P.O. BOX 791 / WOODLAND HILLS, CALIFORNIA 91364 / TELEPHONE (213) 345-6506

CALL-A-COMPUTER

OF CALIFORNIA, INC.

March 13, 1969

Mr. Clark Weissman
Systems Development Corp.
2500 Colorado Avenue
Santa Monica, Calif.

Dear Mr. Weissman:

I am enclosing a list of the Lisp errors which will appear on our system. You were correct as we do have a Lisp interpreter, not a compiler. I have attained a copy of your book and will attempt some programs this weekend.

I hope this preliminary information will be sufficient for further investigation on your part.

Very truly yours,

Richard Swarz
RS:rl
Encs.

TABLE OF CONTENTS

I. Differences Between LISP and Lisp 1.5

The Call-A-Computer Time-Sharing System now has a limited version of the Lisp 1.5 System developed for the IBM 7090 at M.I.T. The C-A-C version is called LISP. Because LISP was developed as an academic exercise, it is intended to be used only for educational purposes. It follows the basic outlines defined in (Lisp 1.5 Primer), by Clark Weissman, Dickenson Publishing Co. and Lisp 1.5 Programmer's Manual, by John McCarthy, et al., M.I.T. Press. However, the user should be aware of the following changes to Lisp 1.5:

1) The C-A-C Lisp Interpreter differs from the interpreter described in Appendix B of the M.I.T. Manual in that the universal function EVALQUOTE of the interpreter has been replaced by EVAL. As a result, the following restrictions apply:

   a) The user's program must consist of forms for EVAL rather doublets for EVALQUOTE. For example,

      CAR((A B C)) becomes (CAR(QUOTE(A B C))) and
      CONS(A B) becomes (CONS(QUOTE A)(QUOTE B))

   b) There is no special check made for NIL in the LISP functionals EVAL and APPLY. Therefore, expression such as (APPLY ()args a-list) is a request that APPLY evaluate an undefined function. In that case Error Message A2 is printed, describing the type of error and the function--NIL--that was undefined.

   c) The user can change the interpreter to EVALQUOTE by executing (INTERPRETQ T).

   d) The user can change the interpreter back to EVAL by executing INTERPRETQ(()).

2) LISP does not contain a compiler or the internal assembly program, LAP, described in the Manual.

3) The monitor for the Lisp 1.5 System is called OVERLORD. Because several changes have been made to the Lisp monitor, the user should disregard the description of OVERLORD in Appendix E of the Manual. Instead, the user must set up his program as a collection of forms. Each of the forms is then evaluated and the results are printed. When a program is finished, an End-of-File Error (Error R4) is printed. In addition, because the READ routine ignores carriage returns and line numbers, a form may run for several lines. A special function in LISP called LISTEN causes EVAL to interpret directly from the teletype instead of the current program.

4) List structure is stored differently in the GE 235 than in the 7090. Each word of list structure comprises a CAR half and a CDR half. The CAR half is stored in an even location word of core memory and the CDR half is stored in the next higher odd location word. All list pointers are internal machine addresses which point to the even half of the word. Therefore, the atom head (i.e., the CAR of the atom) in the 235 is an odd pointer rather than the -1 used in the 7090. Because there is no Tag portion in the 235, numbers are represented as atoms with FIXNUM or FLONUM properties rather than the PNAME property of regular atoms. The property following FIXNUM is a pointer specifying an integer and the property following FLONUM points to a floating-point number. To accomodate both PNAMES and numbers, double-word locations are reserved in full-word space.

5) The following format restrictions apply to LISP:

   a) In the Lisp 1.5 System, an atom cannot have a print name of more than 30 characters. This restriction has been removed for LISP. The user is now restricted only in the amount of storage he is willing to reserve for the print name.

   b) The symbols for true and false are T and NIL, respectively. There are no *T* or F symbols in LISP. Both T and NIL are constants that evaluate into themselves.

   c) Some changes have been made to the format of numbers in input and output operations. If, in an input operation, a string of digits preceeded by a minus sign (or an optional plus sign) contains no decimal point, it is read as an integer in the current input base. If a similar string of digits is followed immediately by only a decimal point, it is read as an integer in base ten. Floating-point numbers have the same format as described in the Lisp 1.5 Manual. If there is an E (decimal scale factor) in a floating-point number, the decimal point may be dropped. In that case, the decimal point is assumed to preceed the E. To set the input and output base of numbers, two new functions were added--SETIBASE and SETOBASE. SETIBASE affects only the input transfer of numbers that do not contain a decimal point or an E. SETOBASE does not affect the output transfer of floating-point numbers.

6) The following restrictions apply to the pseudo-functions READ, RETURN, and GO:

   a) When a programmer makes a call to READ, the function does not read the first list after STOP. Instead, it reads the next list that has not already been read by the interpreter.

   b) The restriction that GO and RETURN may appear only at the top level of a program feature (PROG) has been removed. Instead, they may be used any time after a PROG has been entered. When they are evaluated, GO has as its value the point to which the program transfers and RETURN has as its value the return value of the PROG. GO and RETURN are not executed until the interpreter returns to the top level of the current PROG. At that time PROG transfers to the point specified by GO or RETURN. GO and RETURN may only refer to the last PROG entered by the interpreter.

c) The LISP READ Routine does not recognize the $$ convention of Lisp 1.5 for quoting characters in print names. Instead, LISP uses the slash (/). The character following a slash is placed in the PNAME regardless of what it is. The characters that must be slashed are: space, comma, dot, slash, right parentheses, left parentheses, and carriage return. To prevent confusion between atoms and numbers, leading plus signs, minus signs, and digits are also slashed.

7) The only change that has been made to the special forms of Lisp 1.5 is the redefinition of COND. COND now accepts different forms of arguments from those described in the Manual. COND still accepts expressions coded in the old manner. However, COND now returns NIL instead of printing an error message if none of the propositions is true. For a more detailed description of the changes to COND, see page 17.

II. LISP Atoms

The following is an alphabetic list of the atoms that are initially defined in the C-A-C version of LISP. A dollar sign ($) indicates a change from the "Lisp 1.5 Programmer's Manual."

Atom	Revised	Property	Description
ADD1		(SUBR)	
ADVANCE	$	(SUBR)	- ADVANCE reads the next character from the disk. It ignores line numbers. ADVANCE should be used carefully because the characters it reads are not read by the READ function.
ADVANCE*	$	(SUBR)	- ADVANCE* is identical to ADVANCE except that it does not ignore line numbers.
ADVANCETTY	$	(SUBR)	- ADVANCETTY reads the next character from the user's teletype. If the last character read was a carriage return, then a call of ADVANCETTY causes a corresponding call for an input transfer. In addition, the first character typed is returned. The first character read by ADVANCETTY is a carriage return.
AND		(FSUBR)	
APPEND		(SUBR)	
APPLY	$	(SUBR)	- APPLY no longer makes a special check for NIL as a function name. Instead, the statement (APPLY NIL args a-list) will cause an error message (ERROR A2) to be printed stating that NIL is an undefined function.
ATTRIB		(SUBR)	
BELL	$	(APVAL)	- BELL is for those who like audio-output from their program. BELL is a constant that evaluates into an atom with the print name ⟨BELL⟩.
CAAAR		(SUBR)	
CAADR		(SUBR)	
CAAR		(SUBR)	
CADAR		(SUBR)	
CADDR		(SUBR)	
CADR		(SUBR)	
CAR		(SUBR)	
CDAAR		(SUBR)	
CDADR		(SUBR)	
CDAR		(SUBR)	
CDDAR		(SUBR)	
CDDDR		(SUBR)	
CDDR		(SUBR)	
CDR		(SUBR)	

Atom	Revised	Property	Description
CLEARBUFF		(SUBR)	
CLOCK	$	(SUBR)	- CLOCK is a function of no arguments. Its value is the numerical time of day in sixths of a second.
CONC		(SUBR)	
COND	$	(FSUBR)	- COND has been modified to accept an expanded form of propositions. For a more detailed explanation of the changes to COND, see page 17.
CONS		(SUBR)	
COPY		(SUBR)	
CSET		(SUBR)	
CSETQ		(FSUBR)	
DEFPROP	$	(FSUBR)	- DEFPROP is identical to PUTPROP except that DEFPROP quotes its three arguments. This allows the user to assign properties to atoms without typing "QUOTE".
DIFFERENCE		(SUBR)	
DIVIDE		(SUBR)	
EQ		(SUBR)	
EQUAL		(SUBR)	
ERROR		(SUBR)	
ERSETQ	$	(FSUBR)	- ERSETQ is a function of one argument. Its value is the evaluation of its argument CONSed with NIL (if no error occurs). If an error occurs, then ERSETQ returns NIL. ERSETQ allows the user to try one approach to a solution and, if it fails, to try another path.
EVAL		(SUBR)	
EVLIS		(SUBR)	
FIX		(SUBR)	
FIXNUM	$		- FIXNUM is the property of a fixed-point number.
FIXP		(SUBR)	
FLOATP		(SUBR)	
FLONUM	$		- FLONUM is the property of a floating-point number.
FUNCTION		(FSUBR)	
FWCONS	$	(SUBR)	- FWCONS is a function of one argument. Its argument points to a full word. FWCON's value points to a new full word which is a copy of the argument.
GENSYM		(SUBR)	
GET		(SUBR)	
GO	$	(FSUBR)	- GO may be used any time after a PROG has been entered. If it does not appear at the top level of a PROG, it is not executed until control returns to the PROG.
GO*	$	(SBUR)	- GO* is identical to GO except that it evaluates its argument. It may be used for a switch-type transfer.
GREATERP		(SUBR)	
INTERN		(SUBR)	

Atom	Revised	Property	Description
INTERPRETQ	$	(SUBR)	- INTERPRETQ is a function designed to give EVAL greater compatibility with EVALQUOTE systems. INTERPRETQ takes one argument. If this argument is NIL, the interpretive function is EVAL. However, if the argument is not NIL, the interpretive function becomes EVALQUOTE. Thus, (INTERPRETQ T) calls EVALQUOTE and INTERPRETQ(NIL) brings back EVAL. INTERPRETQ affects both the normal interpreting from the program and the interpreting done on statements typed in from a teletype by means of a LISTEN operation. The value of INTERPRETQ is its previous argument. This is initially NIL so that interpreting starts with EVAL.
LEFTSHIFT or RIGHTSHIFT	$	(SUBR)	- The second argument of LEFTSHIFT or RIGHTSHIFT (i.e., the shift count) must be a fixed-point number. These functions shift the number defined as the first argument the specified number of bit positions. The floating-point argument is handled the same way as in LOGOR. The effect on fixed-point numbers is to multiply or divide by the appropriate power of two. The forms (LEFTSHIFT A B) and (RIGHTSHIFT A(MINUS B)) are identical except when B is zero. In this case, because of the nature of the GE 235, a LEFTSHIFT 0 causes a shift of the sign bit of the lower half into the upper half while a RIGHTSHIFT 0 causes a shift of the sign bit from the upper half into the lower half. In fixed-point arithmetic, the sign of the lower half is ignored and is forced to agree with the sign of the upper half. In floating-point arithmetic, the sign of the lower half is the sign of the number, and the sign of the upper half is the sign of the exponent.
LENGTH		(SUBR)	
LESSP		(SUBR)	
LIST		(FSUBR)	
LISTEN	$	(SUBR)	- LISTEN is a function useful for time-sharing LISP. On entering LISTEN, the interpreter stops evaluating from the disk. Instead, it evaluates statements from the teletype. LISTEN is terminated by typing the atom "STOP". The user must take care to insure that STOP is followed by a space to prevent the time-sharing system from accepting the input as a command to stop LISP. When LISTEN is terminated, it returns NIL. The typical LISP program consists of a set of function definitions followed by LISTEN.

Atom	Revised	Property	Description
LOGAND or LOGOR or LOGXOR	$	(FSUBR)	- These functions operate normally except that they accept floating-point arguments without error. If given a floating-point argument, they do not FIX it but rather use its pattern of bits directly to form their result. The value of these functions is always a fixed-point number.
MAKNUM	$	(SUBR)	- MAKNUM is a function of two arguments. It creates an atom with a property list of MAKNUM's second argument followed by MAKNUM's first argument. MAKNUM is often used as an easy way to create numbers from full words.
MAP or MAPCAR or MAPCON	$	(SUBR)	- For MAP, MAPCAR, and MAPCON, the order of arguments from their original definitions has been reversed.
MAX		(SUBR)	
MEMBER		(SUBR)	
MIN		(SUBR)	
MINUSP		(SUBR)	
NCONC		(SUBR)	
NIL		(APVAL)	
NOT		(SUBR)	
NUMBERP		(SUBR)	
NUMOB		(SUBR)	
OBLIST	$	(APVAL)	- As on the IBM 7090, the OBLIST evaluates into all atomic objects that have been transferred into the system. The OBLIST on the GE 235 is similar to the 7090 except that it has only 32 buckets instead of the usual 64.
ONEP		(SUBR)	
OPEN	$	(SUBR)	- OPEN is a function of one or two arguments. Its purpose is to determine the file from which the READ, ADVANCE, and ADVANCE* operations pull their data. Initially, the file open for reading is the user's current program. However, to read elsewhere on the disk, the user specifies the user number and the file name to OPEN. The first argument of OPEN is the user number; the second is the file name. For example, to read in TRACE from the library, enter (OPEN(QUOTE LIBEVA)(QUOTE TRACE-)). It should be noted that both the user number and the file number are atoms, never numbers. In order to make a user number with all numeric characters look like an atom, the first digit must be quoted with a slash. Also any file name with less than six characters must be followed by blanks. For example, to open file "ABCD" under user number A12345, enter (OPEN(QUOTE / A12345)(QUOTE ABCD/ /)). Once a file is opened, reading begins at the start of the file by using a file-reading command

Atom	Revised	Property	Description
			(i.e., READ, ADVANCE, or ADVANCE*). The first character read by ADVANCE or ADVANCE* will be a carriage return. If, instead of giving OPEN a user number and a file name, the user gives only the single argument NIL, then reading begins at the user's current program. Thus, (OPEN()) acts as a restore operation.
OR		(FSUBR)	
PACK		(SUBR)	
PACKO	$	(SUBR)	- PACKO is a function of one argument which points to a character. It adds this character to the teletype output buffer. PACKO returns NIL.
PAIR		(SUBR)	
PLUS		(FSUBR)	
PRINT		(SUBR)	
PRIN1	$	(SUBR)	- PRIN1 may now be used to print lists as well as atoms. If given a list, PRIN1 prints it normally but does not terminate the print line.
PROG		(FSUBR)	
PROG2		(FSUBR)	
PROP		(SUBR)	
PUTPROP	$	(SUBR)	- PUTPROP represents LISP's method of defining things. It is a function of three arguments. The first of these must be the atom that is being defined. The second and third arguments are the definition and the definition type (usually EXPR or FEXPR), respectively. Any other occurence of the property type on the defined atom is removed.
QUOTE		(FSUBR)	
QUOTIENT		(SUBR)	
READ	$	(SUBR)	- READ now reads the next unread list from the disk. It is initially set to read from the user's current program area.
READTTY	$	(SUBR)	- READTTY is a function with no arguments. Its value is a list that is entered via the user's teletype. It calls for an input transfer and continues the operation until a complete list is typed. If parentheses do not balance on one line, the computer continues to request an input transfer until they are balanced. As with the READ function, READTTY ignores carriage returns. Therefore, a space must follow a single atom. On any call for an input transfer, more than one list may be specified; if so, the extra lists (or part of a list) will be used on the next operation of READTTY.
RECIP		(SUBR)	
RECLAIM		(SUBR)	
REMAINDER		(SUBR)	
REMPROP		(SUBR)	

Atom	Revised	Property	Description
RETURN	$	(SUBR)	- RETURN may now be given any time after a PROG has been entered. If it is not given on the top level of a PROG, it is not executed until control returns to the PROG.
REVERSE	$	(SUBR)	
RIGHTSHIFT	$	(SUBR)	- See the description for LEFTSHIFT.
REPLACA or REPLACD	$	(SUBR)	- A check is made to determine if the user is modifying a location that is not in free storage or full-word space. If he is, the operation is not completed and an error message (ERROR MEM) is printed. Because initial definitions are not located in these areas, the user may not modify them.
SASSOC		(SUBR)	
SET		(FSUBR)	
SETQ		(FSUBR)	
SETIBASE	$	(SUBR)	- SETIBASE is a function of one argument. It changes the base of integers on an input transfer. SETIBASE affects only those integers that do not have a decimal point as the last character. SETIBASE has no effect on floating-point numbers. The value of SETIBASE is the previous input base. This allows routines both to change the base and to restore it. The input base is initially set to ten.
SETOBASE	$	(SUBR)	- SETOBASE is a function of one argument. It changes the output base of registers. The execution of SETOBASE returns the previous output base. The output base is initially set to ten.
STOP	$	(SUBR)	- STOP as an atom is used to terminate LISTEN. STOP as a function is used to give a terminal exit for LISP.
SUBST		(SUBR)	
SUB1		(SUBR)	
T	$	(APVAL)	- T replaces *T* in the Lisp 1.5 Manual. T is a constant that evaluates into itself. Predicates in LISP return either T or NIL.

Atom	Revised	Property	Description
TERPRI		(SUBR)	
TIMES		(FSUBR)	
XEROP		(SUBR)	

III. LISP Error Messages

The following is a list of possible error messages for the C-A-C version of LISP. Some messages have been added to and some have been deleted from the list of messages in the "Lisp 1.5 Programmer's Manual."

Interpreter Errors

ERROR	DESCRIPTION
ERROR A1	- APPLIED FUNCTION CALLED ERROR. This is an error forced by the programmer by the evaluation of the function ERROR. ERROR is a function of one argument. On the line after the error type, the evaluation of ERROR's argument is printed. ERROR is particularly useful as a debugging aid.
ERROR A2	- FUNCTION OBJECT HAS NO DEFINITION - APPLY. APPLY has been asked to evaluate an undefined function. After the error type, the function that was undefined is printed.
ERROR A4	- SETQ GIVEN ON A NONEXISTENT PROGRAM VARIABLE - APPLY. The nonexistent variable is printed after the error type.
ERROR A5	- SET GIVEN ON A NONEXISTENT PROGRAM VARIABLE - APPLY. The nonexistent variable is printed after the error type.
ERROR A6	- GO (or GO*) REFERS TO AN UNLABELLED POINT. The unlabelled point is printed after the error type.
ERROR A8	- EVAL ASKED TO EVALUATE AN UNBOUND VARIABLE. The variable that is not defined (by LAMBDA, PROG, or CSET) is printed. This error often occurs when there is a parentheses miscount.
ERROR A9	- FUNCTION OBJECT HAS NO DEFINITION - EVAL. The undefined function is printed. This error often occurs when there is a parentheses miscount.
ERROR A10	- PUTPROP'S FIRST ARGUMENT IS NOT ATOMIC. This occurs with DEFPROP,CSET, and CSETQ.

Compiler Errors

ERROR	DESCRIPTION
ERROR CH1	- TOO MANY CHARACTERS - PACK. PACK has been called to pack more than 81 characters into BUFFO without using CLEARBUFF.
ERROR CH2	- FLOATING-POINT NUMBER OUT RANGE - NUMOB. It is possible to get this error also on decimal integers of more than 12 characters since, at this point, NUMOB could not have determined whether or not the number is a floating-point number.
ERROR CH4	- BAD CHARACTER - NUMOB

Miscellaneous Errors

ERROR	DESCRIPTION
ERROR DSK	- THERE HAS BEEN A DISK ERROR. If this error occurs during an OPEN operation, then READ, ADVANCE, and ADVANCE * become undefined.
ERROR F2	- PAIR'S FIRST ARGUMENT LIST IS TOO SHORT. The extra elements are printed after the error type. This most often happens when too many arguments are given to a LAMBDA - bound function.
ERROR F3	- PAIR'S SECOND ARGUMENT LIST IS TOO SHORT. Elements are printed after the error type. This most often happens when too few arguments are given to a LAMBDA-bound function.
ERROR G1	- ARITHMETIC OVERFLOW OR DIVIDE CHECK. On the GE235 both fixed - and floating - point operations can be "trapped".
ERROR G2	- OUT OF PUSH-DOWN LIST. This happens when recursion is too deep. It often indicates infinite recursion.

Garbage Collector Errors

ERROR	DESCRIPTION
ERROR GC2	- NO WORDS COLLECTED - GARBAGE COLLECTER. This means that the evaluation of present functions requires more than the machine's capacity for list structure. The GE235 has a very small capacity.

Arithmetic Errors

ERROR	DESCRIPTION
ERROR I3	- ARGUMENT OF A NUMERIC FUNCTION IS NOT A NUMBER.
ERROR I4	- ARGUMENT OF A FUNCTION EXPECTING A FIXED-POINT NUMBER IS A FLOATING-POINT NUMBER.

Memory Errors

ERROR	DESCRIPTION
ERROR MEM	- There has been an attempt to modify a cell that is not in free storage or full-word space. This may happen if the user tries to modify a predefined atom.
ERROR TRA	- This error occurs when the interpreter discovers a transfer into memory using SUBR or FSUBR flags. This prevents the user from transferring to illegal locations in memory.

Input-Output Errors

ERROR	DESCRIPTION
ERROR P2	- ATOM HAS NO PRINT NAME - PRIN1. This happens when the program attempts to print an atom without a PNAME, FIXNUM, or FLONUM on its property list. This most often occurs when the user tries to print CAR's or CDR's beyond the atomic level.
ERROR P3	- BAD BASE - SETOBASE OR SETIBASE. Only number bases between two and ten can be used for integer input-output operations.
ERROR R1	- FIRST OBJECT ON THE INPUT LIST IS ILLEGAL - READ. This most often happens when there is a misplaced ⟨dot⟩ or ⟨right parenthesis⟩ .
ERROR R2	- CONTENT ERROR WITH DOT NOTATION - READ.
ERROR R3	- ILLEGAL CHARACTER - READ. The input transfer of a "bad" end of file causes this error. This occurs when READ, ADVANCE, or ADVANCE* attempt to read beyond the end of file.

Input-Output Errors - Cont'd

ERROR	DESCRIPTION
ERROR R4	- END OF FILE. This message indicates that a program is finished.
ERROR R7	- FILE NOT FOUND - OPEN. An attempt was made to open a nonexistent file. IF OPEN is unsuccessful then READ, ADVANCE, and ADVANCE* are undefined.

IV. LISP Updates

The following is a chronological list of all LISP updates. The changes are listed in reverse order.

CURRENT VERSION: LISP 47

1) NUMOB now rounds off numbers on floating-point conversion so that (FIX 1.0) now returns 1 instead of 0.
2) On output operations using dot notation, the dot is surounded by a pair of blanks.
3) Because the original intent of OPEN has been abused, it is now undefined.
4) New functions: ATTRIB, CONC, PROP, REMPROP, MAPCON (please note that, like MAPLIST and MAPCAR, the arguments of MAPCON have been reversed.)
5) The name of the garbage collector function has been changed to RECLAIM as stated in the "Lisp 1.5 Programmers Manual."
6) The size of the output buffer has been decreased. This causes more swaps with programs that have a substantial amount of output information but gives more free storage.

LISP 46

1) A previous "bug" in NUMBERP has been fixed.
2) The form of numbers has been redefined so that numbers look like atoms whose first property is FIXNUM or FLONUM. However, instead, of having another property list after the number property, it points directly to the number. In other words, (CDDR number) specifies in full-word space a cell that contains the value of the number.
3) Storage has been reallocated to allow for more free-storage but less full-word space and less push-down list.
4) New protection features have been added so that it is more difficult (maybe even impossible) for the user to damage time-sharing.
5) The representation of constants on property lists has been changed. No longer are constants depressed one level in the list structure after the APVAL property flag. This means that CSET is defined as (PROG2(PUTPROP arg1 arg2(QUOTE APVAL))arg2) instead of (PROG2 (PUTPROP arg1(LIST arg2)(QUOTE APVAL))arg2) as previously.
6) The size of BUFFO has been changed from 81 characters to only 40 characters. The reason for this is that BUFFO can only be used for numbers and numbers more than 12 characters are illegal.

LISP 45

1) New functions: CLOCK, LOGOR, LOGAND, LOGXOR, LEFTSHIFT, RIGHTSHIFT, ADD1, SUB1, ONEP, ZEROP, MAX, MIN, RECIP, LENGTH.
2) The float phase of the arithmetic functions has been fixed so that it floats a fixed-point zero correctly.

3) New error--ERROR I4. ERROR I4 is given by an arithmetic function that detects a floating-point argument when it expected a fixed-point argument. Leftshift and rightshift operations cause this error message to be printed if a floating-point number is given as the shift count.

LISP 44

1) A bucket-sorted OBLIST has been added. The OBLIST consists of 32 sublist (not the usual 64) so that, on reading an atom, only a small fraction of the OBLIST need be scanned. The result is that READ and READTTY have been speeded up by almost a factor of five over previous versions.
2) COND has been redefined to take a greater variety of forms of arguments. See COND on page 17 for details.
3) The functions MAPLIST, MAPCAR, and MAP have been added. Their definitions have been changed slightly so that their first argument is a function and their second argument is a list to which this function is applied. In normal order, the first argument is the list and the second argument is the function. This seems awkward, however, considering the mathematical notation of F(x) for functions.
4) EVAL and APPLY have been modified so that they no longer make a special check for NIL. The result is that (APPLY NIL args alist) gives an undefined function--NIL. However, (EVAL NIL alist) still results in NIL, since NIL is a constant that evaluates into itself.
5) Other changes were made to improve the speed of LISP (like all interpretive systems, LISP is intolerabley slow). None of these changes can be noticed by the user.
6) LISP has been given a new error--ERROR TRA. This error is given if the user attempts to transfer into memory using SUBR or FSUBR flags. Only predefined functions are allowed to be SUBRS or FSUBRS.

V. Special Forms

COND is a special form which allows the user to analyze varying situations. It accepts arguments - the number of arguments is undefined - and evaluates them according to their values. COND has been modified in LISP to accept expressions in the form:

(COND(P11,P12,...,PIK)(P21,P22,...,P2L)...(PN1,PN2,...,PNM)

COND evaluates all PI1 until it finds the first that is not NIL. COND then evaluates the remainder of PIJ for all J, returning the last PIJ for its value. If all PI1 are NIL, then the value of COND is NIL. Using this definition, it is possible to have a conditional of one expression if there is only one PIJ. In this case, if PI1 is not NIL, then the value of PI1 is the value of COND. A LISP definition of COND (here called COND*) is:

```
(DEFPROP COND*
   (LAMBDA(L A)(COND
     ((NULL L)NIL)
     (T(COND1(EVAL(CAAR L)A)L A))
))       FEXPR)

(DEFPROP COND1
  (LAMBDA(E L A)(COND
    ((NULL E)(COND
       ((NULL(CDR L))NIL)
       (T(COND1(EVAL(CAADR L)A)(CDR L)A))
    ))
    (T(COND2 E(CDAR L)A))
))     EXPR)

(DEFPROP COND2
  (LAMBDA(E L A)(COND
    ((NULL L)E)
    (T(COND2(EVAL(CAR L)A)(CDR L)A))
))     EXPR)
```

VI. Examples

The LISP programs in this manual may not include all the necessary functions. Those files that contain sample definitions will not run on the LISP System. Since these functions are pre-defined, a memory error will occur if they are redefined. These definitions are intended only to give the user an idea of how the system works:

DEFINE - DEFINE is a defining function. Its argument is a list of pairs.

```
100 (DEFPROP DEFLIST
110          (LAMBDA(L A)(MAPLIST
120             (FUNCTION(LAMBDA(X)(PUTPROP(CAAR X)(CADAR X)(CADR L))))
130             (CAR L)  ))FEXPR)
140 (DEFLIST(
150   (DEFINE(LAMBDA(L A)(EVAL
160         (LIST(QUOTE DEFLIST)(CAR L)(QUOTE EXPR))
170         A  )))   )FEXPR)
180 (LISTEN)
```

BOOLE - BOOLE defines the special forms of the LISP Boolean functions AND, OR, and NOT.

```
100 (DEFPROP NOT NULL EXPR)
110 (DEFPROP AND
120          (LAMBDA(L A)(COND((NULL L)T)
130                           ((EVAL(CAR L)A)
140                            (EVAL(CONS(QUOTE AND)(CDR L))A))
150                           (T))    ))
160          FEXPR)
170 (DEFPROP OR
180          (LAMBDA(L A)(COND((NULL L)())
190                           ((EVAL(CAR L)A)T)
200                           (T(EVAL(CONS(QUOTE OR)(CDR L))A))   ))
210          FEXPR)
```

SETQ - SETQ contains sample definitions of the pseudo-functions SET and SETQ.

```
1 (LAMBDA(L A)(CDR(RPLACD(SASSOC(EVAL(CAR L)A)
120                                    A
130                                    (FUNCTION(LAMBDA()
140                                       (ERROR(QUOTE SET))    ))  )
150                                 (EVAL(CADR L)A)  )))
160          FEXPR)
170 (DEFPROP SETQ
180          (LAMBDA(L A)(CDR(RPLACD(SASSOC(CAR L)
190                                    A
200                                    (FUNCTION(LAMBDA()
210                                       (ERROR(QUOTE SETQ))    ))  )
220                                 (EVAL(CADR L)A)   )))
230          FEXPR)
```

Wang Algorithm - The following example defines the Wang Algorithm for propositional calculus as described in the Lisp 1.5 Programmer's Manual. The entry (ARROW(A1 A2 A3...)(B1 B2 B3...)) defines the elements of a list of premises (A1, A2, A3,..., An) and the elements of a list of conclusions (B1, B2, B3,...,Bn). The Wang Algorithm then states whether or not the statement is valid.

```
1TH1(LAMBDA(A1 A2 A C)(COND((NULL A)(TH2 A1 A2 NIL NIL C))
120 (T(OR(MEMBER(CAR A)C)(COND((ATOM(CAR A))(TH1(COND((MEMBER(CAR A)
130 A1)A1)(T(CONS(CAR A)A1)))A2(CDR A)C))(T(TH1 A1(COND((MEMBER(CAR A)
140 A2)A2)(T(CONS(CAR A)A2)))(CDR A)C)))))))EXPR)
150 (DEFPROP TH2(LAMBDA(A1 A2 C1 C2 C)(COND((NULL C)(TH A1 A2 C1 C2))
160 ((ATOM(CAR C))(TH2 A1 A2(COND((MEMBER(CAR C)C1)C1)(T(CONS(CAR C)
170 C1)))C2(CDR C)))(T(TH2 A1 A2 C1(COND((MEMBER(CAR C)C2)C2)(T(CONS
180 (CAR C)C2)))(CDR C)))))EXPR)
190 (DEFPROP TH(LAMBDA(A1 A2 C1 C2)(COND((NULL A2)(AND(NOT(NULL C2))
200 (THR(CAR C2)A1 A2 C1(CDR C2))))(T(THL(CAR A2)A1(CDR A2)C1 C2))))EXPR)
210 (DEFPROP THL(LAMBDA(U A1 A2 C1 C2)(COND((EQ(CAR U)(QUOTE NOT))
220 (TH1R(CADR U)A1 A2 C1 C2))((EQ(CAR U)(QUOTE AND))(TH2L(CDR U)
230 A1 A2 C1 C2))((EQ(CAR U)(QUOTE OR))(AND(TH1L(CADR U)A1 A2 C1 C2)
240 (TH1L(CADDR U)A1 A2 C1 C2)))((EQ(CAR U)(QUOTE IMPLIES))(AND(TH1L
250 (CADDR U)A1 A2 C1 C2)(TH1R(CADR U)A1 A2 C1 C2)))((EQ(CAR U)
260 (QUOTE EQUIV))(AND(TH2L(CDR U)A1 A2 C1 C2)(TH2R(CDR U)A1 A2 C1 C2)))
270 (T(ERROR(LIST(QUOTE THL)U A1 A2 C1 C2)))))EXPR)
280 (DEFPROP THR(LAMBDA(U A1 A2 C1 C2)(COND((EQ(CAR U)(QUOTE NOT))
290 (TH1L(CADR U)A1 A2 C1 C2))((EQ(CAR U)(QUOTE AND))(AND(TH1R(CADR U)
300 A1 A2 C1 C2)(TH1R(CADDR U)A1 A2 C1 C2)))((EQ(CAR U)(QUOTE OR))
310 (TH2R(CDR U)A1 A2 C1 C2))((EQ(CAR U)(QUOTE IMPLIES))(TH11(CADR U)
320 (CADDR U)A1 A2 C1 C2))((EQ(CAR U)(QUOTE EQUIV))(AND(TH11(CADR U)
330 (CADDR U)A1 A2 C1 C2)(TH11(CADDR U)(CADR U)A1 A2 C1 C2)))(T(ERROR
340 (LIST(QUOTE THR)U A1 A2 C1 C2)))))EXPR)
350 (DEFPROP TH1L(LAMBDA(V A1 A2 C1 C2)(COND((ATOM V)(OR(MEMBER V C1)
360 (TH(CONS V A1)A2 C1 C2)))(T(OR(MEMBER V C2)(TH A1(CONS V A2)C1 C2)))
370 ))EXPR)
380 (DEFPROP TH1R(LAMBDA(V A1 A2 C1 C2)(COND((ATOM V)(OR(MEMBER V A1)
390 (TH A1 A2(CONS V C1)C2)))(T(OR(MEMBER V A2)(TH A1 A2 C1(CONS V C2))))
400 ))EXPR)
410 (DEFPROP TH2L(LAMBDA(V A1 A2 C1 C2)(COND((ATOM(CAR V))(OR(MEMBER
420 (CAR V)C1)(TH1L(CADR V)(CONS(CAR V)A1)A2 C1 C2)))(T(OR(MEMBER
430 (CAR V)C2)(TH1L(CADR V)A1(CONS(CAR V)A2)C1 C2)))))EXPR)
440 (DEFPROP TH2R(LAMBDA(V A1 A2 C1 C2)(COND((ATOM(CAR V))(OR(MEMBER
450 (CAR V)A1)(TH1R(CADR V)A1 A2(CONS(CAR V)C1)C2)))(T(OR(MEMBER(CAR V)
460 A2)(TH1R(CADR V)A1 A2 C1(CONS(CAR V)C2))))))EXPR)
470 (DEFPROP TH11(LAMBDA(V1 V2 A1 A2 C1 C2)(COND((ATOM V1)(OR(MEMBER
480 V1 C1)(TH1R V2(CONS V1 A1)A2 C1 C2)))(T(OR(MEMBER V1 C2)(TH1R
490 V2 A1(CONS V1 A2)C1 C2)))))EXPR)
500 (PROG (A)A(SETQ A(ERSETQ(READTTY)))(COND((NULL A)(GO A)))
510 (SETQ A(ERSETQ(THEOREM(CAR A))))(COND((NULL A)(GO A)))
520 (COND((CAR A)(PRINT(QUOTE VALID)))(T(PRINT(QUOTE INVALID))))
530 (GO A))
1111111111111111111111111111111111
```

BREAK

```
100 (DEFPROP BREAK(LAMBDA(FN WHEN WHAT)(PROG(TYPE DEF)
110  (COND((SETQ DEF(GET FN(QUOTE EXPR)))(SETQ TYPE(QUOTE EXPR)))
120    ((SETQ DEF(GET FN(QUOTE FEXPR)))(SETQ TYPE(QUOTE FEXPR)))
130    ((PUTPROP FN(LIST(QUOTE LAMBDA)(QUOTE(L A))(LIST(QUOTE BREAK1)
140 NIL T(SETQ DEF(LIST FN(QUOTE UNDEFINED)))WHAT))(QUOTE FEXPR))
150     (RETURN DEF)))
160  (COND((EQ(CAR(CADDR DEF))(QUOTE BREAK1))(RETURN(CONS FN(QUOTE
170 (ALREADY BROKEN))))))
180  (PUTPROP FN(LIST(QUOTE LAMBDA)(CADR DEF)(LIST(QUOTE BREAK1)
190 (CADDR DEF)WHEN(LIST FN)WHAT))TYPE)
200  (RETURN FN) ))EXPR)
210 (DEFPROP BREAK1(LAMBDA(L A)(PROG(*X)
220  (COND((NULL(SETQ *X(EVAL(CADR L)A)))(RETURN(EVAL(CAR L)A)))
230   ((NULL(EQUAL *X(QUOTE(NIL))))(GO BO)))
240  (PRINT(APPEND(QUOTE(CRACK IN))(CADDR L)))
250  (COND((NULL(CAR(CDDDR L)))NIL)
260   (T(PRINT(EVAL(CAR(CDDDR L))A))))
270  (GO B3)
280  BO(PRINT(APPEND(QUOTE(BREAK IN))(CADDR L)))
290  (COND((NULL(CAR(CDDDR L)))NIL)
300  (T(PRINT(EVAL(CAR(CADDDR L))A))))
310  B1(COND((NULL(SETQ *X(ERSETQ(READTTY))))(GO BO))
320   ((EQ(SETQ *X(CAR *X))(QUOTE QUIT))(ERROR(CADDR L)))
330   ((EQ *X(QUOTE STOP))(GO B3))
340   ((EQ *X(QUOTE RETURN))(GO B2))
350   ((EQ *X(QUOTE EVAL)))
360   ((AND(SETQ *X(ERSETQ(EVAL *X A)))
370    (ERSETQ(PRINT(CAR *X))) )(GO B1))
380   (T(GO BO)) )
390  (COND((NULL(SETQ *X(ERSETQ(EVAL(CAR L)A))))(GO BO)))
400  (PRINT(CONS(CAR(CADDR L))(QUOTE(EVALUATED))))
410  (SETQ A(CONS(CONS(CAR(CADDR L))(CAR *X))A))
420  (GO B1)
430  B2(COND((OR(NULL(SETQ *X(ERSETQ(READTTY))))
440              (NULL(SETQ *X(ERSETQ(EVAL(CAR *X)A)) ))).(GO BO)))
450  (GO B4)
460  B3(COND((EQ(CAAR A)(CAR(CADDR L)))(SETQ *X(LIST(CDAR A))))
470   ((NULL(SETQ *X(ERSETQ(EVAL(CAR L)A))))(GO BO)))
480  B4(PRINT(APPEND(QUOTE(VALUE OF))(CADDR L)))
490  (COND((NULL(ERSETQ(PRINT(CAR *X))))(PRINT(QUOTE OK))))
500  (RETURN(CAR *X)) ))FEXPR)
510 (DEFPROP UNBREAK(LAMBDA(FN)(PROG(TYPE DEF)
520  (COND((SETQ DEF(GET FN(QUOTE EXPR)))(SETQ TYPE(QUOTE EXPR)))
530   ((SETQ DEF(GET FN(QUOTE FEXPR)))(SETQ TYPE(QUOTE FEXPR)))
540   (T(RETURN(CONS FN(QUOTE(NOT BROKEN))))) )
550  (COND((EQ(CAR(CADDR DEF))(QUOTE BREAK1))(RETURN
560    (PUTPROP FN(LIST(QUOTE LAMBDA)(CADR DEF)(CADR(CADDR DEF)))TYPE))))
570  (RETURN(CONS FN(QUOTE(NOT BORKEN)))) ))EXPR)
580 (DEFPROP BREAKLIST(LABMDA(L A)(MAPCAR
590  (QUOTE(LAMBDA(X)(BREAK X T NIL)))L))FEXPR)
```

BREAK - (CONTINUED)

```
600 (DEFPROP UNBREAKLIST(LAMBDA(L A)(MAPCAR(QUOTE UNBREAK)L))FEXPR)
999 (LISTEN)
```

UNION, INTERSECTION

```
100 (DEFPROP UNION
110          (LAMBDA(X Y)(COND((NULL X)Y)
120                           ((MEMBER(CAR X(Y)(UNION(CDR X(Y))
130                           (T(CONS(CAR X)(UNION(CDR X)Y)))  ))
140          EXPR)
150 (DEFPROP INTERSECTION
160          LAMBDA(X Y)(COND((NULL X)())
170                          ((MEMBER(CAR X)Y)
180                           (CONS(CAR X)(INTERSECTION(CDR X)Y)))
190                          (T(INTERSECTION(CDR X)Y))   ))
200          EXPR)
210 (LISTEN)
```

REMOB - REMOB is a pseudo-function that removes its argument from the object list.

```
100 (DEFPROP REMOB(LAMBDA(L A)(PROG(B C)A(COND((NULL L)(RETURN()))))
110 (SETQ B(CAR L))(SETQ C OBLIST)B(COND((NULL C)())((MEMBER B(CAR C))
120 (PROG()(COND((EQ(CAAR C)B)(RETURN(RPLACA C(CDAR C)))))(SETQ C(CAR C))
130 A(COND((EQ(CADR C)B)(RPLACD C(CDDR C)))((SETQ C(CDR C))(GO A)))))
140 ((SETQ C(CDR C))(GO B)))(SETQ L(CDR L))(GO A)))FEXPR)
150 (DEFPROP RENAME(LAMBDA(L A)(PROG(B)(EVAL(LIST(QUOTE REMOB)(CAR L))NIL
160 (SETQ B OBLIST) A(COND((NULL B)(RETURN()))((MEMBER(CADR L)(CAR B))
170 (GO B)))(SETQ B(CDR B))(GO A)B(SETQ B(REDUCE(CADR L)(CAR B)))
180 (RPLACA B(CAR L))(RPLACA(CDR(REDUCE(QUOTE PNAME)(CDAR L)))(GET
190 (CADR L)(QUOTE PNAME)))(RETURN(CADR L))))FEXPR)
200 (DEFPROP REDUCE(LAMBDA(A L)(PROG()B(COND((NULL)(RETURN()))
210 ((EQUAL(CAR L)A)(RETURN L)))(SETQ L(CDR L))(GO B)))EXPR)
220 (LISTEN)
```

TRACE - TRACE is a pseudo-function that accepts an unlimited number of arguments that are automatically QUOTEd. For example, to trace alpha, beta, and gamma, enter (TRACE ALPHA BETA GAMMA). TRACE will not work on pre-defined functions. TRACE contains a simulator for the TRACE feature in the IBM 7090 version of Lisp.

```
100 (DEPROP UNTRACE(LAMBDA(L A)(PROG(X Y)B(COND((NULL L)(RETURN Y)))
110 (SETQ X(CAR L))C(COND((NULL(CDR X))(GO D))((EQ(QUOTE TRACE)(CADR X))
120 (RPLACD X(CDDR(CDDDR X))))((SETQ X(CDDR X))(GO C)))(SETQ Y(CONS
130 (CAR L)Y))D(SETQ L(CDR L))(GO B)))FEXPR)
140 (DEFPROP TRACE(LAMBDA(L A)(PROG(W X Y Z)B(COND((NULL L)(RETURN Y)))
150 (SETQ X(SETQ Z(CAR L)))(SETQ L(CDR L))C(COND((NULL(CDR Z))(GO B))
160 ((EQ(CADR Z)(QUOTE TRACE))(GO B))((MEMBER(CADR Z)(QUOTE(EXPR
170 FEXPR)))(SETQ Z(CDR Z)))((SETQ Z(CDDR Z))(GO C)))(SETQ Y(NCONC Y
180 (LIST X)))(RPLACD(CDDR(SETQ W(GENSYM)))Z)(RPLACD X(CONS(QUOTE
190 TRACE)(CONS W(CONS(QUOTE FEXPR)(CONS(LIST(QUOTE LAMBDA)(QUOTE(
200 /-L /-A))(NCONC(LIST(SUBST W X(QUOTE PROG))(QUOTE(/-B /-C))(LIST
210 (SUBST W X(QUOTE PRINT))(LIST(SUBST W X(QUOTE QUOTE))(LIST(QUOTE
220 ENTERING)X))))(APPEND(SUBST W X(COND((EQ(CAR Z)(QUOTE FEXPR))
230 (QUOTE((PRINT(SETQ /-C /-L)))))(T(QUOTE((PRINT(SETQ /-B(EVAL
240 (CONS(QUOTE LIST)/-L)/-A)))/-D(COND((NULL /-B)(GO /-E)))
250 (SETQ /-C(NCONC /-C(LIST(CONS(QUOTE QUOTE)(LIST(CAR /-B))))))
260 (SETQ /-B(CDR /-B))(GO /-D))))))(LIST(QUOTE /-E)(SUBST W X(LIST
270 (QUOTE SETQ)(QUOTE /-B)(LIST(QUOTE EVAL)(LIST(QUOTE CONS)(LIST
280 (QUOTE QUOTE)W)(QUOTE /-C))(QUOTE /-A))))(LIST(SUBST W X(QUOTE
290 PRINT))(LIST(SUBST W X(QUOTE QUOTE))(LIST(QUOTE VALUE)(QUOTE OF)
300 X)))(SUBST W X(QUOTE(RETURN(PRINT /-B)))))))(CDR X))))))
310 (GO B)))FEXPR)
320 (LISTEN)
```

TRCSET - TRCSET traces SETQ in functions.

```
LXPR)))A)(T(CAR(GET(CAR L)(QUOTE APVAL)
140 )))))FEXPR)
150 (DEFPROP SWITCH(LAMBDA(L A)(PROG(B C)(SETQ B(EVAL(CADDR L)A))
160 B(COND((NULL B)(RETURN())))(SETQ C(CAR B))(RPLACD(EVAL(LIST
170 (QUOTE GETX)C)A)(SUBST(CAR L)(CADR L)(CDR(EVAL(LIST(QUOTE GETX)
180 C)A))))(SETQ B(CDR B))(GO B)))FEXPR)
190 (DEFPROP TSET(LAMBDA(/-/-L /-/-A)(PROG()(TERPRI)(PRINT(LIST(CAR
200 /-/-L)(QUOTE =)))(PRINT(SETQ /-/-A(EVAL(CADR /-/-L)/-/-A)))
210 (RETURN(SET(CAR /-/-L)/-/-A)))FEXPR)
999 (LISTEN)
```

LSPFNE

```
100(DEFPROP SQR(LAMBDA(N)(PROG(K X FLAG)(SETQ N(QUOTIENT(ADD1
110(SETQ X(PLUS N(SETQ FLAG 0.0))))2))LOOP(COND((GREATERP FLAG 12)
120(RETURN N)))(SETQ N(QUOTIENT(PLUS N(QUOTIENT X N))2))(SETQ FLAG(ADD1
130 FLAG))(GO LOOP)))EXPR)
140(LISTEN)
```