THE NPS LISP 1.5 VERS 1 PROGRAMMING SYSTEM

by

John C. Pilley

United States
Naval Postgraduate School

THESIS

THE NPS LISP 1.5 VERS 1

PROGRAMMING SYSTEM

by

John C. Pilley

April 1970

THE NPS LISP 1.5 VERS 1

PROGRAMMING SYSTEM

by

John C. Pilley
Major, United States Marine Corps
B.A., Vanderbilt University, 1961

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
April 1970

Thesis P54 17
c 1

ABSTRACT

The design and implementation of the NPS LISP 1.5 VERS 1 programming system is described. NPS LISP 1.5 VERS 1 is a complete programming system built around the original implementation of LISP 1.5 on the IBM 360/67 computer at the Naval Postgraduate School. The new version includes a Supervisor and an Editor, as well as the original LISP Interpreter. The VERS 1 system is written in PL/I for time-sharing operation on the IBM 360/67 computer.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

The NPS LISP 1.5 VERS 1 System is an extension of NPS LISP which was developed at the Naval Postgraduate School by Lieutenant Donald G. Gentry, USN, and Major John C. Pilley, USMC, and documented in Lieutenant Gentry's thesis [Ref. 1], dated December 1969. The NPS LISP system was an "on-line interactive version of LISP for the IBM 360/67 computer",[1] and was written in PL/I to run under the CP/CMS time-sharing system. (The CP/CMS time-sharing system is described fully in Reference 5.)

NPS LISP 1.5 VERS 1 system consists of three modules. These modules are:

1. the Supervisor,
2. the Editor,
3. the Interpreter.

The Interpreter module of the extended system is the original NPS LISP system without its supervisor. The Supervisor module is an expansion of the NPS LISP supervisor and includes all the latter's functions, while the Editor module is an entirely new entity, unique to the VERS 1 system.

This thesis describes the expansion of NPS LISP into the system named NPS LISP 1.5 VERS 1. The development of those features of the expanded system which are new or revised is described in detail. It is assumed that the reader has access to Reference 1, since the description of the Interpreter will not be repeated here. A knowledge of PL/I is desirable so that the reader may interpret the statements in the VERS 1 listing.

[1]D. G. Gentry, An Implementation of LIPS 1.5 for the IBM 360/67 Computer (Monterey, 1969), p. 9.

As has been noted, such a supervisor incorporates no housekeeping functions. If it were to be expanded to include such housekeeping chores as an editor and an I/O package, appropriate LISP functions would have to be written and incorporated as permanently defined functions in the LISP system. In general this poses no problems. For example, Q-32 LISP [Ref. 3] uses LISP functions for I/O. The elaborate editor of BBN LISP [Ref. 4] also uses LISP functions for its routines. However, these systems were coded in their particular machine language, and hence supervisor functions written in LISP were natural.

NPS LISP was coded in PL/I. This was done because PL/I, being a higher level language, "...provides major advantages over a system written in machine code: (1) alterations to the system may be easily made; (2) the system may be implemented quite easily on another computer having a PL/I compiler; and (3) the system is self-documenting to some extent."[3] Another important reason for writing the supervisor and editor in PL/I is that most of the <u>LISP</u> memory is still available to the user to store list structures and the numbers his programs might generate. Thus a minimum amount of LISP free storage is taken up by the "nucleus" of permanent LISP functions.

After having decided to write the expanded supervisor (and editor) in PL/I, the next task was to determine just what capabilities to incorporate in the expanded system.

Much of the motivation for features finally incorporated into the new system came from practical experience gained on the terminal with the CP/CMS time-sharing system. This motivation, as well as other influences is discussed in the sections that follow.

---

[3]Gentry, p. 50.

II. THE SUPERVISOR

A. A GENERAL DISCUSSION OF LISP SUPERVISORS

The supervisor of any LISP system must perform certain basic functions. First, it must read in a functional expression (S-expression) and its arguments (also an S-expression). Second, it must apply the function to the arguments to obtain a value, and, third, it must print that value. The basic LISP supervisor is a routine which loops continuously, while LISP is active, performing these functions. Disregarding any system "housekeeping" needs, this basic supervisor meets the minimum requirements of a LISP system.

The NPS supervisor was essentially identical to the supervisor described above. It looped continuously until detecting an 'END LISP' input, which caused it to terminate. The NPS LISP supervisor was a routine written in PL/I, though, in general, a LISP supervisor is a LISP function called EVALQUOTE. An example of a LISP 1.5 coding of EVALQUOTE follows which is taken from Weissman [Ref. 2]. The variables S1 and S2 are the two S-expressions that the supervisor reads.

```
    (EVALQUOTE (LAMBDA ( )  (PROG S1 S2 ARGS)

A   (TEREAD)
    (SETQ ARGS NIL)
    (SETQ S1 (READ))
    (SETA S2 (READ))
B   (COND ((NULL S2)  (GO C)))
    (SETQ ARGS (CONS (LIST (QUOTE QUOTE) ( CAR S2)) ARGS))
    (SETQ S2 (CDR S2))
    (GO B)
C   (PRINT EVAL (CONS S1 (REVERSE ARGS))))
    (GO A) )))[2]
```

---

[2] Clark Weissman, LISP 1.5 Primer, (Belmont, 1968), p. 131.

1. IBM 7090 LISP Supervisor

In 7090 LISP the EVALQUOTE function is called the interpreter, while the supervisor (or monitor) is a separate function called the Overlord.[4] The 7090 Overlord handles the I/O, maintains the system's files and handles dumps. Since 7090 LISP receives punched cards as input, the overlord is essentially a batch processing supervisor. Its main function seems to be that of handling of the five tape drives the 7090 system uses. These tapes are listed here by name and function.

SYSTAP	Contains the System
SYSTMP	Receives the Core Image
SYSPIT	Punched Card Input
SYSPOT	Printed Output
SYSPPT	Punched Card Output

After the LISP system is called by the LISP loader (two special cards), the Overlord takes over and initializes the system. From then on the Overlord is in control, checking each card in the input stream to see if it is an Overlord direction card. These Overlord cards direct the Overlord to perform some specific function or else tell it that a LISP deck (called a packet of doublets) follows. The packet of doublets is a deck of pairs of S-expressions for the interpreter. The Overlord continues as described above until it reads a 'FIN' Overlord card, at which time it halts.

2. Q-32 LISP Supervisor

The Q-32 LISP system is a compiler-oriented rather than an

[4]J. McCarthy, and others, LISP 1.5 Programmer's Manual (Cambridge, 1962), p. 80.

Interpreter-oriented language.[5] When run under time-sharing, the user sees two modes of operation: the Executive mode and the Object Program mode.[6] The Executive mode is the mode used to communicate with the time-sharing executive and is analogous to CMS when running LISP under CP/CMS. The Object Program mode is analogous to the NPS LISP 1.5 VERS 1 Supervisor; therefore, it is this mode we wish to examine here.

The Q-32 LISP supervisor (accessible in the Object Program mode) is a function *SUPV "which calls for two S-expressions to be read from the teletype, terminates the input buffer, then calls for (PRINT (*EVALQT X Y (QUOTE*FUNC)) and loops back to call for two more S-expressions."[7] The prefix * indicates that the function is a basic system function not normally useful to the user. The arguments X and Y of *EVALQT are the two S-expressions read from the teletype. Thus, despite differences such as being compiler-oriented, the Q-32 LISP supervisor is basically similar to the general supervisor described by Weissman [Ref. 2]. The function *EVALQT, of course, is EVALQUOTE though *EVALQT has three arguments (one being required by the Compiler).

Rather extensive I/O is possible by using special LISP I/O functions. The Supervisor handles these functions in the same way it handles the basic LISP functions. The function SAVE (n), for example, dumps LISP core in the same manner as the DUMP$ command of the NPS LISP Supervisor (See Section II.C.3). Thus every Q-32 LISP function is a supervisor command.

---

[5] S. L. Kameny, LISP 1.5 Reference Manual for Q-32 (Santa Monica, 1965) p. 7.

[6] Ibid, p. 8.

[7] Ibid, p. 64.

For further discussion of the Q-32 LISP supervisor see Section 4.3 of Reference 8.

3. <u>BBN LISP Supervisor</u>

BBN 940 LISP is a version of LISP implemented on the SDS 940 computer by Bolt Beranek and Newman, Inc., [Ref. 4]. It is primarily of interest here because of its Editor, which will be discussed in Section III.

The BBN LISP Supervisor is called EVALQUOTE and performs the standard functions of reading S-expression doublets and executing them. Input-Output functions, file manipulation, and editor commands are all in the form of S-expressions doublets and are all handled by EVALQUOTE. Thus, as with Q-32 LISP, all LISP commands are supervisor commands.

4. <u>General Functions and Capabilities of the NPS LISP 1.5 VERS 1 Supervisor</u>

The NPS LISP 1.5 VERS 1 Supervisor is a PL/I program which performs the following tasks:

1. Calls the LISP initializer to start the system,
2. Handles the SPECIAL (automatic) OPTIONS requests,
3. Sets and supervises the two user-oriented modes of system operation (i.e., Editor mode or Interpreter mode), switching the system from one mode to the other as the user requests,
4. Handles user requested dumps,
5. Loads specialized system files on request,
6. Calls the Garbage Collector at the request of the user,
7. Handles the close-out routine when a LISP run is terminated by the user.

These functions, and the Supervisor language, are discussed in the sections that follow. The VERS 1 Supervisor is similar to those described above, except that housekeeping chores are peculiar to the Supervisor. This eliminates the necessity of the VERS 1 system supporting a number of specialized LISP functions.

In conclusion, I wish to point out that though the 'Overlord' of 7090 LISP is essentially a batch processing monitor, it nevertheless does for 7090 LISP what the VERS 1 Supervisor does for NPS LISP. It is therefore fair to say that the NPS LISP 1.5 VERS 1 Supervisor more closely resembles the 7090 LISP system than any of the other systems studies. Thus both the VERS 1 Supervisor and Interpreter use the 7090 system as a pattern.

The terminology 'S-expression doublet' has been borrowed from the description of 7090 LISP [Ref. 7] and will be used in this thesis. The description of the VERS 1 Supervisor begins with the following discussion of the automatic features.

B. AUTOMATIC FEATURES OF THE VERS 1 SUPERVISOR

Experience gained from work on the terminal, using CP/CMS, led to the incorporation into the Supervisor of two optional automatic features. What follows is a description of these features and a discussion of the reasons for their inclusion in the VERS 1 system.

It often happens that a user must make so many corrections, deletions and additions to his input string, that it becomes almost unintelligible. This fact motivated the first automatic option, the printback feature. This feature prints back the string the LISP Interpreter has actually received and numbers the first line of each doublet.

Another feature which seemed desirable, due to sometimes too frefrequent CP "crashes", was an automatic LISP storage saving routine. The term "crash" is a popular but descriptive term which refers to an abrupt halt in service due to a malfunction in the time-sharing system. The CP time-sharing system is not immune to such crashes. Without an automatic dump routine a user would have to reinitialize the LISP system and start over again after each crash. This could be very frustrating and time consuming. The AUTOSAVE feature was therefore incorporated and is described below. Either feature can be abled or disabled at any time after initialization of the system.

Immediately after the system has initialized, the question "SPECIAL OPTIONS?" will be typed on the terminal. This is the only time this explicit question will be asked. If neither option is desired, the user should respond by typing "N" or "NONE". The default condition for a "NONE" response is autosave and printback disabled.

If the user wishes to change his current options, he can do so by using the OPT$ command. This is a Supervisor command and is discussed here since it is used exclusively with the automatic options. The Supervisor returns control to the Interpreter after honoring the OPT$ request.

1. <u>OPT$ command</u>

   OPT$ list

   This command causes the supervisor to change the current special options listed in "list." Options in "list" (if more than one) must be separated by at least one blank. If the "list" is missing, the command is ignored. (The commands 'OPTION$' or 'OPTIONS$' are also valid.

2. <u>PRINTBACK option</u>

   P (NP)

   The 'P' command causes each S-expression doublet received by the Interpreter to be printed back with its sequence number. 'NP', typed in the "list" of the OPT$ command disables the printback feature.

3. AUTOSAVE option

AUTOSAVE [=n] (NAS)

> The autosave feature, which is called by this command, automatically dumps LISP memory into file AUTOLISP every 5 doublets unless a different line parm value has been specified. AUTOLISP is erased during the closeout routine called by the user typing 'END LISP'. The bounds on the line parm, n, are: $1 \leq n \leq 20$. 'NAS' typed in the "list of the OPT$ command erases the file AUTOLISP and disables the autosave feature. The autosave feature also creates an exact copy of file LISPTEXT (the working file of the Editor) each time it dumps the LISP memory. This copy is a file called LISPTEMP. (For further discussion of these files see the following subsection).

If a CP crash occurs, the version of AUTOLISP and LISPTEMP created by the last autosave dump, will remain, unaffected, on the user's disk. The user will see a message, reminding him of the existence of these files when he reinitializes the LISP system. The user should load these files at the first communication from the Interpreter or they will be automatically erased. The Interpreter tells the user it is ready by displaying 'CALL EVALQUOTE, ARGS'.

The PL/I file handling routine IHEFILE[8] is used to create the files LISPTEMP, AUTOLISP, and DUMPLISP. For this reason it is briefly mentioned in the discussion that follows.

a. Files AUTOLISP and DUMPLISP

The PL/I file handling routine IHEFILE is described fully in Reference 5. For our present requirements it is enough to know that this routine creates a file on the user's disk by storing an 80 character buffer in the file each time the routine is called in the PL/I program.

[8] CP/CMS User's Guide (Revised 6.69), Naval Postgraduate School, Computer Facility (Monterey, 1969), p. 385.

Since this routine is used to create files AUTOLISP and DUMPLISP, an immediate problem presented itself. How do we efficiently convert binary fixed data (the LISP memory is in this form) into a character string? The method chosen for the NPS LISP 1.5 VERS 1 system was to use the comparatively fast UNSPEC[9] function of PL/I to put the internal representation of each word of LISP memory into a 4 character substring of the buffer. This stores 20 words of FSTOR each time the IHEFILE routine is called. Since LISP core is initially set up as a sequentially linked list (i.e., the cells are an array called FSTOR, linked by storing in each word the array subscript of the next word), there is usually a sizeable percentage of LISP memory which consist of adjacent cells containing numbers in sequence (see Fig. 1). This is especially true if the Garbage Collector has not been used and/or the user's program is not extremely large. Therefore, to increase the speed of the dump and decrease the size of the storage area required for file AUTOLISP, the following structure for file AUTOLISP (as well as file DUMPLISP) was adopted.

The structure of the file AUTOLISP can be thought of in two ways. One way is to think of it as a set of 80 character strings, each of which was stored from the buffer CARD-BUFFER by one call to the IHEFILE routine. (See Fig. 2, Physical Structure.) Thus, to access an item, one needs the character string number and the location of the item in the character string. The other way of viewing AUTOLISP is as a sequential storage area where an item is accessed by knowing its sequential address. (See Fig. 2, Sequential Structure.) Both representations of AUTOLISP are used by the dump routine.

[9]IBM System/360 Reference Manual, International Business Machines Corporation (1968), p. 237.

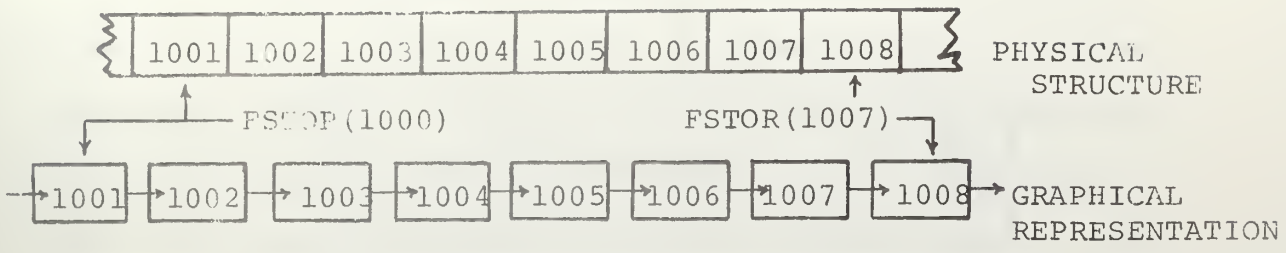


FIGURE 1

A SEGMENT OF FSTOR SHOWING INITIAL LISP MEMORY STRUCTURE

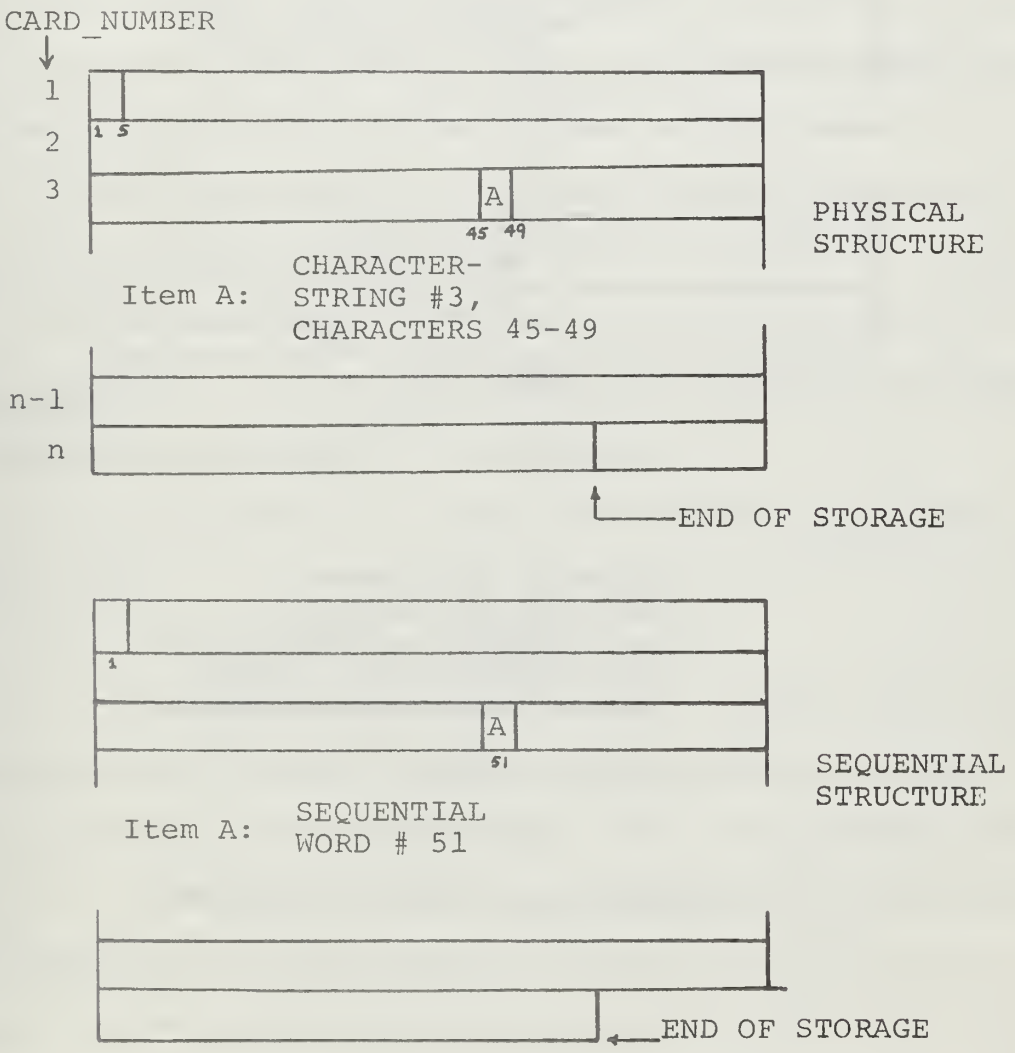
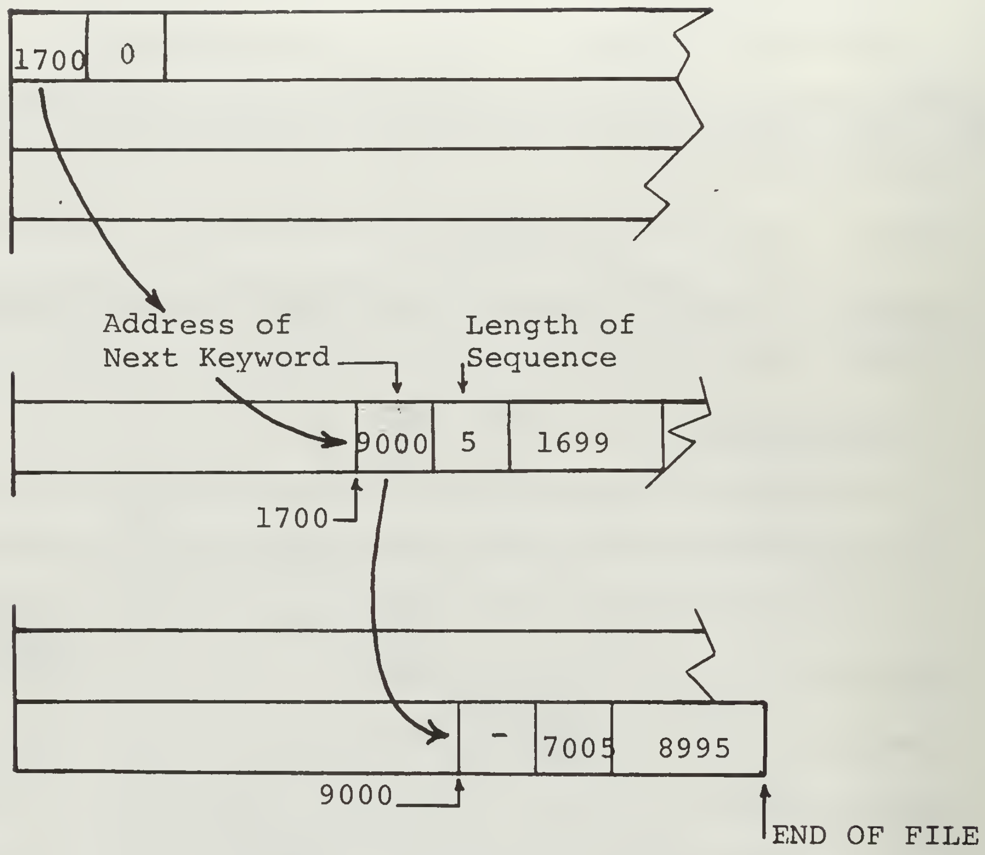


FIGURE 2

TWO WAYS OF REPRESENTING THE STRUCTURE OF THE FILE AUTOLISP



- Keywords are shaded.
- The word adjacent to the keyword contains the address of the first word of the FSTOR sequence.

FIGURE 3

KEYWORD SYSTEM OF FILE AUTOLISP

The first word of AUTOLISP (bytes 1-4 of the first 80 character string) is called a Keyword. The first halfword of a keyword contains the sequential address of the next keyword. (The "sequential address", as opposed to a "physical address", refers to a sequential location of a word.) The second two bytes of a keyword contain a number C. When the dump routine encounters more than two adjacent words in FSTOR which contain sequential numbers, (as for example, FSTOR(1000) FSTOR(1001) FSTOR(1002) in Fig. 1), the next available sequential address becomes the address of a new keyword and is stored in the first two bytes of the last keyword. The first word of the FSTOR sequence (FSTOR(1000) in the example above) is then stored following this new keyword. The number of words in the sequence, C, is stored in the last two bytes of the new keyword. (For clarification see Fig. 3). In this way storage time and area can be minimized. Of course if most of LISP memory is active or if it has been reshuffled due to numerous garbage collections, the advantages of this system will be minimized.

A discussion of the other files affected by the dump routine follows.

b. Files LISPTEXT and LISPTEMP

File LISPTEXT is the file used by the Editor. Each doublet input by the user is stored in LISPTEXT in sequence with its sequence number in the first byte of the first line. The input is stored, starting with line 2 of LISPTEXT. The first line of LISPTEXT is used for two purposes, storing certain values which are required by the load routine and storing a code-byte (C-byte) used by the Supervisor. The C-byte is the first byte of the first line. The possible values of the C-byte and resulting Supervisor actions are given in Table I.

The values stored on line 1 of file LISPTEXT are stored there by the DUSAVE PL/I routine called each time a dump occurs. (DUSAVE is called by both the autosave feature and by the DUMP$ command.) These are the values of certain PL/I variables which are listed in Table II.

The C-byte is set by various Supervisor routines. The PL/I listing of these routines can be found in APPENDIX C. (The major PL/I modules which make up the VERS 1 system are LISPA, LISPB, LISPC, LISPD, LISPE, LISPF, LISPG, and LISPP. Each module is listed in APPENDIX B with its major function.) The C-byte is set to the value '1' by:

1. The close-out procedure if its value is '2' when that procedure is entered. The close-out procedure is located in LISPA following the lable TERM.

2. The procedure which handles the 'OPT$ NAS' input. This procedure disables the autosave feature and is located in LISPA following the label SCN1.

3. The file LISPTEXT handler, when the first S-expression doublet is entered and the autosave feature has not been enabled. This routine follows entry point TEXTWRK in LISPA.

The C-byte is set to '2' by two routines. The first is the routine which processes the 'AUTOSAVE' command. This procedure is located in LISPA following the label SCN1. The second is the TEXTWRK routine of (c) above. This routine will set the C-byte to '2' when the first doublet is input if the autosave option has been chosen. The value '3' is set by the routine which processes the DUMP$ command. This routine is located in LISPA following the label CMD.

Since the file LISPTEXT contains the current C-byte value, some version of it resides on the user's disk at all times. The message telling of the existence of file LISPTEXT actually refers to the existence of file LISPTEMP, an exact copy of LISPTEXT at the time of the dump.

TABLE I

FILE LISPTEXT FLAG VALUES AND
CORRESPONDING SUPERVISOR ACTION

C-byte VALUE	MEANING	SUPERVISOR ACTION
1	Previous exit from LISP system was normal and no dump was requested	reinitializes LISPTEXT
2	Previous exit from LISP system was ab- normal with AUTOSAVE option enabled.	Prints message: "---FILES LISPTEXT AND AUTOLISP EXIST--"
3	A dump was requested before previous exit from LISP system	Prints message: "--FILES LISPTEXT AND DUMPLIST EXIST--"
any other value	File LISPTEXT does not exist in a usable form on the user's disk.	Initializes file LISPTEXT

TABLE II

PL/I VARIABLES STORED IN FILE LISPTEXT

PL/I VARIABLE	FUNCTION	POSITION (f,e)*
LCOUNT	S-expression doublet counter	(2,4)
CARDNOI	Next available line in LISPTEXT	(6,4)
ESTART	Pointer used by the Editor	(10,4)
EFSTOR	Last cell in Free Storage	(14,4)
FREE	First cell in Free Storage	(20,4)
BNUM	Next available cell for numbers	(30,4)

* Location in character string: f=position of first character of substring, l=length of substring

Thus, though the user may 'LOAD$ LISPTEXT', the file that is actually loaded is LISPTEMP. The only time confusion may occur is if the user wishes to edit the LISPTEXT he has dumped, using the CMS editor (See Ref. 5). He must then be sure to 'EDIT LISPTEMP DATA'. LISPTEMP does not exist on the user's disk if no dump has occurred.

The foregoing discussion of the VERS 1 special files concludes the subsection concerned with the automatic features of the Supervisor. These files will be mentioned frequently in the sections that follow. The next subsection describes the various Supervisor commands.

C. SUPERVISOR COMMANDS

As pointed out in Section II.A, all Supervisor commands end in the symbol '$'. The user issues all Supervisor commands in response to the standard "ready" of the interpreter. (See Section II.B.3.) The syntax analyzer, entry point NREAD of LISPC, scans the first S-expression, recognizes that the first S-expression is an atom ending in a '$' and alerts the Supervisor. The Supervisor then takes control, scans the command and takes the appropriate action. The Supervisor commands and their effects are listed below. The PL/I routines that implement these commands are located in LISPA unless otherwise specified. (See APPENDIX C for a listing of LISPA.)

1. <u>OPT$</u> See Section II.B.1.

2. <u>EDIT$</u>

   EDIT$

   In response to the EDIT$ command, the supervisor gives control to the LISP Editor. The Editor will respond by typing 'EDIT LISP'. It is then ready for an input from the user. (The command 'E$' is also valid.)

3. <u>DUMP$</u>

   DUMP$

The DUMP$ command tells the Supervisor to dump LISP memory into file DUMPLISP which is erased automatically by the next DUMP$ command or after it is loaded by the LOAD$ command.

The file LISPTEXT is also saved by the DUMP$ command as well as the values of the PL/I variables ESTART, FREE, BNUM, LCOUNT, EFSTOR, and CARDNO[1] which are needed to restore memory (See Section II.B.3.(b)). The DUMP$ routine uses the PL/I procedure DUSAVE, an entry point in LISPF, which is also used by the AUTOSAVE option. (See APPENDIX C.) Since files DUMPLISP and AUTOLISP have the same structure, the discussion of file AUTOLISP in Section II.B.3. applies also to file DUMPLISP.

Upon initialization of the LISP system, if DUMP$ was used on the previous run (and hence files DUMPLISP and LISPTEMP are on the user's disk), the user will be warned of this fact by a message (See Table I).

4. <u>LOAD$</u>

LOAD$ filename [filename]

The LOAD$ command loads "filename" into LISP memory except when "filename" is LISPTEXT. In this case LISPTEMP is loaded and renamed LISPTEXT. If more than one file is loaded, the filenames must be separated by at least one blank. During the load, a message will be displayed, telling which file is currently being loaded.

5. <u>QUIT$</u>

QUIT$

The QUIT$ command causes the Supervisor to take control from the interpreter, delete what has been typed of the S-expression doublet being input, reinitialize the Interpreter and return control to it. The Editor is also told to remove the same input from LISPTEXT. (Q$ is a recognized abbreviation).

6. <u>COUNT$</u>

COUNT$

The COUNT$ command causes a count to be made of the number of free cells in LISP memory. The result is a message from the Supervisor, CELLS IN FREE STORAGE: n
CELLS AVAILABLE FOR NUMBERS: m
where n and m are decimal numbers.

7. <u>COLLECT$ and COLLECTN$</u>

COLLECT$

This is the call to the Garbage Collector, which will collect in both main storage and number storage. For discussion of the Garbage Collector and LISP memory structure see Section IV.B.

COLLECTN$

In response to the COLLECTN$ command the Garbage Collector will collect only in number storage.

This concludes the discussion of the Supervisor. The following section describes the NPS LISP 1.5 VERS 1 Editor.

III. THE EDITOR

A. EDITING LISP PROGRAMS IN AN INTERACTIVE ENVIRONMENT

The editor of the CP/CMS system is one of that system's most used features. This is natural since one of the prime advantages of an interactive system is the ability to correct mistakes as soon as they are detected or to make program changes in an environment which allows almost immediate analysis of their effects. These factors motivated the design of NPS LISP 1.5 VERS 1 Editor. Other systems studied were 7090 LISP, BBN LISP, Q-32 LISP and the implementation of the Georgia Tech LISP interpreter at the University of Washington. It was evident that little had been documented in the field of LISP editors. Of those studied, the BBN LISP editor was the only system with an editor. It is summarized below.

A fundamental difference between the NPS LISP 1.5 VERS 1 Editor and the BBN system must be emphasized. The editors of all LISP systems have S-expressions as the data on which they operate. The difference between the VERS 1 Editor and the others is found in the structure of the S-expressions. The other systems operate on S-expressions which are stored as lists (their tree structure). The editor commands of these systems are LISP functions which traverse S-expression trees and "edit" using LISP primitive functions to alter the structure of these trees. There are certain obvious advantages to this type of editor. Some of these are

- all editor commands are LISP functions,
- the structure of data for the editor is the same as for the the supervisor and interpreter,
- an operation on an S-expression directly alters the structure of that S-expression.

The VERS 1 Editor operates on the material stored in file LISPTEXT. Here S-expression doublets are stored as character strings, the form in which they were input. The VERS 1 Editor commands are not LISP functions; they make up a separate language. Finally, an operation by the VERS 1 Editor alters the character string structure of a doublet in LISPTEXT, not its list structure in LISP memory. Hence the doublet must be "re-computed" by the Interpreter after editing, for the editing to have effect.

Before going into detail on the VERS 1 Editor, it would be instructive to look at the BBN LISP editor. This editor is rather sophisticated and the next subsection is concerned exclusively with some of its features.

1. BBN LISP Editor

The BBN LISP Editor is an extensive and distinct subsystem of the BBN LISP system. It "allows rapid, convenient modification of list structures. Most often it is used to edit function definitions, often while the function itself is running."[10] It has many functions, allowing the user a choice of ways to accomplish a particular editing requirement. The editor commands are short, usually consisting of a single character.

The BBN LISP Editor operates directly on list structure. The particular list that is to be edited is indicated in the call to the editor. For example,

```
EDITF (APPEND)
```

[10] D. G. Bobrow, and others, The BBN 940 LISP System (1967), p. 40

would bring the editor on line and specify the function definition of APPEND as the list to be edited. There are four different commands for calling the editor, depending on the type of list structure to be edited. The basic function, for editing S-expressions, is the function EDITE. The other three are

1. EDITV, for editing values
2. EDITF, for editing function definitions, and
3. EDITP, for editing property lists.

These functions confine their attention to only one list at a time, usually the sublist of some major structure. Changes may be made only to this particular sublist. "Commands thus fall naturally into four classes: moving around the last structure; making changes in the current list; printing parts of the list being edited; and entering and leaving the editor."[11]

a. List Traversal Commands

List traversal is accomplished by typing a number n to indicate the nth level from the top of the main list structure which is to be examined. A '-n' examines the nth sublist starting from the end of the main structure and counting back n sublists. The command ↑ re-establishes the top level of the main structure as being current. To traverse from a current sublist one can use the command (NTH n) which uses the current sublist as a starting place and causes the nth sublist from that point to become current. The command 'P' is used to print the current list. If the current list is the main structure, a LISP representation of the entire structure will be printed. The command (F _e_) searches

[11] Ibid, p. 41.

for an occurrence of e in the current list, where e is any S-expression. The number 0 causes the previously current list to become current. These commands allow great flexibility in traversing list structures. Even greater flexibility is obtained by a function MARK which allows the user to mark the current state. He may then return later by typing ← which returns him to the last mark without erasing the mark or by ←← which does the same thing, but erases the mark.

COPY and RESTORE are the last two list traversal commands which will be specifically mentioned. Since every change made is made directly to the internal list structure, the COPY command can be issued before changes are made. This copies the "current state of the edit", so that the user may later restore the system to its previous state by issuing a RESTORE command.

b. Modification Commands

The capabilities of inserting, replacing and appending list structures is provided by the three commands, (-n $e_1,\ldots,e_m$), (n $e_1,\ldots,e_m$) and (N $e_1,\ldots,e_m$) where $e_1,\ldots,e_m$ are S-expressions. The 'n's are integers which indicate the nth sublist from the current one. Thus (-1 ABC) would <u>insert</u> ABC before the current sublist, and (2 ABC) would <u>replace</u> the 2nd sublist from the current sublist with ABC. The 'N' is used to append S-expressions. To delete, use the replace command with no replacement S-expression specified. The command (R e_1 e_2) replaces all occurrences of list e_1 with e_2 in the current list and all sublists.

Direct alteration of the list structure itself is also possible. This is done with commands such as LO (take Left paren Out), BO (take Both parenthesis Out), (RO n m) and (RI n m). The last command

(RI n m) will be illustrated. Consider the list (A B(C D E) F G). If we want to bring the D and E up to the level of A B F G and leave (C) as a sublist, we could use

(RI 3 1)

which means move the Right paren at the end of sublist 3 In to sublist 3 and place it after sublist 1 of sublist 3. The results is the list (A B (C) D E F G).

c. Printing Commands

The command P which prints the current list, showing only one level of nesting, was presented in subsection a. To print a selected sublist without changing the current state of the edit, a (P n) command could be used. This would print the nth sublist from the current one. (P n m) prints the nth sublist to m levels of nesting. Finally, (E c) prints the sublist c without changing the state of the edit.

The BBN LISP editor is a highly sophisticated subsystem. A good knowledge of LISP 1.5 is required to use it. It has the dangers which result from being able to directly alter its memory, but the advantages of speed and versatility far outweigh any disadvantages.

A general discussion of the NPS VERS 1 Editor follows.

2. NPS LISP 1.5 VERS 1 Editor

The NPS LISP Editor performs the following functions:

1. Prints S-expression doublets,
2. Changes the character string representation of S-expressions,
3. Recomputes specified doublets,
4. Lists atoms, functions, or special forms from OBLIST,
5. Deletes atoms and functions from OBLIST.

All but the last two of these functions depend upon LISPTEXT. The last two functions are performed by traversing OBLIST.

Every doublet entered into the VERS 1 system is stored in LISPTEXT. The Editor accesses a particular doublet by traversing a list to locate that doublet's sequence number. The pointer to the head of this list is the PL/I variable ESTART (See Table II). Each doublet entered by the user is assigned a unique number. This number is stored in the CAR of a word in FSTOR whose CDR is another word in FSTOR. The CAR of this second word is the line number in LISPTEXT of the first line of the doublet. Thus, to find doublet number, n, the Editor traverses the ESTART list, starting at ESTART. If CAR(ESTART) is not equal to n then it checks CADDR(ESTART), and so on until n is located. The required line number in LISPTEXT is the CADR of this cell. To operate on this doublet, the Editor would then read this line into CARD-BUFFER using the IHEFILE routine. An obvious disadvantage of this system is that only part of a doublet can be worked on at any instant. Thus the CHANGE function requires that all changes be made, one line at a time, not allowing part of a field to be on one line while the remainder is on the next. When all changes are made, the changed doublet must be recomputed before the changes have any effect upon the LISP list structure. This is one advantage, since ill-conceived changes will not take immediate effect, as in the BBN Editor.

The commands which make up the language of the NPS VERS 1 Editor are listed and discussed in the section that follows.

B. EDITOR COMMANDS

The Editor initially tells the user it is on-line by typing the message 'EDIT LISP'. The Editor will remain on-line until it receives the 'RET' command or the 'END LISP' command. Any user error in issuing an edit command will result in one of two responses by the Editor. Either the Editor will ignore the faulty edit command and wait for another input, or it will type an error message and wait for another input. See APPENDIX A for the error messages. The Editor commands are discussed below.

1. <u>RET (RETURN)</u>

   RET (RETURN)

   The RET command returns control to the Interpreter through the Supervisor. When this is done the user will see the standard Interpreter challenge, 'CALL EVALQUOTE, ARGS:' typed on his terminal.

2. <u>PRINT</u>

   P (PRINT) n[,m]

   The PRINT command tells the Editor to find S-expression doublet number "n" and display it. A pointer, used for the CHANGE function, is also positioned at this doublet. If "m" is specified then the m-1 following doublets will also be displayed (for a total of "m" doublets). If this is the case, the CHANGE pointer will be pointing to the last doublet displayed.

3. <u>CHANGE</u>

   C (CHANGE) /string1/string2/

   The CHANGE function replaces the first occurrence of string1 by string2, in the last doublet printed by the PRINT command. It does this by searching each character string, belonging to the doublet, for an occurrence of string1. The first character string of the doublet is indicated by the pointer set by the PRINT command (See <u>PRINT</u> above). When the string "string1" is located, it is replaced by the string "string2". If string1" and "string2" are of such length as to cause the new string to exceed 80 characters (the length of the character strings in LISPTEXT), an error message will be displayed. (See APPENDIX A) Note that "string1" must reside in only one character string of LISPTEXT.

It will not be recognized by the 'C' function if part resides in one string while the remainder resides in the next character string of LISPTEXT. Because of an implementation restriction, the maximum length of "string1" is 35 characters.

4. <u>RECOMPUTE</u>

R   (RECOMPUTE)   n

The command, RECOMPUTE, causes an S-expression doublet "n" to be fed to the Interpreter as an input. It obtains the same results as a direct input to the Interpreter.

5. <u>LIST</u>

LIST type

The LIST command tells the Editor to search OBLIST and print the names of all items stored there with the characteristic "type". The three allowable "type" designators are

A	(ATOM),
F	(FUNCTION),
S	(SPECIAL FORM).

The Editor will display the items in the order in which they appear in OBLIST, along with any indicators they might have on their property lists (except PNAME and user defined indicators). Since these lists can be long, and displaying them can be time consuming, a feature has been incorporated which allows the user to terminate the listing process. After 20 names are displayed, the Editor will ask the question 'CONTINUE?' to which the user must answer either 'Y' or 'N'.

6. <u>DELETE</u>

D   (DELETE)   name

The DELETE command results in a search of OBLIST for the name "name". It is deleted from the system when it is found. If it is not found a message is printed to that effect.

The discussion of the Interpreter, which follows, is primarily concerned with two items. One item is the Garbage Collector and the other is a slight modification of LISP memory necessitated by the implementation of the Garbage Collector.

IV. THE INTERPRETER

A. GENERAL DISCUSSION

In Section IV of Reference 1, Lieutenant Gentry made several recommendations for improvement of the NPS LISP interpreter. The primary concern here is the description of the implementation of one of those recommendations, the Garbage Collector. The Garbage Collector required a slight modification of the NPS LISP memory structure. This modification is explained in the next subsection in conjunction with the Garbage Collector.

The recommendations for future expansion, mentioned above, fell under four major headings. These were, Garbage Collection, Arithmetic Functions, Functions with Functional Arguments and the Prog Feature. The final three of these recommendations have not been investigated and therefore remain unimplemented.

Besides the changes which will be discussed in the next subsection, an error recovery routine and its error message has been added. This routine is called when the user uses too many right parenthesis to close the first S-expression of a doublet. (See APPENDIX A)

B. NPS LISP MEMORY ORGANIZATION AND THE GARBAGE COLLECTOR

The previous NPS LISP memory structure, discussed in Section III.A.6. of Reference 1, was adequate for that system's needs. As long as there was no garbage collector, handling LISP storage as a Linked List, (See Fig. 4(a)) the previous system worked very well. The old scheme was to store numbers from the "top" of memory, working down, while allocating "up" for List structure. This sequential allocation scheme is not compatible with an efficient garbage collector (one based on a linked allocation scheme). Consequently, a few minor alterations were made to change the structure to that shown in Figure 4.(b). Up to the time the user activates the Garbage Collector for the first time, the new storage scheme is apparently identical to the old. After garbage collection, however, the new structure becomes evident, with list structure storage linked and number storage sequential.

The NPS LISP 1.5 VERS 1 Garbage Collector is relatively simple and was built as part of the Supervisor (See Section II.C.7.). It is called explicitly by the user who may determine the necessity of garbage collection by using the Supervisor command COUNT$ (Section I.C.6.). Gentry recommended a warning system in conjunction with this (user called) garbage collector. This warning system would periodically count the calls in free storage and issue a warning if free storage became "dangerously close to becoming empty."[12] This warning system was not incorporated and the user must use the explicit COUNT$ command to determine how much free storage remains. The Garbage Collector is called by the command COLLECT$, at which time the two active lists are traversed and marked. These two lists are (1) the OBLIST and (2) the ESTART list. The OBLIST is fully described in Reference 1 (Section III.A.5.). It is sufficient to note that, through the OBLIST, the Garbage Collector can get to every active atom and its property list.

The ESTART list is a list used by the Editor and is made up of pairs of words. The CAR of the first word is an S-expression doublet number, and the CAR of the second word is the corresponding character string number in file LISPTEXT. Each active cell is marked by placing a '1'

---

[12]Gentry, p. 52.

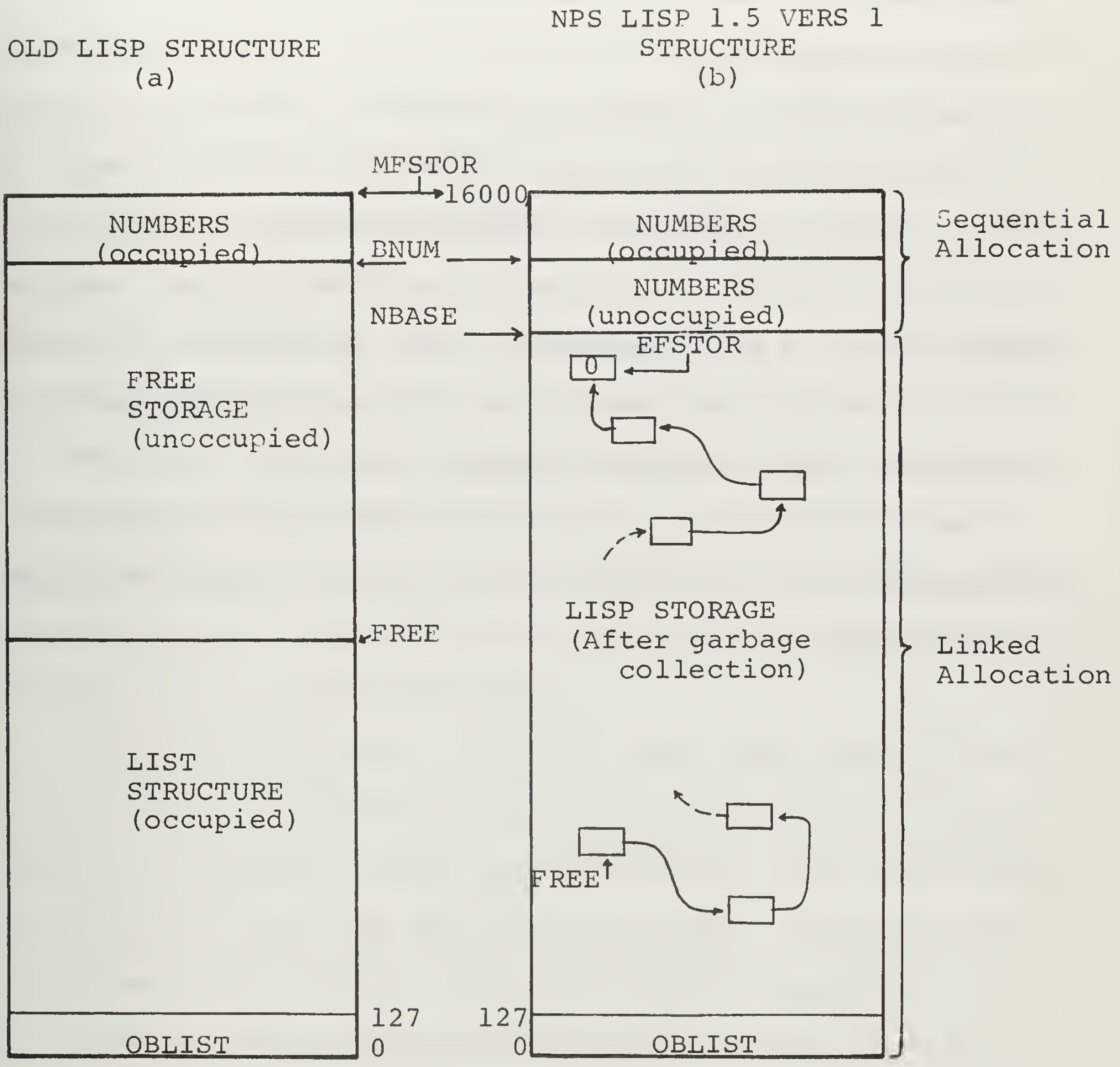


FIGURE 4

IMPROVED LISP MEMORY STRUCTURE
SHOWING OLD AND NEW STRUCTURE

in the 17th bit (which is otherwise '0'). The Garbage Collector then starts at the cell sequentially following the last cell in OBLIST (FSTOR(127)) and progresses sequentially through LISP storage until reaching NBASE (the base of the number storage area). Along the way, every cell is checked for a '1' in the 17th bit. If it is marked, it is then unmarked (the '1' is replaced with a '0'). If the cell is unmarked it is linked to end of free storage. The value of the PL/I variable FREE is unchanged, but the value of EFSTOR is changed to the address of the last cell linked up. The Garbage Collector then sets BNUM equal to MFSTOR (See Fig. 4) and resequences the number storage area. This operation of collecting in the number storage area can be called separately by use of the Supervisor command COLLECTN$ (See Section II.C.7.(a)).

The final section of this thesis is a brief look at future expansion possibilibies. I have only indicated areas of possible development, omitting details.

V. RECOMMENDATIONS FOR FUTURE EXPANSION OF THE NPS LISP 1.5 VERS 1 SYSTEM

The four main areas for expansion, listed by Lieutenant Gentry in Reference 1, were:

1. Garbage Collection,
2. Arithmetic Functions,
3. Functions with Functional Arguments,
4. The Prog Feature.

Of these four main areas, only the Garbage Collector has been implemented. The others were expansions of the Interpreter specifically, and were not attempted in the development of the VERS 1 system. These recommendations remain for future development, since the expansion of the arithmetic functions, the inclusion of the ability to handle the PROG feature and functions with functional arguments, would greatly enhance the capabilities of the present Interpreter.

Although the Supervisor, handles the special dump files, it does not have a true I/O capability. One recommendation would be that the Supervisor be expanded to handle user defined files, I/O to tape drives, and output to a print file for use with the printer. The Q-32 system has extensive I/O capability. This system could be studied and some of its features perhaps implemented as part of the VERS 1 Supervisor. Another helpful feature would be a short IBM 360 Assembler Language routine which would print the S-expression doublet number instead of the standard '_', which follows the 'CALL EVALQUOTE, ARGS:' message. This routine would be called each time the first line of a new doublet is input.

The present Editor provides the basic functions required of an editor. However, a study of the BBN editor is sure to reveal specific functions which would be profitable if implemented on the NPS LISP Editor. One example is the multiple change feature. If present, this function would locate every occurrence (in a doublet) of the string to be replaced, and replace it with the new string.

In conclusion, the first version of NPS LISP 1.5, (VERS 1), meets all the basic requirements of an interactive LISP system under the CP/CMS time-sharing system. Moreover, its structure permits easy expansion. Therefore, it is hoped, future versions of NPS LISP 1.5 will not be restricted by any design feature of the present NPS LISP 1.5 VERS 1.

APPENDIX A

NPS LISP 1.5 VERS 1 ERROR MESSAGES

SUPERVISOR:

S1: COMMAND NOT RECOGNIZED, TRY AGAIN

Special option request is not recognizable. Check Section II.B for proper format.

System Action: Ignores faulty command and waits for next try.

S2: SUPERVISOR COMMAND NOT RECOGNIZED

A regular Supervisor command (one ending in a '$') was not recognizable. Check Section II.C for proper format.

System Action: Ignores faulty command and returns control to the Interpreter.

SA1: LINE PARM OUT OF BOUNDS

AUTOSAVE =

The line parm, n, is out of bounds or not a number. The bounds on the line parm are $1 \leq n \leq 20$.

System Action: Ignores the faulty line parm and calls for a new line parm input by displaying 'AUTOSAVE=' and waiting for a reply.

SA2: AUTOSAVE DEFAULT VALUE ASSUMED

The second line parm input is out of bounds or not a number.

System Action: Assumes default value for line parm. Default value is 5.

UNNUMBERED: FILE NOT FOUND

This reply to a LOAD$ file name command indicates file "filename" is not on the user's disk. Check for incorrect spelling of "filename".

System Action: Returns control to the Interpreter.

INTERPRETER:

I1: PARENTHESES ERROR

There are too many right parentheses closing the first S-expression of the doublet being input.

System Action: Deletes the part of the doublet already input and sets up for the next doublet to be input.

EDITOR:

E1: EDIT CALL NOT VALID, TRY AGAIN

The call for the particular Editor function desired was not recognized. See Section III.B for proper Editor calls.

System Action: Editor ignores faulty call and waits for next try.

E2: NO S-EXPRESSION ENTERED

The call to 'PRINT n' or 'RECOMPUTE n' was made with not only doublet "n" not in the system, but no doublets entered. Perhaps a LOAD$ command was not issued or the "filename" was not the one desired.

System Action: Editor ignores call and waits for the next one.

E3: S-EXPRESSION NUMBER NOT FOUND

The call to 'PRINT n' or 'RECOMPUTE n' was made with doublet 'n' not in the system. Check for the correct "n".

System Action: Editor ignores call, waits for next one.

E3A: FORMAT ERROR: PRINTING S-EXP NUMBER "n"

The call to 'PRINT n,m' was incorrectly formatted. Check for the omission of a comma or of "m".

System Action: Prints doublet "n".

E4: ILLEGAL EDIT COMMAND

The Editor call was correct, but the rest of the call was unrecognizable. Check for incorrect 'CHANGE' call format.

System Action: Editor ignores call and waits for the next one.

E5A: 'TYPE' PARAMETER MISSING

On a 'LIST type' call the "type" parameter was left off. Try again.

System Action: Editor ignores call and waits the next one.

E5B: 'TYPE' PARAMETER NOT RECOGNIZED

On a 'LIST type' call the "type" parameter was not a legal one. Check Section II.B.5 for legal "type" parameters.

System Action: Editor ignores call and waits for next one.

E6A: FIELD NOT FOUND

On a 'CHANGE /string1/string2/' call the field "string1" was not found in the doublet pointed to by the "change" pointer. This pointer is set by the 'PRINT n' call. Check Section III.B.5 for details.

System Action: Editor waits for next call leaving the "change" pointer unmoved.

E6B: TRUNCATED

On a 'CHANGE /string1/string2/' call the field "string2" is longer than "string1" and the change has caused the file LISPTEXT character string, which now contains "string2" as a substring, to exceed 80 characters in length. The excess characters have been dropped on the right.

System Action: Editor makes the change truncating on the right and waits for next call.

E7: ATOM NOT FOUND

On a 'DELETE name" call the atom "name" was not found in the system.

System Action: The Editor waits for the next call.

UNNUMBERED: FIELD TOO LONG

This reply to a 'CHANGE /string1/string2/' call means that the field of "string1" exceeds 35 characters, the maximum field length for "string1".

System Action: Editor waits for next call.

APPENDIX B

MAJOR PL/I BLOCKS AND THEIR FUNCTIONS

MAJOR BLOCK	ENTRY POINTS AND LABELS	
LISPA		LISPA is the Supervisor.
	SCNT:	The Special Options routine.
	LOOP:	The basic supervisor toutine analogous to EVALQUOTE in other systems.
	TERM:	The exit point from the system.
	CMD:	The routines following this label handle the Supervisor commands.
	TEXTWRK:	A function called by several routines but used primarily by the Editor. It stores doublets in LISPTEXT and reads lines of LISPTEXT into CARD_BUFFER.
	TSAVE:	Copies LISPTEXT into LISPTEMP.
LISPB		LISPB contains LISP subroutine functions. The entry point names are the names of the LISP functions with an 'N' prefix.
LISPC		LISPC contains the syntax and lexical analysis functions..
	NREAD:	Reads in an S-expression, and sets up the list structure required by EVALQUOTE. (It manages OBLIST in the process).
	INCWRK:	Handles the storing of input lines in LISPTEXT, the incrementing of the autosave counter, and printback if that option has been chosen.
	SCAN:	The entry point for the routine that reads in a line and carries out the lexical analysis of that line.

	ENTER:	Enters an atom into OBLIST after HASH has returned the atom's OBLIST number.
	HASH:	Hash codes atoms to obtain their OBLIST number.
	LOOKUP:	Looks to see whether or not an atom is stored in OBLIST.
LISPD:		LISPD is the Interpreter, the Print and Trace and FSUBR functions.
LISPE:		LISPE is the initialization routine.
LISPF:		LISPF contains the reader, the Garbage Collector and the dump and load routines.
	READER:	The read routine. It displays whatever string it has been passed and receives the input into whatever buffer has been designated by the calling routine (usually the 72 character string BUFFER).
	DUSAVE:	The dump routine for LISP memory.
	LOADER:	The load routine for files AUTOLISP and DUMPLISP.
	COLLECT:	The Garbage Collector.
LISPG		LISPG is the NPS LISP 1.5 VERS 1 Editor.
	EA:	The routine which handles I/O with the terminal and analyzes the user input.
	E1:	The PRINT routine. It is also used by the RECOMPUTE routine.
	E2:	The LIST routine.
	E3:	The CHANGE routine.
	E4:	The DELETE routine.
	E5:	The RECOMPUTE routine.
LISPP:		LISPP contains all the primitive LISP functions. These correspond to the entry point names without the 'N' prefix.

APPENDIX C

LISTING OF THE NPS LISP 1.5 VERS 1 SYSTEM

```
LISPA: PROC OPTIONS(MAIN);

/* THIS IS THE LISP SUPERVISOR */

        DECLARE
        ACCUM CHAR(30) EXT,
        ASCTR BIN FIXED EXT,
        ASNMBR BIN FIXED EXT,
        ASTAG BIT(1) EXT,   /* '1'B: AUTOSAVE ENABLED */
        AVAR CHAR(2) VARYING,
        BNUM BIN FIXED EXT,
        BP BIN FIXED EXT,
        BUFFER CHAR(72) EXT,
        CARDNO1 BIN FIXED(31,0) EXT,
        CARDNO3 BIN FIXED(31,0) EXT,
        CH4 CHAR(4),
        COUNT BIN FIXED EXT,
        CTAG BIT(1) EXT,   /* '1'B: INPUT S1,LINE 1 */
        DTAG BIT(1) EXT,
        EFSTOR BIN FIXED EXT,
        1 FCB EXT,
             2 COMMAND CHAR(8),
             2 FILENAME CHAR(8),
             2 FILETYPE CHAR(8),
             2 CARD_NUMBER BIN FIXED(31,0),
             2 STATUS BIN FIXED(31,0),
             2 CARD_BUFFER CHAR(80),
        ENDTAG BIT(1) EXT,   /* '1'B: END LISP */
        ERSTAG BIT(1) EXT,   /* '1'B: FILES A_LISP OR D_LISP EXIST
        ERRTAG BIT(1) EXT,   /* '1'B: PAREN ERROR */
        ESTART BIN FIXED(31) EXT,
        FREE BIN FIXED EXT,
        FSEND BIN FIXED,
        FSTOR(0:16000) BIN FIXED(31) EXT,
        LCOUNT BIN FIXED EXT,
        MFSTOR BIN FIXED EXT,
        MODETAG BIT(1) EXT,
        NREAD ENTRY EXT,
        PTAG BIT(1) EXT,
        READER ENTRY(CHAR(25) VARYING,CHAR(72)) RETURNS(BIT(1)),
        RTAG BIT(1) EXT,
        S1 BIN FIXED,
        S2 BIN FIXED,
        S3 BIN FIXED,
        TCOUNT BIN FIXED EXT,
      VARA CHAR(25) VARYING;

      VARA='SPECIAL OPTIONS?'; ESTART=0; ERRTAG='0'B;
      PTAG,DTAG,ASTAG,ENDTAG,ERSTAG,RTAG='0'B; ASNMBR=5; ASCTR=0;
      DTAG1='0'B;
      FILENAME='LISPTEXT'; FILETYPE='DATA'; COMMAND='RDBUF';
      CARD_NUMBER=1; CALL IHEFILE(FCB);
      IF SUBSTR(CARD_BUFFER,1,1)='1' THEN
          DO; COMMAND='ERASE'; CALL IHEFILE(FCB);
          COMMAND='WRBUF'; CARD_BUFFER='1';
          CALL IHEFILE(FCB); GOTO ST;
          END;
      IF SUBSTR(CARD_BUFFER,1,1)='2' THEN
          DO; DISPLAY(' '); ERSTAG='1'B;
          DISPLAY('--FILES LISPTEXT AND AUTOLISP EXIST--');
          DISPLAY(' '); GOTO ST;
          END;
      IF SUBSTR(CARD_BUFFER,1,1)='3' THEN
          DO; DISPLAY(' '); ERSTAG='1'B;
          DISPLAY('--FILES LISPTEXT AND DUMPLISP EXIST--');
          DISPLAY(' ');
          END;
      ELSE DO; COMMAND='WRBUF'; CARD_BUFFER='1';
          CALL IHEFILE(FCB);
          END;
  ST: CARDNO1=2;
      CALL INITIAL;
      TCOUNT=2;
      IF ¬READER(VARA,BUFFER) THEN GOTO TERM; BP=0;
SCNT: CALL SCAN; IF TCOUNT=0 THEN GOTO CK;
SCN1: IF SUBSTR(ACCUM,1,COUNT)='N' | SUBSTR(ACCUM,1,COUNT)='NONE'THEN
          DO; DISPLAY(' '); DISPLAY(' '); GOTO LOOP; END;
      IF SUBSTR(ACCUM,1,COUNT)='NP' THEN DO; PTAG='0'B; GOTO CK; END;
      IF SUBSTR(ACCUM,1,1)='P' THEN DO; PTAG='1'B; GOTO CK; END;
      IF SUBSTR(ACCUM,1,COUNT)='AUTOSAVE' THEN
          DO; ASTAG='1'B; CALL SCAN;
          IF TCOUNT=0 THEN GOTO CK;
          FILENAME='LISPTEXT'; CARD_BUFFER='2';
          COMMAND='WRBUF'; CARD_NUMBER=1; CALL IHEFILE(FCB);
          IF SUBSTR(ACCUM,1,COUNT)¬='=' THEN GOTO SCN1;
          CALL SCAN; ASNMBR=SUBSTR(ACCUM,1,COUNT);
          IF ASNMBR>20 | ASNMBR<1 THEN
              DO; DISPLAY('**SA1: LINE PARM OUT OF BOUNDS**');
              DISPLAY('**AUTOSAVE=') REPLY(AVAR);
              ASNMBR=AVAR; IF ASNMBR>20 | ASNMBR<1 THEN
```

```
                  DO: ASNMBR=5;
                  DISPLAY('**SA2: AUTOSAVE DEFAULT VALUE ASSUMED**
                  GOTO CK; END;
                  END;
            GOTO CK;
            END;
      IF SUBSTR(ACCUM,1,COUNT)='NAS' THEN
            DO; ASTAG='0'B; ASNMBR=5;
            FILENAME='LISPTEXT';
            CARD_BUFFER='1'; COMMAND='WRBUF'; CALL IHEFILE(FCB);
            FILENAME='AUTOSAVE'; COMMAND='ERASE';
            CALL IHEFILE(FCB); GOTO CK;
            END;
      DISPLAY('**S1: COMMAND NOT RECOGNIZED, TRY AGAIN**'); BP=7
      GOTO SCNT;
  CK: IF TCOUNT=2 THEN GOTO SCNT;
      DISPLAY(' '); DISPLAY(' ');
LOOP: S1=NREAD; IF S1<0 THEN
                     IF S1=-1 THEN GOTO TERM; ELSE GOTO CMD;
                  IF S1=127 THEN GOTO LOOP;
      S2=NREAD; IF S2<0 THEN
                     IF S2=-1 THEN GOTO TERM; ELSE GOTO CMD;
            IF S2=127 THEN GOTO LOOP;
      S3=NEVALQ(S1,S2);
      S3=NPRINT(S3);
      DISPLAY('    '); DISPLAY('    ');
      GO TO LOOP;
TERM: FILENAME='AUTOLISP'; COMMAND='ERASE'; CALL IHEFILE(FCB);
      FILENAME='LISPTEXT'; I=1; RTAG='1'B;
      CALL TEXTWRK(I); COMMAND='ERASE'; CALL IHEFILE(FCB);
      COMMAND='WRBUF';
      IF SUBSTR(CARD_BUFFER,1,1)='3' THEN
            DO; CALL IHEFILE(FCB); GOTO EX; END;
      CARD_BUFFER='1'; CALL IHEFILE(FCB);
  EX: COMMAND='FINUFD'; CALL IHEFILE(FCB);
      DISPLAY('EXIT LISP SYSTEM');
      RETURN;
 CMD: IF ERRTAG THEN DO; ERRTAG='0'B; GOTO C1; END;
      IF SUBSTR(ACCUM,1,3)='OPT' THEN
            DO; TCOUNT=2; GOTO SCNT; END;
      IF SUBSTR(ACCUM,1,1)='Q' THEN
       C1: DO; IF MODETAG THEN
                  DO; MODETAG='0'B; LCOUNT=LCOUNT-1;
                  CARDNO1=CARDNO3;
                  END;
      GOTO LOOP;
      END;
IF SUBSTR(ACCUM,1,4)='DUMP' THEN
      DO; IF ASTAG THEN
             DO; ASTAG='0'B; FILENAME='AUTOLISP'; COMMAND='ERASE';
             CALL IHEFILE(FCB);
             END;
      FILENAME='DUMPLISP'; COMMAND='ERASE'; CALL IHEFILE(FCB);
      DTAG='1'B; ERSTAG='0'B;
      CALL DUSAVE; CALL TSAVE;
      GOTO LOOP;
      END;
IF SUBSTR(ACCUM,1,4)='LOAD' THEN
      DO; TCOUNT=2;
     AGN: CALL SCAN; IF TCOUNT=0 THEN GOTO LOOP;
      FILENAME=SUBSTR(ACCUM,1,COUNT); CALL OPEN;
      DISPLAY('LOADING '||FILENAME);
      IF FILENAME='LISPTEXT' THEN
        DO; CALL CLOSE; FILENAME='LISPTEMP'; CALL OPEN;
        COMMAND='RDBUF'; CARD_NUMBER=1; CALL IHEFILE(FCB);
        CH4=SUBSTR(CARD_BUFFER,10,4);
        UNSPEC(ESTART)=UNSPEC(CH4);
        CH4=SUBSTR(CARD_BUFFER,20,4);
        UNSPEC(FREE)=UNSPEC(CH4); CH4=SUBSTR(CARD_BUFFER,30,4);
        UNSPEC(BNUM)=UNSPEC(CH4); CH4=SUBSTR(CARD_BUFFER,2,4);
        UNSPEC(LCOUNT)=UNSPEC(CH4); CH4=SUBSTR(CARD_BUFFER,6,4);
        UNSPEC(CARDNO1)=UNSPEC(CH4);
        CH4=SUBSTR(CARD_BUFFER,14,4);
        UNSPEC(EFSTOR)=UNSPEC(CH4);
        CARD_NUMBER=2; CALL IHEFILE(FCB);
        DO WHILE(STATUS¬=12);
        FILENAME='LISPTEXT'; COMMAND='WRBUF'; CALL IHEFILE(FCB);
        CARD_NUMBER=CARD_NUMBER+1;
        FILENAME='LISPTEMP'; COMMAND='RDBUF'; CALL IHEFILE(FCB);
        END;
        COMMAND='ERASE'; CALL IHEFILE(FCB); /* ERASE LISPTEMP */
        CARD_NUMBER=1; FILENAME='LISPTEXT';
        IF ASTAG THEN CARD_BUFFER='2'; ELSE
        CARD_BUFFER='1'; ERSTAG='0'; COMMAND='WRBUF';
        CALL IHEFILE(FCB); GOTO AGN;
        END;
      CALL LOADER; COMMAND='ERASE'; CALL IHEFILE(FCB); GOTO AGN;
      END;
IF SUBSTR(ACCUM,1,1)='E' THEN
      DO; CALL EDITOR; IF ENDTAG THEN GOTO TERM; ELSE GOTO LOOP;
```

```
            END;
        IF SUBSTR(ACCUM,1,5)='COUNT' THEN
            DO; FSEND=FSTOR(FREE); I=0;
                    DO WHILE(FSEND¬=0);
                    FSEND=FSTOR(FSEND); I=I+1;
                    END;
            I=I+1;
            DISPLAY('CELLS IN FREE STORAGE:           '||I);
            DISPLAY('CELLS AVAILABLE FOR NUMBERS:'||300-MFSTOR+BN
            GOTO LOOP;
            END;
        IF SUBSTR(ACCUM,1,COUNT)='COLLECTN' THEN
            DO; DO WHILE(BNUM<=MFSTOR);
                FSTOR(BNUM)=BNUM+1; BNUM=BNUM+1;
                END; BNUM=BNUM-1; GOTO LOOP;
            END;
        IF SUBSTR(ACCUM,1,7)='COLLECT' THEN
            DO; CALL COLLECT; GOTO LOOP;
            END;

        DISPLAY('**S2: SUPERVISOR COMMAND NOT RECOGNIZED**');
        GOTO LOOP;

TEXTWRK: ENTRY(LC1);

        DECLARE
        LC1 BIN FIXED,
        NIL BIN FIXED EXT,
        NUMB CHAR(8),
        NUMB1 FIXED DECIMAL;

        FILENAME='LISPTEXT'; FILETYPE='DATA';
        IF ERSTAG THEN
            DO; COMMAND='ERASE'; CALL IHEFILE(FCB);
            CARD_NUMBER=1; IF ASTAG THEN CARD_BUFFER='2';
                ELSE CARD_BUFFER='1';
            COMMAND='WRBUF'; CALL IHEFILE(FCB); ERSTAG='0'B;
            END;
        IF RTAG THEN
            DO; CALL OPEN; COMMAND='RDBUF'; GOTO READ; END;
        IF ¬CTAG THEN CARD_BUFFER='          '||BUFFER;
           ELSE DO; NUMB1=LC1; NUMB=NUMB1; CARDNO3=CARDNO1;
             IF LC1<10 THEN DO;
                    CARD_BUFFER=SUBSTR(NUMB,8,1)||'         '||BUFFER;
                    GOTO T2; END;
```

```
          IF LC1<100 THEN DO; CARD_BUFFER=SUBSTR(NUMB,7,2)||
               '        '||BUFFER;
               GOTO T2; END;
          IF LC1<1000 THEN CARD_BUFFER=SUBSTR(NUMB,6,3)||
               '        '||BUFFER;
             END;
    T2: COMMAND='WRBUF'; CARD_NUMBER=CARDNO1; CALL IHEFILE(FCB);
      IF CTAG THEN
      DO; IF LC1=1 THEN ESTART=NCONS(LC1,NCONS(CARDNO1,NIL));
      ELSE ESTART=NCONS(LC1,NCONS(CARDNO1,ESTART));
      END;
      CARDNO1=CARDNO1+1;
      RETURN;
READ: CARD_NUMBER=LC1; CALL IHEFILE(FCB); CALL CLOSE; RETURN;

 OPEN: ENTRY;

      COMMAND='STATE'; CALL IHEFILE(FCB);
      COMMAND='SETUP'; CALL IHEFILE(FCB);
      RETURN;

 CLOSE: ENTRY;

      COMMAND='FINIS'; CALL IHEFILE(FCB);
      RETURN;

TSAVE: ENTRY;

      DCL KT BIN FIXED STATIC;

      I=1; RTAG='1'B; CALL TEXTWRK(I); RTAG='0'B;
      IF DTAG THEN SUBSTR(CARD_BUFFER,1,1)='3'; CALL CLOSE;
      COMMAND='WRBUF'; CALL IHEFILE(FCB); COMMAND='FINUFD';
      CALL IHEFILE(FCB); DTAG='0'B; KT=1;
      FILENAME='LISPTEMP'; COMMAND='ERASE'; CALL IHEFILE(FCB);
      RTAG='1'B; CALL TEXTWRK(KT);
      DO WHILE(KT<CARDNO1);
           CARD_NUMBER=KT; FILENAME='LISPTEMP'; COMMAND='WRBUF';
           CALL IHEFILE(FCB); KT=KT+1; CALL TEXTWRK(KT);
      END;
      FILENAME='LISPTEMP'; COMMAND='FINUFD'; CALL IHEFILE(FCB);
      RTAG='0'B; FILENAME='LISPTEXT'; RETURN;
END LISPA;
```

```
LISPB: PROC;

/* LISP SUBROUTINE FUNCTIONS. ENTRY POINT NAMES CORRESPOND TO LIS
   FUNCTION NAMES WITH AN 'N' PRECEEDING THEM. */

     DECLARE
         APVAL BIN FIXED EXT,
         BFREE BIN FIXED EXT,
         BNUM BIN FIXED EXT INITIAL(-1),
         EXPR BIN FIXED EXT,
         F BIN FIXED EXT INITIAL(-1),
         FEXPR BIN FIXED EXT,
         FLAG BIN FIXED EXT,
         FREE BIN FIXED EXT,
         FSTOR(0:15000) BIN FIXED(31) EXT,
         FSUBR BIN FIXED EXT,
         GB16 BIT(16) STATIC ALIGNED,
         GB32 BIT(32) STATIC ALIGNED,
         GT1 BIN FIXED EXT,
         MFSTOR BIN FIXED EXT INITIAL(-1),
         MONE BIN FIXED EXT INITIAL(-1),
         NIL BIN FIXED EXT INITIAL(-1),
         NC BIN FIXED EXT,
         PNAME BIN FIXED EXT,
         SUBR BIN FIXED EXT,
         T BIN FIXED EXT INITIAL(-1),
         TRACE BIN FIXED EXT,
         TRCONS BIN FIXED EXT;

NCAAR: ENTRY(KAA); DCL KAA; RETURN(NCAR(NCAR(KAA)));
NCADR: ENTRY(KAD); DCL KAD; RETURN(NCAR(NCDR(KAD)));
NCDAR: ENTRY(KDA); DCL KDA; RETURN(NCDR(NCAR(KDA)));
NCDDR: ENTRY(KDD); DCL KDD; RETURN(NCDR(NCDR(KDD)));
NCAAAR: ENTRY(KAAA); DCL KAAA; RETURN(NCAR(NCAAR(KAAA)));
NCAADR: ENTRY(KAAD); DCL KAAD; RETURN(NCAR(NCADR(KAAD)));
NCADAR: ENTRY(KADA); DCL  KADA; RETURN(NCAR(NCDAR(KADA)));
NCDAAR: ENTRY(KDAA); DCL KDAA; RETURN(NCDR(NCAAR(KDAA)));
NCADDR: ENTRY(KADD); DCL KADD; RETURN(NCAR(NCDDR(KADD)));
NCDADR: ENTRY(KDAD); DCL KDAD; RETURN(NCDR(NCADR(KDAD)));
NCDDAR: ENTRY(KDDA); DCL KDDA; RETURN(NCDR(NCDAR(KDDA)));
NCDDDR: ENTRY(KDDD); DCL KDDD; RETURN(NCDR(NCDDR(KDDD)));

NULL: ENTRY(NLL);
         /* RETURNS T IF NLL IS THE NULL LIST, F OTHERWISE */
             DCL NLL;
```

```
          IF NLL=NIL THEN RETURN(T);
          RETURN(F);

NEQUAL: ENTRY(JEQL,KEQL) RECURSIVE;
        /* RETURNS T IF J AND K ARE THE SAME S-EXPRESSION
           AND F OTHERWISE  */
          DCL (JEQL,KEQL);
          IF NATOM(JEQL)=T & NATOM(KEQL)=T THEN
               RETURN(NEQ(JEQL,KEQL));
          IF NATOM(JEQL)=T | NATOM(KEQL)=T THEN
               RETURN(F);
          IF NEQUAL(NCAR(JEQL),NCAR(KEQL))=T THEN
               RETURN(NEQUAL(NCDR(JEQL),NCDR(KEQL)));
          RETURN(F);

NAPPEND: ENTRY(JAPP,KAPP) RECURSIVE;
        /*  APPENDS LIST KAPP TO END OF LIST JAPP  -  VALUE RETURNED
          IS NEW LIST  */
          DCL (JAPP,KAPP);
          IF JAPP=NIL THEN RETURN(KAPP);
          RETURN(NCONS(NCAR(JAPP),NAPPEND(NCDR(JAPP),KAPP)));

NCOPY:ENTRY(JCOP) RECURSIVE;
        /*  RETURNS A COPY OF LIST JCOP  */
          DCL (JCOP,MCOP,NCOP);
          IF JCOP=NIL THEN RETURN(NIL);
          IF NATOM(JCOP)=T THEN RETURN(JCOP);
          MCOP=NCOPY(NCAR(JCOP));
          NCOP=NCOPY(NCDR(JCOP));
          RETURN(NCONS(MCOP,NCOP));

MEMBER: ENTRY(JMEM,KMEM) RECURSIVE;
        /*  RETURNS T IF LIST JMEM IS A MEMBER IF LIST KMEM,
          F OTHERWISE  */
          DCL (JMEM,KMEM);
          IF KMEM=NIL THEN RETURN(F);
          IF NEQUAL(JMEM,NCAR(KMEM))=T THEN RETURN(T);
          RETURN(MEMBER(JMEM,NCDR(KMEM)));

NPAIRLS: ENTRY(JPAIR,KPAIR,LPAIR) RECURSIVE;
        /*  CREATES A LIST OF PAIRS OF CORRESPONDING ELEMENTS OF
           LISTS JPAIR AND KPAIR AND PUTS ON FRONT OF LIST LPAIR*/
          DCL (JPAIR,KPAIR,LPAIR,MPAIR,NPAIR);
          IF JPAIR=NIL THEN RETURN(LPAIR);
          MPAIR=NCONS(NCAR(JPAIR),NCAR(KPAIR));
```

```
             NPAIR=NPAIRLS(NCDR(JPAIR),NCDR(KPAIR),LPAIR);
             RETURN(NCONS(MPAIR,NPAIR));

NASSOC: ENTRY(JAS,LAS);
        /*  RETURNS POINTER TO PAIR ON LIST LAS (A-LIST) WHOSE
             FIRST TERM = JAS */
             DCL (JAS,LAS);
        NASS:IF NEQUAL(NCAAR(LAS),JAS)=T THEN RETURN(NCAR(LAS));
             LAS=NCDR(LAS);
             IF LAS=NIL THEN RETURN(NIL);
             GO TO NASS;

NSUB2: ENTRY(LSUB,KSUB) RECURSIVE;
        /*  RETURNS SECOND TERM OF PAIR ON LIST LSUB HAVING FIRST
             TERM EQUAL TO KSUB  */
             DCL (LSUB,KSUB);
             IF LSUB=NIL THEN RETURN(KSUB);
             IF NEQ(NCAAR(LSUB),KSUB)=T THEN RETURN(NCDAR(LSUB));
             RETURN(NSUB2(NCDR(LSUB),KSUB));

NSUBLIS: ENTRY(LSBL,KSBL) RECURSIVE;
        /*  RETURNS AN S-EXPRESSION  IN WHICH ALL VARIABLES IN
             S-EXP. KSBL HAVE BEEN REPLACED BY THE VALUES TO
             WHICH THEY ARE CURRENTLY BOUND ON THE A-LIST (LSBL) *
             DCL (LSBL,KSBL);
             IF NATOM(KSBL)=T THEN RETURN(NSUB2(LSBL,KSBL));
             RETURN(NCONS(NSUBLIS(LSBL,NCAR(KSBL)),
                        NSUBLIS(LSBL,NCDR(KSBL))));

NSASSOC: ENTRY(JSAS,KSAS,LFN);
        /*  RETURNS DOTTED PAIR FROM LIST KSAS WHOSE CAR IS EQ
             TO JSAS, IF NO SUCH PAIR EXISTS, RETURNS THE
             FUNCTION LFN  */
             DCL (JSAS,KSAS,LFN,MSAS);
             MSAS=KSAS;
        SAS: IF MSAS=NIL THEN RETURN(LFN);
             IF NEQ(NCAAR(MSAS),JSAS)=T THEN RETURN(NCAR(MSAS));
             MSAS=NCDR(MSAS);
             GO TO SAS;

NCONC: ENTRY(JNC,KNC) ;
        /*  RETURNS LIST JNC WITH LIST KNC ADDED ON TO THE END   *
             DCL (JNC,KNC,MNC,NCONA);
             IF JNC=NIL THEN RETURN(KNC);
             MNC=JNC;
        NCON:IF NCDR(MNC)=NIL THEN
                  DO; NCONA=NRPLACD(MNC,KNC); RETURN(JNC); END;
             MNC=NCDR(MNC);
             GO TO NCON;


NATTRIB: ENTRY(JATR,KATR);
        /*  ATTACHES LIST K ONTO END OF LIST J AND RETURNS K   */
             DCL (JATR,KATR,N);
             N=NCONC(JATR,KATR);
             RETURN(KATR);

NSUBST: ENTRY(JST,KST,LSBST) RECURSIVE;
        /*  RETURNS LIST FORMED BY SUBSTITUTING JST FOR EVERY
             OCCURRENCE OF KST IN LSBST   */
             DCL (JST,KST,LSBST);
             IF NEQUAL(KST,LSBST)=T THEN RETURN(JST);
             IF NATOM(LSBST)=T THEN RETURN(LSBST);
             RETURN(NCONS(NSUBST(JST,KST,NCAR(LSBST)),
                  NSUBST(JST,KST,NCDR(LSBST))));

 NPROP: ENTRY (JPR,KPR,IPFN);      /* 11-12-69 */
        /* RETURNS CDR OF CELL WHOSE CAR IS EQ TO KPR IF ONE EXISTS,
             RETURNS THE FUNCTION IPFN OTHERWISE */
             DCL (JPR,JPR1,KPR,IPFN) FIXED BIN;
             IF NATOM(JPR)=T THEN JPR1=NCDR(JPR); ELSE JPR1=JPR;
        PROP:IF JPR1=NIL THEN RETURN(IPFN);
             IF NEQ(NCAR(JPR1),KPR)=T THEN RETURN(NCDR(JPR1));
             JPR1=NCDR(JPR1);
             GO TC PROP;

NGET: ENTRY(JGET,KGET);     /* 11-12-69 */
        /* RETURNS CADR OF CELL WHOSE CAR IS EQUAL TO KGET IF ONE
             EXISTS, RETURNS NIL OTHERWISE */
             DCL (JGET,JGET1,KGET,KGET1) FIXED BIN;
             IF NATOM(JGET)=T THEN JGET1=NCDR(JGET);
                       ELSE JGET1=JGET;
       GETA: IF JGET1=NIL THEN RETURN(NIL);
             IF NEQ(NCAR(JGET1),KGET)=T THEN RETURN(NCADR(JGET1));
             JGET1=NCDR(JGET1);
             GO TO GETA;

NPAIR: ENTRY(JPAR,KPAR) ;
        /*  RETURNS LIST OF DOTTED PAIRS OF CORRESPONDING ELEMENTS OF
             LISTS JPAR AND KPAR WHICH MUST BE OF EQUAL LENGTH  */
```

```
              DCL (JPAR,KPAR,LPAR,MPAR,NPAR);
              LPAR=NIL;
              MPAR=JPAR; NPAR=KPAR;
      PAIRA:  IF MPAR=NIL | NPAR=NIL THEN
              DO;  IF MPAR=NIL THEN
                        IF NPAR=NIL THEN RETURN(NREVERS(LPAR));
                   RETURN(NIL);
              END;
              LPAR=NCONS(NCONS(NCAR(MPAR),NCAR(NPAR)),LPAR);
              MPAR=NCDR(MPAR); NPAR=NCDR(NPAR);
              GO TO PAIRA;

NUMBERP: ENTRY(NUMP) ;
        /*  RETURNS T IF NUMP IS AN INTEGER NUMBER  -  F OTHERWISE
              DCL NUMP;
              IF NCDR(NUMP)>BNUM THEN RETURN(T);
              RETURN(F);

NEFFACE: ENTRY(JEF,KEF)  ;
        /* DELETES ITEM JEF FROM THE TOP LEVEL OF LIST KEF AND
              RETURNS ALTERED LIST KEF  */
              DCL (JEF,KEF,KEF1,EF1,EF2) FIXED BIN;
              KEF1=KEF;
              DO WHILE (NEQUAL(JEF,NCAR(KEF1))=T);
                   KEF1=NCDR(KEF1); END;
              EF2=KEF1; EF1=NCDR(KEF1);
              DO WHILE (EF1¬=NIL);
                   IF NEQUAL(NCAR(EF1),JEF)=T THEN
                   DO;  EF1=NCDR(EF1);
                        EF2=NRPLACD(EF2,EF1);
                   END;
                   ELSE DO; EF2=EF1; EF1=NCDR(EF2); END;
              END;
              RETURN(KEF1);


 LENGTH: ENTRY(LGTH) ;
        /*  RETURNS LENGTH OF TOP LEVEL OF LIST LGTH  */
              DCL (LGTH,KLG,NLG);
              KLG=LGTH;
              NLG=0;
              DO WHILE(KLG¬=NIL);  NLG=NLG+1; KLG=NCDR(KLG); END;
              FSTOR(BNUM)=NLG;
              NLG=NCONS(MONE,BNUM);
              BNUM=BNUM-1;
```

```
            RETURN(NLG);

LAST: ENTRY(JLST) ;
        /*  RETURNS THE LAST ITEM ON LIST JLST  */
            DCL (JLST,KLST);
            KLST=JLST;  IF JLST=NIL THEN RETURN(NIL);
            DO WHILE(NCDR(KLST)¬=NIL); KLST=NCDR(KLST); END;
            RETURN(KLST);

NDEFINE: ENTRY(LDEF) RECURSIVE;
        /*  LDEF IS A LIST OF PAIRS OF NAMES AND LAMBDA EXPRESSIONS.
            NDEFINE PUTS AN 'EXPR' INDICATOR AND THE LAMBDA EXPRESSION
            ON THE PROPERTY LIST FOR EACH NAME.  THE VALUE OF NDEFINE
            IS A LIST OF NAMES DEFINED IN THIS WAY  */

            DCL (LDEF,KDEFT,LDEFT);
            IF LDEF=NIL THEN RETURN(NIL);
            LDEFT=NPROP(NCAAR(LDEF),PNAME,NIL);
            KDEFT=NREMPRP(NCAAR(LDEF),EXPR);
            KDEFT=NCDR(LDEFT);
            KDEFT=NCONS(NCAR(NCDAR(LDEF)),KDEFT);
            KDEFT=NCONS(EXPR,KDEFT);
            KDEFT=NRPLACD(LDEFT,KDEFT);
            RETURN(NCONS(NCAAR(LDEF),NDEFINE(NCDR(LDEF))));

NDEFLST: ENTRY(NDEF,NIND) RECURSIVE;
        /*  THIS FUNCTIONS WORKS JUST LIKE NDEFINE EXCEPT THE INDICATOR
            'NIND' SUPPLIED AS AN ARGUMENT IS PUT ON THE PROPERTY LIST
            RATHER THAN 'EXPR'.  VALUE IS A LIST OF NAMES DEFINED  */

            DCL (NDEF,NIND,MDEFT,NDEFT);
            IF NDEF=NIL THEN RETURN(NIL);
            NDEFT=NPROP(NCAAR(NDEF),PNAME,NIL);
            MDEFT=NREMPRP(NCAAR(NDEF),NIND);
            MDEFT=NCDR(NDEFT);
            MDEFT=NCONS(NCAR(NCDAR(NDEF)),MDEFT);
            MDEFT=NCONS(NIND,MDEFT);
            MDEFT=NRPLACD(NDEFT,MDEFT);
            RETURN(NCONS(NCAAR(NDEF),NDEFLST(NCDR(NDEF),NIND)));

NREVERS: ENTRY(LSTRU);
        /* RETURNS THE REVERSE OF THE TOP LEVEL OF LIST LSTRU */
            DCL (LSTRU,LSTRV);
            LSTRV=NIL;
        AREV:IF LSTRU=NIL THEN RETURN(LSTRV);
```

```
            LSTRV=NCONS(NCAR(LSTRU),LSTRV);
            LSTRU=NCDR(LSTRU); GO TO AREV;

NREMPRP: ENTRY(NRM,NRMP) ;
        /*  DELETES ALL OCCURRENCES OF INDICATOR AND RELATED
            PROPERTY FROM PROP. LIST OF ATOM NRM */
            DCL (NRM,NRMP,NRMP1,NRM2);
            NRMP1=NRM;
            DO WHILE (NCDR(NRMP1)¬=NIL);
                IF NEQ(NCADR(NRMP1),NRMP)=T THEN
                     NRM2=NRPLACD(NRMP1,NCDDDR(NRMP1));
                ELSE NRMP1=NCDR(NRMP1);
            END;
            RETURN(NIL);

NFLAG: ENTRY(FLST,FID);
        /* PUTS THE FLAG INDICATOR 'FID' ON THE PROPERTY LIST
           OF EACH ATOMIC SYMBOL ON LIST 'FLST' IMMEDIATELY
           FOLLOWING THE ATOM HEADER  */
            DCL (FLST,FID,FLST1,FLST2) FIXED BIN;
            FLST1=FLST;
        NFLG:IF FLST1=NIL THEN RETURN(NIL);
            FLST2=NRPLACD(NCAR(FLST1),NCONS(FID,NCDAR(FLST1)));
            FLST1=NCDR(FLST1);
            GO TO NFLG;

NREMFLG: ENTRY(RMFG,RID);
        /* REMOVES ALL OCCURRENCES OF FLAG 'RID' FROM THE PROPERT
           LIST OF ALL ATOMIC SYMBOLS ON LIST 'RMFG  */
            DCL (RMFG,RID,RMFG1,RMFG2,NFG) FIXED BIN;
            RMFG1=RMFG;
            DO WHILE (RMFG1¬=NIL);
                 RMFG2=NCAR(RMFG1);
                 DO WHILE (NCDR(RMFG2)¬=NIL);
                      IF NEQ(NCADR(RMFG2),RID)=T THEN
                           NFG=NRPLACD(RMFG2,NCDDR(RMFG2));
                 ELSE RMFG2=NCDR(RMFG2);
                 END;
                 RMFG1=NCDR(RMFG1);
            END;
            RETURN(NIL);

NOT: ENTRY(NOTARG);
        /* RETURNS F IF NOTARG IS TRUE, OTHERWISE RETURNS T */
            DCL NOTARG;
            IF NOTARG=F | NOTARG=NIL THEN RETURN(T);
            RETURN(F);


NTRACE: ENTRY(TRLST);
            /* PUTS TRACE INDICATOR ('FLAG') ON THE PROPERTY LIST
                OF EACH ATOMIC SYMBOL IN LIST TRLST */
            DCL TRLST FIXED BIN;
            RETURN(NFLAG(TRLST,FLAG));

NUNTRCE: ENTRY(UTRL);
        /* REMOVES TRACE INDICATORS PLACED BY FUNCTION TRACE */
            DCL UTRL FIXED BIN;
            RETURN(NREMFLG(UTRL,FLAG));

NADD1: ENTRY(NAD1);
        /* ADDS ONE TO THE VALUE REPRESENTED BY NAD1 */
            DCL (NAD1,NAD2);
            FSTOR(BNUM)=FSTOR(NCDR(NAD1))+1;
            NAD2=NCONS(MONE,BNUM);
            BNUM=BNUM-1;
            RETURN(NAD2);

NSUB1: ENTRY(NSB1);
        /* SUBTRACTS ONE FROM THE VALUE REPRESENTED BY NSB1 */
            DCL (NSB1,NSB2);
            FSTOR(BNUM)=FSTOR(NCDR(NSB1))-1;
            NSB2=NCONS(MONE,BNUM);
            BNUM=BNUM-1;
            RETURN(NSB2);

END LISPB;
```

```
LISPC: PROC;

/* LISPC: SYNTAX & LEXICAL ANALYSIS */

 NREAD: ENTRY BIN FIXED;

 /* THIS PROCEDURE IS CALLED ONCE PER S-EXPRESSION. IT RETURNS THE
    POINTER TO THE S-EXPRESSION TREE IT HAS BUILT. IT RETURNS A
    "-1" ON AN "END LISP" INPUT, AND A "-2" WHEN IT SCANS A SUPERV
    COMMAND, I.E. ONE WHICH ENDS IN A "$". */

        DECLARE
        ACCUM CHAR(30) EXTERNAL,
        ASCTR BIN FIXED EXT,
        ASTAG BIT(1) EXT,
        BNUM BIN FIXED EXT,
        BP BIN FIXED EXTERNAL,
        BUFFER CHAR(72) EXTERNAL,
        BUFFNO BIN FIXED EXTERNAL,
        CADD BIN FIXED EXTERNAL,
        COUNT BIN FIXED EXTERNAL,
        CTAG BIT(1) EXT,
        CTR BIN FIXED EXTERNAL,
        DOTTAG BIT(1) STATIC ALIGNED,
        ERRTAG BIT(1) EXT,
        1 FCB EXT,
             2 COMMAND CHAR(8),
             2 FILENAME CHAR(8),
             2 FILETYPE CHAR(8),
             2 CARD_NUMBER BIN FIXED(31,0),
             2 STATUS BIN FIXED(31,0),
             2 CARD_BUFFER CHAR(80),
        INTAG BIT(1) EXT,
        JJ BIN FIXED EXT,
        LDIG FIXED BIN(31) STATIC,
        LCOUNT BIN FIXED EXT,
        LINE BIN FIXED EXTERNAL,
        MODE BIN FIXED EXT,
        MODETAG BIT(1) EXT,
        MONE BIN FIXED INITIAL(-1) EXT,
        MTWO BIN FIXED INITIAL(-2) EXT,
        NIL BIN FIXED EXTERNAL,
        N127 BIN FIXED STATIC,
        PLEV BIN FIXED EXTERNAL,
        POSIT BIN FIXED(31) EXTERNAL,
      PTAG BIT(1) EXT,
      RESULT BIN FIXED EXTERNAL,
      RE1TAG BIT(1) EXT,
      SEXP BIN FIXED EXTERNAL,
      STAC(200) BIN FIXED(31) EXTERNAL,
      STKTAB BIT(1) EXTERNAL,
      SWTCH(1:3) LABEL(ATM,DIG,SPEC)
               INITIAL(ATM,DIG,SPEC),
      T BIN FIXED EXTERNAL,
      TCOUNT BIN FIXED EXTERNAL,
      VARC CHAR(25) VARYING;

      N127=127; DOTTAG='0'B;
      IF ¬STKTAB THEN
           DO; BP=0; JJ=-1; BUFFNO=BUFFNO+1;
           END;
      ELSE DO; IF ¬MODETAG THEN
              DO; TCOUNT=0; IF ¬RE1TAG THEN BP=72; CTAG='1'B;
              END;
              ELSE TCOUNT=1;
           CTR=200; SEXP,PLEV=0; JJ=1;
           END;
CONT: CALL SCAN;
      IF RESULT=0 THEN RETURN(MONE);  /* 'END LISP' WAS INPUT */
      IF SUBSTR(ACCUM,COUNT,1)='$' THEN RETURN(MTWO);
      IF STKTAB & INTAG THEN DO; CALL INCWRK; INTAG='0'B; END;
      GOTO SWTCH(RESULT);
 ATM: CALL HASH;  /* HASH CODE */
      IF ¬STKTAB THEN
           DO; IF BUFFNO=2 | BUFFNO=3 THEN
                DO; L=LINE; CALL INIT1(L); END;
           GOTO IN;
           END;
      IF JJ<0 THEN
  IN: DO; CALL ENTER(POSIT); JJ=1; IF STKTAB THEN
           DO; L=NEVAL2; IF L<0 THEN RETURN(SEXP); END;
      ELSE DO; L=LINE; LINE=LINE+1;
           IF BUFFNO>4 THEN
              IF BUFFNO=13 THEN CALL INITL3(L);
              ELSE CALL INITL2(L);
           ELSE CALL INITL(L);
           END;
      GOTO CONT;
      END;
      JJ=LOOKUP(POSIT); IF JJ<0 THEN GOTO IN;
      ELSE DO; IF STKTAB THEN
            DO; L=NEVAL2; IF L<0 THEN RETURN(SEXP); END;
           GOTO CONT;
           END;
 DIG:  LDIG=SUBSTR(ACCUM,1,COUNT);
       IF MODE=7 THEN LDIG=-LDIG;
       FSTOR(BNUM)=LDIG; CADD=NCONS(MONE,BNUM);
       BNUM=BNUM-1; FSTOR(BNUM)=0;
       L=NSTACK(CADD); GOTO CONT;
 SPEC: IF MODE=6 THEN DO; PLEV=PLEV+1; GOTO CNT; END;
       IF MODE=4 THEN DO; L=1; RETURN(L); END;
       IF MODE=16 THEN GOTO CONT;
       IF MODE=7 THEN
            DO; PLEV=PLEV-1;
            IF PLEV<0 THEN
                 DO; DISPLAY('*I1: PARENTHESES ERROR*');
                 ERRTAG='1'B; RETURN(MTWO);
                 END;
            IF NEVAL3<0 THEN RETURN(SEXP); GOTO CONT;
            END;
       IF MODE=2 THEN DO; DOTTAG='1'B; GOTO CNT; END;
       DISPLAY('ILLEGAL SPECIAL CHARACTER IN S-EXPRESSION');
       RETURN(N127);
  CNT: L=NSTACK(MODE); GOTO CONT;

EVAL1: ENTRY;
 /* EVALUATES AN ATOM S-EXPRESSION */
       MODETAG=¬MODETAG; L=NUNSTK; SEXP=L;
       RETURN;

NEVAL2: ENTRY BIN FIXED;
       L=NSTACK(CADD);
       IF L=200 THEN
            DO; CALL EVAL1; RETURN(MONE); END;
       RETURN(SEXP);

NEVAL3: ENTRY BIN FIXED;
/* CALLED UPON SCANNING A ')'. RETURNS A "-1" ON PLEV=0. */
       IF DOTTAG THEN GOTO DOTT;
       L1=NUNSTK;
       IF L1=6 THEN   /* NIL LIST: () */
            DO; L1=NIL;
            L=NSTACK(L1); IF PLEV=0 THEN
                 DO; MODETAG=¬MODETAG; SEXP=L1; RETURN(MONE); END;
            RETURN(L1);
```

```
              END;
       IF L1=2 THEN GOTO CH;
       SEXP=NCONS(L1,NIL);
   CH: DO WHILE('1'B); L1=NUNSTK;
       IF L1=6 THEN
              DO; IF PLEV=0 THEN
                     DO; MODETAG=¬MODETAG; RETURN(MONE); END;
              L1=NSTACK(SEXP); RETURN(L1);
              END;
       SEXP=NCONS(L1,SEXP);
       END;
 DOTT: L1=NUNSTK; L2=NUNSTK;
       IF L2¬=2 THEN
              DO; DOTTAG='0'B; GOTO CH; END;
       L=NUNSTK; SEXP=NCONS(L,L1);
       L1=NUNSTK; IF PLEV=0 THEN
              DO; MODETAG=¬MODETAG; RETURN(MONE); END;
       L1=NUNSTK; L=NSTACK(L1);
       IF L1=2 THEN DOTTAG='1'B;
                ELSE DOTTAG='0'B;
       L1=NSTACK(SEXP); RETURN(L1);

NSTACK: ENTRY(J3) BIN FIXED;
       DECLARE
       J3 BIN FIXED,
       L1 BIN FIXED;

       STAC(CTR)=J3; L1=CTR; CTR=CTR-1; RETURN(L1);

NUNSTK: ENTRY BIN FIXED;

       DECLARE
       J4 BIN FIXED;

       CTR=CTR+1; J4=STAC(CTR); STAC(CTR)=0; RETURN(J4);

INCWRK: ENTRY;

       IF RE1TAG THEN GOTO JP1;
              CALL TEXTWRK(LCOUNT);
              IF ASTAG & CTAG THEN ASCTR=ASCTR+1;
              IF CTAG THEN LCOUNT=LCOUNT+1;
  JP1: IF PTAG THEN
              DO; DISPLAY(' ');
              IF ¬CTAG THEN DISPLAY('             '||BUFFER);
```

```
                    ELSE DISPLAY(LCOUNT-1||'  '||BUFFER);
            END;
        CTAG='0'B;
        RETURN;

  HASH: ENTRY;

        POSIT=MOD(UNSPEC(SUBSTR(ACCUM,1,3))+CCUNT+
            16777214,N127)+1;
        RETURN;

/* LEXICAL ANALYSIS PHASE */

 SCAN: ENTRY;
        DECLARE
        ALPBASE BIN FIXED EXTERNAL,
        CHARSET CHAR(52) EXTERNAL,
        DIGBASE BIN FIXED EXTERNAL,
        FF BIN FIXED STATIC,
        KK BIN FIXED STATIC,
        NO BIN FIXED,
        NUMBERSW(0:3) LABEL (SCANMANT,SCANFRAC,SCANEXSN,SCANEX)
                    INITIAL (SCANMANT,SCANFRAC,SCANEXSN,SCANEX),
        PNAME BIN FIXED EXTERNAL,
        READER ENTRY(CHAR(25) VARYING,CHAR(72)) RETURNS(BIT(1)),
        SWITCH(0:3) LABEL (DEBLANK,IDENT,DIGIT,QNUMBER)
                  INITIAL (DEBLANK,IDENT,DIGIT,QNUMBER),
        TT CHAR(1) STATIC;

        RESULT,COUNT,FF,MODE=0;
        ACCUM=' ';
        DO WHILE('1'B);
            IF BP=72 & TCCUNT=2 THEN DO; TCOUNT=0; RETURN; END;
            BP=BP+1;
            IF BP>72 THEN
            DO; IF REITAG THEN
              DO; CALL GBUFF; BUFFER=SUBSTR(CARD_BUFFER,9,72);
              GOTO JP;
              END;
            IF CTAG THEN VARC='CALL EVALQUOTE, ARGS:';
            ELSE VARC=' '; IF ¬READER(VARC,BUFFER) THEN
            DO; RESULT=0; RETURN; END;
     JP: BP=1; INTAG='1'B;
            END;
        TT=SUBSTR(BUFFER,BP,1); KK=INDEX(CHARSET,TT);
```

```
         GOTO SWITCH(RESULT);
DEBLANK: IF KK>1 THEN
              IF KK>ALPBASE THEN
                   IF KK>DIGBASE THEN RESULT=2;
                   ELSE RESULT=1;
              ELSE IF KK=7 | KK=8 THEN    /* +-.*/
                   DO; RESULT=3; MODE=KK;
                   END;
                   ELSE   /* SPECIAL CHARACTER*/
                   DO; MODE=KK; COUNT=1; SUBSTR(ACCUM,1,1)=TT;
                   RESULT=3; GOTO RETURNL;
                   END;
         ELSE GOTO XIT;
         GOTO STORET;
IDENT: IF KK>ALPBASE THEN GOTO STORET;
         IF KK=3 THEN GOTO STORET;
         GOTO BACKUP;
DIGIT: GOTO NUMBERSW(FF);
 SCANMANT: IF KK>DIGBASE THEN GOTO STORET;
         IF KK=2 THEN    /* DECIMAL POINT */
              DO; MODE=1; FF=1;
              GOTO STORET;
              END;
         ELSE GOTO CHECKEXP;
 SCANFRAC: IF KK>DIGBASE THEN GOTO STORET;
 CHECKEXP: IF TT='E' THEN
              DO; FF=2; MODE=1;
              GOTO STORET;
              END;
         GOTO BACKUP;
 SCANEXSN: IF KK=7 | KK=8 THEN
         DO; FF=3; GOTO STORET;
         END;
         IF KK>DIGBASE THEN
              DO; FF=3; GOTO STORET;
              END;
         GOTO BACKUP;
 SCANEX: IF KK>DIGBASE THEN GOTO STORET;
         GOTO BACKUP;
 QNUMBER: IF KK>DIGBASE THEN
              DO; RESULT=2;
              IF MODE=5 THEN
                   DO; MODE=1; FF=1;
                   END;
              ELSE MODE=0;
```

```
                GOTO STORET;
                END;
           GOTO BACKUP;
STORET:    IF COUNT>30 THEN GOTO RETURNL;
           COUNT=COUNT+1;
           SUBSTR(ACCUM,COUNT,1)=TT;
   XIT:    END;
BACKUP:    BP=BP-1;
RETURNL:   RETURN;

 ENTER:    ENTRY(J1);

 /*  THIS PROCEDURE RECEIVES THE LOCATION IN OBLIST OF AN ATOM AND
     THEN SETS UP THE ATOM WITH ITS PRINT NAME STRING, SETTING
     THE ADDRESS OF ITS HEADER CELL TO THE EXTERNAL VARIABLE 'CADD'.

           DECLARE
           FSTOR(0:16000) BIN FIXED(31) EXT,
           J1 BIN FIXED,
           LB32 BIT(32) ALIGNED,
           MCOUNT BIN FIXED,
           PN BIN FIXED,
           TE BIN FIXED,
           TWO16 FIXED BIN (31) STATIC INITIAL(65536);

           FSTOR(J1)=FSTOR(J1)+TWO16;
           L=NIL; MCOUNT=COUNT-1;
           DO I=0 TO MCOUNT;
                K=COUNT-I; TE=UNSPEC(SUBSTR(ACCUM,K,1));
                L=NCONS(TE,L);
           END;
           PN=NCONS(NCONS(MONE,NCONS(PNAME,NCONS(L,NIL))),NIL);
           CADD=NCAR(PN); /* CADD=ADDRESS OF THE ATOM HEADER CELL */
           IF NCAR(J1)=1 THEN L=NRPLACD(J1,PN);
           ELSE DO;
                L=NCDR(J1);
                DO WHILE (NCDR(L)¬=NIL); L=NCDR(L); END;
                K=NRPLACD(L,PN);
                END;
           RETURN;

 LOOKUP:   ENTRY(J2) BIN FIXED;

 /*  THIS PROCEDURE RECEIVES A LOCATION IN OBLIST AND THEN LOOKS AT
     THE ATOMS STRUNG FROM THAT LOCATION. IF IT FINDS A MATCH WITH
```

```
    THE ATOM STORED IN ACCUM THEN IT RETURNS A "1", SETTING CADD
    EQUAL TO THE ATOM HEADER CELL, ELSE IT RETURNS A "-1".  */

        DECLARE
        CP1 CHAR(1),
        CP2 CHAR(1),
        II BIN FIXED STATIC,
        J2 BIN FIXED,
        TE1 BIN FIXED;

        NO=NCAR(J2);
        IF NO=0 THEN RETURN(MONE);
        TE1=NCDR(J2);
        DO II=1 TO NO;
             TE=NCAR(TE1);
             LL=NGET(TE,PNAME);
             KK=1;
             DO WHILE (KK<=COUNT);
                  CP1=SUBSTR(ACCUM,KK,1);
                  UNSPEC(CP2)=SUBSTR(UNSPEC(FSTOR(LL)),9,8);
                  IF CP1¬=CP2 THEN GO TO LOOKA;
                  KK=KK+1; LL=NCDR(LL);
                  IF LL=NIL THEN GO TO LOOKA;
             END;
    LOOKA:   IF KK=COUNT+1 THEN
                  DO; CADD=TE; TE=1; RETURN(TE); END;
             TE1=NCDR(TE1);
        END;
        RETURN(MONE);

END LISPC;
```

```
/* LISPD: INTERPRETER,PRINT & TRACE, AND FSUBR FUNCTIONS */
NEVALQ: PROC(FNEVQ,AEVQ);

        DECLARE
        APPLY BIN FIXED EXT,
        APVAL BIN FIXED EXT,
        AEVQ BIN FIXED,
        BFREE BIN FIXED EXT,
        BLANK BIN FIXED EXT,
        BUFFCON BIN FIXED STATIC,
        CAPPLY CHAR(5) INITIAL('APPLY') STATIC,
        CEVAL CHAR(4) INITIAL('EVAL') STATIC,
        CEVALQUOTE CHAR(9) INITIAL('EVALQUOTE') STATIC,
        CEVCON CHAR(5) INITIAL('EVCON') STATIC,
        CEVLIS CHAR(5) INITIAL('EVLIS') STATIC,
        COMMA BIN FIXED EXT,
        COND BIN FIXED EXT,
        DASH BIN FIXED EXT,
        DOLLAR BIN FIXED EXT,
        EIGHT BIN FIXED INITIAL(8) STATIC,
        EQSIGN BIN FIXED EXT,
        ERFLAG BIN FIXED STATIC,
        EVAL BIN FIXED EXT,
        EVCON BIN FIXED EXT,
        EVLIS BIN FIXED EXT,
        EXPR BIN FIXED EXT,
        F BIN FIXED EXT,
        FEXPR BIN FIXED EXT,
        FIVE BIN FIXED INITIAL(5) STATIC,
        FLAG BIN FIXED EXT,
        FNEVQ BIN FIXED,
        FNCTION BIN FIXED EXT,
        FOUR BIN FIXED INITIAL(4) STATIC,
        FSTOR(0:16000) BIN FIXED(31) EXT,
        FSUBR BIN FIXED EXT,
        FUNARG BIN FIXED EXT,
        LABEL BIN FIXED EXT,
        LAMBDA BIN FIXED EXT,
        LAST ENTRY EXT,
        LENGTH ENTRY EXT,
        LPAR BIN FIXED EXT,
        NBLKS BIN FIXED STATIC,
        NIL BIN FIXED EXT,
        NINE BIN FIXED INITIAL(9) STATIC,
        NULL ENTRY EXT,
        ONE BIN FIXED INITIAL(1) STATIC,
        PERIOD BIN FIXED EXT,
        PLUSS BIN FIXED EXT,
        PNAME BIN FIXED EXT,
        PRBUFF CHAR(128) STATIC,
        PRNAME RETURNS(CHAR(30) VARYING),
        QUOTE BIN FIXED EXT,
        RECUR BIN FIXED STATIC,
        RPAR BIN FIXED EXT,
        SEVEN BIN FIXED INITIAL(7) STATIC,
        SIX BIN FIXED INITIAL(6) STATIC,
        SLASH BIN FIXED EXT,
        STAR BIN FIXED EXT,
        SUBR BIN FIXED EXT,
        T BIN FIXED EXT,
        THREE BIN FIXED INITIAL(3) STATIC,
        TRACE BIN FIXED INITIAL(0) EXT,
        TRACE1 BIN FIXED INITIAL(0)EXT,
        TWO BIN FIXED INITIAL(2) STATIC,
        VAL1 BIN FIXED;

/* EVALQUOTE */

        NBLKS=1; ERFLAG,RECUR=0; PRBUFF=(128)' ';
        IF NATOM(FNEVQ)=T THEN  /* CHECK FOR TOP LEVEL SP. FORMS */
             DO; IF NGET(FNEVQ,FEXPR)¬=NIL THEN
                DO; VAL1=NEVAL(NCONS(FNEVQ,AEVQ),NIL); GOTO EXEVQ; END;
                 IF NGET(FNEVQ,FSUBR)¬=NIL THEN
                DO; VAL1=NEVAL(NCONS(FNEVQ,AEVQ),NIL); GOTO EXEVQ; END;
             END;
             VAL1=NAPPLY(FNEVQ,AEVQ,NIL);
   EXEVQ: IF ERFLAG>0 THEN VAL1=NIL; DISPLAY(' ');
          DISPLAY('VALUE IS:');
          RETURN(VAL1);         /* RETURN TO LISP SUPERVISOR */

NAPPLY: PROC(FN,ARGS,NALST) RECURSIVE;
 /* APPLY: BINDS VARIABLES AND FUNCTION NAMES ON THE A-LIST AND
           HANDLES FUNARG DEVICE (FUNCTIONAL ARGUMENTS) */
             DCL (ABIND,FN,ARGS,NALST,CARFN,NEXPR,NSUBR,VAL2)FIXED BIN,
             (TRCA,TRC1) FIXED BIN INITIAL(0),
             FNCTN CHAR(30) VARYING;
             IF NPROP(APPLY,FLAG,NIL)¬=NIL THEN
                  DO; TRCA=1;
                      CALL PTRACE(CAPPLY,FN,ARGS,NALST,THREE,TWO);
```

```
              END;
         IF FN=NIL THEN
              DO; VAL2=NIL; GO TO EXAP; END;
         IF NATOM(FN)=T THEN

              /* LOOK FOR BINDING OF FUNCTION ON PROP. LIST */

              DO: FNCTN=PRNAME(FN);
        /*SUBR*/       NSUBR=NGET(FN,SUBR);
                   IF NSUBR¬=NIL THEN
                   DO; IF NPROP(FN,FLAG,NIL)¬=NIL THEN TRACE1=1
                       VAL2=NPROC(NSUBR,ARGS,F,FNCTN);
                       GO TO EXAP; END;
        /*EXPR*/       NEXPR=NGET(FN,EXPR);
                   IF NEXPR¬=NIL THEN
                   DO; IF NPROP(FN,FLAG,NIL)¬=NIL THEN
                       DO: CALL PTRACE(FNCTN,ARGS,NIL,NIL,ONE,1
                               TRC1=1; END;
                        VAL2=NAPPLY(NEXPR,ARGS,NALST);
                        IF TRC1=1 THEN
                        DO:  CALL PTRACE(FNCTN,VAL2,NIL,NIL,
                             ONE,ONE); TRC1=0; END;
                        GO TO EXAP;
                   END;

              /* LOOK FOR BINDING OF FUNCTION ON A-LIST */

                   ABIND=NSASSOC(FN,NALST,TWO);
                   IF ABIND=TWO THEN
                   DO;   ERFLAG=2;
         DISPLAY('UNDEFINED FUNCTION - APPLY');
                        GO TO EXAP;
                   END;
                   VAL2=NAPPLY(NCDR(ABIND),ARGS,NALST);
                   GO TO EXAP;
              END;
         CARFN=NCAR(FN);

         /* CHECK FOR SPECIAL FORMS */

/*LAMBDA*/ IF NEQ(CARFN,LAMBDA)=T THEN

         /* BIND VARIABLES AND ARGUMENTS ON A-LIST */

         DO; VAL2=NEVAL(NCADDR(FN),NCONC(NPAIR(NCADR(FN),ARGS),
                NALST)); GO TO EXAP; END;
/* LABEL */ IF NEQ(CARFN,LABEL)=T THEN

            /* CONS THE FUNCTION NAME AND DEFINITION, ADD TO A-LIST,
               AND CALL APPLY  */

              DO; VAL2=NAPPLY(NCADDR(FN),ARGS,NCONS(NCONS(NCADR(FN),
                       NCADDR(FN)),NALST));GO TO EXAP; END;

/*FUNARG*/  IF NEQ(CARFN,FUNARG)=T THEN
                DO; VAL2=NAPPLY(NCADR(FN),ARGS,NCADDR(FN));
                       GO TO EXAP; END;
            VAL2=NAPPLY(NEVAL(FN,NALST),ARGS,NALST);
     EXAP:  IF ERFLAG>0 THEN VAL2=NIL;
            IF TRCA=1 THEN DO;
                CALL PTRACE(CAPPLY,VAL2,NIL,NIL,ONE,ONE);
                TRCA=0;
                END;
            RETURN(VAL2);
            END NAPPLY;

/* EVAL */  NEVAL: PROC(FORM,ALST) RECURSIVE;
       /* EVALUATES FORMS */
            DCL ( TRCE , NCEV,TRFEX) FIXED BIN;
            DCL (FORM, ALST, ALBND,NAP,CFORM,ALBD,VAL3,INDIC,TREX)
                FIXED BIN, FNCTN CHAR(30) VARYING;
/*TRACE*/   IF NPROP(EVAL,FLAG,NIL)¬=NIL THEN
                DO; TRCE=1;
                    CALL PTRACE(CEVAL,FORM,ALST,NIL,TWO,TWO);
                    END;
/* NIL */   IF FORM=NIL THEN DO; VAL3=NIL; GO TO EXEV; END;
/*NUMBER*/  IF NUMBERP(FORM)=T THEN DO; VAL3=FORM; GO TO EXEV; END;
/* ATOM */  IF NATOM(FORM)=T THEN

                /* LOOK FOR VALUE BINDING ON PROP. LIST */

            DO; NAP=NGET(FORM,APVAL);
                IF NAP¬=NIL THEN DO; VAL3=NCAR(NAP); GO TO EXEV; END;

                /* LOOK FOR VALUE BINDING ON A-LIST */

                ALBND=NSASSOC(FORM,ALST,THREE);
                IF ALBND=THREE THEN
                DO;  ERFLAG=3;
            DISPLAY('UNBOUND VARIABLE - EVAL');
```

```
                   GO TO EXEV;
               END;
               VAL3=NCDR(ALBND);  GO TO EXEV;
           END;
           CFORM=NCAR(FORM);

           /* CHECK FOR SPECIAL FORMS */

 /*QUOTE*/ IF CFORM=QUOTE THEN
           DO;  VAL3=NCADR(FORM); GO TO EXEV; END;
/*FUNCTION*/IF CFORM=FNCTION THEN
           DO; VAL3=LIST3(FUNARG,NCADR(FORM),ALST);
               GO TO EXEV; END;
 /* COND */ IF CFORM=COND THEN
           DO;  VAL3=NEVCON(NCDR(FORM),ALST); GO TO EXEV; END;

           /* TEST FOR PROG HERE WHEN IT IS IMPLEMENTED */

           IF NATOM(CFORM)=T THEN
           DO; FNCTN=PRNAME(CFORM);

           /* LOOK FOR FUNCTION DEFINITION ON PROP. LIST */

/*SUBR*/       INDIC=NGET(CFORM,SUBR); IF INDIC¬=NIL THEN
               DO; IF NPROP(CFORM,FLAG,NIL)¬=NIL THEN TRACE1=T;
                   VAL3=NPROC(INDIC,NEVLIS(NCDR(FORM),ALST),
                    F,FNCTN); GO TO EXEV; END;
/*EXPR*/       INDIC=NGET(CFORM,EXPR); IF INDIC¬=NIL THEN
               DO;  NCEV=NEVLIS(NCDR(FORM),ALST);
                    IF NPROP(CFORM,FLAG,NIL)¬=NIL THEN
                    DO;  TREX=1;
                         CALL PTRACE(FNCTN,NCEV,NIL,NIL,ONE,TWO)
                    END;
                    VAL3=NAPPLY(INDIC,NCEV,ALST);
                    IF TREX=1 THEN
                    DO;  TREX=0;
                         CALL PTRACE(FNCTN,VAL3,NIL,NIL,ONE,ONE)
                    END;
                    GO TO EXEV;
               END;
/*FSUBR*/      INDIC=NGET(CFORM,FSUBR); IF INDIC¬=NIL THEN
               DO; IF NPROP(CFORM,FLAG,NIL)¬=NIL THEN TRACE1=T;
                   VAL3=NPROC(INDIC,NCDR(FORM),ALST,FNCTN);
                    GO TO EXEV; END;
/*FEXPR*/      INDIC=NGET(CFORM,FEXPR); IF INDIC¬=NIL THEN
```

```
                    DO; IF NPROP(CFORM,FLAG,NIL)¬=NIL THEN
                        DO; TRFEX=1;
                           CALL PTRACE(FNCTN,NCDR(FORM),NIL,NIL,ONE,TWO);
                        END;
                        VAL3=NAPPLY(INDIC,LIST2(NCDR(FORM),ALST),
                              ALST);
                        IF TRFEX=1 THEN
                        DO; TREX=0;
                            CALL PTRACE(FNCTN,VAL3,NIL,NIL,ONE,ONE);
                        END;
                        GO TO EXEV;
                    END;

               /* LOOK FOR FUNCTION DEFINITION ON A-LIST */

                    ALBD=NSASSOC(CFORM,ALST,ONE);
                    IF ALBD=ONE THEN
                    DO;  ERFLAG=1;
               DISPLAY('UNDEFINED FUNCTION - EVAL');
                         GO TO EXEV;
                    END;
                    VAL3=NEVAL(NCONS(NCDR(ALBD),NCDR(FORM)),ALST);
                    GO TO EXEV;
               END;
               VAL3=NAPPLY(NCAR(FORM),NEVLIS(NCDR(FORM),ALST),ALST);
          EXEV:IF ERFLAG>0 THEN VAL3=NIL;
               IF TRCE=1 THEN DO;
                    CALL PTRACE(CEVAL,VAL3,NIL,NIL,ONE,ONE);
                    TRCE=0;
                    END;
               RETURN(VAL3);
               END NEVAL;

/* EVLIS */  NEVLIS: PROC(MEV,AEV) RECURSIVE;
          /* EVALUATES ITEMS IN A LIST */
               DCL (MEV,AEV,VAL4) FIXED BIN;
               IF MEV=NIL THEN RETURN(NIL);
               RETURN(NCONS(NEVAL(NCAR(MEV),AEV),NEVLIS(NCDR(MEV),AEV)));
               END NEVLIS;

/* EVCON */  NEVCON: PROC(CEVC,AEVC) RECURSIVE;
          /* EVALUATES CONDITIONAL FORMS */
               DCL (CEVC,AEVC,VAL5) FIXED BIN;
               IF CEVC=NIL THEN
               DO;  ERFLAG=4;
```

```
           DISPLAY('CONDITIONAL UNSATISFIED - EVCON');
                GO TO EXEVC;
           END;
           VAL5=NEVAL(NCAAR(CEVC),AEVC);
           IF VAL5¬=F THEN
                IF VAL5¬=NIL THEN
                DO;  VAL5=NEVAL(NCADAR(CEVC),AEVC);
                     GO TO EXEVC;
                END;
           VAL5=NEVCON(NCDR(CEVC),AEVC);
    EXEVC: IF ERFLAG>0 THEN VAL5=NIL;
           RETURN(VAL5);
           END NEVCON;

PRNAME: PROC(JPN) CHAR(30) VARYING;
        /* RETURNS CHARACTER STRING OF PRINT NAME OF FUNCTION 'JPN'
           WHICH IS LIMITED TO 30 CHARACTERS */
           DCL (JPN,PRN1,KPN,I) FIXED BIN,
           FNCTN CHAR(30) VARYING,
           CH CHAR(1);
           PRN1=NGET(JPN,PNAME);
           FNCTN=(30)' ';
           I=1; KPN=PRN1;
           DO WHILE (KPN¬=NIL);
                UNSPEC(CH)=SUBSTR(UNSPEC(FSTOR(KPN)),9,8);
                SUBSTR(FNCTN,I,1)=CH;
                I=I+1; KPN=NCDR(KPN);
           END;
           FNCTN=SUBSTR(FNCTN,1,I-1);
           RETURN(FNCTN);
        END PRNAME;


NPROC: PROC(IX,NARGS,FINDX,FNCTN) RECURSIVE;
        /* RETURNS VALUE OF SYSTEM SUBR AND FSUBR FUNCTIONS.
           NUMBER OF SYSTEM FUNCTIONS IS LIMITED TO 255 WHICH
           IS THE MAX NUMBER REPRESENTABLE IN 8 BITS */
           DCL (IX,NARGS,FINDX,IND,KARGS,ARG(3),VAL6,JX,J)FIXED B
           DCL A(128) LABEL ;
           DCL FNCTN CHAR(30) VARYING, AR(3) LABEL ;
           IF FINDX¬=F THEN       /* FSUBR */
           DO;  IND=SUBSTR(UNSPEC(FSTOR(IX)),9,8);
                IF TRACE1=T THEN
                     CALL PTRACE(FNCTN,NARGS,NIL,NIL,ONE,TWO);
                GO TO A(IND);
```

```
          END;
      /*  SPREAD ARGUMENTS INTO STANDARD CELLS ARG(1),ARG(2),ETC   */
          KARGS=NARGS;
          J,JX=1;
    SPRD: IF KARGS=NIL THEN GO TO INDEX;
          ARG(J)=NCAR(KARGS);
          KARGS=NCDR(KARGS);        /* MAX NO. ARGS = 3*/
          J=J+1; GO TO SPRD;

          /* GET SUBR NUMBER FROM CAR OF CELL 'IX' */

  INDEX:  IND=SUBSTR(UNSPEC(FSTOR(IX)),9,8);
          IF TRACE1=T THEN DO; JX=J-1; GO TO AR(JX); END;
          ELSE GO TO A(IND);
   AR(1):  CALL PTRACE(FNCTN,ARG(1),NIL,NIL,ONE,TWO);
           GO TO A(IND);
   AR(2):  CALL PTRACE(FNCTN,ARG(1),ARG(2),NIL,TWO,TWO);
           GO TO A(IND);
   AR(3):  CALL PTRACE(FNCTN,ARG(1),ARG(2),ARG(3),THREE,TWO);
           GO TO A(IND);

/* CONS */       A(1):  VAL6=NCONS(ARG(1),ARG(2)); GO TO EXPROC;
/* CAR */        A(3):  VAL6=NCAR(ARG(1)); GO TO EXPROC;
/* CDR */        A(2):  VAL6=NCDR(ARG(1)); GO TO EXPROC;
/* EQ */         A(4):  VAL6=NEQ(ARG(1),ARG(2)); GO TO EXPROC;
/* ATOM */       A(5):  VAL6=NATOM(ARG(1)); GO TO EXPROC;
/* CAAR */       A(6):  VAL6=NCAAR(ARG(1)); GO TO EXPROC;
/* CADR */       A(7):  VAL6=NCADR(ARG(1)); GO TO EXPROC;
/* CDAR */       A(8):  VAL6=NCDAR(ARG(1)); GO TO EXPROC;
/* CDDR */       A(9):  VAL6=NCDDR(ARG(1)); GO TO EXPROC;
/* CAAAR */      A(10): VAL6=NCAAAR(ARG(1)); GO TO EXPROC;
/* CAADR */      A(11): VAL6=NCAADR(ARG(1)); GO TO EXPROC;
/* CADAR */      A(12): VAL6=NCADAR(ARG(1)); GO TO EXPROC;
/* CDAAR */      A(13): VAL6=NCDAAR(ARG(1)); GO TO EXPROC;
/* CADDR */      A(14): VAL6=NCADDR(ARG(1)); GO TO EXPROC;
/* CDADR */      A(15): VAL6=NCDADR(ARG(1)); GO TO EXPROC;
/* CDDAR */      A(16): VAL6=NCDDAR(ARG(1)); GO TO EXPROC;
/* CDDDR */      A(17): VAL6=NCDDDR(ARG(1)); GO TO EXPROC;
/* RPLACA */     A(18): VAL6=NRPLACA(ARG(1),ARG(2)); GO TO EXPROC;
/* RPLACD */     A(19): VAL6=NRPLACD(ARG(1),ARG(2)); GO TO EXPROC;
/* NULL */       A(20): VAL6=NULL(ARG(1)); GO TO EXPROC;
/* EQUAL */      A(21): VAL6=NEQUAL(ARG(1),ARG(2)); GO TO EXPROC;
/* APPEND */     A(22): VAL6=NAPPEND(ARG(1),ARG(2)); GO TO EXPROC;
/* COPY */        A(23): VAL6=NCOPY(ARG(1)); GO TO EXPROC;
/* MEMBER */      A(24): VAL6=MEMBER(ARG(1),ARG(2)); GO TO EXPROC;
/* PAIRLIS */     A(25): VAL6=NPAIRLS(ARG(1),ARG(2),ARG(3));
                            GO TO EXPROC;
/* ASSOC */       A(26): VAL6=NASSOC(ARG(1),ARG(2)); GO TO EXPROC;
/* SUB2 */        A(27): VAL6=NSUB2(ARG(1),ARG(2)); GO TO EXPROC;
/* SUBLIS */      A(28): VAL6=NSUBLIS(ARG(1),ARG(2)); GO TO EXPROC;
/* SASSOC */      A(29): VAL6=NSASSOC(ARG(1),ARG(2),ARG(3));
                            GO TO EXPROC;
/* NCONC */       A(30): VAL6=NCONC(ARG(1),ARG(2)); GO TO EXPROC;
/* ATTRIB */      A(31): VAL6=NATTRIB(ARG(1),ARG(2)); GO TO EXPROC;
/* SUBST */       A(32): VAL6=NSUBST(ARG(1),ARG(2),ARG(3));
                            GO TO EXPROC;
/* PROP */        A(33): VAL6=NPROP(ARG(1),ARG(2),ARG(3));
                             GO TO EXPROC;
/* GET */         A(34): VAL6=NGET(ARG(1),ARG(2)); GO TO EXPROC;
/* PAIR */        A(35): VAL6=NPAIR(ARG(1),ARG(2));
                            IF VAL6=NIL THEN
                            DO;  ERFLAG=5;
          DISPLAY('ARGUMENTS NOT LISTS OF EQUAL LENGTH - PAIR');
                                 END;
                            GO TO EXPROC;
/* NUMBERP */     A(36): VAL6=NUMBERP(ARG(1)); GO TO EXPROC;
/* EFFACE */      A(37): VAL6=NEFFACE(ARG(1),ARG(2)); GO TO EXPROC;
/* LENGTH */      A(38): VAL6=LENGTH(ARG(1)); GO TO EXPROC;
/* LAST */        A(39): VAL6=LAST(ARG(1)); GO TO EXPROC;
/* DEFINE */      A(40): VAL6=NDEFINE(ARG(1)); GO TO EXPROC;
/* DEFLIST */     A(41): VAL6=NDEFLST(ARG(1),ARG(2)); GO TO EXPROC;
/* CSET */        A(42): VAL6=NCSET(ARG(1),ARG(2)); GO TO EXPROC;
/* REMPROP */     A(43): VAL6=NREMPRP(ARG(1),ARG(2)); GO TO EXPROC;
/* FLAG */        A(44): VAL6=NFLAG(ARG(1),ARG(2)); GO TO EXPROC;
/* REMFLAG */     A(45): VAL6=NREMFLG(ARG(1),ARG(2)); GO TO EXPROC;
/* REVERSE */     A(46): VAL6=NREVERS(ARG(1)); GO TO EXPROC;
/* NOT */         A(47): VAL6=NOT(ARG(1)); GO TO EXPROC;
/* PRIN1 */       A(48): VAL6=NPRIN1(ARG(1)); GO TO EXPROC;
/* TERPRI */      A(49): VAL6=NTERPRI; GO TO EXPROC;
/* PRINT */       A(50): VAL6=NPRINT(ARG(1)); GO TO EXPROC;
/* READ */        A(51): VAL6=NREAD(ARG(1)); GO TO EXPROC;
/* EVAL */        A(52): VAL6=NEVAL(ARG(1),NIL); GO TO EXPROC;
/* APPLY */       A(53): VAL6=NAPPLY(ARG(1),ARG(2),NIL);
                            GO TO EXPROC;
/* TRACE */       A(54):  VAL6=NTRACE(ARG(1)); GO TO EXPROC;
/* UNTRACE */     A(55):  VAL6=NUNTRCE(ARG(1)); GO TO EXPROC;
/* ADD1 */        A(56):  VAL6=NADD1(ARG(1)); GO TO EXPROC;
/* SUB1 */        A(57):  VAL6=NSUB1(ARG(1)); GO TO EXPROC;
```

```
 /* AND */        A(58):  VAL6=NAND(NARGS,FINDX); GO TO EXPROC:
 /* OR */         A(59):  VAL6=NOR(NARGS,FINDX); GO TO EXPROC;
 /* CSETQ */      A(60):  VAL6=NCSETQ(NARGS,FINDX); GO TO EXPROC;
 /* LIST */       A(61):  VAL6=LIST(NARGS,FINDX); GO TO EXPROC;

  EXPROC:  IF TRACE1=T THEN CALL PTRACE(FNCTN,VAL6,NIL,NIL,ONE,ONE);
             TRACE1=F:
             RETURN(VAL6);


   END NPROC;

 /* LISP FUNCTIONS WHICH REQUIRE ACCESS TO CURRENT A-LIST */

 /* LIST */  LIST: PROC(LSARG,ALS);
       /* RETURNS A LIST OF ITEMS ON LIST LSARG AFTER THEY ARE
             EVALUATED */
             DCL (LSARG,ALS) FIXED BIN:
             IF NCAR(LSARG)=QUOTE THEN RETURN(NCADR(LSARG));
             RETURN(NEVLIS(LSARG,ALS));
             END LIST;

LIST2: PROC(LARG2,LARG3);
       /* RETURNS A LIST OF THE TWO ARGUMENTS */
             DCL (LARG2,LARG3);
             RETURN(NCONS(LARG2,NCONS(LARG3,NIL)));
             END LIST2;

LIST3: PROC (LARG4,LARG5,LARG6);
       /* RETURNS A LIST OF THE THREE ARGUMENTS */
             DCL (LARG4,LARG5,LARG6);
             RETURN(NCONS(LARG4,NCONS(LARG5,NCONS(LARG6,NIL))));
             END LIST3;

 /* CSET */  NCSET: PROC(SETOB,SETVAL);
       /* PUTS APVAL INDICATOR ON PROPERTY LIST OF VALUE OF SETOB
             WHICH POINTS TO VALUE OF SETVAL */
             DCL (SETOB,SETVAL,CST) FIXED BIN;
             IF NPROP(SETOB,APVAL,NIL)=NIL THEN
                  CST=NRPLACD(LAST(NCDR(SETOB)),NCONS(APVAL,
                       NCONS(NCONS(SETVAL,NIL),NIL)));
             ELSE CST=NRPLACA(NGET(SETOB,APVAL),SETVAL);
             RETURN(SETOB);
             END NCSET;
```

```
/* CSETQ */  NCSETQ: PROC(SETQOB,SETQAL);
        /* WORKS LIKE NCSET BUT THE FIRST ARGUMENT IS QUOTED */
           DCL (SETQOB,SETQVAL,SETQVAL1,SETQAL) FIXED BIN;
           SETQVAL1=NEVAL(NCADR(SETQOB),SETQAL);
           RETURN(NCSET(NCAR(SETQOB),SETQVAL1));
           END NCSETQ;

/* AND */  NAND: PROC(ANDLST,AALS);
        /* EVALUATES EACH ITEM IN THE LIST ANDLST UNTIL ONE EVALUAT
           TO F OR NIL.  RETURNS F IN THIS CASE OR T IF THE END OF
           THE LIST IS REACHED */
           DCL (ANDLST,ANDLST1,AND1,AALS) FIXED BIN;
           ANDLST1=ANDLST;
     AND:  IF ANDLST1=NIL THEN RETURN(T);
           AND1=NEVAL(NCAR(ANDLST1),AALS);
           IF AND1=F | AND1=NIL THEN RETURN(F);
           ANDLST1=NCDR(ANDLST1); GO TO AND;
           END NAND;

/* OR */  NOR: PROC(ORLST,OALS);
        /* EVALUATES EACH ITEM IN THE LIST ORLST UNTIL ONE EVALUATES
           TO SOMETHING OTHER THAN T.  RETURNS T IN THIS CASE OR F
           IF ALL ITEMS EVALUATE TO F OR NIL */
           DCL (ORLST,ORLST1,EVALOR,OALS) FIXED BIN;
           ORLST1=ORLST;
     OR:   IF ORLST1=NIL THEN RETURN(F);
           EVALOR=NEVAL(NCAR(ORLST1),OALS);
           IF EVALOR¬=F THEN
                IF EVALOR¬=NIL THEN RETURN(T);
           ORLST1=NCDR(ORLST1); GO TO OR;
           END NOR;

/* OUTPUT FUNCTIONS */

/* PRINT */ NPRINT: ENTRY(XPRIN);
        /* PRINTS THE S-EXPRESSION PASSED AS THE ARGUMENT */
           DCL (XPRIN,DP1) FIXED BIN;
           PRBUFF=(128)' ';
           BUFFCON=NBLKS;
           DP1=NPRIN(XPRIN);
           DP1=NTERPRI;
           RETURN(XPRIN);

NPRIN: PROC(KPRIN) RECURSIVE;
        /* PUTS REPRESENTATION OF S-EXPRESSION KPRIN IN PRINT BUFFER
        DCL (DUM1,DUM2,DUM3,JPRIN,KPRIN) FIXED BIN;
        IF NATOM(KPRIN)=T THEN GO TO DP;
        JPRIN=KPRIN;
        SUBSTR(PRBUFF,BUFFCON,1)='('; BUFFCON=BUFFCON+1;
   AP:  DUM2=NPRIN(NCAR(JPRIN));
        IF NCDR(JPRIN)=NIL THEN GO TO CP;
        SUBSTR(PRBUFF,BUFFCON,1)=' '; BUFFCON=BUFFCON+1;
        IF NATOM(NCDR(JPRIN))=T THEN GO TO BP;
        JPRIN=NCDR(JPRIN);
        GO TO AP;
   BP:  SUBSTR(PRBUFF,BUFFCON,1)='.'; BUFFCON=BUFFCON+1;
        SUBSTR(PRBUFF,BUFFCON,1)=' '; BUFFCON=BUFFCON+1;
        DUM3=NPRIN1(NCDR(JPRIN));
   CP:  SUBSTR(PRBUFF,BUFFCON,1)=')'; BUFFCON=BUFFCON+1;
        RETURN(KPRIN);
   DP:  DUM3=NPRIN1(KPRIN);
        RETURN(KPRIN);
        END NPRIN;

/* PRIN1 */ NPRIN1: PROC(NPRN);
   /* PUTS PRINT NAME OF ATOM 'NPRN' ON PRINT BUFFER */
        DECLARE
        TBUFF CHAR(14) STATIC,
        TBUFF1 CHAR(14) VARYING,
        TPBUFF CHAR(30) STATIC,
        LENGTH BUILTIN,
        CH CHAR(1) STATIC,
        (KPRN,NPRN,DPRN,PC,LC,LPR,LPR1) FIXED BIN ;
        IF NUMBERP(NPRN)=T THEN    /* IS ATOM A NUMBER? */
        DO;  TBUFF=(14)' ';
             TBUFF=FSTOR(NCDR(NPRN));
             PC=1;
             DO WHILE (SUBSTR(TBUFF,PC,1)=' '); PC=PC+1; END;
             TBUFF1=SUBSTR(TBUFF,PC,15-PC);
             LC=LENGTH(TBUFF1);
             SUBSTR(PRBUFF,BUFFCON,LC)=TBUFF1;
             BUFFCON=BUFFCON+LC;
             RETURN(NPRN);
        END;
        LPR=1; TPBUFF=(30)' ';
        KPRN=NGET(NPRN,APVAL);    /* CHARACTER OBJECT? */
        IF NCAR(KPRN)<128 THEN GO TO NPRA;
        KPRN=NGET(NPRN,PNAME);
   NPRA:DO WHILE (KPRN¬=NIL);
             UNSPEC(CH)=SUBSTR(UNSPEC(FSTOR(KPRN)),9,8);
```

```
                  SUBSTR(TPBUFF,LPR,1)=CH;
                  KPRN=NCDR(KPRN);
                  LPR=LPR+1;
              END;
              LPR1=LPR-1;
              IF (LPR1+BUFFCON)>120 THEN DPRN=NTERPRI;
              SUBSTR(PRBUFF,BUFFCON,LPR1)=SUBSTR(TPBUFF,1,LPR1);
              BUFFCON=BUFFCON+LPR1;
              RETURN(NPRN);
              END NPRIN1;

 /* TERPRI */  NTERPRI: PROC;
         /* PRINTS CURRENT CONTENTS OF PRINT BUFFER */
              DISPLAY(PRBUFF);
              PRBUFF=(128)' ';
              BUFFCON=NBLKS;
              RETURN(NIL);
              END NTERPRI;

PTRACE: PROC(FNAME,ARG1,ARG2,ARG3,NGS,IO);
         /* DISPLAYS TRACE INFO ON TERMINAL WHEN FUNCTIONS ARE
              ENTERED AND EXITED - WILL NOW WORK ONLY FOR FUNCTIONS WI
              LESS THAN FOUR ARGUMENTS   */
           DCL (ARG1,ARG2,ARG3,NGS,IO,LIM,LF) FIXED BIN,
              LENGTH BUILTIN,
              FNAME CHAR(30) VARYING;
              LF=LENGTH(FNAME);
              IF IO=1 THEN GO TO LEAVE;
              IF NBLKS>80 THEN DO; RECUR=RECUR+1; NBLKS=1; END;
              SUBSTR(PRBUFF,NBLKS,12+LF)='CALL '||FNAME||', ARGS:';
              LIM=NTERPRI;
              LIM=NPRINT(ARG1);
              IF NGS=1 THEN GO TO EXPTR;
              LIM=NPRINT(ARG2);
              IF NGS=2 THEN GO TO EXPTR;
              LIM=NPRINT(ARG3);
      EXPTR: NBLKS=NBLKS+4;
              RETURN;
       LEAVE: NBLKS=NBLKS-4;
              IF NBLKS<1 THEN DO;
                  IF RECUR>0 THEN DO;
                      RECUR=RECUR-1; NBLKS=80; END;
                  ELSE NBLKS=1;
                  END;
              SUBSTR(PRBUFF,NBLKS,13+LF)='VALUE OF '||FNAME||' IS:';
```

```
            LIM=NTERPRI;
            LIM=NPRINT(ARG1);
            RETURN;
       END PTRACE;

   END NEVALQ;
```

```
/* LISP2: INITIALIZATION ROUTINE */

 INITIAL: PROC:

        DECLARE
        ACCUM CHAR(30) EXT,
        ALPBASE BIN FIXED EXT,
        APPLY BIN FIXED EXT,
        APVAL BIN FIXED EXT,
        BLANK BIN FIXED EXT,
        BP BIN FIXED EXT,
        BNUM BIN FIXED EXT,
        BUFFER CHAR(72) EXT,
        BUFFNO BIN FIXED EXT,
        CADD BIN FIXED EXT,
        CHARSET CHAR(53) EXT,
        COMMA BIN FIXED EXT,
        COND BIN FIXED EXT,
        COUNT BIN FIXED EXT,
        CTR BIN FIXED EXT,
        DASH BIN FIXED EXT,
        DIGBASE BIN FIXED EXT,
        DOLLAR BIN FIXED EXT,
        EFSTOR BIN FIXED EXT,
        EXPR BIN FIXED EXT,
        EQSIGN BIN FIXED EXT,
        EVAL BIN FIXED EXT,
        F BIN FIXED EXT,
        FEXPR BIN FIXED EXT,
        FNCTION BIN FIXED EXT,
        FREE BIN FIXED EXT,
        FSUBR BIN FIXED EXT,
        FLAG BIN FIXED EXT,
        FSTOR(0:16000) BIN FIXED(31) EXT,
        FUNARG BIN FIXED EXT,
        JJ BIN FIXED EXT,
        JK BIN FIXED,
        LABEL BIN FIXED EXT,
        LAMBDA BIN FIXED EXT,
        LCOUNT BIN FIXED EXT,
        LINE BIN FIXED EXT,
        LPAR BIN FIXED EXT,
        MFSTOR BIN FIXED EXT,
        MODETAG BIT(1) EXT,
        NBASE BIN FIXED EXT,
   NIL BIN FIXED EXT,
   NREAD ENTRY EXT,
   PERIOD BIN FIXED EXT,
   PLUSS BIN FIXED EXT,
   PNAME BIN FIXED EXT,
   QUOTE BIN FIXED EXT,
   RETAG BIT(1) EXT,
   RE1TAG BIT(1) EXT,
   RPAR BIN FIXED EXT,
   SLASH BIN FIXED EXT,
   STAC(200) BIN FIXED EXT,
   STAR BIN FIXED EXT,
   STKTAB BIT(1) EXT,
   SUBR BIN FIXED EXT,
   T BIN FIXED EXT,
   TCOUNT BIN FIXED EXT,
   VAR CHAR(10) VARYING,
   VAR1 CHAR(1),
   VAR2 BIN FIXED EXT;

   DISPLAY(' ');
   DISPLAY('NPS LISP 1.5 MOD 1 INITIALIZING');
   DISPLAY(' '); RETAG,RE1TAG='0'B;
   MFSTOR=16000; NBASE=MFSTOR-300; EFSTOR=NBASE-1;
   DO I=1 TO 127; FSTOR(I)=0; END;  /* ZERO OUT OBLIST */
   DO I=128 TO MFSTOR-1; FSTOR(I)=I+1; END;  /* LINK UP */
   FREE=128; FSTOR(MFSTOR)=0; FSTOR(EFSTOR)=0;
   BNUM=MFSTCR;
   DO I=1 TO 200; STAC(I)=0; END;
   APPLY=1030; EVAL=1018;
CHARSET=' .$&*()-+="'':;?,/ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
   ALPBASE=INDEX(CHARSET,'A')-1;
   DIGBASE=INDEX(CHARSET,'0')-1;
   LINE,LCOUNT=1;
   MODETAG,STKTAB='0'B;
   TCOUNT,BUFFNO=0;
   NIL=142; PNAME=135; APVAL=154;  /* BOOTSTRAP VALUES */
   BUFFER='PNAME NIL APVAL EXPR SUBR FEXPR FSUBR COND QUOTE &';
   JK=NREAD;
   BUFFER='SLASH PLUSS DASH STAR BLANK EQSIGN COMMA PERIOD &';
   JK=NREAD;
   BUFFER='DOLLAR RPAR LPAR &';
   JK=NREAD;
   BUFFER='FUNCTION FUNARG LAMBDA LABEL F T FLAG &';
   JK=NREAD; LINE=1;
```

```
       BUFFER='CONS CDR CAR EQ ATOM CAAR CADR CDAR CDDR CAAAR CAADR
        JK=NREAD;
        BUFFER='CADAR CDAAR CADDR CDADR CDDAR CDDDR RPLACA RPLACD &
        JK=NREAD;
        BUFFER='NULL EQUAL APPEND COPY MEMBER PAIRLIS ASSOC SUB2 &'
        JK=NREAD;
        BUFFER='SUBLIS SASSOC NCONC ATTRIB SUBST PROP GET PAIR &';
        JK=NREAD;
        BUFFER='NUMBERP EFFACE LENGTH LAST DEFINE DEFLIST CSET &';
        JK=NREAD;
        BUFFER='REMPROP FLAG REMFLAG REVERSE NOT &';
        JK=NREAD;
        BUFFER='PRIN1 TERPRI PRINT READ EVAL APPLY &';
        JK=NREAD;
        BUFFER='TRACE UNTRACE ADD1 SUB1 &';
        JK=NREAD;
        BUFFER='AND OR CSETQ LIST &'; JK=NREAD;
        STKTAB='1'B;
        RETURN;

 INIT1: ENTRY(I1);
        DECLARE
        SWT(10:20) LABEL(A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11)
                INITIAL(A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11);

        GOTO SWT(I1);
        A1: VAR1='/'; GOTO A12;  A2: VAR1='+'; GOTO A12;
        A3: VAR1='-'; GOTO A12;  A4: VAR1='*'; GOTO A12;
        A5: VAR1=' '; GOTO A12;  A6: VAR1='='; GOTO A12;
        A7: VAR1=','; GOTO A12;  A8: VAR1='.'; GOTO A12;
        A9: VAR1='$'; GOTO A12;  A10: VAR1=')'; GOTO A12;
        A11: VAR1='(';
   A12: L=UNSPEC(VAR1); VAR2=NCONS(L,NIL); JJ=-1; RETURN;

  INITL: ENTRY(I2);
 /* SETS UP APVALS AND PROGRAM VARIABLE VALUES */
        DECLARE LB(27) LABEL;
        GOTO LB(I2);
        LB(1):  PNAME=CADD; RETURN;
        LB(2):  NIL=CADD; VAR2=NCONS(NIL,NIL); CALL SETAP(NIL);
                  RETURN;
        LB(3):  APVAL=CADD; RETURN;
        LB(4):  EXPR=CADD; RETURN;
        LB(5):  SUBR=CADD; RETURN;
        LB(6):  FEXPR=CADD; RETURN;
```

```
      LB(7):  FSUBR=CADD; RETURN;
      LB(8):  COND=CADD; RETURN;
      LB(9):  QUOTE=CADD; RETURN;
      LB(10): SLASH=CADD; CALL SETAP(SLASH); RETURN;
      LB(11): PLUSS=CADD; CALL SETAP(PLUSS); RETURN;
      LB(12): DASH=CADD; CALL SETAP(DASH); RETURN;
      LB(13): STAR=CADD; CALL SETAP(STAR); RETURN;
      LB(14): BLANK=CADD; CALL SETAP(BLANK); RETURN;
      LB(15): EQSIGN=CADD; CALL SETAP(EQSIGN); RETURN;
      LB(16): COMMA=CADD; CALL SETAP(COMMA); RETURN;
      LB(17): PERIOD=CADD; CALL SETAP(PERIOD); RETURN;
      LB(18): DOLLAR=CADD; CALL SETAP(DOLLAR); RETURN;
      LB(19): RPAR=CADD; CALL SETAP(RPAR); RETURN;
      LB(20): LPAR=CADD; CALL SETAP(LPAR); RETURN;
      LB(21): FNCTION=CADD; RETURN;
      LB(22): FUNARG=CADD; RETURN;
      LB(23): LAMBDA=CADD; RETURN;
      LB(24): LABEL=CADD; RETURN;
      LB(25): F=CADD; M=NRPLACD(NCDDR(F),NCONS(APVAL,
                 NCONS(NCONS(NIL,NIL),NIL))); RETURN;
      LB(26): T=CADD; VAR2=NCONS(T,NIL); CALL SETAP(T); RETURN;
      LB(27): FLAG=CADD; RETURN;
      RETURN;

 SETAP: ENTRY(SAPVL);
      DCL SAPVL BIN FIXED;
      M=NRPLACD(NCDDR(SAPVL),NCONS(APVAL,NCONS(VAR2,NIL)));
      RETURN;

 INITL2: ENTRY(I3);
/* SET UP SUBR INDICATORS */
      LM=NCDDR(CADD);
      M=NRPLACD(LM,NCONS(SUBR,NCONS(NCONS(I3,NIL),NIL)));
      RETURN;

 INITL3: ENTRY(I4);
/* SET UP FSUBR INDICATORS */
      LM=NCDDR(CADD);
      M=NRPLACD(LM,NCONS(FSUBR,NCONS(NCONS(I4,NIL),NIL)));
      RETURN;

 END INITIAL;
```

```
LISPF: PROC;
/* LISPF: READER, DUMP, LOAD, AND GARBAGE COLLECTER */
 READER: ENTRY(VAR,INPT) BIT(1);
        DECLARE
        ASCTR BIN FIXED EXT,
        ASNMBR BIN FIXED EXT,
        ASTAG BIT(1) EXT,
        BNUM BIN FIXED EXT,
        CARDNO1 BIN FIXED(31,0) EXT,
        CARDNO2 BIN FIXED(31,0) STATIC,
        CB32 BIT(32),
        DB32 BIT(32) STATIC,
        DTAG BIT(1) EXT,
        EFSTOR BIN FIXED EXT,
        ESTART BIN FIXED(31) EXT,
        FREE BIN FIXED EXT,
        1 FCB EXT,
             2 COMMAND CHAR(8),
             2 FILENAME CHAR(8),
             2 FILETYPE CHAR(8),
             2 CARD_NUMBER BIN FIXED(31,0),
             2 STATUS BIN FIXED(31,0),
             2 CARD_BUFFER CHAR(80),
        FSTOR(0:16000) BIN FIXED(31) EXT,
        FCTR BIN FIXED STATIC,
        INPT CHAR(72),
        LC BIN FIXED STATIC,
        LCOUNT BIN FIXED EXT,
        NBASE BIN FIXED EXT,
        P1 BIN FIXED STATIC,
        P2 BIN FIXED STATIC,
        PPTR BIN FIXED STATIC,
        PTR BIN FIXED(31) STATIC,
        RTAG BIT(1) EXT,
        STOR ENTRY RETURNS(BIT(1)),
        TCOUNT BIN FIXED EXT,
        TEMPBUFF CHAR(80),
        TEST ENTRY RETURNS(BIT(1)),
        TTAG BIT(1) STATIC,
        VAR CHAR(25) VARYING,
        VAR2 CHAR(2),
        VAR4 CHAR(4);
```

```
          TTAG='0'B;
          IF TCOUNT=2 THEN GOTO R1;
          INPT=' ';
          IF ASTAG THEN IF ASCTR=ASNMBR THEN
          DO; DISPLAY(VAR) REPLY(INPT); CALL DUSAVE; CALL TSAVE;
          GOTO R2;
          END;
     R1:  DISPLAY(VAR) REPLY(INPT);
     R2:  IF INPT='END LISP' THEN RETURN('0'B);
              ELSE RETURN('1'B);

DUSAVE: ENTRY;

               IF ¬DTAG THEN DO; FILENAME='AUTOLISP'; COMMAND='ERASE';
                             CALL IHEFILE(FCB);
                             END;
                 ELSE FILENAME='DUMPLISP'; FILETYPE='DATA';
               CARD_BUFFER=''; ASCTR=0; LC=1;
               COMMAND='WRBUF'; PTR,PPTR=5; CARDNO2=1; P1=1;
           IF: IF FSTOR(LC+1)=FSTOR(LC)+1 &
                  FSTOR(LC+2)=FSTOR(LC+1)+1 THEN
                    DO; UNSPEC(VAR2)=SUBSTR(UNSPEC(PPTR),17,16);
                    CARD_NUMBER=DIVIDE(P1,80,31,0)+1;
                    IF CARD_NUMBER¬=CARDNO2-1 THEN
                         DO; TEMPBUFF=CARD_BUFFER; TTAG='1'B; END;
                    CALL CLOSE; CALL OPEN;
                    COMMAND='RDBUF'; CALL IHEFILE(FCB);
                    CALL CLOSE;
                    I=CARD_NUMBER*80;
                    P1=80-(I-P1); COMMAND='WRBUF';
                    SUBSTR(CARD_BUFFER,P1,2)=VAR2; P1=PPTR; FCTR=0;
                    CALL IHEFILE(FCB);
                    IF TTAG THEN CARD_BUFFER=TEMPBUFF;
                    P2=LC; TTAG='0'B;
                      DO WHILE(FSTOR(LC+1)=FSTOR(LC)+1 & LC<15999);
                      LC=LC+1; FCTR=FCTR+1;
                      END;
                    LC=LC+1; UNSPEC(VAR2)=SUBSTR(UNSPEC(FCTR),17,16);
                    SUBSTR(CARD_BUFFER,PTR+2,2)=VAR2; PTR=PTR+4;
                    PPTR=PPTR+4; IF ¬TEST THEN GOTO R3;
                    UNSPEC(VAR4)=UNSPEC(FSTOR(P2));
                    SUBSTR(CARD_BUFFER,PTR,4)=VAR4; PTR=PTR+4;
                    PPTR=PPTR+4; IF ¬TEST THEN GOTO R3;
                    END;
```

```
             IF LC>16000 THEN IF ¬STOR THEN GOTO R3;
             UNSPEC(VAR4)=UNSPEC(FSTOR(LC));
             SUBSTR(CARD_BUFFER,PTR,4)=VAR4; PTR=PTR+4; PPTR=PPTR+4;
             LC=LC+1; IF ¬TEST THEN GOTO R3; GOTO IF;
         R3: COMMAND='FINUFD'; CALL IHEFILE(FCB); I=1; RTAG='1'B:
             CALL TEXTWRK(I); UNSPEC(VAR4)=UNSPEC(ESTART);
             SUBSTR(CARD_BUFFER,10,4)=VAR4; RTAG='0'B;
             UNSPEC(VAR4)=UNSPEC(FREE); SUBSTR(CARD_BUFFER,20,4)=VAR
            UNSPEC(VAR4)=UNSPEC(EFSTOR); SUBSTR(CARD_BUFFER,14,4)=VA
             UNSPEC(VAR4)=UNSPEC(BNUM); SUBSTR(CARD_BUFFER,30,4)=VAR
             UNSPEC(VAR4)=UNSPEC(LCOUNT); SUBSTR(CARD_BUFFER,2,4)=VA
            UNSPEC(VAR4)=UNSPEC(CARDNO1); SUBSTR(CARD_BUFFER,6,4)=VA
             COMMAND='WRBUF'; CALL IHEFILE(FCB); COMMAND='FINUFD';
             CALL IHEFILE(FCB); RETURN;

  STOR: ENTRY BIT(1);

        CARD_NUMBER=CARDNO2; CALL IHEFILE(FCB); PTR=1;
        CARDNO2=CARDNC2+1;
        CARD_BUFFER=''; IF LC>16000 THEN RETURN('0'B);
             ELSE RETURN('1'B);

  TEST: ENTRY BIT(1);

        IF PTR=81 THEN
           IF ¬STOR THEN RETURN('0'B);
           ELSE RETURN('1'B);
        RETURN('1'B);

LOADER: ENTRY;

        COMMAND='RDBUF'; CARD_NUMBER=1; CALL IHEFILE(FCB);
        IF STATUS=1 THEN DO; DISPLAY('FILE NOT FOUND'); RETURN; END;
        VAR2=SUBSTR(CARD_BUFFER,1,2); DB32=(32)'0'B;
        SUBSTR(DB32,17,16)=UNSPEC(VAR2); PPTR=DB32;
        P1,PTR=5; P2=1;
    L1: DO WHILE(PTR<PPTR);
        VAR4=SUBSTR(CARD_BUFFER,P1,4); DB32=UNSPEC(VAR4);
        FSTOR(P2)=DB32;
        P2=P2+1; P1=P1+4; PTR=PTR+4; IF P2=16001 THEN RETURN;
        IF P1=81 THEN DO; P1=1; CALL RDR; END;
        END;
        VAR2=SUBSTR(CARD_BUFFER,P1,2); DB32=(32)'0'B;
        SUBSTR(DB32,17,16)=UNSPEC(VAR2); PPTR=DB32;
        P1=P1+2; VAR2=SUBSTR(CARD_BUFFER,P1,2);
```

```
       DB32=(32)'0'B; SUBSTR(DB32,17,16)=UNSPEC(VAR2);
       FCTR=DB32; LC=1; P1=P1+2; PTR=PTR+4;
       IF P1=81 THEN DO; P1=1; CALL RDR; END;
       VAR4=SUBSTR(CARD_BUFFER,P1,4);
       UNSPEC(FSTOR(P2))=UNSPEC(VAR4); PTR=PTR+4; P1=P1+4; P2=P2+1;
       IF P1=81 THEN DO; P1=1; CALL RDR; END;
       DO WHILE(LC<=FCTR);
       FSTOR(P2)=FSTOR(P2-1)+1; P2=P2+1;
       IF P2=16001 THEN RETURN; LC=LC+1;
       END;
       GOTO L1;

  RDR: ENTRY;

       CARD_NUMBER=CARD_NUMBER+1; CALL IHEFILE(FCB);
       RETURN;

COLLECT: ENTRY;

/* THIS IS THE GARBAGE COLLECTER */

       DECLARE
       APVAL BIN FIXED EXT,
       CTR BIN FIXED EXT,
       EXPR BIN FIXED EXT,
       FEXPR BIN FIXED EXT,
       FSUBR BIN FIXED EXT,
       I1 BIN FIXED STATIC,
       II BIN FIXED STATIC,
       JJ BIN FIXED STATIC,
       JOIN ENTRY(BIN FIXED),
       L4 BIN FIXED STATIC,
       LL BIN FIXED STATIC,
       MARK ENTRY(BIN FIXED),
       MEXPR ENTRY(BIN FIXED) RETURNS(BIN FIXED),
       MFSTOR BIN FIXED EXT,
       NUNSTK ENTRY EXT,
       NIL BIN FIXED EXT,
       PNAME BIN FIXED EXT,
       PNME ENTRY(BIN FIXED) RETURNS(BIN FIXED),
       SUBR BIN FIXED EXT;

       DB32='00000000000000001000000000000000'B;
       II=1;  /* II: OBLIST NUMBER */
   C1: IF NCAR(II)=0 THEN
```

```
              IF II=127 THEN GOTO C2;
            ELSE DO; II=II+1; GOTO C1; END;
       IF II=128 THEN GOTO C2;
       L4,I1=NCDR(II);
       DO WHILE(I1¬=NIL);
            JJ=NCAR(I1);  /* JJ: ATOM HEADER CELL */
            LL=JJ; JJ=NCDR(JJ); CALL MARK(LL); LL=JJ;
            IF NCAR(JJ)=PNAME THEN JJ=PNME(JJ); ELSE
            DO; JJ=NCDR(JJ); CALL MARK(LL); LL=JJ; JJ=PNME(JJ); END
            CALL MARK(LL); LL=JJ;
       C3: JJ=NCDR(JJ); CALL MARK(LL); IF JJ=NIL THEN GOTO INC; LL=
            IF NCAR(JJ)=APVAL | NCAR(JJ)=SUBR |
               NCAR(JJ)=FSUBR THEN
               DO; JJ=PNME(JJ); CALL MARK(LL); LL=JJ; GOTO C3; END;
            IF NCAR(JJ)=EXPR | NCAR(JJ)=FEXPR THEN JJ=MEXPR(JJ);
            CALL MARK(LL); LL=JJ; GOTO C3;
       INC: I1=NCDR(I1); CALL MARK(L4); L4=I1;
       END;
       II=II+1; GOTO C1;
   C2: IF ESTART=0 THEN GOTO C4; LL,II=ESTART;
       DO WHILE(II¬=NIL);
       II=NCDR(II); CALL MARK(LL); LL=II;
       END;
   C4: LL,II=FREE;
       DO WHILE(FSTOR(II)¬=0);
       II=FSTOR(II); CALL MARK(LL); LL=II;
       END;
       CALL MARK(LL);
       DO WHILE(BNUM<=MFSTOR);
       FSTOR(BNUM)=BNUM+1; BNUM=BNUM+1;
       END;
       BNUM=BNUM-1; II=EFSTOR;
       DO I=128 TO NBASE-1;
       CB32=UNSPEC(FSTOR(I)) & DB32; IF CB32=(32)'0'B THEN
            CALL JOIN(I);
          ELSE DO; CB32=UNSPEC(FSTOR(I)) &
               '11111111111111110111111111111111'B; FSTOR(I)=CB32;
               END;
       END;
       EFSTOR=II; FSTOR(II)=0;
       RETURN;

MEXPR: ENTRY(M1) BIN FIXED;

       DCL F BIN FIXED EXT;
```

```
        M1=NCDR(M1); M2=NCAR(M1);
  ME1:  M=NSTACK(M2); M2=NCAR(M2);
        DO WHILE(NATOM(M2)=F);
                M=NSTACK(M2); M2=NCAR(M2);
        END;
  ME2:  L,M2=NUNSTK; M2=NCDR(M2); CALL MARK(L); L=M2;
  ME3:  IF M2=NIL THEN
                IF CTR=200 THEN RETURN(M1);
                ELSE DO; L,M2=NUNSTK; M2=NCDR(M2); CALL MARK(L); L=M2;
                        GOTO ME3;
                        END;
        GOTO ME1;

MARK:   ENTRY(J5);

        CB32=UNSPEC(FSTOR(J5)) | DB32; FSTOR(J5)=CB32;
        RETURN;

PNME:   ENTRY(J6) BIN FIXED;

        J6=NCDR(J6); L2,J7=NCAR(J6); J7=NCDR(J7); CALL MARK(L2); L2=J7;
        DO WHILE(J7¬=NIL);
        J7=NCDR(J7); CALL MARK(L2); L2=J7;
        END;
        RETURN(J6);

JOIN:   ENTRY(J8);

        FSTOR(II)=J8; IF II=EFSTOR THEN CALL MARK(II); II=J8;
        RETURN;

END LISPF;
```

```
      E2C: J=J+1;
      END;
      I=I+1; IF MOD(I,20)=0 THEN
           DO; DISPLAY('--CONTINUE?') REPLY(VARA);
           IF SUBSTR(VARA,1,1)='Y' THEN GOTO E2A; ELSE GOTO EA;
           END;
      GOTO E2A;
      END;

  E3: DO;
      DECLARE
      BP1 BIN FIXED STATIC,
      COMPARE ENTRY RETURNS(BIT(1)),
      CHEK ENTRY RETURNS(BIT(1)),
      CTR3 BIN FIXED STATIC,
      LE5 BIN FIXED STATIC,
      LEN BIN FIXED STATIC,
      LEN1 BIN FIXED STATIC,
      PTR1 BIN FIXED STATIC,
      PTR2 BIN FIXED STATIC,
      SCNA ENTRY RETURNS(BIT(1)),
      VARD CHAR(35) VARYING;

      VARA=''; CALL SCAN; FILENAME='LISPTEXT'; CALL CLOSE;
      IF TCOUNT=0 | SUBSTR(ACCUM,1,1)¬='/' THEN GOTO ER3;
      PTR2= BP+1; IF ¬SCNA THEN GOTO EA; LEN=CTR3;
      BP1=BP; BP=PTR2; RTAG='1'B; LE5=LE1;
 E3A: CALL TEXTWRK(LE5);
      LE5=LE5+1;
      IF ¬COMPARE THEN
           IF LE5>IE1 THEN DO; DISPLAY('*E6A: FIELD NOT FOUND*');
                           GOTO EA; END;
           ELSE GOTO E3A;
      PTR2=BP+1; IF ¬SCNA THEN GOTO EA; LEN1=CTR3;
      IF LEN1>LEN THEN   /* NEW FIELD LONGER THEN OLD */
         DO; IF ¬CHEK THEN DISPLAY('**E6B: TRUNCATED**');
           I=PTR1+LEN1; J=80-I+1; K=PTR1+LEN;
           SUBSTR(CARD_BUFFER,I,J)=SUBSTR(CARD_BUFFER,K,J);
           SUBSTR(CARD_BUFFER,PTR1,LEN1)=SUBSTR(BUFFER,BP+1,LEN1);
         END;
      ELSE IF LEN1¬=LEN THEN   /* NEW FIELD SHORTER */
         DO; I=PTR1+LEN1; K=PTR1+LEN; J=80-K+1; L=80-LEN+LEN1;
           SUBSTR(CARD_BUFFER,I,J)=SUBSTR(CARD_BUFFER,K,J);
           SUBSTR(CARD_BUFFER,PTR1,LEN1)=SUBSTR(BUFFER,BP+1,LEN1);
           SUBSTR(CARD_BUFFER,L+1,LEN-LEN1)=' ';
 E2: DO;

     DECLARE
     APVAL BIN FIXED EXT,
     CH CHAR(1),
     CHR CHAR(5) VARYING,
     EXPR BIN FIXED EXT,
     FEXPR BIN FIXED EXT,
     FSUBR BIN FIXED EXT,
     KK BIN FIXED(31),
     PL(3) LABEL,
     PNAME BIN FIXED EXT,
     SUBR BIN FIXED EXT;

     CALL SCAN; IF TCOUNT=0 THEN
          DO; DISPLAY('**E5A: ''TYPE'' PARAMETER MISSING**');
          GOTO EA;
          END;
     IF SUBSTR(ACCUM,1,1)='A' THEN DO; KJ=1; GOTO E2B; END;
     IF SUBSTR(ACCUM,1,1)='F' THEN DO; KJ=2; GOTO E2B; END;
     IF SUBSTR(ACCUM,1,1)='S' THEN DO; KJ=3; GOTO E2B; END;
     DISPLAY('**E5B: ''TYPE'' PARAMETER NOT RECOGNIZED**'); GOTO EA;
E2B: I=1;
E2A: IF I=128 THEN GOTO EA; JA=NCAR(I);
     IF JA=0 THEN DO; I=I+1; GOTO E2A; END;
     KK=I; J=1;
     DO WHILE(J<=JA);
     KK=NCDR(KK); L=NCAR(KK); GOTO PL(KJ);
     PL(1): IF NGET(L,SUBR)¬=NIL | NGET(L,EXPR)¬=NIL |
          NGET(L,FSUBR)¬=NIL | NGET(L,FEXPR)¬=NIL
          THEN GOTO E2C; IF NGET(L,APVAL)=NIL THEN CHR=' ';
               ELSE CHR='APVAL'; GOTO E2D;
     PL(2): IF NGET(L,SUBR)=NIL & NGET(L,EXPR)=NIL THEN GOTO E2C;
          IF NGET(L,SUBR)=NIL THEN CHR='EXPR';
               ELSE CHR='SUBR'; GOTO E2D;
     PL(3): IF NGET(L,FSUBR)=NIL & NGET(L,FEXPR)=NIL THEN GOTO E2C;
          IF NGET(L,FSUBR)=NIL THEN CHR='FEXPR';
               ELSE CHR='FSUBR';
     E2D: L=NGET(L,PNAME); M=1; ACCUM=' ';
     DO WHILE(L¬=NIL);
          UNSPEC(CH)=SUBSTR(UNSPEC(FSTOR(L)),9,8);
          SUBSTR(ACCUM,M,1)=CH; M=M+1; L=NCDR(L);
          END;
     DISPLAY(ACCUM||CHR);
```

```
      IF SUBSTR(ACCUM,1,3)='RET' THEN RETURN;
      IF SUBSTR(ACCUM,1,1)='R' THEN GOTO E5;
      IF SUBSTR(ACCUM,1,COUNT)='LIST' THEN GOTO E2;
      IF SUBSTR(ACCUM,1,1)='D' THEN GOTO E4;
      DISPLAY('**E1: EDIT CALL NOT VALID. TRY AGAIN**'); GOTO EA;
 E1:  DO; IF ESTART=0 THEN
        DO; DISPLAY('**E2: NO S-EXPRESSION ENTERED**'); GOTO EA; E
      CALL SCAN; IF TCOUNT=0 THEN GOTO EA; IE=SUBSTR(ACCUM,1,COUNT
      JE=1; IF RETAG THEN GOTO E1C;
      CALL SCAN;
      IF TCOUNT¬=0 THEN
        IF SUBSTR(ACCUM,1,COUNT)¬=',' THEN GOTO ER1;
        ELSE DO; CALL SCAN; IF TCOUNT=0 THEN GOTO ER1;
             JE=SUBSTR(ACCUM,1,COUNT);
             END;
E1C:  FILENAME='LISPTEXT'; CALL CLOSE;
E1B:  LN=ESTART; JE=IE+(JE-1);
      DO WHILE(IE<=JE);
         DO WHILE(NCAR(LN)¬=IE);
         IF LN=NIL THEN GOTO ER2; LN=NCDDR(LN);
         END;
      LE=NCADR(LN);  /* LE=LINE NUMBER IN LISPTEXT */
      RTAG='1'B; CALL TEXTWRK(LE); IE1,LE1=LE;
      IF RETAG THEN
           DO; BUFFER=SUBSTR(CARD_BUFFER,9,72); RE1TAG='1'B;
           BP=72; INTAG='1'B;
           LE1=NREAD; IE1=NREAD; LN=NEVALQ(LE1,IE1); LN=NPRINT(LN)
           RE1TAG='0'B; TCOUNT=2; GOTO E5A;
           END;
      DISPLAY(CARD_BUFFER); LE=LE+1;
      IF LE=CARDNOI THEN GOTO E1A; IE1=IE1+1;
      CALL GBUFF;
         DO WHILE(SUBSTR(CARD_BUFFER,1,1)=' ' & STATUS=0);
         DISPLAY(CARD_BUFFER);
         IF LE=CARDNOI THEN GOTO E1A; IE1=IE1+1;
         CALL GBUFF;
         END;
      IE=IE+1; LN=ESTART;
      END;
E1A:  RTAG='0'B; GOTO EA;
ER2:  DISPLAY('**E3: S-EXPRESSION NUMBER NOT FOUND**'); GOTO EA;
ER1:  DO; DISPLAY('**E3A: FORMAT ERROR; PRINTING S-EXP NUMBER'||IE
      GOTO E1B; END;
ER3:  DISPLAY('**E4: ILLEGAL EDIT COMMAND**'); GOTO EA;
      END;
```

```
LISPG: PROC;
/* THIS IS THE LISP EDITOR */
 EDITOR: ENTRY;
         DECLARE
         ACCUM CHAR(30) EXT,
         BP BIN FIXED EXT,
         BUFFER CHAR(72) EXT,
         CARDNO1 BIN FIXED(31,0) EXT,
         COUNT BIN FIXED EXT,
         ENDTAG BIT(1) EXT,
         ERSTAG BIT(1) EXT,
         ESTART BIN FIXED(31) EXT,
         FSTOR(0:16000) BIN FIXED(31) EXT,
         1 FCB EXT,
              2 COMMAND CHAR(8),
              2 FILENAME CHAR(8),
              2 FILETYPE CHAR(8),
              2 CARD_NUMBER BIN FIXED(31,0),
              2 STATUS BIN FIXED(31,0),
              2 CARD_BUFFER CHAR(80),
         IE BIN FIXED STATIC,
         IE1 BIN FIXED STATIC,
         INTAG BIT(1) EXT,
         JE BIN FIXED STATIC,
         LE BIN FIXED STATIC,
         LE1 BIN FIXED STATIC,
         LN BIN FIXED STATIC,
         NIL BIN FIXED EXT,
         NREAD ENTRY EXT,
         READER ENTRY(CHAR(25) VARYING,CHAR(72)) RETURNS(BIT(1)),
         RETAG BIT(1) EXT,
         RE1TAG BIT(1) EXT,
         RTAG BIT(1) EXT,
         TCOUNT BIN FIXED EXT,
         VARA CHAR(25) VARYING;

         DISPLAY(' '); DISPLAY('EDIT LISP');
     EA: VARA=' '; IF ¬READER(VARA,BUFFER) THEN
              DO; ENDTAG='1'B; RETURN; END;
         TCOUNT=2; BP=0; CALL SCAN;
         IF SUBSTR(ACCUM,1,1)='C' THEN GOTO E3;
         IF SUBSTR(ACCUM,1,1)='P' THEN GOTO E1;
```

```
          END;
          ELSE  SUBSTR(CARD_BUFFER,PTR1,LEN)=SUBSTR(BUFFER,BP+1,L
       DISPLAY(CARD_BUFFER); LE5=LE5-1; RTAG='0'B; COMMAND='WRBU
       CARD_NUMBER=LE5; CALL IHEFILE(FCB);
       GOTO EA;
       END;

SCNA: ENTRY BIT(1);

       CTR3=0;
       DO WHILE(SUBSTR(BUFFER,PTR2,1)¬='/');
       PTR2=PTR2+1; CTR3=CTR3+1; IF CTR3=36 THEN
            DO; DISPLAY('**FIELD TOO LONG**'); RETURN('0'B); END;
       END;
       RETURN('1'B);

COMPARE: ENTRY BIT(1);

       PTR1=2; VARD=SUBSTR(BUFFER,BP1+1,LEN);
       DO WHILE(VARD¬=SUBSTR(CARD_BUFFER,PTR1,LEN) & PTR1+LEN<81
       PTR1=PTR1+1;
       END;
       IF VARD=SUBSTR(CARD_BUFFER,PTR1,LEN) THEN RETURN('1'B);
       ELSE RETURN('0'B);

 CHEK: ENTRY BIT(1);

       I=80; J=0;
       DO WHILE(SUBSTR(CARD_BUFFER,I,1)=' ' & J<=LEN1-LEN);
       I=I-1; J=J+1;
       END;
       IF J>=LEN1-LEN THEN RETURN('1'B);
       ELSE RETURN('0'B);

   E4: DO;
       DECLARE
       CADD BIN FIXED EXT,
       LOOKUP ENTRY(BIN FIXED) RETURNS(BIN FIXED),
       POSIT BIN FIXED(31) EXT;

  E4A: CALL SCAN; IF TCOUNT=0 THEN GOTO EA;
       CALL HASH; IF LOOKUP(POSIT)<0 THEN
            DO; DISPLAY('**E7: ATOM NOT FOUND**'); GOTO EA; END;
       I=POSIT; LEN=NCAR(POSIT); LEN1=NCDR(POSIT);
       DO WHILE(NCAR(LEN1)¬=CADD);
```

```
        I=LEN1; LEN1=NCDR(LEN1);
        END;
        IF LEN=1 THEN DO; FSTOR(POSIT)=0; GOTO E4A; END;
        LEN1=NRPLACD(I,NCDR(LEN1)); LEN=LEN-1;
        LEN1=NRPLACA(POSIT,LEN);
        GOTO E4A;
        END;

    E5: DO;
        RETAG='1'B; GOTO E1;
        E5A: RETAG='0'B; GOTO EA;
        END;

  GBUFF: ENTRY;

        CALL TEXTWRK(LE); LE=LE+1;
        RETURN;
END LISPG;
```

```
LISPP: PROC;

          /* LISP PRIMITIVE (ELEMENTARY) FUNCTIONS */

     DECLARE
          BFREE BIN FIXED EXT,
          BNUM BIN FIXED EXT,
          F BIN FIXED EXT,
          FREE BIN FIXED EXT,
          FSTOR(0:16000) BIN FIXED(31) EXT,
          GB16 BIT(16) STATIC,
          GB32 BIT(32) STATIC,
          GT1 BIN FIXED EXT,
          MFSTOR BIN FIXED EXT,
          NC BIN FIXED EXT,
          NIL BIN FIXED EXT,
          T BIN FIXED EXT,
          TRACE BIN FIXED EXT,
          TRCONS BIN FIXED EXT;

NCONS: ENTRY(CNA,CNB);
        DCL (CNA,CNB) FIXED BIN;
        GB32=UNSPEC(CNB);
        GB32=GB32 | SUBSTR(UNSPEC(CNA),17,16);
        GT1=NCELL;
        FSTOR(GT1)=GB32;
        RETURN(GT1);

NCELL: ENTRY;
        /* GETS THE NEXT AVAILABLE CELL IN FREE STORAGE */
        NC=FREE; FREE=FSTOR(FREE);
        RETURN(NC);

 NCAR: ENTRY(CA);
        DCL (CA) FIXED BIN;
        GB16=SUBSTR(UNSPEC(FSTOR(CA)),1,16);
        GT1=GB16;
        RETURN(GT1);

 NCDR: ENTRY(CD);
        DCL (CD) FIXED BIN;
        GB16=SUBSTR(UNSPEC(FSTOR(CD)),17,16);
        GT1=GB16;
        RETURN(GT1);
 NEQ: ENTRY(JEQ,KEQ);
      DCL (JEQ,KEQ) FIXED BIN;
        IF NATOM(JEQ)=T THEN
             IF NATOM(KEQ)=T THEN
                  IF JEQ=KEQ THEN RETURN(T);
                  IF NCDR(JEQ)>BNUM THEN
                       IF NCDR(KEQ)>BNUM THEN
                            IF FSTOR(NCDR(JEQ))=FSTOR(NCDR(KEQ))
                            THEN RETURN(T);
        RETURN(F);

 NATOM: ENTRY(JAT);
      DCL JAT;
        IF JAT>MFSTOR THEN RETURN(F);
        IF JAT=NIL THEN RETURN(T);
        IF FSTOR(JAT)<0 THEN RETURN(T);
        RETURN(F);

NRPLACA: ENTRY(JCA,KCA);
        /*  REPLACES CAR(JCA) WITH KCA  -  VALUE IS JCA  */
             DCL (JCA,KCA);
             GB32='00000000000000001111111111111111'B;
             GB32=GB32 & UNSPEC(FSTOR(JCA));
             GB32=GB32 | SUBSTR(UNSPEC(KCA),17,16);
             FSTOR(JCA)=GB32;
             RETURN(JCA);

NRPLACD: ENTRY(JCD,KCD);
        /*  REPLACES CDR(JCD) WITH KCD  -  VALUE IS JCD  */
             DCL (JCD,KCD);
             IF JCD=NIL THEN RETURN(NIL);
             GB32=SUBSTR(UNSPEC(FSTOR(JCD)),1,16) | UNSPEC(KCD);
             FSTOR(JCD)=GB32;
             RETURN(JCD);

   END LISPP;
```

APPENDIX D

USING THE NPS LISP 1.5 VERS 1 SYSTEM, AN EXAMPLE

```
EXECUTION BEGINS

   --FILES LISPTEXT AND DUMPLISP EXIST--

NPS LISP 1.5 VERS 1 INITIALIZING

SPECIAL OPTIONS?
_n

CALL EVALQUOTE, ARGS:
     Load$ lisptext dumplisp
LOADING LISPTEXT
LOADING DUMPLISP
CALL EVALQUOTE, ARGS:
_revs (a b)

VALUE IS:
(B . A)

CALL EVALQUOTE, ARGS:
_count$
CELLS IN FREE STORAGE:              14509
CELLS AVAILABLE FOR NUMBERS:          300
CALL EVALQUOTE, ARGS:
_dump$
CALL EVALQUOTE, ARGS:
_collect$

CALL EVALQUOTE, ARGS:

_count$
CELLS IN FREE STORAGE:              14531
CELLS AVAILABLE FOR NUMBERS:          300
CALL EVALQUOTE, ARGS:
_e$

EDIT LISP

_p 1,3
1    CONS (A B)
2    DEFINE ((
     (REVS (LAMBDA (X Y) (CONS Y X))) ))
3    REVS (A B)

_c /a b/(a b) c/
3    REVS ((A B) C)

_r 3
```

```
VALUE IS:
(C A B)

_ret
CALL EVALQUOTE, ARGS:
_dump$
CALL EVALQUOTE, ARGS:
_end lisp
EXIT LISP SYSTEM
R; T=20.62/32.68 19.47.01
```

GLOSSARY

ATOM:

A synonym for atomic symbol, the basic constituent of an S-expression.

BYTE:

A unit of the IBM/360 memory structure. The byte consists of 8 binary digits (bits) and is one-quarter of an IBM/360 word.

CODE-BYTE (C-BYTE):

The first byte of the first line of file LISPTEXT. The C-byte is used to store certain codes used by the Supervisor to handle the system's special files. (See Section II.B.3.(b))

CRASH:

A term which refers to the abrupt halt in service due to a malfunction in a time-sharing system.

DOUBLET:

A pair of S-expressions, arguments for the Interpreter function EVALQUOTE.

DUMP:

The action of copying LISP memory, including the making of copies of specific values and files needed to restore the LISP system to the state existing at the time of the dump.

KEYWORD:

A special word used in the files AUTOLISP and DUMPLISP to link and describe areas of condensed storage. The first 2 bytes of the keyword contain the address of the next keyword. The last 2 bytes contain the number of words of FSTOR which were condensed (i.e., the number of adjacent words in the current section of FSTOR which contain sequential numbers). A keyword (other than the first) is always followed by a word containing the address of the first word of the condensed section. (See Section II.B.3.(a)).

OVERLORD:

The Supervisor of the 7090 LISP system.

S-expression:

Short for SYMBOLIC EXPRESSION. All data and all programs written in LISP 1.5 are in the form of S-expressions which are made up of either an ATOM, 'A', a dotted pair of atoms (A.B), or a dotted pair of S-expressions, ((A.B).C).

S-EXPRESSION DOUBLET:

See doublet.

LIST OF REFERENCES

1. Gentry, D. G., An Implementation of LISP 1.5 for the IBM 360/67 Computer, M. S. Thesis, Naval Postgraduate School, 1969.

2. Weissman, C. LISP 1.5 Primer, Belmont: The Dickenson Publishing Company, Inc., 1968.

3. System Development Corporation Report TM-2237/103/00, Operating System, Input-Output, File, and Library Functions, by S. L. Kameny and C. Weissman, 11 April 1966.

4. Bolt Beranek and Newman, Inc., The BBN 940 LISP System, by D. G. Bobrow, and others, 15 July 1967.

5. Naval Postgraduate School, CP/CMS User's Guide (Revised 6/69). Monterey: Naval Postgraduate School Press, 1969.

6. International Business Machines Corp., IBM System/360 PL/I Reference Manual, Form C-28-8201-1, March 1968.

7. McCarthy, J. and others, LISP 1.5 Programmer's Manual, Cambridge: The M.I.T. Press, 1962.

8. System Development Corporation Report TM-2337/101/00, LISP 1.5 Reference Manual for Q-32, by S. L. Kameny, 9 August 1965.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	20
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Chief of Naval Operations (OP-91) Department of the Navy Washington, D. C. 20350	1
4.	Department of Operations Analysis (Code 55) Naval Postgraduate School Monterey, California 93940	1
5.	LTJG Gary A. Kildall, USNR, Code 53kd Department of Mathematics Naval Postgraduate School Monterey, California 93940	2
6.	Commandant of the Marine Corps, Code AX Headquarters Marine Corps Washington, D. C. 20380	1
7.	Commandant of the Marine Corps, Code AO3C Headquarters Marine Corps Washington, D. C. 20380	1
8.	Major John C. Pilley, USMC 1940 Paralta Avenue Seaside, California 93955	1

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

ORIGINATING ACTIVITY *(Corporate author)* Naval Postgraduate School Monterey, California 93940	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

REPORT TITLE
The NPS LISP 1.5 VERS 1 PROGRAMMING SYSTEM

DESCRIPTIVE NOTES *(Type of report and, inclusive dates)*
Master's Thesis; April 1970

AUTHOR(S) *(First name, middle initial, last name)*
John C. Pilley

REPORT DATE April 1970	7a. TOTAL NO. OF PAGES 105	7b. NO. OF REFS 8
CONTRACT OR GRANT NO. PROJECT NO.	9a. ORIGINATOR'S REPORT NUMBER(S) 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*	

DISTRIBUTION STATEMENT This document has been approved for public release and sale, its distribution is unlimited.

SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Naval Postgraduate School Monterey, California 93940

ABSTRACT

The design and implementation of the NPS LISP 1.5 VERS 1 programming system is described. NPS LISP 1.5 VERS 1 is a complete programming system built around the original implementation of LISP 1.5 on the IBM 360/67 computer at the Naval Postgraduate School. The new version includes a Supervisor and an Editor, as well as the original LISP Interpreter. The VERS 1 system is written in PL/I for time-sharing operation on the IBM 360/67 computer.

Unclassified
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Complete LISP programming system						
IBM 360/67 computer						
LISP Time-sharing operation						

DD FORM 1 NOV 65 1473 (BACK)
S/N 0101-807-6821

Unclassified
Security Classification

A-31409