SLIP - A Symmetric List Processor

SLIP is a language developed by Prof. Joseph Weizenbaum of M.I.T. It is a set of subroutines designed to be embedded in a higher order language. In the implementation described herein, SLIP is embedded in MAD. The University of Michigan's version of SLIP was obtained from the Yale Computing Center, and the following writeup is a direct reproduction of their writeup (Memorandum N° 62 L, dated May 3, 1965). This writeup constitutes the most up-to-date description of SLIP available, but is not guaranteed to be an accurate description of SLIP at other installations. (For the original description of SLIP as embedded in FORTRAN, see Communications of the ACM, 1963, 6, 524-544.)

Digitized by Google

Table of Contents

		Page
1.0	Introduction	1
2.0	Some Internal Details of SLIP	1
2.1	SLIP Storage	1
2.2	Available Space	3
2.3	SLIP Embedded in MAD	4
2.4	Definition of SLIP Arguement and Function Classes	5
3.0	The Functions	6
3.1	Creating Available Space	6
3.2	Creating and Copying Lists	6
3.3	Placing Information On Lists	7
3.4	Retrieving Information From a List	10
3.5	Deleting Lists and List Cells	11
3.6	Functions Which Obtain the Address of a List Cell	12
3.7	Tests	13
3.8	The Sequence Reader Mechanism For Linear Lists	13
3.9	The Advance Reader Mechanism for Lists and List Structures	16
3.10	List Marking Functions	22
3.11	Description Lists	23
3.12	Input-Output Functions	24
3.13	String and Text Manipulating Functions	25
3.14	Functions Which Operate Upon the Information Contained in Specific SLIP Cell Fields	27
3.15	Miscellaneous Functions	30
4.0	Index of Functions	31

Digitized by Google

1.0 Introduction

List processing programming languages (e.g., LISP, IPL-V, FLPL) may be characterized as providing a means for manipulating information in ordered sets. It is the programmer's responsibility to decide if his application is most easily programmed in such a language. However, the fact that SLIP is embedded in a general purpose compiler provides the programmer with the opportunity to combine the best features of each type of language.

The distinguishing features of SLIP are (1) that each list cell carries both a forward and a backward link, and (2) that each cell may contain an actual datum rather than a symbolic designation for a datum. SLIP is symmetric in that its lists do not have a preferred orientation. Any operation which can be carried out on the top element of a list can just as easily be performed on the bottom element.

2.0 Some Internal Details of SLIP

2.1 SLIP Storage

The fundamental information module with which SLIP deals is a word-pair. The first of the pair is divided into an identifier field, a link-left field and a link-right field. The second word usually contains a datum, but for certain uses is divided as is the first word. Thus:

ID	LNKL	LNKR
DATUM		

The codes in the ID field indicate the type of data term contained in the second word. They are as follows:

0: datum is not a name of a list;

1: datum is a name of a list;

2: word pair is a header of a list;

3: word pair is a reader.

Digitized by Google

Discussion of the reader is postponed to a later section.

A SLIP list is characterized by a direct linear string of SLIP cells such that the LNKR field of each cell points to (i.e., contains the machine address of) the next cell to the right (below) on the list, and its LNKL field points to the next cell to the left (above) on the list. On every list there is a single SLIP cell known as the header. The LNKR of the header points to the first cell on the list and the LNKL points to the bottom cell. Any cell containing the machine address of the header in both its LNKR and LNKL fields is said to contain the name of the list.

The header may not contain a data term. Instead the second word of the header cell word-pair may contain the name of a description list (or the value zero if there is no description list) in its LNKL field and always contains a reference counter in its LNKR field. This counter is an integer indicating the number of times the list appears as a sublist of other lists.

A SLIP list structure is defined recursively as a list containing data terms which themselves may be names of sublists. An example of a simple list structure consisting of a main list and one sublist (which has a description list) is given in Figure 1.

If Y is the programmer's designation of a cell such as

54321	54321

then Y is said to be the name of the list in Figure 1.

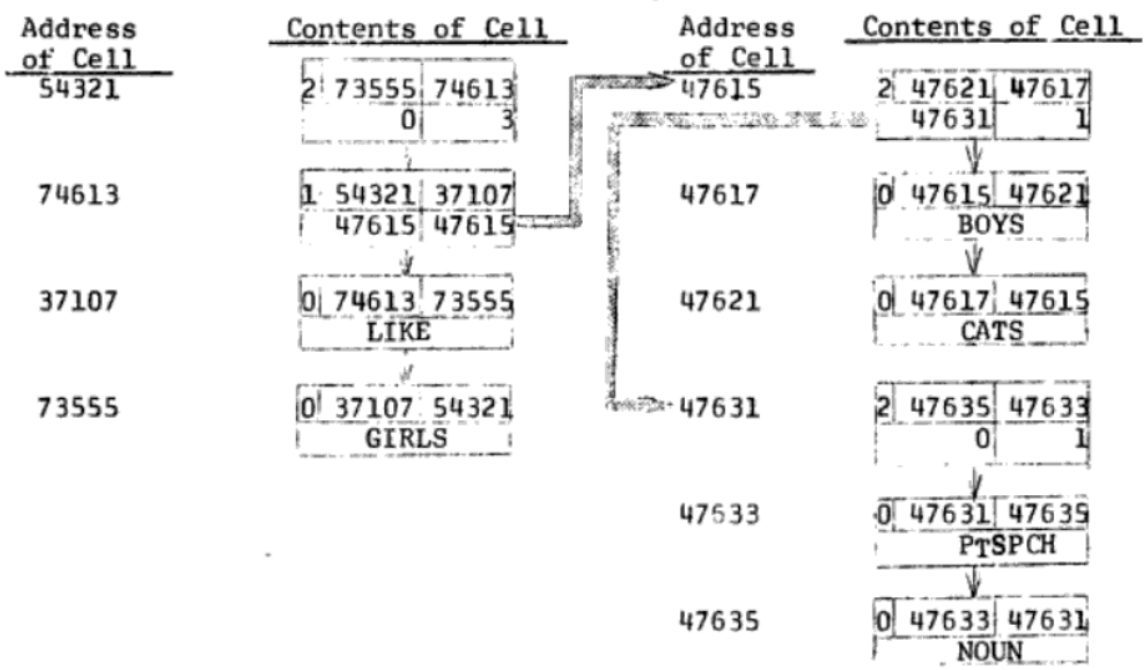

Figure 1.

Digitized by Google

2.2 Available Space

As in most list processing languages, SLIP uses a list of available space i.e., a list of (initially) empty SLIP cells comprising all of the machine storage not occupied by SLIP functions, UMES routines, or the programmer's own program. When lists are erased, they are returned to the bottom of the available space list. When lists are created, the necessary cells are taken from the top of available space. Most of this cell manipulation is carried out automatically by the various SLIP functions.

In one way or another every list processing system must face the question "who is entitled to erase a datum (i.e., to return it to available space) when a cell containing that datum is returned to available space?". In SLIP the issue arises only because (a) a list may be a sublist of several lists, and (b) SLIP stores data terms themselves in list cells rather than addresses of data terms. Thus, when a list is erased, a decision has to be made as to whether or not to erase its sublists also. If any of its sublists are also sublists of other lists not yet ready for erasure, then the erasure of such sublists can only lead to chaos.

SLIP employs the following solution to the problem. Each erased list is returned to available space by linking its top cell to the bottom cell of the available space list. The information contained in the erased list is not erased. Then, whenever a new cell is taken from the top of available space, its ID field is examined. If it is found to contain the name of a list (ID = 1), the header of that list is secured and its reference counter is decremented by one. If the new count is zero the list is returned to available space. Otherwise, the list is left as is.

The SLIP erasure technique clearly implies that during the running of the program certain lists will disappear without any direct command from the programmer. However, the programmer will wish to construct many lists, the names of which he knows explicitly, i.e., to which he has given some symbolic designation. He will not want these lists to disappear when he stacks and unstacks their names for, say, temporary storage considerations. The function LIST. (to be described in a later section) provides a mechanism for resolving this problem.

Digitized by Google

2.3 SLIP embedded in MAD

The SLIP language is defined by a set of functions (subroutines) called by programs written in MAD. In the MAD language, function names are distinguished from variable names by the addition of a terminal period. Thus, "X" is the name of a variable, and "X." is the name of a function. Furthermore, the value of a function is obtained by following the name of the function by a series of expressions or variable names enclosed within parentheses. If there is more than one of these arguments, they must be separated by commas. Therefore, "X.(Y,Z)" obtains the value of the function X when evaluated using arguments Y and Z.

Expressions (E) are defined as follows in MAD†:

If: op = any operator with two operands (eg. *)
op′ = any operator with one operand (eg. .ABS.)
F. = any function name (eg. SQRT.)
V = any variable name
C = any constant

Then the following are expressions:

	Examples
V	X, ABC, L1
C	1, 3.2, XYZ
E_1 op E_2	A+B, X*3
op′E_1	-2, .ABS.Y
F.(E_1), F.(E_1,E_2), etc.	LIST.(A) NEWTØP.(X,LIST.(Y))
V(E), V(E_1,E_2) etc. (subscription)	X(I), A(3,7)

Note that since F.(E_1,...) is an expression, then

F.′ (F.″ (E),F.‴ (E′))

is also an expression. This is called function nesting.

In evaluating expressions the following rule is followed:

Any expressions within parentheses is evaluated before being used in evaluating expressiong at the next higher level. Therefore in the expression F1.(X,F2.(A+F3.(L))+3) the functions will be evaluated in the order F3., F2., F1.

† This simplified discussion is paraphrased from pages 10-13 of the MAD manual.

Digitized by Google

Some function arguments are merely variable identifiers; that is, the names of variables which do not yet have meaningful values. In this case the functions are programmed to assign a value to the variable named and not to evaluate and use that variable. These cases are rare (but important) in SLIP and will be carefully delineated.

2.4 Definition of SLIP Argument and Function Classes

It is important to note that some expressions yield values which are used as addresses by SLIP functions, rather than being used as data in some more conventional sense. Therefore, it is useful to establish some classes into which arguments and results of function evaluation may fall.

I (MAD Identifier): This is the class which is used merely as a variable name in the sense of the last paragraph of the preceding section. That is, the variable has no value until the function is executed. Its use will be explained as it occurs. This can be a variable name, or a variable name followed by a subscript expression.

D (datum) : any MAD expression

L (list): any MAD expression whose value is the name of a list (as defined earlier). These expressions are also members of class A (see below).

N (integer) : any D the value of which is interpreted to be an integer; e.g., 13, M, where M has been equated with an integer, for instance by the statement M = 13.

A (address) : any member of D which is the address of a cell in the computer. (Since any expression could actually be a member of A, this could be defined more precisely as any expression which is purposely computed to be a machine address.) Addresses are usually obtained by evaluating functions whose values are addresses.

R (reader) : a member of L which is used as a reader.

X (undefined) : a class into which all unspecified function values or arguments fall.

V (vector) : A MAD array containing elements of the class D.

S(sequence reader): a member of D which is used as a sequence reader.

Digitized by Google

Examples of the use of these functions:

X:F.(I,A)

A function named F. which doesn't return a value and whose arguments are a MAD identifier and an expression to be interpreted as a machine address respectively:

D:G.(L,D_1,D_2)

a function named G which returns a datum as its result and requires a list and two data respectively as arguments.

When using SLIP, all expressions are of integer mode unless they are meant specifically to have floating point values. Therefore, it is strongly urged that the programmer use the statement

NORMAL MODE IS INTEGER

in all MAD programs using SLIP functions.

Caution: when using the SLIP functions LISTRD, and LPRINT., names of lists (L) appearing on lists to be read or printed, i.e., MAD identifiers used as list names, may consist only of letters. Therefore, X, SAM and LIST are legitimate list names, while X1, S3B, Y003 and X(2) are not.

Also, no variable name may be the same as the name of any SLIP function used. E.g., if the function LIST. is called in a MAD program, then LIST may not be used as a variable name.

3.0 The Functions

3.1 Creating Available Space

X:INITAS.

This function requires no arguments. It forms the list of available space from all core storage not otherwise used. This must be the first executable statement in all programs using the SLIP functions.

3.2 Creating and Copying Lists

L:LIST.(I or 9)

I becomes a member of L as a result of LIST.

This function creates an empty list and leaves the address of its header in the cell named I, i.e., I becomes a name for the new list. The value of the function is the name of the list. Two names may be established for a single list by writing, for example,

Digitized by Google

LA = LIST.(LB)

which causes both LA and LB to be alternative names of the same newly created list.

If the argument is the literal "9", the name of the list is left only as the value of the function and the reference counter is set to zero. The literal "9" does not act as a name of the list. Furthermore the list will be automatically erased when all the lists upon which it appears are erased. On the other hand, if the argument is not "9" then the reference counter is set to 1 and the list will not be erased under any circumstances unless specifically commanded in the program.

L:LSSCPY.(L,I)

I becomes a member of L as a result of LSSCPY.

The arguments of this function are a list L and an identifier I. This function makes I the name of a copy of L and returns the name of the copy as its value. List L remains unchanged.

These two functions represent the primary methods for creating variable names of class L.

3.3 Placing information on lists.

A: NEWTØP.(D,A or L)

If the second argument is L, this function places the datum D on the top of list L. The value of the function is the address of the added cell. If the second argument is A, then D is inserted to the right of the cell specified by A.

A: NEWBØT.(D,A or L)

This function is identical to NEWTØP except that the datum is added at the end of list L or to the left of A.

For example, to put the word "cat" on the top of a new list, P, one could write:

```
LIST.(P)
W = $CAT$
NEWTØP.(W,P)
```

A more efficient method of achieving the same goal is by using nested function calls:

Digitized by Google

NEWTØP.(CAT,LIST.(P))

Another example of construction of lists is the following creation of a simple list structure:

NEWTØP.(5,LIST.(A))

NEWBØT.(13,A)

NEWTØP.(LIST.(B),A)

NEWTØP.(10,B)

This structure may be diagrammed as follows:

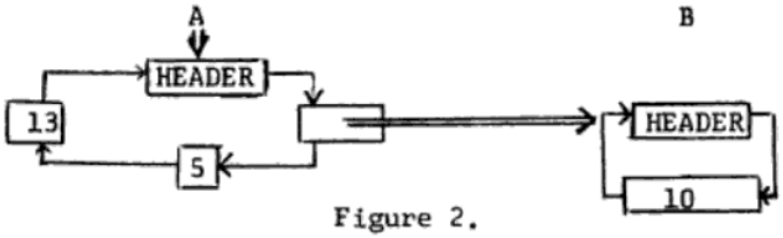

Figure 2.

This figure shows the list A which contains, in order, the address of the header of a sublist, the datum 5, and the datum 13. The sublist (whose name is B) contains the datum 10. Note that the single arrows indicate the circular and symmetric structure of the lists.

<u>L: MANY.(L,D,...,D)</u>

This function puts the contents of the D's on the bottom of list L. The value of the function is the name of the list. Example:

LIST.(A)

MANY.(A,CAR,X,Y,Z)

MANY.(A,B,C,D)

The first command creates an empty list A. The second command leaves A in this form: ("CAR",X,Y,Z). The third command leaves A in this form: ("CAR",X, Y,Z,B,C,D).

Note that the construction "$...$" causes the enclosed datum itself to be put on list A, i.e., the expression "CAR" appears in the top cell of list A. The other names, X,Y, etc., are not physically present in A. Rather it is the datum to which the variable name has been equated which appears on A. Thus if the statement X=$JANE$ had occured just prior to above statements, then "JANE", not "X", becomes the second datum in A. For convenience, however, it may be referred to as X.

Digitized by Google

The same list can also be constructed with the single command:

MANY.(LIST.(A),CAR,X,Y,Z,B,C D)

Example: The list structure created in the example Figure 2 may also be created as follows:

MANY.(LIST.(A),MANY.(LIST.(B),10),5,13)

D:SUBSBT.(D,L)
D:SUBSTP.(D,L)

The datum D is substituted for the datum presently stored on the top (bottom) of list L. The value of this function is the datum replaced by D.

D:SUBST.(D,A or L)

If the second argument is A, the datum D replaces that stored in list cell A. The value of this function is the datum replaced by D. If the second argument is the name of a list, SUBST. will replace the header of that list with datum D. Although this is a legal operation, the programmer is warned that it is potentially dangerous since the list's reference counter is lost, as is the name of its description list.

L:INLSTR.(L_1,A or L_2)
L:INLSTL.(L_1, A or L_2)

This function takes the set of linked cells constituting the body of list L_1 (i.e., all but the header of that list) and inserts it to the right (left) of cell A or L_2. L_1 is made into an empty list and its name delivered as the value of the function.

N:SUBMNY.(L,V)

This routine replaces each occurrence of any datum on list L which also appears in some odd subscripted location of the dimensioned variable V with the symbol which appears in the following even location of V. V(0) must contain the number of items in the array V . (See function DECLAR.) For example, if,

V(0) ... V(4) contain 4, CAT,$DØG$,CAR,$AUTØ$

then SUBMNY. (L,V) will replace all occurrences of data on list L having the

Digitized by Google

value CAT with $DØG$; similarly, CAR with $AUTØ$.

Note that the data elements are not restricted to BCD values.

The value of the function is similar to that of a test (see section 3.7). The value is 1B if any substitutions occured, 0B otherwise.

X:DECLAR.(V,D_1,D_2,...,D_n)

This function places the data D_1.....,D_n sequentially into elements of the dimensioned variable V(V(0)+1) to V(V(0)+n). Then it stores V(0)+n in V(0). If V(0) is initially > 500 or < 0, it will be set to zero. V must be dimensioned in the MAD program. For example, when V is empty (V(0)=0), then

DECLAR.(V,DA,DB,DC,DD)

changes the contents of V to

V(0) ... V(4) : 4,DA,DB,DC,DD

A second call with the same arguments would produce V(0) ... V(8) = 8,DA,DB, DC,DD,DA,DB,DC,DD.

One use of this function is to set up a correspondence between the name of a list in alphabetic and internal form. E.g.:

DECLAR.(V,L,L)

produces

V(0) ... V(2) : 2,L, internal name of L.

V can then be used by SUBMNY. to substitute internal names for alphabetic names of sublists.

3.4 Retrieving Information From a List

D:TØP.(L)

This function has a list as its argument. Its value is the datum stored on the top (leftmost) cell of that list. The list is not modified by this operation.

D:BØT.(L)

This function behaves as does TØP. except that the datum stored in the bottom (rightmost) cell of list L is delivered as the value.

Digitized by Google

Example: to duplicate the first datum on list A:

NEWTØP.(TØP.(A),A)

Example: to retrieve the datum "10" in the list structure of Figure 2:

TØP.(TØP.(A))

D:PØPTØP.(L)

The top cell of list L is "popped off", i.e., returned to available space, and the datum contained therein delivered as the value of the function.

D:PØPBØT.(L)

This function behaves exactly as does PØPTØP. except that the bottom cell of L is popped off.

Example: to move a datum from the top of list A to the bottom:

NEWBØT.(PØPTØP.(A),A)

L:NULSTR.(A,L)

L:NULSTL.(A,L)

The arguments of these functions are A, the address of a cell on list L, and L, the name of the list. The list L is split into two separate lists by creating a new list having all cells to the right (left) of and including cell A as its members. The cells thus associated with the newly created list are removed from L. The name of the new list is delivered as the value of the function.

3.5 Deleting Lists and List Cells

D:REMØVE.(A)

The argument of this function is a list cell. The indicated cell is deleted from the list on which it appears and the datum stored in that cell delivered as the value of the function. If the address of a list header is inadvertently given as the argument, then the value of the function will be zero and an error message printed. However, the program will continue to be executed.

Digitized by Google

D:IRALST.(L)

The argument is the name of a list. This function erases L. I.e., all of its cells including its header are returned to available space. However, this list (and its contents) may still be the sublist of another list. If this condition is detected, the erasure is not carried out. Instead the list's reference counter is stepped down by one. The value of the function is the reference counter, an integer giving the number of lists of which L is a sublist. If that value is zero then L has actually been erased.

L:MTLIST.(L)

The argument of this function is the name of a list. The list L is emptied, i.e., its cells restored to available space. The value of the function is the name of the now empty list.

3.6 Functions Which Obtain the Machine Address of a List Cell.*

A number of functions depend on their arguments being machine addresses of list cells, e.g., REMØVE., NULSTL., SUBST., etc. Certain functions produce such machine addresses as their values, including the following:**

L or A: MADLFT.(A or L)
L or A: MADRGT.(A or L)
(machine address left (right))

The arguments of this function are L, the name of a list, or a cell address. The value of the function is the machine address of the cell to the left or right of the cell specified. If that cell is the header of list L, then the value is delivered in the format of a list name, otherwise as an integer.

---

* Many of these functions may return an element of class L or A. Since L is a subset of A, no problems arise.

** Note: NEWTØP and NEWBØT, previously defined, are also in this class.

Digitized by Google

A or L:MADNTP.(L,N)

A or L:MADNBT.(L,N)

(machine address N top (bottom))

The arguments of these functions are L, the name of a list, and N, an integer. The value of the function is the machine address of the nth cell from the top (bottom) of the list. If that cell is the header of list L, then the value is delivered in the format of a list name, otherwise as an integer. If N is greater than the number of cells on list L (including the header cell), then the effect of this function is as if N modulo M had been the second argument, where M is the number of cells (including the header cell) on list L.

Digitized by Google

3.7 Tests

These functions should be declared BØØLEAN to be used most effectively in MAD. They return the values 1B(TRUE) and 0B(FALSE).

N:LEMPTY.(L)
(list empty)

Returns 1B if list L is empty; 0B otherwise.

N:NAMLST.(D)

Returns 1B if D is the name of a list; 0B otherwise.

N:LSTSEQ.(L_1,L_2)

Returns 1B if lists L_1 and L_2 are identical in contents (data terms and sublist names); 0B otherwise.

```
Example: WHENEVER LSTSEQ. (L(I),L(I+1))
             PRINT CØMMENT $THEY'RE EQUAL$
         ØTHERWISE
             PRINT CØMMENT $THEY'RE UNEQUAL$
         END ØF CØNDITIØNAL
```

3.8 The "Sequence" Reader Mechanism For Linear Lists

In order to take full advantages of the power of list processing, the programmer must be able to traverse lists and list structures which, from the point of view of the program, are only partially specified. It may be required, for instance, to construct a process which replaces every occurrence of the symbol X with the symbol Y on an input list. Furthermore, the process may be required to operate properly on any list, e.g., a list with no X's, a list with X's in consecutive cells, an empty list, etc. The reader mechanism and its associated functions make such processes possible. This section will discuss a simple sequence reader mechanism for scanning a list in a "linear" mode, i.e., ignoring sublists. The following section will discuss a more complex reader for scanning list structures.

Digitized by Google

A reader is analogous to a bookmark in that it enables the programmer to keep his place while traversing a list. A reader has the following format:

LL	LR

where LL and LR are the link-left and link-right fields of the cell pointed to.
The reader for the list in Figure 1, for example, is

73555	74613

As the reader is moved through a list, it serves the function of pointing to successive cells on that list.

S:SEQRDR.(L)

The statement S = SEQRDR.(LST) causes S to be appointed as a sequence reader of the list LST. That is, the value of the function is a single SLIP cell in the sequence reader format. In its initial state the sequence reader points at the header of LST.

D:SEQLL.(S,I)
D:SEQLR.(S,I)
(sequence linear left (right)) (I becomes a member of D)

The arguments of these functions are R, a previously appointed sequence reader, and I, an identifier. Execution causes the LPNTR field of R to point to the next list cell to the left (right) on the list for which it has been made a sequence reader. The value of the function is the contents of the cell pointed at. In addition, I is set according to the nature of the contents of the cell as follows:

+1	if cell contains a header
0	if cell contains an element (not a name)
-1	if cell contains a list name.

Example: to test if the datum "X" is on list LT:

```
        READ = SEQRDR.(LT)
LOOP    DATUM = SEQLR.(READ,FLAG)
            (continued)
```

Digitized by Google

```
WHENEVER FLAG .E. 1
     TRANSFER TØ FAIL
ØR WHENEVER DATUM .E. $X$
     TRANSFER TØ FØUND
ØTHERWISE
     TRANSFER TØ LØØP
END ØF CØNDITIØNAL
```

A: LPNTR.(R)

The argument of this function is a previously appointed reader. The value of the function is the machine address of the cell to which the sequence reader points.

Example: search a list structure for the datum "JOHN" and substitute for it the datum "MARY". One way of accomplishing this is to write a general subroutine to search a structure for a given datum, as shown in the external function, SRCHL., which searches LST for ØBJ:

```
        EXTERNAL FUNCTIØN (LST,ØBJ)
        NØRMAL MØDE IS INTEGER
        ENTRY TØ SRCHL.
        LIST.(STACK)
        NEWBØT.(LST,STACK)
TEST    W'R LEMPTY.(STACK),FUNCTIØN RETURN -1
        S = SEQRDR.(PØPTØP.(STACK))
READ    X = SEQLR.(S,F)
        W'R X .E. ØBJ
             IRALST.(STACK)
             FUNCTION RETURN LPNTR.(S)
        Ø'R F .L. 0
             NEWBØT.(X,STACK)
             T'Ø READ
        Ø'R F .E. 0
             T'Ø READ
        Ø'E
             T'Ø TEST
        E'L
```

Digitized by Google

(Note: W'R, T'Ø, etc., are abbreviations of MAD statement types recognizable by the translator.)

This function returns as its value the address of the cell containing ØBJ unless ØBJ does not occur on the list in which case -1 is returned. A program using SRCHL. which substitutes MARY for JØHN on list L is:

```
ADR = SRCHL.(L,$JØHN$)
W'R ADR .GE. 0
SUBST.($MARY$,ADR)
Ø'E ...  etc.
```

3.9 The "Advance" Reader Mechanism For Lists and List Structures

the advance reader mechanism may be operated in such a way that all sublists of a main list are scanned as are all sublists of sublists, etc. This is called the "structure mode". When readers are used in this way, sublists are scanned as soon as their names are encountered. When the sublist has been completely scanned, the advance reader resumes scanning in the main list. In order to accomplish this, the reader must actually be a pushdown list. Whenever a sublist is encountered, a new SLIP cell is taken from available space, made into a reader for the sublist, and put on top of the reader stack. Thus, at any given moment in the scanning of a list structure, the reader stack constitutes a scanning "history" up to that moment.

The advance reader mechanism is able to scan quite complicated list structures, even structures in which a sublist is named on more than one higher level list, as, for example, structure A in Figure 3. In this case the elements are scanned (using ADVSER.) in the order X,P,Q,U,V,R,Y, U,V.

List structure B, on the other hand, cannot be scanned without looping. In this case the scanning order is X,P,Q,X,P,Q,... and so on endlessly. Although it is impossible to input list structures with multiple occurrences of sublist names, it is possible to construct them during processing. It is the programmer's responsibility to handle such structures properly. One method of scanning this type of structure involves using the list marking functions to be described later.

Digitized by Google

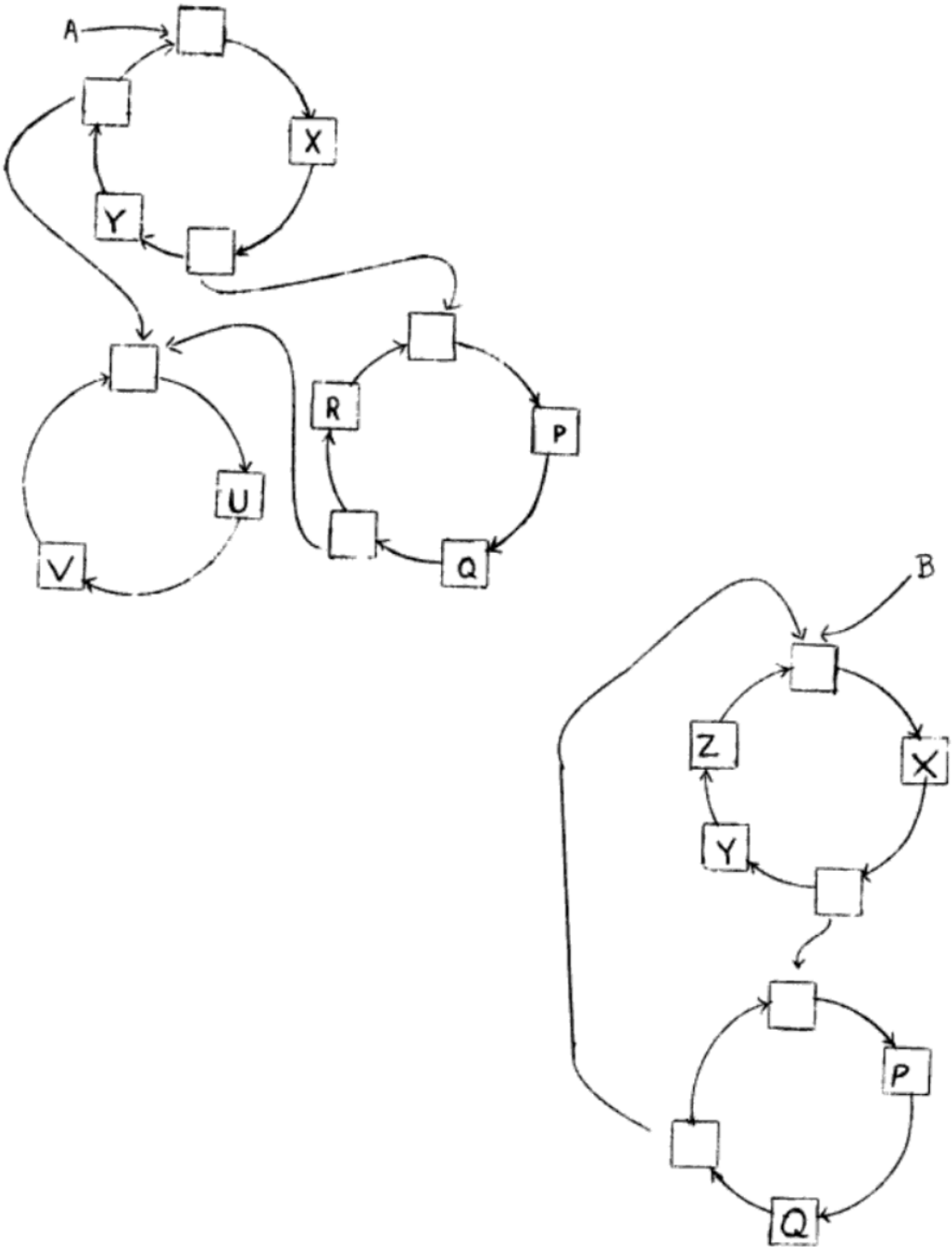

Figure 3.

Digitized by Google

The format of an advance reader cell is as follows:

3	LPNTR	LINK
3	LØFRDR	LCNTR

where LPNTR is the address of the cell pointed to, LØFRDR is the name of the (sub)list into which LPNTR is pointing, and LCNTR is an integer giving the sublist level to which the reader has descended. (Top level = 0, first sublist level = 1, etc.)

R:LRDRØV.(L)

The statement RDR = LRDRØV.(LST) causes RDR to be a reader of list LST. That is, the value of the function is a single SLIP cell in the reader format.

D:ADVLWR.(R,I)
D:ADVLER.(R,I)
D:ADVLNR.(R,I)
D:ADVLWL.(R,I)
D:ADVLEL.(R,I)
D:ADVLNL.(R,I)

(I becomes a member of D)

There are six linear advance functions, each with two arguments: a previously appointed reader R, and an identifier I. The last letter of the function's name determines the direction in which the advance is to proceed, i.e., R = right, L = left. The next to last letter determines the target at which the advance aims and thus the termination criterion: W = word, (i.e., the next cell regardless of contents), E = element (i.e., any non-name datum) and N = name of a list. The identifier I is set to zero if the target is found, otherwise non-zero.

The task of the linear advance functions is as follows: (a) to cause the LPNTR of the reader to advance through a list cell by cell and come to rest pointing at the first cell which contains a target of the designated type; (b) to deliver as its value the datum pointed at by LPNTR; (c) to set the identifier I to zero if the target was found, non-zero otherwise. In the latter case LPNTR is left pointing at the head of the list and zero is delivered as the value of the function.

Digitized by Google

D:ADVSWR,(R,I)
D:ADVSER,(R,I)
D:ADVSNR,(R,I)
D:ADVSWL,(R,I)
D:ADVSEL,(R,I)
D:ADVSNL,(R,I)
(I becomes a member of D)

There are six structural advance functions. The letters W,E,N,R, and L in the function name have the same meaning as for the linear advance functions. Structural advances permit the reader to descend and ascend in list structures, i.e., to traverse sublists as well as lists. The structural advances perform the same three functions as the linear advances and, in addition, (d) modify the LCNTR of the reader such that it indicates the level within the list structure to which the reader has descended (zero indicates the main list); (e) to update, if required, that field of the reader which is retrievable by the LØFRDR. function such that it always contains the name of the list into which the reader is pointing; (f) to descend into sublists as soon as they are encountered and update the reader stack accordingly; (g) to ascend back into the higher level list as soon as its sublist has been scanned and update the reader stack accordingly.

The structural advance operations proceed as follows. The ID of the cell currently being pointed to is examined to see if it contains a name. If it does then a reader is created for the sublist and put on top of the reader stack. If the cell being pointed at is not a name, then the next cell is examined for correspondence to the target criterion. If this cell corresponds to the target sought, the reader advance terminates in exactly the same way a linear reader advance does, i.e., the value of the function is the datum contained in the target cell, and the identifier is set to zero. If this cell does not correspond to the target and is not a header, then the whole process begins again.

If an advance is made to a list header, the LCNTR of the reader determines whether the advance terminates in a failure or continues the search. If LCNTR is zero then the header of the main list has been encountered and the target specified by the advance function could not be found and therefore the identifier is set to non-zero. If LCNTR is not zero, then the encountered

Digitized by Google

and the target specified by the advance function could not be found and therefore the identifier is set to non-zero. If LCNTR is not zero then the encountered header is that of a sublist. The reader therefore ascends from that sublist to that list cell on the higher order list which led from that sublist initially. This is accomplished by excmining LCNTR and, if it is greater than zero, popping off (i.e., erasing) the top cell in the reader stack.

Example: to scan through a structure A (without changing any elements):

```
        RA = LRDRØV.(A)
SCAN    T'H SCAN, FOR I = ADVSWR.(RA,F),ADVSWR.(RA,F),F.NE.0
```

Example: to perform some process on every element on a list structure:

```
        R = LRDRØV.(A)
MØRE    T = ADVSER.(R,FLAG)
        W'R FLAG .NE. 0, T'O DONE
             (process ...)
        T'Ø MØRE
DØNE    ...
```

A:LSPNTR.(R)

The argument of this function is a previously appointed reader. Its value is the machine address of the cell at which the reader points. This function applies to advance readers and should not be confused with LPNTR, which applies to sequential readers.

N:LCNTR.(R)

(level counter)

The argument of this function is a previously appointed reader. Its value is an integer giving the current level of the reader.

L:LØFRDR.(R)

(list of reader)

The argument of this function is a previously appointed reader. The value is the name of the list into which LPNTR is pointing.

Digitized by Google

R:LVLRVT.(R)

R:LVLRV1.(R)

(level revert (top) (one level up))

The argument of each of these functions is a previously appointed reader. These functions cause the reader to ascend (i.e., reverse) in the list structure until it is again pointing into the top level list (list one level up). If the reader is already pointing into the main list nothing happens. Otherwise, the LPNTR of the reader is left pointing to that cell in the main list (list one level up) which contains the name of the substructure into which the reader was pointing when the function was executed. The value returned by the functions is the address of the reader.

R:INITRD.(R)

(initialize reader)

The argument is a previously appointed reader. This function causes the LPNTR of the reader R to point to the header of the list to which the reader is currently pointing. The value of the function is the address of the reader

Example: if a reader RDR is currently pointing to X in Figure 4., then INITRD (RDR), LVLRV1.(RDR), and LVLRVT.(RDR) will reposition RDR at points A, B, and C respectively.

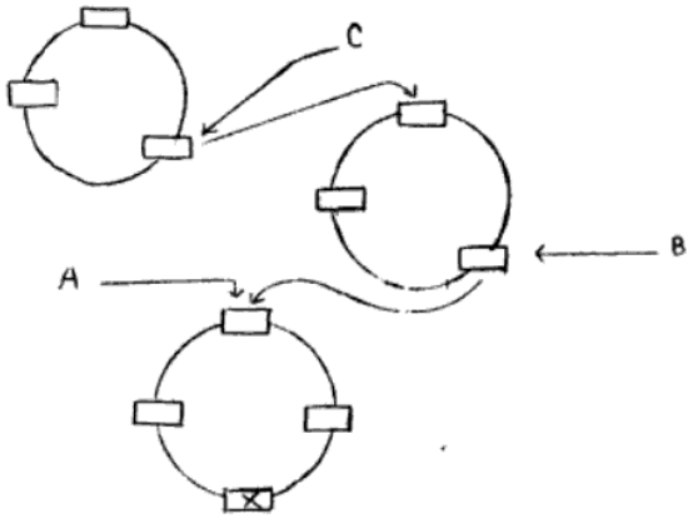

Figure 4.

Digitized by Google

R:RDRCPY.(R)

The argument of this function is a previously appointed reader. The function copies the reader and delivers the name of the copy as its value. The utility of RDRCPY. lies in the fact that the reader mechanism carries with it a record of its history. The programmer may wish to leave this history intact but also to advance the reader. The copy produced by this function may be advanced as if it were the original reader, leaving the original intact.

D:REED.(R)

The argument of this function is a previously appointed reader. The value is the datum contained in the cell to which the reader is currently pointing.

N:IRARDR.(R)

(erase reader)

The argument of this function is a previously appointed reader. The function erases reader R and leaves as its value LCNTR, i.e., an integer which indicates how deeply within the list structure the reader had penetrated prior to its erasure.

3.10 List Marking Functions

L:MRKLST.(N,L)

The arguments of this function are L, the name of a list, and N, an integer = 0,1,2, or 3. This function places the mark, N, on list L. All lists are marked with 0 initially. The value of this function is the name of the list L.

L:MRKLSS.(N,L)

This function is identical to MRKLST. with the exception that every list of the list structure L is given the mark N.

Digitized by Google

N:LSTMRK.(L)

The argument of this function is the name of a list. Its value is the mark on that list.

These three functions provide a means of scanning list structures in which lower level sublists contain higher level lists as sublists. To accomplish this, each sublist must be marked prior to scanning. Thereafter, no sublist should be scanned if it has already been marked.

3.11 Description Lists

Any list may have a description list. A description list becomes an integral part of its host list, not a sublist in the ordinary sense. The erasure of a list (but not its emptying) also erases its description list. Access to the description list of a list is generally via the name of the host list. A description list contains information about the list with which it is associated and is composed of attribute-value pairs (e.g., "part of speech", "noun"; see Figure 1).

D:NEWVAL.(D_1,D_2,L)

This function assigns D_2 as the value of attribute D_1 on the description list associated with list L. NEWVAL will put D_1 on the description list if it is not already there, and if L has no description list it will create one. The value of the function is the original value of D_1 if there was one, otherwise it is zero.

D:ITSVAL.(D,L)

This function retrieves the value of attribute D on list L. The value of the function is the value of D if that attribute is on the description list of L, otherwise it is zero.

D:NØATVL.(D,L)

This function deletes attribute D and its value from the description list of L. The value of the function is the value of D if there was one, otherwise it is zero.

Digitized by Google

L:MAKEDL.(I or L_1, L_2)

(I becomes a member of L)

This function makes I or L_1 the description list of L_2. The value of the function is the name of the host list.

L:NØDLST.(L)

This function removes the description list from L. Unless it is referred to by some other list, it is erased. The value of the function is the name of the description list.

L:LSTNAM. (L or R)

If the argument is L, the name of a list, the value of this function is the name of its description list. If the argument is a reader, R, the value is the name of the list to which it points.

3.12 Input-Output Functions

X:LISTRD.(L,N)

This function inputs a list from the tape unit specified by the integer N, and gives it the name L. L must have been previously defined by the LIST. function.

The input format of lists is as follows:

1. The list must begin in column 1 of some card but may continue across any number of cards.
2. Elements within a list are separated by spaces.
3. All lists are enclosed in parentheses.
4. Lists and sublists are not given names. Sublists aee enclosed in parentheses and appear as elements of the main list.
5. Description lists are indicated by preceding the list by "DLIST". A description list must be punched as an element of the list to which it refers.

The list elements are inserted into consecutive SLIP cells. If an element consists of more than six characters, it is continued in the following cell. Cells containing elements which are continued in the following cell are indicated internally by a "1" in the sign bit of the second of the word-pairs.

Digitized by Google

Example: (A B DLIST ($CØLØR$ RED $SIZE$ $GIGANTIC$) C
(DØNE DTWØ (EØNE ETWØ)))

If the above list is read in with the statement LISTRD.(LIST.(CAR),3) the structure will be as follows:

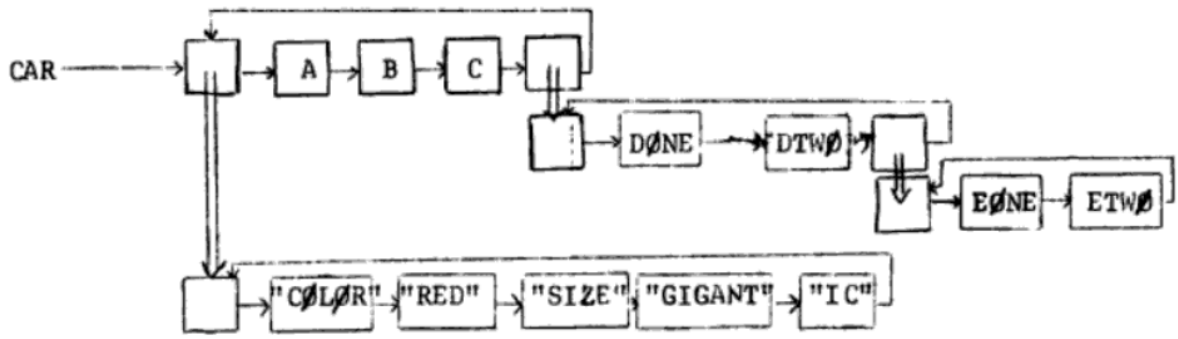

Figure 5.

L:LPRINT.(L,N)

This function prints list L on the tape unit specified by the integer N. The output format is the same as the input format except that description lists are printed as the first element of the list to which they refer. The value of the function is the name of the list printed.

Note: any list elements processed by LISTRD. and LPRINT. must consist entirely of either alphabetic or numeric characters. Thus

$L3$ and L(I)

will not be processed correctly by these functions.

3.13 String and Text Manipulating Functions*

X:TREAD.(L,N)

(text read)

This function reads text from the tape unit specified by the integer N. and puts the text into list format, and assigns it the name L. All characters including commas and parentheses are considered to be literals. Spaces are interpreted as element separators. Each element is inserted left-justified

*Although there exist languages better designed for string manipulation (e.g., SNOBOL), these functions may be so used. Programs intended primarily to manipulate strings might be written more conveniently in one of these other languages.

Digitized by Google

into a separate SLIP cell. If an element contains more than 6 characters it is continued in the following cell and the first cell is marked with a "1" in its sign position.

X:TXTPRT.(L,N)

This function prints list L as a linear text string on the tape unit specified by the integer N.

N:YMATCH.(L_1, L_2, L_3)

This function operates upon a list L_2, according to the specifications of L_1 and produces a list structure , L_3, as output. The list L_1 defines a set of segments into which the elements of L_2 are divided. The list L_3 contains copies of these segments as sublists. The value of the function is zero if L_1 can be successfully applied to L_2, non-zero otherwise.

The list L_1 defines a pattern using literals and integers as follows:

0 -	indicates a string of elements of any length with contents unspecified
1,2,...	indicate a string of elements of length 1,2,...
literals	indicate themselves

Example: YMATCH,(RULE,TEXT,RSULTS). If TEXT contains the sentence (MARY HAD A LITTLE LAMB ITS PRØBABILITY WAS ZERØ) and if RULE is the six element list (MARY 2 2 ITS 1 0) then YMATCH produces the list RSULTS which contain 6 sublists as follows

((MARY) (HAD A) (LITTLE LAMB) (ITS) (PRØBABILITY)
(WAS ZERØ))

and the value returned is zero.

This function begins by keying on the two words, MARY and ITS, specified in RULE. MARY is put on the first sublist of RSULTS. The next two words are put on the second sublist. The next two words are put on the third sublist. (Note that if there were not exactly 4 words between MARY and ITS, the match would not succeed and the function would return a non-zero value). ITS is put on the fourth sublist. The word following ITS is put on the fifth sublist and the remaining words go on the sixth sublist.

Digitized by Google

X:ASSMBL.(L_1,L_2,L_3)

This function operates on a list L_2 (the product of the function YMATCH for example) according to the rule of list L_1. The results are assembled in list L_3.

The list L_1 is composed of integers and literals as follows:

1,2,...	indicate the first, second, etc., sublist of list L_2
literals	indicate themselves

Example: ASSMBL.(SPEC,RSULTS,ØUTPUT). If RSULTS is the list obtained in the preceding example and if SPEC is the five element list (DID 1 HAVE A 3), then ØUTPUT is constructed as follows:

(DID MARY HAVE A LITTLE LAMB)

where the integers 1 and 3 refer to the first and third sublists of RSULTS, respectively.

X:RULE.(L_1,L_2,L_3)

This function combines the functions YMATCH and ASSMBL as follows: the list L_2 is operated upon according to the specifications of L_1 and produces the resulting list L_3.

The list L_1 consists of two segments separated by an equal sign. The first segment specifies a rule for segmenting L_1, and is composed according to the rules for the first argument of YMATCH. The second segment specifies a rule for assembling these segments into the list L_3, and is composed according to the rules for the first argument of ASSMBL.

Example: RULE: (RUL,INPUT,ØUTPUT). If INPUT is the sentence (MARY HAD A LITTLE LAMB ITS PRØBABILITY WAS ZERØ) and RUL is (1 0 2 ITS 0 = DID 1 HAVE A 3) then ØUTPUT is constructed as follows:

(DID MARY HAVE A LITTLE LAMB)

3.14 Functions Which Operate Upon the Information Contained in Specific SLIP Cell Fields, As Opposed to the Datum of the Cell

The following functions have as their values the extracted information in integer format:

Digitized by Google

N:ID.(A)
N:LNKL.(A)
N:LNKR.(A)

The following two functions store information in the three fields of a SLIP cell:

X:SETDIR.(N_1,N_2,N_3,A)
(set direct)

This function stores N_1, N_2, and N_3 in the ID, LNKL, and LNKR fields respectively of A. If any of the first three arguments has the value -1, the corresponding field of A is left unchanged.

X:SETIND.(N_1,N_2,N_3,A)
(set indirect)

This function has the same effect as SETDIR except that the cell modified is the cell whose address is contained in A.

D:STRIND.(D,A)

This function stores D in the cell whose address is in A. The value of the function is D.

N:MADØV.(A)

This function has as its value the machine address of the cell A, left in the form of an integer.

D:CØNT.(A)

This function returns as its value the contents of the cell whose address appears in A.

Digitized by Google

Examples: One way of doing nothing is

SETIND.(ID.(CELL),LNKL.(CELL),LNKR.(CELL),MADØV.(CELL))

and another way is

STRIND.(CØNT.(MADØV.(A)),MADØV.(A))

D:SQØUT.(D_1,D_2)

(squeeze out)

The argument D_1 is an extract mask. The value of the function is that field extracted from D_2, and shifted as far right as possible. Leading zeros are introduced in the process of right justification.

D:SQIN.(D_1,D_2,A)

(squeeze in)

D_1 is an extract mask which is complemented and then applied to the cell named A. D_2 is then shifted left so that its right hand end matches the first one-bit (from the right) in D_1. This shifted quantity is then masked by D_1 and stored into the cell named in A. The value of the function is the new value of the cell whose name is in A.

Example:

X = 7700K

Y = ABC

Z = a cell which contains the address of a cell which contains $LMNØPQ$

T = SQIN.(X,Y,Z)

yields $LMNØCQ$ as its result, stored in T and the cell named in Z.

X:SHININ(N,D_1,D_2)

(shift and insert)

This function shifts the contents of D_2 N binary places to the left and inserts D_1 in the now vacant low-order portion of D_2.

D:LANØRM(D)

The argument of this function is a set of alphameric characters. The value of the function is a new cell containing the left-justified contents of WORD, i.e., leading blanks removed. D is left unmodified.

Digitized by Google

3.15 Miscellaneous Functions

A:NUCELL.(0)

This function has a dummy argument of zero. Its value is the address of an empty cell taken from available space. The programmer should seldom, if ever, need this function.

X:RCELL.(A)

This function returns A to available space. The programmer should seldom have to use this function. Following execution, A contains all zeros.

D:MRKPØS.(A)

D:MRKNEG.(A)

These functions set the sign of A positive (negative). The altered contents of the cell is returned as the value of the function.

N:CNTSPC.(0)

This function has a dummy argument of 0. It returns as its value an integer giving the number of cells on available space. It is intended primarily as a debugging device.

N_1: HASH.(D,N_2)

This function classifies D according to a pseudo-random scheme. The number of classes is 2^{N_2} where N_2 is the second argument of the function. The value of the function is a pseudo-random number N_1 in the range of 0 to $2^{N_2} - 1$. The contents of D can now be placed on the $N_1{}^{th}$ list. It is necessary to have previously defined 2^{N_2} empty lists. This function enables the programmer to subdivide an unmanageably large amount of information into convenient segments in order to facilitate retrieval.

Digitized by Google

4.0 Index of Functions

Subroutine	Purpose	Page
D:ADVLEL.(R,I)	Advance linear element left	17
D:ADVLER.(R,I)	Advance linear element right	17
D:ADVLNL.(R,I)	Advance linear name left	17
D:ADVLNR.(R,I)	Advance linear name right	17
D:ADVLWL.(R,I)	Advance linear word left	17
D:ADVLWR.(R,I)	Advance linear word right	17
D:ADVSEL.(R,I)	Advance structural element left	19
D:ADVSER.(R,I)	Advance structural element right	19
D:ADVSNL.(R,I)	Advance structural name left	19
D:ADVSNR.(R,I)	Advance structural name right	19
D:ADVSWL.(R,I)	Advance structural word left	19
D:ADVSWR.(R,I)	Advance structural word right	19
X:ASSMBL.(L_1,L_2,L_3)	Assemble	27
D:BØT.(L)	Bottom element of list	10
N:CNTSPC.(O)	Count space	30
D:CØNT(A)	Contents	28
X:DECLAR.(V,D_1,....,D)	Declare	10
N:HASH	Hash	30
N:ID(A)	Identification	28
X:INITAS.	Initialize space	6
R:INITRD.(R)	Initialize reader	20
L:INLSTL.(L_1, A or L)	Insert list left	9
L:INLSTR.(L_1, A or L)	Insert list right	9
D:IRALST.(L)	Erase list	12
N:IRARDR.(R)	Erase reader	22
D:ITSVAL.(D,L)	It's value	23
D:LANØRM.(D)	Left adjust normalize	29
N:LCNTR.(R)	Level counter	20
N:LEMPTY.(L)	List empty	13
L:LIST.(I or 9)	List	6
X:LISTRD.(L,N)	List read	24

Digitized by Google

N:LNKL.(A)	Link left	28
N:LNKR.(A)	Link right	28
L:LØFRDR.(R)	List of reader	20
A:LPNTR.(R)	Linear pointer	15
L:LPRINT.(L,N)	List print	25
R:LRDRØV.(L)	List reader of	17
A:LSPNTR,(R)	Structural pointer	20
L:LSSCPY.(L,I)	List copy	7
N:LSTMRK.(L)	List mark	23
L:LSTNAM.(L or R)	List name	24
N:LSTSEQ.(L_1, L_2)	List equal	13
R:LVLRV1.(R)	Level revert one	21
R:LVLRVT.(R)	Level revert top	21
L or A: MADLFT.(A or L)	Machine address left	12
L or A: MADNBT.(L,N)	Machine address N from bottom	12.1
L or A: MADNTP.(L,N)	Machine address N from top	12.1
A:MADØV.(A)	Machine address of	28
L or A: MADRGT.(A or L)	Machine address right	12
L:MAKEDL.(I or L_1,L_2)	Make description list	24
L:MANY.(L,D_1,...,D)	Many data	8
L:MRKLSS.(N,L)	Mark lists	22
L:MRKLST.(N,L)	Mark list	22
D:MRKNEG.(A)	Mark negative	30
D:MRKPØS.(A)	Mark positive	30
L:MTLIST.(L)	Empty a list	12
N:NAMLST.(D)	Name of list	13
A:NEWBØT.(D, A or L)	New bottom on list	7
A:NEWTØP.(D,A or L)	New top on list	7
D:NEWVAL.(D_1, D_2, L)	New (attribute) value	23
D:NØATVL.(D,L)	No attribute value	23
L:NØDLST.(L)	No description list	24
A:NUCELL.(0)	New cell	30
L:NULSTL.(A,L)	New list left	11

Digitized by Google

L:NULSTR.(A,L)	New list right	11
D:PØPBØT.(L)	Pop off bottom	11
D:PØPTØP.(L)	Pop off top	11
X:RCELL.(A)	Remove cell	30
R:RDRCPY.(R)	Reader copy	22
D:REED.(R)	Read datum	22
D:REMØVE.(A)	Remove list cell	11
X:RULE.(L_1,L_2,L_3)	Rule	27
D:SEQLL.(S,I)	Sequence linear left	14
D:SEQLR.(S,I)	Sequence linear right	14
S:SEQRDR.(L)	Sequence reader	14
X:SETDIR.(N_1,N_2,N_3,A)	Set directly	28
X:SETIND.(N_1,N_2,N_3,A)	Set indirectly	28
X:SHININ.(N,D_1,D_2)	Shift and insert in	29
D:SQIN.(D_1,D_2,A)	Squeeze in	29
D:SQØUT.(D_1,D_2)	Squeeze out	29
D:STRIND.(D,A)	Store indirectly	28
N:SUBMNY.(L,V)	Substitute many	9
D:SUBSBT.(D,L)	Substitute bottom	9
D:SUBST.(D,A or L)	Substitute	9
D:SUBSTP.(D,L)	Substitute top	9
D:TØP.(L)	Top element on list	10
X:TREAD.(L,N)	Text read	25
X:TXTPRT.(L,N)	Text print	26
N:YMATCH(L_1,L_2,L_3)	Match	26

Digitized by Google