP BUFFER_NAME
                               (COPY (APPEND (GET BUFFER_NAME 'CONTENTS)
                                             (GET_LINES_FOR_BUFFER
                                               BUFFER_NAME N)))
                               'CONTENTS)))))

(DEFUN GET_LINES_FOR_BUFFER (BUFFER N)
       (PROG (COUNT L)
             (SETQ COUNT 1)
             (SETQ L (LIST (CAR *EDIT|*)))
            LOOP
             (SETQ COUNT (ADD1 COUNT))
             (COND ((OR (NULL (DELETE|)) (LT N COUNT))
                    (ERROR_MESSAGE (SUB1 COUNT) 'CHANGES)
                    (RETURN L)))
             (SETQ L (APPEND L (LIST (CAR *EDIT|*))))
             (GO LOOP)))

(DEFUN REPLACE (BUFFER_NAME N START)
      (PROG (BUFFER)
            (SETQ BUFFER (GET BUFFER_NAME 'CONTENTS))
            (COND ((NULL BUFFER)
                   (RETURN (ERROR_MESSAGE NIL 'NULL_BUFFER)))
                  ((NOT (LEGAL_NUMBER_ARG N)) (RETURN NIL))
                  ((OR (NOT NUMBERP START) (LE START 0))
                     (RETURN (ERROR_MESSAGE START 'ILLEGAL ARGUMENT)))
                  ((= N '*) (SETQ N (ADD1 (MINUS (LENGTH
                                                   (GET BUFFER_NAME
                                                    'CONTENTS))
                                                 START))))
                  ((= N 0) (RETURN (ERROR_MESSAGE 0 'ILLEGAL_ARGUMENT)))
             ((GT START (LENGTH (GET BUFFER_NAME 'CONTENTS)))
               (RETURN (ERROR_MESSAGE START 'BEYOND_END_OF_BUFFER_CONTENTS))))
       (INSERT_AFTER|
                   (COPY (FIRST_N N (NMARK_START START
                                                 (GET BUFFER_NAME
```

```
                                                    'CONTENTS)))))))

(DEFUN MARK_START (N L)
       (COND ((= N 1) L)
             ((NULL L) L)
             (T  (MARK_START (SUB1 N) (CDR L)))))

(DEFUN DISPLAY_BUFFER_NAMES () (PRINT *BUFFERS*))

(DEFUN DISPLAY_BUFFER_CONTENTS (BUFFER_NAME)
      (PRINT (GET BUFFER_NAME 'CONTENTS)))

(DEFUN DELETE_N (N)
       (PROG (COUNT)
             (COND ((NOT (LEGAL_NUMBER_ARG N)) (RETURN NIL))
                   ((= N '*) (SETQ N (LENGTH *EDIT|*)))
                   ((= N 0) (RETURN (ERROR_MESSAGE 0 'ZERO_ARGUMENT))))
             (SETQ COUNT 1)
             LOOP
             (COND ((LT N COUNT) (RETURN T))
                   ((NOT (DELETE|))
                      (RETURN (ERROR_MESSAGE COUNT 'DELETIONS))))
             (SETQ COUNT (ADD1 COUNT))
             (GO LOOP)))

(DEFUN DELETE| () (COND ((L) (RPLACD *EDIT|* (CDDR *EDIT|*))
                             (COND ((AT_END_OF_EDIT_LIST)
                                    (ERROR_MESSAGE NIL 'END_OF_LEVEL))
                                   ( T (R))))
                        ((U) (RPLACA *EDIT|* (CDAR *EDIT|*))
                             (COND ((AT_BOTTOM_LEVEL)
                                    (ERROR_MESSAGE NIL 'AT_BOTTOM_LEVEL))
                                   ( T (D))))
                        (T (SETQ *EDIT|* (CDR *EDIT|*))
                           (SETQ *EXPRESSION* *EDIT|*)
                           (SETQ *LEVEL* *EDIT|*) )))


(DEFUN GROUP (N)
       (COND ((NOT (LEGAL_NUMBER_ARG N)) NIL)
             (T (COND ((= N '*) (SETQ N (LENGTH *EDIT|*))))
                (COND ((AT_BEGINNING)
                       (ERROR_MESSAGE NIL 'FIRST_SEXPR_OF_LIST))
                      ((= N 0) (ERROR_MESSAGE NIL 'ZERO_ARGUMENT))
                      ((LT (LENGTH *EDIT|*) N)
                           (ERROR_MESSAGE N 'GT_LENGTH_OF_LIST))
                      ( T (RPLACA *EDIT|* (FIRST_N N *EDIT|*))
                          (RPLACD *EDIT|*
                           (ALL_BUT_FIRST N *EDIT|*)) )))))

(DEFUN AT_BEGINNING () (= *EDIT|* *EXPRESSION*))

(DEFUN LEGAL_NUMBER_ARG (N)
       (COND ((OR (AND (NUMBERP N) (LT N 0))
                  (AND (NOT (NUMBERP N)) (NOT (= N '*))))
              (ERROR_MESSAGE N 'ILLEGAL_ARGUMENT))
             (T T)))
```

```
(DEFUN UNGROUP ()
       (COND ((ATOM (CAR *EDIT|*))
              (ERROR_MESSAGE NIL 'CANNOT_UNGROUP_ATOM))
             ((AT_BEGINNING) (CHANGE| (CAR *EDIT|*)))
             (T (NCONC (CAR *EDIT|*) (CDR *EDIT|*))
                (COND ((L) (RPLACD *EDIT|* (CADR *EDIT|*)) (R))
                 ((U)    (RPLACA *EDIT|* (CAAR *EDIT|*)) (D)) ) ) ) )

(DEFUN UN* ()
       (COND ((ATOM (CAR *EDIT|*))
              (RETURN NIL))
             ((AT_BEGINNING) (CHANGE| (CAR *EDIT|*)))
             (T (NCONC (CAR *EDIT|*) (CDR *EDIT|*))
                (COND ((L) (RPLACD *EDIT|* (CADR *EDIT|*)) (R))
                 ((U)    (RPLACA *EDIT|* (CAAR *EDIT|*)) (D)) ) ) ) )

(DEFUN INSERT_AFTER| (INSERTION)
       (COND ((NULL INSERTION) (ERROR_MESSAGE NIL 'NO_INSERTIONS))
             ( T (NCONC INSERTION (CDR *EDIT|*))
                 (RPLACD *EDIT|* INSERTION)) )

(DEFUN INSERT_BEFORE| (INSERTION)
      (COND ((NULL INSERTION) (ERROR_MESSAGE NIL 'NO_INSERTIONS))
            (T (COND ((L) (INSERT_AFTER| INSERTION) (R))
                     ((U) (NCONC INSERTION (CAR *EDIT|*))
                          (RPLACA *EDIT|* INSERTION)
                          (D) )
               (T (NCONC INSERTION *EDIT|*)
                  (SETQ *EDIT|* INSERTION)
                  (SETQ *EXPRESSION* *EDIT|*)
                  (SETQ *LEVEL*      *EDIT|*))))))

(DEFUN R ()
        (COND ((AT_END_OF_EDIT_LIST) NIL)
              (T (SETQ *EDIT|* (R_MOVE)) )) )

(DEFUN R_MOVE () (CDR *EDIT|*))

(DEFUN AT_END_OF_EDIT_LIST () (NULL (CDR *EDIT|*)))

(DEFUN L ()
        (COND ((AT_START_OF_EDIT_LEVEL) NIL)
              ( T (SETQ *EDIT|* (L_MOVE)) ) ))

(DEFUN AT_START_OF_EDIT_LEVEL () (= *EDIT|* *LEVEL*))

(DEFUN L_MOVE ()
       (ALL_BUT_FIRST
          (SUB1 (DIFFERENCE
                   (LENGTH *LEVEL*)
                   (LENGTH *EDIT|*) ))
          *LEVEL* ))

(DEFUN ALL_BUT_FIRST (N L)
       (COND ((NULL L) NIL)
             ((= N 0) L)
```

```
              ( T (ALL_BUT_FIRST (SUB1 N) (CDR L)) )))

(DEFUN U ()
         (COND ((AT_TOP_LEVEL)  NIL)
               (T (RESET|) (RESET_LEVEL)
                  (SETQ *EDIT_POSITIONS* (CDDR *EDIT_POSITIONS*)) T ) ))

(DEFUN AT_TOP_LEVEL () (= (CAR *EDIT_POSITIONS*) '=TOP=))

(DEFUN RESET| () (SETQ *EDIT|* (CAR *EDIT_POSITIONS*)))

(DEFUN RESET_LEVEL () (SETQ *LEVEL* (CADR *EDIT_POSITIONS*)))

(DEFUN D ()
         (COND ((AT_BOTTOM_LEVEL)  NIL)
               (T (SETQ *EDIT_POSITIONS* (CONS *LEVEL* *EDIT_POSITIONS*))
                  (SETQ *EDIT_POSITIONS* (CONS *EDIT|* *EDIT_POSITIONS*))
                  (NEW_LEVEL) (NEW|) T) ))

(DEFUN AT_BOTTOM_LEVEL () (ATOM (CAR *EDIT|*)))

(DEFUN NEW_LEVEL () (SETQ *LEVEL* ( CAR *EDIT|* )))

(DEFUN NEW| () (SETQ *EDIT|* (CAR *EDIT|*)))

(DEFUN TOP ()
       (COND ((U) (TOP))))

(DEFUN BACK_TO_START ()
       (PROG ()
             (TOP)
            LOOP
             (COND ((NULL (L)) (RETURN T)))
             (GO LOOP) ) )

(DEFUN CHANGE_N (KEY CHANGES N)
        (PROG (COUNT COPY_CHANGES)
              (COND ((NULL CHANGES)
                     (RETURN (ERROR_MESSAGE NIL 'NULL_CHANGES))))
              (COND ((NULL KEY)
                     (RETURN (ERROR_MESSAGE NIL 'NULL_KEY))))
              (COND ((NOT (LEGAL_NUMBER_ARG N)) (RETURN NIL))
                    ((= N '*) (SETQ N (LENGTH *EDIT|*)))
                    ((= N 0) (RETURN (ERROR_MESSAGE 0 'CHANGES))))
              (SETQ COUNT 1)
             LOOP
              (SETQ COPY_CHANGES (COPY CHANGES))
              (COND ((OR (LT N COUNT) (NOT (LOCATE_NEXT KEY)))
                     (RETURN (ERROR_MESSAGE (SUB1 COUNT) 'CHANGES))))
              (REPEAT ' (R) (SUB1 (LENGTH KEY)))
              (NCONC COPY_CHANGES (CDR *EDIT|*))
              (REPEAT ' (L) (SUB1 (LENGTH KEY)))
              (COND ((L) (RPLACD *EDIT|* COPY_CHANGES) (R))
                    ((U) (RPLACA *EDIT|* COPY_CHANGES) (D))
                    ( T  (SETQ *EDIT|* COPY_CHANGES)
                         (SETQ *EXPRESSION* *EDIT|*)
                         (SETQ *LEVEL*      *EDIT|*)))
```

```
            (SETQ COUNT (ADD1 COUNT)) (GO LOOP)))

(DEFUN CHANGE! (CHANGES)
       (COND ((NULL CHANGES) (ERROR_MESSAGE NIL 'NO_CHANGES))
             ( T (NCONC CHANGES (CDR *EDIT!*))
                 (COND ((L) (RPLACD *EDIT!* CHANGES) (R))
                       ((U)   (RPLACA *EDIT!* CHANGES) (D))
                       ( T    (SETQ *EDIT!* CHANGES)
                              (SETQ *EXPRESSION* *EDIT!*)
                              (SETQ *LEVEL*      *EDIT!*))) ) ) )

(DEFUN CHANGE_ATOM (ATM OLD NEW N)
       (PROG (NEW_ATOM CHK)
             (COND ((NOT (AND (ATOM ATM) (ATOM OLD) (ATOM NEW)))
                    (RETURN (ERROR_MESSAGE NIL 'ARGUMENT_NOT_ATOM)))
                   ((OR (NULL ATM) (NULL OLD))
                    (RETURN (ERROR_MESSAGE NIL 'NULL_ATOM/SUBATOM))))
             (SETQ NEW_ATOM (CHANGED_ATOM ATM OLD NEW))
             (SETQ CHK (MAKE_NUMBER (EXPLODE NEW_ATOM)))
             (COND ((NUMBERP CHK) (SETQ NEW_ATOM CHK))
                   ((= NEW_ATOM ATM)
                    (RETURN (ERROR_MESSAGE OLD 'NOT_FOUND))))
             (RETURN (CHANGE_N (LIST ATM)
                               (LIST NEW_ATOM)
                               N))))

(DEFUN NEXT () (PROG ()
                     LOOP
                     (COND ((NULL (U)) (RETURN NIL))
                           ((R) (RETURN T))
                           ( T  (GO LOOP)) )))

(DEFUN FIRST_N (N L)
       (COND ((NULL L) NIL)
             ((LE N 0) NIL)
             ( T (CONS (CAR L) (FIRST_N (SUB1 N) (CDR L)))) ))


(DEFUN CHANGED_ATOM  (ATM OLD NEW)
       (IMPLODE (CHLIST (EXPLODE1 OLD)
                        (EXPLODE1 NEW)
                        (EXPLODE1 ATM)
                        (EXPLODE ATM))))

(DEFUN EXPLODE1 (ATM)
       (PROG (L NEW_ATOM)
             (COND ((NUMBERP ATM) (RETURN (EXPL_NUM ATM)))
                   ((NULL ATM) (RETURN NIL)))
             (SETQ L (EXPLODE ATM))
             (SETQ NEW_ATOM (LIST (CAR L)))
            LOOP
             (SETQ L (CDR L))
             (COND ((NULL L) (RETURN NEW_ATOM)))
             (SETQ NEW_ATOM (APPEND NEW_ATOM
                                    (LIST (COND ((DIGP (CAR L))
                                                 (MKNUM (CAR L)))
                                                ( T (CAR L))))))))
```

```
            (GO LOOP)))

(DEFUN EXPL_NUM (N)
     (COND ((= N 0) NIL)
           ( T (APPEND (EXPL_NUM (QUOTIENT N 10))
                       (LIST (REMAINDER N 10))))))

(DEFUN CHLIST (OLD NEW L1 L2)
       (COND ((NULL L1) NIL)
             ((= OLD (FIRST_N (LENGTH OLD) L1))
               (APPEND (EXPLODEDIG NEW) (REST OLD L2)))
             ( T (CONS (CAR L2) (CHLIST OLD NEW (CDR L1) (CDR L2)))) ))

(DEFUN EXPLODEDIG (L)
       (COND ((NULL L) NIL)
             (T (APPEND (LIST (COND ((NUMBERP (CAR L)) (MKDIG (CAR L)))
                                    (T (CAR L))))
                        (EXPLODEDIG (CDR L))))))

(DEFUN COUNTATOMS (S)
       (COND ((NULL S) 0)
             ((ATOM S) 1)
             ((PLUS (COUNTATOMS (CAR S)) (COUNTATOMS (CDR S))))))

(DEFUN EXPT (N X)
       (PROG (COUNT PROD)
             (COND ((= X 0) (RETURN 1)))
             (SETQ COUNT 1)
             (SETQ PROD N)
             LOOP
             (COND ((= COUNT X) (RETURN PROD)))
             (SETQ PROD (TIMES PROD N))
             (SETQ COUNT (ADD1 COUNT))
             (GO LOOP) ))

(DEFUN CHANGE_STR (OLD NEW N)
       (PROG (COUNT NEW_ATM)
             (COND ((NOT (LEGAL_NUMBER_ARG N)) (RETURN NIL))
                   ((= N '*) (SETQ N (COUNTATOMS *EDIT|*)))
                   ((= N 0) (RETURN (ERROR_MESSAGE 0 'CHANGES))))
             (SETQ COUNT 1)
             LOOP
             (COND ((OR (END_OF_FILE) (LT N COUNT))
                    (RETURN (SUB1 COUNT))))
             (COND ((ATOM (CAR *EDIT|*))
             (SETQ NEW_ATM (CHANGED_ATOM (CAR *EDIT|*) OLD NEW))
             (COND ((NOT (= (CAR *EDIT|*) NEW_ATM))
                    (CHANGE| (LIST NEW_ATM))
                   (SETQ COUNT (ADD1 COUNT)))))
              (COND ((OR (D) (R) (NEXT)) (GO LOOP)))))

(DEFUN REST (SUBSTRING STRING)
       (COND ((NULL SUBSTRING) STRING)
             (T (REST (CDR SUBSTRING) (CDR STRING))) ))

(DEFUN END_OF_FILE () (AND (AT_END_OF_EDIT_LIST) (AT_TOP_LEVEL)))
```

FILE: EDJANA LISP A1 KANSAS STATE UNIVERSITY VM/SP CMS

```
(DEFUN FIND (KEY)
       (PROG (FOUND SAVE LENGTH_KEY)
             (COND ((NULL KEY) (RETURN (ERROR_MESSAGE NIL 'NULL_KEY))))
             (SETQ SAVE (LIST *EDIT|* *LEVEL* *EDIT_POSITIONS*))
             (SETQ LENGTH_KEY (LENGTH KEY))
             (SETQ FOUND NIL)
            LOOP
             (COND (FOUND (RETURN T)))
             (COND ((OR (D) (R) (NEXT))
                    (SETQ FOUND (= (FIRST_N LENGTH_KEY *EDIT|*) KEY))
                    (GO LOOP))
                   ( T (RESET_SAVE)
                       (RETURN (ERROR_MESSAGE KEY 'NOT_FOUND))) )))

(DEFUN RESET_SAVE () (SETQ *EDIT|* (CAR SAVE))
                     (SETQ *LEVEL* (CADR SAVE))
                     (SETQ *EDIT_POSITIONS* (CADDR SAVE)) )

(DEFUN LOCATE_NEXT (KEY)
        (PROG (FOUND)
              (COND ((NULL KEY) (RETURN NIL)))
              (SETQ FOUND (= (FIRST_N (LENGTH KEY) *EDIT|*) KEY))
             LOOP
              (COND (FOUND (RETURN T)))
              (COND ((OR (D) (R) (NEXT))
                     (SETQ FOUND (= (FIRST_N (LENGTH KEY) *EDIT|*) KEY))
                     (GO LOOP))
                    ( T (RETURN NIL)))))
```

APPENDIX II

References

REFERENCES

{1} Siklossy, Laurent, Let's Talk LISP, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1976.

{2} Allen, John, Anatomy of LISP, McGraw Hill, New York, 1978.

{3} Waters, Richard C.,'GPRINT: A LISP Pretty Printer Providing Extensive User Format-Control Mechanisms,' AI Memo NO. 611, Massachusetts Institute of Technology Artificial Intelligence Laboratory, October, 1981.

{4} Barstow, David R., 'Overview of a Display-Oriented Editor for INTERLISP,' Proceeding of the Seventh International Joint Conference on Artificial Intelligence, Volume II, pp. 927-929.

{5} Stallman, Richard M.,'EMACS The Extensible, Customizable Self-Documenting Display Editor,' AI Memo 519a, Massachusetts Institute of Technology Artificial Intelligence Laboratory, 26 March 1981.

{6} Charniak, Eugene; Riesbeck, Christopher K.; McDermott, Drew V., Artificial Intelligence Programming, Lawrence Eilbaum Associates, Publishers, Hillsdale, New Jersey, 1980.

THE DESIGN, IMPLEMENTATION, AND USE OF LEDIT: A REAL-TIME EDITOR FOR LISP

by

JANA TAYLOR GOODMAN

B. S., Kansas State University, 1973
Ms Ed., University of Southern California, 1975

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1982

THE DESIGN, IMPLEMENTATION, AND USE OF LEDIT: A REAL-TIME EDITOR FOR LISP

'Programming languages can be divided
into two categories.
In one category, there is LISP;
in the second category,
all the other programming languages!'

Jean Sammett {1}

LEDIT is a collection of LISP functions which enable the programmer to edit LISP programs and data bases without ever exiting from the LISP interpreter.

A comprehensive range of edit commands are included for implementation. LEDIT provides the user with the same capability that a conventional text editor provides. In addition, commands are available which are designed specifically to aid in editing LISP structures.

Since it is simply a relatively small set of LISP functions, LEDIT can be easily implemented on any LISP system; and the editor can be used with any line-oriented terminal.

Even novice and casual LISP programmers will find LEDIT easy to use. When LEDIT is used by the LISP programmer, editing will be more consistent with the LISP edit structures and more convenient for the user.

Part 1 of this report is the presentation of the motivation for the design and implementation of LEDIT. Part 2 is the overview of the LEDIT design. Part 3 is the detailed analysis of the LEDIT implementation. Part 4 is the User's Guide to LEDIT.