TRM-2
April, 1968

THE UNIVERSITY OF TEXAS

6400/6600 LISP 1.5

an Adaptation of

MIT LISP 1.5

Revision II, April 2, 1968

by

James B. Morris, Jr.

Don J. Singleton

Sponsored by

Dr. W.W. Bledsoe

THE UNIVERSITY OF TEXAS AT AUSTIN

COMPUTATION CENTER

TABLE OF CONTENTS

THE UNIVERSITY OF TEXAS

6400/6600 LISP 1.5

An Adaptation of MIT LISP 1.5

I Introduction

In approximately 1961, a group of computer scientists under the direction of Professor John McCarthy at MIT developed a LISt Processing language called LISP. The concept of LISP was introduced in a paper by McCarthy entitled "Recursive Functions of Symbolic Expressions and Their Computation by Machine," which was published in Communications of the ACM April 1960; this language was to be the forerunner of the present LISP 1.5 programming system. Its use in methods of analysis of symbolic expressions rather than numerical expressions has grown rapidly since 1961. Perhaps the most striking example of manipulation of symbolic expressions comes from a report by Hearn (1) at Stanford which describes the use of LISP for symbolic calculation of particle scattering properties in high energy physics. Hearn reports that "Six man-months of effort was reduced to fifteen minutes of 7094 computer time through the use of LISP." Slagle (6), at MIT, reported on a LISP program which does heuristic symbolic integration of complex calculus integrals. Other uses of LISP include symbolic differentiation programs (2,3), simplification of symbolic arithmetic expressions (4,5), and many other uses in electrical circuit theory (4,5), mathematical logic, game playing, and other fields of artificial intelligence

In September of 1966, a project sponsored by Professor W. W. Bledsoe was begun to develop a LISP 1.5 interpreter system for the Control Data 6000 computer series. The completion of the system is anncunced with the publishing of this document.

The standard user's manual for acquiring a knowledge of the use of the LISP 1.5 language is the LISP 1.5 Programmer's Manual available for reference in The University of Texas Computation Center library, or from the MIT Press, Cambridge, Massachusetts, at a price of $3.00. A second publication, The Programming Language LISP: Its Operation and Applications, is also available from the MIT Press for $5.00 and is suggested as interesting reading for those already familiar with the LISP language. The METEOR interpreter, also available at The University of Texas and running under the LISP interpreter, is explained in the above publication.

The purpose of this document is to point out the differences between 6400/6600 LISP and MIT LISP. It is assumed that the reader has a knowledge of LISP or has the LISP 1.5 Programmer's Manual available for reference.

II List Structures

With the exception of the property list, the representation of list structures given on page 36 of the LISP 1.5 Programmer's Manual applies to 6400/6600 LISP. In this system, the addresses (CAR and CDR) are each 18 bits in length. A 6-bit identification field (used by the interpreter, but not available to the programmer) has been set up to identify the use of any particular 6400/6600 word, i.e., whether it is an atom, an element in the property list, a list element, etc. Since a 6400/6600 word contains 60 bits, this leaves 18 bits otherwise unused (i.e., unused in the list structure--they are used in the property list). Two functions have been created to allow the LISP programmer to reference this "special address" in each list element. The function "CSR" has as its value the contents of the "special address" of the list element specified by its argument. The corresponding function RPLACS will store an address in the "special address" portion of any word in a list, just as RPLACA and RPLACD store in the CAR and CDR portions of a word.

The property lists in 6400/6600 LISP are structured somewhat differently from MIT LISP property lists (see Figures 1 and 2).* Because of the availability of a third address, the 6400/6600 system is able to use one word where the MIT system needs two words. This results in a considerable gain in property list search time, which is a critical factor in the execution time required by the interpreter. In addition, a 2-to-1 savings in memory space is gained for property list storage. Thus, for large LISP programs such as METEOR, this is a distinct advantage.

One 6400/6600 computer word contains a complete unit of information about an atom. The property list is made up of many of these units of information. Each of the three address portions in the computer word (or list cell) plays a different role. The CSR portion of the list cell identifies the property type (denoted as PCODE in Figure 2) which this cell specifies. This property type will be an atom

---

In these figures, a "" in the CAR portion of the first list cell of an atomic symbol property list denotes "address of the cell in which the '*' appears."

FIGURE 1

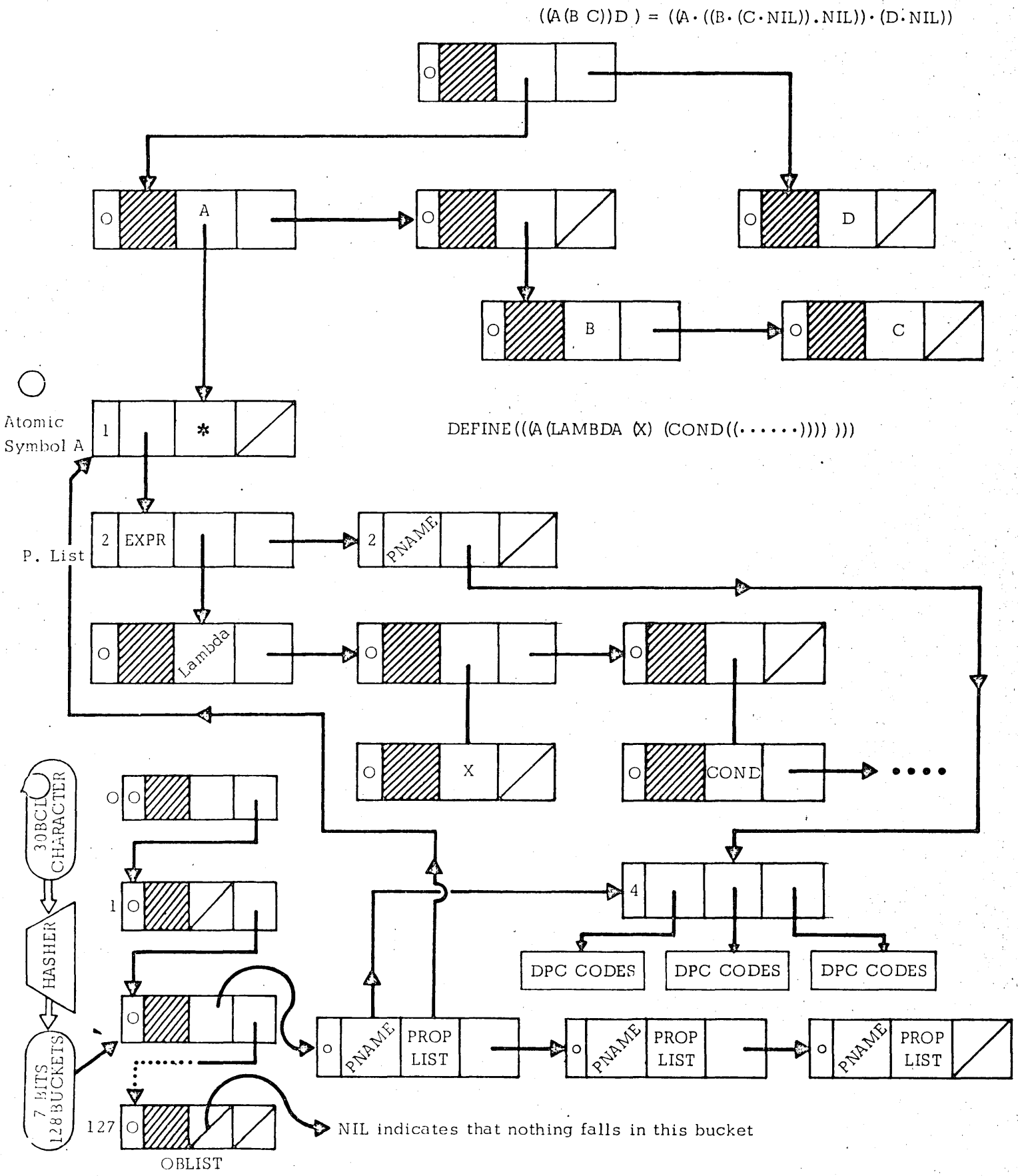
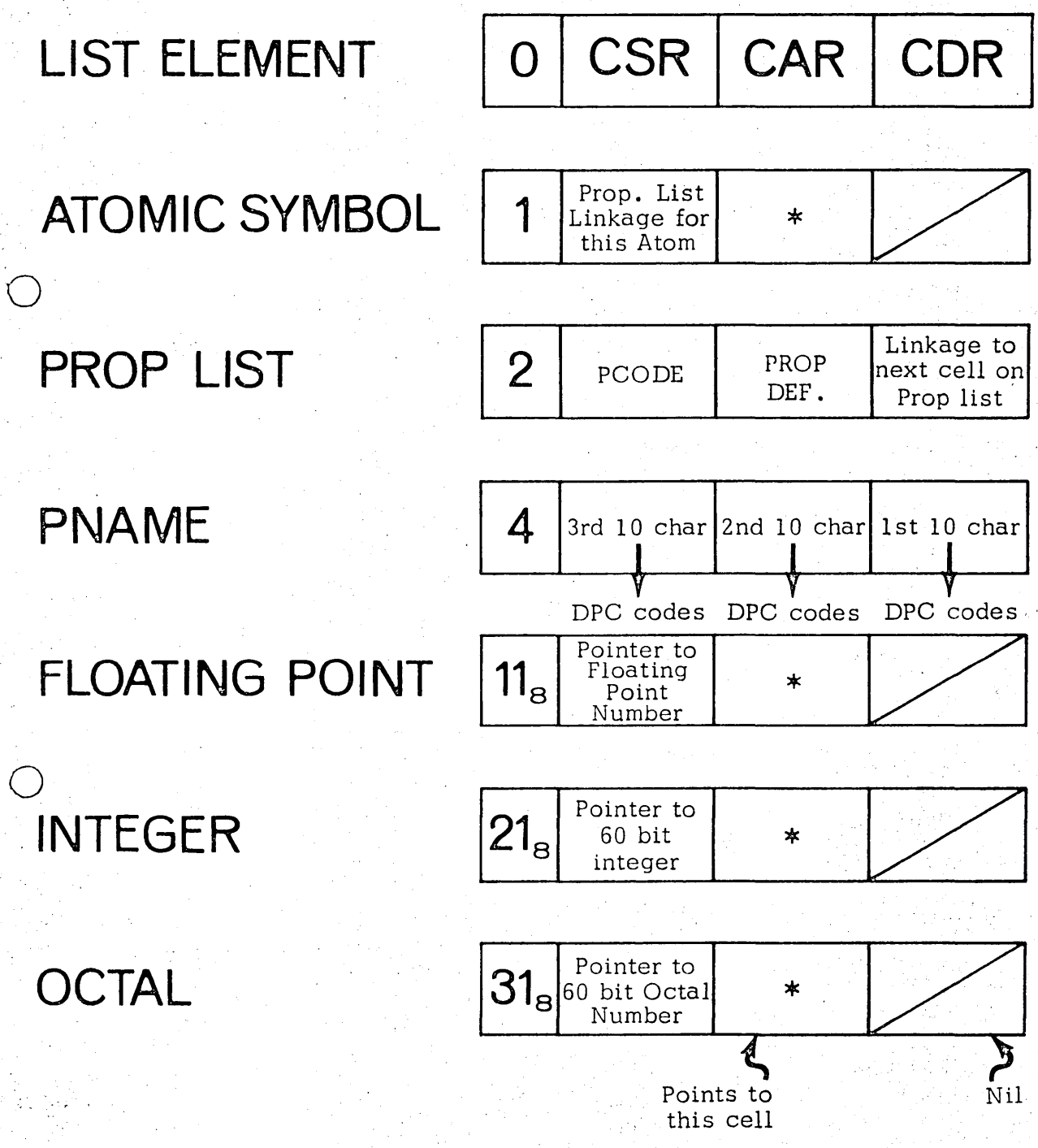

((A(B C))D) = ((A·((B·(C·NIL)).NIL))·(D·NIL))
O
A
O
D
B
C
Atomic Symbol A
1
*
DEFINE(((A(LAMBDA (X) (COND((·····)))))))
P. List
2
EXPR
2
PNAME
Lambda
X
COND
30BCL CHARACTER
HASHER
7 BITS 128 BUCKETS
1
4
DPC CODES
DPC CODES
DPC CODES
PNAME
PROP LIST
PNAME
PROP LIST
PNAME
PROP LIST
127
OBLIST
NIL indicates that nothing falls in this bucket

FIGURE 2


LIST ELEMENT
0
CSR
CAR
CDR
ATOMIC SYMBOL
1
Prop. List Linkage for this Atom
*
PROP LIST
2
PCODE
PROP DEF.
Linkage to next cell on Prop list
PNAME
4
3rd 10 char
2nd 10 char
1st 10 char
DPC codes
DPC codes
DPC codes
FLOATING POINT
11₈
Pointer to Floating Point Number
*
INTEGER
21₈
Pointer to 60 bit integer
*
OCTAL
31₈
Pointer to 60 bit Octal Number
*
Points to this cell
Nil

such as PNAME, EXPR, FEXPR, SUBR, FSUBR, APVAL, etc. The CAR portion of the list cell contains a pointer to the property definition for the specified property. For EXPR's and FEXPR's this is a pointer to the list structure for a LISP function; for SUBR's and FSUBR's this is a pointer to a machine language routine; for PNAME it is a pointer to a cell which points to a print name. The CDR portion of the list cell will contain a link to the next cell on the property list, or NIL in the case of the last cell.

The implementation of the garbage collector in 6400/6600 LISP is essentially the same as that in MIT LISP. The A0 register of the computer always contains a pointer to the top of the free-storage list. In the garbage collector, active list structures are marked by linking to all cells which can be reached from car-cdr-csr chains beginning in special header cells. These special header cells are: the OBLIST, the pushdown stack, the a-list, all A, B, and X registers, the TEMLIS (temporary lists), and doublets which remain to be executed. Marking in full word space is accomplished by a bit table.

When garbage collection occurs, the following message will be printed in the event that the programmer has included a "G" in the list of parameters on the LISP control card:

```
FULL FREE = WORDS COLLECTED BY GARBAGE COLLECTOR
nnnn mmmm
```

where

nnnn is the number of words collected in full word space,

mmmm is the number of words collected in free-storage space.

III. Available System Functions

The following functions (not including the I/O functions listed in Section IV) are available to the LISP programmer and operate exactly as explained in the LISP 1.5 Programmer's Manual:

ADD1	DIGIT	GREATERP	NULL	RPLACA
AND	DIVIDE	LABEL	NUMBERP	RPLACD
APPEND	EFFACE	LEFTSHIFT	ONEP	RPLACS
APPLY	EQ	LENGTH	OR	SASSOC
ATOM	EQUAL	LESSP	PAIR	SEARCH
ATTRIB	ERROR	LIST	PLUS	SELECT
CAR	ERRORSET	LOGAND	PROG	SET
CDR	EVAL	LOGOR	PROG2	SETQ
CSR	EVALQUOTE	LOGXOR	PROP	SPEAK
CONC	EVLIS	MAP	QUOTE	SUB1
COND	FIX	MAPCON	QUOTIENT	SUBLIS
CONS	FIXP	MAPLIST	RECIP	SUBST
COPY	FLAG	MAX	RECLAIM	TEMPUS
COUNT	FLOATP	MEMBER	REMAINDER	TIMES
CSET	FUNCTION	MIN	REMFLAG	TRACE
CSETQ	GENSYM	MINUS	REMOB	TRACESET
DEFINE	GET	MINUSP	REMPROP	UNCOUNT
DEFLIST	GETPN	NCONC	RETURN	UNTRACE
DIFFERENCE	GO	NOT	REVERSE	UNTRACESET
				ZEROP

The following function names are unique to this LISP system and are explained in detail:

tempus[ ] - A function of no arguments which returns as value the elapsed program time in seconds as a floating-point number. (Caution: Calling this function a large number of times will result in large amounts of pp usage time since a pp routine is involved in the implementation of this function.)

alist[ ] - A function of no arguments which returns as value the current association list. (See page 17 in [3].)

gradp[a;b] This function has two arguments, a and b, both of which must be atomic symbols. GRADP returns *T* if the core memory address at which a is stored is greater than the core memory address at which b is stored, NIL otherwise. This function may be used as a fast, arbitrary ordering function. It may be assumed that the core memory locations of atomic symbols do not change during the entire run.

alphap[a;b] This function has two arguments, a and b, both of which must be atomic symbols. alphap returns *T* if a precedes b in alphabetical order, NIL otherwise (using display code sequence as collating sequence).

random[n] - This function is concerned with (pseudo-) random number generation and operates in the following manner:

- $n<0$ returns the last random number which was generated, and does not generate a new random number. Thus random may be called with $n<0$ and not change the state of the generator.
- $n=0$ causes generation of a new random number X, $0.0 \leq X \leq 1.0$. The value X is returned.
- $0<n<1$ n is the new seed from which subsequent random sequences will be formed. This option allows the user to start a new random sequence beginning with n. The value returned is n.

In addition, it should be mentioned at this time that the operation of several of the functions which manipulate property lists is peculiar to 6400/6600 LISP due to the methods of packing atomic property list structures (see Section II). These peculiarities are explained below.

CSR and RPLACS functions are available (see Section II) for manipulation of the packed list structures.

ATRRIB expects its second argument to be a packed property list structure. This is added on to the end of the packed property list structure given as the first argument.

GET expects its first argument to be a packed property list structure. One difference, however, is that GET should not be used to "get" a print name of an atom. Instead, the programmer should use

(GETPN X)

which, under the MIT LISP system would be called by

(GET (QUOTE PNAME) X)

GETPN will return a full word list of the print name.

PROP must be given a packed property list structure as its first argument. It returns a packed property list structure as value.

REMPROP, FLAG, REMFLAG, CSET, CSETQ, DEFINE, and DEFLIST all search a packed property list structure and must be given a structure of this type as their first argument.

The user must supply his own routines to manipulate the normal list structures if he intends to utilize an equivalent operation as normally performed in the MIT LISP system by one of these functions with the above-mentioned peculiarity.

IV. LISP Input and Output

The LISP input/output routines allow the user running under the SCOPE operating system to read from files named INPUT (the standard input file) and up to two arbitrary files. It also allows writing to files named OUTPUT (the standard output file) and up to two arbitrary files which need not necessarily be the same arbitrary files as used with read operations.

The following MIT LISP input/output routines are available in 6400/6600 LISP and are explained in the LISP 1.5 Programmer's Manual :

PRINT	PACK
PRIN1	UNPACK
TERPRI	OPCHAR
READ	DIGIT
CLEARBUFF	LITER
ENDREAD	NUMOB
ADVANCE	MKNAM
STARTREAD	INTERN

The "Class A" as mentioned on page 83 has been expanded to include the following characters due to the SCOPE 2.0 63-character set:

Class A ABCDEFGHIJKLMNOPQRSTUVWXYZ=*/$[]↑∨∧↑≠≤>;:≦≧¬

The section "Characters and Character Objects" has been changed in the following ways:

(1) A 63-character set and the character "EOR" are the 64 characters available in 6400/6600 LISP. The extra 16 characters not mentioned in the LISP 1.5 Manual, except for the "↓", may be used anywhere that an alphabetic character may be used. The "↓" character is discussed on page 17 of this document.

(2) Numeric character objects may not be used in arithmetic. Sending the character object through NUMOB will solve this problem, since NUMOB always returns a number, even for the digits 0-9.

(3) The new special character objects are as follows:

Name	Character Representation
EQUIV	≡
LBRACK	[
RBRACK	]
COLON	:
NEQUAL	≠
RARROW	→
ORSIGN	∨
ANDSIGN	∧
UPARROW	↑
DARROW	↓
LESS	<
GREATER	>
LESSEQ	≤
GREATEREQ	≥
NOTSIGN	¬
SEMICOLON	;

The atomic symbols CURCHAR, CHARCOUNT, and CPPI are not available.

The following functions have been added to the already existent standard I/O functions in the 6400/6600 LISP system. In each case, the right-hand column specifies an MIT LISP function. In the corresponding left column is the name of a 6400/6600 LISP function which will produce the same results except input/output is to some arbitrary programmer-defined file given as argument to the function. The normal input/output routines as described in the LISP 1.5 Programmer's Manual send information to system files named INPUT or OUTPUT, whichever the case may be.

6400/6600 LISP	MIT LISP
(OUTPUT (QUOTE ARB) X)	(PRINT X)
(OUTPUT1 (QUOTE ARB) X)	(PRINT1 X)
(ARBTERPRI (QUOTE ARB))	(TERPRI)
(ARBENDREAD (QUOTE ARB))	(ENDREAD)
(INPUT (QUOTE ARB))	(READ)
(ARBADVANCE (QUOTE ARB))	(ADVANCE)
(ARBSTART (QUOTE ARB))	(STARTREAD)

where ARB is some user defined file (limit of two). Also, the function call

(OUTPUT (QUOTE OUTPUT) X)

is equivalent to

(PRINT X)

The same is true for all of the "arbitrary file" functions. In addition, the functions

(REWIND (QUOTE ARB)) and (ENDFILE (QUOTE ARB))

will rewind and write an end-of-file, respectively, on file ARB.

The format of all input/output files is 512-word odd-parity records (binary records).

V. Running the LISP System

The 6400/6600 LISP system operates under the standard SCOPE operating system in a batch mode of operation. This section describes the deck and control card setup necessary for running a LISP program.

Deck setup

(1) The first card in the deck is a comment card and is not processed by the interpreter.

(2) The next series of cards is a sequence of doublets for EVALQUOTE. Card format is free field and only the first 72 columns of the card are processed. Atomic symbols do not continue over to the next card, even if the atomic symbol ends in column 72.

(3) STOP)))) card. The number of right parentheses is optional, but at least one must be present.

(4) Repeat (2) and (3) as many times as desired. Each series of statements between STOP cards is called a "packet" and the first packet is executed by the interpreter before discovering that the second packet (and succeeding packets) exists. Function and other definitions carry over to each new packet.

(5) The last card in the deck must be a card with FIN punched on it in any column

Control card setup

The interpreter automatically assigns the available memory in the user's field length, after loading of the interpreter, to available space lists. Approximately 80% is used for the free storage list and the remaining for the full word list. Thus a user need only specify the field length on his JOB card and the interpreter will see that the entire amount of memory is utilized. The field length must contain room for the interpreter over and above the amount of available free storage space required by the program. The current size of the interpreter is $\approx 9700_{10}$ words ($\approx 22730_8$ words).

(a) The first control card is the JOB card and is explained in the SCOPE Reference Manual.

(b) The second control card calls the LISP interpreter and appears as

LISP(L,G)

beginning in column one. If the L is omitted as in

LISP(G)

a source deck listing of the LISP program is suppressed. Similarly, if the G is omitted, garbage collector messages are suppressed. Both parameters may be omitted by a card which contains only

LISP.

It is also possible to obtain a listing in which a parenthesis count is inserted under left and right parentheses in the listing (except within comments). This parenthesis-count option is an invaluable feature for preliminary LISP debug runs since mismatched parentheses may be detected very easily from the source deck listing.

This option is obtained through the use of a P parameter in the LISP control card. Thus,

LISP(P,G) is the same as LISP(L,G)

or

LISP(P) is the same as LISP(L)

except that the parenthesis count is added to the source deck listing.

(c) The third card is an end-of-record card.

Following the above is the LISP deck. Of course the control card setup shown is for a normal run only, and other SCOPE control cards, such as for core dumps, etc., may be added.

VI. The LISP Library

A feature which has recently been added to the LISP system is the LISP library. By means of this feature, symbolic LISP functions may be stored on a permanent file and directly loaded into the system for subsequent use within the interpreter. The library file may be maintained through the use of special LISP functions as explained below.

Organization. The LISP library itself is a file named LISPLIB. The LISPLIB file is a series of logical records (each of which is a LISP list structure) in the following format:

(record-name logical-record-information)

The record-name specifies the name of the logical record, and the logical-record-information contains the record information.

The format of the LISPLIB file is as follows:

logical record 1: (LISPLIB:LOADER:PROGRAM loader function)
logical record 2: (LISPLIB:CATALOG:PROGRAM catalog function)
logical record 3: (LISPLIB:LIBLIST:PROGRAM liblist function)
" " 4: (LISPLIB:UPDATE:PROGRAM update function)
" " 5: (LISPLIB:DELETE:PROGRAM delete function)
" " 6: (LISPLIB:LIBCOPY:PROGRAM libcopy function)
" " 7: (LISPLIB:DIRECTORY($(rn_k.k)\ldots(rn_2.2)(rn_1.1)$))

where rn_j is the name of the j^{th} logical record following the LISPLIB directory

logical record $6+j$ [j goes from 1 to k]: (name of the j^{th} record (one or more named LISP function s-expression(s)))

For example,

```
(EXAMPLE*FUNCTIONS((EJECT (LAMBDA () (PROG2 (PRIN1 1)(TERPRI))))
   (FF (LAMBDA (L) (COND ((ATOM L) L) (T (FF (CAR L))))))))
```

Use and maintenance. The LISPLIB library tape is filed in The University of Texas Computation Center tape vault under tape no. 510/0473. The following LISP functions are available to enable the user to maintain and use the LISP library:

LOAD(L)

L is a list of record names. All the named S-expression functions in each record given in the list L are DEFINEd for the current LISP run. The value returned is a list of all function names which were defined.

Example:

```
LOAD((EXAMPLE*FUNCTIONS))
will DEFINE the functions EJECT and FF
value is: (EJECT FF)
```

CATALOG(LIB)

LIB is the name of a library file. This program lists the name of all records in the file. The value returned is a list of all record names in the file.

Example:

```
CATALOG(LISPLIB)
value: (list of all record names, e.g., EXAMPLE*FUNCTIONS)
```

LIBLIST(L)

L is a list of record names. This program lists all function S-expressions in each record given in the list L. Value returned is NIL.

Example:

```
LIBLIST((EXAMPLE*FUNCTIONS))
will list the functions EJECT and FF
```

UPDATE(L)

L is a list of logical records to be added to the end of the current LISP library. If any duplicate record names are encountered, an error message occurs and the new duplicated record is not added to the library. The new LISP library is produced on a file called NEWLIB. Value returned is *T* if no duplicate record names were found on the library, otherwise NIL.

Example:

```
UPDATE((
   (EXAMPLE*FUNCTIONS((EJECT(LAMBDA()
   (PROG2(PRIN1 1)(TERPRI))))
   (FF (LAMBDA (L) (COND (ATOM L)L)
   (T(FF(CAR L))))))))  ))
```

DELETE(L)

L is a list of logical records to be deleted from the current LISP library. The new LISP library is produced on a file called NEWLIB. Value returned is L.

Example:

```
DELETE((EXAMPLE*FUNCTIONS))
value:  (EXAMPLE*FUNCTIONS)
```

LIBCOPY((F1 F2))

Copies the file named F1 to the file named F2. Value returned is NIL.

Example:

```
LIBCOPY((NEWLIB LISPLIB))
```

Note: A minimal LISPLIB library file configuration is the 6 LISP functions denoted in the first 6 logical records, a directory which appears as

```
(LISPLIB:DIRECTORY ((PRETTYPRINT.1)))
```

and a logical record named PRETTYPRINT which contains the 7 functions included in the PRETTYPRINT system.

REFERENCES

1. Hearn, A. C. "Computation of Algebraic Properties of Elementary Particle Reactions Using a Digital Computer," Communications of the ACM, Vol. 9 (August, 1966), pp. 573-577.

2. Maling, K. "The LISP Differentiation Demonstration Program," Memo 10, Artificial Intelligence Project, MIT, Cambridge, Massachusetts, 1959, 6 pp.

3. McCarthy, et al. "LISP I Programmer's Manual," Computation Center and Research Laboratory of Elec., MIT, Cambridge, Mass., March 1960, p. 22.

4. Goldberg, S. H. "Solution of an Electrical Network Using a Digital Computer," M. S. thesis, Elec. Engr. Dept., MIT, Cambridge, Mass., 1959, 203 pp.

5. Edwards, D. "Symbolic Circuit Analysis with the 704 Electronic Computer," M. S. thesis, Elec. Engr. Dept., MIT, Cambridge, Mass., 1959, 48 pp.

6. Slagle, J. R. "A Heuristic Program That Solves Symbolic Integration Problems in Freshman Calculus," Journal of the ACM, Vol. 10 (October, 1963), pp. 507-520.

APPENDIX

GENERAL PECULIARITIES OF THE SYSTEM

There are several minor differences between 6400/6600 LISP 1.5 and MIT LISP 1.5, and this appendix will attempt to list these differences. References will be made to pages in the LISP 1.5 Programmer's Manual, so that the user can update his manual to conform with 6400/6600 LISP.

1. Page 2, last line should read:

   car [A] = A

   In 6400/6600 LISP the CAR and CDR of atomic symbols are given particular values instead of leaving them undefined, to prevent accidental reference to addresses outside a job's field length.

2. Page 3, 7th line should read

   cdr [A] = NIL

3. Page 4, 5th line from bottom should have inserted:

   "The 6400/6600 LISP system automatically recognizes multiple CAR's, CDR's, and CSR's having up to eight letters (A, D, or S) between the C and the R."

4. Page 16, 5th line should read:

   "A comma is equivalent to a string of one or more blanks, thus only one comma may be used to separate atoms in list notation."

5. Page 20, line 15 should read:

   "... the second is peculiar to the LISP Programming Systems on the IBM 7090 and the CDC 6400/6600 computer series."

6. Page 24, lines 19 and 20 should read:

   4. Absolute values must lie between 10^{308} and 10^{-308}

   5. Significance is limited to 10 decimal digits

7. Page 24, line 31 should have added:

"In 6400/6600 LISP the number 600E-1 is an acceptable floating-point number, even though it does not contain a dot."

8. Page 25, line 4 should read:

"2. Up to 20 digits (0 through 7)"

9. Page 25, lines 13 through 28 should read:

"The effect of the read program on octal numbers is as follows:

1. The number is placed in the accumulator three bits per octal digit with zeros added to the left-hand side to make twenty digits. The rightmost digit is placed in bits 2-0; the leftmost digit is placed in bits 59-57.

2. The accumulator is shifted one bit (one binary digit) times the scale factor. Thus the scale factor is an exponent to the base 2.

3. If there is a negative sign, the number is complemented after it is shifted by the Q scaling factor. The examples in a through e above will be converted to the following octal words:

   a. 0000000000000000777
   b. 0000000000000017760
   c. 7777777777777763777
   d. 7777777777777743777
   e. 0000000000000034000 "

10. Page 26, lines 19 through 21 should be deleted. EXPT is not implemented in 6400/6600 LISP.

11. Page 26, 4th line from bottom should read:

"The logical functions operate on 60-bit words."

12. Pages 27 and 28: Delete section on the Array Feature; it is not implemented in 6400/6600 LISP.

13. Page 30, line 20 should read:

    "The form GO can occur at any level of a PROG (and must return to the top level of the PROG)."

14. There is no assembler or compiler, hence all references to these routines should be deleted.

15. Disregard Appendices C, D, E, G, H, and I.

16. Comments may be inserted in LISP expressions in 6400/6600 LISP. These comments must be separated from the LISP expression by punching a "↓" on each side of the comment. The "↓" is a character found on keypunches with the 64-character set; it is formed by an 11, 6, 8 punch. The effect of the "↓" is to cause the interpreter to scan looking for another "↓" and disregard anything in between. For example,

    (CA↓THIS IS A COMMENT↓R A)

    is the same as

    (CAR A).

17. Any atomic symbol beginning with a plus or minus sign and followed immediately by a non-numeric character will not be converted into a numeric atom but rather will function as a non-numeric atom. "+" and "-" by themselves are included in this group.

Notes on the Use of LISP Version 1.5.6

6/24/68

LISP Version 1.5.6 makes the following corrections to LISP Version 1.5.5:

1. CONC was entirely rewritten. The old version stopped concatenating when it encountered a null list.

   For example,

   in the old version:

   conc [(A B); NIL; (C)]

   returned (A B)

   In the new version it returns (A B C).

2. MAPCON produced an effect very similar to that explained in (1) and was also fixed.

3. Information appearing in column 73-80 would cause the interpreter to lose information in a user's program. This has been corrected.

4. The number of garbage collections which occurred in a run is now printed at the end of the run (regardless of G parameter setting).

5. In rare cases, our failure to normalize after additions and subtractions caused incorrect arithmetic results. This has been corrected.

6. The READ function has been extended to read not only lists, but also atoms (as it should have originally been implemented).

7. The RANDOM function has been added. See page 6 of the enclosed supplement manual.

8. MKNAM spelling has been corrected from MKANM.

9. More than one doublet for evalquote may appear on a card.

Installing the LISP 1.5.6 System

The tape which is enclosed is 556 BPI density and is in UPDATE format. The tape is unlabelled and should be declared external on SCOPE 3.X systems.

The following program will produce a relocatable binary deck version of LISP 1.5.6 suitable for execution:

```
JOB card

REQUEST,OLDPL,HI,X.

UPDATE(Q)

COMPASS(I=COMPILE, L=0,B=PUNCHB)

7
 8
  9

*COMPILE, LISP

6
 7
  8
   9
```