AN INTRODUCTION TO LISP 1.5 PROGRAMMING LANGUAGE

AND

ITS IBM 1620 INTERPRETER

THOMAS A. BRODY

UNIVERSIDAD AUTONOMA DE MEXICO
INSTITUTO DE FISICA
CIUDAD UNIVERSITARIA
MEXICO 20, M

GEORGES SCHWACHHEIM
ADILSON T. DE MEDEIROS
CENTRO BRASILEIRO DE PESQUISAS FISICA
DIVISAO DO COMPUTADOR
RIO DE JANEIRO - ZC 82
BRASIL

TABLE OF CONTENTS

****
* PREFACE *.
****

THE PRESENT ATTEMPT TO CREATE A SYMBOL MANIPULATION LANGUAGE PROCESSOR FOR A RELATIVELY SMALL COMPUTER WAS MADE BECAUSE OF THE POSSIBILITY IT WOULD OFFER TO OPEN UP THE FIELD OF SYMBOL MANIPULATION TO THE MANY GROUPS WHO DO NOT HAVE A LARGE COMPUTER AT THEIR DISPOSAL. THIS IS THE CASE, FOR INSTANCE, IN LATIN AMERICA AND IN SEVERAL EUROPEAN COUNTRIES. THE RESULTING PROCESSOR HAS PROVED USEFUL BOTH IN THE TEACHING OF SYMBOL MANIPULATION TECHNIQUES AND FOR SEVERAL OTHER APPLICATIONS OF A NOT TOO EXTENSIVE CHARACTER.

THE PROCESSOR DESCRIBED IN THIS MANUAL WILL RUN ON AN I.B.M. 1620 MOD. II MACHINE EQUIPPED WITH THE FOLLOWING ADDITIONAL FEATURES -

AT LEAST 40000 POSITIONS OF CORE MEMORY
INDEX REGISTERS FEATURE
1311 DISK STORAGE UNIT

IF MORE THAN 40000 POSITIONS OF CORE STORAGE ARE AVAILABLE THE INTERPRETER IS AUTOMATICALLY MODIFIED TO MAKE USE OF ADDITIONAL MEMORY. ANY INPUT OR OUTPUT MEDIUM, CARD READ/PUNCH, PAPER TAPE READER/PUNCH, TYPEWRITER, 1443 HIGH SPEED PRINTER, CAN BE USED IF AVAILABLE.

1. COMPUTERS AND LANGUAGES

SINCE 1945, WHEN THE FIRST COMPUTER WAS INTRODUCED, THE ITERA TION MAN-MACHINE BECOMES INDISPENSABLE. IN THIS FEW YEARS THAT COME TO PASS THE USE OF THE MACHINE IN MANY FIELDS OF HUMAN ATIVITY WAS SO DIRECT AND UNDERTOOK TOO EFFICIENT THAT MAN WITH THE ADVANCE OF ELETRONICS IMPROVE HIS OWN MACHINES AND THE METHOD OF HIS USE. THUS, THE DEVELOPMENT OF COMPUTERS SCIENCE WAS VERY FAST AND TODAY MACHINE FULFIL AN IMPORTANT PLACE IN ALL THE SOCIALS ACTIVITIES.

THE PROGRAMMING LANGUAGES HAD TOO THEIR OWN DEVELOPMENT. AT THE BEGINNING THE COMPUTERS, BUILT BY SMALL GROUPS OF ELETRONIC ENGINEERS MAINLY FOR THEIR OWN USE, THE CONTACT MAN-MACHINE WAS MADE USUALLY IN MACHINE-LANGUAGE.

FURTHER ON, WITH THE GENERAL APPEARANCE OF COMMERCIALLY BUILT COMPUTERS, METHODS HAVE BEEN DEVELOPED WHICH ALLOW THE MACHINE ITSELF TO TAKE OVER THE ROUTINE PARTS OF PROGRAMMING. FOR THIS PURPOSE, A NEW LANGUAGE IS DEFINED IN WHICH COMPUTER INSTRUCTIONS ARE WRITTEN, AND A SPECIAL PROGRAM CALLED A PROCESSOR ACCEPTS STATEMENTS IN THIS LANGUAGE AS DATA, AND PRODUCES AS OUTPUT THE TRANSLATION INTO MACHINE LANGUAGE.

PROCESSORS ARE MOSTLY OF TWO TYPES, ASSEMBLERS AND COMPILERS, AND EACH HAS ITS CORRESPONDING LANGUAGES.

BY MEANS OF A VERY SIMPLE MACHINE LANGUAGE THE MACHINE READS AN INSTRUCTION OF THE TYPE -

```
21 10000 15000
```

(MEANING CARRY OUT OPERATION 21, ADDING, FOR EXAMPLE, THE DATA IN LOCATION 15000 OF MEMORY TO DATA IN LOCATION 10000). BUT, THIS KIND OF INSTRUCTION FORCES THE USER TO LEARN WHAT KIND OF OPERATION IS CARRIED OUT BY THE MACHINE IN READING 21 AND ALSO TO TAKE INTO ACCOUNT THE PRECISE ADDRESS WHERE IT IS GOING TO BE USED.

AN ASSEMBLER COULD BE CALLED A MORE SOPHYSTICATED LANGUAGE AND WILL ACCEPT INSTRUCTIONS OF THE FORM

```
ADD A,B
```

THAT WILL BE TRANSLATED INTO A CONVENIENT MACHINE LANGUAGE.

NEVERTHELESS, FOR SUCH AN ASSEMBLER, ONE MUST STILL WRITE ONE INSTRUCTION IN THE ASSEMBLY LANGUAGE FOR EACH MACHINE LANGUAGE INSTRUCTION PRODUCED. FOR A LARGE PROGRAM, AND PROGRAMS WITH MANY THOUSANDS OF INSTRUCTIONS ARE NO LONGER A RARITY, THIS IS STILL A LENGTHY AND BORING TASK.

BY MEANS OF A COMPILER WE CAN GIVE INSTRUCTIONS IN THE FORM OF ALGEBRAIC FORMULAS WRITING SOMETHING LIKE -

$$X = Z * (A + B + 3)$$

MEANING - CALL X THE QUANTITY OBTAINED FROM QUANTITIES Z,A,B AND 3 SUBJECTED TO THE INDICATED OPERATIONS. IN THIS MANNER ONE CAN HANDLE PROBLEMS THAT ARE VERY COMPLICATED IN A SIMPLE FORM. WITH THESE TECHNIQUES THE PROGRAMMING OF NUMERICAL PROBLEMS HAS BEEN SIMPLIFIED TO THE POINT WHERE IT SURELY RETARDS SERIOUSLY THE SOLUTION OF THE PROBLEM. THUS BASE HAVE APPEARED LANGUAGES SUCH FORTRAN, ALGOL, PL/1 AND OTHERS THAT SUPPLY THE USERS THE SOLUTION OF NUMERICAL PROBLEMS. BUT, SIDE BY SIDE WITH THE IDEA OF LANGUAGES TO SOLVE NUMERICAL PROBLEMS, CAME ALSO THE IDEA OF LANGUAGES TO SOLVE NON-NUMERICAL PROBLEMS, FOR THE SOLUTION ON THE COMPUTER OF MANY ESSENTIALLY LOGICAL PROBLEMS AS THE HANDLING OF ALGEBRAIC EXPRESSIONS AND INFORMATION HANDLING SUCH AS INFORMATION RETRIEVAL, DETECTION OF RELEVANT EXPERIMENTAL DATA, GAMES THEORY, PATTERN RECOGNITION AND OTHER NON-NUMERICAL QUESTIONS.

THE RESEARCH ON THIS FIELD OF LANGUAGES THAT HAVE BEEN CALLED SYMBOL MANIPULATION LANGUAGES HAD ALSO A FAST DEVELOPMENT. THE FIRST SUCCESFULL ATEMPT WAS MADE BY A GROUP OF THREE PSYCHOLOGISTS, NEWELL, SIMON AND SHAW (1956-1957), WHO ATEMPTED TO SIMULATE HUMAN INTELLIGENCE WITH A MACHINE. WHAT IS IMPORTANT IN THEIR WORK IS A METHOD OF DYNAMIC MEMORY ALLOCATION. IN A NUMERICAL COMPILER THE ADDRESSES WHERE ALL QUANTITIES WILL BE STORED ARE FIXED AT THE TIME OF TRANSLATION, BUT IN USING A COMPUTER TO SOLVE LOGICAL PROBLEMS IT IS GENERALLY NOT POSSIBLE AT THE BEGINNING TO MAKE AN ESTIMATION OF WHAT KIND AND QUANTITY OF INFORMATION IS TO BE USED AT A PARTICULAR STAGE-THEREFORE THESE AUTHORS INCLUDED THE POSSIBILITY OF A CONTINUAL CHANGE OF LOCATION OF DATA AND ALSO THE POSSIBILITY OF RECOVERING MEMORY LOCATIONS WHICH HOLD INFORMATION THAT WILL NO LONGER BE NEEDED.

NEWELL, SIMON AND SHAW WROTE A COMPUTER LANGUAGE TO HANDLE THESE THINGS - THE INFORMATION PROCESSING LANGUAGE, CALLED LATTER IPL - V. THE MOST IMPORTANT CONCEPT FEATURE INTRODUCED BY THESE AUTHORS ON THE STRUCTURE OF THE LANGUAGE WAS THE CONCEPT OF A LIST STRUCTURE THAT WILL BE STUDIED IN CHAPTER TWO.

BETWEEN 1958 AND 1959, AT MIT, JONH MCCARTHY AND OTHERS DEVELOPED A SYMBOL MANIPULATION LANGUAGE CALLED LISP (LIST PROCESSING) THAT WILL BE SUBJECT OF THIS MANUAL. THIS LANGUAGE HAS BEEN LARGELY USED IN HIS FIELD OF APPLICATIONS BECAUSE OF ITS EXTREMELY SIMPLE NOTATION. TAKING ADVANTAGE OF THE IDEA OF AUTHORS OF IPL-V, THE STRUCTURAL FEATURES OF THE LANGUAGE ARE REPRESENTED BY MEANS OF LIST STRUCTURES. IT USES ALSO THE GENERAL IDEA OF RECURSIVE FUNCTION DEFINITION THAT WILL BE SUBJECT OF CHAPTER THREE.

* 2. LIST STRUCTURE *

* 2.1 ATOMS AND LISTS *

THE PROBLEM WE ARE CONCERNED WITH IS HOW TO PRESENT DATA WHOSE STRUCTURE MAY BE FAR MORE COMPLEX THAN THAT OF A NUMBER OR ARRAY TO THE COMPUTER. THIS CAN BE ACHIEVED BY MEANS OF LIST STRUCTURE, A METHOD WHICH WAS FIRST INTRODUCED BY NEWELL, SIMON AND SHAW.

TO INTRODUCE THE CONCEPT OF LIST STRUCTURE, WE NEED FIRST TO INTRODUCE ANOTHER ONE - THE CONCEPT OF ATOM. AN ATOM CAN BE DEFINED AS ANY SEQUENCE OF LETTERS, NUMBERS OR OTHER SYMBOLS EXCEPT FOR THE RIGHT AND LEFT PARENTHESIS AND BLANK SPACES.

* EXAMPLES *

A
A5
COMPUTER
EVEN
COMPUTER1
RIODEJANEIROISAGREATCITY

ON THE IBM 1620 AN ATOM NAME MAY NOT EXCEED 30 CHARACTERS IN LENGTH.

A LIST IS AN ARRANGEMENT OF ATOMS. WE CAN REPRESENT A SIMPLE LIST BY ENCLOSING THE SEQUENCE OF ATOMS COMPOSING IT IN PARENTHESES.
FOR INSTANCE

(A B C)

THIS LIST IS COMPOSED OF THREE ATOMS - A, B AND C. NOTE THE STRUCTURE OF THIS REPRESENTATION. IT BEGINS WITH A LEFT PARENTHESES, THE ATOMS SEPARATE BY BLANK SPACES AND AT LAST, A RIGHT PARENTHESES.

A LIST MAY HAVE SUBLISTS, AND THESE SUBLISTS MAY ALSO HAVE SUBLISTS. IT IS CONVENIENT TO SPEAK OF ELEMENTS OF A LIST. AN ELEMENT MAY BE AN ATOM, A LIST OF ATOMS, A LIST OF LISTS OR A LIST OF ATOMS AND LISTS.

* EXAMPLES *

1)

((A B)(C D)(E F) F)

COMMENTS - THIS IS A LIST OF FOUR ELEMENTS. THE FIRST ELEMENT IS THE LIST - (A B), THE SECOND, THE LIST (C D), THE THIRD, THE LIST (E F) AND THE FOURTH ELEMENT, THE ATOM F.

2)

(((A B) C D) E F)

COMMENTS - THIS IS A LIST OF THREE ELEMENTS. THE FIRST IS THE LIST ((A B) C D) AND THE OTHERS, THE ATOMS E AND F.

3)

((G1 (H5 I)(F (M O P4) S) T V))

COMMENTS - THIS IS A LIST WITH ONE ELEMENT - THE LIST
(G1 (H5 I)(F (M O P4) S) T V)

4)

((THIS IS A (LIST) STRUCTURE IN) LISP)

5)

(THIS IS A LIST OF ATOMS)

IT MUST BE NOTED THAT WE CAN HAVE A LIST WITH NO ELEMENTS CALLED - EMPTY-LIST AND REPRESENTED BY

( )

IN DEALING WITH LOGICAL PROBLEMS IT IS IMPORTANT TO REMEMBER THE DISTINCTION BETWEEN AN OBJECT ANS ITS NAME. IN ORDINARY DISCOURSE THERE ARE RARELY PROBLEMS SINCE NAME AND OBJECT ARE OF DIFFERENT NATURE. NO ONE WILL FALL INTO CONFUSION BETWEEN

MEXICO HAS 35 MILLION INHABITANTS

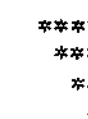
AND

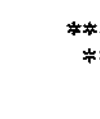
MEXICO HAS SIX LETTERS

IF PRECISION IS REQUIRED ONE WRITES THAT THE NAME MEXICO HAS SIX LETTERS OR ELSE ONE USES QUOTES TO SIGNAL THAT THE NAME ITSELF IS MEANT RATHER THAN THE OBJECT NAMED.

IN LISP THE DISTINCTION HAVE TO BE MADE CLEARLY, SINCE MES AND OBJECTS MAY BE OF THE SAME TYPE, THAT IS, ATOMS. MOREOVER, THE OBJECT DICATED BY A NAME MAY ITSELF BE A NAME FOR ANOTHER OBJECT. THE SITUATION IS MILAR TO THAT OF MATHEMATICAL NOTATION - 5 + A WHICH CANNOT BE EVALUATED TIL WE KNOW WHAT A STANDS FOR, THAT IS, OF WHAT NUMBER IT IS THE NAME.

WE WILL LEARN AT CHAPTER 5 HOW TO INTRODUCE NAMES AND THE SPECTIVELY OBJECTS. BY NOW, ONE MUST REMEMBER THAT LISTS OR ATOMS MAY HAVE NA S.





3. RECURSIVE FUNCTIONS

TO INTRODUCE THE CONCEPT OF RECURSIVE FUNCTIONS, LET US CONSIDER A SIMPLE PROBLEM - WE WISH TO KNOW HOW MANY ELEMENTS THERE ARE IN A GIVEN LIST.

LET US WRITE IN ENGLISH THE STEPS TO SOLVE OUR PROBLEM.

1) LET N BE A COUNTER FOR THE NUMBER OF ELEMENTS OF A LIST.

2) INITIALIZE N TO ZERO.

3) TAKE THE NEXT ELEMENT OF THE LIST.

4) ADD 1 TO COUNTER.

5) IF THE LIST HAS MORE ELEMENTS, BACK TO STEP 3. OTHERWISE, END THE PROCESS AND THE RESULT IS GIVEN BY THE COUNTER.

THIS METHOD OF COUNTING HAS THE FOLLOWING CHARACTERISTICS -

1) EACH CYCLE OF THE PROCESS (FROM ITEM 3 TO ITEM 5) IS COMPLETELY EXECUTED BEFORE THE NEXT CYCLE IS BEGUN, AND THE DECISION ON WHENEVER TO CONTINUE THE CYCLE IS USUALLY TAKEN AT THE END OF THE CYCLE, USING THE RESULTS OBTAINED DURING THE CURRENT EXECUTION OF IT.

2) THE RESULTS OBTAINED IN EACH CYCLE OF EXECUTION SERVE AS STARTING POINT FOR THE NEXT CYCLE, BUT THEREAFTER THEY CAN BE ABANDONED - IN PRACTICE, THE RESULTS OF EACH EXECUTION OF THE CYCLE OVERWRITE THOSE OF THE PREVIOUS ONE.

A DIFFERENT TYPE OF PROCEDURE IS CALLED RECURSIVE. A PROCE - DURE IS CALLED RECURSIVE WHEN IT IS USED WITHIN ITS OWN DEFINITION. FOR INSTAN- CE, THE PROBLEM DESCRIBED ABOVE COULD BE SOLVED RECURSIVELY AS FOLLOWS -

1) IF THE LIST IS EMPTY THEN THE NUMBER OF ITS ELEMENTS IS ZERO.

2) IF THE LIST IS NOT EMPTY, THE NUMBER OF ITS ELEMENTS WILL BE 1 PLUS THE NUMBER OF ELEMENTS OF THE LIST OBTAINED BY REMOVING FROM THE ORIGINAL LIST ITS FIRST ELEMENT.

WE SEE THAT THIS METHOD FOR COUNTING THE NUMBER OF ELEMENTS USE ITSELF IN THE SECOND PART OF THE DEFINITION THAT IS CARACTERISTIC OF A RECUR SIVE PROCEDURE.

LET US FOLLOW STEP BY STEP THIS METHOD FOR THE LIST (A B). (A B) IS NOT AN EMPTY LIST, SO THE NUMBER OF ITS ELEMENTS IS EQUAL TO 1 PLUS THE NUMBER OF ELEMENTS OF THE LIST (B). WE MUST REMEMBER THIS CONCLUSION AND TRY TO DETERMINE THE NUMBER OF ELEMENTS OF (B) BY THE SAME METHOD.

AS (B) IS NOT EMPTY, AGAIN THE NUMBER OF ITS ELEMENTS IS EQUAL TO ONE PLUS THE NUMBER OF ELEMENTS OF AN EMPTY LIST. HERE THE METHOD GIVE DIRECTLY THE INDICATION THAT THE NUMBER OF ELEMENTS IS ZERO. GOING BACKWARDS WE CAN NOW COMPLETE THE DETERMINATION OF THE NUMBER OF ELEMENTS OF THE LIST (B). IT IS EQUAL TO 1 PLUS ZERO, THAT IS, 1.

WE CAN USE THIS RESULT TO COMPLETE THE DETERMINATION OF THE NUMBER OF ELEMENTS OF THE LIST (A B), EQUAL (AS WE REMEMBER) TO 1 PLUS 1, GIVING US THE FINAL RESULT THAT THE LIST (A B) HAS TWO ELEMENTS.

ALTHOUGH THE MECHANISM OF THIS SECOND METHOD IS MUCH MORE INVOLVED THAN FOR THE FIRST ONE, THE DEFINITION OF THE METHOD IS SIMPLER. THIS TENDS TO BE A RULE FOR THE RECURSIVE METHOD.

THIS TYPE OF PROCEDURE ARE CHARACTERIZED BY -

A) EACH CYCLE OF RECURSION (WHICH WILL BE CALLED A LEVEL IN WHAT FOLLOWS) IS NOT COMPLETED BEFORE THE NEXT IS BEGUN.

B) PARTIALLY COMPLETED RESULTS FROM ALL PREVIOUS LEVELS MUST BE KEPT AVAILABLE, SO THAT THEY MAY BE PICKED UP AGAIN ON RETURN.

C) THE POINT AT WHICH THE PROGRAM REPRESENTING THE RECURSIVE FUNCTION WAS LEFT MUST BE REMEMBERED, SINCE ON RETURNING WE MUST PICK UP EXECUTION WHERE IT LEFT ON LEAVING THIS LEVEL.

TO RESUME, IN A RECURSIVE PROCEDURE, ALL LEVELS ARE CALLED UP TO MAXIMUM RECURSION DEPTH, BUT THEIR EXECUTION IS ONLY COMPLETED UPON COMING BACK UP. ALMOST THE FUNCTIONS USED BY LISP LANGUAGE ARE DEFINED BY RECURSION SO THAT IT IS INTERESTING TO KNOW WHAT KIND OF FUNCTIONS MAY BE DEFINED RECURSIVELY - IT CAN BE SHOWN THAT ALL COMPUTABLE FUNCTIONS CAN BE DEFINED BY RECURSION.

THE CONCEPT OF COMPUTABLE FUNCTIONS IS AT FIRST RATHER VAGUE AND INTUITIVE, BUT TWO CONDITIONS MUST EVIDENTLY BE FULFILLED.

A) THERE MUST BE A CERTAIN PROCEDURE, TO BE CARRIED OUT UNAMBIGUOUSLY, WHICH DEFINES THE FUNCTIONS.

B) THE NUMBER OF STEPS INVOLVED MUST BE FINITE.

WHEN WE CONSTRUCT A RECURSIVE FUNCTION, IT IS NECESSARY THAT FOR SOME VALUES OF THE ARGUMENT THE FUNCTION IS DEFINED NON-RECURSIVELY. OTHERWISE, THE RECURSION WILL NOT END, AND ITS VALUE (IF IT EXISTS) COULD NOT BE OBTAINED IN A FINITE NUMBER OF STEPS. THE NON-RECURSIVE PART OF THE DEFINITION IS KNOWN AS THE TERMINAL CONDITION.

THE RECURSION MAY BE COMPLICATED. THE FUNCTION MAY HAVE SEVERAL ARGUMENTS, IN ITS DEFINITION SEVERAL TERMINAL CONDITIONS AND SEVERAL RECURSIVE ELEMENTS MAY OCCUR. THE RECURSION MAY ALSO BE INDIRECT, AS WHEN A FUNCTION F CALLS ANOTHER FUNCTION G, WHICH IN TURNS CALL F AND SO ON.

4. BASIC CONCEPTS OF LISP

THE EXECUTION OF A LISP PROGRAM CONSISTS TO A SERIES OF EVALUATIONS OF LIST STRUCTURES.

WE MUST DEFINE THE CONCEPT OF EVALUATION RECURSIVELY. THE EVALUATION OF A LIST IS MADE ELEMENT BY ELEMENT, FROM LEFT TO RIGHT. WHEN A GIVEN ELEMENT IS NOT AN ATOM BUT A SUBLIST, THIS SUBLIST IS EVALUATED BEFORE PROCEEDING WITH THE EVALUATION OF THE LIST. THE RESULT CAN BE -

A LIBRARY DEFINED FUNCTION OF LISP.

A PREDICATE

A LITERAL

A FUNCTION

A LIBRARY DEFINED FUNCTION OF LISP ARE FUNCTIONS LIKE CAR, CDR ETC, THAT WE WILL SEE IN FURTHER CHAPTERS, THAT HAVE THEIR OWN RULES OF EVALUATION.

A PREDICATE IS A BOOLEAN VARIABLE WITH ONLY THE TWO VALUES - TRUE OR FALSE.

A LITERAL IS AN ATOM, LIST OR NUMBER WHOSE EVALUATION IS PREVENTED.

A FUNCTION NOT OF THE SYSTEM LIBRARY IS DEFINED THROUGH THE LAMBDA NOTATION.

COMPARED WITH OTHER LANGUAGES, LISP HAS A COMPLETELY DIFFE - RENT GRAFIC FORM. ONLY AS AN EXAMPLE, WE WILL SHOW BELOW A PROGRAM WRITTEN IN LISP LANGUAGE.

```
(DEFINE  (LEVEL  (LAMBDA (L) (COND
                   ((ATOM (CAR L))(QUOTE FIRST))
                   ((ATOM(CAAR L))(QUOTE SECOND))
                   ((AND)(QUOTE(THIRD OR MORE))) )))  )
(APPLY LEVEL (A))
```

NOTE THAT THERE ARE NO LABELS AND NO BRANCHES, DEVICES SO FAMILIAR TO A PROGRAMMER. THE STRUCTURE IS QUITE SIMPLE DESPITE THE MANY PARENTHESES.

LISP IS NOT A NUMERICAL LANGUAGE, BUT FOR MANY APPLICATIONS AT LEAST A SIMPLE ARITHMETIC IS REQUESTED. DUE TO THE LIMITATIONS OF THE IBM 1620 WE CAN REPRESENT ONLY INTEGER NUMBERS OF UP TO FOUR DIGITS. ARITHMETIC WILL BE TREATED IN THE CHAPTER 9.

* 5. MACHINE FUNCTIONS *

IN THIS CHAPTER WE WILL DESCRIBE FOUR MACHINE FUNCTIONS THAT WILL HELP US TO UNDERSTAND SUCCEEDING CHAPTERS.
THIS FUNCTIONS ARE -

DEFINE
QUOTE
VAL
LET

* 5.1 DEFINE *

THIS FUNCTION HAS AN INDEFINITE NUMBER OF PAIRS OF ARGUMENTS AND GIVES NAMES TO ATOMS, LISTS OR FUNCTIONS. IT HAS THE FORM -

```
(DEFINE   (A1  X1)
          (A2  X2)
          ........
          ........
          (AN  XN)  )
```

WHRE A1, A2, ... , AN ARE ATOMS AND X1, X2, ... , XN ARE EXPRESSIONS THAT ARE THERE DEFINED AS THE VALUES OF CORRESPONDING ATOMS. AN EXPRESSION IS COMPOSED OF ATOMS AND NAMES OF FUNCTIONS WITH PARENTHESIS THAT DEFINE THEIR EVALUATION.
THE EVALUATION OF AN EXPRESSION CAN BE REDUCED TO THE EVALUATION OF A FUNCTION AND TO THE EVALUATION OF AN ATOM. EACH FUNCTION IS EVALUATED ACCORDING TO THEIR RULES AND THE EVALUATION OF AN ATOM CORRESPONDS TO ITS REPLACEMENT BY THE CORRESPONDING EXPRESSION GIVEN BY DEFINE. IT IS ILLEGAL TO EVALUATE AN ATOM WHOSE VALUE HAS NOT BEEN DEFINED. THE VALUE OF DEFINE IS ALLWAYS TRUE.

* EXAMPLES *

1)

```
(DEFINE   (A  B)
          (D  (L M N)) )
```

COMMENTS - IN THIS EXAMPLE WE USE THE SIMPLEST WAY OF INTRODUCING THE VALUE OF AN ATOM AND OF A LIST. FIRST WE DEFINE AN ATOM NAMED A, WHOSE VALUE IS B AND THEN THE LIST NAMED D WHOSE VALUE IS (L M N).

2)

```
(DEFINE   (X ((A B)(C D)(F G)))
          (Y (((A B) C) D))
          (Z K)
          (V (1. 2. 3. ))   )
```

3)

```
(DEFINE   (L  X)
          (X (A B))
          (A (C D F))
          (B (1 2 3)) )
```

4)

```
(DEFINE   (S 0)
          (P 1)
          (X (A S P)) )
```

5)

```
(DEFINE   (TAB (1 . 2 . 3 . 4 . 5 . 6 . 7 . 8 . 9 . ))
          (CARDS ((KING SPADES)(SEVEN CLUBS)))
          (NAMES (JOHN MARY PAUL))  )
```

* ERRORS IN THE USE OF DEFINE *
****

WHEN THE FUNCTION DEFINE IS EXECUTED CORRECTLY BY THE LISP INTERPRETER OF THE IBM 1620 THE MESSAGE

DEFINE/SETPROP EXECUTED

IS TYPED ON TYPEWRITER. BUT, IF AN ERROR OCCURS, THE MESSAGE

ERROR... INCORRECT DEFINE/SETPROP

IS TYPED. FOR INSTANCE -

```
(DEFINE (() A))
ERROR... INCORRECT DEFINE/SETPROP
```

IN THIS CASE, THIS MESSAGE IS TYPED BECAUSE THE NAME OF AN ATOM COULD NOT BE AN EMPTY LIST.
THE SAME MESSAGE IS GIVEN IN THE FOLLOWING CASES -

```
(DEFINE((A B C)(A))
(DEFINE A)
```

* 5.2 QUOTE *

THIS FUNCTION HAS ONE ARGUMENT AND HAS THE FORM

(QUOTE X)

WHERE X MAY BE ANYTHING. THE ARGUMENT X IS NOT EVALUATED AND THE VALUE OF THIS FUNCTION IS ITS ARGUMENT. THIS FUNCTION IS USED WHEN IT IS NECESSARY TO PREVENT THE FURTHER EVALUATION OF THE ARGUMENT. FOR INSTANCE -

(QUOTE A)

RESULTS IN THE LITERAL A BUT NOT THE VALUE OF A.

EXAMPLES

(QUOTE B) = B

(QUOTE 1) = 1

(QUOTE ()) = ()

(QUOTE (A B C)) = (A B C)

(QUOTE ((A B C) D)) = ((A B C) D)

(QUOTE (A (A B) C)) = (A (A B) C)

(QUOTE LISP) = LISP

(QUOTE A B) = A

(QUOTE EMPTY-LIST) = EMPTY-LIST

(QUOTE (ATOM NAME)) = (ATOM NAME)

(QUOTE ATOM-NAME) = ATOM-NAME

* 5.3 VAL *

IT HAS ONE ARGUMENT AND THE FORM -

(VAL A)

WHERE A IS EVALUATED AND MUST HAVE AN ATOM AS ITS VALUE. IT RETURNS THE VALUE OF THIS ATOM. FOR INSTANCE, IF WE EXECUTE -

(DEFINE (C D))

AND THEN

(VAL (QUOTE C))

WE WILL HAVE AS RESULT THE ATOM D BECAUSE (QUOTE C) EVALUATES TO THE ATOM C, AND C HAS THE VALUE D. THUS, WE HAVE THE FOLLOWING IDENTITY -

```
X = (VAL (QUOTE X))
```

WHEN THE ARGUMENT EVALUATES TO A LIST INSTEAD OF AN ATOM, THE RESULT WILL BE THE FIRST ELEMENT OF THIS LIST.
FOR INSTANCE, IF WE EXECUTE -

```
(DEFINE   (V A)
          (A (B C)))
```

AND THEN -

```
(VAL A)
```

AS A EVALUATES TO THE LIST (B C), THE RESULT WILL BE THE ATOM B, THAT IS, THE FIRST ELEMENT OF THE LIST (B C).

* EXAMPLES *

IF WE EXECUTE

```
(DEFINE   (SBL +)
          (A (X Y Z))
          (X (F G H))
          (+ PLUS) )
```

THEM -

```
(VAL (QUOTE SBL))  =  +
(VAL (QUOTE A))  =  (X Y Z)
(VAL A)  =  X
(VAL (QUOTE X))  =  (F G H)
(VAL X)  =  F
(VAL SBL)  =  PLUS
(VAL (QUOTE (A B)))  =  A
```

* ERRORS IN THE USE OF VAL *
****

```
(VAL Y)
ERROR... VARIABLE NAME HAS NO VALUE

(VAL +)
ERROR... UNDEFINED ARGUMENT FOR FUNCTION DEFINITION

(VAL (SBL))
ERROR... UNDEFINED ARGUMENT FOR FUNCTION DEFINITION
```

* 5.4 LET *

THIS FUNCTION HAS TWO ARGUMENTS AND HAS THE FORM -

(LET X Y)

BOTH ARGUMENTS ARE EVALUATED. THE FIRST MUST EVALUATE TO AN ATOM THAT WILL ADQUIRE AS ITS VALUE THE VALUE OF THE SECOND ARGUMENT.
THE VALUE OF LET WILL BE THE VALUE OF ITS SECOND ARGUMENT.
THIS FUNCTION ALLOWS A PERMANENT CHANGE OF VALUE FOR A GIVEN QUANTITY. THE PREVIOUS VALUE IS LOST AND THE NEW ONE IS RETAINED UNTIL ANOTHER APPLICATION OF LET.

* EXAMPLES *

LET US FIRST EXECUTE -

```
(DEFINE   (X (A B C))
          (Y (Z M L))
          (Z V)
          (V (C K)) )
```

1)

```
(LET (QUOTE X)(QUOTE Y))
```

COMMENTS - THE NEW VALUE OF X WILL BE THE ATOM Y. THE OLD VALUE THAT IS, THE LIST (A B C) IS LOST.

2)

```
(LET (QUOTE X) Y)
```

COMMENTS - THE NEW VALUE OF X WILL BE THE VALUE OF Y, THAT IS, THE LIST (Z M L).

3)

```
(LET (QUOTE Y)(VAL V))
```

COMMENTS - THE VALUE OF Y WILL BE THE VALUE OF V, THAT IS, THE ATOM C

4)

```
(LET (VAL(QUOTE Z))(QUOTE Y))
```

COMMENTS - (VAL(QUOTE Z)) EVALUATES TO THE ATOM V. IT WILL TAKE A NEW VALUE, THE ATOM Y.

* ERRORS IN THE USE OF LET *

```
(LET A B)
ERROR... VARIABLE NAME HAS NO VALUE

(LET 1 2)
```

* 6. THE BASIC FUNCTIONS IN LISP *

THE BASIC FUNCTIONS DESCRIBED IN THIS CHAPTER WILL ENABLE US TO MAKE A LOT OF INTERESTING SIMBOLIC MANIPULATIONS.
THERE ARE SEVEN BASIC FUNCTIONS -

CAR
CDR
CONS
ATOM
EQ
NULL
LIST

* 6.1 THE BASIC FUNCTION CAR *

THIS FUNCTION HAS ONE ARGUMENT. AND IS WRITTEN IN THE FORM -

(CAR X)

THIS FUNCTION EVALUATES X AND IF IT IS A LIST, THE RESULT IS THE FIRST ELEMENT OF THIS LIST, WHETHER THIS IS AN ATOM OR A SUBLIST. IF THE VALUE OF X IS NOT A LIST OR DO NOT EXIST, (CAR X) IS UNDEFINED.

* EXAMPLES *

```
(CAR (QUOTE (1 5 2)))  =  1
(CAR (QUOTE (A B (1 2) C)))  =  A
(CAR (QUOTE ((A B) G H)))  =  (A B)
(CAR (QUOTE (A)))  =  A
(CAR (QUOTE ((A))))  =  (A)
(CAR (QUOTE (((A B) C D) E F)))  =  ((A B) C D)
(CAR (QUOTE ( )))  =  UNDEFINED
(CAR (QUOTE 1))  =  UNDEFINED
(CAR (CAR (QUOTE ((1 5 2)))))  =  1
```

IN THE LAST EXAMPLE THE PAIR OF INTERNAL PARENTHESIS IS EVALUATED FIRST AND THE RESULT IS THE LIST (1 5 2). THEN, (CAR (1 5 2)) IS EVALUATED AND THE RESULT IS THE ATOM 1.

* 6.2 THE BASIC FUNCTION CDR *

THIS IS A FUNCTION OF ONE ARGUMENT. IT HAS THE FORM -

(CDR X)

THIS FUNCTION EVALUATES THE ARGUMENT AND IF THE VALUE IS A LIST, GIVES AS A RESULT THE SAME LIST WITHOUT ITS FIRST ELEMENT. WHEN THE VALUE OF THE ARGUMENT IS A LIST WITH ONE ELEMENT, THE RESULT IS AN EMPTY LIST. WHEN THE VALUE OF THE ARGUMENT IS AN EMPTY LIST, OR AN ATOM OR DO NOT EXISTS, THE CDR OF IT IS UNDEFINED.

* EXAMPLES *

```
(CDR (QUOTE (F J K L))) = (J K L)
(CDR (QUOTE ((B)))  = ()
(CDR (QUOTE A)) = UNDEFINED
(CDR (QUOTE ( ))) = UNDEFINED
(CDR (QUOTE ((A B)(C D(E F) G)))) = ((C D (E F) G))
```

IN THE LAST EXAMPLE, WE HAVE A LIST WITH TWO ELEMENTS - (A B) AND (C D (E F) G). HENCE, THE CDR WILL BE THE SAME LIST WITHOUT THE FIRST ELEMENT - ((C D (E F) G). NOTE THAT THE PARENTHESES THAT BELONGS TO THE MAIN LIST ARE MANTAINED.

* 6.3 THE COMPOSITES OF CAR AND CDR *

IN ACTUAL PROGRAMS ONE ENCOUNTERS FREQUENTLY COMBINATIONS OF CAR AND CDR, OFTEN TO A CONSIDERABLE DEPTH. IN VIEW OF THE SIZE OF THE IBM 1620 IT HAS BEEN POSSIBLE TO INCLUDE ONLY COMPOSITES UP TO DEPTH 3.

```
(CAAR X)  = (CAR (CAR X))
(CDDR X)  = (CDR (CDR X))
(CADR X)  = (CAR (CDR X))
(CDAR X)  = (CDR (CAR X))
(CAAAR X) = (CAR (CAR (CAR X)))
(CDDDR X) = (CDR (CDR (CDR X)))
(CADDR X) = (CAR (CDR (CDR X)))
(CAADR X) = (CAR (CAR (CDR X)))
(CADAR X) = (CAR (CDR (CAR X)))
(CDAAR X) = (CDR (CAR (CAR X)))
(CDDAR X) = (CDR (CDR (CAR X)))
(CDADR X) = (CDR (CAR (CDR X)))
```

ALL THESE COMPOSITES ARE, OF COURSE, ONLY DEFINED IF THEIR ARGUMENTS FULLFIL THE APPROPRIATE CONDITIONS. THUS FOR (CDAR X), THE ARGUMENT X MUST BE A LIST WHOSE FIRST ELEMENT IS ITSELF A NON-EMPTY LIST, AND SO ON.

EXAMPLES

```
(CAR (QUOTE          (A B C D E F G)))  =  A
(CADR  (QUOTE        (A B C D E F G)))  =  B
(CADDR  (QUOTE       (A B C D E F G)))  =  C
(CAR (CDDDR (QUOTE   (A B C D E F G)))) =  D
(CADR(CDDDR(QUOTE    (A B C D E F G)))) =  E
(CADDR(CDDDR(QUOTE   (A B C D E F G)))) =  F
(CAR(CDDDR(CDDDR(QUOTE (A B C D E F G))))) = G
(CDDR (QUOTE     ((((I J) K) L M) N)))  =  ( )
(CDAR (QUOTE     ((((I J) K) L M) N)))  =  (L M)
(CAAR (QUOTE     ((((I J) K) L M) N)))  =  ((I J) K)
(CAAAR (QUOTE    ((((I J) K) L M) N)))  =  (I J)
(CDDAR (QUOTE    ((((I J) K) L M) N)))  =  (M)
(CDAAR (QUOTE    ((((I J) K) L M) N)))  =  (K)
(CADAR (QUOTE    ((((I J) K) L M) N)))  =  L
(CAADR (QUOTE    (O ((P Q R) S) T)))  =  (P Q R)
(CDADR (QUOTE    (O ((P Q R) S) T)))  =  (S)
```

* ERROR MESSAGES WITH CAR AND CDR *

OFTEN WE MADE MISTAKES IN THE APPLICATIONS OF CAR AND CDR. BELOW THERE ARE SOME BAD PROPOSITIONS AND ITS CORRESPONDING MESSAGE .

1)
```
(CAR A)
ERROR... VARIABLE NAME HAS NO VALUE
```

COMMENTS - THE MACHINE GIVES THIS MESSAGE BECAUSE THE NAME A WAS NOT DEFINED. THE SAME MESSAGE WILL BE GIVEN IN THE FOLLOWING CASES -

```
(CAAR (A B))
(CDR (C DR))
(CAR P Q)
(CADR (L M N))
```

2)
```
(CAR (QUOTE ( ))
ERROR... EMPTY-LIST ARGUMENT FOR CAR

(CDR (QUOTE ( ))
ERROR... EMPTY LIST ARGUMENT FOR CDR
```

COMMENTS - THE ARGUMENT OF CAR OR CDR CANNOT BE AN EMPTY LIST.

3)
```
(CAR (QUOTE A))
ERROR... ATOMIC ARGUMENT FOR CAR

(CDR (QUOTE C))
ERROR... ATOMIC ARGUMENT FOR CDR
```

4)
```
(CAR 1)
ERROR... NUMERICAL ARGUMENT FOR CAR

(CDR 2)
ERROR... NUMERICAL ARGUMENT FOR CDR
```

REMARK - IT MUST BE NOTED THAT THE MESSAGE - VARIABLE NAME HAS NO VALUE WILL BE TYPED EVERY TIME WE USE A LISP FUNCTION WITH AN ARGUMENT NOT DEFINED.

* 6 4 THE BASIC FUNCTION CONS *

THIS IS A FUNCTION WITH TWO ARGUMENTS USED TO CREATE A LIST. IT HAS THE FORM -

(CONS X Y)

WHERE THE VALUE OF X MAY BE ANYTHING, BUT THE VALUE OF Y MUST BE A LIST, WHICH MAY BE EMPTY. THIS FUNCTION EVALUATES BOTH X AND Y AND THEN CREATES A LIST THE FIRST ELEMENT OF WHICH IS X, WHILE ALL THE OTHERS ARE THOSE OF Y. IF THE VALUE OF Y IS NOT A LIST, THE RESULT IS UNDEFINED.

THE FOLLOWING IDENTITY IS TRUE FOR ANY LIST L

(CONS (CAR L)(CDR L)) = L

* EXAMPLES *

```
(CONS (QUOTE A)(QUOTE (B C D E)))  =  (A B C D E)
(CONS (QUOTE X)(QUOTE ( )))  =  (X)
(CONS (QUOTE F)(CDR (QUOTE (A B C))))  =  (F B C)
(CONS (CAR (QUOTE (A B)))(QUOTE (B C D)))  =  (A B C D)
(CONS (QUOTE ( ))(QUOTE (X)))  =  (( ) X)
(CONS (CONS (QUOTE X)(QUOTE ()))(QUOTE ()))  =  ((X))
(CONS (QUOTE X)(QUOTE ((A) B C)))  =  (X (A) B C)
```

IN THE LAST EXAMPLE A LIST IS CONSTRUCTED WITH THE FIRST ELEMENT BEING THE FIRST ARGUMENT OF CONS, IN THIS CASE THE ATOM X AND THE REMAINING ELEMENTS THOSE OF THE SECOND ARGUMENT.

* ERRORS IN THE USE OF CONS *

```
(CONS (QUOTE A) (QUOTE B))
ERROR... ATOMIC ARGUMENT FOR CONS

(CONS A B)
ERROR... VARIABLE NAME HAS NO VALUE

(CONS (QUOTE A) 1)
ERROR... NUMERICAL ARGUMENT FOR CONS
```

* 6.5 THE PREDICATE ATOM *

THE THREE FOLLOWING FUNCTIONS ARE CALLED PREDICATES FOR, AS WE HAVE SEEN, THEY TAKE ONLY THE VALUES TRUE OR FALSE. SO THE RESULT OF THESE FUNCTIONS IS NOT A LIST OR AN ATOM BUT A LOGICAL RESULT EQUAL TO ONE OF TWO CONDITIONS, TRUE OR FALSE

THE PREDICATE ATOM HAS ONE ARGUMENT AND CAN BE WRITTEN -

(ATOM X)

IT EVALUATES THE ARGUMENT X, AND IF IT IS AN ATOM, THE RESULT IS TRUE - OTHERWISE, IF IT IS A LIST, THE ANSWER IS FALSE.

* EXAMPLES *

```
(ATOM (QUOTE A))  =  TRUE
(ATOM (QUOTE (A)))  =  FALSE
(ATOM (QUOTE (A B C D E)))  =  FALSE
(ATOM (CAR (QUOTE  ((K) P G))))  =  FALSE
(ATOM (CADR(QUOTE  ((K) P G))))  =  TRUE
(ATOM (CDR (QUOTE  ((K) P G))))  =  FALSE
(ATOM (CADDR (QUOTE ((K) P G))))  =  TRUE
```

* 6.6 THE PREDICATE EQ *

THIS FUNCTION HAS TWO ARGUMENTS AND CAN BE WRITTEN -

(EQ X Y)

IT EVALUATES BOTH X AND Y. IF BOTH VALUES ARE THE SAME ATOM THE RESULT IS TRUE. IF THE VALUES ARE DIFFERENT ATOMS, OR IF ONE OF THEM IS A LIST, THE ANSWER IS FALSE. IF BOTH VALUES ARE LISTS, THE FUNCTION IS UNDEFINED.

* EXAMPLES *

```
(EQ (QUOTE A)(QUOTE A))  =  TRUE

(EQ (QUOTE A)(QUOTE (A)))  =  FALSE

(EQ (QUOTE V)(CAR (QUOTE (A B C))))  =  FALSE

(EQ (QUOTE (A))(QUOTE (A)))  =  UNDEFINED

(EQ (QUOTE C)(CDR(QUOTE(C C))))  =  FALSE

(EQ (CADR(QUOTE(S G H)))(CAR(QUOTE(G S H))))  =  TRUE
```

* 6.7 THE PREDICATE NULL *

THIS PREDICATE HAS ONE ARGUMENT AND HAS THE FORM -

(NULL L)

IT WILL BE TRUE ONLY IF L EVALUATES TO AN EMPTY LIST. NOTE THAT THE LIST

(( ))

IS NOT AN EMPTY LIST. IT IS A LIST THAT CONTAINS A SUBLIST WHOSE VALUE IS AN EMPTY LIST.

* EXAMPLES *

(NULL (QUOTE ( ))) = TRUE

(NULL (QUOTE A)) = FALSE

(NULL (CAR (QUOTE (() B C)))) = TRUE

(NULL (CDR (QUOTE (Z)))) = TRUE

(NULL (QUOTE (A B C))) = FALSE

* 6.8 THE FUNCTION LIST *

THIS FUNCTION HAS AN INDEFINITE NUMBER OF ARGUMENTS. IT HAS THE FORM -

(LIST A B C .....)

IT EVALUATES ALL THE ARGUMENTS AND THEN FORMS A LIST OF THE RESULTS, IN THE ORDER OF THE ARGUMENTS. IF THERE ARE NO ARGUMENTS, THAT IS, IF WE WRITE -

(LIST)

THE RESULT WILL BE AN EMPTY LIST.

THE FUNCTIONS LIST AND CONS ENABLE US TO FORM ANY NEW LIST. NOTE THE DIFFERENCE BETWEEN THESE TWO FUNCTIONS. FOR INSTANCE, IF WE EXECUTE -

```
(DEFINE   (X  P)
          (L (A B C) )
```

AND THEN -

(LIST X L)

THE RESULT WILL BE (P (A B C))

BUT, INSTEAD OF LIST WE USE -

(CONS X L)

THE RESULT WILL BE (P A B C)

* EXAMPLES *

```
(LIST (QUOTE A)(QUOTE B))  = (A B)

(LIST (QUOTE (A B C))(QUOTE (C D)))  =  ((A B C)(C D))

(LIST (QUOTE ( ))(QUOTE( )))  =  (( )( ))

(LIST (QUOTE((A B) C))(QUOTE S))  =  (((A B) C) S)

(LIST (QUOTE(V W))(LIST))  =  ((V W)( ))

(LIST (QUOTE A)(CONS(QUOTE(L))(QUOTE(B C))) = (A ((L) B C))

(LIST(CONS(QUOTE C)(QUOTE(C D)))(CAR(QUOTE(A B))))  =
                                        ((C C D) A)

(LIST(CONS(LIST(LIST))(LIST)) = ((( )) ( ))
```

* 7. CONDITIONAL EXPRESSIONS *

FROM THE BASIC FUNCTIONS OF THE CHAPTER 6 IT IS POSSIBLE TO BUILD UP MORE COMPLEX ONES, SINCE ALL OF THEM EVALUATE THEIR ARGUMENTS. IF AN ARGUMENT IS IN TURN A FUNCTION WRITTEN, AS HAS BEEN SEEN, AS A LIST WHICH MAY BE HANDLED BY LISP - IT IS, OF COURSE, EVALUATED BY CARRYING OUT THE FUNCTION IN QUESTION. HOWEVER, MORE SYMBOLS ARE NEEDED IN ORDER TO EXPRESS THE LOGICAL RELATIONS OF THE CONDITIONS FOR A RECURSION. THIS IS THE MAIN(THOUGH NOT THE ONLY) PURPOSE OF THE SYMBOLS COND AND IF AND, THEY DIFFER BASICALLY FROM THE ELEMENTARY FUNCTION BECAUSE THEY DO NOT NECESSARILY EVALUATE ALL THEIR ARGUMENTS.

* 7.1 THE CONDITIONAL EXPRESSION IF *

THE CONDITIONAL EXPRESSION IF HAS THREE ARGUMENTS AND THE FORM -

```
(IF P A B)
```

WHERE P IS A PREDICATE, A AND B EXPRESSIONS. THE PREDICATE IS EVALUATED AND IF IT HAS THE VALUE TRUE THEN THE EXPRESSION A IS EVALUATED AND ITS VALUE IS THAT OF THE IF-EXPRESSION. OTHERWISE, WHEN THE PREDICATE HAS THE VALUE FALSE, THE EXPRESSION B IS EVALUATED AND TAKEN AS THE RESULT OF THE IF-EXPRESSION. IN EITHER CASE THE OTHER EXPRESSION IS NOT EVALUATED AND MAY CONTAIN ELEMENTS WHICH UNDER THESE CONDITIONS ARE UNDEFINED, E.G. THE CDR OF AN EMPTY LIST.

* EXAMPLES *

1)

```
(IF(ATOM(QUOTE G)) (CAR(QUOTE(A B C))) (CAR(QUOTE(D E F))) )
```

COMMENTS - AS THE PREDICATE (ATOM (QUOTE P)) HAS THE VALUE TRUE, THEN THE VALUE OF THE IF - EXPRESSION WILL BE THE SAME AS THAT OF -

```
(CAR (QUOTE (A B C)))
```

THAT IS, THE ATOM A.

2)

```
(IF (ATOM(QUOTE(R T))) (CDR(QUOTE(Q W))) (CADR(QUOTE(S J K))) )
```

COMMENTS - IN THIS CASE, THE PREDICATE IS FALSE BECAUSE THE ARGUMENT OF THE PREDICATE ATOM IS A LIST, NOT AN ATOM. HENCE, THE VALUE OF THE IF EXPRESSION WILL BE THE SAME AS THAT OF THE THE VALUE OF (CADR (QUOTE (S J K))), THAT IS, THE ATOM J.

3)

```
(IF (EQ (CAR(QUOTE(A)))(CAR(QUOTE(B)))) (CDR(QUOTE(M)))(CAR(QUOTE(C))))
```

COMMENTS - AS THE PREDICATE IS FALSE BECAUSE THE VALUE OF - (CAR(QUOTE(A))) DIFFER FROM THE VALUE OF (CAR(QUOTE(B))), THEN THE RESULT OF THIS CONDITIONAL EXPRESSION WILL BE THE VALUE OF (CAR(QUOTE(C))), THAT IS, THE ATOM C.

4)

```
(IF (NULL(QUOTE(( ))))
    (LIST(CONS(QUOTE A)(QUOTE( )))(LIST))
    (LIST(LIST(QUOTE A)(QUOTE B)))(CONS(QUOTE( ))(QUOTE( )))
```

COMMENTS - AS THE PREDICATE IS FALSE, BECAUSE THE LIST (( )) IS A LIST WITH ONE ELEMENT, THE LAST EXPRESSION IS EVALUATED GIVING AS RESULT - ((A B))(( ))

5)

SUPPOSE THE FOLLOWING LISTS WERE DEFINED -

```
(DEFINE (X (A B C))
        (L (B B C)) )
```

WE WOULD LIKE TO PRINT THE WORD EQUAL IF THE FIRST ELEMENT OF X IS EQUAL TO THE SECOND OF L AND THE FIRST OF L IS EQUAL TO THE SECOND OF X. OTHERWISE WE WILL PRINT THE WORD DIFFERENT.

```
(IF      (EQ(CADR L)(CAR X))
         (IF      (EQ(CAR L)(CADR X))
                  (QUOTE EQUAL)
                  (QUOTE DIFFERENT))
         (QUOTE DIFFERENT))
```

HERE WE USE A CONDITIONAL IF INSIDE ANOTHER ONE. IF THE PREDICATE OF THE FIRST EVALUATES TO TRUE, THE ANSWER IS ANOTHER CONDITIONAL EXPRESSION. THUS, WE MAY USE AS MANY IF AS WE DESIRE.

* 7.2 THE CONDITIONAL EXPRESSION COND *

THIS CONDITIONAL EXPRESSION MAY HAVE ANY NUMBER OF PAIRS OF ARGUMENTS AND HAS THE FORM -

```
(COND (P1 E1)(P2 E2) .........(PN EN))
```

HERE ALL P(I) MUST BE PREDICATES, THE E(I) CAN BE ANY KIND OF EXPRESSIONS. P(1) IS FIRST EVALUATED, IF IT IS TRUE E1 IS EVALUATED AND ITS VALUE GIVEN AS THE VALUE OF COND. IF P(1) IS FALSE, E(1) IS NOT EVALUATED, BUT P(2) IS EXAMINED, IF TRUE THEN THE VALUE OF E(2) IS THE VALUE OF COND, OTHERWISE THE NEXT PREDICATE IS EVALUATED UNTIL A TRUE PREDICATE IS REACHED, WHOSE ASSOCIATED EXPRESSION THEN GIVES THE VALUE OF COND. NEITHER THE COMPANION EXPRESSION OF FALSE PREDICATES NOR THE REMAINING PAIRS AFTER A TRUE PREDICATE ARE EVALUATED. IF THE LAST PREDICATE IS REACHED AND IS FALSE, COND IS UNDEFINED. TO AVOID THIS, THE LAST PREDICATE IS FREQUENLY (AND) WHICH FOR REASONS WE WILL SEE LATTER IS ALWAYS TRUE.

* EXAMPLES *
1)

```
(COND   ((ATOM (QUOTE (A B)))                          (CAR (QUOTE (F K))))
        ((ATOM (QUOTE (F K)))                          (CAR (QUOTE (A B))))
        ((EQ (CAR(QUOTE(A B)))(CAR(QUOTE(F K))))       (CADR (QUOTE(F K))))
        ((ATOM (QUOTE P))                              (CADR(QUOTE(A B))))
        ((NULL (QUOTE (A)))                            (CAR(QUOTE(A B)))) )
```

COMMENTS - NOTE THE STRUCTURE OF THE COND. IT HAS FIVE PAIRS IN WHICH WE HAVE THE PREDICATE AND THE CORRESPONDING EXPRESSION. AS THE ARGUMENTS OF THE PREDICATE ATOM IN THE FIRST TWO PROPOSITIONS ARE LISTS, THE RESULT OF THESE PREDICATES IS FALSE. SO, THE RESPECTIVELY EXPRESSIONS ARE NOT EVALUATED. THE THIRD PAIR VERIFY IF THE VALUE OF CAR OF TWO DIFFERENT LISTS ARE EQUAL. THE EQUALITY IS NOT TRUE AND THE CORRESPONDING EXPRESSION IS NOT EVALUATED. THEN, IN THE 4TH PAIR, AS P IS AN ATOM, THE PREDICATE IS TRUE AND THE EXPRESSION (CADR (QUOTE (A B))) IS EVALUATED GIVING AS THE RESULT, THE ATOM B. THIS RESULT WILL BE ALSO THE RESULT OF THE FUNCTION COND AND THE REMAINING PAIR IS NOT EVALUATED.

2)

SUPPOSE WE EXECUTE THE FOLLOWING DEFINITION -

```
(DEFINE   (A (X Y Z K)) )
```

AFTER SOME MANIPULATIONS WITH THE LIST A, WE WISH TO KNOW IF THIS LIST HAS ONE, TWO, THREE OR MORE ELEMENTS. FOR THIS, WE CAN CONSTRUCT A PIECE OF PROGRAM THINKING GENERICALLY ON A LIST NAMED A.

LETS WRITE IN ENGLISH THE STEPS TO GIVE THE SOLUTION TO OUR PROBLEM -

1) IF THE CDR OF THE LIST A IS EMPTY, THE LIST HAS ONE ELEMENT.

2) IF THE CDDR OF THE LIST A IS EMPTY, THE LIST HAS TWO ELEMENTS.

3) OTHERWISE, THE LIST HAS THREE OR MORE ELEMENTS.

FOR WRITE THIS IN LISP LANGUAGE, WE CAN USE THE CONDITIONAL FUNCTION COND. WE MAY WRITE -

```
(COND
 ((NULL (CDR A))   (QUOTE(ONE ELEMENT)))
 ((NULL(CDDR A))   (QUOTE(TWO ELEMENTS)))
 ((AND)            (QUOTE(THREE OR MORE ELEMENTS))) )
```

NOTE THAT THE FUNCTION (AND) EVALUATES TO TRUE AND MAKES THE CONDITIONAL FUNCTION COND ALSO TRUE.

3)

SUPPOSE WE DEFINE A LIST AND AN ATOM -

```
(DEFINE   (L (B C))
          (A B))
```

AFTER SOME MANIPULATIONS WITH THE LIST L, WE WISH TO KNOW IF THE ATOM A IS THE FIRST, THE SECOND OR IS NOT AN ELEMENT OF THE LIST L.

THE PIECE OF PROGRAM COULD BE WRITE -

```
(COND
((EQ (CAR L) A) (QUOTE FIRST-ELEMENT))
((EQ (CADR L) A) (QUOTE SECOND-ELEMENT))
((AND)(QUOTE NO-ELEMENT)) )
```

4)

SUPPOSE THE FOLLOWING LISTS WERE DEFINED -

```
(DEFINE (X (A B C ))
        (L (B B C)))
```

AFTER SOME MANIPULATIONS WE WISH TO KNOW IF THE FIRST ELEMENT OF X IS EQUAL TO SECOND OF L. PRINT THE WORD EQUAL IF THEY REFER TO THE SAME ATOM AND THE WORLD DIFF OTHERWISE.

```
(COND
((EQ(CAR X)(CADR L))(QUOTE EQUAL))
((AND)(QUOTE DIFF)))
```

AS WE HAVE ONLY ONE CONDITION WE MAY USE IF INSTEAD OF COND.

```
(IF       (EQ (CAR X) (CADR L))
          (QUOTE EQUAL)
          (QUOTE DIFF))
```

* ERRORS IN THE USE OF CONDITIONALS *

****

```
(COND ((ATOM (QUOTE (A B)))(QUOTE C)))
ERROR... NO TRUE PREDICATE IN COND

(COND ((1 (QUOTE B))(QUOTE C)))
ERROR... NUMBER AS FUNCTION DEFINITION
```

* 8. LOGICAL FUNCTIONS *

LOGICAL FUNCTIONS DIFFER FROM THE PREDICATES PREVIOUSLY DESCRIBED IN THAT NOT ONLY THEIR VALUES BUT ALSO THEIR ARGUMENTS TAKE ONLY THE TWO POSSIBLE VALUES TRUE OR FALSE. THE ARE ALSO CALLED BOOLEAN FUNCTIONS. IN LISP, ONE OF THEIR PRINCIPAL USES IS TO COMBINE PREDICATES OF VARIOUS KINDS INTO MORE POWERFUL ONES FOR USE WITH IF OR COND.

THE THREE LOGICAL FUNCTIONS ARE -

AND
OR
NOT

* 8.1 THE LOGICAL FUNCTION AND *

THIS FUNCTION HAS ANY NUMBER OF ARGUMENTS AND HAS THE FORM -

(AND P1 P2 ..... PN)

THE ARGUMENTS ARE EVALUATED IN THE ORDER GIVEN. IF ANY OF THE ARGUMENTS IS FALSE, THEN THE VALUE OF THE FUNCTION AND WILL BE FALSE AND THE REMAINING ARGUMENTS ARE NOT EVALUATED. OTHERWISE, IF NO FALSE ARGUMENT IS FOUND, THE VALUE OF THE FUNCTION WILL BE THE VALUE OF THE LAST ARGUMENT.

THE FUNCTION AND WITHOUT ARGUMENTS HAS THE VALUE TRUE. IT HAS THE FORM -

(AND)

THIS IS OFTEN USED AS THE LAST PREDICATE OF THE FUNCTION COND FOR, AS WE HAVE SEEN, ITS LAST PREDICATE MUST BE TRUE.

* EXAMPLES *

1)

```
(AND (ATOM(QUOTE A)) (NULL(QUOTE( ))))
```

COMMENTS - BOTH ARGUMENTS ARE PREDICATES AND THEY EVALUATES TO THE VALUE TRUE, THEN THE FUNCTION AND IS TRUE.

2)

(AND (NULL (QUOTE(L)))(ATOM (QUOTE(L))))

COMMENTS - THE FUNCTIONS NULL AND ATOM ARE PREDICATES. AS THE FIRST ONE EVALUATES TO A VALUE TRUE AND THE SECOND TO A VALUE FALSE THEN, THE FUNCTION AND WILL BE FALSE.

3)

(AND (CAR(QUOTE(A B))) (ATOM (QUOTE A))))

COMMENTS - THE FIRST ARGUMENTS EVALUATES TO THE VALUE A (NOT A PREDICATE), THE SECOND, IS A PREDICATE THAT EVALUATES TO A VALUE TRUE. THE VALUE OF THE FUNCTION IS TRUE.

4)

(AND (ATOM(QUOTE A))(CAR(QUOTE (A B))))

COMMENTS - AS ONE OF THE ARGUMENTS IS NOT A PREDICATE, THE VALUE OF THE FUNCTION WILL BE THE VALUE OF THE LAST ARGUMENT OF AND. THE RESULT IS THE ATOM A.

5)

(AND (CAR(QUOTE(A)))(CAR(QUOTE(B))))

COMMENTS - THE ARGUMENTS ARE NOT PREDICATES, SO THAT THEY ARE EVALUATED AND THE RESULT OF THE FUNCTION WILL BE THE RESULT OF THE EVALUATION OF THE LAST ARGUMENT, THAT IS, THE ATOM B.

6)

(AND(LIST(QUOTE (B C)))(LET(QUOTE A)(CONS(QUOTE A)(QUOTE(B C)))))

COMMENTS - THE TWO ARGUMENTS ARE NOT PREDICATES. THEY ARE EVALUATED AND THE RESULT OF THE FUNCTION WILL BE (A B C), THAT IS THE RESULT OF THE EVALUATION OF THE LAST ARGUMENT.

7)

(AND (ATOM (QUOTE (A)))(CAR(QUOTE(A B))))

COMMENTS - AS THE FIRST ARGUMENT IS A PREDICATE THAT EVALUATES TO A VALUE FALSE, THE RESULT WILL BE FALSE.

* 8.2 THE LOGICAL FUNCTION OR *

THIS FUNCTION HAS ANY NUMBER OF ARGUMENTS AND HAS THE FORM -

(OR P1 P2 ..... PN)

THE ARGUMENTS ARE EVALUATED IN THE ORDER GIVEN. IF ANY OF THE ARGUMENTS IS TRUE, THEN THE VALUE OF THE FUNCTION WILL BE TRUE. OTHERWISE, IF NO TRUE ARGUMENT IS FOUND, THE VALUE OF THE FUNCTION WILL BE THE VALUE OF THE LAST ARGUMENT. THE FUNCTION OR WITH NO ARGUMENTS EVALUATES TO A VALUE FALSE.

* EXAMPLES *

1)

```
(OR (ATOM(QUOTE(A B))) (NULL(QUOTE(A B))))
```

COMMENTS - THE RESULT OF THE FUNCTION OR WILL BE FALSE BECAUSE BOTH ARGUMENTS EVALUATES TO THE VALUE FALSE.

2)

```
(OR (ATOM(QUOTE A))(CAR(QUOTE(A B))))
```

COMMENTS - THE RESULT OF THE FUNCTION OR WILL BE TRUE, BECAUSE THE FIRST ARGUMENT IS A PREDICATE THAT EVALUATES TO A VALUE TRUE. THE REST OF THE ARGUMENTS ARE NOT EVALUATED.

3)

```
(OR (CAR(QUOTE(A B)))(CAR(QUOTE(B C))))
```

COMMENTS - BOTH ARGUMENTS ARE NOT PREDICATES. THEN, THE VALUE OF THE FUNCTION OR WILL BE THE VALUE OF THE LAST ARGUMENT, THAT IS, THE ATOM B.

4)

```
(OR (CAR(QUOTE(A)))(ATOM(QUOTE(B))))
```

COMMENTS - THE FIRST ARGUMENT IS EVALUATED AND HAS THE VALUE A. THE SECOND ARGUMENT IS A PREDICATE THAT EVALUATES TO A VALUE FALSE. AS THIS ARGUMENT IS THE LAST ARGUMENT OF THE FUNCTION OR, ITS VALUE WILL BE THE VALUE OF THE FUNCTION, THAT IS, FALSE.

5)

```
(OR (ATOM (QUOTE A)) (CAR (QUOTE A)))
```

COMMENTS - AS THE FIRST ARGUMENT EVALUATES TO A VALUE TRUE THE RESULT OF THE FUNCTION WILL BE TRUE EVEN IF THE SECOND ARGUMENT CANNOT BE EVALUATED.

* 8.3 THE LOGICAL FUNCTION NOT *

****

THIS FUNCTION HAS ONE ARGUMENT AND HAS THE FORM -

(NOT X)

IT EVALUATES THE ARGUMENT AND IF A PREDICATE THAT EVALUATES TO A VALUE TRUE, THE FUNCTION NOT WILL HAVE THE VALUE FALSE. OTHERWISE, IF THE PREDICATE HAS A VALUE FALSE THE FUNCTION NOT WILL HAVE THE VALUE TRUE. IF THE ARGUMENT IS NOT A PREDICATE, THE ARGUMENT IS EVALUATED, AND THE RESULT WILL BE THE VALUE OF THE FUNCTION NOT.

* EXAMPLES *

1)

```
(NOT (ATOM (QUOTE (A B C))))  =  TRUE

(NOT (ATOM (QUOTE A)))        =  FALSE

(NOT (NULL (QUOTE (A))))      =  TRUE

(NOT (NULL (QUOTE ( ))))      =  FALSE

(NOT (EQ (QUOTE A)(QUOTE A))) =  FALSE

(NOT (QUOTE A) )              =  A
```

2)

SUPPOSE WE EXECUTE -

```
(DEFINE   (A (L (M N) O))
          (P O))
```

AND THEM -

A)

```
(NOT  (ATOM(CADDR A)))
```

COMMENTS - AS THE ARGUMENT OF THE LOGICAL FUNCTION NOT, EVALUATES TO TRUE, THE RESULT OF THIS FUNCTION WILL BE FALSE.

B)

```
(AND (EQ(CAR A) P)(EQ (CADDR A) P))
```

COMMENTS - THE ARGUMENTS OF THE LOGICAL FUNCTION AND ARE EVALUATED.AS THE FIRST ARGUMENT EVALUATES TO FALSE, THE ENDING VALUE OF THIS LOGICAL FUNCTION WILL BE ALSO FALSE.

C)

```
(OR (ATOM(CADR A))(NOT(ATOM(CADDR A)))(ATOM P))
```

COMMENTS - THE LOGICAL FUNCTION OR WILL BE FALSE IF ALL ITS ARGUMENTS EVALUATES TO FALSE BUT IF AT LEAST ONE OF THEM EVALUATES TO TRUE, THEN THE FUNCTION WILL BE TRUE. IN THIS EXAMPLE, THE TWO FIRST ARGUMENTS ARE FALSE BUT THE LAST ONE IS TRUE, SO, THE FUNCTION IS TRUE.

D)

```
(AND (AND)(OR))
```

COMMENTS - THE LOGICAL FUNCTION AND, IS TRUE IF ALL ITS ARGUMENTS EVALUATES TO TRUE. THE FIRST ARGUMENT, (AND), IS TRUE BUT, THE SECOND (OR) EVALUATES TO FALSE. THUS, THE RESULT OF THE PROPOSITION IS FALSE.

* 9. ARITHMETIC FUNCTIONS *

DUE TO THE LIMITATIONS OF THE IBM 1620, THE ARITHMETIC FUNCTIONS OF THE LISP INTERPRETER FOR THIS COMPUTER HAS ALSO ITS LIMITATIONS.

WE CAN HANDLE ONLY INTEGERS. A NUMBER IS DEFINED AS AN INTEGER OF UP TO FOUR DIGITS, WITH AN OPTIONAL SIGN IN FRONT. A NUMBER WILL BE TREATED AS AN ATOM WHOSE VALUE IS ITS NAME. NO OTHER VALUE MAY THEREFORE BE ASSIGNED TO IT, WHICH MEANS THAT A NUMBER CANNOT BE THE NAME OF A FUNCTION OR A VARIABLE. AS A RESULT, WHEN A NUMBER OCCURS AS A CONSTANT WITHIN A FUNCTION DEFINITION, IT IS NOT NECESSARY TO QUOTE IT. AN INTEGER OF MORE THAN FOUR DIGITS IS NOT RECOGNIZED AS NUMBER, IS TREATED AS AN ORDINARY ATOM AND NO FIXED VALUE IS THEREFORE ASSOCIATED WITH IT.

THERE ARE SEVEN FUNCTIONS AVAILABLE -

ADD
SUB
MULT
DIV
SG
NUM
RANDOM

THE RULES OF THE ARITHMETIC IN LISP ARE THE SAME OF THE ALGEBRAIC RULES.

* 9.1 ADD *

THIS FUNCTION HAS TWO ARGUMENTS AND HAS THE FORM -

(ADD N M)

IT EVALUATES BOTH ARGUMENTS AND IF THEY ARE INTEGER NUMBERS, THE RESULT WILL BE THE ALGEBRAIC SUM N + M. IF THE ARGUMENTS ARE NOT NUMBERS, AN ERROR MESSAGE IS GIVEN.

* EXAMPLES *

1) IF WE EXECUTE -

```
(DEFINE   (A  2)
          (B  5) )
```

AND THEN -

```
(ADD A  B)
```

THE RESULT WILL BE THE ALGEBRAIC SUM 2 + 5, THAT IS, 7.

2)

```
(ADD 1  0)    =   1
(ADD +6 +18)  =   24
(ADD -7 8)    =   1
(ADD -9 -8)   =   -17
(ADD 008 -04) =   4
```

* 9.2 SUB *

THIS FUNCTION HAS TWO ARGUMENTS. IT HAS THE FORM -

(SUB N M)

IT EVALUATES BOTH ARGUMENTS AND IF BOTH ARE NUMBERS, THE RESULT WILL BE THE ALGEBRAIC DIFFERENCE N - M.
IF THE ARGUMENTS DO NOT EVALUATES TO NUMBERS AN ERROR MESSAGE IS GIVEN.

* EXAMPLES *

1) IF WE EXECUTE -

```
(DEFINE   (C 10)
          (D  5) )
```

AND THEN -

```
(SUB C D)
```

THE RESULT WILL BE THE ALGEBRAIC DIFFERENCE 10 - 5,THAT IS, 5.

2)

```
(SUB -5 4)    =   -9
(SUB 10 -5)   =   15
(SUB -4 -8)   =   4
(SUB 9 9)     =   0
```

* ERRORS IN THE APPLICATION OF ADD/SUBTRACT *

WHEN WE TRY TO ADD OR SUBTRACT TWO INTEGERS AND THE RESULT IS AN INTEGER WITH MORE THAN FOUR DIGITS, THE ERROR MESSAGE --

ERROR... ADD/SUBTRACT OVERFLOW

IS GIVEN. FOR INSTANCE, WE WILL HAVE THIS MESSAGE WE EXECUTE -

```
(ADD 1235 9999)
(SUB 1234 -9999)
```

BY OTHER HAND IF WE USE AN ARITHMETIC FUNCTIONS WITH AN ARGUMENT WITH MORE THAN FOUR DIGITS WE WILL HAVE ANOTHER ERROR MESSAGE -

ERROR... VARIABLE NAME HAS NO VALUE

THIS WILL BE TYPED IF WE TRY -

```
(SUB 19999 1)
(ADD 1 42568)
```

* 9.3 MULT *

THIS FUNCTION HAS TWO ARGUMENTS AND HAS THE FORM -

(MULT N M)

IT EVALUATES BOTH ARGUMENTS AND IF THEY ARE NUMBERS, GIVES THE PRODUCT NM. IF THE ARGUMENTS DO NOT EVALUATES TO NUMBERS, AN ERROR MESSAGE IS GIVEN.

* EXAMPLES *

1) IF WE EXECUTE -

```
(DEFINE (E 10)
        (F  6) )
```

AND THEN -

```
(MULT E F)
```

THE RESULT WILL BE 60.

2)

```
(MULT 5 -10) =   -50
(MULT -2 -5) =   10
(MULT 7 2)   =   14
(MULT -4 5)  =   -20
(MULT 2 8)   =   16
```

IF A PRODUCT HAS MORE THAN FOUR DIGITS AN ERROR MESSAGE IS GIVEN. FOR INSTANCE,

```
(MULT 145 789)
ERROR... MULTIPLY OVERFLOW

(MULT 9999 2)
ERROR... MULTIPLY OVERFLOW

(MULT 1235 986)
ERROR... MULTIPLY OVERFLOW

(MULT 23568 1)
ERROR... VARIABLE NAME HAS NO VALUE
```

* 9.4 DIV *

THIS FUNCTION HAS TWO ARGUMENTS AND HAS THE FORM -

(DIV N M)

IT EVALUATES BOTH ARGUMENTS AND, IF THEY ARE NUMBERS, GIVES THE INTEGER QUOTIENT BETWEEN N AND M. IF THE ARGUMENTS DO NOT EVALUATE TO NUMBERS OR IF A DIVISION BY ZERO OCCURS, AN ERROR MESSAGE IS GIVEN.

* EXAMPLES *

```
(DIV 8 2)      =   4
(DIV 100 2)    =   50
(DIV 7 2)      =   3
(DIV 45 8)     =   5
(DIV -9 -8)    =   1
(DIV -9 7)     =   -1
(DIV 40 1000)  = 0
(DIV 1154 1000) = 1
(DIV 2486 1000) = 2
```

IF A DIVISION BY ZERO IS ATTEMPTED, AN ERROR MESSAGE IS TYPED FOR INSTANCE -

```
(DIV 82 0)
ERROR... DIVISION BY ZERO

(DIV 45 12345)
ERROR... VARIABLE NAME HAS NO VALUE
```

* 9.5 RANDOM *

THIS FUNCTION HAS NO ARGUMENTS AND HAS THE FORM -

(RANDOM)

IT GIVES AN INTEGER RANDOM NUMBER BETWEEN 0 AND 9999 WITH AN UNIFORM DISTRIBU - TION.

* EXAMPLES *

1)

```
(LET (QUOTE A) (DIV (RANDOM) 1000))
```

COMMENTS - THIS PROPOSITION ASSIGNS TO THE ATOM A, AN INTEGER NUMBER FROM ZERO TO NINE. THE FUNCTION RANDOM GIVES AN INTEGER BETWEEN ZERO AND 9999. AS THE DIVISION IS INTEGER THE QUOTIENT WILL BE THE DIGIT OF THE THOUSANDS OF THE RANDOM NUMBER.

* 9.6 SG *

THIS FUNCTION IS A PREDICATE WITH TWO ARGUMENTS. IT HAS THE FORM -

```
(SG N M)
```

IT WILL HAVE THE VALUE TRUE IF THE VALUE OF N IS GREATER THAN THE VALUE OF M, FALSE OTHERWISE. IF THE ARGUMENTS DO NOT EVALUATE TO NUMBERS, AN ERROR MESSAGE IS GIVEN.

* EXAMPLES *

```
(SG  2  1)     =   TRUE
(SG 1  5)      =   FALSE
(SG 123 489)   =   FALSE
(SG 58 24)     =   TRUE
```

* 9.7 NUM *

THIS FUNCTION IS A PREDICATE WITH ONE ARGUMENT. IT HAS THE FORM -

```
(NUM  X)
```

IT EVALUATES THE ARGUMENT AND IF IT IS A NUMBER, THE VALUE OF THE PREDICATE WILL BE TRUE. OTHERWISE, IT WILL BE FALSE.

*EXAMPLES *

1)

```
(NUM  4)   =  TRUE
(NUM -9)   =  TRUE
(NUM (QUOTE  A))   =  FALSE
(NUM (QUOTE A1))   =  FALSE
```

2)

SUPPOSE WE EXECUTE -

```
(DEFINE   (N (1 2 3))
          (L (A B C)) )
```

A)

```
(AND (LET (CAR L) 6)
     (ADD (VAL (CAR L)) (CADR N)))
```

COMMENTS - THE RESULT OF THE FUNCTION AND WILL BE 8.

B)

```
(AND (LET (CAR L) 1)
     (LET (CADR L) 2)
     (LET (CADDR L) 3)
     (ADD (ADD (VAL(CAR L))(CADDR N))(VAL (CADR L))))
```

COMMENTS - THE RESULT WILL BE 4

10. LAMBDA NOTATION

WE ARE NOW CONCERNED WITH THE DEFINITION OF FUNCTIONS BY THE PROGRAMMER. FOR INSTANCE, WE WANT DEFINE A FUNCTION TO DETERMINE IF A GIVEN LIST HAS ONE, TWO, THREE OR MORE ELEMENTS. BY MEANS OF THE FUNCTIONS GIVEN IN THE PREVIOUS CHAPTER WE ARE ABLE TO DO IT - THE FUNCTION DESCRIPTION IS -

```
(COND     ((NULL(CDR L))(QUOTE(ONE ELEMENT)))
          ((NULL(CDDR L))(QUOTE (TWO ELEMENTS)))
          ((AND)(QUOTE(THREE OR MORE ELEMENTS))))
```

HOWEVER, FOR A COMPLETE DEFINITION, WE MUST ALSO SPECIFY -

1) HOW TO ASSOCIATE A NAME WITH A FUNCTION DESCRIPTION.

2) HOW TO INDICATE PRECISELY WHAT ARE VARIABLES AND THEREFORE TO GIVE THEM VALUES.

THIS WAS ANSWERED BY THE LAMBDA THEORY INTRODUCED BY CHURCH. HE SOLVED THESE PROBLEMS BY MEANS OF A LIST OF THREE ELEMENTS -

```
(LAMBDA (A B C) (FORM))
```

THE FIRST ELEMENT IS THE WORD LAMBDA AND IT INDICATES TO THE MACHINE THAT THE FOLLOWING ELEMENT IS THE LIST OF VARIABLES FOR THE FUNCTION. THE THIRD ELEMENT, (FORM), IS SOME EXPRESSION THAT DEFINES THE FUNCTION. IN THE SAMPLE GIVEN ABOVE THERE ARE THREE VARIABLES - A, B AND C.

THE LIST OF VARIABLES CAN BE AN EMPTY LIST BUT MUST EXIST AS OTHERWISE, THE MACHINE TAKES THE FOLLOWING EXPRESSION AS THE LIST OF VARIABLES.

WE CAN APPLY A FUNCTION WITH A LIST OF VARIABLES TO BE ASSOCIATED WITH THE ARGUMENTS. IT HAS THE FORM -

```
((LAMBDA (V1 V2 ..... VN) (FORM))  A1 A2 ..... AN)
```

HERE, THE VALUES OF A1,A2 ..... AN GIVE THE VALUES TO BE ASSOCIATED WITH THE VARIABLES V1, V2 .... VN, BY PAIRING THEM OFF, A1 WITH V1, A2 WITH V2, AND SO ON.

THIS NOTATION IS THE SAME AS THE ONE USED IN THE PREVIOUS CHAPTER - THE FIRST ELEMENT OF THE LIST IS THE FUNCTION TO BE CALCULATED -

```
(LAMBDA (V1 V2 ..... VN) (FORM))
```

AND THE SECOND, ARE THE ARGUMENTS -

```
(( ..............) A1 A2 ..... AN)
```

THE ARGUMENTS A1, A2 .... AN CAN BE FUNCTIONS (INCLUDING, OF COURSE, THEIR ARGUMENTS) THAT ARE EVALUATED BEFORE PROCEEDING TO FIND THE VALUE OF THE FUNCTION.

IT MUST BE NOTED THAT THE VARIABLES V1, V2 ... VN IN THESE EXPRESSIONS ARE DUMMY VARIABLES. IF ONE OF THEM IS EXCHANGED FOR ANOTHER NAME, THROUGHOUT THE LAMBDA EXPRESSION, THE VALUE ASSOCIATED ARE THE SAME AND HENCE THE VALUE OF THE FULL EXPRESSION IS UNCHANGED. FOR SHORT, I WILL CALL THE DUMMY VARIABLES JUST VARIABLES, AND THE ASSOCIATED VALUES, ARGUMENTS.

WE NOTE THEN THAT IN ORDER TO EVALUATE A FUNCTION, ARGUMENTS MUST BE ASSOCIATED TO ITS VARIABLES. THIS ASSOCIATION, CALLED BINDING, IS VALID ONLY DURING THE LAMBDA EXPRESSION. ANY VARIABLES CONTAINED IN THE FUNCTIONAL FORM BUT NOT BOUND BY THE VARIABLE LIST AFTER THE LAMBDA ARE CALLED FREE VARIA - BLES. BEFORE THE FUNCTION CAN BE EVALUATED, THEY MUST OF COURSE HAVE BEEN BOUND THEMSELVES - THIS USUALLY OCCURS ON A HIGHER LEVEL, THAT IS, THROUGH ANOTHER FUNCTION WHICH CALLS THE FUNCTION WHERE THE FREE VARIABLES OCCUR.

WE HAVE ALREADY SEEN HOW TO CONSTRUCT FUNCTIONS BUT NOT HOW TO GIVE IT A NAME. THIS PROBLEM IS SOLVED WITH THE FUNCTION DEFINE (CHAPTER 5). THE WHOLE DEFINITION OF ONE OR MORE FUNCTIONS WILL HAVE THE FORM -

```
(DEFINE
        (NAME1 (LAMBDA (V1 ...VN) (FORM1)))
        (NAME2 (LAMBDA (X1 ... XN) (FORM2)))
        ..................................
        (NAMEN (LAMBDA (Z1 ... ZN) (FORMN)) )
```

EACH PAIR IS FORMED BY AN ATOM (NOT A LIST) THAT WILL BE THE NAME OF THE FUNCTION AND BY THE LAMBDA EXPRESSION WHICH WILL BE CALLED BY THIS NAME.

NOW, WE CAN COMPLETE OUR FUNCTION TO FIND IF A LIST HAS ONE, TWO, THREE OR MORE ELEMENTS. LET US CALL THE FUNCTION ELEM.

```
(DEFINE (ELEM (LAMBDA(L) (COND
                ((ATOM L)(QUOTE (NOT A LIST)))
                ((NULL (CDR L))(QUOTE (ONE ELEMENT)))
                ((NULL (CDDR L))(QUOTE(TWO ELEMENTS)))
                ((AND)(QUOTE(THREE OR MORE ELEMENTS))) ))) )
```

THE NAME OF THE FUNCTION IS ELEM AND IT HAS ONE VARIABLE CALLED L.

FUNCTIONS MUST ALWAYS HAVE BEEN DEFINED BEFORE BEING USED, BUT WITHIN THE DEFINITION THE NAME OF THE FUNCTION BEING DEFINED MAY OCCUR - THIS MAKES RECURSIVE FUNCTIONS POSSIBLE. FOR INSTANCE, LET US CONSIDER A FUNCTION THAT GIVES THE FIRST ATOM NAME ON A LIST WHATEVER MAY BE ITS LEVEL DEPTH. THIS PROBLEM CAN BE HANDLE ONLY BY MEANS OF A RECURSIVE FUNCTION, AS THERE IS NO KNOWLEDGE OF THE LEVEL IN WHICH THE ATOM MAY BE FOUND.

THE FUNCTION MAY BE CALLED FIRSTATOM AND CAN BE DEFINED BY -

```
(DEFINE (FIRSTATOM (LAMBDA (L) (IF (ATOM (CAR L))
                                   (CAR L)
                                   (FIRSTATOM (CAR L))) )))
```

THERE IS ALSO ONLY ONE RECURSIVE CONDITION. IF AT ANY STAGE (CAR L) IS AN ATOM, THEN (CAR L) IS OUR ANSWER. BUT IF NOT, THEN WE MUST REPEAT THE RECURSION, EXCEPT THAT NOW THE ARGUMENT OF FIRSTATOM WILL NOT BE THE ORIGINAL VALUE OF L, BUT THE CAR OF IT.

SUPPOSE WE USE FIRSTATOM WITH THE ARGUMENT (((A)B)C). AT FIRST STAGE (CAR L) IS ((A)B) AND NOT AN ATOM. HENCE THE PREDICATE OF THE IF IS FALSE, FIRSTATOM IS CALLED AGAIN, AND ((A) B) GIVEN TO IT AS ITS ARGUMENTS. THE CAR OF THIS IS (A), STILL NOT AN ATOM - AND FIRSTATOM IS THEREFORE CALLED A THIRD TIME, THIS TIME, WITH ARGUMENT (A). BUT NOW, THE CAR OF THIS IS A, AN ATOM THE IF PREDICATE IS THEREFORE TRUE, AND A IS RETURNED AS THE ANSWER OF THIS IF, HENCE ALSO OF THE THIRD - LEVEL CALL. IN TURN THIS GIVES THE ANSWER FOR THE SECOND LEVEL, AND THEN FOR THE FIRST LEVEL. THUS THE FINAL RESULT, WHICH WILL BE PRINTED OUT, IS A.

NOTE THAT THIS FUNCTION, AS WE HAVE DEFINED IT, WOULD NOT RETURN AN ANSWER IF APPLIED TO L = ((( ) A), SINCE (CAR ( )) IS NOT DEFINED.
WE CAN ALSO DEFINE THE FUNCTION -

```
(DEFINE (FIRSTATOM (LAMBDA (L) (IF (ATOM L)
                                   L
                                   (FIRSTATOM (CAR L))) )))
```

NOTE THAT WRITING THE DEFINITIONS ON VARIOUS LINES DO NOT IN ANY WAY AFFECT THE MEANING, BUT DOES HELP TO UNDERSTAND HOW THE FUNCTION WORKS.

* EXAMPLES *

SUPPOSE WE EXECUTE -

```
(DEFINE   (L (A B C D))
(M (A B C))
(X  B))
```

1)

WE WISH TO VERIFY AFTER SOME MANIPULATIONS IF THE ATOM X STILL IS MEMBER OF THE LIST L. LET US CALL THE FUNCTION MEMBER AND WRITE IN ENGLISH THE STEPS TO SOLVE THIS PROBLEM.

1) IF THE LIST L IS NULL, THE RESULT IS FALSE.

2) IF THE FIRST ELEMENT OF L IS EQUAL TO X, THE RESULT IS TRUE.

3) OTHERWISE WE MAY APPLY THE FUNCTION MEMBER TO THE REMAINDER OF THE LIST(THE CDR, OF COURSE).

NOTE THAT THE DESCRIPTION IS RECURSIVE. IN THE LAST ITEM, WE APPLY THE FUNCTION MEMBER TO THE CDR OF THE LIST BECAUSE WE ALREADY KNOW THAT THE CAR IS NOT EQUAL TO THE ATOM X. THUS, THE NEW ARGUMENT OF MEMBER IS THE ATOM X, THAT REMAIN UNCHANGED AND THE LIST (B C D). WITH THESE ARGUMENTS, THE STEP 1 IS FALSE (LOOKING TO THE LIST L PREVIOUSLY DECLARED) BUT THE STEP 2 IS TRUE BECAUSE THE CAR OF (B C D) IS EQUAL TO THE ATOM X. SO, THE RECURSION IS ENDED, AND THE FUNCTION MEMBER WILL HAVE THE VALUE TRUE.
IN LISP WE CAN WRITE -

```
(DEFINE   (MEMBER (LAMBDA (X L) (COND
                  ((NULL L) (OR) )
                  ((EQ (CAR L) X) (AND))
                  ((AND)  (MEMBER X (CDR L))) ))) )
```

LET US ANALYSE THIS DEFINITION. THE DEFINE STATEMENT HAS ONE PAIR OF ARGUMENTS. THE FIRST ONE IS A NAME AND THE SECOND A FUNCTION DEFINITION INDICATED BY THE LAMBDA NOTATION. FOLLOWING, THE LIST OF VARIABLES FOR THE FUNCTION - THE ATOM AND THE LIST AND THEN THE BODY OF THE DEFINITION.

2)

AS THE FUNCTION EQ ONLY DEALS WITH ATOMS, WE WISH TO WRITE A LISP PROGRAM TO VERIFY IF TWO GIVEN ARGUMENTS ARE EQUALS. THE ARGUMENTS CAN BE ATOMS AND/OR LISTS AND IF THEY BOTH REFER TO SAME QUANTITIES THE FUNCTION MUST EVALUATE TO A VALUE TRUE, OTHERWISE IT MUST BE FALSE.

LET US WRITE IN ENGLISH THE STEPS TO SOLVE THIS PROBLEMS. LET US CALL THE FUNCTION EQUAL, AND THE ARGUMENTS X AND Y.

1) IF X IS AN ATOM, WE VERIFY IF X HAS THE SAME VALUE THAN Y, TREATING BOTH X AND Y AS ATOMS.

2) IF X IS NOT AN ATOM AND Y IS AN ATOM, THEN THE FUNCTION MUST BE FALSE.

3) AS WE KNOW THAT X IS A LIST, WE ASK IF IT IS AN EMPTY-LIST. IF THIS IS TRUE, WE ASK AGAIN IF Y IS NULL. IF BOTH ARE EMPTY-LISTS THE RESULT WILL BE TRUE BUT IF X IS EMPTY AND Y IS NOT, THE RESULT IS FALSE.

4) AS WE KNOW SURELY THAT X IS NOT AN EMPTY LIST, WE ASK IF Y IS NULL. IF IT IS TRUE, THE RESULT IS FALSE.

5) NOW WE KNOW THAT X AND Y ARE BOTH LISTS WITH ELEMENTS. WE MUST VERIFY IF THE LISTS HAS THE SAME ELEMENTS. FOR THIS WE CAN APPLY THE FUNCTION EQUAL RECURSIVELY, HAVING AS ARGUMENTS, THE FIRST ELEMENT OF EACH LIST. IF THIS EVALUATES TO TRUE, THEN WE MUST APPLY THE FUNCTION EQUAL TO THE REMAINDER OF THE LISTS.

6) OTHERWISE, THE FUNCTION IS FALSE.

IN LISP THIS FUNCTION SHOULD BE WRITTEN -

```
(DEFINE
        (EQUAL (LAMBDA (X Y) (COND
                ((ATOM X) (EQ X Y))
                ((ATOM Y) (OR))
                ((NULL X) (NULL Y))
                ((NULL Y)(OR))
                ((EQUAL (CAR X)(CAR Y)) (EQUAL (CDR X)(CDR Y)))
                ((AND) (OR))))) )
```

3)

GIVEN A LIST WITH N ELEMENTS, DEFINE A PROCEDURE TO REVERSE THE ORDER OF THE ELEMENTS OF THIS LIST. IN WORDS, THE STEPS TO SOLVE THIS PROBLEM COULD BE WRITTEN -

1) WE INTRODUCE A LIST CALLED FOR EXAMPLE, M, INITIALLY EMPTY.

2) IF THE LIST OF ELEMENTS IS EMPTY, THEN THE ANSWER IS THE LIST M.

3) OTHERWISE, APPLY THE WHOLE DEFINITION TO THE GIVEN LIST EXCEPT THE FIRST ELEMENT AND PUT THIS FIRST ELEMENT AS FIRST ELEMENT OF THE LIST M.

```
(DEFINE (REVERSE ( LAMBDA (L M) (IF
                        (NULL L) M
                        (REVERSE (CDR L) (CONS (CAR L) M))) )))
```

IN ORDER TO APPLY THIS FUNCTION WE MUST GIVE TO L THE VALUE OF THE LIST WHICH WE WANT TO REVERSE AND THE VALUE OF THE LIST M MUST BE EMPTY.
TO FACILITATE THE APPLICATION OF THIS DEFINITION, WE CAN CONSTRUCT AN AUXILIARY DEFINITION TO GIVE ONLY THE VALUE OF THE LIST L.

```
(DEFINE (REV+ (LAMBDA (L) (REVERSE L (LIST)))) )
```

NOTE THAT THE AUXILIARY FUNCTION REV+ APPLIES THE GIVEN LIST L AND THE LIST M IS EMPTY.

4)

DEFINE A FUNCTION TO FIND THE LAST ELEMENT OF A GIVEN LIST. THE STEPS TO SOLVE THIS PROBLEM ARE -

1) IF THE REMAINDER OF THE LIST, EXCEPT FOR THE FIRST ELEMENT IS EMPTY, THEN THE LAST ELEMENT IS THE FIRST ELEMENT OF THE GIVEN LIST.

2) OTHERWISE, APPLY THE WHOLE DEFINITION TO THE REMAINDER OF THE LIST EXCEPT THE FIRST ELEMENT.

```
(DEFINE (LAST (LAMBDA (L)  (IF
                        (NULL (CDR L)) (CAR L)
                        (LAST(CDR L)) ))))
```

THIS SIMPLE DEFINITION ONLY WORKS IF THE ARGUMENT IS A NON-EMPTY LIST.

5)

DEFINE A PROCEDURE TO CONSTRUCT A LIST FROM THE ELEMENTS O
TWO GIVEN LISTS. SUPPOSE THE LISTS ARE CALLED A AND B. THE STEPS ARE -

1) IF THE LIST A IS NULL THEN THE ANSWER IS THE LIST B.

2) OTHERWISE, CONSTRUCT A LIST WITH THE FIRST ELEMENT BEING THE FIRS
ELEMENT OF A AND THE REMAINDER BEING THE ELEMENTS EVALUATED BY TH
WHOLE DEFINITION APPLIED WITH THE FOLLOWING ARGUMENTS - THE LIST
EXCEPT THE FIRST ELEMENT AND THE LIST B.

```
(DEFINE (APPEND (LAMBDA (A B) (IF
                        (NULL A) B
                        (CONS(CAR A)(APPEND (CDR A)B)) ))))
```

NOTE THE DIFFERENCE BETWEEN CONS, LIST AND APPEND. SUPPOSE
WE HAVE DEFINED -

```
(DEFINE (L (A B C))
        (M (O P Q)))
```

THEN -

```
(CONS   L M)   =   ((A B C)O P Q)
(LIST   L M)   =   ((A B C)(O P Q))
(APPEND L M)   =   (A B C O P Q)
```

6)

GIVEN A LIST OF N ELEMENTS AND A NUMBER OF AN ELEMENT, DEFINE
A PROCEDURE TO GIVE THE ELEMENT OF THE LIST THAT CORRESPONDS TO THAT NUMBER.

1) IF THE LIST IS EMPTY SEND A MESSAGE

2) IF THE NUMBER IS 1 THE ANSWER IS THE FIRST ELEMENT OF THE LIST.

3) OTHERWISE APPLY THE WHOLE DEFINITION HAVING AS ARGUMENTS THE NUMBER
MINUS ONE AND THE LIST EXCEPT THE FIRST ELEMENT.

```
(DEFINE (ASSOCIATION (LAMBDA (X L) (COND
                ((NULL L)(QUOTE (ELEMENT REQUESTED DO NOT EXISTS)))
                ((EQ X 1)(CAR L))
                ((AND)(ASSOCIATION (SUB X1)(CDR L)))) )))
```

SUPPOSE WE WANT TO KNOW THE THIRD ELEMENT OF THE LIST (A B C)
THEN WE MAY APPLY THE FUNCTION ASSOCIATION HAVING AS ARGUMENTS THE NUMBER 3 AND
THE LIST (A B C). THE ANSWER WILL BE THE ATOM C.

7)

DEFINE A PROCEDURE TO DELETE FROM A GIVEN LIST A GIVEN ELEMENT BEING A LIST OR AN ATOM.

IN WORDS, THE STEPS TO SOLVE THIS PROBLEM ARE -

1) IF THE LIST IS NULL THEN THE ANSWER IS AN EMPTY LIST

2) IF THE FIRST ELEMENT OF THE LIST IS EQUAL TO THE GIVEN ELEMENT, THEN APPLY THE WHOLE DEFINITION TO THE REMAINDER OF THE LIST EXCEPT THE FIRST ELEMENT.

3) OTHERWISE CONSTRUCT A LIST HAVING AS FIRST ELEMENT THE FIRST ELEMENT OF THE GIVEN LIST AND THE REMAINDER THOSE EVALUATED BY THE WHOLE DEFINITION APPLIED TO THE REMAINDER OF THE LIST EXCEPT THE FIRST ELEMENT.

```
(DEFINE (DELETE (LAMBDA (A L) (COND
                  ((NULL L) (LIST))
                  ((EQUAL A (CAR L))(DELETE A (CDR L)))
                  ((AND)(CONS (CAR L) (DELETE A (CDR L))) ))))
```

NOTE THAT IN THIS DEFINITION WE USE ANOTHER FUNCTION DEFINED BY THE PROGRAMMER, THAT IS, THE FUNCTION EQUAL. BEFORE THE FUNCTION DELETE IS EXECUTED, THE FUNCTION EQUAL MUST BE ENTERED TO THE COMPUTER.

* ERRORS IN THE USE OF LAMBDA *

THERE ARE THREE TYPES OF ERROR MESSAGE IN THE APPLICATION OF LAMBDA NOTATION.

1) WHEN THERE ARE EXTRA PARENTHESIS IN THE LIST OF VARIABLES. FOR INSTANCE -

```
(LAMBDA ((A) (B) ) C)
ERROR... UNPAIRED ARGUMENT OR VARIABLE IN LAMBDA
```

2) WHEN THERE IS NO FORM. FOR INSTANCE -

```
(LAMBDA (A B))
ERROR... UNPAIRED ARGUMENT OR VARIABLE IN LAMBDA
```

3) WHEN THERE IS A NUMBER AS FORM. FOR INSTANCE -

```
(LAMBDA (A B) 4)
ERROR... UNPAIRED ARGUMENT OR VARIABLE IN LAMBDA
```

* 11. PROPERTIES OF LISTS *

IN MANY APPLICATIONS IT IS USEFULL TO HAVE SIDE BY SIDE WITH THE VALUES, A LIST OF PROPERTIES. EVERY ATOM THAT IS NOT A FUNCTION NAME, CAN HAVE PROPERTIES.

THERE ARE THREE FUNCTIONS FOR TO HANDLE WITH PROPERTIES -

SETPROP
LETPROP
PROP

* 11.1 SETPROP *

THIS FUNCTION HAS ANY NUMBER OF PAIRS OF ARGUMENTS AND HAS THE FORM -

(SETPROP (X1 P1)(X2 P2) ..... (XN PN))

SETPROP IS SIMILAR TO THE FUNCTION DEFINE. IT NOT EVALUATES ITS ARGUMENTS. P1 P2 ..... PN WILL BE ASSIGNED RESPECTIVELY AS PROPERTIES OF X1, X2 ..... XN.

* EXAMPLES *

```
(SETPROP (A 1)
         (B (X Y Z))
         (K (6 A))
         (KING SPADES) )
```

2)

```
(SETPROP (A (Q R T))
         (Q (S V))
         (R (U X)) )
```

* 11.2 PROP *

THIS FUNCTION HAS ONE ARGUMENT AND HAS THE FORM -

(PROP X)

IT EVALUATES THE ARGUMENT AND GIVES ITS PROPERTY. FOR INSTANCE, IF WE EXECUTE -

```
(SETPROP (A 5)
         (B (X Y Z)) )
```

AND THEN -

```
(PROP (QUOTE A))
```

THE RESULT WILL BE 5. AND,

```
(PROP (QUOTE B))
```

THE RESULT WILL BE (X Y 7)

* EXAMPLES *

1) IF WE EXECUTE

```
(SETPROP (G (L M))
         (P Q))
THEN -
         (PROP (QUOTE G))  =  (L M)
         (PROP (QUOTE P))  =  Q
         (PROP(PROP(QUOTE G)))  =  (M)
```

IN THIS EXAMPLE, THE MACHINE WILL EVALUATE THE INTERNAL PROP AND THE RESULT WILL BE (L M). THEN, THE EXTERNAL ONE IS EVALUATED WITH THE AR - GUMENT (L M). WHEN THE ARGUMENT EVALUATES TO A LIST, THEN THE RESULT OF PROP IS THE CDR OF THIS LIST. THUS, THE RESULT OF THE LAST EXAMPLE IS THE LIST (M).

2)

IF WE EXECUTE

```
(SETPROP (A (Q R T U))
         (Q (S B)))
THEN -
         (PROP (QUOTE A))  =  (Q R T U)
         (PROP(QUOTE Q))  =  (S B)
         (PROP(PROP(QUOTE A)))  =  (R T U)
         (PROP(PROP(PROP(QUOTE Q))))  =  ( )
```

* 11. 3 LETPROP *

THIS FUNCTION HAS TWO ARGUMENTS AND HAS THE FORM -

```
(LETPROP  X  A)
```

IT EVALUATES BOTH ARGUMENTS AND THE VALUE OF A WILL BE THE PROPERTY OF THE VALUE OF X. FOR INSTANCE, IF WE EXECUTE

```
(SETPROP (A (L M N))
         (B  K))
AND THEN -
         (LETPROP(QUOTE A)(QUOTE X))
```

THEN THE OLD PROPERTY OF A, (L M N) IS REPLACE BY THE NEW ONE, X.

```
(LETPROP (QUOTE B)(PROP(QUOTE A)))
```

IN THIS EXAMPLE THE NEW PROPERTY OF B IS (PROP(QUOTE A)), THAT IS X.

```
(LETPROP(QUOTE A)(LIST(PROP(QUOTE A))(PROP(QUOTE B))))
```

THE NEW PROPERTY OF A WILL BE THE LIST (X X).

IT MUST BE NOTED THAT WE CANNOT EVALUATE THE CAR, CDR ETC OF PROPERTIES. THIS IS BECAUSE THEY ARE DEFINED AS PROPERTIES, NOT AS VALUES.
BUT, WE CAN EXECUTE -

```
(SETPROP (A (L M)))
(DEFINE  (A (X Y)))
AND THEN -
         (PROP (QUOTE A))  =  (L M)
         (VAL (QUOTE A))  =  (X Y)
```

* EXAMPLES *

1) SUPPOSE WE HAVE DEFINED THE FOLLOWING -

```
(SETPROP  (M VECTOR)
          (VECTOR (5 CM, FROM LEFT TO RIGHT))
          (N NUMBER)
          (NUMBER 45)
          (I COMPLEX)
          (COMPLEX (A + IB)) )
```

AND, IF WE DEFINE -

```
(DEFINE (MAGNITUDE (LAMBDA (L) (COND
          ((EQ L (PROP(QUOTE M)))(PROP(PROP(QUOTE M))))
          ((EQ L (PROP(QUOTE N)))(PROP(PROP(QUOTE N))))
          (EQ L (PROP(QUOTE I)))(PROP(PROP(QUOTE I))))
          ((AND)(QUOTE(MAGNITUDE NOT DEFINED))) ))))
```

IF THE ARGUMENT L IS THE ATOM VECTOR, THE RESULT OF THE FUNCTION WILL BE THE PROPERTY OF VECTOR, THAT IS, (5 CM, FROM LEFT TO RIGHT). IF IT IS THE ATOM NUMBER, THE RESULT IS THE PROPERTY OF NUMBER, 4K. IF IT IS THE ATOM COMPLEX, THE RESULT IS THE EXPRESSION (A + IB). IF ANYTHING ELSE, THE MESSAGE (MAGNITUDE NOT DEFINED) IS TYPED.

2) SUPPOSE IT WAS DEFINED THE FOLLOWING PROPERTIES -

```
(SETPROP  (A 24)
          (B 153)
          (C 258)
          (D 1))
```

AND THE LIST -

```
(DEFINE (L (A B C D)))
```

WRITE A PROGRAM THAT PUT THE LIST L IN DESCENDING ORDER OF ITS PROPERTIES.

```
(DEFINE
          (COLOCA(LAMBDA(X M)(COND
               ((NULL M)(CONS X M))
               ((SG (PROP X)(PROP(CAR M)))(CONS X M))
               ((AND)(CONS(CAR M)(COLOCA X (CDR M)))) )))

          (COL* (LAMBDA(L M)( IF (NULL L) M
               (COL* (CDR L)(COLOCA (CAR L) M))) ))

          (ORDENACAO(LAMBDA(M)(LET (QUOTE L) (COL* L (LIST)) ))) )
```

```
* ERRORS IN THE USE OF LETPROP *
  ******************************

(LETPROP A B)
ERROR... VARIABLE NAME HAS NO VALUE

(LETPROP 1 2)
ERROR... NUMERICAL ARGUMENT FOR LET/LETPROP

(LETPROP (QUOTE A) (B))
ERROR... UNDEFINED ARGUMENT FOR FUNCTION DEFINITION
```

12. APPLICATIONS OF FUNCTIONS

TO COMPLETE THE ENTIRE MECHANISM OF LISP WE NEED SOME FUNCTIONS FOR THE EXECUTION OF OTHER FUNCTIONS. THESE FUNCTIONS ARE -

EVAL
APPLY

A DEFINITION USING THE FUNCTION DEFINE MAY BE APPLIED TO VALUES OF ITS ARGUMENTS BY MEANS OF THESE FUNCTIONS. THEY HAVE THE FORM -

```
(APPLY  F  A1 A2 ..... AN)
(EVAL   F  A1 A2 .....  AN)
```

THE DIFFERENCE BETWEEN THEM IS THAT APPLY TAKES THE NAMES A1, A2 ..... AN AS ITS ARGUMENTS, THAT IS, IT QUOTES THEM, AND GIVES THEM TO THE FUNCTION F TO CARRY OUT, WHILE EVAL DOES NOT QUOTE THEM BUT EVALUATES THE ARGUMENTS. HENCE, THE FOLLOWING IDENTITY IS TRUE -

```
(APPLY  F A1 A2 ... AN)  =  (EVAL F (QUOTE A1)(QUOTE A2) ... (QUOTE AN))
```

FOR INSTANCE, LETS USE THE FUNCTION REVERSE, EARLIER DEFINED AND LETS APPLY THIS FUNCTION USING AS ARGUMENTS THE LISTS A AND B DEFINED AS FOLLOWS -

```
(DEFINE   (A (1 2 3 4 5 6) )
          (B (8 10 12 14 16 18) ) )
```

WE CAN USE THE FUNCTION EVAL AS FOLLOWS -

```
(EVAL REVERSE A)
```

THE ANSWER WILL BE

```
(6 5 4 3 2 1)
```

```
(EVAL REVERSE B)
```

THE ANSWER WILL BE

```
(18 16 14 12 10 8)
```

THE FUNCTION APPLY PERFORMS THE SAME OPERATION BUT THE ARGUMENTS NEED NOT BE DEFINED PREVIOUSLY. FOR INSTANCE -

```
(APPLY REVERSE (9 7 6 5 8))
```

BOTH FUNCTIONS CAN BE USED AS A PROPOSITION IN A FUNCTION DESCRIPTION. FOR INSTANCE.

```
(DEFINE   (X (LAMBDA (V) V))
          (TEST (LAMBDA (A B) (IF (EQ A B)
                                  (APPLY X EQUALS)
                                  (APPLY X DIFFERENTS)))))
```

THE PURPOSE OF THESE DEFINITIONS IS TO PRINT OUT THE MESSAGE-EQUALS, IF TWO ATOMS ARE EQUALS OTHERWISE, THE MESSAGE IS DIFFERENTS. THE FUNCTION X ONLY PRINTS THE VALUE OF ITS ARGUMENT. THE FUNCTION APPLY IS USED TO ALLOW THE EXECUTION OF THE FUNCTION X.

* EXAMPLES *

IF WE DEFINE THE FUNCTIONS MEMBER, EQUAL AND REVERSE (PAGE 36), WE CAN EXECUTE THEM BY WRITING -

```
(APPLY EQUAL (A B)(A B))
(APPLY REVERSE (A B C))
(APPLY MEMBER X (A B X))
```

OR, IF WE EXECUTE -

```
(DEFINE (L (A C))
        (M (A D))
        (X A))
```

AND THEM -

```
(EVAL MEMBER X M)
(EVAL REVERSE L)
(EVAL EQUAL L M)
```

* ERRORS IN THE APPLICATION OF APPLY/EVAL *

1)

```
(APPLY 1 A)
(EVAL  1 A)
```

ERROR... NUMERICAL ARGUMENT FOR APPLY/EVAL

2)

```
(APPLY () A)
(EVAL  () A)
```

ERROR... EMPTY LIST ARGUMENT FOR APPLY/EVAL

3)

```
(APPLY A B)
(EVAL  A B)
```

ERROR... UNDEFINED ARGUMENT FOR FUNCTION DEFINITION

* 13. SPECIAL FUNCTIONS *

TO COMPLETE THE LIST OF FUNCTIONS AVAILABLE IN LISP FOR THE IBM 1620 WE WILL PRESENT THREE MORE. THEY ARE -

PUT
PUTQ
ITER

* 13.1 PUT *

THIS FUNCTION HAS THE FORM -

(PUT X F A B C .....)

WHERE F IS A FUNCTION AND A B C .... ITS ARGUMENTS. THE FUNCTION AND THE ARGUMENTS ARE EVALUATED BUT X, IS NOT EVALUATED. THE FUNCTION PUT ENABLES THE EVALUATION OF THE FUNCTION F APPLIED WITH ITS ARGUMENTS A B C ... AND THE RESULT OF THIS EVALUATION WILL BE STORED AS THE VALUE OF X.

THE FOLLOWING IDENTITY IS TRUE -

(PUT X F A B ...) = (LET (QUOTE X) (EVAL F A B ...))

* EXAMPLES *

1) IF WE EXECUTE -

```
(DEFINE   (A (9 8))
          (B (7 6)))
```

AND THEN -

```
(PUT  A  CAR  B)
```

THE NEW VALUE OF A WILL BE THE ATOM 7, AND THE OLD VALUE IS LOST. NOTE THAT THE FIRST ARGUMENT DO NOT NEED TO BE QUOTED AS IT IS NOT EVALUATED. IN THIS CASE, THE FUNCTION IS CAR AND ITS ARGUMENT IS THE LIST B, PREVIOUSLY DEFINED.

2)

```
(PUT CD LIST A B)
```

HERE, WE ARE DEFINING A NEW LIST NAMED CD WITH THE VALUE ((9 8)(7 6)).

* 13.2 PUTQ *

THIS FUNCTION HAS THE FORM -

(PUTQ X F A B C ....)

IT IS SIMILAR TO THE FUNCTION PUT. THE ONLY DIFFERENCE IS THAT THE ARGUMENTS OF THE FUNCTION F ARE CONSIDERED AS QUOTED.

THE FOLLOWING IDENTITY IS TRUE -

(PUTQ X F A B ...) = (LET (QUOTE X) (APPLY F A B

* EXAMPLES *

1)

```
(PUTQ  A  CAR (C D E))
```

COMMENTS - THE ARGUMENTS A AND (C D E) ARE NOT EVALUATED. THE FUNCTION CAR IS APPLIED TO THE LIST (C D E) AND THE RESULT IS THE ATOM C. THIS WILL BE THE VALUE OF A.

2)

```
(PUTQ CD LIST A B C D E F G)
```

RESULT - CD = (A B C D E F G)

* 13.3 ITER *

THIS FUNCTION HAS TWO ARGUMENTS. IT HAS THE FORM -

```
(ITER  P  A)
```

WHERE P IS A PREDICATE. IT EVALUATES A, THEN EVALUATES P. IF P EVALUATES TO TRUE, IT RETURNS THE VALUE TRUE. OTHERWISE, IT RECYCLES TO THE EVALUATIONS OF A THEN P AND SO ON, UNTIL P EVALUATES TO TRUE.

THE PREDICATE P MUST SOMETIMES EVALUATES TO TRUE AS OTHERWISE THE PROGRAM WILL GET INTO A NON-ENDING CYCLE. THE USEFULLNESS OF ITER IS IN THE SIDE EFFECTS OF THE EVALUATION OF A. THESE SIDE EFFECTS CAN BE ANY SETTING OF VALUE OR PROPERTIES.

* EXAMPLES *

THE PROGRAM WRITTEN BELOW CONSTRUCTS A LIST V WITH THE OCURRENCES OF THE DIGITS 0 UNTIL 9 GIVEN BY THE FUNCTION RANDOM.

```
(DEFINE

        (V (0 0 0 0 0 0 0 0 0 0))

        (TESTE(LAMBDA()(ORDEM(DIV(RANDOM)1000))))

        (ORDEM(LAMBDA(N)(LET(QUOTE V)(ORD* 0 V)) ))

        (ORD* (LAMBDA(K L)(IF (EQ N K)
                  (CONS (ADD 1 (CAR L))(CDR L))
                  (CONS (CAR L)(ORD* (ADD 1 K)(CDR L))) )))
```

FIRST WE DEFINE THE LIST V WITH 10 ELEMENTS TO ACUMULATE THE OCURRENCES OF THE RANDOM NUMBERS (RESPECTIVELY FROM ZERO TO NINE). THEN WE HAVE THREE FUNCTIONS - TESTE,ORDEM AND ORD*. TESTE APPLIES ORDEM WITH ONE ARGUMENT. THIS ARGUMENT IS THE RANDOM NUMBER DIVIDED BY 1000. SO, THIS ARGUMENT WILL BE AN INTEGER DIGIT. ORDER LETS THE LIST V BE THE VALUE GIVEN BY THE FUNCTION ORD*. THIS FUNCTION PUTS THE OCURRENCE OF THE RANDOM NUMBER IN ITS RESPECTIVE PLACE ON THE LIST. IT MUST BE NOTED THAT WE ADD 1 TO THE VALUE OF EACH ELEMENT ON THE LIST, SO AS NOT TO LOSE THE EARLIER EVENTS.

NOW WE CAN APPLY THE FUNCTION ITER. SUPPOSE WE WISH TO APPLY THE FUNCTION TESTE 10 TIMES.

```
(DEFINE (P 0))

        (ITER(AND(LET(QUOTE P)(ADD P 1))(EQ P 10))(TESTE))
```

THE PREDICATE IS THE FUNCTION (AND(LET ...)) USES THE ATOM P TO COUNT THE NUMBER OF APPLICATIONS OF THE FUNCTION (TESTE). THE PROCESS WILL STOP WHEN THE PREDICATE BECOMES TRUE, THAT IS, WHEN P EQUALS 10. THE ANSWER WILL BE TYPED AS BELOW, THAT IS, IT WILL BE THE LIST V THAT WILL CONTAIN THE NUMBER OF OCURRENCES GIVEN BY TESTE.

```
(1 0 1 1 1 2 0 2 1 1)
```

* ERRORS IN THE APPLICATION OF PUT/PUTQ *

1)

```
        (PUT A Z (QUOTE Z))
        (PUTQ A Z (QUOTE Z))
ERROR... UNDEFINED ARGUMENT FOR FUNCTION DEFINITION
```

2)

```
        (PUT (QUOTE X) Z (QUOTE A))
        (PUTQ (QUOTE X) Z (QUOTE A))
ERROR... FULL LIST ARGUMENT FOR PUT/PUTQ
```

3)

```
        (PUT 1 2 3)
        (PUTQ 1 2 3)
ERROR... NUMERICAL ARGUMENT FOR PUT/PUTQ
```

4)

```
        (PUT () A B)
        (PUTQ () A B)
        (PUT A () B)
        (PUTQ A () B)
        (PUT A B ())
        (PUTQ A B ())
ERROR... EMPTY LIST ARGUMENT FOR PUT/PUTQ
```

5)

```
        (PUTQ A 2 T)
ERROR... NUMERICAL ARGUMENT FOR APPLY/EVAL
```

14. A COMPLETE LISP PROGRAM

TO ILUSTRATE THE APPLICATION OF LISP LANGUAGE LET US PRESENT A COMPLETE PROGRAM. WE ARE CONCERNED WITH THE ALGEBRAIC DIFFERENTIATION. THUS, OUR FUNCTION MUST GIVE THE ALGEBRAIC DERIVATIVE OF A GIVEN ALGEBRAIC EXPRESSION. THE FUNCTION WILL HAVE TWO ARGUMENTS.

1) THE EXPRESSION
2) THE VARIABLE OF DERIVATION

THE SIMPLEST CASES ARE THE DERIVATIVE OF A VARIABLE WITH RESPECT TO ITSELF AND OF A VARIABLE WITH RESPECT TO ANOTHER VARIABLE THAT IS, A CONSTANT. THUS,

```
IF      F = X     THEN     DF(X) = 1
IF      G = A     THEN     DG(X) = 0
```

THE MAIN PROGRAN OF DERIVATIVES IS CALLED DIF AND IS SUB DIVIDED INTO TWO SUB PROGRAMS -

```
DIF2  -  OPERATIONS WITH ONE ARGUMENT
DIF3  -  OPERATIONS WITH TWO ARGUMENTS
```

THE MAIN FUNCTION DIF WILL BE A FUNCTION WITH TWO ARGUMENTS -

(EXPRESSION VARIABLE OF DERIVATION)

THE SUB PROGRAM DIF2 WILL PROCESS THE FOLLOWING FUNCTIONS

```
+ X                    SQRT X
- X                    COSEC X
SIN X                  ARCSIN X
COS X                  ARCCOS X
TAN X                  ARCTAN X
COT X                  ARCCOT X
SEC X                  ARCSEC X
EXP X                  ARCCOSC X
LN  X
```

IN THE LISP PROGRAM THESE FUNCTIONS, THAT IS THE EXPRESSION, WILL APPEAR AS A LIST WITH TWO ELEMENTS -

(OPERATION EXPRESSION)

FOR INSTANCE,

```
(SIN X)
(ARCCOS X)
```

THE SUB PROGRAM DIF3 WILL PROCESS THE FOLLOWING OPERATIONS -

```
ADDITION .          +
SUBTRACTION         -
MULTIPLICATION      *
DIVISION            /
EXPONENTIATION      **
```

IN THE LISP PROGRAM THESE OPERATIONS WILL APPEAR AS A LIS
WITH THREE ELEMENTS -

(EXPRESSION OPERATION EXPRESSION)

IT MUST BE POINTED THAT THE EXPRESSION COULD BE A SIMPLE VA
RIABLE, THAT IS AN ATOM, OR A LIST WITH OTHER OPERATIONS. ANOTHER POINT IS THA
THE ANSWER GIVED BY MACHINE MUST BE IN ACCORDING WITH THESE RULES, THAT IS, TH
EXPRESSION DIFFRENTIATED COULD BE COMPOSED BY ATOMS, REPRESENTING SINGLE VARIA
BLES OR BY LISTS WITH TWO OR THREE ELEMENTS DEPENDING ON THE RESULT OF THE OPER
TION EMPLOYED.

FOR INSTANCE, THE DERIVATIVE OF SIN X * COS X COULD BI
WRITTEN IN THE FORM -

((SIN X) * (COS X))

THAT IS,

(SIN X)	-	EXPRESSION
*	-	OPERATION
(COS X)	-	EXPRESSION

AS THE WHOLE EXPRESSION HAS THREE ELEMENTS, DIF3 IS APPLIED
AND THE FINAL RESULT IS -

((((COS X) * 1) * (COS X)) + ((- (SIN X) * 1)) * (SIN X)))

THIS IS A LIST WITH THREE ELEMENTS. THE EXPRESSION -
(((COS X) * 1) * (COS X))
THE OPERATION SIGN +
AND BY LAST ANOTHER EXPRESSION -
((- (SIN X) * 1)) * (SIN X))

IT MUST BE NOTED THAT THIS RESULT IS STILL IN A VERY PRIMITI
VE, THAT IS, MANY SIMPLIFICATIONS CAN BE DONE IN THIS RESULT. FOR THIS, A FUNC-
TION NAMED SIMP IS INTRODUCED AND IT FOLLOWS THE SAME LOGIC OF DIF, THAT IS,
CONSTRUCTING TWO SUB PROGRAMS SIMP2 AND SIMP3 FOR SIMPLIFICATIONS OF EXPRESSI -
ONS WITH TWO ELEMENTS AND THREE ELEMENTS RESPECTIVELY.

THUS THE DERIVATIVE OF ((SIN X) * (COS X)) WITH RESPECT
TO X WILL BE -

(((COS X) ** 2) - ((SIN X) ** 2))

FOR SIMPLICITY WE CONSTRUCT TWO MORE PROGRAMS - THE FUNCTION
DERIV, WITH TWO ARGUMENTS, THE SAME AS THAT DIF WHICH APPLIES THE FUNCTIONS SIMP
AND DIF, AND THE FUNCTION DERN FOR THE APPLICATION OF DERIVATIVES OF FUNCTIONS
OF ANY ORDER. THIS FUNCTION HAS THREE ARGUMENTS, THE FIRST TWO, THE SAME AS
THAT DIF AND THE THIRD THE ORDER OF THE DERIVATIVE. FOR INSTANCE, IF WE WISH TO
FIND THE DERIVATIVE OF ORDER 3 OF SIN X WE MAY WRITE -

(APPLY DERN (SIN X) X 3)

```
(DEFINE

(DERN(LAMBDA(E X N)(DRN(SIMP E) N)))

(DRN(LAMBDA(E N)(IF (EQ 0 N) E (DRN (DERIV E X)(SUB N 1))) ))

(DERIV (LAMBDA (E X) (SIMP (DIF E X))) )

(DIF (LAMBDA (E X) (COND ((EQ E X) 1)
                         ((ATOM E) 0)
                         ((NULL E) (QUOTE NULLIST))
                         ((NULL (CDR E)) (DIF (CAR E) X))
                         ((NULL (CDDR E)) (DIF2 (CAR E) (CADR E)))
                         ((NULL (CDR (CDDR E))) (DIF3 (CAR E) (CADR E)
                                                      (CAR (CDDR E))) )
                         ((AND) (QUOTE INACCEPTABLE )) ))

(DIF2 (LAMBDA (OP E) (COND
      ((EQ OP (QUOTE +)) (DIF E X ))
      ((EQ OP (QUOTE -)) (LIST OP (DIF E X)))
      ((EQ OP (QUOTE SIN)) (LIST (LIST (QUOTE COS) E)
                                       (QUOTE *)
                                       (DIF E X)))
      ((EQ OP (QUOTE COS)) (LIST     (QUOTE -)
                                     (LIST (LIST (QUOTE SIN) E)
                                           (QUOTE *)
                                           (DIF E X))) )
          ((EQ OP (QUOTE TAN))(LIST(CONS(LIST(QUOTE SEC) E)(QUOTE (** 2)
                                    ))
                                   (QUOTE *)
                                   (DIF E X)))
  ((EQ OP (QUOTE COT)) (LIST (QUOTE -)
                       (LIST(CONS(LIST(QUOTE COSEC) E) (QUOTE (** 2)))
                                          (QUOTE *)
                                             (DIF E X))) )
     ((EQ OP (QUOTE SEC)) (LIST (LIST (LIST OP E)
                                      (QUOTE *)
                                      (LIST (QUOTE TAN) E))
                                (QUOTE *)
                              (DIF E  X)))
     ((EQ OP (QUOTE COSEC)) ( LIST (QUOTE -)
                                      (LIST (LIST (LIST OP E)
                                                  (QUOTE *)
                                                  (LIST (QUOTE COT) E))
                                            (QUOTE *)
                                            (DIF E X ))) )
      ((EQ OP (QUOTE SQRT))  (LIST (DIF E X)
                                   (QUOTE /)
                                   (LIST 2 (QUOTE *) (LIST (QUOTE SQRT)
                                                           E))) )
      ((EQ OP (QUOTE EXP)) (LIST (LIST OP E) (QUOTE *) (DIF E X)))
      ((EQ OP (QUOTE LN)) (LIST(DIF E X)(QUOTE /) E ))
```

```
     ((EQ OP (QUOTE ARCSIN)) (LIST (DIF E X )
                                   (QUOTE /)
                                    (LIST (QUOTE SQRT)
                                          (LIST 1 (QUOTE -)
                                           (CONS E (QUOTE (** 2)))
                                                   ))))
     ((EQ OP (QUOTE ARCCOS)) (LIST (QUOTE -)
                                   (LIST (DIF E X)
                                         (QUOTE /)
                                         (LIST (QUOTE SQRT)
                                               (LIST 1 (QUOTE -)
                                                     (LIST E
                                                           (QUOTE **)
                                                           2))) )))
     ((EQ OP (QUOTE ARCTAN)) (LIST (DIF E X)
                                   (QUOTE /)
                                   (LIST 1 (QUOTE +)
                                         (LIST E
                                               (QUOTE **)
                                                2))) )
     ((EQ OP (QUOTE ARCCOT)) (LIST (QUOTE -)
                                   (LIST (DIF E X)
                                         (QUOTE /)
                                         (LIST 1 (QUOTE +)
                                           (LIST E(QUOTE **)2)))))
      ((EQ OP (QUOTE ARCSEC)) (LIST (DIF E X)
                                    (QUOTE /)
                                    (LIST E
                                          (QUOTE *)
                                            (LIST (QUOTE SQRT)
                                            (LIST(CONS E (QUOTE(** 2))
                                                )
                                                    (QUOTE -)
                                                    1))) ))
      ((EQ OP (QUOTE ARCCOSEC)) (LIST (QUOTE -)
                                      (LIST (DIF E X)
                                            (QUOTE /)
                                            (LIST E
                                                 (QUOTE *)
                                                 (LIST (QUOTE SQRT)
                                                       (LIST(CONS E(QUOTE(
                                         ** 2)))
                                                        (QUOTE -)
                                                       1))))))
      ((AND) (QUOTE BADOP))) ))


(DIF3 (LAMBDA (A OP B) (COND
     ((OR (EQ OP (QUOTE +))
          (EQ OP (QUOTE -))) (LIST (DIF A X ) OP (DIF B X)))
      ((EQ OP (QUOTE *)) (LIST (LIST (DIF A X) OP B)
                               (QUOTE +)
                               (LIST A OP (DIF B X))) )
```

```
      ((EQ OP (QUOTE /)) (LIST (LIST (LIST (DIF A X) (QUOTE *) B)
                                     (QUOTE -)
                                     (LIST A (QUOTE *) (DIF B X)))
                               OP
                               (LIST B (QUOTE **) 2)))
      ((EQ OP (QUOTE **)) (LIST(LIST(LIST B (QUOTE *)
                               (LIST A OP (CONS B (QUOTE ( - 1 )) )))  -
                                     (QUOTE *) (DIF A X))
                                (QUOTE +)
                               (LIST(LIST(LIST A OP B)
                                     (QUOTE *)(LIST (QUOTE LN) A) )
                                     (QUOTE *) (DIF B X)  )) )
       ((AND) (QUOTE BADOP))) ))


(SIMP (LAMBDA (E) (COND
     ((ATOM E) E)
     ((NULL (CDR E)) (SIMP (CAR E)))
     ((NULL (CDDR E)) (SIMP2 (CAR E) (SIMP (CADR E))) )
     ((NULL (CDR (CDDR E))) (SIMP3 (SIMP (CAR E)) (CADR E)
                    (SIMP (CAR (CDDR E))) ))
     ((AND) E))) )

(SIMP2 (LAMBDA (A B) (COND
     ((EQ A (QUOTE +)) B)
      ((EQ A (QUOTE -)) (IF(NUM  B ) (SUB 0 B) (LIST A B)))
      ((AND (EQ B 0)
            (OR (EQ A (QUOTE EXP))
                (EQ A (QUOTE COS))
                (EQ A (QUOTE SEC))) )          1)
      ((AND (EQ B 0)
           (OR (EQ A (QUOTE SIN))
               (EQ A (QUOTE TAN))
               (EQ A (QUOTE COSEC))
               (EQ A (QUOTE ARCSIN))
               (EQ A (QUOTE ARCTAN))
               (EQ A (QUOTE SQRT))) )          0)
     ((AND (EQ B 1)
           (EQ A (QUOTE SQRT)))                   1)
     ((AND (EQ B 1)
           (OR (EQ A (QUOTE LN))
               (EQ A (QUOTE ARCCOS))) )          0)
      ((AND) (LIST A B))) ))

 (SIMP3 (LAMBDA (A B C)   (COND
     ((EQ B (QUOTE +)) (COND  ((AND (NUM A) (NUM C)) (ADD A C))
                              ((EQ A 0) C)
                              ((EQ C 0) A)
                              ((EQUAL A C) (LIST 2 (QUOTE *) A))
                              ((AND) (LIST A B C))) )
     ((EQ B (QUOTE -)) (COND ((AND (NUM A) (NUM C)) (SUB A C))
                              ((EQ A 0) (LIST (QUOTE -) C))
                              ((EQ C 0) A)
                              ((EQUAL A C) 0)
                              ((AND) (LIST A B C))) )
```

```
     ((EQ B (QUOTE *)) (COND ((AND (NUM A) (NUM C))(MULT A C))
                             ((EQ A 0) 0)
                             ((EQ C 0) 0)
                             ((EQ A 1) C)
                             ((EQ C 1) A)
                             ((EQ A -1) (LIST (QUOTE -) C))
                             ((EQ C -1) (LIST (QUOTE -) A))
                             ((EQUAL A C) (LIST A (QUOTE **) 2))
                             ((AND) (LIST A B C ))) )
     ((EQ B (QUOTE /)) (COND  ((EQ A 0) 0)
                              ((EQ C 1) A)
                              ((EQ C -1) (LIST (QUOTE -) A))
                              ((EQUAL A C) 1)
                             ((AND) (LIST A B C))) )
     ((EQ B (QUOTE **)) (COND ((EQ C 0) 1)
                              ((EQ C 1) A)
                              ((EQ C -1)  (LIST 1 (QUOTE /) A))
                              ((EQ A 0) 0)
                              ((EQ A 1) 1)
                              ((AND) (LIST A B C))) )
     ((AND) (LIST A B C ))) ))


(EQUAL (LAMBDA (X Y) (COND
     ((ATOM  X) (EQ X Y ))
     ((ATOM Y) (OR))
     ((NULL X) (NULL Y))
     ((NULL Y) (OR))
     ((EQUAL (CAR X) (CAR Y)) (EQUAL (CDR X) (CDR Y)))
     ((AND) (OR))) ))
            )
```

BELOW THERE ARE SOME APPLICATIONS OF THE FUNCTIONS DERIV AND DERN. EACH INQUIRY TO THE MACHINE IS FOLLOWED BY THE ANSWER.

```
(APPLY DERIV (- Y) X)
0

(APPLY DERIV (LN X) X)
(1 / X)

(APPLY DERIV (SIN X) X)
(COS X)

(APPLY DERIV (COS X) X)
(- (SIN X))

(APPLY DERIV (TAN X) X)
((SEC X) ** 2)

(APPLY DERIV (COT X) X)
(- ((COSEC X) ** 2))

(APPLY DERIV (SEC X) X)
((SEC X) * (TAN X))

(APPLY DERIV (COSEC X) X)
(- ((COSEC X) * (COT X)))

(APPLY DERIV (SQRT X) X)
(1 / (2 * (SQRT X)))

(APPLY DERIV (EXP X) X)
(EXP X)

(APPLY DERIV (ARCSIN X) X)
(1 / (SQRT (1 - (X ** 2))))

(APPLY DERIV (ARCCOS X)X)
(- (1 / (SQRT (1 - (X ** 2)))))

(APPLY DERIV (ARCCOT X) X)
(- (1 / (1 + (X ** 2))))

(APPLY DERIV (ARCTAN X) X)
(1 / (1 + (X ** 2)))

(APPLY DERIV (ARCCOSEC X) X)
(- (1 / (X * (SQRT ((X ** 2) - 1)))))

(APPLY DERIV ((SIN X) * (LN X)) X)
(((COS X) * (LN X)) + ((SIN X) * (1 / X)))

(APPLY DERIV (EXP (SIN X)) X)
((EXP (SIN X)) * (COS X))
```

```
(APPLY DERIV (SQRT (SIN (2 * X))) X)
(((COS (2 * X)) * 2) / (2 * (SQRT (SIN (2 * X)))))

(APPLY DERIV (X * (COS X)) X)
((COS X) + (X * (- (SIN X))))

(APPLY DERIV (LN (SQRT (COS (2 * X)))) X)
(((- ((SIN (2 * X)) * 2)) / (2 * (SQRT (COS (2 * X))))) / (SQRT (COS (2
* X))))

(APPLY DERIV (SEC (4 * X))X)
(((SEC (4 * X)) * (TAN (4 * X))) * 4)

(APPLY DERIV (LN (A * (X ** N))) X)
((A * (N * (X ** (N - 1)))) / (A * (X ** N)))

(APPLY DERIV (X * (LN X)) X)
((LN X) + (X * (1 / X)))

(APPLY DERIV (2 / (EXP X)) X)
((- (2 * (EXP X))) / ((EXP X) ** 2))

(APPLY DERIV (LN (X ** 3)) X)
((3 * (X ** 2)) / (X ** 3))

(APPLY DERIV (LN (X ** 5)) X)
((5 * (X ** 4)) / (X ** 5))

(APPLY DERIV ((SIN (2 * X)) * (COS X)) X)
((((COS (2 * X)) * 2) * (COS X)) + ((SIN (2 * X)) * (- (SIN X))))

(APPLY DERIV ((EXP (A * X)) * (SIN (B * X))) X)
((((EXP (A * X)) * A) * (SIN (B * X))) + ((EXP (A * X)) * ((COS (B * X)
* B)))

(APPLY DERIV (ARCSIN (SQRT X)) X)
((1 / (2 * (SQRT X))) / (SQRT (1 - ((SQRT X) ** 2))))

(APPLY DERIV ((X * 2)*(ARCCOS X)) X)
((2 * (ARCCOS X)) + ((X * 2) * (- (1 / (SQRT (1 - (X ** 2)))))))

(APPLY DERN (SQRT((A ** 2) + (V ** 2))) V 2)
(((2 * (2 * (SQRT ((A ** 2) + (V ** 2))))) - ((2 * V) * (2 * ((2 * V) /
(2 * (SQRT ((A ** 2) + (V ** 2)))))))) / ((2 * (SQRT ((A ** 2) + (V **
2)))) ** 2))

(APPLY DERN (TAN X) X 2)
((2 * (SEC X)) * ((SEC X) * (TAN X)))

(APPLY DERN ((SIN X ) / X ) X 2)
((((((- (SIN X)) * X) + (COS X)) - (COS X)) * (X ** 2)) - ((((COS X) *
X) - (SIN X)) * (2 * X))) / ((X ** 2) ** 2))

(APPLY DERN ((EXP T ) * (COS T )) T 2)
((((EXP T) * (COS T)) + ((EXP T) * (- (SIN T)))) + (((EXP T) * (- (SIN
T))) + ((EXP T) * (- (COS T)))))
```

* 15. DEBUGGING IN LISP *

THE MOST FREQUENT ERROR IN A LISP PROGRAM IS AN UNBALANCED PARENTHESIS. THE FIRST STEP IN DEBUGGING A SET OF FUNCTIONS SHOULD THEREFORE BE A CAREFUL CHECK OF ALL PARENTHESES. THE ERROR ROUTINE WILL DETECT AN UNPAIRED PARENTHESIS, BUT WILL NOT DETECT AN IMPROPERLY PLACED ONE.

AS A SECOND STEP, PARTICULARLY IF SOME OF THE FUNCTIONS ARE COMPLEX IN LOGICAL STRUCTURE, IT IS RECOMMENDED TO TRY OUT EACH FUNCTIONS INDIVIDUALLY. FIRST THOSE WHICH REQUIRE NO AUXILIARY FUNCTIONS IN THEIR DEFINITIONS SHOULD BE TESTED, AND IF THEY INVOLVE CONDS OR IFS, ALL POSSIBLE PATHS THROUGH THE FUNCTIONS SHOULD RECEIVE A TRY-OUT. BUILDING UP FROM TESTED FUNCTIONS STEP BY STEP, ONE FINALLY ARRIVE AT THE TESTS FOR THE COMPLETE SET.

* 15.1 THE SPECIAL FUNCTION PRINT *

SOMETIMES IT IS HELPFUL TO BE INFORMED OF INTERMEDIATE VALUES, FOR INSTANCE, IN A COMPLICATED RECURSION. THIS MAY BE ACHIEVED WITH A SPECIAL FUNCTION WITH THE FORM -

```
(PRINT  X)
```

IT HAS ONE ARGUMENT AND THE VALUE OF X IS PRINTED OUT BY THE OUTPUT ROUTINE. THIS MEANS THAT (PRINT X) HAS ITS VALUE THE VALUE OF X. A USEFUL TRICK IN DEBUGGING IS TO PLACE VARIABLES OF FUNCTIONS WHOSE VALUE MIGHT BE OF INTEREST AT INTERMEDIATE PRINTS ON SEPARATE CARDS, PLACED BETWEEN ONE CARD IN WHICH (PRINT IS PUNCHED AND ANOTHER WITH ONLY). THESE TWO CARDS CAN BE REMOVED LATER ON.

CARE MUST BE TAKEN IN THE USE OF PRINT SINCE TOO FREE A USE OF IT WILL GIVE RISE TO VERY LARGE QUANTITIES OF OUTPUT WHOSE UTILITY IS DOUBTFUL. USUALLY NOT MORE THAN ONE PRINT PER FUNCTION SHOULD BE USED.

* EXAMPLES *

```
        (DEFINE (A  (X Y)) )

(PRINT A)
.       (X Y Z)
.       (X Y Z)

(PRINT (QUOTE A))
.       A
.       A
```

* 15.2 ERROR MESSAGES *

ALMOST ALL THE ERROR MESSAGES OF THE LISP INTERPRETER HAVE BEEN MENTIONED IN THE PRECEEDING CHAPTERS. BELOW IS A LIST OF THESE MESSAGES AND OTHERS NOT YET MENTIONED.

```
ERROR... VARIABLE NAME HAS NO VALUE
ERROR... MULTIPLY OVERFLOW
ERROR... ADD/SUBTRACT OVERFLOW
ERROR... DIVISION BY ZERO
ERROR... NUMBER AS FUNCTION DEFINITION
ERROR... EMPTY LIST AS FUNCTION DEFINITION
ERROR... NO FUNCTION GIVEN IN PUT/PUTQ
ERROR... UNPAIRED ARGUMENT IN DEFINE/SETPROP
ERROR... NO TRUE PREDICATE IN COND
ERROR... UNPAIRED ARGUMENT OR VARIABLE IN LAMBDA
ERROR... PUSH DOWN LIST EXHAUSTED
ERROR... NO MORE VACUUM
```

OTHER MESSAGES ARE GIVEN WHEN SOME ERROR OCCURS IN THE USE OF FUNCTIONS SUCH CAR, CDR, CONS, COND, PUT/PUTQ, LET/LETPROP, DEFINE/SETPROP, AND APPLY/EVAL. THE DESCRIPTION OF THIS MESSAGES IS GIVEN IN EACH CHAPTER WHERE THESE FUNCTIONS WAS MENTIONED.

THE MESSAGE -

```
ERROR... PUSH DOWN LIST EXHAUSTED
```

INDICATES THAT THE PUSH-DOWN LIST IS FULL. THIS IS USUALLY CAUSED BY A RECURSION WHICH IS INFINITE. HOWEVER IT IS POSSIBLE TO WRITE A PERFECTLY VALID FUNCTION WITH AN EXCESSIVE RECURSION DEPTH, SUCH A FUNCTION SHOULD BE REWRITTEN SO AS TO USE LESS PUSH-DOWN DEPTH.

THE MESSAGE -

```
ERROR... NO MORE VACUUM
```

REFERS TO THE GARBAGE COLLECTOR WHICH IS A ROUTINE TO FIND SUFFICIENT FREE STORAGE TO ALLOW THE PROGRAM TO CARRY ON. AGAIN, WHEN THIS MESSAGE IS GIVEN, ITS CAUSE IS USUALLY A SIGN OF AN INFINITE RECURSION WHOSE TERMINAL CONDITION CANNOT BE REACHED, BUT MAY BE DUE TO A CORRECT FUNCTION WHICH REQUIRES TOO MANY ELEMENTS OFF THE VACUUM. THE REMEDY IS TO REWRITE THE FUNCTION.

THE PROCESSOR ENTERS AND LEAVES THE GARBAGE COLLECTOR AUTOMATICALLY AS REQUIRED. IT IS, HOWEVER, SOMETIMES USEFUL TO KNOW HOW MANY GARBAGE COLLECTIONS HAVE TAKEN PLACE. IF THIS IS DESIRED, TURN CONSOLE SWITCH 3 ON. FOR EACH GARBAGE COLLECTION THE LETTERS GC WILL BE TYPED.

* 16. OPERATING INSTRUCTIONS *

* 16.1 THE PROGRAMME *

INPUT IN LISP MAY BE ON CARDS, PAPER TAPE OR TYPEWRITER. WE MUST NOTE THAT EACH INPUT LINE IN CARDS HAS 72 CHARACTERS IN COLS. 1 TO 72. COLS. 73 - 80 MAY BE USED FOR IDENTIFICATION. THERE IS NO PARTICULAR FORMAT TO BE OBSERVED, SINCE THE PARENTHESES WILL INDICATE EVERYTHING THAT IS NEEDED TO THE PROCESSOR. HENCE, A PROGRAMME MAY BE PUNCHED WITH AS MANY OR AS FEW SYMBOLS PER INPUT LINE AS DESIRED - INDENTING SUCESSIVE LINES IN ACCORD WITH THE LOGICAL STRUCTURE, AS HAS BEEN DONE IN MANY OF THE EXAMPLES GIVEN IN EARLIER CHAPTERS, HELPS TOWARDS UNDERSTANDING A PROGRAM. BLANK CARDS MAY BE INTERSPESED TO MAKE A READABLE LISTING.

FOR DEBUGGING PURPOSES IT IS IN GENERAL USEFUL TO PUNCH THE FUNCTIONS WITH FEW SYMBOLS PER CARD, INDENTED SO AS TO MAKE A READABLE LISTING.

ONCE THEY WORK, THEY MAY BE CONDENSED INTO A MINIMUM OF CARD SPACE BY PLACING THEM WITHIN A SINGLE DEFINE STATEMENT AND GIVING THIS AS ARGUMENT TO A FUNCTION.

```
(DEFINE
(CONDENSE (LAMBDA (X)  X))
)
```

WHICH WILL ELIMINATE ALL SUPERFLUOUS BLANK COLUMNS AN RE-NUMBER THE CARDS IN SEQUENCE.

* 16.2 LOADING THE PROCESSOR *

THE LISP INTERPRETER IS COMPOSED OF TWO DECKS. THE FIRST ONE IS THE MAIN PROGRAM CALLED LISP WHOSE OBJECT CARDS MUST BE LOADED ON DISKS USING THE FOLLOWING CARDS -

```
*DLOADLISP                          10000100900I
```

THIS PROGRAM WILL BE LOADED IN ANY AVAILABLE SECTOR OF DISK STORAGE.

THE SECOND PART OF THE DECK ARE THE LISP SUBROUTINES. THESE SUBROUTINES MUST BE LOADED IN SECTOR 15200 TO 15381 OF DISK UNIT 0. THIS IS CILINDER 76 AND MUST BE AVAILABLE FOR TO LOAD THE SUBROUTINES WHOSE NAME IS LISPSR.

THE FOLLOWING CARDS ARE USED TO LOAD THEM -

```
*DLOADLISPSR                  115200          CI
```

* 16.3 CALLING THE LISP INTERPRETER *

TO EXECUTE A LISP PROGRAM WE MUST TO LOAD THE LISP INTERPRE - TER FROM DISCS, WITH THE FOLLOWING CONTROL CARDS -

THE COMPUTER WILL PUT THE LISP INTERPRETER ON THE CORE STORA- GE AND BEGINS THE EXECUTION WRITING THE FOLLOWING MESSAGES -

LISP INTERPRETER
READ FROM

THE CONSOLE TYPEWRITER WILL WAIT ONE OF THE FOLLOWING ALTERNA TIVES -

CODE	MEANING
TY	CONSOLE TYPEWRITER
TY+	CONSOLE TYPEWRITER WITH PRINTING
CD	PUNCHED CARDS
CD+	PUNCHED CARDS WITH PRINTING
PT	PAPER TAPE
EX	EXIT
IN	INICIALIZATION OF LISP

AFTER ONE OF THEM IS TYPED WE ONLY CAN CHANGE THE INPUT DEVI- CE BY SETTING SWITCH 1 ON. THEN, THE MESSAGE -

READ FROM

IS TYPED AGAIN AND THE TYPEWRITER WAITS FOR ANOTHER INPUT DEVICE. IF ONE OF THOSE INPUT CODES ARE INCORRECTLY TYPED THE MESSAGE -

TRY AGAIN

IS TYPED AND THE MACHINE WAITS FOR ANOTHER CODE.

AFTER THE EVALUATION OF A LISP PROGRAM, THE MACHINE WILL ASK FOR THE OUTPUT DEVICE WRITTING THE MESSAGE

OUTPUT ON

ONE OF THE ALTERNATIVES EARLIER DEFINED CAN BE CHOOSEN AND ANOTHER ONE, TO ALLOW THE OUTPUT BE TYPED ON THE PRINTER, THE CODE FOR PRINTER IS PR.

DURING THE INPUT BY TYPEWRITER, IF SOME STATEMENT IS INCORRECT WE CAN SET SWITCH 4 ON BEFORE R/S. THAT STATEMENT WILL BE REJECTED AND THE MESSAGE -

SW4 ON. REPEAT READ IN.

IS TYPED.

THE INPUT ROUTINE OF THE LISP INTERPRETER WILL IGNORE THE BLANK SPACES IN THE PROGRAM WHEN THE RIGHT PARENTHESES CORRESPONDING TO THE FIRST LEFT PARENTHESES IS FOUND THE EXECUTION WILL START AND WHAT COMES AFTER THAT PARENTHESES IS IGNORED.

WHEN A READ CHECK OCCURS, THE MACHINE SEND THE MESSAGE -

READ CHECK. PRESS START TO RE-READ

PRESSING START THE MACHINE DELETES THE LAST STATEMENT AND TRY A NEW READ.

AT TIME OF EXECUTION OF A LISP PROGRAM IF SWITCH 3 IS ON, THE INTERPRETER WILL NOT SEND ANY ERROR MESSAGE AND THE EXECUTION IS FASTER. IT IS CONVENIENT TO LEAVE SWITCH 3 OFF WHEN THE PROGRAM STILL HAVE ERRORS BUT, WHEN ALL IS CORRECT, SWITCH 3 MAY BE ON.

17. LISP EXERCICES

IN THIS CHAPTER WE WILL PRESENT SOME SIMPLE PROBLEMS AND A WAY TO SOLVE THEM. OF COURSE, THE ANSWER GIVEN IS ONLY ONE OF THE WAYS TO SOLVE THAT SPECIFIC PROBLEM AND YOU PROBABLY MAY FIND A BETTER SOLUTION.

1)

```
(LST X)
```

ARGUMENT - X AN ATOM OR A LIST.

THIS IS A PREDICATE THAT WILL BE TRUE IF THE ARGUMENT X IS A LIST AND FALSE OTHERWISE.

```
(LST(LAMBDA(X) (NOT (ATOM X))))
```

APPLICATIONS

```
(APPLY LST A)
.       F
(APPLY LST (A B))
.       T
(APPLY LST ())
.       T
```

2)

```
(ONELEM L)
```

ARGUMENT - L AN ATOM OR A LIST.

THIS IS A PREDICATE THAT WILL BE TRUE IF A GIVEN LIST HAS AT LEAST ONE ELEMENT. OTHERWISE IT WILL BE FALSE.

```
(ONELEM(LAMBDA(L)
        (NOT (OR (ATOM L)(NULL L)) ) ))
```

APPLICATIONS

```
(APPLY ONELEM (A B) )
.       T
(APPLY ONELEM A)
.       F
(APPLY ONELEM ())
.       F
(APPLY ONELEM (A) )
.       T
```

3)

```
        (TWOLEM L)
```

ARGUMENT - L AN ATOM OR A LIST.
****

THIS IS A PREDICATE THAT WILL BE TRUE IF A GIVEN ARGUMENT IS A LIST AND HAS AT LEAST TWO ELEMENTS. OTHERWISE THE PREDICATE WILL BE FALSE.

```
        (TWOLEM(LAMBDA(L)
               (NOT  (OR    (ATOM L)
                            (NULL L)
                            (NULL (CDR L)) ) )  ))
```

APLICATIONS
****

```
(APPLY TWOLEM () )
.       F
(APPLY TWOLEM A)
.       F
(APPLY TWOLEM (A B) )
.       T
(APPLY TWOLEM (A) )
.       F
(APPLY TWOLEM (A B C D) )
.       T
```

4)

```
        (SINGLE L)
```

ARGUMENT - L A LIST OR AN ATOM.
****

THIS IS A PREDICATE THAT WILL BE TRUE IF THE GIVEN ARGUMENT IS A LIST AND HAS EXACTLY ONE ELEMENT, OTHERWISE IT WILL BE FALSE. SINGLE USES THE PREDICATE ONELEM.

```
        (SINGLE(LAMBDA(L)
               (AND  (ONELEM L)
                     (NULL (CDR L))  )  ))
```

APLICATIONS
****

```
(APPLY SINGLE A)
.       F
(APPLY SINGLE () )
.       F
(APPLY SINGLE (A) )
.       T
(APPLY SINGLE (A B) )
.       F
```

5)

```
(ODD L)
```

ARGUMENT - L A LIST.
****

THIS IS A PREDICATE THAT WILL BE TRUE IF THE GIVEN LIST (L HAS AN ODD NUMBER OF ELEMENTS AND FALSE OTHERWISE.

```
(ODD (LAMBDA(L)
     (NOT  (OR       (NULL L)
                     (ODD (CDR L) )  ) )   ))
```

APLICATIONS
****

```
(APPLY ODD (A B C) )
.       T
(APPLY ODD (A B) )
.       F
(APPLY ODD ( ( (A) B C) D E)  )
.       T
```

6)

```
(MEMBERG A L)
```

ARGUMENTS - A AN ATOM OR A LIST.
********* L A LIST.

THIS IS A PREDICATE THAT WILL TAKE THE VALUE TRUE IF THE ELEMENT A IS MEMBER OF THE LIST L. OTHERWISE THE PREDICATE RETURNS THE VALUE FALSE. MEMBERG USES THE PREDICATE EQUAL.

```
(MEMBERG(LAMBDA(A L) (COND
        ((NULL L) (OR) )
        ((EQUAL A (CAR L) ) (AND) )
        ((AND) (MEMBERG A (CDR L)) ) )  )))
```

APLICATIONS
****

```
(APPLY MEMBERG F ( (A (D R))F G)  )
.       T
(APPLY MEMBERG (F) ( (A (D R) ) F G) )
.       F
(APPLY MEMBERG () ((A)()) )
.       T
(APPLY MEMBERG (A B) (A B C) )
.       F
(APPLY MEMBERG (A) ((C (A) ) D) )
.       F
(APPLY MEMBERG () (()))
.       T
```

7)

```
        (SAMELEM L1 L2)
```

ARGUMENTS - L1,L2 LISTS.
****

THIS IS A PREDICATE THAT WILL BE TRUE IF ALL THE ELEMENTS OF THE LIST L1 ARE ELEMENTS OF THE LIST L2, OTHERWISE THE PREDICATE WILL RETURNS A VALUE FALSE. SAMELEM USES THE PREDICATE MEMBERG,

```
        (SAMELEM(LAMBDA(L1 L2)(COND
                ((NULL L1) (AND) )
                ((MEMBERG (CAR L1) L2) (SAMELEM (CDR L1) L2) )
                ((AND) (OR) )   )))
```

APLICATIONS
****

```
(APPLY SAMELEM (A B C) (B C A) )
.       T
(APPLY SAMELEM (X Y (Z K)) (A X (Z K) Y) )
.       T
(APPLY SAMELEM (A (B) C) (B C A) )
.       F
(APPLY SAMELEM (A A A A) (A) )
.       T
(APPLY SAMELEM (F) (A B (F) C) )
.       F
```

8)

```
        (FORALL P L)
```

ARGUMENTS - L A LIST.
********* P A PREDICATE.

THIS IS A PREDICATE THAT WILL BE TRUE IF THE GIVEN PREDICATE P IS TRUE FOR ALL THE ELEMENTS OF A GIVEN LIST AND FALSE OTHERWISE.

```
        (FORALL(LAMBDA(P L)(OR
               (NULL L)  (AND (P (CAR L) )
                              (FORALL P (CDR L) ) ) ) ))
```

APLICATIONS
****

```
(APPLY FORALL ATOM (A B C) )
.       T
(APPLY FORALL NULL ( () ()) )
.       T
(APPLY FORALL ATOM ( (A B C) D) )
.       F
(APPLY FORALL ODD ( (A B C) (A) (A A A A A) ) )
.       T
(APPLY FORALL ODD ( (A) (B) (C D)) )
.       F
```

9)

```
        (FORSOME P L)
```

ARGUMENTS - P A PREDICATE.
********* L A LIST.

THIS IS A PREDICATE THAT WILL TAKE THE VALUE TRUE IF A GIVEN PREDICATE P IS TRUE FOR ANY ELEMENT OF THE GIVEN LIST L, OTHERWISE IT WILL BE FALSE.

```
        (FORSOME(LAMBDA(P L)(AND
                (NOT  (NULL L))
                (OR   (P (CAR L) )
                      (FORSOME P (CDR L)) )) ))
```

APLICATIONS
****

```
(APPLY FORSOME NULL (A B () ) )
.       T
(APPLY FORSOME ATOM ((A)() A))
.       T
(APPLY FORSOME ATOM ( ()) )
.       F
(APPLY FORSOME ODD ( (A) (A B) ) )
.       T
(APPLY FORSOME ODD ( (A B) (C D E) ) )
.       F
(APPLY FORSOME ODD ( () () ) )
.       T
```

10)

```
        (ORDER X Y L)
```

ARGUMENTS - X,Y,L LISTS.
****

THIS IS A PREDICATE THAT WILL BE TRUE IF THE ELEMENT X IS ON THE LIST L AND THE ELEMENT Y IS NOT - OR IF BOTH ARE ON THE LIST L AND X COMES BEFORE Y. FALSE OTHERWISE. ORDER USES THE PREDICATE EQUAL.

```
        (ORDER(LAMBDA(X Y L)(COND
              ((NULL L) (OR) )
              ((EQUAL X (CAR L) )  (AND) )
              ((EQUAL Y (CAR L) )  (OR) )
              ((AND) (ORDER X Y (CDR L)) )  ) ) )
```

APLICATIONS
****

```
(APPLY ORDER B A (A L E) )
.       F
(APPLY ORDER A B ( A L E) )
.       T
(APPLY ORDER A B (B A L E) )
.       F
(APPLY ORDER L I (H I J K L) )
.       F
(APPLY ORDER F G (E F G H) )
.       T
```

11)

```
        (EQUIV X Y L)
```

ARGUMENTS - X,Y,L - LISTS.
****

THIS PREDICATE TAKES THE VALUE TRUE IF THE TWO GIVEN ELEMENTS X AND Y ARE CONTAINED IN THE SAME SUB-LIST OF THE GIVEN LIST L, OTHERWISE FALSE. EQUIV USES THE PREDICATE MEMBERG.

```
        (EQUIV(LAMBDA(X Y L)(COND
                ((NULL L) (OR) )
                ((MEMBERG X (CAR L))(MEMBERG Y (CAR L)))
                ((AND) (EQUIV X Y (CDR L))) )))
```

APLICATIONS
****

```
(APPLY EQUIV (A)(B)((A C)(B D)(B C)(A B)))
.       F
(APPLY EQUIV A B ( (F G) (A B)))
.       T
(APPLY EQUIV A B ((A G)(A B)))
.       F
(APPLY EQUIV A B ((A B)))
.       T
```

12)

```
        (MAPCAR F L)
```

ARGUMENTS - F A FUNCTION
********** L A LIST

THIS FUNCTION APPLIES A GIVEN FUNCTION F TO EVERY ELEMENT OF A LIST L AND LISTS THE RESULTS.

```
        (MAPCAR(LAMBDA(F L)(COND
                ((NULL L)(LIST))
                ((AND)(CONS (F (CAR L))(MAPCAR F (CDR L) ))) )))
```

APLICATIONS
****

```
(APPLY MAPCAR CAR ((A B)(B C)(C D)(D E)(E F)))
.       (A B C D E)
(APPLY MAPCAR CDR ((A B)(B C)(C D)(D E)(E F)))
.       ((B)(C)(D)(E)(F))
(APPLY MAPCAR CADR ((A B)(B C)(C D)(D E)(E F)))
.       (B C D E F)
(APPLY MAPCAR ATOM ((A) B X (V U L)))
.       (F T T F)
```

13)

```
        (MAPCON F L)
```

ARGUMENTS - F A FUNCTION
********* L A LIST

THIS FUNCTION APPLIES A GIVEN FUNCTION F TO A WHOLE LIST THEN TO CDR OF THE LIST, AND SO ON. THE LISTS RESULTING ARE JOINED TOGETHER.

```
        (MAPCON(LAMBDA(F L)(COND
              ((NULL L)(LIST))
              ((AND)(APPEND (F L)(MAPCON F (CDR L))))  )))
```

APLICATIONS
****

```
(APPLY MAPCON CDR (A B C D) )
.       (B C D C D D)
(APPLY MAPCON LIST (A B C D))
.       ((A B C D)(B C D)(C D)(D))
```

14)

```
        (MAPLIST F L)
```

ARGUMENTS - F A FUNCTION
********* L A LIST

THIS FUNCTION APPLIES A GIVEN FUNCTION F TO A LIST, THEN TO CDR OF THE LIST, AND SO ON. THE RESULTS OF EACH APPLICATION FORM THE ELEMENTS OF A NEW LIST.

```
        (MAPLIST(LAMBDA(F L)(COND
              ((NULL L)(LIST))
              ((AND)(CONS(F L)(MAPLIST F (CDR L))))  )))
```

APLICATIONS
****

```
(APPLY MAPLIST ATOM (A B C D))
       (F F F F)
(APPLY MAPLIST CAR (A B C D)
.       (A B C D)
(APPLY MAPLIST CDR (A B C D))
.       ((B C D)(C D)(D)())
(APPLY MAPLIST LIST (A B C D))
.       (((A B C D))((B C D))((C D))((D)))
```

15)

```
        (FOREACH G X L)
```

ARGUMENTS - G A FUNCTION
********** L A LIST
X AN ATOM OR A LIST

FOREACH APPLIES A FUNCTION G OF TWO ARGUMENTS TO THE ELEMENTS OF A LIST L AND TO X AND LISTS THE RESULTS.

```
        (FOREACH(LAMBDA(G L X)(COND
                ((NULL L)(LIST))
                ((AND)(CONS(G (CAR L) X)
                            (FOREACH G (CDR L) X))) )))
```

APLICATIONS
****

```
(APPLY FOREACH CONS (A B C) (D) )
.       ((A D)(B D)(C D))
(APPLY FOREACH LIST (A B C) (D) )
.       ((A (D))(B (D))(C (D)))
(APPLY FOREACH MEMBERG (B R A S I L) (U S A))
.       (F F T T F F)
```

16)

```
        (NOTPOSS P L)
```

ARGUMENTS - P A PREDICATE
********** L A LIST

THIS FUNCTION WRITES A LIST OF ALL THE ELEMENTS OF A GIVEN LIST FOR WHICH A GIVEN PREDICATE IS FALSE.

```
        (NOTPOSS(LAMBDA(P L)(COND
                ((NULL L)(LIST))
                ((NOT (P (CAR L))) (CONS(CAR L)
                                         (NOTPOSS P (CDR L))))
                ((AND)(NOTPOSS P (CDR L))) )))
```

APLICATIONS
****

```
(APPLY NOTPOSS NULL (A () B C () D ()))
.       (A B C D)
(APPLY NOTPOSS ATOM (A B (C) () (D) E () F G)) )
.       ((C)()(D)())
(APPLY NOTPOSS (LAMBDA(X)(NOT (ATOM (X))) (A B (C) (D) E F)))
.       (A B E F)
```

17)

(REMOVE X L)

ARGUMENTS - X AN ATOM OR A LIST
********* L A LTST.

THIS FUNCTION WRITES A LIST OF ALL THE ELEMENTS ON A GIVEN. LIST EXCEPT ALL THOSE EQUAL TO A GIVEN ELEMENT.

```
(REMOVE(LAMBDA(X L)(COND
        ((NULL L)(LIST))
        ((EQUAL X (CAR L))(REMOVE X (CDR L)))
        ((AND)(CONS (CAR L)(REMOVE X (CDR L))))))
```

APLICATIONS
****

```
(APPLY REMOVE A (F D A) )
.       (F D)
(APPLY REMOVE (A B C) (A B C (D F (A b C))) )
.       (A B C (D F (A B C)))
(APPLY REMOVE (A B C)(A (B C)(D)(A B C)))
.       (A (B C)(D))
```

18)

(REALTER L M)

ARGUNENTS - M AN ATOM OR A LIST
********* L A LIST

THIS FUNCTION PLACES THE GIVEN LIST ELEMENT M AFTER EACH ELEMENT OF A GIVEN LIST L.

```
(REALTER(LAMBDA(L M)(COND
        ((NULL L)(LIST))
        ((NULL (CDR L))L)
        ((AND)(CONS(CAR L)(CONS M(REALTER (CDR L)M)))) )))
```

APLICATIONS
****

```
(APPLY REALTER (C R V N S) A)
.       (C A R A V A N A S)
(APPLY REALTER (B B S) O)
.       (B O B O S)
(APPLY REALTER (1 4) 5)
.       (1 5 4)
```

19)

(SUBS X Y L)

ARGUMENTS - X,Y AN ATOM OR A LIST.
********* L A LIST.

THIS FUNCTION REPLACES EVERY OCURRENCE OF ONE GIVEN ELEMENT Y ON A GIVEN LIST L BY A SECOND GIVEN ELEMENT X.

```
(SUBS(LAMBDA(X Y L)(COND
     ((NULL L)(LIST))
     ((EQUAL Y (CAR L)) (CONS X (SUBS X Y (CDR L))) )
     ((AND)(CONS (CAR L)(SUBS X Y (CDR L)))) )))
```

APLICATIONS
****

```
(APPLY SUBS A G (G G N G N G S).)
     (B A N A N A S)
```

20)

(AMONG X L)

ARGUMENTS - X AN ATOM OR A LIST.
********* L A LIST

THIS FUNCTION ADDS THE GIVEN LIST ELEMENT X TO THE GIVEN LIST L IF IT IS NOT ON THE LIST L.

```
(AMONG(LAMBDA(X L)(COND
      ((NULL L)(LIST X))
      ((EQUAL X (CAR L))L)
      ((AND)(CONS(CAR L)(AMONG X (CDR L)))) )))
```

APLICATIONS
****

```
(APPLY AMONG T (MUS) )
.       (M U S T)
(APPLY AMONG C (C O M M))
.       (C O M M)
```

21)

```
        (INDEX X L)
```

ARGUMENTS - X AN ATOM OR A LIST
******** L A LIST

THIS FUNCTION LISTS THE ELEMENTS OF THE GIVEN LIST L UP TO THE FIRST OCURRENCE OF THE GIVEN LIST ELEMENT X.

```
        (INDEX(LAMBDA(X L)(COND
              ((NULL L)(LIST))
              ((EQUAL X (CAR L))(LIST))
              ((AND)(CONS(CAR L)(INDEX X (CDR L)))) )))
```

APLICATIONS
****

```
(APPLY INDEX A (X Y Z A T) )
.       (X Y Z)
(APPLY INDEX A (A B C) )
.       ()
(APPLY INDEX A (S D A) )
        (S D)
```

22)

```
        (UNION M N)
```

ARGUMENTS - M,N LISTS.
****

THIS FUNCTION WILL PRODUCE A LIST OF ALL THE ATOMS WHICH ARE IN EITHER OF TWO LISTS. UNION USES MEMBER.

```
        (UNION(LAMBDA(M N)(COND
              ((NULL M)N)
              ((MEMBER (CAR M) N)(UNION (CDR M) N))
              ((AND)(CONS (CAR M) (UNION (CDR M) N))) )))
```

APLICATIONS
****

```
(APPLY UNION (U V W) (W X Y) )
.       (U V W X Y)
(APPLY UNION (A B C) (B C D) )
.       (A B C D)
(APPLY UNION (A B CD E X Y Z) (B C G H W V Y Z) )
.       (A D E X B C G H W V Y Z)
```

23)

```
        (INTERSECTION M N)
```

ARGUMENTS - -.M,N LISTS.
****

THIS FUNCTION WILL PRODUCE A LIST OF ALL THE ATOMS THAT ARE COMMON TO TWO LISTS. INTERSECTION USES MEMBER.

```
        (INTERSECTION(LAMBDA(M N)(COND
                    ((NULL M) (LIST) )
                    ((MEMBER (CAR M) N) (CONS  (CAR M)
                                               (INTERSECTION (CDR M) N)))
                    ((AND)(INTERSECTION (CDR M) N)) )))
```

APLICATIONS
****

```
(APPLY INTERSECTION (A B C) (B C D) )
.       (B C)
(APPLY INTERSECTION (A B C) (A B C) )
.       (A B C)
(APPLY INTERSECTION (A-B C)(D E F) )
.       ()
(APPLY INTERSECTION (U V W)(W X Y) )
.       (W)
(APPLY INTERSECTION (A B C D E X Y Z) (B C G H W V Y Z) )
.       (B C Y Z)
```

24)

```
        (SUPREVERSE M)
```

ARGUMENT - M A LIST.
****

THIS FUNCTION WILL REVERSE ALL THE LEVELS OF LIST. IT USES THE FUNCTION APPEND.

```
        (SUPREVERSE(LAMBDA(M)(COND
                    ((ATOM M) M)
                    ((NULL M) (LIST) )
                    ((AND) (APPEND (SUPREVERSE (CDR M))
                                   (LIST(SUPREVERSE (CAR M))) ))  )))
```

APLICATIONS
****

```
(APPLY SUPREVERSE (A B (C D)) )
.       ((D C) B A)
(APPLY SUPREVERSE ((U X)((X Z) Y)) )
.       ((Y (Z X))(X U))
(APPLY SUPREVERSE ((A B C (D (E ((F G) H) I) J) K) L M (N O P) Q R) )
.       (R Q (P O N) M L (K (J (I (H (G F)) E) D) C B A))
```

25)

```
          (LISTATOM S)
```

ARGUMENT - S A LIST.
****

THIS FUNCTION PRODUCES A LIST OF ALL DIFFERENT ATOMS OF A GIVEN LIST S. LISTATOM USES UNION.

```
          (LISTATOM(LAMBDA(S)(COND
                    ((NULL S)(LIST))
                    ((ATOM S)(LIST S))
                    ((AND)(UNION (LISTATOM(CAR S)) (LISTATOM(CDR S)) )) )))
```

APLICATIONS
****

```
(APPLY LISTATOM (A (B C) D))
.         (A B C D)
(APPLY LISTATOM ((A)(B C)(A B D)(F G E)))
.         (C A B D F G E)
(APPLY LISTATOM ((((A))B)))
.         (A B)
```

26)

```
          (LENGTH X)
```

ARGUMENT - X AN ATOM OR A LIST.
****

THIS FUNCTION GIVES AS RESULT THE NUMBER OF ELEMENTS OF THE ARGUMENT X. IF X IS AN EMPTY LIST OR AN ATOM THE ANSWER WILL BE ZERO.

```
          (LENGTH(LAMBDA(X) (IF
                 (OR (NULL X) (ATOM X) )
                           0
                           (ADD 1 (LENGTH(CDR X)))  )))
```

APLICATIONS
****

```
(APPLY LENGTH ())
.         0
(APPLY LENGTH A)
.         0
(APPLY LENGTH (A B C D))
.         4
(APPLY LENGTH (A (B C)(D E)) )
.         3
```

27)

```
        (REMAINDER Y X)
```

ARGUMENTS - Y,X NUMERICAL ATOMS.
****

THIS FUNCTION GIVES THE REMAINDER OF THE INTEGER DIVISION OF Y BY X.

```
        (REMAINDER(LAMBDA(Y X)(COND
                ((EQ Y X) 0)
                ((SG X Y) Y)
                ((AND)(REMAINDER (SUB Y X) X)) )))
```

APLICATIONS
****

```
(APPLY REMAINDER 2 2)
.       0
(APPLY REMAINDER 364 14)
.       0
(APPLY REMAINDER 1523 10)
.       3
(APPLY REMAINDER 261 2)
.       1
(APPLY REMAINDER 85 24)
.       13
```

28)

```
        (GCD X Y)
```

ARGUMENTS - X,Y NUMERICAL ATOMS.
****

THIS FUNCTIONS GIVES AS RESULT THE GREATEST COMMON DIVISOR BETWEEN X AND Y. GCD USES REMAINDER.

```
        (GCD(LAMBDA(X Y)(COND
            ((SG X Y)(GCD Y X))
            ((EQ (REMAINDER Y X) 0) X)
            ((AND)(GCD X (REMAINDER Y X))) )))
```

APLICATIONS
****

```
(APPLY GCD 4 12)
.       4
(APPLY GCD 243 19)
.       1
(APPLY GCD 39 9)
.       3
(APPLY GCD 28 7)
.       7
```

29)

```
          (SMALL X Y)
```

ARGUMENTS - X,Y NUMERICAL ATOMS.
****

THIS IS AN ARITHMETIC FUNCTION THAT COMPARES X AND Y GIVING AS RESULT THE SMALLER NUMBER.

```
          (SMALL(LAMBDA(X Y)(IF
                (SG X Y)   Y   X)  ))
```

APLICATIONS
****

```
(APPLY SMALL 12 5)
.         5
(APPLY SMALL 2 2)
.         2
(APPLY SMALL 0 123)
.         0
```

30)

```
          (BIN X)
```

ARGUMENT - X, A NUMERICAL ATOM.
****

THE VALUE OF THIS FUNCTION IS THE BINARY EQUIVALENT OF THE ATOMIC SYMBOL X WHICH IS SUPPOSED TO BE A DECIMAL NUMBER OF NO MORE THAN FOUR DIGITS.

```
          (BIN(LAMBDA(X)(IF   (EQ (DIV X 2) 1)
                         (LIST 1 (REMAINDER X 2))
                         (APPEND (BIN (DIV X 2))(LIST (REMAINDER X 2))) )))
```

APLICATIONS
****

```
(APPLY BIN 11)
.         (1 0 1 1)
(APPLY BIN 33)
.         (1 0 0 0 0 1)
(APPLY BIN 158)
.         (1 0 0 1 1 1 1 0)
(APPLY BIN 386)
.         (1 1 0 0 0 0 0 1 0)
```

31)

```
        (MIN X)
```

ARGUMENT - X A LIST WHOSE ELEMENTS ARE NUMBERS.

****

THIS FUNCTION GIVES AS RESULT THE MINIMUM OF ALL THE ELEMENTS OF A LIST. MIN USES THE FUNCTION SMALL.

```
        (MIN(LAMBDA(X)(COND
            ((NULL X)(LIST))
            ((NULL (CDR X))(CAR X))
            ((AND)(SMALL(CAR X)(MIN(CDR X))))))
```

APLICATIONS

****

```
(APPLY MIN (12 11 3 5 4 1) )
.       1
(APPLY MIN (12 258 0 145) )
.       0
(APPLY MIN (145 24 15 12 65) )
.       12
```

32)

```
        (DEC L)
```

ARGUMENT - X A LIST.

****

THE VALUE OF THIS FUNCTION IS THE DECIMAL EQUIVALENT OF THE BINARY NUMBER CONTAINED IN THE LIST L. DEC USES POT AND LENGTH.

```
        (DEC(LAMBDA(L)(IF   (NULL L)
                        0
                        (ADD (MULT (CAR L)(POT 2 (SUB(LENGTH L) 1)))
                             (DEC (CDR L)) )) ))
```

APLICATIONS

****

```
(APPLY DEC (1 1 1))
.       7
(APPLY DEC (1 0 1 0 1))
.       21
(APPLY DEC (1 1 1 1 1))
.       31
(APPLY DEC (1 0 1 1 1 1 1 0 1))
.       381
(APPLY DEC (1 1 1 0 0 1 1 0 1 1 1 1))
.       3695
(APPLY DEC (1 1 0 0 0 0 0 1 0))
.       386
```

****
* REFERENCES *
****

J. MCCARTHY
RECURSIVE FUNCTIONS OF SYMBOLIC EXPRESSIONS AND THEIR COMPUTATION BY MA
CHINE - COMM. ACM, 3 (1960), 184.

J. MCCARTHY
LISP 1.5, PROGRAMMERS MANUAL
MIT PRESS,1962.

MARTIN DAVIS
COMPUTABILITY AND UNSOLVABILITY - MCGRAW-HILL, NEW YORK, 1958.

ALLEN NEWELL
INFORMATION PROCESSING LANGUAGE-V MANUAL - PRENTICE HALL, N.J. 1961.

ALONZO CHURCH
THE CALCULI OF LAMBDA CONVERSION / ANNALS OF MATHEMATICAL STUDIES NUM 6
PRINCETON UNIV. PRESS, N.J.

T.A.BRODY
SYMBOL MANIPULATION TECHNIQUES FOR PHYSICS
GORDON AND BREACH,1967.

CLARK WEISSMAN
LISP 1.5 PRIMER
DICKENSON PUB. INC. 1967.

E.C.BERKELEY AND D.G.BOBROW
THE PROGRAMMING LANGUAGE LISP
MIT PRESS

A.K.GRIFFITH
AN INTRODUCTION TO LISP - QUANTUM THEORY PROJECT, UNIV. OF FLORIDA.

H.V. MCINTOSH
A HANDBOOK OF LISP FUNCTIONS - RIAS TECHNICAL REPORT 61-11 BALTIMORE.

INDEX

INDEX

INDEX