Computing Centre

University of Waterloo

LISP /360

A description of the University of Waterloo
LISP 1.5 Interpreter for the IBM System /360

Second Edition

March, 1968

J. F. Bolce,
Computing Centre,
University of Waterloo,
Waterloo, Ontario.

Documentation by -

J. F. Bolce,
R. H. Cooper.

DISCLAIMER

ACKNOWLEDGMENTS

I wish to thank Mr. Jan Kent for his experience and guidance in the initial design and implementation of the interpreter.

This project was made possible by the support of Professor J. W. Graham, Director of the Computing Center.

J. F. Bolce.

CONTENTS

Pages

PART I INTRODUCTION 1

Machine Requirements 1

PART II ORGANIZATION OF THE SYSTEM 2

2.1 The Lisp Cell 3
2.2 The Object List 3
2.3 Atoms 3
Property Lists 4
Fullwords 4
Printnames 5
Numbers 5
Binary Markers 6

PART III ORGANIZATION OF THE INTERPRETER 7

3.1 Register Assignments 7
3.2 Passing Arguments 8
3.3 Program Initialization 8
3.4 Defining the Object List 8
3.5 Recursion Techniques 10
The Pushdown Stack 10
SAVE 10
UNSAVE 11
3.6 The Management of Freeword Storage 11
CAR 12
CDR 12
ATOM 12
CONS 12
The Garbage Collector 13
3.7 Lisp Output 14
PRINT 14
PCKOVR 15
PUTATOM 15
PRIN1 15
TERPRI 15
Print DCB 15
3.8 Lisp Input 16
Program Format 16
READ 16
TRYATOM 17
ALLATOM 18
Read DCB 18
Character Handling Routines 19
STARTREAD 19
ADVANCE 19
ENDREAD 19
UNPACK 19
CCLASS 19
PACK 20
CLEARBUFF 20
MKATOM 20

		Pages
3.9	Data Types	20
	Alphabetic	20
	Integers	21
	Floating Point	21
	Logical	21
3.10	The PROG Feature	22
	PROG	22
	GO	22
	RETURN	23
3.11	Arithmetic Routines	23
3.12	Other Functions	24
	APPEND1	24
	TTAB	24
	XTAB	24
	EVENP	24
	CCLASS	25
	MKATOM	25
	The Lisp Library	26
	The Library Function	26.1
	Initializing the Library	26.2
3.13	The Trap Supervisor	27
3.14	Tracing	27.1
	Function Level Trace	27.1
	Statement Level Trace	27.1
	Trace	27.1
	Untrace	27.1
3.15	Possible Differences	27.1
3.16	Future Improvements	27.2
PART IV	ERROR DIAGNOSTICS	28
4.1	Syntax Errors	28
4.2	Runtime Errors	28
	Interpreter Errors	29
PART V	IMPLEMENTATION GUIDE	32
5.1	Assembly Options	32
5.2	Execute Card Options	32
5.3	Sample Procedures	33
5.4	Reporting Errors	34
APPENDIX I	THE LISP INTERPRETER	35
	Definition in M-expressions	36
	Flowcharts	39
APPENDIX II	SAMPLE OUTPUT	45
APPENDIX III	IMPLEMENTED FUNCTIONS	
References		

I INTRODUCTION

Lisp /360 is an interpretative Lisp 1.5 system written at the University of Waterloo. It has been modelled after the Lisp 1.5 program on the IBM 7090 (1) although many ideas have been borrowed from the CDC 3600 Lisp interpreter (2).

The interpreter was written with several features in mind:

(A) More understandable error diagnostics.

(B) More effective read routines.

(C) Compatibility with the previous Lisp language for the IBM 7090.

(D) Speed in Lisp interpretation.

(E) The use of 24-bit fullword addresses to ensure the addressability of large memories.

(F) Maximum use of the Universal Instruction Set for the IBM /360.

(G) Acceptance of Lisp problem programs from both IBM keypunches -the 026 and the 029.

BASIC MACHINE REQUIREMENTS

A /360 computer with the following requirements:

(a) operating system OS /360

(b) the Universal Instruction Set.

II ORGANIZATION OF THE SYSTEM

In this section is described the internal structure of the Lisp /360 Interpreter.

It is assumed throughout these sections that the reader has a working knowledge of the Lisp 1.5 programmer's manual (1).

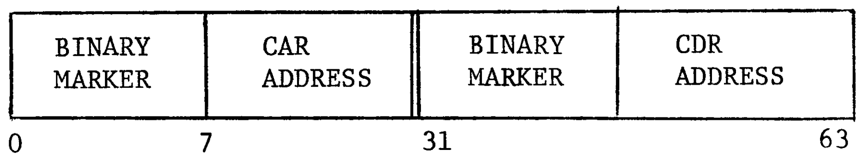
Lisp /360 is a problem program operating under the System /360 Operating System. The memory which is available to the interpreter depends upon its allocation by the operating system on the particular machine involved.

The coding of the interpreter itself occupies about 12 K bytes. This is directly followed by the pushdown stack and the object list (5 K bytes). The size of the pushdown stack is set at assembly time (now at 4 K words). Freeword storage occupies the remainder of available core and is obtained from the operating system by the GETMAIN macro.

The structure of the atoms and their property lists is organized in a manner similar to Lisp on the CDC3600. This particular organization has certain disadvantages as far as the speed of the read and print routines and the utilization of memory is concerned. However, the added speed obtained in the interpreter routines and the garbage collector provide reasonable justification.

2.1 THE LISP CELL

A Lisp cell occupies one double word of storage. This allows the use of two 24-bit addresses enabling the interpreter to cover any size of /360 memory. Each of these 24-bit addresses points to a list. The 8 bits remaining in each of the two full-words is used to store binary markers. These can easily be tested with the Test Under Mask instruction.



2.2 THE OBJECT LIST

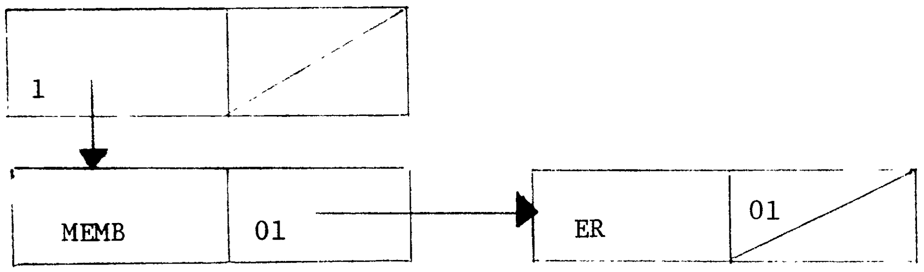
The object list is a sequential list which initially contains all the predefined atoms. This initial object list is generated at assembly time. When an atom is produced by the READ function, this list is searched to see if the atom already exists. If not, the atom is appended to the object list. The object list is accessible as the APVAL property of the atom OBLIST.

2.3 ATOMS

If the first byte of the first fullword in a Lisp cell has a 1-bit in the sign position then that particular cell is an atomhead.

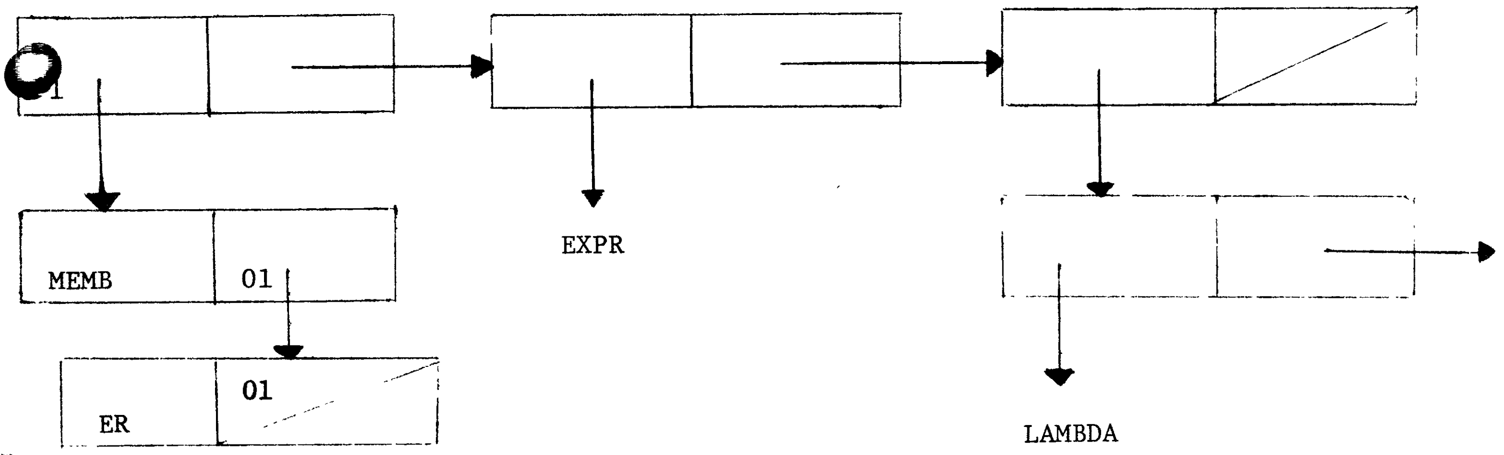
In the atomhead the first address in the double-word cell points to the atom's fullword list. The second address points to the atom's property list.

The atom MEMBER, with an empty property list is illustrated below.



Property Lists

The EXPR-(MEMBER(LAMBDA... has the following structure.



Fullwords

Fullwords in freeword storage are used to replace the "fullword storage" in Lisp 1.5.

A fullword is a particular type of Lisp cell which has a) the second bit of the second word turned on and b) one of the following in the upper word.

1) Four BCD characters from a printname (padded on the right with the null character X'00' if necessary.

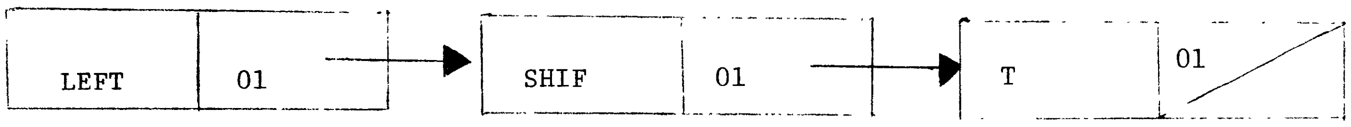
2) A 32 bit integer or floating point number.

3) The binary address of a Lisp routine.

Printnames

The address in the upper word of the atomhead of a non-numeric atom points to the linear list of the BCD printname.

The atom LEFTSHIFT would have, for example, the following fullword list.



Numbers

There are four types of numbers:

1) integer

2) Floating point

3) Logical (read as octal)

All numbers are stored internally in 32 bit binary form and must be converted to BCD for printing.

Binary Markers

Bits 0-7 of each atomhead contain several binary markers indicating the type of fullword list pointed at by the upper address.

bit	0	1	2	3	
	1	0	0	0	BCD PRINTNAME
	1	1	0	0	INTEGER
	1	1	1	0	FLOATING POINT
	1	1	0	1	LOGICAL

The first bit of the lower address word is used by the garbage collector to mark active cells.

III ORGANIZATION OF THE INTERPRETER

This section contains a description of the interpreter and some of its main routines. Flowcharts of some of these routines are given in Appendix 1. The words written in capital letters refer to the interpreter coding.

3.1 REGISTER ASSIGNMENTS

Register	Name	Usage
0		LOCAL WORK REGISTER
1		LOCAL WORK REGISTER
2		SUBROUTINE LINKAGE
3		WORK REGISTER
4	K4	CONSTANT VALUE '4'
5	NILR	ADDRESS OF NIL, BASE REGISTER OF THE OBJECT LIST
6	FREE	POINTER TO FREEWORD STORAGE LIST
7	PDS	POINTER TO TOP OF PUSHDOWN STACK
8	A	FIRST FUNCTION ARGUMENT
9	Q	SECOND FUNCTION ARGUMENT
10	M	SAVE AREA FOR LIST POINTERS
11		BASE REGISTER
12		BASE REGISTER
13		POINTER TO SYSTEM SAVE AREA AND BASE REGISTER
14		LOCAL WORK REGISTER
15		LOCAL WORK REGISTER

The usage of the registers will be described in greater detail in the descriptions of the routines which follow.

3.2 PASSING THE ARGUMENTS

The A and Q registers and the 20 locations starting at ARGS are used to pass up to 22 arguments to a LISP function. Register A contains the return value (list pointer).

Register M is used normally to point to a list that should not bee lost if a garbage collection occurs when doing a CONS.

3.3 PROGRAM INITIALIZATION

Program Initialization consists of:

1) Testing the PARM field of the EXEC control card.-if the parameter 'BCD' is encountered the type of brackets and the plus sign is set to the BCD code.
2) Opening the files.
3) Issuing an STIMER macro to set the clock.
4) Converting the object list from relocatable to fixed form.
5) Issuring a GETMAIN to obtain all remaining core for use as freeword storage.

3.4 DEFINING THE OBJECT LIST

The macro ECHO is used to define an atom and attach it to the object list.

The address constants generated by the macro are relative to the NIL atom, i.e. are relocatable. This reduces the amount of text to be processed by the system loader. During program initialization a sequential scan is made to add the address of NIL to all address constants. The examples following illustrate the absolute addresses.

The macro's parameters are arranged as follows:

PARAMETER	
1	PRINT NAME (1 TO 12 CHARACTERS)
2	INDICATOR eg. SUBR
3	SUBROUTINE ENTRY POINT NAME
4	NAME OF ARGUMENTS TO A SUBR

An example of the coding generated for an atom with no property list:

```
M        ECHO      MEMBER
         DC        A(*+8,*+32)    Object List Link
M        DC        X'80',AL3(*+8),A(NIL)    Atomhead
         DC        C'MEMB',X'40',AL3(*+8)    Fullword List
         DC        C'ER',X'0000',X'40',AL3(NIL)
```

An atom defined withaa property list:

```
         ECHO      DEFINE,SUBR,DEFINE,1
         DC        A(*+8,*+52)    Object List Link
         DC        X'80',AL3(*+8),A(*+21)    Atomhead
         DC        C'DEFI',X'40',AL3(*+8)    Fullword List
         DC        C'NE',X'0000',X'40',AL3(NIL)
         DC        A(SUBR,*+4)    Indicator
         DC        A(*+8,NIL)    Link to Any Other Properties
         DC        AL1(1),AL3(DEFINE),X'40',AL3(NIL)

                      ↑            ↑
                   argument   entry point
                    count       address
```

3.5 RECURSION TECHNIQUES

All interpreter routines that are FSUBR's must be re-entrant, as they are either recursive or may be re-entered in the process of evaluating their arguments. Of prime importance in effecting recursion is the pushdown stack.

The Pushdown Stack

One pushdown stack is used to save the linkage addresses and list pointers for the recursive and re-entrant routines. The pushdown stack is a linear block of core. It is preceded by a work area which may contain pointers to lists that must be collected if a garbage collection occurs. The pushdown stack is followed by the object list beginning with the atom NIL.

Two macros, SAVE and UNSAVE, are used to pass data to and from the stack.

The Macro SAVE

SAVE has one argument - a register. Its purpose is to:

(a) Store the contents of the designated register on the top of the stack.

(b) Increment the stack pointer (PDS).

(c) Check for the occurrence of stack overflow.

For example,

```
     SAVE          A
```

would generate the following coding:

```
     ST            A,0(PDS)
     BXH           PDS,K4,ERG2
```

The register PDS points to the top of the stack. The BXH instruction increments PDS by the constant 4 (in K4) and transfers control to ERG2 if the result is greater than the address of NIL. The register following K4 (see previous chart of registers) contains the address of NIL.

The Macro UNSAVE

This macro decrements the stack pointer PDS and loads the designated register with the top of the stack.

For example:

```
          UNSAVE        A
```

would generate

```
          SR            PDS,K4
          L             A,0(PDS)
```

3.6 THE MANAGEMENT OF FREEWORD STORAGE

The primitive Lisp functions are those used most frequently by the Interpreter and the Lisp programmer. With this in mind Lisp /360 has been designed to ensure that these functions are coded with as few instructions as possible.

For this reason (as has been mentioned previously) the Lisp cell has the form -

BINARY MARKER	UPPER ADDRESS	BINARY MARKER	LOWER ADDRESS

Hence, fullword instructions are easily used to effect cell address referencing and immediate instructions may be used to manipulate the binary markers.

CAR

To execute the function CAR(A) → A requires the one instruction:

```
        L           A,CAR(A)
```

where CAR has the value zero.

CDR

The function CDR(A) → A requires the one instruction:

```
        L           A,CDR(A)
```

where CDR has the value four.

ATOM

The testing of whether A is an atom illustrates the use of the binary markers.

```
        TM          CAR(A),ATOM     - Is A an atom
        BO          ITSATOM         - Yes, bit is on
```

The label ATOM has the value X'80'.

CONS

A freeword list, pointed at by register FREE, is produced by the garbage collector.

The routine CONS:

1) Stores the contents of registers A and Q in the top cell of the freeword list.
2) Places the address of the cell into register A.
3) Sets FREE to point at the next cell in the freeword list.

This is accomplished by doing:

```
    ST      A,CAR(FREE)     Store A in the CAR
    LR      A,FREE          Point A to the cell
```

```
L       FREE,CDR(FREE)    Point FREE to rest of list
ST      Q,CDR(A)          Store Q in the CDR
```

The end of the freeword list is recognized by having the CDR address of the last cell in the list set to the value '1'. When CONS attempts to store in the cell at location 1, a specification interrupt occurs. When this programme interrupt occurs the contents of FREE is checked for the value '1' and if present a garbage collection is effected. The CONS is then completed.

The Garbage Collector

The garbage collector is entered whenever the CONS routine causes a specification interrupt by attempting to store in storage location 1. This is an indication that the freeword list has been exhausted.

At this point the garbage collector must mark all the lisp cells that are currently needed, and then link together all of the unmarked cells to produce a new freeword list.

Between the address TEMPORAR and the bottom of the pushdown stack is stored the address of:

1) The object list.
2) The association list.
3) Any function or interpreter list pointers that are needed further.

The garbage collector saves the A,Q, and M registers on the top of the stack and does a scan from TEMPORAR to the top of the used part of the stack looking for all addresses pointing into freeword storage. All such lists are marked.

A sequential scan is then made through all freeword storage to unmark the marked cells and to collect all unmarked cells into a new freeword list. If no cells are collected (FREE still equals 1) error GC2 occurs.

Since freeword storage is obtained from the operating system by means of the GETMAIN macro and this area is not necessarily contiguous with the object list, the garbage collector must scan two blocks. The garbage collector is written to collect a list of areas. Each area is preceded by a two word cell, the first word containing the address of the start of the next area (zero signals last), the second containing the address of the end of this area.

The garbage collector is used to initialize freeword storage by setting FREE to the value 1. The first CONS triggers the garbage collection.

The marking is done by setting 'ON' the first bit (bit 0) of the lower address of all cells that can be reached by CAR and CDR chains. If a cell is reached that is already marked the chain is stopped (the atom NIL rapidly becomes marked). If a fullword list is encountered then only the CDR chain is followed.

3.7 LISP OUTPUT

The output of S-expressions is handled by the PRINT routine. PRINT makes use of PCKOVR and PUTATOM. The primitives PRIN1 and TERPRI are also used for output.

PRINT

The buffer area used by PRINT is the area labelled LINE. PRINT loads register P from PRTAB, which contains the current buffer pointer. A check is made on the argument in A to see whether it is an atom or a list. If it is a list, PUTLIST is entered. The value zero is saved and an iteration commences resulting in the printing of the S-expression. The end of the iteration is signalled by unsaving the zero.

PCKOVR

This routine adds '1' to register P and if the result is over the value of LINEMAX the buffer line is printed by WRLINE.

PUTATOM

When PRINT encounters an atom it calls upon PUTATOM to do the necessary conversion and to move the resulting PNAME to the output buffer. Floating-point numbers are output in the form sd.ddddddEsdd

PRIN1

PRIN1 loads P with the PRTAB value and calls upon PUTATOM to convert the atom and to move it to the buffer.

TERPRI/WRLINE

This routine prints the data stored in the buffer LINE, resets it blanks and sets PRTAB to LINE+5.

If a PRINT follows a PRIN1, the list generated by PRINT follows the data output by PRIN1 rather than overlaying it.

```
eg.     (PRIN1(QUOTE $$'RESULTING LIST IS -'))
        (PRINT LIST)

        would print

        RESULTING LIST IS -(---- LIST ------
```

PRINT DCB

The PRINT data control block has the following parameters set:

DSORG=PS,MACRF=(PM),DDNAME=SYSPRINT,RECFM=VBA,LRECL=136,BLKSIZE=500

3.8 LISP INPUT

All Lisp input is handled by the READ routine which in turn uses the routines GETCHAR, TRYATOM, and ALLATOM.

PROGRAM FORMAT

The LISP programs are punched free form in card columns 1 to 72. A Lisp program consists of a series of doublets that are read and presented as arguments to the function EVALQUOTE(FN,ARGS). No control cards such as TEST and SET are recognized nor is STOP.

Extra right parentheses may be placed at the end of an S-expression to prevent a bad bracket count ruining the evaluation of the next doublet. On reading the next S-expression the extra right parentheses are ignored.

If the last program card has been read and there are insufficient right parentheses, the necessary brackets are supplied, and the structure is printed with the error R2-BAD BRACKET COUNT.

READ

READ uses TRYATOM to attempt to form an atom from the succeeding characters on the cards. TRYATOM returns an atom if it is successful, otherwise it indicates that it found a left or right parenthesis or a dot. Commas are treated in the same manner as blanks.

READ initiallly calls on TRYATOM to form an atom. If an atom is found READ returns to the calling routine with the atom. If a left parenthesis is indicated by TRYATOM, READ links to UPPER. UPPER is the recursive entry point for reading a list that is entered whenever a left parenthesis is encountered. LOWER on the other hand is entered fecursively whenever a '.(' is encountered.

TRYATOM

This subroutine scans one syntactical unit from the card input area and attempts to identify the unit.

- commas and blanks act as delimiters and are otherwise ignored.
- left and right parentheses and dots are scanned off and followed by a special return to the calling routine.
- to aid in the identification of character strings, a series of bit switches in the byte ATOMIND are used. These indicators are as follows:

BIT	NAME	USAGE
8	ATOMIND	A character other than a delimiter has been encountered. Construction of an atom is in progress.
7	NUMIND	The first character was a digit. The atom being constructed is numeric.
6	FLOATIND	A dot was encountered when NUMIND was on. The number is floating point.
5	EXPIND	The letter E was encountered when FLOATIND was on. The number has an exponent.
4	NEGEXP	A negative sign was encountered when EXPIND was on. The exponent is negative.
3	NEGINT	A minus sign followed by a digit was encountered. The number is negative.
2	LOGICAL	The letter Q was encountered while NUMIND was on. The number must be octal. EXPIND is set on.

The register CHAR contains the address of the character being examined. The routine GETCHAR is used to move the pointer to the next character, which may require the reading of another card. If the two characters $$ are encountered the LISP literal that follows is recognized. The characters between the delimiters are assembled into an alphabetic atom.

As the characters are picked off the cards they are stored in three areas depending on the setting of indicators. These areas are as follows:

CHARATA	- to store alphabetic strings
DIGITA	- to store a numeric string
EXPA	- to store digits defining an exponent

When a Lisp delimiter is encountered with ATOMIND 'ON' control will pass to ALLATOM.

ALLATOM

If only ATOMIND is 'ON' then this routine scans the object list to find an atom with the same PNAME as the character string in CHARATA. If none is found an atom is created and added to the object list. Numeric strings in DIGITA and EXPA are converted into internal fixed or floating point numbers. The proper type of atom is then created.

READ DCB

The READ data control block has the following parameters set:

```
DSORG=PS,MACRF=(GL),DDNAME=SYSIN,RECFM=FB,LRECL=80,BLKSIZE=80
```

CHARACTER HANDLING ROUTINES

The functions STARTREAD, ADVANCE, and ENDREAD are used to acquire data from a data card one character at a time. The atom returned is not placed on the object list thus gaining processing speed and preventing the object list from becoming cluttered. The function CCLASS may be used to identify the type of character returned.

STARTREAD( )

STARTREAD causes a new card to be read. The value of the function is an atom made from the first character on the card.

ADVANCE( )

ADVANCE returns as an atom the next character on the card. After the 72'd character is read, a STARTREAD occurs. No end of card or last card indication is given. The objects CURCHAR and CHARCOUNT are not implemented.

ENDREAD( )

This function causes the remainder of the card to be ignored.

UNPACK(A)

The characters in the PNAME of the atom A are returned as a list of atoms. The atoms are not on the object list.

CCLASS(X,Y)

CCLASS returns T if the first character of the PNAME of the atom X is in the set of characters of the PNAME of Y.

```
e.g.  CCLASS(A,BDAC) → T
      CCLASS(1,B1C)  → NIL
      CCLASS($$'1',A0123456789) →T
```

The functions PACK, CLEARBUFF and MKATOM are used to assemble a character string and to convert it to internal form. Atoms generated are placed on the object list.

PACK(X)

The function PACK places the PNAME of the atom X at the end of the string of characters in CBUFF. The size of CBUFF is an assembly parameter currently set at 80.

CLEARBUFF( )

This function sets CBUFF to blanks. CBUFF is initially blank and is reset to blanks by MKATOM and CLEARBUFF.

MKATOM ( )

The function MKATOM expects a valid S-expression in CBUFF. This character string is given to the READ function which returns an atom or a list corresponding to the S-expression found. Any brackets in the character string must be of the type indicated on the EXEC control card. If CBUFF is blank or there are unmatched parentheses, the error CH4 is given. CBUFF is reset to blanks.

3.9 DATA TYPES

Alphabetic

The READ routine will accept input punched on either an IBM 029 or 026 keypunch (BCD or EBCDIC). The characters which are recognized as being different are the left and right parentheses and the plus sign. The character set used is indicated in the PARM field of the EXEC card (ref section V). All characters are accepted as valid.

The maximum length of a BCD printname is presently set at 80 characters. This may be changed at assembly time by setting the variable -ATMSZ to the desired value.

Integers

A fixed point number (integer) consists of an optional sign followed by up to 16 digits with or without a positive scale factor. The maximum integer is $2^{32}-1$.

Floating Point

A floating point number consists of an optional sign followed by up to 16 digits with a decimal point that is not the first or last character. An exponent may be present in the form of the letter E followed by an optional sign and one or two digits.

Floating point numbers are stored in the short precision format giving 7 decimal places of precision and a magnitude range of 10^{-78} to 10^{+75}.

Logical

A logical number consists of a maximum of 11 digits where each digit is derived from the character set 0 through 7. Each logical number is followed by the letter Q and if desired an optional scale factor. A sign will be ignored by the interpreter.

Logical numbers are considered to be in base 8 notation and are used to produce a 32-bit signed integer with any extra high-order bits being lost. The scale factor itself is an exponent to the base 8 and has the effect of shifting an octal digit left the number of times specified by this exponent.

3.10 THE PROG FEATURE

PROG

When PROG is entered, each PROG variable is paired with NIL and added to the top of the association list. The remainder of the programme is searched for atomic symbols understood to be labels. A GOLIST is formed in which each label is paired with a pointer to that part of the programme following the label.

The execution of the programme then commences. The PROGIND is set on. The list of statements is executed, one statement at a time. Since a statement is a list, all atoms are taken to be labels and are recognized and ignored. Before executing a statement by calling EVAL, the GOLIST, the association list, and the point to the rest of the programme is saved.

Since statements within PROG are evaluated for their effect rather than their value, the function COND must act somewhat differently. That is, if COND does not find a TRUE clause it must not give an error diagnostic, but rather it must return. To differentiate between a normal COND and a COND with a PROG, EVCON (which is used to evaluate the COND in both places), must test the PROGIND. Since, however, PROGIND is turned off whenever LAMBDA is encountered and when returning from a PROG, EVCON must save the PROGIND and restore it whenever control is returned to EVCON.

GO

GO unsaves the pushdown stack until the return address to the call on EVAL from PROG is encountered. The next three unsaves produce the ALIST,GOLIST, and the rest of the programme. The GOLIST is searched for the argument of GO. The stack is then restored by saving the pointer to the new statement to be executed, the GOLIST,ALIST, and the above-mentioned return address. A branch is then made to EVAL to evaluate the labelled statement.

<u>RETURN</u>

As in GO, RETURN unsaves from the pushdown stack all addresses stored by the PROG, then returns control to EVAL with the argument of RETURN in the A register.

3.11 <u>ARITHMETIC ROUTINES</u>

The arithmetic routines that perform similar operations are combined into one routine with multiple entry points. These routines all accept integer or floating point arguments and in the cases where there is more than one argument, the modes may be mixed, in which case the answers are returned in floating point form. All routines produce error diagnostics if non-numeric arguments are supplied to them. The logical variable is treated as integer.

Below is a list of functions implemented, grouped as to their common routines:

ADD1(X)→X+1

SUB1(X)→X-1

MINUS(X)→(-X)

PLUS(X,Y,Z,...)→X+Y+Z+.....

TIMES(X,Y,Z,..)→X*Y*Z*.....

DIFFERENCE(X,Y)→X-Y

QUOTIENT (X,Y)→X/Y

REMAINDER(X,Y)→X-INTEGER(X/Y)*Y

ZEROP(X)→T if X INTEGER and X=0,T if FLOAT and |X| GT 1.E-6 otherwise NIL

MINUSP(X)→T if X LT 0 otherwise NIL

LESSP(X,Y)→MINUSP(DIFFERENCE(X,Y))

GREATERP(X,Y)→MINUSP(DIFFERENCE(Y,X))

EQUAL(X,Y)→ZEROP(DIFFERENCE(X,Y))

The arguments of EQUAL may be S-expressions and mixed alphabetic and numeric.

EXPT(X,Y) X**Y

If Y is integer, EXPT is computed by repetitive multiplication and X may be negative. Otherwise logarithms are used and X may not be negative.

3.12 FUNCTIONS NOT IN THE LISP 1.5 PROGRAMMER'S MANUAL

APPEND1(X,Y)

The function APPEND1 is used to add an atom Y to the end of the list X. It has the lisp definition below -

```
(APPEND1 (LAMBDA (X,Y) (APPEND X(CONS Y NIL))))
```

TTAB (N)

The function TTAB is used to move the print buffer pointer PRTAB to buffer position N. N must be a positive integer. The S-expressions printed by PRIN1 or PRINT will begin at print position N.

XTAB(N)

This function is similar to TTAB but rather moves the buffer pointer N positions to the right of its present location. If the end of the buffer is passed the line is printed.

EVENP(N)

EVENP accepts an integer as an argument. It returns T if the integer is even, NIL if odd.

CCLASS(X,Y)

CCLASS returns true if the character X is in the set of characters Y. (refer to section 3.7).

MKATOM( )

Refer to section 3.7.

The Lisp Library

The library facility allows the convenient storage and retrieval of standard lisp functions such as SELECT. The library essentially functions as a set of possible inputs which may be selected by the interpreter.

The library is a partitioned dataset where each member is a sequence of cards containing valid lisp input. For example, this member SELECT would contain the doublet defining the lisp function SELECT. The member COMPILER would contain the definition statements for the lisp compiler functions. By execution the doublet -

```
LIBRARY((SELECT,COMPILER))
```

input to the interpreter would be switched from SYSIN, first to the member SELECT, then to the member COMPILER and then back to SYSIN.

The Library Function - LIBRARY((X))

The library function has one argument which is a list of desired library members to be selected as input. On entering the function, the argument list is NCONC'd to the list of presently outstanding requests. Thus the requests are serviced on a last-in first-out basis. If, on entry to the library function, input is still coming from SYSIN, various pointers are initialized such that any further requests for input will come from the first and then subsequent members in the list. When the end of the list is reached, input is switched back to SYSIN.

The value of the library function is the list of outstanding members.

When using the library function, the DD control card - //LISPLIB DD DSNAME=LISPLIB,DISP=OLD must be added to the control cards for executing a lisp program.

Initializing the Library

The library is a partitioned dataset with records that are unblocked card image. The cards must be EBCDIC.

The following control card contains the necessary information to define the direct access space required for the library.

```
//LISPLIB DD DSNAME=LISPLIB,VOLUME=SER=-----,DISP=(NEW,CATLG),    C
//           SPACE=(TRK,(30,10,2)),UNIT=2314,                     C
//           DCB=(RECFM=F,BLKSIZE=80,LRECL=80)
```

Members may be added to the library using the IEBUPDTE utility program.

```
// JOB
// EXEC PGM=IEBUPDTE
//SYSUT1 DD DSNAME=LISPLIB,DISP=OLD
//SYSUT2 DD DSNAME=LISPLIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
// SYSIN DD *
./ ADD NAME=CDDDR
DEFINE(((CDDDR(LAMBDA(Z)(CDR(CDDR X))) )))
./ ENDUP
/*
```

Interdependencies of functions defined in the library may be satisfied by using the LIBRARY function. For example, a compiler defined in the library requires the functions LENGTH and MAPLIST also defined in the library. The library member defining the compiler should then have a LIBRARY function request for the other functions.

The following example illustrates the interdependencies and also how to eliminate the printout of the defined function.

```
./ ADD NAME=COMPILER
LIBRARY((LENGTH,MAPLIST))
EVAL((DEFINE(READ))NIL)
(lambda expressions for the compiler functions))
./ ENDUP
```

Beware that circular interdependencies are not set up as the library function will then not terminate.

3.13 THE TRAP SUPERVISOR

All programme interrupts are accepted by the interrupt supervisor. On an interrupt, a check is first made for a garbage collection signal (register FREE equals 1). If not, the information below is printed.

PROGRAM INTERRUPT	- interrupt type
TRAP-PSW	- program status word
REGS0-7	- registers 0-7
REGS8-15	- registers 8-15

If the interrupt code is 7 to F, an arithmetic error has occurred. Execution of the function causing the interrupt is resumed.

If the interrupt code is 1 to 6, a function has indiscriminatly used CAR's and CDR's within an atom. Execution of the doublet causing the error is terminated with the error message F4 and the next doublet is read in for evaluation.

3.14 TRACING

Two levels of tracing have been implemented.

Function Level Trace

Trace prints the name of the function and its argument when the function is entered and its value on completion.

Statement Level Trace

A full trace of all entries to APPLY or EVAL is produced.

- examples in Appendix II.

TRACE(X)

If the argument of TRACE is an atom, a statement trace is produced of all entries to APPLY or EVAL.

If the argument of TRACE is a list of functions, a function trace of the named functions if produced. Only EXPRS are traced.

UNTRACE(X)

If the argument X is an atom, the statement trace is turned off. If a list, tracing of the named functions terminates.

3.15 POSSIBLE DIFFERENCES FROM OTHER LISP 1.5 SYSTEMS

1) The function MKATOM is a replacement and extention of the functions INTERN, MKNAM and NUMOB

   MKATOM=INTERN(MKNAM)

   MKATOM will accept any S-expression placed in CBUFF by PACK and returns an atom or a list structure.

2) UNPACK takes an atom as an argument.

3) GO must only be given atomic labels.

4) + andd- must not be used as characters in an atom.

5) The function CCLASS has been used to replace LITER, DIGIT, OPCHAR, and DASH. It is also more general.

6) The functions ARRAY, ERRORSET, RECLAIM, COUNT, UNCOUNT and SPEAK are not implemented. All other undefined functions should be definable in terms of those supplied.

7) The function TRACE may be used to give either a function trace or a statement trace.

8) Fixed point numbers may have a scale factor e.g. 17E5.

9) The functions STARTREAD and ADVANCE generate non-unique alphabetic atoms. They do not signal EOF nor EOR.

3.16 POSSIBLE EXTENSIONS OF THE SYSTEM

1) The ERROR routine should obtain and print more information when an error occurs.

2) The READ and TRYATOM routines should be rewritten to a more efficient logic.

3) A checkpoint facility should be added to allow TEST and SET and to allow batch processing of student jobs.

4) A compiler and LAP function should be added.

5) Floating point numbers could be stored in long precision form. The arithmetic routines perform computations in long precision, then truncate to single.

IV ERROR DIAGNOSTICS

4.1 SYNTAX ERRORS

If the READ routine finds syntactical errors in an S-expression one of the following special atoms will be inserted at the place in error:

ERRB - a '.' was encountered as the first non-blank character after a '('.

DOTERR1 - the second S-expression in a dotted pair is not followed by a right parenthesis.

DOTERR2 - a '.' or ')' was encountered as the first non-blank character after a dot.

The message R1-SYNTAX ERROR precedes the printing of the S-expression with the error. Execution is attempted.

4.2 RUNTIME ERRORS

When an execution time error occurs in a LISP 1.5 programme the following type of error diagnostic occurs:

```
*** error code - error message
*   S-expression 1
*   S-expression 2
*** TRACE BACK FOLLOWS
*   S-expression 3
*   S-expression 4
          .
          .
          .
```

S-expressions 1 and 2 are related to the type of error encountered and are described below with the error messages. The trace-back is a list of the functions entered recursively but not completed at the time of the error. The most recent function is printed first. The statement being executed precedes the call on the function containing the statement.

Interpreter Errors

A1-CALL TO ERROR

The Function ERROR has been called

S-expression 1 is the argument of ERROR

A2-FUNCTION NOT DEFINED

A construction of the form (FN...) has been encountered. However, the atom FN is not a defined function.

S-expression 1 is the FN in question.

S-expression 2 is the association list.

A3-NO ARGS OF COND TRUE

On evaluating the function COND, no true propositons were found.

S-expression 1 is the list of the arguments given COND.

S-expression 2 is the association list.

A5-SET VARIABLE UNDEF.

The function SET or SETQ was given an undefined programme variable.

S-expression 1 is the programme variable.

S-expression 2 is the association list.

A6-UNDEF LABEL IN GO

The label given as the argument of GO has not been defined.

S-expression 1 is the label.

S-expression 2 is the list of the labelled statements.

A7-MORE THAN 22 ARGS

The interpreter can handle only 22 arguments for a function.

S-expression 1 is the list of the arguments to the function.

A8-UNDEFINED VARIABLE

The variable is not defined as a function argument on the association list and does not have an assigned value.

S-expression 1 is the variable in question

S-expression 2 is the association list

CH4-INVALID S-EXPR GIVEN MKATOM

The area CBUFF is blank or contains unmatched parentheses.

F2-TOO MANY ARGUMENTS-EXPR

F3-TOO FEW ARGUMENTS-EXPR

The wrong number of arguments have been given to a defined function.

S-expression 1 is the list of the function variables.

S-expression 2 is the list of the supplied arguments.

F2-TOO MANY ARGUMENTS-SUBR

F3-TOO MANY ARGUMENTS-SUBR

The wrong number of arguments have been given to a lisp function.

S-expression 1 is the function.

S-expression 2 is the list of the arguments

F4-FN ERROR INSIDE ATOM

A function has indiscriminatly used CAR's and CDR's within an atom.

G2-PUSHDOWN STACK OVERFLOW

The depth of recursion is beyond the capacity of the interpreter's save area. Non-terminating recursion will cause this error.

GC1-NO STORAGE ALLOCATED

The operating system could not provide sufficient space for freeword storage (fewer than 2500 cells).

GC2-STORAGE EXHAUSTED

The garbage collector is unable to find any unused storage.

I2-(-X)**Y

The function EXPT has been requested to raise a negative number to a floating exponent.

I3-BAD ARITHMETIC ARGUMENT

An arithmetic routine was given a non-arithmetic argument.

I5-EXPT TOO LARGE

The maximum exponent is 174.6731.

L1-NOT IN LIBRARY-XXX

The member XXX is not in the library.

R1-SYNTAX ERROR

A syntax error has occurred while reading an S-expression. Reference the section on syntax errors.

R2-BAD BRACKET COUNT

An end of file was reached while reading an S-expression. S-expression 1 is the list as read with needed brackets generated.

R5-NAME OR NUMBER TOO LONG

A BCD printname or a number is longer than that accepted by the interpreter. Truncation occurs on the right.

V IMPLEMENTATION GUIDE

The distributed tape consists of the interpreter source deck, card image, with a blocking factor of 20.

5.1 ASSEMBLY OPTIONS

The stack size is set at assembly time. It may be changed by replacing the appropriate card in the source by either updating the tape or by first punching the card deck (about 4000 cards).

The pertinent cards as set in the distributed deck are as follows:

```
STACKSIZE   EQU   4000   WORDS FOR PDS
ATMSZ       EQU   80     MAX PNAME
CBUFFSZ     EQU   80     SIZE OF CBUFF
```

The distributed program requires about 33K plus space for freeword storage.

5.2 EXECUTE CARD OPTIONS

One option is recognized in the PARM field of the EXEC control card: PARM=BCD indicates that the Lisp programs were produced on an IBM026 Keypunch. The 029 keypunch is assumed as the default.

5.3 SAMPLE PROCEDURES

Assemble and test from the distributed tape

The object module is left in the data set LISP on the desired volume

```
//TSTLISP        JOB    MSGLEVEL=1

//               EXEC   ASMFCLG

//ASM.SYSIN      DD     VOLUME=SER=LISP,UNIT=2400,DSNAME=SOURCE,            X
//                      DCB=(RECFM=FB,BLKSIZE=1600,LRECL=80),DISP=OLD

//LKED.SYSLMOD   DD     VOLUME=SER=-----,DSNAME=LISP(LISP),                 X
//                      DISP=(NEW,KEEP),SPACE=(TRK,(11,5,1)),UNIT=2311

//GO.SYSPRINT    DD     SYSOUT=A

//GO.SYSIN       DD     *

     any test decks

/*
```

Following the above prodedure further tests may be run with the following control cards.

```
//LISP           JOB    MSGLEVEL=1

//JOBLIB         DD     DSNAME=LISP,VOLUME=SER=----,UNIT=2311,DISP=OLD

//               EXEC   PGM=LISP,PARM=BCD

//SYSPRINT       DD     SYSOUT=A

//SYSIN          DD     *

     Lisp tests, punched on an 026 keypunch

/*
```

Adding Lisp to LINKLIB

Lisp may be added to LINKLIB by modifying the corresponding card in the first procedure to -

```
//LKED.SYSLMOD  DD  VOLUME=SER=----,DSNAME=SYS1.LINKLIB(LISP),UNIT=2311,DISP=OLD
```

The JOBLIB card is then not required to run Lisp.

NOTE: The above assumes that the user is using the system ASMFCLG procedure or a comparable one with the same procedure name and step names. (i.e. ASM & LKED).

5.4 REPORTING ERRORS

Any comments, suggestions or indications of interpreter errors would be greatly appreciated by the author.

Please contact -

Mr. J. F. Bolce
Computing Centre
University of Waterloo
Waterloo, Ontario, Canada

Corrections and additions to the interpreter will be distributed as the need arises.

The author would appreciate receiving any interesting or useful Lisp programs and documentation. The programs could be returned on the supplied reel of tape.

APPENDIX I

The Lisp Interpreter

- Definition in M-expressions

- Flowcharts

THE LISP INTERPRETER

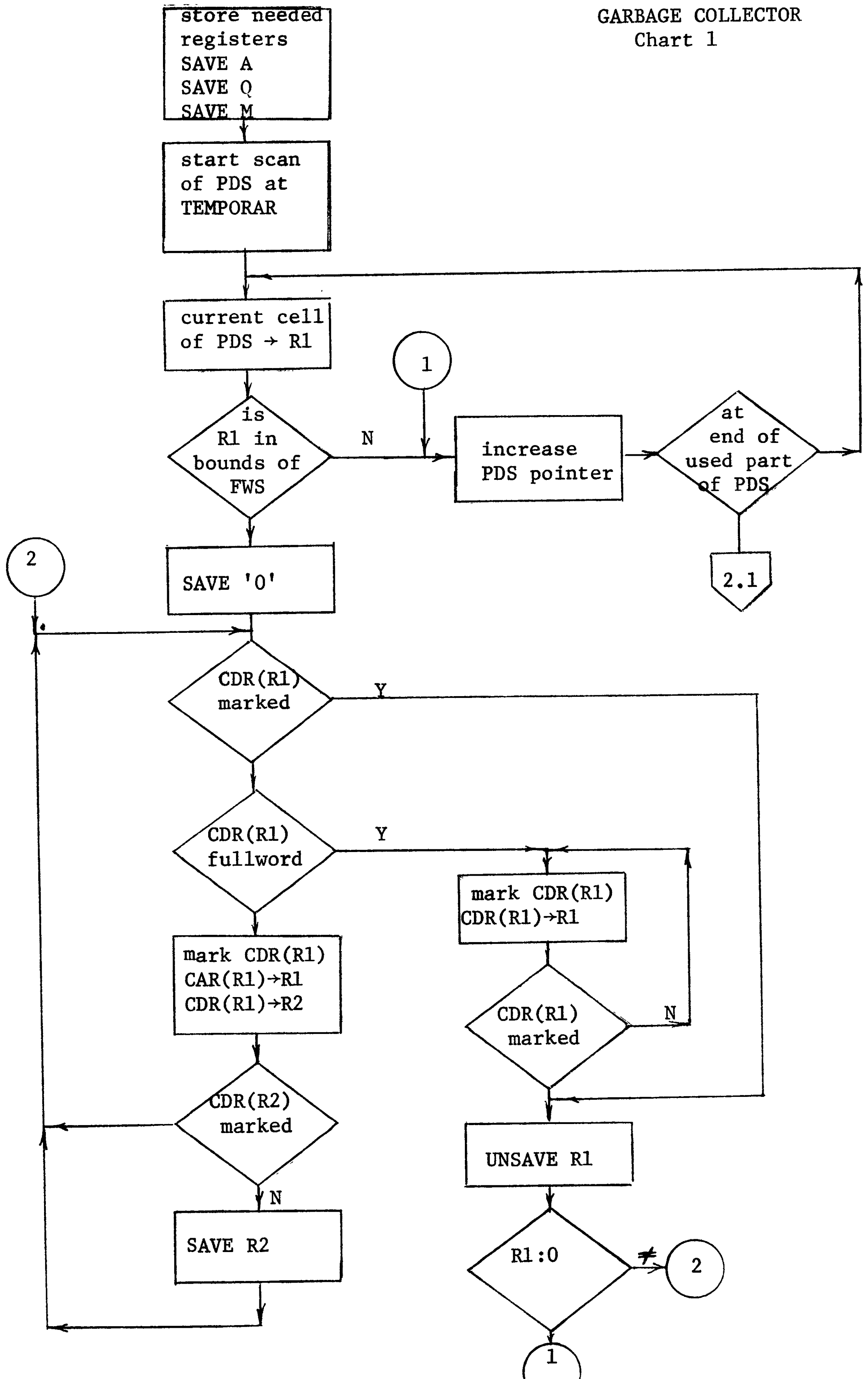
```
evalquote[fn;args] = [get[fn;FEXPR] V get[fn;FSUBR]→
                eval[cons[fn;args];NIL]
            T→apply[fn;args;NIL]]

apply[fn;args;a] = [
     null[fn] → NIL;
     atom[fn]→[get[fn;EXPR]→apply[expr;[1] args;a];
                                  ⎧ spread[args];          ⎫
                  get[fn;SUBR] → ⎨ ALIST: = a;            ⎬     ;
                                  ⎩ link to address-subr[1] ⎭
                  T → apply[cdr[sassoc[fn;a;λ[[];error[A2]]]];args;a];
     eq[car[fn];LABEL]→apply[caddr[fn];args;cons[cons[cadr[fn];caddr[fn]];a]];
     eq[car[fn];FUNARG]→apply[cadr[fn];args;caddr[fn]];
     eq[car[fn];LAMBDA]→eval[caddr[fn];nconc[pair[cadr[fn];args];a]];
     T → apply[eval[fn;a];args;a]]
```

```
eval[form;a] = [
     null[form] → NIL;
     numberp[form] → form;
     atom[form]  → [get[form;APVAL] →  car[apval[1]];
           T → cdr[sassoc[form;a;λ[[ ];error[A8]]]]];
     eq[car[form];QUOTE] → cadr[form];
     eq[car[form];COND] → evcon[cdr[form];a];
     atom[car[form]]→[get[car[form];EXPR]→apply[expr;[1]evlis[cdr[form];a];a];
                get[car[form];FEXPR]→apply[fexpr;[1]list[cdr[form];a];a];
                                     ⎧spread[evlis[cdr[form];a]];⎫
                get[car[form];SUBR]→ ⎨ALIST: = a;                ⎬      ;
                                     ⎩link to address-subr[1]    ⎭
                                     ⎧A:= cdr[form];             ⎫
                get[car[form];FSUBR]→⎨Q: = ALIST: = a;           ⎬      ;
                                     ⎩link to address-fsubr[1]   ⎭
                T→eval[cons[cdr[sassoc[car[form];a;λ[[];error[A9]]]];
                                cdr[form]];a]];
                T→apply[car[form];evlis[cdr[form];a];a]]
```

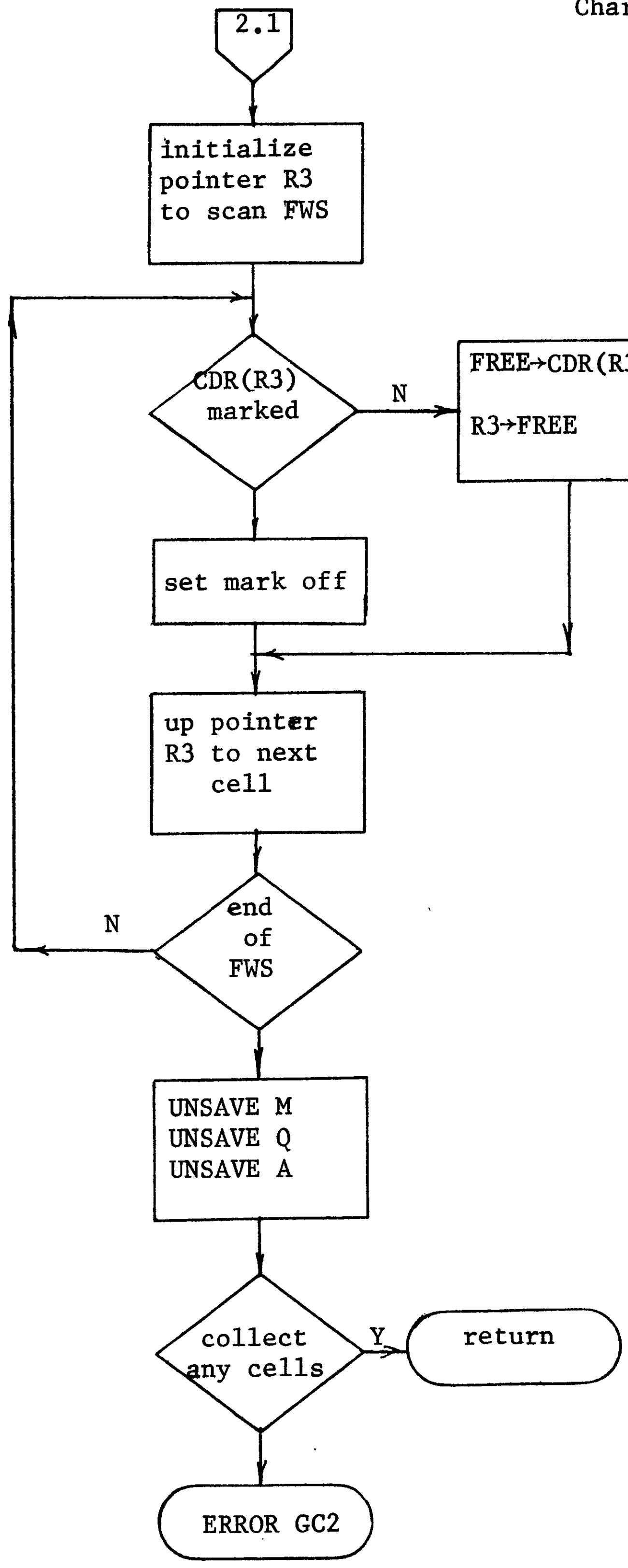
[1]This is the value returned by get.

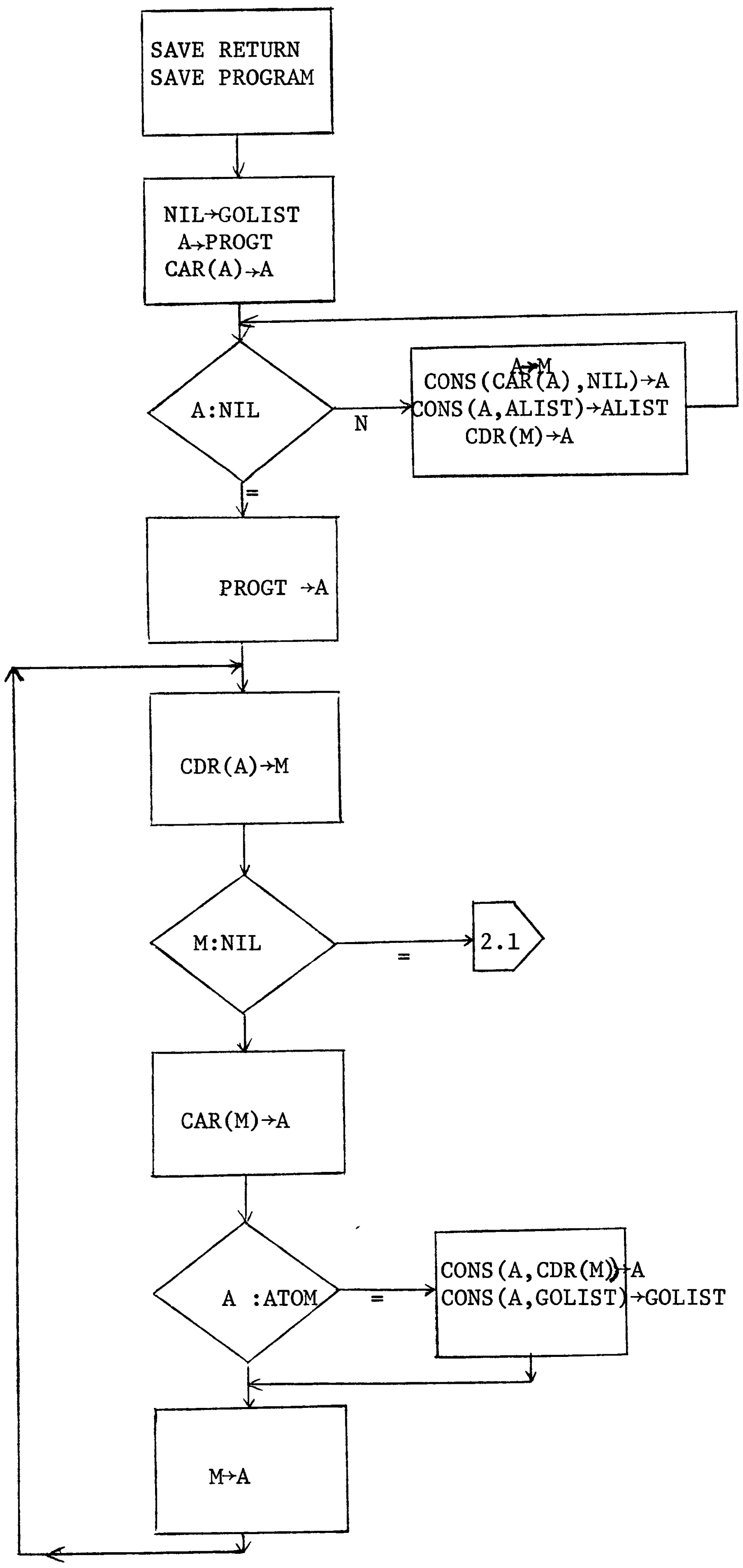
```
evlis[m;a] = prog[[w];
             w: = NIL;
   EVLISS    w: = nconc[w;cons[eval[car[m];a];NIL]];
             m: = cdr[m];
             [null[m] → return[w]];
             go[EVLISS] ];


evcon[c;a] = prog[[];
   EVCONN    [null[c] → [progind → return [c];
                         T → error[A3]]];
             [eval[caar[c];a] → return[eval[cadar[c];a]]];
             c = cdr[c];
             go[EVCONN];


define  [x] = deflist  [x;EXPR]
deflist  [x;ind] = prog[[m]
                     m: = x;
             A       remprop[caar[m];ind];
                     replacd[caar[m];cons[ind;cons[cadar[m];cdaar[m]]]];
                     replaca[m;caar[m]];
                     m: = cdr[m];
                     [null[m] → return[x]];
                     go[A]];
```

GARBAGE COLLECTOR
Chart 1



Garbage Collector
Chart 2

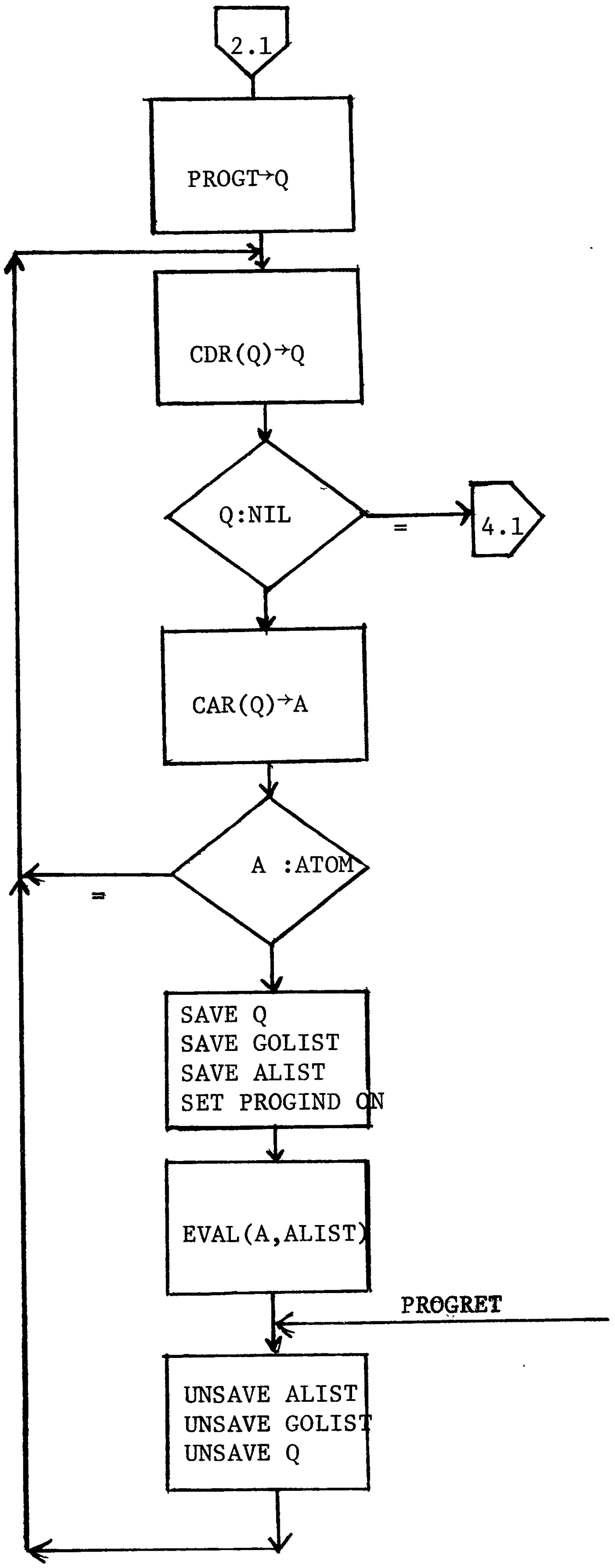




PROG Chart 1

The PROG variables are added to the ALIST.

The GOLIST is constructed.



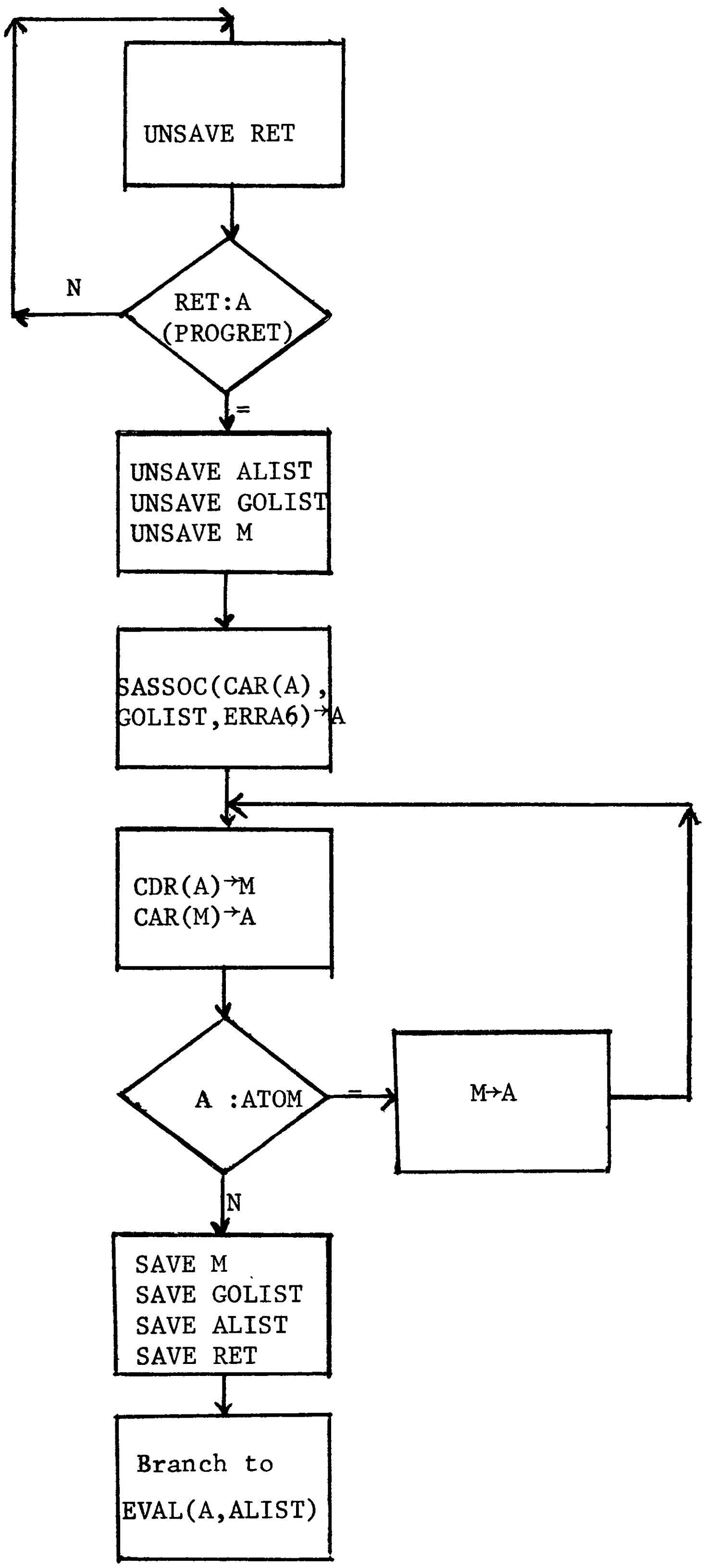
PROG Chart 2

The program is executed.

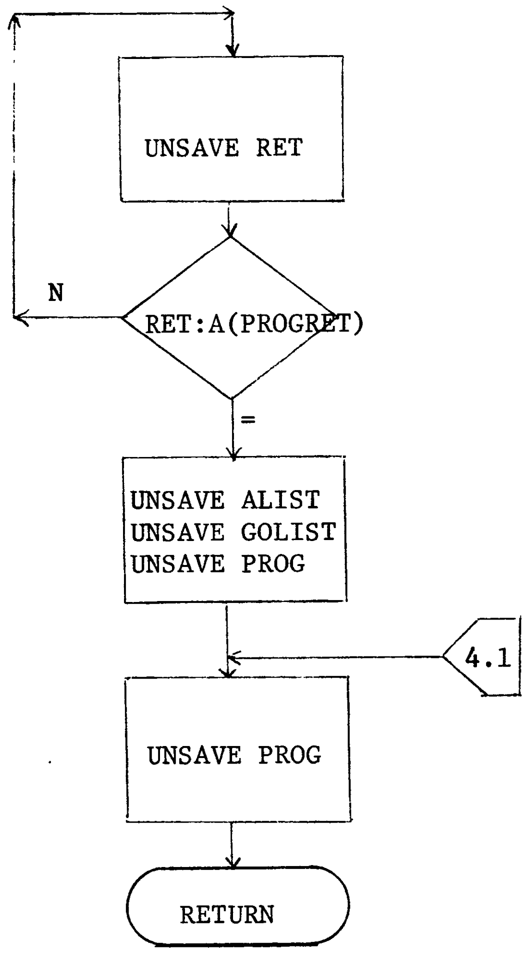
NOTE: At the call on EVAL the PDS contains the return address (PROGRET). This is used in GO and RETURN.

PROG Chart 3

Execution of GO



PROG Chart 4



Execution of RETURN

APPENDIX II

A sample Lisp run showing the initial object list and PRETTYPRINT printing itself.

```
//
//LISP   JOB   C0010J.F.BOLCE,LISP,MSGLEVEL=1                                      JOB 294
//JOBLIB DD DSNAME=LISP,DISP=OLD
//  EXEC   PGM=LISP
//LISPLIB DD DSNAME=LISPLIB,DISP=OLD
//SYSPRINT  DD   SYSOUT=A
//SYSIN  DD  *
IEF236I ALLOC. FOR LISP
IEF237I JOBLIB    ON 233
IEF237I LISPLIB   ON 233
IEF237I SYSIN     ON 00C


   26975 LISP CELLS ALLOCATED.
COLLECTING

 ARGUMENTS FOR EVALQUOTE ...
   EVAL
   (OBLIST NIL)

 TIME    .   0MS,  VALUE IS ...
   (NIL CAR CDR QUOTE CONS EVAL DEFINE EQ EQUAL ATOM LEFTSHIFT DIFFERENCE REMAINDER QUOTIENT NULL
   ADD1 SUB1 MINUS PLUS TIMES COND LAMBDA T APPEND PROG GO RETURN SET CSET CSETQ SETQ CADR CDDR CAAR
   CDAR CADDR CADAR CAADR PRINT READ F GET MEMBER EVLIS NCONC PAIR APPLY APPEND1 APVAL EXPR SUBR FEXPR
   FSUBR LABEL FUNARG ERRB DOTERR1 DOTERR2 - + AND OR LOGOR LOGAND LOGXOR EVENP MINUSP ZEROP LESSP
   GREATERP ERROR XTAB TTAB NOT FIXP FLOATP LIST LOGP PRIN1 TERPRI DEFLIST REMPROP FUNCTION ATTRIB
   [illegible]RO32 NUMBERP RPLACA RPLACD OBLIST LIBRARY GENSYM EXPT UNPACK ADVANCE ENDREAD STARTREAD UNTRACE
   [illegible]RACE PACK CLEARBUFF MKATOM CCLASS SASSOC)

 ARGUMENTS FOR EVALQUOTE ...
   LIBRARY
   ((PEVAL))

 TIME        0MS,  VALUE IS ...
   (PEVAL)

 ARGUMENTS FOR EVALQUOTE ...
   EVAL
   ((DEFINE (READ)) NIL)

 TIME     1098MS,  VALUE IS ...
   (PEVAL PRETTYPRINT SUPERPRINT ENDLINE)

 ARGUMENTS FOR EVALQUOTE ...
   CSET
   (RPAR ))

 TIME        0MS,  VALUE IS ...
   (APVAL ()))

 ARGUMENTS FOR EVALQUOTE ...
   CSET
   (LPAR ()

 TIME        0MS,  VALUE IS ...
   (APVAL (())

 ARGUMENTS FOR EVALQUOTE ...
```

```
PRETTYPRINT
((PRETTYPRINT SUPERPRINT ENDLINE))

(PRETTYPRINT
   (LAMBDA (L) (PROG (TT I)
        A      (COND
                  ((NULL L) (RETURN NIL)))
               (TERPRI)
               (PRIN1 LPAR)
               (PRINT (CAR L))
               (SETQ I 3)
               (XTAB 3)
               (SUPERPRINT (COND
                  ((SETQ TT (GET (CAR L) (QUOTE EXPR))) TT)
                  ((SETQ TT (GET (CAR L) (QUOTE FEXPR))) TT)
                  (T (QUOTE UNDEFINED))))
               (PRINT RPAR)
               (SETQ L (CDR L))
               (GO A))))

(SUPERPRINT
   (LAMBDA (E) (COND
        ((ATOM E) (PRIN1 E))
        (T (PROG (EP M)
                  (SETQ EP E)
                  (PRIN1 LPAR)
           A      (COND
                     ((MEMBER (CAR EP) (QUOTE (AND
                        OR
                        LIST
                        PLUS
                        TIMES
                        COND
                        IF
                        SELECT
                        MAX
                        MIN
                        PROG2))) (GO PL))
                     ((EQ (CAR EP) (QUOTE PROG)) (GO PP)))
                  (SUPERPRINT (CAR EP))
                  (SETQ EP (CDR EP))
                  (COND
                     ((NULL EP) (RETURN (PRIN1 RPAR)))
                     ((ATOM EP) (GO PD)))
                  (XTAB 1)
                  (GO A)
           PK     (SETQ I (SUB1 I))
           PD     (PRIN1 (QUOTE  . ))
                  (PRIN1 EP)
                  (RETURN (PRIN1 RPAR))
           PL     (SETQ I (ADD1 I))
                  (SUPERPRINT (CAR EP))
           PM     (SETQ EP (CDR EP))
                  (COND
                     ((NULL EP) (GO PJ))
                     ((ATOM EP) (GO PK)))
                  (ENDLINE)
                  (SUPERPRINT (CAR EP))
                  (GO PM)
           PJ     (SETQ I (SUB1 I))
```

```
                (RETURN (PRIN1 RPAR))
          PP    (PRIN1 (CAR EP))
                (SETQ EP (CDR EP))
                (SETQ I (ADD1 I))
                (COND
                   ((NULL EP) (GO PJ))
                   ((ATOM EP) (GO PK)))
                (XTAB 1)
                (SUPERPRINT (CAR EP))
          PY    (SETQ EP (CDR EP))
                (COND
                   ((NULL EP) (GO PJ))
                   ((ATOM EP) (GO PK)))
                (ENDLINE)
                (COND
                   ((ATOM (CAR EP)) (GO PZ)))
                (XTAB 6)
          PX    (SETQ I (PLUS
                   I
                   2))
                (SUPERPRINT (CAR EP))
                (SETQ I (PLUS
                   I
                   -2))
                (GO PY)
          PZ    (PRIN1 (CAR EP))
                (TTAB (TIMES
                   (PLUS
                      I
                      2)
                   3))
                (SETQ EP (CDR EP))
                (COND
                   ((NULL EP) (GO PJ))
                   ((ATOM EP) (GO PK))
                   ((ATOM (CAR EP)) (GO PZ)))
                (GO PX))))))

 (ENDLINE
    (LAMBDA NIL (PROG NIL
                (TERPRI)
                (TTAB (TIMES
                   I
                   3))))))

TIME     1223MS,  VALUE IS ...
 NIL

ARGUMENTS FOR EVALQUOTE ...
 DEFINE
 (((FACTORIAL (LAMBDA (N) (COND ((ZEROP N) 1) (T (TIMES N (FACTORIAL (SUB1 N)))))))))

TIME        0MS,  VALUE IS ...
 (FACTORIAL)

ARGUMENTS FOR EVALQUOTE ...
 TRACE
 (T)

TIME        0MS,  VALUE IS ...
```

```
  T

 ARGUMENTS FOR EVALQUOTE ...
   FACTORIAL
   (2)
*** ENTERING APPLY WITH FN-        FACTORIAL
*** AND ARGS-                      (2)
*** ENTERING APPLY WITH FN-        (LAMBDA (N) (COND ((ZEROP N) 1) (T (TIMES N (FACTORIAL (SUB1 N)))
   )))
*** AND ARGS-                      (2)
*** ENTERING EVAL WITH FORM-       (COND ((ZEROP N) 1) (T (TIMES N (FACTORIAL (SUB1 N)))))
*** ENTERING EVAL WITH FORM-       (ZEROP N)
*** ENTERING EVAL WITH ATOM-       N=2
*** ENTERING EVAL WITH ATOM-       T=T
*** ENTERING EVAL WITH FORM-       (TIMES N (FACTORIAL (SUB1 N)))
*** ENTERING EVAL WITH ATOM-       N=2
*** ENTERING EVAL WITH FORM-       (FACTORIAL (SUB1 N))
*** ENTERING EVAL WITH FORM-       (SUB1 N)
*** ENTERING EVAL WITH ATOM-       N=2
*** ENTERING APPLY WITH FN-        (LAMBDA (N) (COND ((ZEROP N) 1) (T (TIMES N (FACTORIAL (SUB1 N)))
   )))
*** AND ARGS-                      (1)
*** ENTERING EVAL WITH FORM-       (COND ((ZEROP N) 1) (T (TIMES N (FACTORIAL (SUB1 N)))))
*** ENTERING EVAL WITH FORM-       (ZEROP N)
*** ENTERING EVAL WITH ATOM-       N=1
*** ENTERING EVAL WITH ATOM-       T=T
*** ENTERING EVAL WITH FORM-       (TIMES N (FACTORIAL (SUB1 N)))
*** ENTERING EVAL WITH ATOM-       N=1
*** ENTERING EVAL WITH FORM-       (FACTORIAL (SUB1 N))
*** ENTERING EVAL WITH FORM-       (SUB1 N)
*** ENTERING EVAL WITH ATOM-       N=1
*** ENTERING APPLY WITH FN-        (LAMBDA (N) (COND ((ZEROP N) 1) (T (TIMES N (FACTORIAL (SUB1 N)))
   )))
*** AND ARGS-                      (0)
*** ENTERING EVAL WITH FORM-       (COND ((ZEROP N) 1) (T (TIMES N (FACTORIAL (SUB1 N)))))
*** ENTERING EVAL WITH FORM-       (ZEROP N)
*** ENTERING EVAL WITH ATOM-       N=0

 TIME      23MS,  VALUE IS ...
   2

 ARGUMENTS FOR EVALQUOTE ...
   UNTRACE
   (T)
*** ENTERING APPLY WITH FN-        UNTRACE
*** AND ARGS-                      (T)

 TIME       0MS,  VALUE IS ...
   T

 ARGUMENTS FOR EVALQUOTE ...
   TRACE
   ((FACTORIAL))

 TIME       0MS,  VALUE IS ...
   NIL

 ARGUMENTS FOR EVALQUOTE ...
```

```
    FACTORIAL
    (5)
TRACE ENTERING-         FACTORIAL(4)
TRACE ENTERING-         FACTORIAL(3)
TRACE ENTERING-         FACTORIAL(2)
TRACE ENTERING-         FACTORIAL(1)
TRACE ENTERING-         FACTORIAL(0)
TRACED FUNCTION/VALUE- FACTORIAL 1
TRACED FUNCTION/VALUE- FACTORIAL 1
TRACED FUNCTION/VALUE- FACTORIAL 2
TRACED FUNCTION/VALUE- FACTORIAL 6
TRACED FUNCTION/VALUE- FACTORIAL 24

 TIME       10MS,   VALUE IS ...
    120

  ARGUMENTS FOR EVALQUOTE ...
    DEFINE
    (((TESTS (LAMBDA NIL (PROG (U) (SETQ U (STARTREAD)) C (PRINT U) (PACK U) (COND ((CCLASS U (QUOTE
    C)) (RETURN (MKATOM)))) (SETQ U (ADVANCE)) (GO C))))))

  TIME        0MS,   VALUE IS ...
    (TESTS)

  ARGUMENTS FOR EVALQUOTE ...
    TESTS
    NIL
    [illegible]
    [illegible]
    S
    G
    H
    J
    K
    P
    Q
    R
    R
    T
    T
    Y
    C

 TIME       24MS,   VALUE IS ...
    ADSGHJKPQRRTTYC
*** END OF DATA
```

Appendix III

The following list are those functions that have been implemented. Those marked by an asterisk are new functions or differ from the function as described in McCarthy [1].

addl[n]

* advance [ ]

and $[x_1;x_2;\ldots;x_n]$

* append [x;y]

appendl [x;y]

apply [fn;args;a]

atom [x]

attrib [x;y]

car [x]

caar [x]

cadr [x]

cddr [x]

caddr [x]

cadar [x]

caadr [x]

cdr [x]

* cclass [x;y]

clearbuff [ ]

cond $[p_1 \rightarrow e_1;p_2 \rightarrow e_2;\ldots;p_k \rightarrow e_k]$

cons [x;y]

cset [x;y]

csetq [x;y]

define [x;y]

deflist [x;ind]

difference [x;y]

endread [ ]

eq [x;y]

equal [x;y]

error [fn]

eval [form;a]

evenp [n]

evcon [x;a]

evlis [x;a]

expt [n;c]

fixp [n]

floatp [n]

function [x]

gensym [ ]

get [x;ind]

go [label]

greaterp [x;y]

label [name;fn]

leftshift [m;n]

lessp [x;y]

* library [x]

list [x_1;x_2;...x_k]

logand [n_1;n_2;...;n_k]

logor [n_1;n_2;...;n_k]

logp [n]

logxor [n_1;n_2;...;n_k]

member [x;y]

minus [n]

minusp [n]

* mkatom [ ]

nconc [x;y]

not [x]

null [x]

numberp [x]

or [x_1;x_2;...;x_k]

* pack [x]

pair [x;y]

plus [x_1;x_2;...;x_k]

prin1 [x]

print [x]

prog [p;a]

prog2 [x;y]

quote [x]

quotient [x;y]

read [ ]

remainder [n;m]

remprop [x,ind]

rplaca [x;y]

rplacd [x;y]

sassoc [x;a;fn]

set [x;y]

setq [x;y]

* startread [ ]

subl [n]

terpri [ ]

times [n_1;n_2;...;n_k]

* trace [n]

* ~~t~~tab [n]

* unpack [x]

* untrace [x]

* xtab [n]

zerop [n]

REFERENCES

1. John McCarthy etal., Lisp 1.5 Programmer's Manual, Cambridge, Massachusetts, MIT Press, 1962.

2. Jan Kent, An Interpretive System for the Programming of Recursive Functions on a Digital Computer, Thesis in Mathematics, University of Oslo (1966).

3. OS/360 Supervisor and Data Management Macro-Instructions, IBM Manual C28-6647

4. System/360 Principles of Operation IBM Manual A22-6821

5. C. Weissman, Lisp 1.5 Primer Dickenson Publishing Co. Inc.

6. E.C. Berkeley, D.G. Bobrow (eds.), The Programming Language Lisp: Its Operation and Applications. Cambridge, Massachusetts, MIT Press, 1966

7. K. Korsvoll, An Online Algebraic Simplify Program, Memo 37, Stanford Intelligence Project, Stanford (1965)