```
************************************************************************
************************************************************************

**    **    ****   ***         *     ***   ****  ****        **     ***
 *     *        * *   *        *      *   *      *   *        *    *
 *     *     ***  *   *        *      *    ***   ****         *    ****
 *     *        * *   *        *      *       *  *            *    *   *
***   ***   ****   ***         ***** ***  ****   *           ***  *  ***

*   *  ****  ***** ****   *   ****        ****  *   *  ***  ****   *****
*   * *      *     *   *  *  *           *      *   *   *   *   *  *
*   *  ***   ****  ****       ***        * ***  *   *   *   *   *  ****
*   *     *  *     *  *          *       *   *  *   *   *   *   *  *
 ***  ****   ***** *   *     ****         ***    ***   ***  ****   *****

************************************************************************
************************************************************************
```

BY GUY L. STEELE JR.

A GUIDE TO THE 1130 LISP 1.6 SYSTEM
IMPLEMENTED FOR THE IBM 1130 BY GUY L. STEELE JR.

****
***** TABLE OF CONTENTS *****
****

```
****************************************
***** CHAPTER 0 ***** INTRODUCTION *****
****************************************
```

CHAPTER 0 — INTRODUCTION

1130 LISP 1.6 IS A SMALL BUT POWERFUL IMPLEMENTATION OF THE LISP 1.6 LIST-PROCESSING LANGUAGE. THE AMOUNT OF FREE STORAGE AVAILABLE IS RATHER LIMITED, BUT THE SYSTEM IS QUITE GENERAL AND POWERFUL; MANY OF THE FEATURES OF LARGER LISP SYSTEMS ARE INCORPORATED. IN PARTICULAR, THE "SHALLOW-ACCESS" ARRANGEMENT IS USED INTERNALLY; NO A-LISTS ARE USED, AND THE SO-CALLED "FUNARG" PROBLEM IS COMPLETELY IGNORED; BUT VARIABLE EVALUATION IS EXTREMELY FAST.

IT SHOULD BE NOTED THAT 1130 LISP 1.6 IS NOT DESIGNED TO BE A PARTICULARLY FAST SYSTEM. SINCE THE IBM 1130 IS A RELATIVELY SLOW MACHINE, AND SINCE CORE MEMORY IS RATHER LIMITED, SPEED IS DIFFICULT TO OBTAIN. 1130 LISP IS RATHER DESIGNED FOR EASE OF USE AND DEBUGGING. EXTENSIVE ERROR DETECTION FACILITIES ARE AVAILABLE, AND ERROR MESSAGES, UNLIKE THOSE OF MANY OTHER LISPS, ARE CLEAR AND SELF-EXPLANATORY, AND PROVIDE EXTRA HELPFUL INFORMATION ABOUT THE ERROR. THUS 1130 LISP IS SUITABLE FOR DEVELOPMENT OF LISP FUNCTIONS WHICH AFTER DEBUGGING MAY BE TRANSFERRED TO OTHER LISP SYSTEMS.

THIS USER'S GUIDE IS INTENDED TO BE A CONCISE BUT COMPLETE DESCRIPTION OF 1130 LISP 1.6. ALL SYSTEM-SUPPLIED FUNCTIONS ARE MENTIONED AND DESCRIBED, AS WELL AS A NUMBER OF FEATURES UNIQUE TO 1130 LISP, E.G. COMPOSITE CAR/CDR FUNCTIONS OF ANY LENGTH, AND A UNIQUE INPUT/OUTPUT STRUCTURE. IT IS ASSUMED THAT THE READER IS ALREADY FAMILIAR WITH THE CONCEPTS OF LISP, AS IN THE "LISP 1.5 PROGRAMMER'S MANUAL" (JOHN MCCARTHY, ET AL.) OR THE "LISP 1.5 PRIMER" (CLARK WEISSMAN).


```
****************************
***** ACKNOWLEDGEMENTS *****
****************************
```

THE BASIC STRUCTURE OF 1130 LISP 1.6 IS A COMPOSITE OF MANY FEATURES FROM FOUR LISP SYSTEMS: 7090 LISP, AS DESCRIBED IN THE "LISP 1.5 PROGRAMMER'S MANUAL"; MACLISP, DEVELOPED AT PROJECT MAC AT THE MASSACHUSETTS INSTITUTE OF TECHNOLOGY (MIT); MULTICS LISP, DEVELOPED ON THE MULTICS SYSTEM AT MIT; AND STANFORD A.I. LISP, DEVELOPED AT THE STANFORD ARTIFICIAL INTELLIGENCE LABORATORY AT STANFORD UNIVERSITY.

THE FORMAT AND CONTENTS OF THIS USER'S GUIDE WERE BORROWED TO A GREAT EXTENT FROM THE "STANFORD A.I. LISP 1.6 MANUAL" (LYNN H. QUAM) AS DISTRIBUTED BY THE DIGITAL EQUIPMENT COMPUTER USERS SOCIETY AS DECUS NO. 6/10-38A.

```
*******************************************
***** CHAPTER 1 ***** ATOMIC QUANTITIES *****
*******************************************
```

IN 1130 LISP THERE ARE THREE KINDS OF ATOMIC QUANTITIES: IDENTIFIERS, NUMBERS, AND STRINGS.

```
<IDENTIFIER> ::= <ANY STRING OF LETTERS NOT A NUMBER OR A STRING>
<NUMBER> ::= /<HEX-DIGITS>|<FIRST-DEC><DEC-DIGITS>
<HEX-DIGITS> ::= <HEX-DIGIT>|<HEX-DIGIT><HEX-DIGITS>
<HEX-DIGIT> ::= 0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F
<FIRST-DEC> ::= +|-|<DEC-DIGIT>
<DEC-DIGITS> ::= <DEC-DIGIT>|<DEC-DIGIT><DEC-DIGITS>
<DEC-DIGIT> ::= 0|1|2|3|4|5|6|7|8|9
<STRING> ::= ,,|,<CHARACTERS>,
<LETTERS> ::= <LETTER>|<LETTER><LETTERS>
<LETTER> ::= <ANY CHARACTER EXCEPT A DELIMITER>|&<CHARACTER>
<CHARACTERS> ::= <CHARACTER>|<CHARACTER><CHARACTERS>
<CHARACTER> ::= <ANY CHARACTER ON THE IBM 029 KEYPUNCH EXCEPT 0-8-2>
<DELIMITER> ::= <SPACE>|(|)|.|,|&|'
```

IDENTIFIERS ARE STRINGS OF CHARACTERS WHICH DO NOT CONTAIN A DELIMITING CHARACTER, AND WHICH DO NOT CONSTITUTE NUMBERS OR STRINGS. IT IS POSSIBLE TO CREATE IDENTIFIERS WHICH LOOK LIKE NUMBERS OR STRINGS BY PRECEDING THE OFFENDING CHARACTER(S) WITH AN AMPERSAND (&). THUS 3645 IS A NUMBER, BUT &3465 IS A VALID IDENTIFIER, AS IS 36&45. ALSO, (BERF) IS NOT A VALID IDENTIFIER, BUT &(BERF&) IS. THE & CAUSES THE FOLLOWING CHARACTER TO BE TAKEN LITERALLY, AND IS NOT ACTUALLY PART OF THE IDENTIFIER. THUS &A&B IS THE SAME AS &AB IS THE SAME AS A&B IS THE SAME AS AB.

EXAMPLES: A   FOOBAR   QU&UX   &(&)   &&& &345   "?#@%<>   54-40

NOTE THAT SINCE 54-40 IS NOT A VALID NUMBER AND DOES NOT CONTAIN DELIMITERS, IT IS A VALID IDENTIFIER. NORMALLY ONE AVOIDS PECULIAR IDENTIFIERS LIKE THIS.

INTERNALLY AN IDENTIFIER IS REPRESENTED AS A DOTTED PAIR WHOSE CAR IS THE IDENTIFIER'S PRINT NAME AND WHOSE CDR IS ITS VALUE. (NOTE: 1130 LISP DOES NOT USE PROPERTY LISTS.) THE PRINT NAME IS A LIST OF POINTERS TO A TABLE OF CHARACTER CODES. (NORMALLY THE USER NEED NOT BE CONCERNED WITH THEIR STRUCTURE.) THIS LIST OF POINTERS POINTS TO THE CHARACTERS WHICH REPRESENT THE IDENTIFIER. THUS THE PRINT NAME OF THE IDENTIFIER &(QX&) IS A LIST POINTING TO TABLE ENTRIES FOR THE CHARACTERS (, Q, X, AND). (REMEMBER, THE &'S ARE NOT ACTUALLY PART OF THE PRINT NAME.) THE VALUE OF AN IDENTIFIER MAY BE EITHER AN S-EXPRESSION (Q.V.) ASSOCIATED WITH THE IDENTIFIER, OR A SPECIAL MARKER WHICH INDICATES THAT THE IDENTIFIER HAS NO ASSOCIATED S-EXPRESSION, I.E. ITS VALUE IS "UNDEFINED".

IN ORDER THAT OCCURRENCES OF THE SAME PRINT NAME WILL REFER TO THE SAME IDENTIFIER INTERNALLY, THERE IS A SPECIAL LIST ASSOCIATED WITH THE IDENTIFIER "OBLIST" WHICH CONTAINS ALL IDENTIFIERS SEEN BY CERTAIN FUNCTIONS SUCH AS "READ". THIS LIST IS SORTED INTO ALPHABETICAL ORDER (ACTUALLY, BY EBCDIC CODE VALUES, WHICH FOR LETTERS ARE ALPHABETICAL).

NUMBERS IN 1130 LISP ARE INTEGERS ONLY (I.E. NO "FLOATING-POINT" NUMBERS) IN THE RANGE -32768 TO 32767 (IN HEXADECIMAL, /8000 TO /7FFF, OR /0000 TO /FFFF). THEY MAY BE WRITTEN IN DECIMAL OR HEXADECIMAL (BASE SIXTEEN) NOTATION, AS FOR 1130 ASSEMBLER LANGUAGE. INTERNALLY THERE IS NO DISTINCTION BETWEEN DECIMAL AND HEXADECIMAL NUMBERS.

EXAMPLES: 0 -7 32746 800 -747 /FFFF /FACE /89 /376 /0

NOTE THAT HEXADECIMAL NUMBERS MAY NOT BE SIGNED. NOTE ALSO THAT THINGS LIKE +, -, /, 5/3, 4-, AND 5-2 ARE NOT NUMBERS, BUT IDENTIFIERS. IF ANY ONE CHARACTER IN A NUMBER IS PRECEDED BY A &, IT IS NO LONGER A NUMBER, E.G. /80&00.

STRINGS CONSIST OF A COMMA FOLLOWED BY ZERO OR MORE CHARACTERS FOLLOWED BY A COMMA. A COMMA WITHIN A STRING MUST BE WRITTEN AS TWO CONSECUTIVE COMMAS. NOTE THAT OTHER DELIMITERS WITHIN A STRING NEED NOT BE PRECEDED BY A &. INTERNALLY, STRINGS ARE LIKE IDENTIFIERS, BUT HAVE A SPECIAL MARKER AS THEIR VALUE, DENOTING THEM AS STRINGS. THE ENCLOSING COMMAS ARE NOT PART OF THE PRINT NAME OF THE STRING.

EXAMPLES: ,, ,THE GREAT QUUX, ,HI,, GUY, ,()'.&,,, , QQQ ,

****
***** CHAPTER 2 ***** S-EXPRESSIONS *****
****

S-EXPRESSIONS (SHORT FOR SYMBOLIC EXPRESSIONS) ARE THE BASIC DATA STRUCTURES OPERATED ON BY LISP.

```
<ATOM> ::= <IDENTIFIER>|<NUMBER>|<STRING>
<S-EXPRESSION> ::= <ATOM>|(<S-EXPR-LIST>)|(<S-EXPR-LIST>.<S-EXPRESSION>)
                        |()|'<S-EXPRESSION>
<S-EXPR-LIST> ::= <S-EXPRESSION>|<S-EXPRESSION><S-EXPR-LIST>
```

THE IDENTIFIER "NIL" IS SPECIALLY DEFINED TO BE EQUIVALENT TO THE NULL LIST (). AN S-EXPRESSION OF THE FORM (<S-EXPRESSION>.<S-EXPRESSION>) IS CALLED A DOTTED PAIR. S-EXPRESSIONS OF OTHER FORMS BESIDES ATOMS AND NIL CAN BE EXPRESSED IN TERMS OF DOTTED PAIRS AS FOLLOWS (LET S1, S2, S3, ETC. BE S-EXPRESSIONS, AND LET --- REPRESENT AN ELLIPSIS, I.E. WHAT ... USUALLY MEANS):

```
(S1 S2 S3 --- SN-1 SN) = (S1.(S2.(S3.( --- .(SN-1.(SN.NIL)) --- ))))
(S1 S2 S3 --- SN-2 SN-1.SN) = (S1.(S2.(S3.( --- .(SN-2.(SN-1.SN)) --- ))))
```

THE FORM '<S-EXPRESSION> IS DEFINED TO MEAN (QUOTE <S-EXPRESSION>).

EXAMPLES:

```
FOOBAR
593
,THIS IS A STRING.,
() = NIL
(THIS IS A LIST)
(DOTTED . PAIR)
(THIS IS A SO-CALLED DOTTED . LIST)
(ANOTHER DOTTED . LIST) = (ANOTHER . (DOTTED . LIST))
(THIS IS ANOTHER LIST) = (THIS.(IS.(ANOTHER.(LIST.NIL))))
'(QUUX FOOBAR) = (QUOTE (QUUX FOOBAR)) = (QUOTE.((QUUX.(FOOBAR.NIL)).NIL))
''(A B.C) = (QUOTE (QUOTE (A B . C)))
          = (QUOTE . ((QUOTE . ((A . (B . C)) . NIL)) . NIL))
```

THIS IS NOT A VALID S-EXPRESSION: (THIS IS A BAD . DOTTED LIST)

NOTE THAT SPACES ARE GENERALLY NON-SIGNIFICANT AND MAY BE USED FREELY TO IMPROVE READABILITY. EXCEPTIONS: (1) WITHIN STRINGS SPACES ARE LIKE ANY OTHER CHARACTER AND ARE PART OF THE STRING. (2) AT LEAST ONE SPACE MUST SEPARATE TWO IDENTIFIERS, A NUMBER AND AN IDENTIFIER, TWO NUMBERS, OR TWO STRINGS WHICH WOULD OTHERWISE BE ADJACENT MEMBERS OF AN <S-EXPR-LIST>. (THIS RESTRICTION IS TO PREVENT AMBIGUITIES.) IN THIS SITUATION MORE THAN ONE SPACE MAY BE USED IF DESIRED. (3) SPACES MUST NOT OCCUR WITHIN AN IDENTIFIER (SPACE IS A DELIMITER. ONE CAN INCLUDE A SPACE AS PART OF IDENTIFIER ANYWAY BY PRECEDING IT WITH A &.) (4) SPACES MAY NOT OCCUR INSIDE A NUMBER.

S-EXPRESSIONS MAY BE WRITTEN OVER SEVERAL LINES ALSO, TO IMPROVE READABILITY. THUS THE S-EXPRESSION:

```
(LAMBDA (N)(COND((MINUSP N)-1)((ZEROP N)0)(T 1)))
```

MAY INSTEAD BE WRITTEN:

```
(LAMBDA (N)
     (COND     (MINUSP N) -1)
               ('ZEROP N) 0)
```

```
                    (T 1)     ))
```

NOTE, HOWEVER, THAT SUCH FORMATTING IS NOT REQUIRED; THE USER MAY WRITE S-EXPRESSIONS IN WHATEVER FORMAT SUITS HIS PURPOSES BEST.

****
***** CHAPTER 3 ***** FUNCTIONS AND LAMBDA EXPRESSIONS *****
****

THERE ARE SEVERAL TYPES OF OBJECTS IN 1130 LISP WHICH MAY BE APPLIED AS FUNCTIONS TO S-EXPRESSIONS. THEY ARE CLASSIFIED ACCORDING TO THREE SEPARATE AND INDEPENDENT PROPERTIES:

(1) IS THE FUNCTION ITSELF AN S-EXPRESSION, OR IS IT A "COMPILED" FUNCTION, I.E. PRE-CODED INTO THE LISP SYSTEM IN ASSEMBLER LANGUAGE?

(2) HOW MANY ARGUMENTS DOES IT WANT, AND HOW DOES IT WANT THEM? IT MAY TAKE A CERTAIN FIXED NUMBER OF ARGUMENTS, OR A LIST OF ALL (ZERO OR MORE) ARGUMENTS PRESENTED TO IT. THERE ARE ACTUALLY MORE POSSIBILITIES; MORE ON THAT LATER.

(3) DOES IT WANT ITS ARGUMENTS EVALUATED OR NOT, AND SHOULD THE FUNCTION'S RESULT BE RE-EVALUATED? THIS IS A GENERALIZATION OF THE EXPR/FEXPR/MACRO PROPERTIES OF OTHER LISPS, AND IS INDEPENDENT OF PROPERTY (2) ABOVE.

****
***** INTERPRETED FUNCTIONS *****
****

FUNCTIONS WHICH ARE THEMSELVES S-EXPRESSIONS ARE CALLED "INTERPRETED" FUNCTIONS (AS OPPOSED TO "COMPILED").

```
<INTERPRETED-FUNCTION> ::= <LAMBDA-EXPR>|<LABEL-EXPR>
<LAMBDA-EXPR> ::= (<LAMBDA><PARAMETER-LIST><S-EXPR-LIST>)
<LAMBDA> ::= LAMBDA|NLAMBDA|MLAMBDA
<PARAMETER-LIST> ::= ()|(<VARIABLES>)|<VARIABLE>
                       |(<VARIABLES>.<VARIABLE>)
<VARIABLES> ::= <VARIABLE>|<VARIABLE><VARIABLES>
<VARIABLE> ::= <ANY IDENTIFIER EXCEPT NIL>
<LABEL-EXPR> ::= (LABEL <VARIABLE><INTERPRETED-FUNCTION>)
```

A LAMBDA EXPRESSION CONSISTS OF A LIST OF THE IDENTIFIER "LAMBDA" OR "NLAMBDA" OR "MLAMBDA" FOLLOWED BY A LIST OF FORMAL PARAMETERS (VARIABLES) FOLLOWED BY ONE OR MORE S-EXPRESSIONS (ACTUALLY THERE MAY BE ZERO OF THEM); THE VALUE OF THE LAST S-EXPRESSION (OR NIL IF THERE ARE NONE) IS THE VALUE OF THE FUNCTION WHEN APPLIED TO ITS ARGUMENTS.

THE FORMAL PARAMETER LIST MAY BE () (I.E. NIL), IN WHICH CASE THE FUNCTION EXPECTS NO ARGUMENTS; OR A VARIABLE, IN WHICH CASE ITS OLD VALUE IS SAVED AND A LIST OF THE FUNCTION'S ARGUMENTS BECOMES ITS NEW VALUE; OR A LIST OF VARIABLES, EACH OF WHICH IS GIVEN AS ITS NEW VALUE ("BOUND TO") ONE OF THE FUNCTION'S ARGUMENTS; OR A DOTTED LIST OF VARIABLES, EACH OF WHICH BUT THE LAST IS BOUND TO ONE ARGUMENT, AND THE LAST IS BOUND TO A LIST OF ALL THE REST OF THE ARGUMENTS. A RECURSIVE FUNCTION WHICH MAY BE USED TO REPRESENT THE BINDING OF PARAMETERS TO ARGUMENTS IS AS FOLLOWS:

```
(SETQQ BIND (LAMBDA (PARAM-LIST ARG-LIST) (COND
      ((AND (NULL PARAM-LIST) (NULL ARG-LIST)) NIL)
      ((NULL PARAM-LIST) (ERROR TOO MANY ARGUMENTS))
      ((ATOM PARAM-LIST) (PARAM-SET PARAM-LIST ARG-LIST))
      ((NULL ARG-LIST) (ERROR TOO FEW ARGUMENTS))
      (T (PARAM-SET (CAR PARAM-LIST) (CAR ARG-LIST))
           (BIND (CDR PARAM-LIST) (CDR ARG-LIST))) )))
```

WHERE "PARAM-SET" HAS THE EFFECT OF SAVING THE OLD VALUE OF A VARIABLE AND

GIVING IT A NEW VALUE. THUS THE FOLLOWING ARE VALID PARAMETER LISTS:

```
()    X    (X)    (A B C D E F)    (A B C D . E)    (A.B)    (M N P)    QUUX    NIL
```

THESE LISTS, RESPECTIVELY, DENOTE FUNCTIONS WHICH TAKE THE FOLLOWING NUMBERS OF ARGUMENTS: ZERO, ANY NUMBER, ONE, SIX, FOUR OR MORE, ONE OR MORE, THREE, ANY NUMBER, AND ZERO (NOTE THAT NIL=()).

IF THE FOLLOWING ARGUMENTS WERE GIVEN TO A FUNCTION: A B C D E THE FOLLOWING INDICATES HOW THE PARAMETERS WOULD BE BOUND FOR VARIOUS PARAMETER LISTS:

```
(V W X Y Z)              V=A  W=B  X=C  Y=D  Z=E
(X Y Z)                  ERROR: TOO MANY ARGUMENTS
(I J K L M N)            ERROR: TOO FEW ARGUMENTS
K                        K=(A B C D E)
(X Y . Z)                X=A  Y=B  Z=(C D E)
(U V W X Y . Z)          U=A  V=B  W=C  X=D  Y=E  Z=()=NIL
(T U V W X Y . Z)        ERROR: TOO FEW ARGUMENTS
```

NLAMBDA FUNCTIONS ARE EXACTLY LIKE LAMBDA FUNCTIONS, EXCEPT THAT NLAMBDA FUNCTIONS DO NOT HAVE THEIR ARGUMENTS EVALUATED, WHILE LAMBDA FUNCTIONS DO. MLAMBDA FUNCTIONS ARE LIKE NLAMBDA FUNCTIONS, EXCEPT THAT MLAMBDA FUNCTIONS HAVE THEIR FINAL RESULT RE-EVALUATED BY EVAL; THIS PROVIDES SOMETHING SIMILAR TO THE MACRO CAPABILITY OF OTHER LISPS (SEE BELOW).

EXAMPLES: THE FOLLOWING CONSISTS OF DEFINITIONS OF VARIOUS STANDARD FUNCTIONS IN TERMS OF THE ABOVE FORMALISMS. (NOTE THAT FUNCTIONS ARE DEFINED BY "SETQQ".)

```
(SETQQ LIST (LAMBDA LIST-OF-ARGS LIST-OF-ARGS))

(SETQQ QUOTE (NLAMBDA (X) X))

(SETQQ PROG2 (LAMBDA (ARG1 ARG2 . REST-OF-ARGS) ARG2))
```

ONE MIGHT DEFINE A FUNCTION CALLED "DEFINE", WHICH TAKES ANY NUMBER OF ARGUMENTS, EACH OF WHICH IS A LIST OF TWO ITEMS, A VARIABLE AND AN INTERPRETED FUNCTION. THE ARGUMENTS ARE NOT EVALUATED. "DEFINE" USES THE FUNCTION "SET" TO BIND EACH VARIABLE TO THE CORRESPONDING FUNCTION.

```
(SETQQ DEFINE (NLAMBDA DEFNS (COND
      ((NULL DEFNS) NIL)
      (T (SET (CAAR DEFNS) (CADAR DEFNS))
           (CONS (CAAR DEFNS) (APPLY 'DEFINE (CDR DEFNS)))) )))
```

THUS THE FOLLOWING FUNCTION INVOCATION WOULD DEFINE "LIST" AND "QUOTE" AS ABOVE, AND RETURN (LIST QUOTE) AS ITS VALUE:

```
(DEFINE (LIST(LAMBDA(LIST-OF-ARGS LIST-OF-ARGS))(QUOTE(NLAMBDA(X)X)) )
```

NOTE THE USE OF THE FUNCTION "APPLY" IN THE DEFINITION OF "DEFINE"; THIS IS DONE SO THAT THE REST OF THE DEFINITIONS (CDR DEFNS) WILL BE PASSED TO "DEFINE" IN THE PROPER MANNER; (DEFINE (CDR DEFNS)) WOULD NOT WORK, SINCE "DEFINE" DOES NOT EVALUATE ITS ARGUMENTS.

```
*******************************
***** COMPILED FUNCTIONS *****
*******************************
```

COMPILED FUNCTIONS ARE COLLECTIVELY REFERRED TO AS "SUBRS" (AS OPPOSED TO INTERPRETED FUNCTIONS, SOMETIMES CALLED "EXPRS"). INTERNALLY A COMPILED FUNCTIONAL OBJECT IS A DOTTED PAIR OF THE IDENTIFIER "SUBR" AND THE ADDRESS OF THE WORD PRECEDING THE MACHINE LANGUAGE CODING OF THE FUNCTION. IF THE SUBR OBJECT IS PRINTED, THE HEADER WORD PRINTS AS A NUMBER, WHICH IS THE SUM OF THREE ITEMS:

(1) FOR LAMBDA, ADD /0000; FOR NLAMBDA, ADD /4000; FOR MLAMBDA, ADD /8000.
(2) ADD IN THE NUMBER OF REQUIRED ARGUMENTS.
(3) IF IN ADDITION TO THE REQUIRED ARGUMENTS A LIST OF ALL ADDITIONAL ARGUMENTS IS DESIRED, ADD /2000.

THUS THE HEADER WORDS FOR THE FUNCTIONS "LIST", "QUOTE", "PROG2", AND "DEFINE" ABOVE, IF THEY WERE SUBRS AND NOT EXPRS, WOULD BE RESPECTIVELY /2000, /4001, /2002, AND /6000. IN THIS WAY THE HEADER WORD PROVIDES THE NEEDED INFORMATION ABOUT PARAMETER BUNDING AND ARGUMENT EVALUATION.

****
***** MACROS AND MLAMBDA EXPRESSIONS *****
****

MLAMBDA EXPRS AND SUBRS, AS MENTIONED ABOVE, CAN BE USED IN A WAY SIMILAR TO THE MACRO FEATURE OF OTHER LISPS. NO MLAMBDA SUBRS ARE PROVIDED WITH 1130 LISP; BUT AN EXAMPLE OF AN MLAMBDA EXPR IS GIVEN BELOW.

A FUNCTION TO CONS TOGETHER ANY NUMBER OF ITEMS COULD BE DEFINED IN THE FOLLOWING WAY:

```
(SETQQ CONSCONS (MLAMBDA X (COND
     ((NULL X) NIL)
     ((NULL (CDR X)) (CAR X))
     (T (LIST 'CONS (CAR X) (CONS 'CONSCONS (CDR X)))) )))
```

(CONSCONS A B C) WOULD EVALUATE TO (CONS A (CONSCONS B C)). EVALUATING (CONSCONS B C) WOULD YIELD (CONS B (CONSCONS C)). EVALUATING (CONSCONS C) WOULD GIVE C. THEN THE RESULT OF EACH CONSCONS RESULT WOULD AGAIN BE EVALUATED; THUS (CONSCONS A B C) WOULD GIVE THE SAME RESULT AS (CONS A (CONS B C)).

IN GENERAL, MACROS ARE OF PARTICULAR USE ONLY IN CONJUNCTION WITH "LISP COMPILERS".

****
***** LABEL EXPRESSIONS *****
****

A LABEL EXPRESSION, IN EFFECT, CREATES A TEMPORARY NAME FOR A FUNCTION TO PERMIT RECURSIVE CALLING. WHEN A LABEL EXPRESSION IS APPLIED AS A FUNCTION TO SOME ARGUMENTS, THE VARIABLE IN THE EXPRESSION IS BOUND TO THE FUNCTION AND THE FUNCTION IS THEN APPLIED TO THE ARGUMENTS. AFTER THE FUNCTION RETURNS ITS VALUE, THE VARIABLE HAS ITS OLD VALUE RESTORED TO IT, AND THE VALUE RETURNED BY THE FUNCTION IS THE VALUE OF THE LABEL FUNCTIONAL EXPRESSION.

EXAMPLE: THE FUNCTION "REVERSE", WHICH REVERSES A LIST.

```
(SETQQ REVERSE (LAMBDA (X)
     ((LABEL REVERSE1 (LAMBDA (M N) (COND
          ((NULL M) N)
          (T (REVERSE1 (CDR M) (CONS (CAR M) N))) ))) X NIL)))
```

ORDINARILY THE FUNCTION "REVERSE1" DOES NOT EXIST; BUT WHEN "REVERSE" IS USED, THE LABEL EXPRESSION GIVES THE CONTAINED LAMBDA EXPRESSION AS A VALUE TO THE VARIABLE "REVERSE1", SO THAT THE FUNCTION MAY BE REFERRED TO BY THE INNER LAMBDA EXPRESSION. AFTER "REVERSE" RETURNS ITS VALUE, THE FUNCTION "REVERSE1" AGAIN DOES NOT EXIST; THE VARIABLE "REVERSE1" NOW HAS WHATEVER VALUE IT STARTED WITH, IF ANY.

```
*****************************************************
***** CHAPTER 4 ***** EVALUATION OF S-EXPRESSIONS *****
*****************************************************
```

THIS CHAPTER DESCRIBES THE MAIN FUNCTIONS OF THE LISP SYSTEM. (NOTE THAT IN THIS AND SUCCEEDING CHAPTERS, FUNCTIONS MAY BE PARTIALLY OR WHOLLY DEFINED IN TERMS OF LAMBDA EXPRESSIONS. IF NOTHING ELSE, THE FIRST PART OF A LAMBDA EXPRESSION WILL BE GIVEN FOR EACH FUNCTION, TO INDICATE THE NUMBER OF ARGUMENTS AND WHETHER THEY ARE EVALUATED.)

```
**********************************
***** S-EXPRESSION EVALUATION *****
**********************************
```

EVAL   (LAMBDA (X) ---

(EVAL E) RETURNS THE VALUE OF THE S-EXPRESSION E. (ACTUALLY THE ARGUMENT IS EVALUATED TWICE: ONCE BEFORE EVAL SEES IT, AND ONCE AFTER. THUS IF THE IDENTIFIER "A" HAS THE VALUE (B.C) AND "B" HAS THE VALUE (Q R S), THEN THE RESULT OF (EVAL (CAR A)) IS (Q R S).)
IN GENERAL, AN S-EXPRESSION IS EVALUATED AS FOLLOWS:

(1) THE VALUE OF AN IDENTIFIER IS ITS VALUE (CDR).
(2) THE VALUE OF A NUMBER OR STRING IS THE NUMBER OR STRING ITSELF.
(3) THE VALUE OF A LIST IS OBTAINED BY APPLYING THE FIRST ITEM OF THE LIST, AS A FUNCTION, TO THE REST OF THE ITEMS, AS ARGUMENTS.

```
(SETQQ EVAL (LAMBDA (X) (COND
     ((NULL X) NIL)
     ((ATOM X) (COND
          ((NUMBERP X) X)
          ((STRINGP X) X)
          ((DEFINEDP X) (CDR X))
          (T (ERROR 23 UNBOUND VARIABLE))))
     ((NULL (CAR X)) NIL)
     ((ATOM (CAR X)) (COND
          ((OR (NUMBERP (CAR X)) (STRINGP (CAR X))) (ERROR 24 BAD
               (ERROR 24 INVALID FUNCTION))
          ((DEFINEDP (CAR X)) (EVAL (CONS (CDAR X) (CDR X))))
          (T (ERROR 25 UNDEFINED FUNCTION))))
     ((OR (EQ (CAAR X) 'LAMBDA)
          (EQ (CAAR X) 'C-R)
          (AND (EQ (CAAR X) 'SUBR) (ZEROP (LSH (CDAR X) -14))))
      (APPLY (CAR X) (MAPCAR 'EVAL (CDR X))))
     ((OR (EQ (CAAR X) 'NLAMBDA)
          (AND (EQ (CAAR X) 'SUBR) (EQUAL 1 (LSH (CDAR X) -14))))
      (APPLY (CAR X) (CDR X)))
     ((OR (EQ (CAAR X) 'MLAMBDA)
          (AND (EQ (CAAR X) 'SUBR) (EQUAL 2 (LSH (CDAR X) -14))))
      (EVAL (APPLY (CAR X) (CDR X))))
     ((EQ (CAAR X) 'LABEL (PROG (Q)
          (COND ((OR (NULL (CDAR X)) (NULL (CDDAR X) (NOT (NULL
                    (CDDDAR X))))) (ERROR 27 BAD LABEL EXPRESSION))
               ((NOT (ATOMP (CADAR X))) (ERROR 26 BAD 1ST ARG FOR LABEL)))
          (BIND (LIST (CADAR X)) (LIST (CADDAR X)))
          (SETQ Q (EVAL (CONS (CADDAR X) (CDR X))))
          (UNBIND 1)
```

```
        (RETURN Q)))
    (T (EVAL (CONS (EVAL (CAR X)) (CDR X)))) )))
```

WHERE "BIND" IS A FUNCTION SIMILAR TO THE ONE USED IN CHAPTER 3 AND "UNBIND" RESTORES THE VALUES OF AS MANY VARIABLES AS SPECIFIED (I.E. BIND SAVES THE OLD VALUES OF VARIABLES ON A PUSH-DOWN LIST BEFORE GIVING THEM NEW VALUES; UNBIND IS USED TO RESTORE THESE OLD VALUES.) THIS DEFINITION SHOULD NOT BE TAKEN TOO LITERALLY; THERE IS ACTUALLY SOME EXTRA PROCESSING AND ERROR-DETECTION INVOLVED. THE ABOVE, HOWEVER, IS A REASONABLY ACCURATE DESCRIPTION.

****
***** FUNCTION APPLICATION *****
****

APPLY (LAMBDA (FN ARGS) ---

APPLY APPLIES A FUNCTION TO A SET OF ARGUMENTS. IT BINDS EACH S-EXPRESSION IN "ARGS" TO THE PROPER PARAMETER OF THE FUNCTION "FN", THEN EVALUATES THE FUNCTION AND RETURNS ITS VALUE. NOTE THAT APPLY NEVER EVALUATES ANY ARGUMENTS; IT ASSUMES THAT THEY ARE PROPERLY EVALUATED ALREADY FOR THE GIVEN FUNCTION. (THIS PROCESS IS NORMALLY DONE THROUGH EVAL (Q.V.))

```
(SETQQ APPLY (LAMBDA (FN ARGS) (COND
     ((NULL FN) NIL)
     ((ATOM FN) (APPLY (EVAL FN) ARGS))
     ((MEMBER (CAR FN) '(LAMBDA NLAMBDA MLAMBDA))
      (PROG (Q)
          (BIND (CDAR FN) ARGS)
          (SETQ Q (LAST (MAPCAR 'EVAL (CDDR FN))))
          (UNBIND ((LABEL LENGTH (LAMBDA (X) (COND
                    ((ATOM X) 0)
                    (T (ADD1 (LENGTH (CDR X)))) ))) (CDAR FN)))
          (RETURN Q)))
     ((EQ (CAR FN) 'SUBR)
          (SPREAD (BOOLE 1 /3FFF (CDR FN)) ARGS)
          (PUSHJ (ADD1 FN)))
     ((EQ (CAR FN) 'C-R) (COND
          ((OR (NULL ARGS) (NOT (NULL (CDR ARGS))))
               (ERROR 35 WRONG NUMBER OF ARGS FOR C-R FUNCTION))
          (T (C-R-APPLY (CADR FN) (CAR ARGS)))))
     (T (APPLY (EVAL FN) ARGS)) )))
```

WHERE "SPREAD" PERFORMS FOR SUBRS WHAT "BIND" DOES FOR EXPRS; "PUSHJ" HAS THE EFFECT OF CALLING A COMPILED FUNCTION; AND "C-R-APPLY" HANDLES APPLYING A COMPOSITE CAR/CDR FUNCTION TO AN ARGUMENT. THE FUNCTION "LENGTH" WAS DEFINED WITH A LABEL EXPRESSION BECAUSE THE DEFINITION USED HERE IS NON-STANDARD. LIKE THE DESCRIPTION OF "EVAL", ABOVE, THIS DESCRIPTION SHOULD NOT BE TAKEN TOO LITERALLY, SINCE SOME PROCESSING IS NOT SHOWN BY THE ABOVE DEFINITION.

****
***** ARGUMENT QUOTING *****
****

QUOTE (NLAMBDA (X) X)

QUOTE TAKES A SINGLE ARGUMENT AND RETURNS IT UNEVALUATED. THIS IS THE

USUAL WAY TO PASS AN ARGUMENT UNEVALUATED TO A FUNCTION WHICH NORMALLY EVALUATES ITS ARGUMENTS; THUS (CAR (QUOTE (A.B)))=A AND NOT THE VALUE OF A. NOTE THAT THE EXPRESSION 'X IS THE SAME AS (QUOTE X) FOR ANY S-EXPRESSION X; ONE COULD WRITE (CAR '(A.B)) FOR THE ABOVE. NOTE THAT THERE IS NO FUNCTION NAMED "FUNCTION" IN 1130 LISP; "QUOTE" IS USED FOR ITS PURPOSE.

```
*****************************************************
***** CHAPTER 5 ***** CONDITIONAL EXPRESSIONS *****
*****************************************************
```

COND (NLAMBDA X ---

A CONDITIONAL EXPRESSION CONSISTS OF AN INVOCATION OF THE FUNCTION "COND" IN THE FOLLOWING MANNER:

```
(COND     (E1-1 E1-2 E1-3  ---  E1-N1)
          (E2-1 E2-2 E2-3  ---  E2-N2)
          (E3-1 E3-2 E3-3  ---  E3-N3)
                    ---
          (EM-1 EM-2 EM-3  ---  EM-NM))
```

WHERE THE EI-J'S ARE ANY S-EXPRESSIONS; EACH NI MAY BE ANY POSITIVE NUMBER.
THE EI-J'S ARE CONSIDERED TO BE PREDICATES, I.E. TO EVALUATE TO A TRUTH VALUE. THE EI-1'S ARE EVALUATED IN ORDER: E1-1, E2-1, E3-1, ETC., UNTIL THE FIRST EK-1 IS FOUND WHOSE VALUE IS NOT "NIL". THEN THE CORRESPONDING EK-2, EK-3, EK-4, --- EK-NK ARE EVALUATED RESPECTIVELY AND THE VALUE OF EK-NK IS THE VALUE OF THE COND EXPRESSION. IT IS POSSIBLE FOR NK=1, IN WHICH CASE THE VALUE OF EK-1 IS THE VALUE OF THE COND EXPRESSION. IF ALL EI-1'S EVALUATE TO "NIL", THEN "NIL" IS THE VALUE OF THE COND EXPRESSION.

EXAMPLES:

```
(SETQQ NOT (LAMBDA (X) (COND (X NIL) (T))))

(SETQQ AND (LAMBDA (X Y) (COND (X (COND (Y T))))))

(SETQQ OR (LAMBDA (X Y) (COND (X T) (Y T))))

(SETQQ IMPLIES (LAMBDA (X Y) (COND (X (COND (Y T))) (T))))

(SETQQ EXCLUSIVE-OR (LAMBDA (COND (X (NOT Y)) (Y T))))
```

```
****************************************
***** CHAPTER 6 ***** PREDICATES *****
****************************************
```

A PREDICATE IS A FUNCTION WHICH RETURNS A TRUTH VALUE. UNLESS OTHERWISE NOTED, ALL PREDICATES RETURN T TO REPRESENT TRUE. ALL PREDICATES RETURN NIL TO REPRESENT FALSE. SOME PREDICATES CAN CAUSE ERRORS OR UNPREDICTABLE RESULTS IF APPLIED TO ARGUMENTS OF THE WRONG TYPE.

```
****************************
***** BASIC PREDICATES *****
****************************
```

ATOM (LAMBDA (X) ---

THE VALUE OF ATOM IS T IF X IS AN IDENTIFIER, A NUMBER, OR A STRING.

ATOMP (LAMBDA (X) ---

THE VALUE OF ATOMP IS T IF X IS A NON-NIL IDENTIFIER. (THIS FUNCTION IS NON-STANDARD AND WILL NOT BE PRESENT IN OTHER LISPS.)

DEFINEDP (LAMBDA (X) ---

THE VALUE OF DEFINEDP IS T IF X IS NOT AN IDENTIFIER OR IF IT IS AN IDENTIFIER WITH A DEFINED VALUE. DEFINEDP RETURNS NIL IF X IS AN IDENTIFIER WITH NO DEFINED VALUE.

EQ (LAMBDA (X Y) ---

THE VALUE OF EQ IS T IF X AND Y ARE THE SAME THING INTERNALLY, I.E. HAVE THE SAME INTERNAL ADDRESS. IDENTIFIERS ON THE OBLIST HAVE UNIQUE ADDRESSES AND THUS EQ WILL RETURN T IF X AND Y ARE THE SAME IDENTIFIER. IN GENERAL, HOWEVER, EQ WILL NOT COMPARE NUMBERS OR STRINGS FOR EQUALITY.

EQUAL (LAMBDA (X Y) ---

THE VALUE OF EQUAL IS T IF X AND Y ARE EQUIVALENT S-EXPRESSIONS. EQUAL WILL COMPARE NUMBERS AND STRINGS FOR EQUALITY. EQUAL STRINGS MUST HAVE THE SAME CHARACTERS IN EACH EXACTLY; ,ABC , IS NOT EQUAL TO ,ABC,.

```
(SETQQ EQUAL (LAMBDA (X Y) (COND
      ((EQ X Y) T)
      ((ATOM X) (COND
           ((NOT (ATOM Y)) NIL)
           ((AND (NUMBERP X) (NUMBERP Y)) (ZEROP (ZIFF X Y)))
           ((AND (STRINGP X) (STRINGP Y))
                ((LABEL EQSTR (LAMBDA (M N) (COND
                     ((NULL M) (NULL N))
                     ((NULL N) NIL)
                     ((EQ (CAR M) (CAR N)) (EQSTR (CDR M) (CDR N))))))
                 (CAR X) (CAR Y))) ))
```

```
        ((ATOM Y) NIL)
        ((EQUAL (CAR X) (CAR Y)) (EQUAL (CDR X) (CDR Y))) )))
```

NULL (LAMBDA (X) (COND (X NIL) (T)))

THE VALUE OF NULL IS T IF X IS NIL; OTHERWISE ITS VALUE IS NIL.

MEMBER (LAMBDA (X Y) ---

IF X IS NOT EQUAL TO ANY TOP LEVEL ELEMENT OF Y, MEMBER RETURNS NIL. OTHERWISE IT RETURNS THE PART OF THE LIST BEGINNING WITH THE EQUAL ITEM.

```
(SETQQ MEMBER (LAMBDA (X Y) (COND
      ((NULL Y) NIL)
      ((EQUAL X (CAR Y)) Y)
      (T (MEMBER X (CDR Y))) )))
```

****
***** PREDICATES ON NUMBERS *****
****

NUMBERP (LAMBDA (N) ---

THE VALUE OF NUMBERP IS T IF N IS A NUMBER; OTHERWISE NIL.

ZEROP (LAMBDA (N) ---

THE VALUE OF ZEROP IS T IF N=0; ELSE NIL. ERROR IF N IS NOT A NUMBER.

MINUSP (LAMBDA (N) ---

THE VALUE OF MINUSP IS T IF N<0; ELSE NIL. ERROR IF N IS NOT A NUMBER.

LESSP (LAMBDA (M.N) ---

```
(LESSP X) = T
(LESSP X Y) = T IF X<Y; ELSE NIL.
(LESSP X1 X2 X3 X4  ---  XN-1 XN) = T IF (LESSP X1 X2)
                                     AND (LESSP X2 X3)
                                     AND (LESSP X3 X4)
                                              ---
                                     AND (LESSP XN-1 XN); ELSE NIL.
```

ERROR IF ANY ARGUMENT LOOKED AT IS NOT A NUMBER; NOTE THAT ONCE ANY CONDITION IS UNSATISFIED AND NIL IS RETURNED NO MORE ARGUMENTS ARE EXAMINED. THUS (LESSP 5 3 ,Q,) IS NOT AN ERROR (IT RETURNS NIL) BECAUSE LESSP DOES NOT HAVE TO LOOK AT THE ,Q, TO KNOW THAT NIL SHOULD BE RETURNED. HOWEVER, (LESSP 3 5 ,Q,) WILL CAUSE AN ERROR.

****
***** BOOLEAN PREDICATES *****
****

```
NOT        (LAMBDA (X) (COND (X NIL) (T)))
```

NOTE THAT "NOT" IS EQUIVALENT AS A FUNCTION TO "NULL".

```
AND        (LAMBDA X ---
```

THE VALUE OF AND IS T IF ALL ARGUMENTS EVALUATE TO NON-NIL VALUES.

```
(SETQQ AND (NLAMBDA X (COND
      ((NULL X) T)
      ((NULL (EVAL (CAR X))) NIL)
      (T (APPLY 'AND (CDR X))) )))
```

NOTE THAT IF ANY ARGUMENT IS NIL THE SUCCEEDING ARGUMENTS ARE NOT EVALUATED. THUS (AND NIL (MINUSP NIL)) WOULD NOT CAUSE AN ERROR FROM MINUSP. NOTE ALSO THAT (AND)=T.

```
OR         (LAMBDA X ---
```

THE VALUE OF OR IS T IF ANY ARGUMENT EVALUATES TO A NON-NIL VALUE.

```
(SETQQ OR (NLAMBDA X (COND
      ((NULL X) NIL)
      ((EVAL (CAR X)) T)
      (T (APPLY 'OR (CDR X))) )))
```

NOTE THAT IF ANY ARGUMENT IS NON-NIL THE SUCCEEDING ARGUMENTS ARE NOT EVALUATED. THUS (OR T (MINUSP NIL)) WOULD NOT CAUSE AN ERROR FROM MINUSP. NOTE ALSO THAT (OR)=NIL.

****
***** PREDICATES ON STRINGS *****
****

```
STRINGP    (LAMBDA (X) ---
```

THE VALUE OF STRINGP IS T IF X IS A STRING; ELSE NIL.

****
***** INPUT/OUTPUT PREDICATES *****
****

```
INDEVP     (LAMBDA (N) ---
```

THE VALUE OF INDEVP IS T IF AN INPUT DEVICE IS AVAILABLE WHOSE NUMBER IS N. AVAILABILITY IMPLIES ONLY THAT A DEVICE HANDLER FOR THAT DEVICE IS A PART OF THE LISP SYSTEM; THE DEVICE MAY ACTUALLY NOT BE PHYSICALLY PRESENT. ERROR IF N IS NOT A NUMBER.

DEVICE NUMBERS ARE EQUIVALENT TO 1130 FORTRAN DEVICE NUMBERS, AS FOLLOWS:

```
     1    UNASSIGNED
     2    1442 CARD READ/PUNCH
     3    UNASSIGNED
     4    1134 PAPER TAPE READER
```

```
     5     UNASSIGNED
     6     KEYBOARD
     7     UNASSIGNED
     8     2501 CARD READER
```

OUTDEVP   (LAMBDA (N) ---

THE VALUE OF OUTDEVP IS T IF AN OUTPUT DEVICE IS AVAILABLE WHOSE NUMBER IS N. AVAILABILITY IMPLIES ONLY THAT A DEVICE HANDLER FOR THAT DEVICE IS A PART OF THE LISP SYSTEM; THE DEVICE MAY ACTUALLY NOT BE PHYSICALLY PRESENT. ERROR IF N IS NOT A NUMBER.

DEVICE NUMBERS ARE EQUIVALENT TO 1130 FORTRAN DEVICE NUMBERS, AS FOLLOWS:

```
     1     TYPEWRITER (CONSOLE PRINTER)
     2     1442 CARD READ-PUNCH OR 1442 CARD PUNCH
     3     1132 PRINTER
     4     1055 PAPER TAPE PUNCH
     5     1403 PRINTER
     6     UNASSIGNED
     7     1627 PLOTTER
```

NOTE THAT NO OUTPUT DEVICE NUMBER 9 IS ASSIGNED; USE DEVICE NUMBER 2.

SWITCH   (LAMBDA (N) ---

SWITCH REDUCES N MODULUS 16, THEN TESTS THE CONSOLE SWITCH OF THAT NUMBER. IT RETURNS T IF THE SWITCH IS ON, NIL IF OFF. AN ERROR OCCURS IF N IS NOT A NUMBER.

EXAMPLES: (ASSUME PRIME-NUMBERED SWITCHES ARE ON: 2, 3, 5, 7, 11, 13)

```
          (SWITCH 0) = NIL
          (SWITCH 5) = T
          (SWITCH -12) = (SWITCH 4) = NIL
          (SWITCH 99) = (SWITCH 3) = T
```

```
*******************************************************
***** CHAPTER 7 ***** FUNCTIONS ON S-EXPRESSIONS *****
*******************************************************
```

THIS CHAPTER DESCRIBES FUNCTIONS WHICH MANIPULATE S-EXPRESSIONS IN VARIOUS WAYS. NOTE THAT IN GENERAL FUNCTIONS WHICH EXPECT LISTS AS ARGUMENTS WILL MALFUNCTION IF GIVEN DOTTED LISTS, I.E. LISTS NOT ENDING WITH "NIL". THUS (A B C D) IS A VALID ARGUMENT FOR THE FUNCTION "APPEND", BUT (A B.C) ISN'T.

```
********************************************
***** S-EXPRESSION BUILDING FUNCTIONS *****
********************************************
```

```
CONS        (LAMBDA (X Y) ---
```

THE VALUE OF CONS IS THE DOTTED PAIR OF THE S-EXPRESSIONS X AND Y.

```
EXAMPLES:        (CONS 'A 'B) = (A . B)
                 (CONS '(A . B) '(C . D)) = ((A . B) . (C . D))
                                          = ((A . B) C . D)
```

```
LIST        (LAMBDA X X)
```

LIST RETURNS A LIST OF ITS EVALUATED ARGUMENTS.

```
EXAMPLES:        (LIST 'A 'B 'C) = (A B C)
                 (LIST 'A) = (A)
                 (LIST) = NIL
```

```
APPEND      (LAMBDA X ---
```

APPEND TAKES ANY NUMBER OF LISTS AND STRINGS THEM INTO ONE LONG LIST.

```
(SETQQ APPEND (LAMBDA X (COND
      ((NULL X) NIL)
      ((NULL (CDR X)) (CAR X))
      (T (APPLY 'APPEND (CONS
            ((LABEL APPEND1 (LAMBDA (M N) (COND
                  ((NULL M) N)
                  (T (CONS (CAR M) (APPEND1 (CDR M) N)))))) (CAR X) (CADR X))
            (CDDR X)))) )))
```

```
EXAMPLES:        (APPEND '(A B C) '(D E) '(F G H I)) = (A B C D E F G H I)
                 (APPEND '(A B C)) = (A B C)
                 (APPEND) = NIL
```

```
***********************************************
***** S-EXPRESSION FRAGMENTING FUNCTIONS *****
***********************************************
```

```
CAR         (LAMBDA (X) ---
```

CAR OF A NON-ATOMIC S-EXPRESSION IS THE FIRST ELEMENT OF THAT DOTTED PAIR. CAR OF AN IDENTIFIER OR STRING IS ITS PRINT NAME, A LIST OF POINTERS TO A CHARACTER TABLE INTERNAL TO THE LISP SYSTEM. (NOTE: THE CAR OF AN IDENTIFIER OR STRING IS NOT ITSELF A STRING.) THE CAR OF A NUMBER IS UNPREDICTABLE AND THEREFORE UNDEFINED.

CDR (LAMBDA (X) ---

CDR OF A NON-ATOMIC S-EXPRESSION IS THE SECOND (AND LAST) ELEMENT OF THAT DOTTED PAIR. THE CDR OF AN IDENTIFIER IS ITS VALUE IF IT HAS ONE; OTHERWISE IT IS UNDEFINED. THE CDR OF A STRING IS DEFINABLE BUT MEANINGLESS. THE CDR OF A NUMBER YIELDS A POINTER TO THE ADDRESS GIVEN BY THE NUMBER; THIS IS USEFUL, BUT DO NOT DO IT UNLESS YOU KNOW EXACTLY WHAT YOU'RE TRYING TO ACCOMPLISH.

```
EXAMPLES:     (CAR '(A . B)) = A            (CDR '(A . B)) = B
              (CAR '(A B C D)) = A          (CDR '(A B C D)) = (B C D)
```

CAAR, CADR, CDAR, CDDR, CAAAR, CAADR, --- , CADDAADDDAR, --- (LAMBDA (X) ---

NONE OF THE COMPOSITE CAR/CDR FUNCTIONS ARE PREDEFINED IN 1130 LISP. HOWEVER, THERE ARE SPECIAL PROVISIONS IN THE EVAL, APPLY, AND INTERN FUNCTIONS (Q.V.) WHICH AUTOMATICALLY RECOGNIZE SUCH FUNCTIONS WHEN ASKED FOR AND EVALUATE THEM PROPERLY. (NOTE THAT AT SOME POINT THE ATOM WHICH IS THE NAME OF THE FUNCTION MUST HAVE BEEN GIVEN TO INTERN TO PUT ON THE OBLIST.) THUS, ANY COMPOSITE CAR/CDR FUNCTION (WITHIN REASON) MAY BE USED.

```
EXAMPLES:     (CADR X) = (CAR (CDR X))
              (CADDADR X) = (CAR (CDR (CDR (CAR (CDR X)))))
              (CAADADAR X) = (CAR (CAR (CDR (CAR (CDR (CAR X))))))
```

LAST (LAMBDA (X) ---

LAST RETURNS THE LAST PART OF A LIST AS FOLLOWS:

```
(SETQQ LAST (LAMBDA (X) (COND
     ((NULL X) NIL)
     ((NULL (CDR X)) X)
     (T (LAST (CDR X))) )))
```

```
EXAMPLES:     (LAST '(A B C D E)) = (E)
              (LAST '(LAMBDA (X) (COND(X NIL)(T)) )) = ((COND(X NIL)(T)))
```

****
***** S-EXPRESSION MODIFYING FUNCTIONS *****
****

THESE FUNCTIONS, UNLIKE MOST OTHERS, MODIFY EXISTING LIST STRUCTURES RATHER THAN CONSTRUCTING NEW ONES. THERE FUNCTIONS SHOULD BE USED WITH CARE SINCE IT IS VERY EASY TO CREATE LIST STRUCTURES WHICH WILL CONFUSE, HANG UP, OR DESTROY THE LISP INTERPRETER.

RPLACA (LAMBDA (X Y) ---

REPLACES THE CAR OF X WITH Y. THE VALUE IS THE MODIFIED S-EXPRESSION X. NOTE THAT THE RESULT IS EQ TO THE ORIGINAL X.

EXAMPLE: (RPLACA '(A B C) '(C D)) = ((C D) B C)

RPLACD (LAMBDA (X Y) ---

REPLACES THE CDR OF X WITH Y. THE VALUE IS THE MODIFIED S-EXPRESSION X. NOTE THAT THE RESULT IS EQ TO THE ORIGINAL X.

EXAMPLE: (RPLACD '(A B C) '(C D)) = (A C D)

IF THE IDENTIFIER X HAS (A B C) AS ITS VALUE, THEN

(RPLACD (LAST X) X) = (C A B C A B C A B C A B C A B C ---

THIS IS CALLED A CIRCULAR LIST; IT LOOKS INFINITE TO THE PRINT ROUTINE AS WELL AS MOST OTHER FUNCTIONS. SUCH CONSTRUCTS ARE TO BE AVOIDED.

```
**************************************************
***** S-EXPRESSION TRANSFORMING FUNCTIONS *****
**************************************************
```

LENGTH (LAMBDA (X) ---

RETURNS THE NUMBER OF ELEMENTS IN THE LIST X.

```
(SETQQ LENGTH (LAMBDA (X) (COND
     ((NULL X) 0)
     (T (ADD1 (LENGTH (CDR X)))) )))
```

```
EXAMPLES:     (LENGTH '(A B C D E)) = 5
              (LENGTH NIL) = 0
              (LENGTH '((A B) (C D E) 'QUUX (VIOLINIST SPEED FREAK))) = 4
```

REVERSE (LAMBDA (X) ---

RETURNS THE REVERSE OF THE LIST X.

```
(SETQQ REVERSE (LAMBDA (X) ((LABEL REVERSE1 (LAMBDA (M N) (COND
     ((NULL M) N)
     (T (REVERSE1 (CDR M) (COND (CAR M) N))) ))) X NIL)))
```

```
EXAMPLES:     (REVERSE '(A B C D E)) = (E D C B A)
              (REVERSE '((QUUX) (VIOLIN . BERF) ((((BIG . AL))))))
                       = ((((BIG . AL)))) (VIOLIN . BERF) (QUUX))
              (REVERSE NIL) = NIL
```

SUBST (LAMBDA (X Y Z) ---

SUBST SUBSTITUTES X FOR ALL EQUAL OCCURRENCES OF Y IN THE S-EXPRESSION Z.

```
(SETQQ SUBST (LAMBDA (X Y Z) (COND
     ((EQUAL Y Z) X)
     ((ATOM Z) Z)
```

```
     (T (CONS (SUBST X Y (CAR Z)) (SUBST X Y (CDR Z)))) )))
```

(SUBST NIL NIL Z) IS USEFUL FOR CREATING AN INTERNAL COPY OF Z.

```
EXAMPLES:      (SUBST 5 'FIVE '((FIVE + FIVE = 10) (FIVE + TWO = 7)))
                         = ((5 + 5 = 10) (5 + TWO = 7))
               (SUBST '(BIG . AL) '(QUUX) '(QUUX BERF ((QUUX)) (QUUX) (A)))
                         = (QUUX BERF ((BIG . AL)) (BIG . AL) (A))
```

SUBLIS   (LAMBDA (X Y) ---

THE ARGUMENT X OF SUBLIS SHOULD BE A LIST OF PAIRS OF THIS FORM:

```
     ((A1.X1)(A2.X2)(A3.X3) --- (AN.XN))
```

WHERE ALL OF THE AI ARE IDENTIFIERS. THE VALUE OF SUBLIS IS THE RESULT OF SUBSTITUTING EACH X FOR THE CORRESPONDING A IN THE S-EXPRESSION Y. A CLEVER METHOD IS USED WHEREBY AS MUCH OF THE ORGINAL STRUCTURE OF Y IS SHARED WITH THE RESULT AS POSSIBLE.

```
(SETQQ SUBLIS (LAMBDA (X Y) ((LABEL SUBA (LAMBDA (Y) (COND
          ((ATOM Y) ((LABEL SUBB (LAMBDA (Q) (COND
                            ((NULL Q) Y)
                            ((EQ (CAAR Q) Y) (CDAR Q))
                            (T (SUBB (CDR Q)))) )))  X))
          (T ((LAMBDA U V) (COND
                       ((AND (EQUAL (CAR Y) U) (EQUAL (CDR Y) V)) Y)
                       (T (CONS U V)) ))   (SUBA (CAR Y)) (SUBA (CDR Y)) )))))
     Y)))
```

```
EXAMPLES:      (SUBLIS '((A.GUY)(B.STEELE)(C THE GREAT QUUX)) '(A B IS C))
                         = (GUY STEELE IS (THE GREAT QUUX))
               (SUBLIS '((X . ANTE) (Y . CONS))
                              '(LAMBDA (X Y) (COND (X (COND (Y T))) (T))))
                         = (LAMBDA (ANTE CONS) (COND
                              (ANTE (COND (CONS T)))
                              (T)))
```

REMOVE   (LAMBDA (X Y N) ---

REMOVE RETURNS THE RESULT OF REMOVING THE FIRST N EQUAL OCCURRENCES OF X FROM THE LIST Y. IF N<1 NO REMOVALS ARE MADE. IF X OCCURS FEWER THAN N TIMES IN Y, ALL OCCURRENCES ARE REMOVED; THUS, TO REMOVE ALL OCCURRENCES, USE SOMETHING LIKE (REMOVE X Y 32767).

```
(SETQQ REMOVE (LAMBDA (X Y N) (COND
     ((LESSP N 1) Y)
     ((NULL Y) NIL)
     ((EQUAL X (CAR Y)) (REMOVE X (CDR Y) (SUB1 N)))
     (T (REMOVE X (CDR Y) N))) )))
```

```
EXAMPLE:       (REMOVE '(QUUX) '((QUUX)(BERF)QUUX((QUUX))(QUUX)A(QUUX))2)
                         =((BERF) QUUX ((QUUX)) A (QUUX))
```

```
*******************************************
***** S-EXPRESSION MAPPING FUNCTIONS *****
*******************************************
```

MAP (LAMBDA (FN . X) ---

MAP TAKES AS ARGUMENTS A FUNCTION AND ONE OR MORE LISTS OF ARGUMENTS. IF THERE ARE NO ARGUMENT LISTS, MAP DOES NOTHING AND RETURNS NIL. OTHERWISE IT APPLIES FN TO THE LISTS AS ARGUMENTS, THEN TAKES THE CDR OF EACH LIST AND APPLIES FN TO THE RESULTING LISTS, AND SO ON, UNTIL ONE OF THE LISTS IS REDUCED TO NIL. MAP THEN RETURNS NIL.

```
(SETQQ MAP (LAMBDA (FN . X) (COND
      ((NULL X) NIL)
      ((NOT (MEMBER NIL X))
           (APPLY FN X)
           (APPLY 'MAP (CONS FN ((LABEL CDRS (LAMBDA (Q) (COND
                      ((NULL Q) NIL)
                      (T (CONS (CDAR Q) (CDRS (CDR Q)))) ))) X))) ) )))
```

```
EXAMPLES: (MAP '(LAMBDA (X) (PRINT 1 X)) '(A B C D))
     PRINT:    (A B C D)
     PRINT:    (B C D)
     PRINT:    (C D)
     PRINT:    (D)
     RESULT:   NIL
          (MAP '(LAMBDA(X Y)(PRIN1 X Y)(PRINT 1)) '(A B C D) '(X Y Z))
     PRINT:    (A B C D)(X Y Z)
     PRINT:    (B C D)(Y Z)
     PRINT:    (C D)(Z)
     RESULT:   NIL
```

MAPC (LAMBDA (FN . X) ---

MAPC IS SIMILAR TO MAP, EXCEPT THAT MAPC TAKES THE CAR OF EACH LIST BEFORE APPLYING FN.

```
(SETQQ MAPC (LAMBDA (FN . X) (COND
      ((NULL X) NIL)
      ((NOT (MEMBER NIL X))
           (APPLY FN ((LABEL CARS (LAMBDA (Q) (COND
                      ((NULL Q) NIL)
                      (T (CONS (CAAR Q) (CARS (CDR Q)))) ))) X))
           (APPLY 'MAPC (CONS FN ((LABEL CDRS (LAMBDA (Q) (COND
                      ((NULL Q) NIL)
                      (T (CONS (CDAR Q) (CDRS (CDR Q)))) )))  X))) ) )))
```

```
EXAMPLES: (MAPC '(LAMBDA (X) (PRINT 1 X)) '(A B C D))
     PRINT:    A
     PRINT:    B
     PRINT:    C
     PRINT:    D
     RESULT:   NIL
          (MAPC 'SET '(A B C D) '(QUUX BERF VIOLINIST FOOBAZ NURDLE))
```

RESULT IS NIL. AFTER EVALUATION, THE IDENTIFIERS A, B, C, AND D RESPECTIVELY WILL HAVE AS VALUES QUUX, BERF, VIOLINIST, AND FOOBAZ. NOTE THAT SINCE THE LIST (A B C D) IS THE SHORTER, THE LAST ITEM OF THE OTHER LIST ("NURDLE") IS IGNORED.

MAPLIST (LAMBDA (FN . X) ---

MAPLIST IS SIMILAR TO MAP, EXCEPT THAT MAPLIST ACCUMULATES THE RESULTS

OF THE APPLICATIONS OF FN AND RETURNS A LIST OF THEM AS ITS RESULT.

```
(SETQQ MAPLIST (LAMBDA (FN . X) (COND
     ((NULL X) NIL)
     ((NOT (MEMBER NIL X)) (CONS
          (APPLY FN X)
          (APPLY 'MAPLIST (CONS FN ((LABEL CDRS (LAMBDA (Q) (COND
                    ((NULL Q) NIL)
                    (T (CONS (CDAR Q) (CDRS (CDR Q)))) )))  X))) )) )))
```

```
EXAMPLES: (MAPLIST 'REVERSE '(A B C D)) = ((D C B A) (D C B) (D C) (D))
          (MAPLIST 'CONS '(A B C D E) '(X Y Z))
                    = (((A B C D E) X Y Z) ((B C D E) Y Z) ((C D E) Z))
```

MAPCAR   (LAMBDA (FN . X) ---

MAPCAR IS SIMILAR TO MAPC, EXCEPT THAT MAPCAR ACCUMULATES THE RESULTS OF THE APPLICATIONS OF FN AND RETURNS A LIST OF THEM AS ITS RESULT.

```
(SETQQ MAPCAR (LAMBDA (FN . X) (COND
     ((NULL X) NIL)
     ((NOT (MEMBER NIL X)) (CONS
          (APPLY FN ((LABEL CARS (LAMBDA (Q) (COND
                    ((NULL Q) NIL)
                    (T (CONS (CAAR Q) (CARS (CDR Q)))) )))  X))
          (APPLY 'MAPCAR (CONS FN ((LABEL CDRS (LAMBDA (Q) (COND
                    ((NULL Q) NIL)
                    (T (CONS (CDAR Q) (CDRS (CDR Q)))) )))  X))) )) )))
```

```
EXAMPLES: (MAPCAR 'EQ '(A B C D E) '(X B Z D)) = (NIL T NIL T)
          (MAPCAR 'SUBST '(A B C) '(X Y Z) '((X IS Y) (Y IS Z) (X IS Z)))
                    = ((A IS Y) (B IS Z) (X IS C))
          (MAPCAR 'ATOM '(A (B) ((C)) D (E) F)) = (T NIL NIL T NIL T)
```

****
***** S-EXPRESSION SEARCHING FUNCTIONS *****
****

ASSOC   (LAMBDA (X L) ---

ASSOC SEARCHES A LIST OF PAIRS L FOR A PAIR WHOSE CAR IS EQ TO X. IF SUCH A PAIR IS FOUND, ASSOC RETURNS THAT PAIR; OTHERWISE IT RETURNS NIL.

```
(SETQQ ASSOC (LAMBDA (X L) (COND
     ((NULL L) NIL)
     ((EQ X (CAAR L)) (CAR L))
     (T (ASSOC X (CDR L))) )))
```

```
EXAMPLE:  (ASSOC 'TWO '((ONE.EINS)(TWO.ZWEI)(THREE.DREI))) = (TWO . ZWEI)
```

SASSOC   (LAMBDA (X L FN) ---

SASSOC DOES WHAT ASSOC DOES TO X AND L; BUT IF NO PAIR IS FOUND, INSTEAD OF NIL SASSOC RETURNS THE VALUE OF FN, A FUNCTION OF NO ARGUMENTS.

```
(SETQQ SASSOC (LAMBDA (X L FN) (COND
```

```
      ((NULL L) (FN))
      ((EQ X (CAAR L)) (CAR L))
      (T (ASSOC X (CDR L) FN)) )))
```

EXAMPLE: (SASSOC 'THREE '((ONE.EINS)(TWO.ZWEI)) '(LAMBDA NIL 'LOSE))
= LOSE

```
******************************************************
***** CHAPTER 8 ***** FUNCTIONS ON IDENTIFIERS *****
******************************************************
```

THESE FUNCTIONS PERFORM VARIOUS OPERATIONS INVOLVING IDENTIFIERS, INCLUDING ALTERING THEIR VALUES, CREATING THEM, AND MAINTAINING THE OBLIST.

```
************************************************
***** IDENTIFIER VALUE-ALTERING FUNCTIONS *****
************************************************
```

SET (LAMBDA (X Y) ---

GIVES THE IDENTIFIER X THE VALUE Y. ERROR IF X IS NOT A NON-NIL IDENTIFIER. THE VALUE OF SET IS THE VALUE Y.

```
(SETQQ SET (LAMBDA (X Y) (COND
     ((NOT (ATOMP X)) (ERROR 36 BAD FIRST ARG FOR SET/SETQ/SETQQ))
     (T (RPLACD X Y) Y))))
```

SETQ (NLAMBDA (X Y) (SET X (EVAL Y)))

SETQ IS SIMILAR TO SET, BUT DOES NOT EVALUATE ITS FIRST ARGUMENT.

SETQQ (NLAMBDA (X Y) (SETQ X Y))

SETQQ IS SIMILAR TO SET, BUT DOES NOT EVALUATE EITHER ARGUMENT. ESPECIALLY USEFUL FOR DEFINING FUNCTIONS AND INITIALIZING VARIABLES.

```
******************************************
***** OBLIST MANAGEMENT FUNCTIONS *****
******************************************
```

INTERN (LAMBDA (X) ---

INTERN TAKES AS ITS ARGUMENT AN IDENTIFIER OR A STRING (ERROR IF NOT). IF AN ATOM WITH THE SAME PRINT NAME IS ON THE OBLIST, THAT ATOM IS FOUND AND RETURNED. IF NOT, SUCH AN ATOM IS CREATED AND PUT ON THE OBLIST IN ALPHABETICAL ORDER AND RETURNED. THE CREATED ATOM WILL HAVE AN UNDEFINED VALUE, UNLESS ITS PRINT NAME CONSISTS OF A "C" FOLLOWED BY ZERO OR MORE "A"S AND "D"S FOLLOWED BY AN "R", IN WHICH CASE THE ATOM IS GIVEN A SPECIAL FUNCTIONAL VALUE, CONSISTING OF THE DOTTED PAIR OF THE IDENTIFIER C-R AND THE CREATED ATOM. THIS VALUE WILL CAUSE EVAL AND APPLY TO SEE THIS AS A COMPOSITE CAR/CDR FUNCTION.

REMOB (NLAMBDA (X) ---

REMOB REMOVES THE GIVEN IDENTIFIER FROM THE OBLIST AND RETURNS NIL. IF THE IDENTIFIER IS NOT ON THE OBLIST, NO ACTION IS TAKEN. NOTE THAT X IS NOT EVALUATED.

```
******************************************
***** IDENTIFIER CREATION FUNCTIONS *****
******************************************
```

GENSYM (LAMBDA X ---

GENSYM TAKES ZERO OR ONE ARGUMENTS. GENSYM INCREMENTS THE "GENERATED SYMBOL COUNTER" AND CREATES A NEW IDENTIFIER AS SPECIFIED BY THIS COUNTER. THIS IDENTIFIER IS AUTOMATICALLY GIVEN TO INTERN (Q.V.) AND THEN RETURNED. (NOTE THAT GENSYM'S IN MOST OTHER LISPS DO NOT INTERN THE CREATED ATOMS.) IF AN ARGUMENT IS GIVEN, IT MUST BE A STRING, WHICH IS USED TO INITIALIZE THE GENERATED SYMBOL COUNTER, AND THEN A NEW IDENTIFIER IS CREATED, INTERNED, AND RETURNED. THE GENERATED SYMBOL COUNTER IS INITIALLY SET TO ,QX999,; THUS, THE FIRST SYMBOLS GENERATED WILL BE QX000, QX001, ETC.

```
EXAMPLE:  (GENSYM) = QX000
          (GENSYM) = QX001
          (GENSYM) = QX002
          (GENSYM ,BARF596,) = BARF596
          (GENSYM) = BARF597
          (GENSYM) = BARF598
          (GENSYM ,VIOLINIST9,) = VIOLINIST9
          (GENSYM) = VIOLINIST0
          (GENSYM) = VIOLINIST1
          (GENSYM ,PETERBAKER999BIGAL9999998,) = PETERBAKER999BIGAL9999998
          (GENSYM) = PETERBAKER999BIGAL9999999
          (GENSYM) = PETERBAKER999BIGAL0000000
          (GENSYM) = PETERBAKER999BIGAL0000001
          (GENSYM) = PETERBAKER999BIGAL0000002          ETC.
```

```
************************************************
***** CHAPTER 9 ***** FUNCTIONS ON NUMBERS *****
************************************************
```

THE FOLLOWING FUNCTIONS EXPECT NUMBERS AS ARGUMENTS. IF ANY ARGUMENT IS NOT A NUMBER AN ERROR OCCURS. ALL ARGUMENTS ARE EVALUATED.

THE FUNCTIONS WHICH TAKE AN INDEFINITE NUMBER (ONE OR MORE) OF ARGUMENTS ALL WORK THE SAME WAY: THE OPERATION IS PERFORMED ON THE FIRST TWO, THEN ON THE RESULT AND THE THIRD, THEN ON THE RESULT AND THE FOURTH, ETC. THUS:

```
(DIFF A B C D E) = (DIFF (DIFF (DIFF (DIFF A B) C) D) E)
```

IF ONLY ONE ARGUMENT IS GIVEN, IT IS RETURNED: (DIFF 5) = 5. NOTE THAT THE FUNCTION BOOLE TAKES TWO OR MORE ARGUMENTS, BUT OTHERWISE WORKS IN MUCH THE SAME WAY.

```
********************************
***** ARITHMETIC FUNCTIONS *****
********************************
```

```
PLUS      (PLUS X1 X2 X3 --- XN) = X1+X2+X3+ --- +XN         (ADDITION)

DIFF      (DIFF X1 X2 X3 --- XN) = X1-X2-X3- --- -XN         (SUBTRACTION)
```

NOTE THAT IN OTHER LISPS THIS FUNCTION IS NAMED "DIFFERENCE".

```
TIMES     (TIMES X1 X2 X3 --- XN) = X1*X2*X3* --- *XN        (MULTIPLICATION)

QUOTIENT  (QUOTIENT X1 X2 X3 --- XN) = X1/X2/X3/ --- /XN     (DIVISION)
```

NOTE THAT SINCE 1130 LISP NUMBERS ARE INTEGERS ONLY, THE RESULT OF A DIVISION OPERATION IS TRUNCATED TO THE INTEGER NEXT LOWEST IN ABSOLUTE VALUE. THUS: 5/3=1 11/2=5 (-12)/5=-2

```
MINUS     (MINUS X) = -X                                     (NEGATION)

REMAINDER (REMAINDER X1 X2 X3 --- XN) = X1%X2%X3% --- %XN    (REMAINDER)
```

WHERE A%B = A-(A/B)*B

```
MAX       (MAX X1 X2 X3 --- XN) = LARGEST OF NUMBERS X1 X2 X3 --- XN  (MAXIMUM)

MIN       (MIN X1 X2 X3 --- XN) = SMALLEST OF NUMBERS X1 X2 X3 --- XN (MINIMUM)

GCD       (GCD X1 X2 X3 --- XN) = GREATEST COMMON DENOMINATOR OF X1 X2 X3 --- XN

EXPT      (EXPT X Y) = X**Y                                  (EXPONENTIATION)
```

A COMPLEX ALGORITHM IS USED SO THAT EXPRESSIONS LIKE 4**0, (-1)**N,

1**N, AND N**(-14) WILL EVALUATE TO EXPECTED VALUES.

ADD1 (ADD1 X) = X+1

SUB1 (SUB1 X) = X-1

ABS (ABS X) = |X| (ABSOLUTE VALUE)

****
***** LOGICAL FUNCTIONS *****
****

BOOLE (BOOLE K X1 X2 X3 --- XN) = X1?X2?X3? --- ?XN

WHERE ? IS ONE OF SIXTEEN BOOLEAN FUNCTIONS WHICH IS APPLIED BIT BY BIT TO TWO SIXTEEN-BIT LOGICAL NUMBERS. FOR EACH BIT IN A AND B, THE RESULTING BIT IS DEFINED AS FOLLOWS:

```
K    RESULT                              K    RESULT
--------------------------------         --------------------------------
0    0                                   8    (^A)&(^B)        ^(A|B)
1    A&B              AND                9    A=B              EQUIVALENCE
2    (^A)&B                              10   ^A
3    B                                   11   (^A)|B           IMPLIES
4    A&(^B)                              12   ^B
5    A                                   13   A|(^B)
6    ^(A=B)           EXCLUSIVE OR       14   (^A)|(^B)        ^(A&B)
7    A|B              INCLUSIVE OR       15   1
```

WHERE &, |, ^, AND = REPRESENT THE LOGICAL AND, LOGICAL OR, LOGICAL NOT, AND LOGICAL EQUIVALENCE FUNCTIONS RESPECTIVELY. NOTE THAT IF K<0 OR K>15, K IS REDUCED MODULUS 16.

EXAMPLES:
(BOOLE 1 /FF00 /F00F) = /F000
(BOOLE 7 /0123 /1234 /2345) = /3377
(BOOLE 10 /5678 0) = /A987
(BOOLE 10 /5678) = /5678
(BOOLE 48 3 5) = (BOOLE 0 3 5) = 0

LSH (LSH X Y) = RESULT OF A LOGICAL SHIFT OF THE NUMBER X BY Y PLACES
LSH PERFORMS A LOGICAL SHIFT ON X OF Y BITS: TO THE LEFT IF Y IS POSITIVE, TP THE RIGHT IF NEGATIVE. THE ABSOLUTE VALUE OF Y IS REDUCE MODULUS 64 BEFORE THE SHIFT IS PERFORMES. BITS SHIFTED OUT ARE LOST; ZERO BITS ARE SHIFTED IN.

EXAMPLES:
(LSH /1234 1) = /2468
(LSH /FACE 0) = /FACE
(LSH /F947 -7) = /01C2
(LSH X 16) = (LSH X -16) = 0

****
***** RANDOM NUMBER GENERATOR *****
****

RANDOM   (RANDOM N) = RANDOM NUMBER FROM Ø TO N-1

IF N IS ANY POSITIVE NUMBER, RANDOM RETURNS A RANDOM NUMBER FROM Ø TO N-1, WITH EQUAL PROBABILITY GIVEN TO EACH CHOICE. FOR N=Ø RANDOM PROCEEDS TO DO TWO DISK SEEKS WHICH PROVIDE RANDOM TIMING FOR RANDOMIZING THE SEED USED TO GENERATE RANDOM NUMBERS. (NOTE: THIS OPERATION TAKES ON THE ORDER OF .1 SECOND TO 1 SECOND.) THE POWER-RESIDUE METHOD OF PSEUDO-RANDOM NUMBER GENERATION IS EMPLOYED.