Programming Languages

T. E. CHEATHAM, JR., Editor

Symmetric List Processor

J. WEIZENBAUM
General Electric Co., Sunnyvale, Calif.*

A list processing system in which each list cell contains both a forward and a backward link as well as a datum is described. This system is intended for imbedding in higher level languages capable of calling functions and subroutines coded in machine language. The presentation is in the form of FORTRAN programs depending on only a limited set of "primitive" machine language subroutines which are also defined. Finally, a set of field, particularly character, manipulation primitives are given to round out the system.

I. Introduction

SLIP is a list processing system in which each list cell carries both a forward and a backward link as well as a datum. It is symmetric in the sense that its lists do not have a preferred orientation. Any operation which can be carried out on the top of a list can just as easily be carried out on the bottom. It is a language system designed to be imbedded in a higher order language capable of calling machine language subroutines. In its original implementation, it was imbedded in FORTRAN and it is in this context that it will be described.

SLIP is a descendant of at least four earlier list processors: (1) FLPL by Gelernter, et al.; (2) IPL-V by Newell, et al.; (3) Threaded Lists by Perlis, et al.; and (4) the author's own KLS. The manner and extent to which SLIP is a derivation of Threaded Lists and KLS will become obvious when its internal operations are described. Its relationship to FLPL arises out of the fact that it, like FLPL, is a higher language compiled list processing system. Its points of tangency with, as well as its departures from, FLPL will be discussed in a subsequent section of this paper. Suffice it to say here that Gelernter's original FLPL paper referenced above served the author and should serve the reader as extremely relevant background and is therefore highly recommended as adjuvant reading. SLIP's debt to Newell is both general, in that he first elaborated basic list processing concepts (e.g., that of a list of available space), and specific, in that certain SLIP processes are direct analogues of "*J*-Processes" to be found in IPL-V.

* Computer Laboratory.

Editor's Note. Publication of the detailed appendix to the paper "Symmetric List Processor" is a departure from the normal practice of the Programming Languages department of omitting detailed FORTRAN listings except for examples. This exception is being made because of considerable interest in the SLIP system and because of the manner in which references to FORTRAN program are integrated into the text. The Communications assumes no responsibility for the correctness of these listings nor for maintaining such listings in the future. Any communications relative to them should be directly with Mr. Weizenbaum.

The purpose of this paper is twofold: On the one hand, it is addressed to the language specialist whose interests are presumed to be in the architecture of the system, the techniques used for imbedding it into FORTRAN, the system viewed as a language, and the possible use he may make of it in his own work; on the other hand, it is intended to serve the programmer interested solely in the use of SLIP as an extension of the tools presently available to him. In the latter context, at least part of this paper should serve as a minimum reference guide, but certainly not as a programming manual.

The organization of the paper is intended to reflect its dual purpose. There are six sections of which this Introduction is the first. The others are:

II. TECHNICAL MATTERS—which deals with matters of special interest to the computer language specialist, but which should be part of the background of the general user as well. Topics covered are: the primitives, the format of the information modules, the administration of the list of available space (garbage collection), and recursion.

III. THE PROCESSES—which describes the operations available to the SLIP programmer in the following order:
   1) Creating the LIST OF AVAILABLE SPACE (LAVS) and the PUBLIC LISTS (W's)
   2) Creating lists
   3) Placing information on lists
   4) The READER mechanism and the traversal of lists and list structures via that mechanism
   5) Other ways of retrieving information from lists
   6) Tests
   7) Miscellaneous processes
   8) Recursion

IV. BIT, CHARACTER, AND LOGICAL OPERATIONS which describes some non-list processes supplied as part of the SLIP system in order to round out its general symbol manipulation capabilities.

V. DISCUSSION which contains some prejudicial arguments tending to advertise SLIP.

VI. APPENDIX which is a listing of the primitive based FORTRAN SLIP system and, as such, constitutes an unambiguous and precise documentation of its major mechanism.

Since a paper of the present type cannot presume to be a programming manual, a number of assumptions concerning its readership must necessarily be made. Among those governing what is and what is not said here are: (1) that the reader has at least a thorough reading knowledge of FORTRAN, and (2) that he has at least a speaking acquaintance with the underlying principles of list processing and is, if necessary, willing to enlarge his understanding by reading and practice.

II. Technical Matters

SLIP was first written as a set of machine language subroutines intended for imbedding in a particular FORTRAN system. But, the manipulations of information which, when suitably composed, form the SLIP processes involve

only a few specific fields within computer words. It is therefore possible to write a few very primitive subroutines for dealing with these fields, and then to rebuild the entire system in FORTRAN using only these PRIMITIVES as constituent parts. This procedure has three advantages. (1) Documentation in this form can be rigorously checked by simply running it. (2) Anyone setting out to imbed SLIP within his own FORTRAN, or FORTRAN-like, system may be assumed to know FORTRAN already and therefore to be in a position to follow the code easily. (3) The primitive based system can be run on another machine as soon as equivalent primitives have been written for that machine (and possibly some minor FORTRAN convention differences accounted for). Machine differences such as word size are irrelevant in that they are reflected in the construction of the primitives and do not appear on the surface. (The card deck listed in the Appendix has run on both a 36- and a 48-bit machine. Great care was taken in the construction of the primitive based system to ensure that it contains no assumption about the specific way integers are treated by any particular FORTRAN implementation.)

An additional general observation which appears pertinent in the present context is that a primitive based system is extremely easy to modify even radically. Writing in terms of primitives is very close to manipulating flow charts. Consequently, even strategic changes of a system are introduced with a minimum of debugging effort.

The fundamental information module with which SLIP deals is a word pair. The first word of the pair is thought of as being divided into three distinct fields—namely, the *identifier* field of length 2 bits (ID), the *left link* field of machine address length (LNKL), and the *right link* field also of machine address length (LNKR). The second word of the pair is often thought of as containing a gross data word, i.e., of not being subdivided into subcomponents, but when it is split, then the same field definitions applicable to the first word are carried over.

The primitives which serve to extract the information contained in the three fields are *functions* which have as their *values* the extracted information in integer format. They are:

1. ID(CELL) which extracts the ID portion of CELL.
2. LNKL(CELL) which extracts the machine address stored in the left link field of CELL.
3. LNKR(CELL) which extracts the machine address stored in the right link field of CELL.

Two primitives with identical functions exist whose purpose is to fetch the contents of words indirectly. The duplication serves to avoid difficulties introduced by the fixed/floating-point conventions of FORTRAN.

4. CONT(A) and 5. INHALT(A) are functions which have as their values the information stored in the word the machine address of which appears as an integer in A.

The primitive

6. MADOV(A) is a function which has for its value the machine address of the cell A, left in the form of an integer.

The two primitives which serve to store information in the three fields of a SLIP word are:

7. SETDIR(I, L, R, CELL) which has the effect of storing I in the ID field, L in the LNKL field, and R in the LNKR field of CELL, respectively. If any of the first three parameters has the value −1, then the corresponding field of CELL is left unchanged.
8. SETIND(I, L, R, A) has the same effect as No. 7 except that the cell modified is that cell the machine address of which appears as an integer in A.

Finally, there are two primitives which store full-length words into specified cells. These are:

9. STRDIR(DATUM, CELL) which is a function having DATUM as its value and which performs the operation of storing DATUM in CELL. It exists entirely to avoid fixed/floating point naming difficulties, but has the side advantage that it can be nested. (E.g., CALL STRDIR(STRDIR (0, N), X) has the effect of placing zero in both N and X.)
10. STRIND(DATUM, A) which is a function having DATUM as its value and which performs the operation of storing DATUM in the cell the address of which appears as an integer in A.

The reader may check his understanding of the primitives by verifying that one way of doing nothing is

```
CALL SETIND(ID(CELL), LNKL(CELL), LNKR(CELL),
     MADOV(CELL))
```

and another way is

```
CALL STRIND(INHALT(MADOV(A)), MADOV(A)).
```

During the compilation of a FORTRAN/SLIP program, the FORTRAN compiler sets aside certain cells according to the various data declaration and dimension statements the programmer has written. Among the dimension statements which the programmer must write is one which creates the single dimensional array which will eventually become the LIST OF AVAILABLE SPACE (LAVS). All SLIP cells (i.e., word pairs) will subsequently be taken from LAVS and returned to it when appropriate. At any particular stage of the execution of the program, all SLIP cells are either in active use, or on LAVS directly, or on a list structure the main list of which is a sublist of LAVS. A SLIP cell in active use is either the READER, the HEADER, or a member of a list. If it is a member of a list, then the second of the word pair contains either a list NAME or some datum which is not a list NAME. The ID field of the SLIP cell discriminates among these four

possible circumstances according to the following table:

ID TABLE

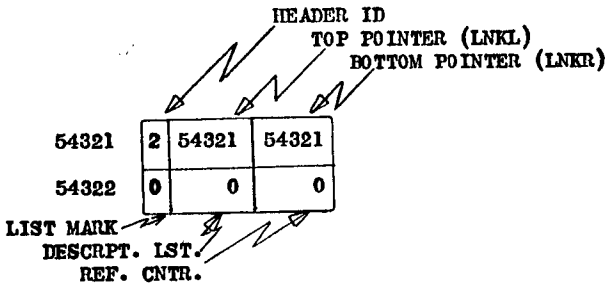
ID	*Function*
0	Datum not a NAME
1	Datum a NAME
2	HEADER
3	READER

Discussion of the READER mechanism is postponed. The way SLIP deals with HEADERS and list NAMES is, however, a crucial part of the philosophy underlying the entire system. Its description can therefore be made to serve as an entry to the understanding of the system.

A SLIP-list (henceforth called simply "list") is characterized by a directed linear string of SLIP cells such that the LNKR field of each SLIP cell points to (i.e., contains the machine address of) the next-to-the-right (or next-below) SLIP cell on the list, and its LNKL field points to the next-to-the-left (or next-above) SLIP cell on that list. On every list there is a single SLIP cell which has the HEADER ID. Any cell containing the machine address of the first word of the HEADER pair of words in both its LNKL and LNKR fields is said to contain the NAME of that list. The programmer's symbolic designation or the machine address of such a cell may be considered to be an ALIAS for the list name. Clearly, a list may have any arbitrary number of aliases. The LNKR field of the first word of the HEADER pair is said to point to the top and the LNKL field of that word, to the bottom of the list with which the HEADER is associated. The LNKR field of the second word of the HEADER pair is called the REFERENCE COUNTER of the HEADER'S list. Its function is to count the number of lists of which the list with which the HEADER is associated is a sublist. Its role will be clarified shortly.

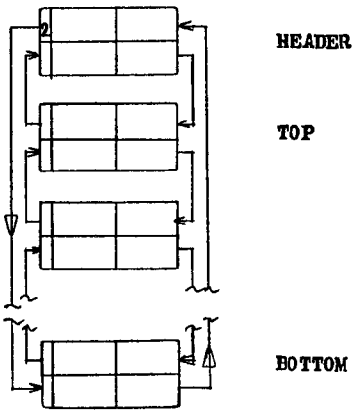
A list *structure* is a set of lists such that all but one of the lists of the set are sublists of the exceptional member of the set (often called the "main list"), or of sublists of such lists, or of sublists of sublists of such lists, etc. A list S is said to be a sublist of a list L if the NAME of S is contained in the datum field (i.e., the second word of a word pair) of a SLIP cell on the list L and the ID field of that SLIP cell has the value 1. Figures 1, 2, and 3 illustrate an empty list, a simple list, and a simple list structure.

Much of the power of list processing systems derives from the fact that it is possible to create and manipulate list *structures*, i.e., put the name of one list on another list and then to manipulate the resulting structure as a single entity. However, in one way or another, every list processor must face the so-called "responsibility issue." This issue arises in the attempt to answer the question, "Who is entitled to return a datum to LAVS when a cell containing that datum is returned to LAVS?". The reason this issue is difficult is that certain data may be on several lists simultaneously. In SLIP the issue arises only in that a list may be a sublist of several lists. Unlike certain other systems, SLIP stores symbols themselves in list cells rather than storing addresses of symbols.



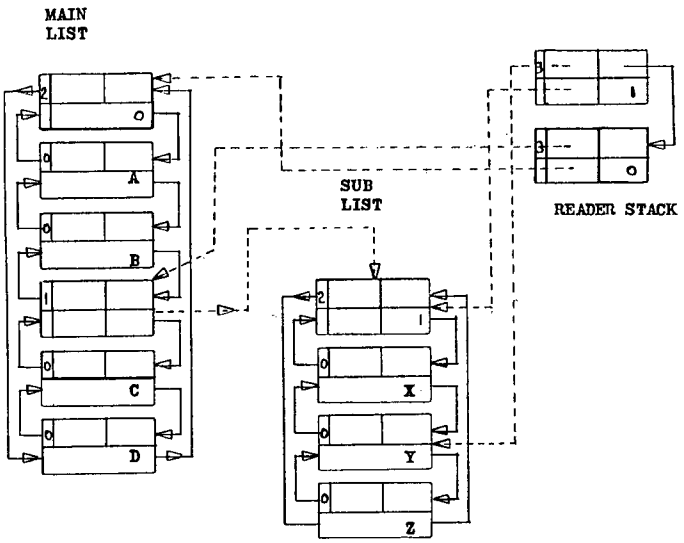
An Empty List

FIG. 1



A Simple List

FIG. 2



(A, B, (X, Y, Z), C, D)

A Simple List Structure

The READER is shown pointing to the cell containing Y.

FIG. 3

From the SLIP point of view, if the programmer wishes to store the machine address of a symbol on a list, then that address becomes a symbol in its own right. But when a list is "erased", i.e., its cells restored to LAVS, a decision has to be made as to whether or not to also erase its sublists. If any of its sublists are also sublists of other lists not yet ready for erasure, then the erasure of such sublists can lead to nothing but chaos.

In order to understand the way in which SLIP resolves the responsibility issue, it is first necessary to describe the structure of LAVS. The subroutine[1]

INITAS(SPACE, N) (1–18)

transforms the single dimensional array here called SPACE (previously declared by a dimension statement written by the programmer) of length N such that the LNKR field of every odd numbered word contains the machine address of the next odd numbered word except for that of the (N−1)-st word which i zero. All other fields of words in the array are set to zero. A cell internal to the system (AVSL) is loaded so that its LNKR field contains the address of the first word pair of LAVS, and its LNKL field, the address of the last word pair of LAVS. When a cell is needed from LAVS, the function NUCELL (19–34) delivers the contents of the LNKR field of AVSL as its value, i.e., assigns the cell pair then on top of LAVS to answer the need, and then stores in the LNKR field of AVSL the machine address stored in the LNKR field of the first word of the designated word pair. (See Appendix, line 28.) Cells are returned to LAVS by means of the subroutine RCELL (36–43). In particular, they are attached to the bottom of LAVS by means of suitable modification of LNKR of AVSL, and of LNKR of the cell previously on the bottom of LAVS. The removal from, or the addition to LAVS of individual cells thus maintains LAVS as a set of linked word pairs. But a list is already a set of linked cells. Lists can therefore be restored to LAVS in bulk, so to speak. When a list is to be restored to LAVS all that needs to be done is that the cell presently on the bottom of LAVS be made to point to the top cell of the list being erased, the bottom cell of that list given the appearance of a bottom cell of LAVS, and the bottom pointer of AVSL, i.e., its LNKL field, set to point to that bottom cell. (See MTLIST (55–68).) The responsibility issue arises if any of the data on the list just so erased contains the name of another list. When is that other list to be erased?

It will now be recalled that the notation which signifies that one list is the sublist of another is that a list cell on the higher order list carries an ID of 1 and a datum in the form of the NAME of the sublist. Since cells can be retrieved from the LAVS only one at a time, the opportunity to administer the erasure of sublists exists whenever NUCELL is at work. NUCELL checks the ID of every cell it is about to deliver (Appendix, line 26). If the value of that ID is 1, then the NAME of a sublist of a list already restored to LAVS has been encountered. The reference counter contained in the HEADER of the list named indicates the number of lists of which that list is a sublist. That counter is decremented by one and, if the new count is zero, the list restored to LAVS by means of the mechanism already discussed. Clearly, any sublists of any sublist so erased will be encountered eventually and also erased. Since list erasure always places the erased list on the bottom of LAVS, erasure is postponed to the last possible moment. One effect of this, is that some erasure effort is saved if the problem terminates before the space taken up by potentially erased list structures is required for further use. An important consequence of the "bulk erasure" of lists is that the time required by this mechanism is independent of the length of the list being erased.

The proper functioning of the mechanism discussed above depends, of course, on the reference counter of a list HEADER being counted up whenever that list is made a sublist of any list. By virture of the symmetry of SLIP lists, the placing of a datum on a list is always an *insertion* operation, even when the receiving list is initially empty. Every such datum placement operation therefore eventually appeals to one of only two basic placement functions; namely, NXTLFT (119–132) and NXTRGT (133–146). Each of these has the responsibility of checking whether the datum being placed is a list NAME (e.g., Appendix, line 127), and of counting the relevant counter up whenever appropriate.

The SLIP erasure technique clearly implies that during the running of the program, certain lists will "disappear" without any direct command from the programmer. Indeed, this is the intention.

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(K) (44–54) embodies a mechanism for resolving this problem.

The job of the function LIST(K) is to create an empty list and leave the NAME of the created list as its value. Unless the input parameter K is the literal "9", the NAME of the created list is also left in the cell K and the reference counter of that list set to the value 1. If K is the literal "9", the reference counter of the created list is set to zero, and its NAME left only as the value of the function. Consequently, any list which has been created by a call which gave that list a symbolic ALIAS (e.g., CALL LIST(L5)—L5 is the symbolic ALIAS of the created list and is, of course, "known" to the programmer) will have a reference count of one. It will not be erased by the erasure of lists on which its NAME may appear, i.e., of which it is a sublist. Only explicit, intentional erasure by programmer command will cause its cells to be restored to LAVS. (See IRALST(P) (69–78).)

The subroutine INITAS not only organizes the LAVS,

[1] Numbers in parentheses following names of SLIP processes refer to card numbers associated with the listing of those processes in the Appendix.

but also creates a set of 100 empty lists with aliases W(1), W(2), . . . , W(100), respectively. These are "public lists" in that, by means of COMMON statements, their aliases can be known to all subprograms. Their reference counts are made very large (4095) in order to ensure against inadvertent erasure. One of the more important uses of these lists is in the process of recursion where they serve as vehicles for the communication of parameters.

The basic machine language subroutines which make recursion possible are VISIT and TERM. Any statement within a subprogram to which control is to be transferred recursively, i.e., by means of the VISIT machinery, has first to be assigned a symbolic label by means of the ASSIGNMENT statement. Fundamentally, VISIT stacks the normal return address FORTRAN communicates to VISIT on an internal stack, cells for which are supplied via NUCELL, and transfers control to the statement the symbolic name of which was supplied as an input parameter. TERM, on the other hand, pops up the same internal stack and transfers control to the statement (or substatement) the machine address of which was found on that stack. Popped-up cells are returned to LAVS by means of RCELL.

The problem in recursion is not only to achieve a proper stacking of return addresses, but also to provide a means for communicating and stacking parameters. VISIT and TERM are therefore functions of two parameters, the second of which push parameters down on and pop them off the public lists, respectively, as appropriate. The pushing of parameters onto the public lists is accomplished by a set subroutines PARMT2(A1, A2), PARMT4(A1, A2, A3, A4), etc., (621–628), while the popping up of the public lists is achieved by RESTOR(N) (604–610) which pops up, i.e., restores to LAVS, the top cell of each of the first N of the public lists. PARMT2(A1, A2), to pick a specific example of one of the PARMT routines, pushes A1 down on W(1) and A2 on W(2). The statement

$$X = \text{VISIT(LOCO, PARMT2(A, B))}$$

can thus be seen to have the effect of placing A and B on top of W(1) and W(2), respectively, transferring control to a statement which has been assigned the label "LOCO", and eventually returning to the replacement operation implied by the "=" sign. The value of VISIT will be determined by the TERM subroutine. Whatever process begins at LOCO has, of course, to know that it will find its input parameters on top of W(1) and W(2), respectively.

TERM is a function with two input parameters the first of which is its value, while the second is simply a call to the RESTOR(N) subroutine. Thus, if the portion of the subprogram beginning with the statement labeled LOCO were VISITed by means of the VISIT statement shown above and TERMinated with the statement

CALL TERM(C, RESTOR(2))

then the overall result of the VISIT would have been to give X the value C. The two public lists W(1) and W(2) would have been left in exactly the condition in which the VISIT call found them.

Since VISIT is essentially a GOTO operation, whatever rules apply to such operations, particularly with respect to DO loops, must be observed as normally applied to all other GOTO uses. While the applicability of such rules restricts recursion within SLIP somewhat, it may nevertheless be used to considerable advantage as the two functions LSTEQL(LA, LB) (527–551) and LSSCPY(LA) (552–569) illustrate. The first of these test two list structures for "equality", while the second creates a copy of the list structure the ALIAS of which is its input parameter. Both are described in Section III.

In order to make the full power of list processing available to the programmer, he must be given the ability to traverse list structures in very comprehensive ways. In particular, it must be made possible to treat complex tree structures which, from the point of view of the currently activated program, are only partially specified. The program may know, for example, what to do with the next symbol on a list if there is one, but not how many symbols a given list contains. It may know that a certain list has sublists, but not whether any of the sublists have sublists in turn. If the programmer is left to deal with the problems engendered by such uncertainties by devices he must tailor to fit each specific instance, an unbearable bookkeeping burden will have been placed on his shoulders. SLIP relieves this burden by means of the READER mechanism.

The READER is essentially an indexing apparatus. But indexing over list structures is considerably more complicated than the equivalent operation over regular arrays. If an index register is pointing to a particular datum within an array, then a computation of which the contents of that index register is a parameter can be specified such that the value produced by that computation is the machine address of a cell within the array standing in any desired relation to that originally pointed to. The important point about such computations, for the present purpose, is that (in general) they do not involve any idea of the *history* of the affected index register. If, however, some register is pointing to a list cell deep within a list structure (i.e., on a list which is the sublist of a sublist, etc.) of some main list, and it is desired to have that register point successively to the nodes of the structure (i.e., those cells which contain the NAMEs of the various sublists, which ultimately lead to the sublist currently being pointed into) in an order the reverse of that in which they were encountered during the descent into the structure, then, clearly, historical information must be available to the effectuating procedure. A READER is a stack—often a depth of only one—associated with a list structure which, together with the list structure itself, contains sufficient information to make the tracing and retracing of the entire structure possible. No marginal notes, so to speak, need to be kept by the programmer.

A READER is created, i.e., a SLIP cell taken from LAVS

for the purpose, for a specific list or list structure. The statement invoked is (for example)

MYRDR = LRDROV(MYLST) (396–402).

This statement assumes that there exists a list with the ALIAS MYLST. A cell is taken from LAVS, its address left in the form of an integer in MYRDR (from the programmer's point of view MYRDR *is* the READER), and the cell fields filled as follows:

First word of the pair:

ID = 3
LNKL = address of the HEADER of MYLST. This field is called the LIST POINTER (LPNTR) of the READER.
LNKR = 0

Second word of the pair:

ID = 0
LNKL = address of the HEADER of MYLST
LNKR = 0 This field is called the LEVEL COUNTER (LCNTR) of the READER.

A READER in the condition described above is said to be in its *initial state.*

There exists a set of ADVANCE operations (212–374) which have as one of their responsibilities the modification of the state of the READER which is one of their input parameters. (The detailed functioning of these operations especially as seen from the programmer's point of view, is described in Section III.) These operations fall into essentially two main subclasses: the LINEAR and the STRUCTURAL advances. By virtue of the symmetry of SLIP, each class is again divided according to the direction (left or right) in which the advance is to proceed. For the present purpose, only advances to the right (or down) lists and list structures will be discussed.

A LINEAR advance begins with the list cell the address of which is currently specified by the LPNTR of the READER given to the operation as a parameter. It proceeds by following the links (LNKR in the right case, LNKL, otherwise) stored in the list cells themselves until an advance termination criterion is invoked. The address of the cell then pointed to replaces the LPNTR of the READER. The READER can be initialized by means of the function INITRD(K) (442–447) which simply replaces the LNKL field of the first word of the READER pair by the contents of the LNKL field of the second word.

STRUCTURAL advances proceed exactly as do LINEAR ones providing that neither at the beginning of the advance, nor at any stage prior to its termination is a cell containing a list NAME (i.e., with ID = 1) or a HEADER (ID = 2) encountered. Should a cell containing a list NAME be encountered in the course of a STRUCTURAL advance, a new cell is taken from LAVS, the current READER cell (with the address of the encountered cell in its LPNTR field) copied into the newly fetched cell, and the original READER cell fields set up so as to make the READER appear to be an initialized READER for the list whose NAME was just encountered. The LCNTR field is, however, counted up by one so that it reflects the *level* to which that READER has descended into the structure. In order not to lose the historical thread, the address of the cell just taken from LAVS is placed in the LNKR field of the first word of the READER. Repeated operation of this machinery can be seen to establish what is essentially a READER stack which preserves the entire traversal history of the list structure to which it applies as long as the stack is not popped up.

The encounter of a HEADER in the course of a LINEAR advance is one advance termination criterion. However, in the STRUCTURAL advance case the LCNTR of the READER is consulted to determine whether or not the advance should terminate. If the value of that counter is not zero, then the READER stack is popped up by one level and the advance continued from the cell indicated by the state of the newly "uncovered" LPNTR.

III. The Processes

1. CREATING THE LIST OF AVAILABLE SPACE (LAVS) AND THE PUBLIC LISTS (W'S)

INITAS(SPACE, N) (1–18)

A linear array of N words has been declared by the programmer by means of a DIMENSION statement. The statement

CALL INITAS(SPACE, N)

where N is the size of the array so set aside, will create a LIST OF AVAILABLE SPACE (LAVS). In addition, a set of empty lists with ALIASes W(1), W(2), . . . , W(100), respectively, will be created. The storage space required for these lists and their ALIASes has been set aside by the system and is *not* taken from LAVS. Any subprogram which requires access to any of the public lists must declare them by means of the following COMMON statement (or a variation thereof depending on the conventions associated with the particular FORTRAN system used):

COMMON AVSL, W(100)

2. CREATING LISTS

LIST(K) (44–54)

The function LIST(K) creates an empty list and leaves its name both as its value and in the cell K. The cell K thus becomes an ALIAS for the list. If K = 9, then the reference count of the list is zero, otherwise it is one. Two ALIASes may be established at once by writing, for example,

LA = LIST(LB)

which causes both LA and LB to contain the NAME of the created list, i.e., to be ALIASes for that list. Since LIST is a function, it may be nested.

3. Placing Information on Lists. In the following, the term DATUM will be used to indicate the information which is to be put on a list. A DATUM may be any proper FORTRAN symbol, e.g., a fixed- or floating-point number, a set of Hollerith characters, a machine address, the NAME of a list, etc., providing that symbol is containable in a single word. The bookkeeping associated with placing NAMES of lists on lists is an automatic system function.

NEWTOP(P, Q) (109–113)

NEWTOP pushes the datum P down on top of the list Q. It is a function which has for its value the machine address of the cell taken from LAVS for the storage of the datum and the associated linking information. The following example creates a quite complex list structure:

```
      DIMENSION LST(17)
                :
      CALL LIST(LST(1))
      DO 1 I = 2, 17
    1 CALL NEWTOP(LIST(LST(I)), LST(I−1))
```

NEWBOT(P, Q) (114–118)

NEWBOT is a function which achieves the same result as does NEWTOP except that the datum is pushed up on the bottom of the list Q.

NXTLFT(P, A) (119–132)

NXTLFT inserts the datum P to the left of the list cell the machine address of which is specified by A. The value of the function is the machine address of the cell taken from LAVS for datum and linkage storage. (Machine addresses of list cells come to the "knowledge" of a program mainly via the data placing operations now being described and by access to the LPNTR of READERs which will be discussed below.)

NXTRGT(P, A) (133–146)

NXTRGT inserts the datum P to the right of the list cell the machine address of which is specified by A. Otherwise, it behaves just as does NXTLFT.

INLSTL(M, A) (147–160)

M must be the ALIAS of a list and A the machine address of a list cell. INLSTL takes the set of linked cells constituting the body of the list M, i.e., all but the HEADER of that list, and inserts it to the left of the cell A. It thus lengthens the list of which A is a member by whatever the length of M was. M is made into an empty list and its NAME delivered as the value of this function.

INLSTR(M, A) (161–174)

Functions just as does INLSTL except that the list M is inserted to the right of the list cell whose address is specified by A.

SUBSTP(P, M) (182–186)

SUBSBT(P, M) (187–191)

The datum P replaces the datum presently stored on top (bottom) of the list M. The value of this function is the datum replaced by it, i.e., the previous contents of the top (bottom) of the list M.

SUBST(P, M) (175–181)

The datum P replaces that stored in the list cell whose address is specified by A. The replaced datum is the value of this function.

4. The Reader Mechanism and the Traversal of Lists and List Structures via that Mechanism. Note: Many details concerning the internal functioning of the READER mechanism were discussed in the previous section. In this section the same apparatus and its operation is viewed entirely in terms of its effects as seen by the programmer. Familiarity with relevant definitions made in the previous section is assumed.

LRDROV(P) (396–402)

The function LRDROV "appoints" a READER for the list whose ALIAS is given as its input parameter. The value of the function is the address of the cell taken from LAVS, i.e., the cell which was actually put in the format of a READER. In its initial state, the READER is "pointing at" the HEADER of the list for which it was created. The function of a READER is analogous to that of a bookmark. Creating a READER for a list is somewhat like identifying a bookmark with a particular book, but not yet having it mark a specific page.

LPNTR(R) (415–419)

The input parameter R to the LPNTR function is the address of a READER. The value of the function is the machine address stored in the LPNTR field of that READER, i.e., the address of the cell to which that READER is currently pointing, delivered in the form of an integer.

LCNTR(R) (410–414)

Behaves as does LPNTR except that the LCNTR (level counter) of the READER is the value of the function.

LOFRDR(R) (403–409)

Behaves as does LPNTR except that the value of the function is the NAME of the list which the READER R is currently reading. This value is delivered in NAME format.

THE ADVANCE FUNCTIONS (212–374)

Linear Advances. There are six LINEAR advance function each with two input parameters; namely, the READER which is to be the instrument for the advance and a flag which is to be set to zero or non-zero depending on the outcome of the advance. As seen by the programmer, the task of the LINEAR advance operations is to do the following three things: (a) To cause the LPNTR of the READER to "advance" to a new position, i.e., to cause it to "point" to one after another of the list cells of which the READER is a READER as guided by the linking information contained in that list itself and to finally come to rest while pointing to a particular cell on the basis of a criterion supplied by the selected advance function. (b) To deliver as the value of the advance function the datum being pointed to by the final state of the LPNTR. (c) To set the flag, the name of which is the second input parameter to the advance function, to zero

if the advance operation came to rest on the basis of the criterion supplied, or to non-zero if the advance function came to rest while pointing at the list HEADER. In the latter case, the advance operation could not find a list cell having the properties specified as a termination criterion by the programmer.

The six LINEAR advance operations are:

ADVLWR(R, F)	Advance LINEAR *WORD RIGHT*
ADVLER(R, F)	Advance LINEAR *ELEMENT RIGHT*
ADVLNR(R, F)	Advance LINEAR *NAME RIGHT*
ADVLWL(R, F)	Advance LINEAR *WORD LEFT*
ADVLEL(R, F)	Advance LINEAR *ELEMENT LEFT*
ADVLNL(R, F)	Advance LINEAR *NAME LEFT*

The last letter of the function name determines the *direction* in which the advance is to proceed. (The terms "left" and "right" can be interpreted to mean "up" and "down", respectively, when convenient.) The next to the last letter of the function name (W, E, or N) determines the *target* for which the advance is to aim and therefore the advance termination criterion.

All list cells are, for the purpose of advance target identification, classified according to whether the datum they hold is a NAME, or not. Those which do hold a NAME (ID = 1) have the target designation "N", while all others are designated "E" (element). When the nature of the datum held by a cell is not an issue, the target specified is "W" (word). The class W is therefore the union of the classes N and E. The HEADER is, of course, separately identified.

Examples:

X = ADVLWR(R5, FLAG)

The LPNTR of the READER R5 will point to the next cell to the right of the list cell to which it was pointing when the operation was initiated. If that list cell is not the list HEADER, then the datum stored in that list cell will replace the contents of X and the flag FLAG will be set to non-zero (it may then be interrogated by an IF statement). If the cell finally pointed to is a list HEADER, FLAG will be set to nonzero and X to zero.

X = ADVLER(R5, FLAG)

The effect of this operation is identical to that of the previous example except that if the next list cell encountered by the READER contains a NAME as a datum, the cell to the right of *it* will be examined for the condition that it does not contain a NAME as a datum. The LPNTR will not stop until either an E cell is encountered, in which case it will terminate the advance operation while pointing at that E cell and deliver the datum stored in it as its value, or the list HEADER is finally pointed to, the FLAG set to nonzero, and zero delivered as the value of the function.

X = ADVLNL(R5, FLAG)

In this case the advance proceeds in the upward (to the left) direction beginning with the list cell at which the LPNTR is currently pointing. The LPNTR continues its search until either a list cell containing a NAME or a HEADER is encountered. If an N cell is found, then the contents of X are replaced by the NAME in NAME format and the FLAG set to zero. Otherwise, X = 0 and FLAG $\neq$ 0.

Structural Advances. The distinction between LINEAR and STRUCTURAL advances is that the latter permit the READER to *descend* and *ascend* in list structures, i.e., to traverse sublists as well as lists. The task of the STRUCTURAL advance operation, again as seen by the programmer, is therefore to do the three things done by the LINEAR operation and, in addition: (d) To modify the LCNTR of the READER such that it indicates the *level* within the list structure to which the READER has descended. (Zero indicates main list.) (e) To update, if required, that field of the READER which is retrievable by the LOFRDR function such that it always contains the name of the list into which the READER is pointing.

The six STRUCTURAL advance operations are:

ADVSWR(R, F)	Advance STRUCTURAL *WORD RIGHT*
ADVSER(R, F)	Advance STRUCTURAL *ELEMENT RIGHT*
ADVSNR(R, F)	Advance STRUCTURAL *NAME RIGHT*
ADVSWL(R, F)	Advance STRUCTURAL *WORD LEFT*
ADVSEL(R, F)	Advance STRUCTURAL *ELEMENT LEFT*
ADVSNL(R, F)	Advance STRUCTURAL *NAME LEFT*

These advance operations proceed as follows: The ID of the cell currently being pointed to by the LPNTR of the READER is interrogated to see if it signifies that that cell is of the N class. If it is, then the READER becomes a READER of the list the NAME of which is stored in that cell and the LCNTR of the READER is counted up by one. That field of the READER which holds the name of the list being read is, of course, also suitably modified. If the cell currently being pointed to by the LPNTR is not of the N class, then the next cell (left or right, depending on which function is operative) is examined for correspondence to the target criterion. That next cell is therefore a *candidate* of the advance.

The candidate may correspond to the termination criterion specified by the advance function. In that case, the advance terminates in exactly the way LINEAR advances do, i.e., the value of the function is the datum contained in the final candidate cell and the FLAG is set to zero. If the candidate does not correspond to the termination criterion and is not a HEADER, then the whole

process begins again as if the advance operation had just been initiated.

If the candidate is a HEADER, the LCNTR of the READER determines whether the advance terminates by the failure criterion, i.e. flag set to non-zero, or continues. The condition indicated by the LCNTR being zero is that the READER is reading the main list. The encounter of a HEADER under that circumstance therefore means that the end of the entire list structure has been encountered. The target specified by the programmer could not be found. If the LCNTR is not zero and a HEADER is encountered in the course of an advance, then that HEADER is merely an indication that the end of a sublist has been encountered. The READER therefore "ascends" from that sublist to that list cell on the higher order list which led into the sublist initially. The next candidate is then determined LINEARLY for one step, but proceeds, if at all, STRUCTURALLY from then on. Of course, the LCNTR is decremented by one in the ascension and the name of the list to which the READER has ascended recorded in the appropriate READER field.

In order to make the discussion of an example possible, the following list structure is hypothesized:

L1: (1, 2, 3, (41, (421, 422), 43), 5, (61, 62), 7)

If names were to be given to the sublists indicated, then another description of the list structure L1 would be:

L1: (1, 2, 3, L4, 5, L6, 7)

i.e., L1 is a list structure with elements 1, 2, 3, 5, and 7 and the two sublists L4 and L6 on it, where

L4: (41, L42, 43)

and

L42: (421, 422)

and

L6: (61, 62).

Examples:

Suppose the READER R1 had just been "appointed" to the list structure L1 (statement: R1 = LRDROV(L1)). It is then in its initial state. The effect of the statement

M = ADVSWL(R1, F)

is that M would be set to 7 and F to zero. The LPNTR of R1 is left pointing to the bottom of the list L1, i.e., to the element "7". If the same statement were to be executed again, M would be left containing the NAME L6 and the flag F again set to zero. One more execution would leave M = 62, while two more executions beyond that would leave M = 5.

Again suppose the READER to be in its initial state. The effect of the statement:

M = ADVSNR(R1, F)

is that M is left containing the NAME L4. If executed once more, the result would be that M = NAME L42. Repeated executions would produce the sequence M = NAME L6 and (finally) M = 0 with F $\neq$ 0.

If, with the READER starting in its initial state, the statement

M = ADVSER(R1, F)

were to be executed repeatedly, M would successively take on the values 1, 2, 3, 5, 7, 0 with F being zero except in the last instance.

LVLRVT(R) (420–430)

The objective of the function LVLRVT is to cause the READER which is its input parameter to ascend (reverse) in the list structure into which it is currently pointing until it is again pointing into the (main) list for which it was originally appointed. If the READER is already pointing to within the main list, nothing happens. Otherwise, the LPNTR of the READER is left pointing to that cell in the main list which contains the NAME of the substructure into which the READER was pointing when this operation was initiated. The value of the function is the address of the READER. Consequently, this function may serve as an input parameter to any other function requiring a READER as one of *its* parameters.

Example: If, in the example list structure displayed above, the READER RA is currently pointing to the element "421" and the function LVLRVT is invoked, then RA would be left pointing to the fourth word from the top of the list L1, i.e., the list cell containing the list NAME L4.

LVLRV1(R) (431–441)

This function behaves as does LVLRVT except that it causes the READER to ascend only one level, if at all.

Example: Under the same conditions as stated in the previous example, the LPNTR of the READER RA would come to rest pointing to the list cell containing the list NAME L42 on the list L4.

INITRD(R) (442–447)

This function causes the LPNTR of the READER which is its input parameter to point to the HEADER of whatever list the READER is currently pointing into. The value of the function is the address of the READER. Thus, to completely initialize a READER, independent of how deeply it may have penetrated a list structure, the statement

CALL INITRD(LVLRVT(R))

where R is the READER to be so initialized, would be invoked.

LRDRCP(R) (459–473)

The objective of this function is to make a copy of the READER which is its input parameter and to deliver the address of that copy as its value. The utility of LRDRCP lies in that the READER being copied carries with it a record of its history. The programmer may wish to leave this history intact but also to further advance the READER. The copy produced by this function may be advanced as if it were the original READER,

leaving the original intact. Conversely, the roles of the original and its copy may be interchanged.

REED(R) (375–379)

The input parameter to this function is a READER. The value of the function is the datum contained in the cell to which the LPNTR of the READER is currently pointing. The READER is not modified in any way. This function may be considered to be a "null" advance.

IRARDR(R) (448–458)

This function has the effect of erasing (restoring to LAVS) the READER which is its input parameter. The value of the function is the LCNTR of the READER, i.e., an integer which indicates how deeply within a list structure the erased READER was pointing prior to its erasure.

SEQUENCE READER

The SEQUENCE READER is a simpler version of the READER. The mechanisms associated with it make possible the traversal of a list or list structure in much the same way as those associated with the READER. The SEQUENCE READER does not, however, store historical information with respect to its descents into list structures and is therefore not able to ascend from a sublist to a main list. Its main application is to those procedures which require a list or list structure to be traversed sequentially without any requirement to return to higher order lists. Its principal advantages are that the SEQUENCE operations associated with it execute faster than the corresponding ADVANCE operations and that the SEQUENCE READER itself is not taken from LAVS and need therefore never be erased.

SEQRDR(LST)

The statement S = SEQRDR(LST) causes S to be a SEQUENCE READER of the list LST. That is, SEQRDR is a function the value of which is a single computer word in the SEQUENCE READER format.

SEQLR(S,F)

SEQLL(S,F)

These functions are equivalent to the functions ADVLWR and ADVLWL, respectively, insofar as the values produced are concerned. Repeated executions yield successive data terms of the list for which S is the SEQUENCE READER. The flag F is set to minus one if the datum located on the subject list is an ELEMENT (not a list NAME), to zero if it is a list NAME, and to one if a HEADER. The SEQ operation can therefore be followed by an IF statement of the form

IF (F) N_1, N_2, N_3

such that control is transferred to appropriate statements as a condition on the outcome of the SEQ operation.

SEQSL(S,F)

SEQSR(S,F)

These functions behave as do ADVS*E*L and ADVS*E*R, respectively, in the same sense as described above. The encounter of a HEADER *on any level* causes the flag F to be set to one, but no ascension within the list structure. The encounter of a list name causes descent into the structure, however.

5. Other Ways of Retrieving Information from Lists

TOP(L) (192–196)

The input parameter to the function TOP is the NAME of a list. Its value is the datum stored on the top (leftmost) cell on that list. The list is not modified by this operation.

BOT(L) (197–201)

This function behaves as does TOP except that the datum stored in the bottom cell of the named list is delivered as the value of the function.

POPTOP(L) (202–206)

The top cell of the list whose NAME is the input parameter to this function is "popped off" that list, i.e., returned to LAVS, and the datum contained therein delivered as the value of the function.

POPBOT(L) (207–211)

The bottom cell of the list whose NAME is the input parameter to this function is popped off that list and the datum contained therein delivered as the value of the function.

DELETE(A) (380–395)

The input parameter to this function is the machine address of a list cell. The list cell so indicated is deleted from the list on which it appears and the datum stored in that cell delivered as the value of the function. If (inadvertently) the address of a list HEADER is given as an input parameter, then the value of the function will be zero and an error message printed. The program will, however, continue to be executed.

NULSTL(A,L) (79–93)

The input parameters to this function are A, the address of a list cell on the list L, and the NAME of the list L. The objective of this function is to split the list L by creating a new list having all cells to the left (or above) the cell indicated by A, as well as the cell A, as its members. The cells thus associated with the newly created list are removed from the list L. The NAME of the newly created list is the value of this function.

NULSTR(A,L) (94–108)

This function is identical to the function NULSTL except that all cells to the right of (or below) the cell A, as well as the cell the address of which is A, are deleted from the list L and placed on the newly created list. The name of the newly created list is again the value of the function.

6. Tests

NAMTST(K) (508–517)

The datum in K is examined. If it is the NAME of a list, the value of the function is zero, otherwise the value of the function is nonzero.

LISTMT(L) (518–526)

The input parameter to this function is the NAME of a list. If that list is empty, the value of the function is zero, otherwise the function has the value of nonzero.

LSTEQL(LA,LB) (527–551)

The two input parameters to this function are both NAMEs of list structures. The objective of this function is to determine whether or not these list structures are equal. If they are found to be so, the value of the function is zero, otherwise it is non-zero. Two list structures are equal if they have identical structures, i.e., sublist NAMEs appearing in corresponding places within both structures, and if corresponding elements appearing in both structures are identical.

7. Miscellaneous Processes. A number of functions discussed above depend on their parameters being machine addresses of list cells, e.g., DELETE, NULSTL, NXTRGT, etc. Certain functions produce such machine addresses as their values, e.g., NXTRGT, NXTLFT, NEWTOP, NEWBOT. The following four functions are additional explicit ways of producing such machine addresses.

MADLFT(A) (494–500)

The input parameter A to this function is the machine address of a list cell or a HEADER. The value of the function is the machine address of the cell to the left of (above) that being specified by A. If that machine address should turn out to be that of a HEADER, then the value will be delivered in the format of a list NAME, otherwise as an integer.

MADRGT(A) (501–507)

This function has the identical behavior to the function MADLFT except that the machine address of the cell to the right (below) that specified by A is delivered as its value.

MADNTP(L,N) (474–483)

The input parameters to this function are, respectively, the NAME of a list L and an integer N. The objective of this function is to deliver the machine address of the *n*th list cell counting from the top of the list L. If that cell is the HEADER of the list L, then the value is delivered in the format of the list NAME L, otherwise the value is an integer. If N is greater than the number of cells on the list L (including the HEADER cell), then the effect of this function is as if N modulo M had been the second input parameter, where M is the actual number of cells (including the HEADER) on the list L.

MADNBT(L,N) (484–493)

This function is identical to MADNTP except that the machine address of the *n*th cell counting from the *bottom* of the list L is produced.

MTLIST(L) (55–68)

The list the NAME of which appears as an input parameter to this function is emptied, i.e., its cells restored to LAVS. The value of the function is the NAME of the empty list L.

IRALST(L) (69–78)

The list the NAME of which appears as an input parameter to this function is erased, i.e., all its cells, including its HEADER, are restored to LAVS by this function. However, this list (and its contents) may still be the sublist of another list. The erasure is not actually carried out if that condition is detected. (The reference counter of the list is merely counted down by one in that event.) The value of the function is an integer specifying of how many lists the subject list is a sublist. If that value is zero, the list has actually been erased.

LSSCPY(L) (552–569)

The input parameter to this function is a list NAME. The objective of this function is to create a list, the NAME of which is the value of the function, such that the newly created list is a copy of the input list (or list structure, of course).

PRESRV(N) (597–603)

The input to this function is an integer N (less than or equal to 100). This function "preserves" the first N of the 100 public lists W, i.e., W(1), W(2), . . ., W(N). To preserve a list, in the intended sense, means to push whatever datum is presently on its top on its top once more. The first two cells of a preserved list are therefore identical (as seen by the programmer).

RESTOR(N) (604–610)

The RESTOR subroutine undoes the work of the PRESRV subroutine, i.e., it pops up the first N public lists.

PARMT2(X, Y) (621–628)

This subroutine causes the input parameters specified to be pushed down on W(1) and W(2) respectively. Its utility is mainly in that it aids in the communication of parameters for recursion. It should also serve the programmer as a model for constructing subroutines PARMTn(X_1, X_2, . . ., X_n) for communicating n parameters through the public lists. (Many Fortran realizations will demand that n always be even.)

RCELL(A) (36–43)

This subroutine returns the cell the address of which is given by A to LAVS. The programmer should seldom have need to appeal to it.

NUCELL(DUMMY) (19–35)

This function of no parameter causes a cell to be taken from LAVS and its machine address delivered (in the form of an integer) as its value. The programmer should never have need to appeal to it.

MRKLST(M, L)

The objective of MRKLST is to place the mark 0, 1, 2, or 3 on the list L. In the sense of this function, all lists are marked with 0 initially. The value of the function is the NAME of the list L, i.e., the second input parameter.

MRKLSS(M, L)

This function is identical to the function MRKLST except that *every* list of the list structure L is given the mark M.

LSTMRK(L)

The input parameter to this function is the NAME of the list L. Its value is the mark on that list.

The following functions operate on description lists (sometimes called "attribute-value" lists). Any list may have a description list. Description lists become integral

parts of the host lists, not sublists 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.

NEWVAL(AT, VAL, L)

The objective of this function is to assign the value VAL to the attribute AT on the description list of the list L. VAL replaces whatever the previous value of AT may have been. If the attribute AT cannot be found, it is added (together with its value) to the description list. If no description list for the list L exists, one is created by this function and the attribute value pair placed on it. The value of the function is the old value of the attribute if there is one, otherwise zero.

NOATVL(AT, L)

This function removes the attribute AT and its value from the description list associated with the list L. The value of the function is the value associated with that attribute.

ITSVAL(AT, L)

This function produces the value of the attribute AT as stored on the description list of the list L as its value. If the attribute AT is not found, the value of the function is zero.

MTDLST(L)

This function empties the description list of the list L. Its value is the NAME of the list L.

NAMEDL(L)

The value of this function is the NAME of the description list of the list L which is its input parameter.

MAKEDL(L, M)

This function makes the list L a description list of the list M. The value of the function is the NAME of the list M.

8. RECURSION

VISIT(PLACE,PARMTn(X_1, X_2, . . ., X_n))

This function has the following effect: The parameters of the PARMTn subroutine are pushed down on the public lists W(1), W(2), . . ., W(n) and control is transferred to a statement which has been ASSIGNED the label PLACE. Control is then sequenced in the normal manner until the TERM statement is called. That statement determines the value of the VISIT function.

TERM(RESULT, RESTOR(n))

This function restores the n public lists W(1), W(2), . . ., W(n) and returns control to that part of the program which would have been executed immediately after the VISIT function, had the latter not effected a transfer of control. The value given to the VISIT function is the first parameter of TERM, i.e., RESULT in this formulation.

IV. Bit, Character, and Logical Operations

The following operations exist primarily to help overcome certain inconveniences caused by fixed/floating point naming conventions associated with FORTRAN.

EQUAL(A, B)

The value of this function is zero if the two parameters A and B are equal, otherwise not.

AND(A, B)

The value of this function is the logical product of the two operands A and B.

OR(A, B)

The value of this function is the logical union of the two operands A and B.

SETDIR(A, B)

This function has the effect B = A, but ignores fixed/floating point incompatibilities which might otherwise be inferred to exist on the basis of the parameter names. The value of the function is the input parameter A. (See discussion of PRIMITIVES, Section II.)

The following functions and subroutines provide bit and character manipulation abilities to SLIP.

SQOUT(FIELD, SOURCE)

The parameter field is an extract mask. The value of the function is that field extracted from SOURCE and shifted right as many times as there are zero bits to the right of the first one bit in the field definition. Leading zeros are introduced in the process of right justification. (In the canonical version of SLIP, a set of "character masks" are prestored as part of the system. Their labels are CP(1), CP(2), etc., where CP(1) consists of six high order one bits.)

SQIN(FIELD, DATUM, DEST)

The specified datum is shifted to the left by the number of trailing zeros in the field specification and inserted in the destination word field specified by the FIELD parameter.

SHIN(N, DATUM, DEST)

SHIN causes the DESTination word to be shifted left N bits and the specified DATUM to be inserted in the vacated low order portion of that word.

LANORM(WORD)

If alphanumeric characters are introduced into a word by means of SHIN, where that word initially contained all blanks, then the high order characters of that word may be blanks. LANORM ring shifts a word until the highest order character of the result is a nonblank. The value of the function is a word so shifted. The word specified as an input parameter is left unmodified.

V. Discussion

List processing has won a number of dedicated converts. Some have, however, become somewhat too fervent in their advocacy of list processing. While there may be some programming tasks which are best solved entirely within some list processing system, most tasks coming to the ordinary programmer require the application of a number of distinct techniques. The packaging of a variety of tools within a single tool box appears to be a good, if not an optimum, way of outfitting a worker setting out to solve complex problems. FORTRAN, ALGOL, and other languages of the same type provide excellent vehicles for

such provisioning. Apart from the fact that they are very powerful in themselves, they have the advantage that they are well known. The task of coming to grips with these new techniques is then that of adding to a vocabulary of an already assimilated language rather than that of learning an entirely new one. Furthermore, most FORTRAN-like systems make it possible for the programmer to pay (in space) for only those augmenting functions he actually uses in a specific program. Thus, in SLIP, a programmer may use only the pushdown and popup features of the list processing system. Then, obviously, the advance operations, for example, would not even be loaded into his program.

The system which obviously invites comparison with SLIP is Gelernter's FLPL. In this connection, it is interesting to note again that SLIP was first written as a set of machine language subroutines such as NUCELL, ADVLL, etc., and that the idea of rewriting the entire system in terms of the primitives displayed before occurred later and then only in terms of documentation. Apart from the fact that SLIP is symmetric while FLPL is not, what are the basic differences between the two systems? Even stronger, what advantages does SLIP have which justifies its acceptance?

One of the most basic distinctions between the two systems is that SLIP consists of a set of machine language subroutines and functions which, although they can be represented in terms of primitives, are not themselves primitive in any sense, while FLPL, as presented to the general user, is essentially a set of primitives out of which higher level functions may be built to suit the special application. The FLPL user is advised to "streamline" frequently used functions and subroutines. He ignores this advice at the risk of generating quite slow programs. The speed advantage which the hand-coded version of SLIP enjoys over the primitive based system can, with considerable justification, be attributed to built-in "streamlining". The fact that SLIP is presented to the user together with the primitives which can be used to generate it, means, in addition, that the SLIP user is not robbed of any degrees of freedom which might have been available to him in FLPL.

That part of the SLIP organization which was taken directly from the older KLS system is also thought to contribute to an overall edge of efficiency of SLIP over FLPL. The FLPL function XLASLCF(J) which "searches down the list . . .J, (and has as its) value the address of the last *l*-word on J as determined by its zero decrement", clearly requires an execution time which is a function of the length of the list J. Furthermore, every *l*-word on J must be interrogated for a zero decrement field and, that test failing, its link to the next *l*-word extracted. It is a nontrivial function from the point of view of execution time. In SLIP, the address of the bottom cell of a list is stored in the LNKR field of its HEADER. The length of time required to retrieve it is therefore independent of the length of the list, and the retrieval operations themselves are few and linear. The most important point about the direct accessibility of the top and bottom pointers for any list is, however, that lists may be "bulk erased" as already described.

One other consequence of the fact that FLPL is primitive based is that much of the burden of awareness of what's going on remains on the shoulders of the programmer. The programmer must decide explicitly what information (e.g. ID) is to be placed in what field on an *l*-word (1, p. 97). He must presumably also test the "ID" of a word retrieved from a list if he wishes to know its nature. Similarly, indexing is a "manual", i.e., directly programmed, operation. The ADVANCE operations of SLIP take over much of this burden. Of course, SLIP has eliminated the idea of "borrowed" data entirely.

The symmetry properties of SLIP obviously distinguish it from FLPL. Unfortunately, the advantages of symmetry cannot be displayed without a thorough analysis of a detailed example. An appeal to the intuition and experience of the reader may, however, not be out of place here since he may well have seen other systems which acquired a whole new order of elegance through the introduction of symmetry.

SLIP, like KLS, overcomes one of the principal difficulties of Threaded Lists; namely, that a list may be a sublist of only one list.

Finally, SLIP viewed as a set of machine language subroutines may be imbedded in any language having the ability to call such subroutines. (In many such languages, restrictions imposed by naming conventions disappear, thus making the system even more attractive.) SLIP is, at least in a limited sense, both machine and host language independent.

Acknowledgment. As is pointed out in the discussion, SLIP owes a considerable debt to previous list processing systems. Certain of its features are, however, more the result of attempts to build a symbol manipulator for the use of behavioral scientists than to generalize other processors. In this connection, the continuing and generous advice and support of Kenneth Colby, M.D., of Stanford University and of Dr. Edward Feigenbaum of the University of California at Berkeley is gratefully acknowledged. The author also wishes to thank Howard Sturgis of the University of California at Berkeley and Larry Breed of Stanford University for their parts in making the system operative on the computers at their respective computation centers.

REFERENCES

1. GELERNTER, H., et. al. A FORTRAN-compiler list-processing language. *J. ACM 7*, 2 (Apr. 1960).
2. NEWELL, A., et. al. Information Processing Language V Manual, Sect. I and II. Rand Corp., P1918 (Mar. 1960).
3. PERLIS, A. J., et. al. Symbol manipulation by threaded lists. *Comm. ACM 3* (1960), 195–204.
4. WEIZENBAUM, J., Knotted list structures. *Comm. ACM 5* (1962), 161–165.

APPENDIX

```
      LISTING OF SLIP SYMBOLIC DECK
      JOSEPH WEIZENBAUM - GENERAL ELECRIC COMPUTER LABORATORY
                          P.O. BOX 1285
                          SUNNYVALE, CALIFORNIA
```

```
      SUBROUTINE INITAS(M,N)                                              001
C                                                          4/1/63         002
      COMMON AVSL,X(100)                                                  003
      DIMENSION W(200),M(2)                                               004
      DO 1 I=1,100                                                        005
         J=2*I-1                                                          006
      CALL STRDIR(STRDIR(STRDIR(0,X(I)),W(J)),W(J+1))                     007
      CALL SETDIR(0,MADOV(W(J)),MADOV(W(J)),X(I))                         008
      CALL SETDIR(2,MADOV(W(J)),MADOV(W(J)),W(J))                         009
    1 CALL SETDIR(-1,-1,4095,W(J+1))                                      010
      DO 2 I=1,N                                                          011
    2    M(I) = 0                                                         012
         K = N-2                                                          013
      DO 3 I=1,K,2                                                        014
    3 CALL SETDIR(-1,-1,MADOV(M(I+2)),M(I))                               015
      CALL SETDIR(0,MADOV(M(N-1)),MADOV(M(1)),AVSL)                       016
      RETURN                                                              017
      END                                                                 018
      FUNCTION NUCELL(X)                                                  019
C                                                          4/1/63         020
      COMMON AVSL                                                         021
         M = LNKR(AVSL)                                                   022
         IF (M)1,2,1                                                      023
    2 PRINT 901                                                           024
      STOP                                                                025
    1 IF (ID(CONT(M))-1)3,4,3                                             026
    4     CALL IRALST(CONT(M+1))                                          027
    3     CALL SETDIR(-1,-1,LNKR(CONT(M)),AVSL)                           028
          CALL STRIND(0,M)                                                029
          CALL STRIND(0,M+1)                                              030
         NUCELL = M                                                       031
      RETURN                                                              032
  901 FORMAT (1H1,6X,55HLIST OF AVAILABLE SPACE EXHAUSTED - PROGRAM TERM  033
     1INATED  )                                                           034
      END                                                                 035
      SUBROUTINE RCELL(CELL)                                              036
C                                                          4/1/63         037
      COMMON AVSL                                                         038
      CALL SETIND(-1,-1,CELL,LNKL(AVSL))                                  039
      CALL SETDIR(-1,CELL,-1,AVSL)                                        040
      CALL SETIND(-1,-1,0,CELL)                                           041
      RETURN                                                              042
      END                                                                 043
      FUNCTION LIST(K)                                                    044
C                                                          4/1/63         045
C                                                          4/1/63         046
         LIST = NUCELL(Z)                                                 047
      CALL SETDIR(0,LIST,LIST,LIST)                                       048
      CALL SETIND(2,LIST,LIST,LIST)                                       049
         IF (K-9)2,1,2                                                    050
    2 CALL SETIND(-1,-1,1,LIST+1)                                         051
         K = LIST                                                         052
    1 RETURN                                                              053
```



```
      END                                                                 054
      FUNCTION MTLIST(P)                                                  055
C                                                          4/1/63         056
      COMMON AVSL                                                         057
         M = LOCT(P)                                                      058
         IF (LISTMT(P))3,4,3                                              059
    3    LR = LNKR(CONT(M))                                               060
         LL = LNKL(CONT(M))                                               061
         CALL SETIND(-1,M,M,M)                                            062
         CALL SETIND(-1,-1,LR,LNKL(AVSL))                                 063
      CALL SETDIR(-1,LL,-1,AVSL)                                          064
         CALL SETIND(-1,-1,0,LNKL(AVSL))                                  065
    4    MTLIST = M                                                       066
      RETURN                                                              067
      END                                                                 068
      FUNCTION IRALST(P)                                                  069
C                                                          4/1/63         070
        L = LOCT(P)                                                       071
      CALL SETIND(-1,-1,LCNTR(L)-1,L+1)                                   072
         IRALST = LCNTR(L)                                                073
        IF(IRALST)1,2,1                                                   074
    2 CALL MTLIST(P)                                                      075
         N = LNKL(CONT(L+1))
           IF(N)3,4,3
    3      NEW = NUCELL(Z)
      CALL SETIND(1,-1,-1,NEW)
      CALL SETIND(-1,N,N,NEW+1)
      CALL RCELL(NEW)
    4 CALL RCELL(L)                                                       076
    1 RETURN                                                              077
      END                                                                 078
      FUNCTION NULSTL(LNKP,LNKH)                                          079
C                                                          4/1/63         080
      NULSTL = LIST(9)                                                    081
      IF (ID(CONT(LNKP))-2)1,2,1                                          082
    2    CALL SETIND(2,NULSTL,NULSTL,NULSTL)                              083
      RETURN                                                              084
    1 LTOP = LNKR(CONT(LNKH))                                             085
      LSUC = LNKR(CONT(LNKP))                                             086
         CALL SETIND(-1,-1,LSUC,LNKH)                                     087
         CALL SETIND(-1,LNKH,-1,LSUC)                                     088
         CALL SETIND(2,LNKP,LTOP,NULSTL)                                  089
         CALL SETIND(-1,-1,NULSTL,LNKP)                                   090
         CALL SETIND(-1,NULSTL,-1,LTOP)                                   091
      RETURN                                                              092
      END                                                                 093
      FUNCTION NULSTR(LNKP,LNKH)                                          094
C                                                          4/1/63         095
      NULSTR = LIST(9)                                                    096
      IF (ID(CONT(LNKP))-2)1,2,1                                          097
    2    CALL SETIND(2,NULSTR,NULSTR,NULSTR)                              098
      RETURN                                                              099
    1 LBOT = LNKL(CONT(LNKH))                                             100
      LPRE = LNKL(CONT(LNKP))                                             101
         CALL SETIND(-1,LPRE,-1,LNKH)                                     102
         CALL SETIND(-1,-1,LNKH,LPRE)                                     103
         CALL SETIND(2,LBOT,LNKP,NULSTR)                                  104
         CALL SETIND(-1,NULSTR,-1,LNKP)                                   105
         CALL SETIND(-1,-1,NULSTR,LBOT)                                   106
      RETURN                                                              107
```

```
      END                                                                  108
      FUNCTION NEWTOP(P,Q)                                                 109
C                                                          4/1/63         110
         NEWTOP = NXTRGT(P,LOCT(Q))                                        111
      RETURN                                                               112
      END                                                                  113
      FUNCTION NEWBOT(P,Q)                                                 114
C                                                          4/1/63         115
         NEWBOT = NXTLFT(P,LOCT(Q))                                        116
      RETURN                                                               117
      END                                                                  118
      FUNCTION NXTLFT(M,A)                                                 119
C                                                          4/1/63         120
         IL = NUCELL(Z)                                                    121
      NXTLFT = IL                                                          122
        LL = LNKL(CONT(A))                                                 123
      CALL SETIND(-1,-1,IL,LL)                                             124
      CALL SETIND(-1,IL,-1,A)                                              125
      CALL SETIND(0,LL,A,IL)                                               126
         IF(NAMTST(M))1,2,1                                                127
    2 CALL SETIND(1,-1,-1,IL)                                              128
      CALL SETIND(-1,-1,LCNTR(M)+1,M+1)                                    129
    1 CALL STRIND(M,IL+1)                                                  130
      RETURN                                                               131
      END                                                                  132
      FUNCTION NXTRGT(M,A)                                                 133
C                                                          4/1/63         134
         IR = NUCELL(Z)                                                    135
      NXTRGT = IR                                                          136
        LR = LNKR(CONT(A))                                                 137
      CALL SETIND(-1,IR,-1,LR)                                             138
      CALL SETIND(-1,-1,IR,A)                                              139
      CALL SETIND (0,A,LR,IR)                                              140
         IF (NAMTST(M))1,2,1                                               141
    2 CALL SETIND(1,-1,-1,IR)                                              142
      CALL SETIND(-1,-1,LCNTR(M)+1,M+1)                                    143
    1 CALL STRIND(M,IR+1)                                                  144
      RETURN                                                               145
      END                                                                  146
      FUNCTION INLSTL(M,N)                                                 147
C                                                          4/1/63         148
         L = LOCT(M)                                                       149
         ITOP = LNKR(CONT(L))                                              150
         IBOT = LNKL(CONT(L))                                              151
      INLSTL = L                                                           152
      CALL SETIND(-1,L,L,L)                                                153
         IPRE = LNKR(CONT(N))                                              154
      CALL SETIND(-1,IBOT,-1,N)                                            155
      CALL SETIND(-1,-1,ITOP,IPRE)                                         156
      CALL SETIND(-1,IPRE,-1,ITOP)                                         157
      CALL SETIND(-1,-1,N,IBOT)                                            158
      RETURN                                                               159
      END                                                                  160
      FUNCTION INLSTR(M,N)                                                 161
C                                                          4/1/63         162
         L = LOCT(M)                                                       163
         ITOP = LNKR(CONT(L))                                              164
         IBOT = LNKL(CONT(L))                                              165
      INLSTR = L                                                           166
      CALL SETIND(-1,L,L,L)                                                167
```



```
         ISUC = LNKR(CONT(N))                                              168
      CALL SETIND(-1,-1,ITOP,N)                                            169
      CALL SETIND(-1,IBOT,-1,ISUC)                                         170
      CALL SETIND(-1,N,-1,ITOP)                                            171
      CALL SETIND(-1,-1,ISUC,IBOT)                                         172
      RETURN                                                               173
      END                                                                  174
      FUNCTION SUBST(D,N)                                                  175
C                                                          4/1/63         176
         LBACK = LNKL(CONT(N))                                             177
         SUBST = DELETE(N)                                                 178
         CALL NXTRGT(D,LBACK)                                              179
      RETURN                                                               180
      END                                                                  181
      FUNCTION SUBSTP(DAT,LST)                                             182
C                                                          4/1/63         183
         SUBSTP = SUBST(DAT,LNKR(CONT(LST)))                               184
      RETURN                                                               185
      END                                                                  186
      FUNCTION SUBSBT(DAT,LST)                                             187
C                                                          4/1/63         188
         SUBSBT = SUBST(DAT,LNKL(CONT(LST)))                               189
      RETURN                                                               190
      END                                                                  191
      FUNCTION TOP(P)                                                      192
C                                                          4/1/63         193
      TOP = CONT( LNKR(CONT(LOCT(P)))+1)                                   194
      RETURN                                                               195
      END                                                                  196
      FUNCTION BOT(P)                                                      197
C                                                          4/1/63         198
      BOT = CONT( LNKL(CONT(LOCT(P)))+1)                                   199
      RETURN                                                               200
      END                                                                  201
      FUNCTION  POPTOP(P)                                                  202
C                                                          4/1/63         203
      POPTOP =  DELETE(LNKR(CONT(LOCT(P))))                                204
        RETURN                                                             205
        END                                                                206
      FUNCTION  POPBOT(P)                                                  207
C                                                          4/1/63         208
      POPBOT =  DELETE(LNKL(CONT(LOCT(P))))                                209
      RETURN                                                               210
      END                                                                  211
      FUNCTION ADVLL(LR,J,K)                                               212
C                                                          4/1/63         213
       CLR = CONT(LR)                                                      214
    5    LK = LNKL(CONT(LNKL(CLR)))                                       215
         CAND = CONT(LK)                                                   216
      CALL SETDIR(-1,LK,-1,CLR)                                            217
        IF (ID(CAND)-2)1,2,1                                               218
    1   IF (ID(CAND)-J)3,4,3                                               219
    3   IF (ID(CAND)-K)5,4,5                                               220
    4 ADVLL = 0.                                                           221
      GOTO 6                                                               222
    2 ADVLL = -1.0                                                         223
    6 CALL STRIND(CLR,LR)                                                  224
      RETURN                                                               225
      END                                                                  226
      FUNCTION ADVLR(LR,J,K)                                               227
```

```
      CLR = CONT(LR)
    5 LK = LNKR(CONT(LNKL(CLR)))
      CAND = CONT(LK)
      CALL SETDIR(-1,LK,-1,CLR)
        IF (ID(CAND)-2)1,2,1
    1   IF (ID(CAND)-J)3,4,3
    3   IF (ID(CAND)-K)5,4,5
    4 ADVLR = 0.
      GOTO 6
    2 ADVLR = -1.0
    6 CALL STRIND(CLR,LR)
      RETURN
      END
      FUNCTION ADVSR(L,J,K)
        R = CONT(L)
        CAND = CONT(LNKL(R))
        IF (ID(CAND)-1)1,6,1
    1 LCP = LNKR(CAND)
      CALL SETDIR(-1,LCP,-1,R)
      CAND = CONT(LCP)
        IF (ID(CAND)-2)3,4,3
    3     IF(ID(CAND)-J)7,8,7
    7     IF(ID(CAND)-K)5,8,5
    5     IF(ID(CAND)-1)1,6,1
    6 M=NUCELL(Z)
      CALL STRIND(R,M)
      CALL STRIND(CONT(L+1),M+1)
      CALL SETIND(-1,INHALT(LCP+1),LCNTR(L)+1,L+1)
      CALL SETDIR(-1,-1,M,R)
      CAND = CONT(INHALT(LNKL(R)+1))
      GOTO 1
    4 IF (LCNTR(L))9,10,9
   10 ADVSR = -1.0
      GOTO 12
    9 LK = LNKR(R)
        R = CONT(LK)
      CALL STRIND(CONT(LK+1),L+1)
      CAND = CONT(LNKL(R))
      CALL RCELL(LK)
      GOTO 1
    8 ADVSR = 0.
   12 CALL STRIND(R,L)
      RETURN
      END
      FUNCTION ADVSL(L,J,K)
        R = CONT(L)
        CAND = CONT(LNKL(R))
        IF(ID(CAND)-1)1,6,1
    1 LCP = LNKL(CAND)
      CALL SETDIR(-1,LCP,-1,R)
      CAND = CONT(LCP)
        IF(ID(CAND)-2)3,4,3
    3     IF(ID(CAND)-J)7,8,7
    7     IF(ID(CAND)-K)5,8,5
    5     IF(ID(CAND)-1)1,6,1
    6 M = NUCELL(Z)
      CALL STRIND(R,M)
      CALL STRIND(CONT(L+1),M+1)
      CALL SETIND(-1,INHALT(LCP+1),LCNTR(L)+1,L+1)
      CALL SETDIR(-1,-1,M,R)
```



```
      CAND = CONT(INHALT(LNKL(R)+1))
      GOTO 1
    4 IF (LCNTR(L))9,10,9
   10 ADVSL = -1.0
      GOTO 12
    9 LK = LNKR(R)
        R = CONT(LK)
      CALL STRIND(CONT(LK+1),L+1)
      CAND = CONT(LNKL(R))
      CALL RCELL(LK)
      GOTO 1
    8 ADVSL = 0.
   12 CALL STRIND(R,L)
      RETURN
      END
      FUNCTION ADVLNR(LR,A)
      A = ADVLR(LR,1,1)
      IF(A)1,2,1
    2 ADVLNR = REED(LR)
    1 RETURN
      END
      FUNCTION ADVLER(LR,A)
      A = ADVLR(LR,0,0)
      IF(A)1,2,1
    2 ADVLER = REED(LR)
    1 RETURN
      END
      FUNCTION ADVLWR(LR,A)
      A = ADVLR(LR,1,0)
      IF(A)1,2,1
    2 ADVLWR = REED(LR)
    1 RETURN
      END
      FUNCTION ADVSNR(LR,A)
      A = ADVSR(LR,1,1)
      IF(A)1,2,1
    2 ADVSNR = REED(LR)
    1 RETURN
      END
      FUNCTION ADVSER(LR,A)
      A = ADVSR(LR,0,0)
      IF(A)1,2,1
    2 ADVSER = REED(LR)
    1 RETURN
      END
      FUNCTION ADVSWR(LR,A)
      A = ADVSR(LR,1,0)
      IF(A)1,2,1
    2 ADVSWR = REED(LR)
    1 RETURN
      END
      FUNCTION ADVLNL(LR,A)
      A = ADVLL(LR,1,1)
      IF(A)1,2,1
    2 ADVLNL = REED(LR)
    1 RETURN
      END
      FUNCTION ADVLEL(LR,A)
      A = ADVLL(LR,0,0)
      IF(A)1,2,1
```

```
    2 ADVLEL = REED(LR)                                                    348
    1 RETURN                                                               349
      END                                                                  350
      FUNCTION ADVLWL(LR,A)                                                351
      A = ADVLL(LR,1,0)                                                    352
      IF(A)1,2,1                                                           353
    2 ADVLWL = REED(LR)                                                    354
    1 RETURN                                                               355
      END                                                                  356
      FUNCTION ADVSNL(LR,A)                                                357
      A = ADVSL(LR,1,1)                                                    358
      IF(A)1,2,1                                                           359
    2 ADVSNL = REED(LR)                                                    360
    1 RETURN                                                               361
      END                                                                  362
      FUNCTION ADVSEL(LR,A)                                                363
      A = ADVSL(LR,0,0)                                                    364
      IF(A)1,2,1                                                           365
    2 ADVSEL = REED(LR)                                                    366
    1 RETURN                                                               367
      END                                                                  368
      FUNCTION ADVSWL(LR,A)                                                369
      A = ADVSL(LR,1,0)                                                    370
      IF(A)1,2,1                                                           371
    2 ADVSWL = REED(LR)                                                    372
    1 RETURN                                                               373
      END                                                                  374
      FUNCTION REED(K)                                                     375
C                                                          4/1/63          376
      REED = CONT(LNKL(CONT(K))+1)                                         377
      RETURN                                                               378
      END                                                                  379
      FUNCTION DELETE(K)                                                   380
C                                                          4/1/63          381
        IF(ID(CONT(K))-2)1,2,1                                             382
    2 PRINT 901                                                            383
        DELETE = 0.                                                        384
      RETURN                                                               385
  901 FORMAT (1H1,98HAN ATTEMPT HAS BEEN MADE TO DELETE A HEADER - ZERO    386
     1HAS BEEN DELIVERED AND THE PROGRAM CONTINUED. )                      387
    1  DELETE = CONT(K+1)                                                  388
        LL = LNKL(CONT(K))                                                 389
        LR = LNKR(CONT(K))                                                 390
      CALL RCELL(K)                                                        391
      CALL SETIND(-1,-1,LR,LL)                                             392
      CALL SETIND(-1,LL,-1,LR)                                             393
      RETURN                                                               394
      END                                                                  395
      FUNCTION LRDROV(P)                                                   396
C                                                          4/1/63          397
        LRDROV = NUCELL(Z)                                                 398
      CALL SETIND(3,LOCT(P),0,LRDROV)                                      399
      CALL SETIND(0,P,0,LRDROV+1)                                          400
      RETURN                                                               401
      END                                                                  402
      FUNCTION LOFRDR(K)                                                   403
C                                                          4/1/63          404
      L = LNKL(CONT(K+1))                                                  405
      CALL SETDIR(0,L,L,L)                                                 406
      LOFRDR = L                                                           407
```



```
      RETURN                                                               408
      END                                                                  409
      FUNCTION LCNTR(K)                                                    410
C                                                          4/1/63          411
      LCNTR = LNKR(CONT(K+1))                                              412
      RETURN                                                               413
      END                                                                  414
      FUNCTION LPNTR(K)                                                    415
C                                                          4/1/63          416
      LPNTR =  LNKL(CONT(K))                                               417
      RETURN                                                               418
      END                                                                  419
      FUNCTION LVLRVT(K)                                                   420
C                                                          4/1/63          421
        LVLRVT = K                                                         422
    1   IF (CONT(LVLRVT+1))2,3,2                                           423
    3 RETURN                                                               424
    2   L = LNKR(CONT(LVLRVT))                                             425
      CALL    STRIND(CONT(L),LVLRVT)                                       426
      CALL    STRIND(CONT(L+1),LVLRVT+1)                                   427
      CALL    RCELL(L)                                                     428
      GOTO 1                                                               429
      END                                                                  430
      FUNCTION LVLRV1(K)                                                   431
C                                                          4/1/63          432
        LVLRV1 = K                                                         433
        IF (CONT(LVLRV1+1))2,3,2                                           434
    3 RETURN                                                               435
    2   L = LNKR(CONT(LVLRV1))                                             436
      CALL    STRIND(CONT(L),LVLRV1)                                       437
      CALL    STRIND(CONT(L+1),LVLRV1+1)                                   438
      CALL    RCELL(L)                                                     439
      RETURN                                                               440
      END                                                                  441
      FUNCTION INITRD(K)                                                   442
C                                                          4/1/63          443
      CALL SETIND(-1,LNKL(K+1),-1,K)                                       444
      INITRD = K                                                           45
      RETURN                                                               446
      END                                                                  447
      FUNCTION IRARDR(K)                                                   448
C                                                          4/1/63          449
      IRARDR = LCNTR(K)                                                    450
        M = K                                                              451
    3   N = LNKR(CONT(M))                                                  452
        CALL RCELL(M)                                                      453
        IF (N)1,2,1                                                        454
    1   M = N                                                              455
        GOTO 3                                                             456
    2 RETURN                                                               457
      END                                                                  458
      FUNCTION LRDRCP(K)                                                   459
C                                                          4/1/63          460
        LRDRCP = NUCELL(Z)                                                 461
        NEWR = LRDRCP                                                      462
        NOW  = K                                                           463
    3 CALL STRIND(CONT(NOW),NEWR)                                          464
      CALL STRIND(CONT(NOW+1),NEWR+1)                                      465
        NOW = LNKR(CONT(NOW))                                              466
          IF (NOW)1,2,1                                                    467
```

```
 1      NEW = NUCELL(Z)                                                  468
       CALL SETIND(-1,-1,NEW,NEWR)                                      469
          NEWR = NEW                                                    470
          GOTO 3                                                        471
   2  RETURN                                                           472
       END                                                              473
       FUNCTION MADNTP(P,N)                                             474
C                                                          4/1/63       475
          L = LOCT(P)                                                   476
       DO 1 I=1,N                                                       477
   1  L = LNKR(CONT(L))                                                478
       IF(ID(CONT(L))-2)2,3,2                                           479
   3  CALL SETDIR(0,L,L,L)                                             480
   2  MADNTP = L                                                       481
       RETURN                                                           482
       END                                                              483
       FUNCTION MADNBT(P,N)                                             484
C                                                          4/1/63       485
          L = LOCT(P)                                                   486
       DO 1 I=1,N                                                       487
   1  L = LNKL(CONT(L))                                                488
       IF(ID(CONT(L))-2)2,3,2                                           489
   3  CALL SETDIR(0,L,L,L)                                             490
   2  MADNBT=L                                                         491
       RETURN                                                           492
       END                                                              493
       FUNCTION MADLFT(K)                                               494
C                                                          4/1/63       495
       MADLFT = LNKL(CONT(K))                                           496
       IF(ID(CONT(MADLFT))-2)1,2,1                                      497
   2  CALL SETDIR(0,MADLFT,MADLFT,MADLFT)                              498
   1  RETURN                                                           499
       END                                                              500
       FUNCTION MADRGT(K)                                               501
C                                                          4/1/63       502
       MADRGT = LNKR(CONT(K))                                           503
       IF(ID(CONT(MADRGT))-2)1,2,1                                      504
   2  CALL SETDIR(0,MADRGT,MADRGT,MADRGT)                              505
   1  RETURN                                                           506
       END                                                              507
       FUNCTION NAMTST(K)                                               508
C                                                          4/1/63       509
       IF (LNKL(K)-LNKR(K))1,4,1                                        510
   4  IF (ID(CONT(K))-2)1,2,1                                          511
   2  IF(CONT(LNKR(CONT(LNKL(CONT(K)))))-CONT(K))1,3,1                 512
   3    NAMTST = 0                                                     513
        RETURN                                                          514
   1    NAMTST = -1                                                    515
        RETURN                                                          516
       END                                                              517
       FUNCTION LISTMT(P)                                               518
C                                                          4/1/63       519
        L = LOCT(P)                                                     520
       IF (EQUAL(CONT(L),CONT(LNKR(CONT(L))))3,4,3                      521
   4   LISTMT = 0                                                       522
        RETURN                                                          523
   3   LISTMT = -1                                                      524
        RETURN                                                          525
        END                                                             526
       FUNCTION LSTEQL(LA,LB)                                           527
```



```
C                                                          4/1/63       528
       COMMON A,W(100)                                                  529
       ASSIGN 100 TO LOCO                                               530
       LSTEQL = INTGER(VISIT(LOCO,PARMT2(LRDROV(LA),LRDROV(LB))))       531
       RETURN                                                           532
 100     LRA = INTGER(TOP(W(1)))                                        533
       LRB = INTGER(TOP(W(2)))                                          534
   8     XA =  ADVLWR(LRA,KA)                                           535
         XB =  ADVLWR(LRB,KB)                                           536
           IF (KA)1,2,1                                                 537
   1       IF (KB)3,4,3                                                 538
   2       IF (KB)4,6,4                                                 539
   6     IF(EQUAL(XA,XB))7,8,7                                          540
   7     IF (NAMTST(XA))4,9,4                                           541
   9     IF (NAMTST(XB))4,10,4                                          542
  10 LSTEQL = INTGER(VISIT(LOCO,PARMT2(LRDROV(XA),LRDROV(XB))))         543
         IF (LSTEQL)4,100,4                                             544
   3  CALL RCELL(LRA)                                                  545
       CALL RCELL(LRB)                                                  546
       CALL TERM(0,RESTOR(2))                                           547
   4  CALL RCELL(LRA)                                                  548
       CALL RCELL(LRB)                                                  549
       CALL TERM(-1,RESTOR(2))                                          550
       END                                                              551
       FUNCTION LSSCPY(LA)                                              552
C                                                          4/1/63       553
       COMMON A,W(100)                                                  554
       ASSIGN 100 TO LOCO                                               555
       LSSCPY = INTGER(VISIT(LOCO,PARMT2(LRDROV(LA),LIST(9))))          556
       RETURN                                                           557
 100     LC = INTGER (TOP(W(2)))                                        558
         LR = INTGER(TOP(W(1)))                                         559
   5     X = ADVLWR(LR,K)                                               560
          IF (K)1,2,1                                                   561
   1  CALL RCELL(LR)                                                   562
       CALL TERM(LC,RESTOR(2))                                          563
   2      IF(NAMTST(X))3,4,3                                           564
   3        CALL NEWBOT(X,LC)                                          565
            GOTO 5                                                      566
   4  CALL NEWBOT(VISIT(LOCO,PARMT2(LRDROV(X),LIST(9))),TOP(W(2)))     567
            GOTO 100                                                    568
       END                                                              569
       FUNCTION LSTPRO(L,K)                                             570
C                                                          4/1/63       571
         NEXT = K                                                       572
   3     IF(LNKL(NEXT+1)-LNKR(L))1,2,1                                  573
   1     NEXT = LNKL(NEXT)                                              574
         IF (NEXT)3,4,3                                                 575
   2     LSTPRO = 0                                                     576
       RETURN                                                           577
   4     LSTPRO = -1                                                    578
       RETURN                                                           579
       END                                                              580
       FUNCTION LPURGE(LST)                                             581
C                                                          4/1/63       582
       K = LRDROV(LST)                                                  583
          LPURGE = 0                                                    584
   3  X = ADVSWR(K,J)                                                  585
   6     IF(J)1,2,1                                                     586
   1  IF (NAMTST(X))3,4,3                                              587
```

```
    4 IF (LSTPRO(X,K))3,5,3
    5    L = LPNTR(K)
      X = ADVLWR(K,J)
      CALL DELETE(L)
         LPURGE = LPURGE+1
      GOTO 6
    2 CALL IRARDR(K)
      RETURN
      END
      SUBROUTINE PRESRV(N)
C                                                              4/1/63
      COMMON AVSL,W(100)
      DO 1 I=1,N
    1 CALL NEWTOP(TOP(W(I)),W(I))
      RETURN
      END
      SUBROUTINE RESTOR(N)
C                                                              4/1/63
      COMMON AVSL,W(100)
      DO 1 I=1,N
    1 CALL POPTOP(W(I))
      RETURN
      END
      FUNCTION LOCT(K)
C                                                              4/1/63
        IF(NAMTST(K))1,2,1
    2   LOCT = K
        RETURN
    1   PRINT 901
        STOP
  901 FORMAT (1H1,94HA LIST WAS REQUIRED AS AN OPERAND BUT WAS NOT FOUND
     1- THE PROGRAM WAS REGRETFULLY TERMINATED .  )
      END
      FUNCTION PARMT2(A,B)
C                                                              4/1/63
      COMMON X,W(100)
      CALL NEWTOP(A,W(1))
      CALL NEWTOP(B,W(2))
      PARMT2 = A
      RETURN
      END
      FUNCTION NOATVL(AT,LST)
         M = MADATR(AT,LST)
         IF(M+1)2,1,2
    2 NOATVL = INTGER(DELETE(LNKR(CONT(M))))
      CALL DELETE(M)
      RETURN
    1 NOATVL = 0
      RETURN
      END
      FUNCTION NEWVAL(AT,VAL,LST)
         M = MADATR(AT,LST)
         IF(M+1)2,1,2
    2 NEWVAL = INTGER(SUBST(VAL,LNKR(CONT(M))))
      RETURN
    1 CALL LDATVL(AT,VAL,LST)
      NEWVAL = 0
      RETURN
      END
      FUNCTION ITSVAL(AT,LST)
```



```
         IF(LNKL(CONT(LST+1)))3,4,3
    3    M = MADATR(AT,LST)
         IF (M+1)1,2,1
    1 ITSVAL = INHALT(LNKR(CONT(M))+1)
      RETURN
    4 CALL DERROR(LST)
    2 ITSVAL = 0
      RETURN
      END
      FUNCTION MTDLST(LST)
         MTDLST = LST
      K = LNKL(CONT(LOCT(LST)+1))
         IF(K)1,2,1
    1 CALL SETDIR(0,K,K,X)
      CALL MTLIST(X)
    2 RETURN
      END
      FUNCTION MAKEDL(L,M)
      CALL MTDLST(M)
      MAKEDL = M
      N = LOCT(M)
      K = LOCT(L)
      CALL SETIND(-1,K,-1,N+1)
      CALL SETIND(-1,-1,LCNTR(L)+1,K+1)
      RETURN
      END
      FUNCTION NAMEDL(L)
      NAMEDL = LNKL(LOCT(L)+1)
      RETURN
      END
      FUNCTION LDATVL(AT,VL,LST)
         IF(LNKL(CONT(LST+1)))1,2,1
    2      LDATVL = LISTAV(LST)
    1 CALL NXTRGT(VL,NXTLFT(AT,LNKL(CONT(LST+1))))
      RETURN
      END
      FUNCTION LISTAV(LST)
      LISTAV = LIST(0)
      CALL SETIND(-1,LNKR(LISTAV),-1,LST+1)
      RETURN
      END
      FUNCTION MADATR(AT,LST)
         LSTDES = LNKL(CONT(LST+1))
         IF(LSTDES)1,4,1
    1    MADATR =  LNKR(CONT(LSTDES))
    8    IF (ID(CONT(MADATR))-2)3,4,3
    3    IF (EQUAL(CONT(MADATR+1) ,AT))5,6,5
    5    M = LNKR(CONT(MADATR))
         IF (ID(CONT(M))-2)7,4,7
    7    MADATR = LNKR(CONT(M))
         GOTO 8
    4 MADATR = -1
    6 RETURN
      END
      SUBROUTINE DERROR(LST)
      PRINT 900,LST
      PRINT 901
      RETURN
  900 FORMAT (1H1,20X,05)
  901 FORMAT (20X,44HATTRIBUTE-VALUE LIST REQUIRED BUT NOT FOUND )
```

```
      END
      FUNCTION MRKLST(M,LST)
      MRKLST = LST
      CALL SETIND(M,-1,-1,LOCT(LST)+1)
      RETURN
      END
      FUNCTION MRKLSS(M,LST)
      MRKLSS = LST
      LR = LRDROV(MRKLST(M,LST))
    3    X = ADVSNR(LR,K)
         IF(K)1,2,1
    2 CALL SETIND(M,-1,-1,LNKR(X)+1)
      GOTO 3
    1 CALL RCELL(LR)
      RETURN
      END
      FUNCTION RDLSTA(Z)                                          629
C                                                      4/1/63     630
      DIMENSION CRDBUF(10)                                        631
      COMMON /XTRCT/ CP(8),BLANK,NULL,ZERO,LP,RP                  632
   80 FORMAT (10A8)                                               633
      ASSIGN 13 TO START                                          634
      ASSIGN 20 TO NEWLST                                         635
      CALL LIST(STACK)                                            636
      IS = 1                                                      637
      WORD  = BLANK                                               638
      KOUNT = 0                                                   639
      PLACE = BLANK                                               640
   12 READ 80,CRDBUF                                              641
      PRINT 80,(CRDBUF(J),J=1,9)                                  642
      IW = 1                                                      643
   10 IC = 1                                                      644
    9 SYMBOL = SQIN(CP(8),SQOUT(CP(IC),CRDBUF(IW)),PLACE)         645
      IF (EQUAL(SYMBOL,BLANK))1,2,1                               646
    1 IF (EQUAL(SYMBOL,LP))3,4,3                                  647
    3 IF (EQUAL(SYMBOL,RP))5,6,5                                  648
   13    IF (IC - 8) 7,8,7                                        649
    7    IC = IC + 1                                              650
         GOTO 9                                                   651
    8    IF (IW - 9) 11,12,11                                     652
   11    IW = IW + 1                                              653
         GOTO 10                                                  654
    4 IF (KOUNT)40,44,40                                          655
   44 CALL NXTRGT(LIST(NEW),STACK)                                656
      KOUNT = 1                                                   657
      CALL VISIT(START)                                           658
      RDLSTA = POPTOP(STACK)                                      659
      CALL MTLIST(STACK)                                          660
      CALL RCELL(STACK)                                           661
      RETURN                                                      662
   40 IF (EQUAL(WORD,BLANK))41,42,41                              663
   41    CALL NXTLFT(LANORM(WORD),TOP(STACK))                     664
         WORD = BLANK                                             665
      IS = 1                                                      666
   42 CALL VISIT(NEWLST)                                          667
      CALL POPTOP(STACK)                                          668
      GOTO 13                                                     669
C     NEWLST                                                      670
   20    CALL NXTLFT(LIST(NEW),TOP(STACK))                        671
      CALL NXTRGT(NEW,STACK)                                      672
```



```
      GOTO 13                                                     673
C     NON-BLANK SYMBOL NOT LP OR RP                               674
    5 CALL  SHIN(6,SYMBOL,WORD)                                   675
         IF (IS - 8)51,52,51                                      676
   51    IS = IS + 1                                              677
         GOTO 13                                                  678
   52    IS = 1                                                   679
   21    CALL NXTLFT(LANORM(WORD),TOP(STACK))                     680
         WORD = BLANK                                             681
      IS = 1                                                      682
         GOTO 13                                                  683
    2 IF (EQUAL(WORD,BLANK))21,13,21                              684
C     RIGHT PARENTHESIS                                           685
    6 IF (EQUAL(WORD,BLANK))61,62,61                              686
   61    CALL NXTLFT(LANORM(WORD),TOP(STACK))                     687
         WORD = BLANK                                             688
      IS = 1                                                      689
   62 CALL TERM(Z)                                                690
      END                                                         691
      SUBROUTINE PRLSTS(OUTLST,I)                                 692
C                                                      4/1/63     693
      EQUIVALENCE (KOUT,OUT)                                      694
  900 FORMAT (1H1,20X,10HBEGIN LIST)                              695
  901    FORMAT (21X,8HEND LIST)                                  696
  902 FORMAT (21X,I14)                                            697
  903 FORMAT (21X,13HBEGIN SUBLIST)                               698
  904    FORMAT(21X,13HEND SUBLIST    )                           699
  905 FORMAT (21X,A8)                                             700
  906 FORMAT (21X,F10.4)                                          701
  907    FORMAT (21X,13HEMPTY SUBLIST)                            702
         PRINT 900                                                703
      LR = LRDROV(OUTLST)                                         704
      LEVEL = 0                                                   705
    7 X = ADVSWR(LR,K)                                            706
         IF(K)1,2,1                                               707
    2 IF(LEVEL - LCNTR(LR))21,22,23                               708
   22 IF(NAMTST(X))3,4,3                                          709
    4 IF(LISTMT(X))5,6,5                                          710
    6 PRINT 907                                                   711
      GOTO 7                                                      712
    5 PRINT 903                                                   713
      LEVEL = LEVEL+1                                             714
      GOTO 7                                                      715
    3 GOTO(11,12,13)I                                             716
   11 OUT = X                                                     717
      PRINT 902,KOUT                                              718
      GOTO 7                                                      719
   12 OUT = X                                                     720
      PRINT 905,KOUT                                              721
      GOTO 7                                                      722
   13 PRINT 906,X                                                 723
      GOTO 7                                                      724
   23 PRINT 904                                                   725
      LEVEL = LEVEL - 1                                           726
      GOTO 2                                                      727
    1 IF (LEVEL - LCNTR(LR))21,32,33                              728
   33 PRINT 904                                                   729
      LEVEL = LEVEL - 1                                           730
      GOTO 1                                                      731
   32 PRINT 901                                                   732
```

Continued from page 505

a FORTRAN IV level specification and a subset; X3.4.2-ALGOL has contributed specification proposals and USA positions to the international standardization of ALGOL, which also has reached the preliminary proposal document state; X3.4.4-COBOL has defined its program of work for realization of a standard, and has progressed actively in concert wtih CODASYL and the COBOL Maintenance Committee. Although resigning from this domestic responsibility, Dr. Clippinger plans to continue his chairmanship of the international ISO/TC97/SC5 —Programming Languages.

An Open Letter to X3.42

To: Subcommittee X3.4.2
American Standards Association

Dear Sirs:

Your "Suggestions on ALGOL 60 (ROME) Issues", appearing in the January *Communications*, p. 20, seemed mostly well thought out and in the spirit of ALGOL. However, at two points I found your language vague.

First, at the end of section 4 you suggest "that the interpretation of the step-until-element might be made more efficient by replacing the ALGOL 60 statements in 4.6.4.2 with" a different sequence of ALGOL statements.

What does "might be made more efficient" mean? For some programs and/or for some ALGOL translators? Would it not be better, as you did in section 2, to omit questions of efficiency of implementation? In any case, the sequence of ALGOL statements which you propose will not always have the same effect as the corresponding sequence in the ALGOL report. This is because after initialization your loop calls "B" once each time through but the sequence of statements in 4.6.4.2 calls "B" twice each time through. Thus if B is a function designator, your interpretation may result in side effects different than those resulting from the ALGOL interpretation.

Second, in section C (Numeric Labels) you state: "Since the only additional facility introduced by numeric labels create an ambiguous case . . .". Consider the following:

```
procedure P (Q);
  begin
    ...;
    if Q <5 then go to Q;
    ...
  end
```

I do not believe this case is ambiguous.

May I conclude with two points which you did not discuss?

First, with respect to section 4.3.5 of the ALGOL report. Whether "a designational expression is a switch designator whose value is undefined" is not a decidable question (it might involve a function designator which computes forever and never delivers a value). Hence it would seem that 4.3.5, as it stands, is not implementable.

Second, with respect to section 4.7.1 of the ALGOL report. A procedure statement qualifies as an actual parameter provided that either (1) the procedure is parameterless or (2) the procedure is a function designator. Is there good reason for not allowing as an actual parameter a procedure statement not qualifying under (1) or (2)?

HERBERT J. HAUER
Mathematics Department,
University of Calif.
Berkeley 4, Calif.,

733

```
      CALL RCELL(LR)
   21 RETURN
      END
      FUNCTION SEQLL(Z,N)
          L = LNKL(Z)
          Z = CONT(L)
          SEQLL = CONT(L+1)
          N = ID(Z)-1
      RETURN
      END
      FUNCTION SEQLR(Z,N)
          L = LNKR(Z)
          Z = CONT(L)
          SEQLR = CONT(L+1)
          N = ID(Z)-1
      RETURN
      END
      FUNCTION SEQSR(Z,N)
          IF (ID(Z)-1)4,5,4
    5 L = LNKR(CONT(LNKL(CONT(LNKR(Z)))+1))
          GOTO 3
    4     L = LNKR(Z)
    3     IF(ID(CONT(L))-1)1,2,1
    1     SEQSR = CONT(L+1)
          Z = CONT(L)
          N = ID(Z)-1
      RETURN
    2     L = LNKR(CONT(CONT(L+1)))
          GOTO 3
      END
      FUNCTION SEQSL(Z,N)
          IF (ID(Z)-1)4,5,4
    5     L = LNKL(CONT(CONT(LNKL(CONT(LNKR(Z)))+1)))
          GOTO 3
    4     L = LNKL(Z)
    3     IF(ID(CONT(L))-1)1,2,1
    1     SEQSL = CONT(L+1)
          Z = CONT(L)
          N = ID(Z)-1
      RETURN
    2     L = LNKL(CONT(CONT(L+1)))
          GOTO 3
      END
      FUNCTION SEQRDR(LST)
      SEQRDR = CONT(LOCT(LST))
      RETURN
      END
```

A15