A METHOD FOR PROCESS COMPUTER ALARM ANALYSIS

by

PETER KENNETH ANDOW

APPENDICES

APPENDIX 1

LIST PROCESSING AND ALGOL 68-R

TABLE OF CONTENTS

A.1. INTRODUCTION

The term "list processing" has come into use in the last few years as a name for techniques used for manipulating lists of items stored in computers. The reasons for the use of lists will be explained in the following sections. At this point, it will merely be stated that list processors manipulate objects in much the same manner as other languages manipulate numbers.

Languages such as LISP(1) and IPL(2) were designed primarily for list processing. ALGOL 68-R(3) was not produced for this purpose. ALGOL 68 is an advanced programming language with many useful features due largely to its generality and flexibility. ALGOL 68-R is an implementation of ALGOL 68(4) produced by an experienced team of programmers at the Royal Radar Establishment at Malvern. The use of the language for list processing is a product of its flexibility.

The literature contains a number of references to list processing and it is not the function of this Appendix to duplicate that information or to teach the beginner how to use ALGOL 68-R. The information given here presents some of the problems encountered by workers in the field of Artificial Intelligence that may be overcome by the use of list processing techniques. In order to prevent the description of these techniques becoming too abstract, the methods are illustrated by examples using specific features of ALGOL 68-R. It is hoped that this approach will render the text more readable.

Some of the basic rules of ALGOL 68-R may conveniently be mentioned here. The ALGOL family of languages generally assume both upper and lower-case characters. Language words are upper-case and variables lower-case. Since they are frequently not available (and in particular on thc 1904 at Loughborough) language words are enclosed between primcs (e.g. 'MØDE" ABC'). Readers familiar with FORTRAN might also find it profitable to note that:

1) FORMATS do not <u>have</u> to be used for input/output

2) Routines may be recursive, i.e. call themselves

3) All identifiers must be declared

4) The "scope" of variables is rather different

5) Even without the facilities of ALGOL 68-R, ALGOL languages are generally very flexible.

Reference is also made in this Appendix to Game-Playing on computers. The main interest in game-playing is due to the relative ease of representation and yet the enormous complexity, from a decision-making viewpoint, of this type of problem. The general name for the field of science which encompasses game-playing problems is "Artificial Intelligence". A general discussion of list-processing for simple problems is given by Foster[(5)].

A.1.1 ALGOL 68-R

A full account of the facilities and syntax of ALGOL 68-R may be found elsewhere(3). In this section, an attempt will be made to define and discuss some of the features of the language which are of particular interest for the present work. An important feature not discussed here is the complete structure of ALGOL 68-R programs. This information may be found in the standard texts.

A.1.1.1 Values, names and identifiers

The definitions given in this subsection are simplified in order that some of the basic features of list-processing may be discussed without giving a complete, formal definition of each language feature encountered.

Values

A value may be defined as the information which the programmer wishes to store. Note that this definition does not restrict values merely to be numbers, characters or booleans. A value has a mode (called a TYPE in some languages) as defined when it is created in the program.

Names

A name may be defined as a machine pointer to a working space where some information (the value) is stored. Since the name refers to a value, it has a mode of REF (mode of value).

Identifiers

Identifiers are the tags in the program used by the programmer to identify particular variables or constants.

The syntax of ALGOL 68-R is complicated by the fact that the strict notation has an alternative form ("extended" notation) which is much more generally used. In order to understand the language fully, it is undoubtedly better to learn the strict form. The programs used for this study use the extended form wherever possible (since this form is shorter to write) and hence this Appendix also uses the extended form of the notation in order to be consistent. For the sake of completeness, a simple declaration is given below in both forms:

Strict Form	Extended Form
'REF''INT' I = 'LOC''INT'← 1;	'INT' I← 1;

These declarations are exactly equivalent, in each case an integer variable is declared, accessed by the programmer using the identifier "I". In core store, a name is created equivalent to "I". This name possesses a working space where the value of "I" is stored. The initial value is set to unity.

Now consider further statements in extended form:

```
1.        'INT' I← 1, J ← 4;
2.        I ← 3;
3.        I ← J;
```

Note that the numbers 1,2 and 3 on the left side of the page are not supplied by the programmer and are used here for reference purposes.

Line 1 is as before. Line 2 assigns an 'INT' to I, whose name is of mode 'REF''INT' and whose value must be an 'INT'. Line 3 assigns one 'INT' variable to another. Now, in machine terms, there is a name on the L.H.S. and a name on the R.H.S., both of mode 'REF''INT'. Since the only value that a 'REF''INT' name can have is of mode 'INT' the R.H.S. must be dereferenced to yield its 'INT' value (4 in this case) and the assignment is made. Note that the dereferencing itself does not change the contents of the area of store considered.

Now consider the declaration below:

'REF"INT' R;

This is a declaration, using the extended form, giving rise to the creation of an unspecified 'REF"INT' value on the stack, a 'REF"REF"INT' name and the identifier R possessing the name.

'REF"INT' S ⟵ J;

This declaration achieves the same result as above but also gives "S" the initial value of "J" (declared earlier with an 'INT' value and a 'REF"INT' name). Since the name of "S" is 'REF"REF"INT' and the value is 'REF"INT' the name of "J" is assigned to "S" - no dereferencing occurs.

Note that:

S ⟵ 2;

is illegal and will give a compilation error.

"S" requires a 'REF"INT' value not an 'INT' value.

The examples below are intended to demonstrate the effects of different REFerence levels in a fairly common situation. Note that the user may insert comments between pairs of 'C's as shown:

PROG 1	Line	PROG 2
'INT' A←1, B←0;	1	'INT' A ← 1, B ← 0;
'INT' R←A;	2	'REF"INT' R ← A;
'C' "A" is dereferenced to yield an INT value 'C'		'C 'No dereferencing occurs 'C'
A ← 2;	3	A ← 2;
B ← R;	4	B ← R;
'C' "R" is dereferenced to yield an 'INT' value 'C'		'C' "R" is derefernced twice to yield an 'INT' value 'C'
PRINT (B);	5	PRINT (B);
'C' prints 1 'C'		'C' prints 2 because "R" REFers to "A" whose value has been altered at line 3 'C'

A.1.1.2. Modes and Chained Data Structures

The previous section showed how one variable can be used as a reference, or pointer, to another. This section extends that idea to show how a chain of items may be constructed.

A powerful feature of ALGOL 68-R is the provision for the programmer to declare new modes. As an example of this, consider a situation where it is desired to store a character and an integer together. This may be achieved using the mode declaration, which effectively defines a new language word (in this case 'CHARINT'):

```
'MØDE"CHARINT' = 'STRUCT'('CHAR' CHARACTER, 'INT' INTEGER);
```

This defines a new mode (called CHARINT) as a structure possessing two fields, one containing a CHAR value and one an INT value. The selectors "CHARACTER" and "INTEGER" may be chosen by the programmer and are used to access the fields of CHARINT variables. "CHARACTER" and "INTEGER" are not names although the CHARINT itself has a name in the same way as any other variable. Quotes marks are used to denote an actual character to be stored as shown:

```
'CHARINT' X ← ("A", 47);
```

This line declares a variable of mode 'CHARINT', gives it a 'REF"CHARINT' name and makes the identifier "X" possess this name. The value is initialised by the collateral to consist of the character "A" and the integer 47. The 'CHARINT' value of X is therefore the pair of quantities "A" and 47.

The short section of program below shows how the individual fields may be accessed using the selectors:

```
'INT' I;
'CHARINT' CI ← ("A", 47);
CHARACTER'ØF' CI ← "B";
INTEGER'ØF' CI ← 48;
I ← INTEGER'ØF' CI;
```

The fields of a mode may contain any other previously declared* mode, arrays or references:

```
'MODE"ANØTHER' = 'STRUCT'('REAL' RE,
                          'REF' 'BØØL' TF,
                          'REF"INT' II);
```

* note that this is not quite true; modes may be "mentioned" as will be shown in Section A.1.2.1.

Note the similarities with the previous mode declaration:

1. The first word 'MODE'
2. The name of the new mode
3. The = sign
4. The word 'STRUCT' to denote that the new mode is a structure containing several parts
5. The definition of the contents of each field and the relevant selector.

Of particular interest here are modes that form chains:

```
'MØDE''CHAIN' = 'STRUCT'('REAL' THIS,
                         'REF''CHAIN' NEXT);
```

The value of a variable of mode 'CHAIN' consists of two fields, a 'REAL' number and the name of another 'CHAIN' variable. This is shown by the example:

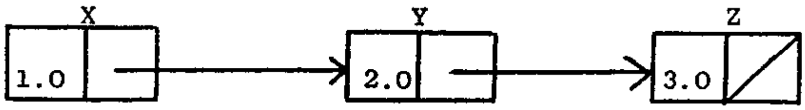
```
'CHAIN' X, Y, Z;

X ← (1.0, Y);

Y ← (2.0, Z);

Z ← (3.0, 'NIL');
```

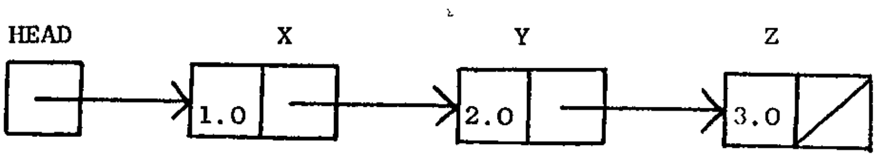
The second field of Z contains a REFerence to "no place". NIL is a plug and effectively terminates the chain as shown by the diagrammatic representation below:



Pointers may be used to "remember" a place in the list, for instance, the top:

```
'REF"CHAIN' HEAD ⟵ X;
```

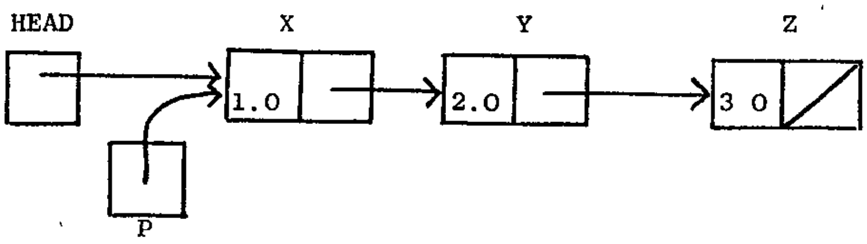
This may be represented as:



In general at least 2 pointers are required to process lists, the first is used to point to the top of the list, as above, and the second is used to manipulate the list:

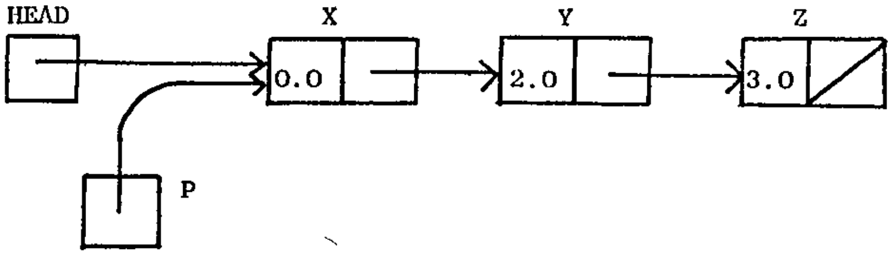
```
'REF"CHAIN' P ⟵ HEAD;
```

Note that "P" cannot point to "HEAD" (this would only be true if P had been declared as: 'REF"REF"CHAIN' P;) but to the same CHAIN that "HEAD" points to



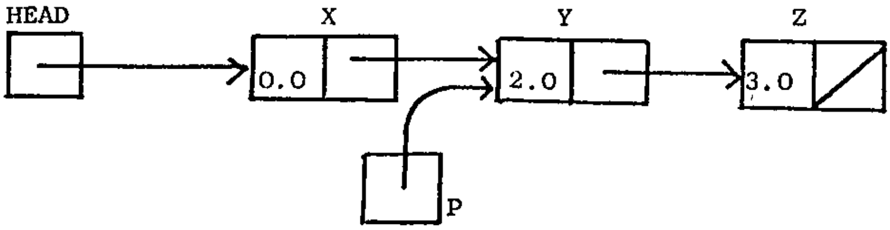
In this position "P" may be used to alter the fields of "X" as below:

```
THIS'ØF'P ⟵   0.0;
```



"P" may be made to move down the chain:

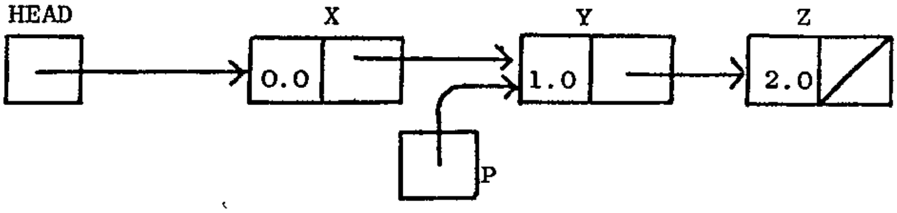
```
P ←— NEXT'ØF'P;
```



Now consider:

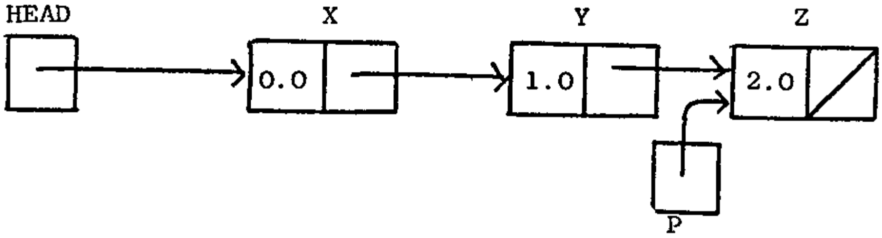
```
THIS'ØF'P ←— 1.0;

THIS'ØF'NEXT'ØF'P ←— 2.0;
```



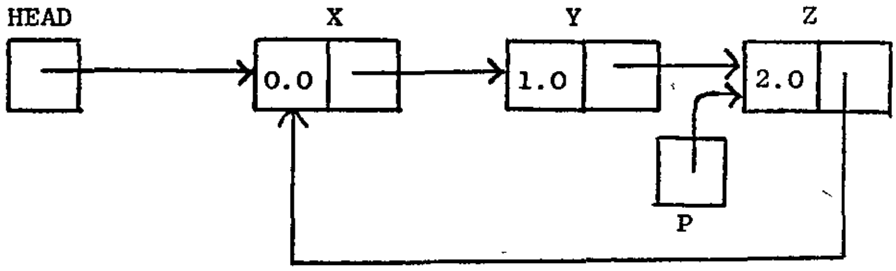
As before:

```
P ←—  NEXT'ØF'P;
```



The pointers may also be used to alter the chain:

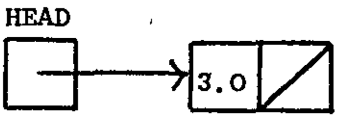
```
NEXT'ØF'P ←—— HEAD;
```



Note that since the introduction of "P", the identifiers X, Y and Z have not been used at all. This suggests that, provided enough pointers are used, the identifiers X, Y and Z are completely superfluous. The use of unidentified references is in fact a positive advantage that will be put to good use. ALGOL 68-R provides generators for the programmer to create such unidentified references. The previous example may be repeated using generators to demonstrate this:

```
'REF"CHAIN' HEAD ←— 'CHAIN' ←—— (3.0, 'NIL');
```

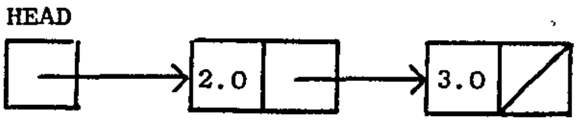
This creates a variable with a value of mode 'CHAIN'* and initialises it. "HEAD" points to this unidentified reference: the value of "HEAD" is the name of the reference.



```
HEAD ←—— 'CHAIN' ←—— (2.0, HEAD);
```

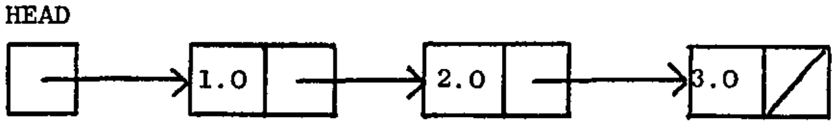
* on the heap - this term is more conveniently explained later in Section A.1.1.3.

Now another 'CHAIN' is created and given the value 2.0 for its THIS field. The name stored in "HEAD" is found by dereferencing and assigned to the NEXT field of the new variable. "HEAD" is then made to point to the new 'CHAIN':



Finally, another element of the list is added:

```
HEAD ←— 'CHAIN' ←— (1.0, HEAD);
```



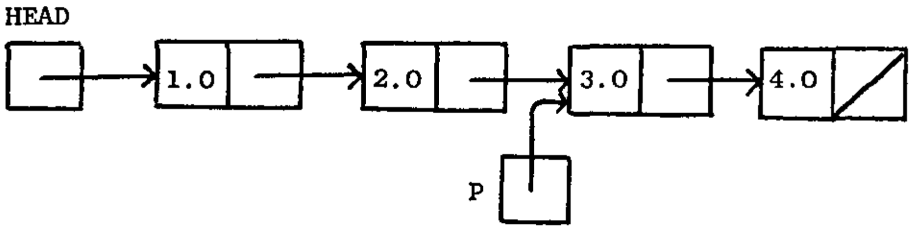
Note that, if a generator exists in a loop, then the number of calls to it is fixed at run time. This is convenient for reading in or processing sets of items where the number in the set varies greatly. The above structure could be created in one step:

```
'REF''CHAIN' HEAD ←— 'CHAIN' ←— (1.0,'CHAIN' ←—
                                 (2.0,'CHAIN' ←—
                                 (3.0,'NIL')));
```

So far, all new elements have been added to the top of the list. Elements may also be added to the tail of the list by searching for the 'NIL':

```
'REF''CHAIN' P ←— HEAD;
'WHILE' NEXT'ØF'P 'ISNT'('REF''CHAIN''VAL''NIL')'DØ'
P ←— NEXT'ØF'P;
NEXT'ØF'P ←— 'CHAIN' ←— (4.0, 'NIL');
```

The DO loop and condition for repetition shown in line 2 causes lines 2 and 3 to be repeated until NEXT'ØF'P contains a 'NIL'. Line 4 then adds the new element to give:



Note that this piece of program would have failed during execution if the list was empty (i.e. if HEAD had a 'NIL' value). In order to make the piece of program general, a dummy element may be added to the top of the list or a special test written in to detect the empty list. Both of these methods are shown below:

<u>Adding dummy element</u>:

```
1.          'REF''CHAIN'P ←— 'CHAIN' ←— (0.0,HEAD);
2.          HEAD ←— P;
3.          'WHILE'NEXT'ØF'P'ISNT'('REF''CHAIN''VAL''NIL')'DØ'
4.          P ←— NEXT'ØF'P;
5.          NEXT'ØF'P ←— 'CHAIN' ←— (4.0,'NIL');
6.          HEAD ←— NEXT'ØF'HEAD;
```

Note that the dummy element has no pointers to it and cannot be accessed by the programmer. The significance of this will be seen later.

Adding special test:

```
'REF''CHAIN'P ←— HEAD;
'IF'P'IS' ('REF''CHAIN''VAL''NIL')
'THEN' HEAD ←— 'CHAIN' ←— (4.0, 'NIL')
'ELSE' 'WHILE' NEXT'ØF'P'ISNT'('REF''CHAIN''VAL''NIL')'DØ'
            P← NEXT'ØF'P;
            NEXT'ØF' P←'CHAIN'← (4.0,'NIL')
'FI';
```

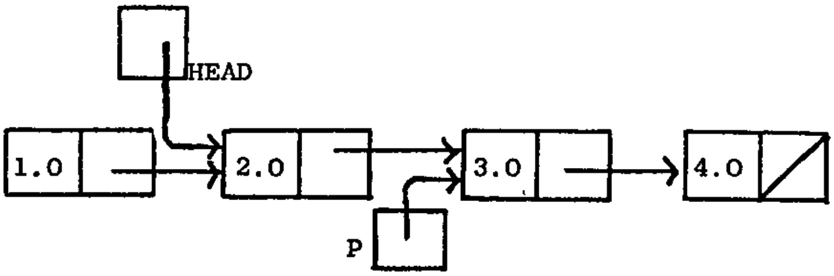
This example also illustrates the use of simple conditional statements in ALGOL 68-R. The conditional facilities of the language are extremely powerful. Many examples of their use in complex logical systems may be seen in the program listings in Appendix 3.

The example shown illustrates the general case which may be expressed as below:

```
'IF'   〈 the expression here is TRUE 〉
'THEN' 〈 do whatever is written here 〉
'ELSE' 〈 do whatever is written here 〉
'FI'; 'C'    This is the end of the conditional 'C'
```

Now consider the effect of:

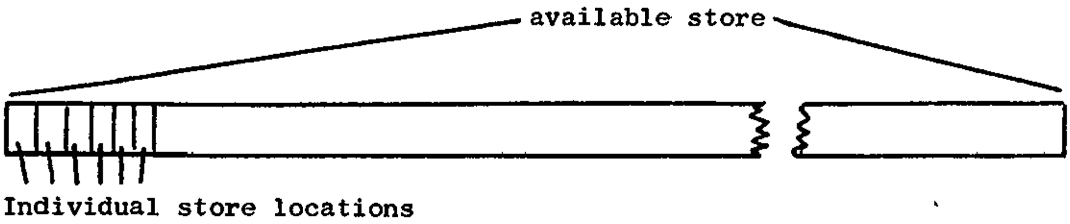
HEAD ←— NEXT'ØF'HEAD;



At this point, the first element of the chain is completely inaccessible to the programmer: he has no identifier possessing its name and no way of accessing it via the two pointers. In a typical list processing problem, this may occur frequently while generators are continually requesting more space. This rapidly leads to a situation where no more core-store is available. Having recognised this problem, it is now convenient to consider how the use of generators affects the scheme for the allocation of core-store within a computer.

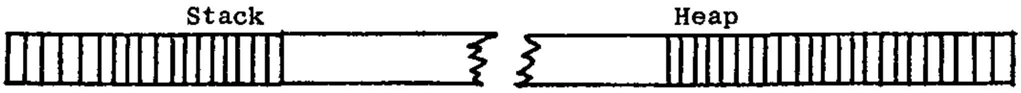
A.1.1.3. Stack and Heap Storage

In this section, a conceptual model of the core store will be used to illustrate how the use of generators (and certain other ALGOL 68-R features) necessitates the use of a dual scheme for storage allocation. Normally the compiler allocates storage for program variables before a program is run. This storage may be thought of as existing at one end of the available store:



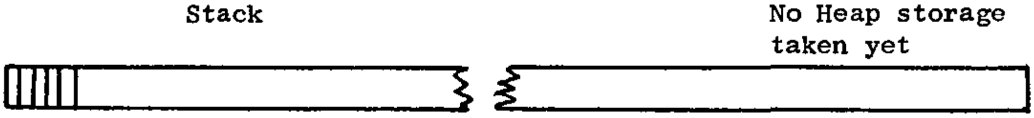
In the representation above, this type of store is depicted as existing at the left-hand end of the available store. This is known as the "stack". Stack storage is organised so that programs run very efficiently.

Now consider the use of generators. It has been shown in the previous section that generators are used when needed and therefore the amount of space for use by the generators is not determined at compile time. In order to overcome this difficulty, it is convenient to imagine that the space needed for the generators is at the right-hand end of the model of the store. This is called the heap:

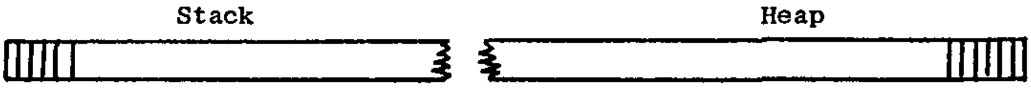


As heap storage is required, it will be allocated from positions progressively closer to the stack:

Start of run:



After generators used:



Eventually, the two will meet. If all of the heap storage is still accessible to the program, then the computation cannot proceed. However, if some of the storage is no longer accessible, then it would be useful if this space could be reassigned.

In order to organise a system of this type, all the free storage is placed in a list known as the "free list". When the free list is empty, an internal routine inspects the pointers and identifiers in the program and marks all heap space that can still be accessed. The remainder is added to the free list and computation proceeds as before. This process is known as "garbage collection". The use of this facility in a program will cause considerable overheads in time and space even if the particular run of the program does not use the heap. The use of ALGOL 68-R features which invoke the heap should therefore be restricted to problems which are not easily handled by the use of, for instance, arrays.

A.1.1.4 Use of Segments

The ALGOL 68-R system allows the user to assemble complete programs from a series of previously-compiled segments. The details of programs used to store, amend, forget and print out the specification of segments may be found in the R.R.E. documentation. The system used at Loughborough is basically similar although modified to fit in with the version of the GEORGE 2 operating system currently used. Details of all ALGOL 68-R facilities at Loughborough are contained in the Computer Centre Documentation (6).

Segments are primarily intended for storage of frequently used "library" routines (the system library is stored in a number of segments) but, for the present work, the use of segments was mandatory because of the large program size. There are three relevant factors:

1. There is a finite limit to the size of a programm that can be compiled at one time. A large program may be compiled as a series of segments, thus avoiding this difficulty,
2. The use of excessive numbers of cards for each program run is avoided,
3. For a large program, the compilation time is excessive if the program is compiled prior to each run. In the case of the present study, compilation time was approximately 120 mills (about 1 minute) before the use of segments became possible on the system implemented at Loughborough.

All segments are stored in an "album" on a magnetic disk. The ALGOL 68-R system allows a hierarchy of albums to be built up so allowing individual users to share programs available at higher levels in the hierarchy (the system library is at the highest level in the hierarchy and is therefore available to every user). For the present study, a private album called PKALBUM was used to store the segments containing the alarm analysis programs.

Variables, operators, modes and procedures declared in a segment are available in later segments if their names are included in the 'KEEP' list of the segment. As an example, consider the complete job shown in Figure A.1.1. The example is not intended to be self-explanatory but is included in full to illustrate the differences between the Loughborough system and that defined in the R.R.E. documentation.

```
JOB  G674,G,PKA1157
JOBCORE   37000
CARTRIDGES  ≠≠100061
SELECT LANGUAGEMACS
AL68SETUP
AL68COMPILE SEG1
AL68UPAL PKALBUM,UP1
AL68COMPILE SEG2,PKALBUM,SEG1
AL68RUN
****
DOC SEG1
'BEGIN'
       'MODE"ONE' = 'STRUCT'('REAL'A,'INT'B);
       [ ]'INT'I = (1,2,3,4,5,6);
       'ONE'WON ← (23.0,4);
       'PROC'PRINTSPACES = ('INT'I):
       'BEGIN'  [1:I]  'CHAR' SPACES;
                'CLEAR'SPACES;
                PRINT (SPACES)
       'END';
       'SKIP'
'END'
'KEEP' PRINTSPACES,'ØNE',I,WON
'FINISH'
****
DOC UP1
PUTIN SEG1
END
****
DOC SEG2
'BEGIN'
       'INT'L;
       READ((NEWLINE,L));
       PRINTSPACES(L)
'END'
'FINISH'
****
DOC DATA
10
****
```

FIG. A.1.1

Example of Use of Segments

The functions of the macros shown in the JOB description may be found in the Computer Centre documentation(6). For the purpose of this example note that:

1. At line 6, the command to compile SEG1 is given.

2. At line 7, the user specifies that the album PKALBUM is to be updated according to the instructions in DOCument UP1. (All input data and programs must be in DOCuments for GEORGE).

3. At line 8, the command to compile SEG2 using items (as yet unspecified) from SEG1 of PKALBUM.

4. At line 11, SEG1 starts. Note that the system employed uses the DOCument name as the segment name.

5. At line 16, the procedure PRINTSPACES is declared.

6. At line 23, PRINTSPACES is included in the KEEP list of the segment SEG1.

7. At line 27, SEG1 is put in to the album PKALBUM.

8. At line 34, PRINTSPACES is used in SEG2.

A.1.2. List Processing

It was noted in the first section of this Appendix that problems encountered in the field of Artificial Intelligence led to the development of list processing languages. It is the purpose of this section to examine more closely the type of problems involved in terms of the ALGOL 68-R features described in the preceeding section.

A.1.2.1. Game Playing Problems

In order to be more specific than would otherwise be possible, a particular class of games will be considered. Games within this class are certainly not the only ones that are conveniently solved using list processing techniques.
The particular class of games are defined by:

1) Two players (only) moving pieces that are capable of capture,
2) No outside influence or random effects,
3) The domain is limited,
4) There is a finite but large number of possible positions,
5) Moves are made alternately,
6) There is a well-defined goal and possibly sub-goals,
7) A fixed set of constraints.

Chess and draughts are probably the most well-known examples of games within this class. Whilst a knowledge of these games in particular is not necessary for this discussion, it may be useful to visualise the problems in terms of these games.

It is also assumed that criteria of merit are defined in order to enable a position to be evaluated. These criteria must take into account which of the two players will make the next move. It will also be assumed that the game is between a computer (the machine) and a human (the player). The particular aspect of game-playing problems that concerns this discussion is the organisation and programming of the machine.

Consider now the three basic steps involved when the machine has to make a move:

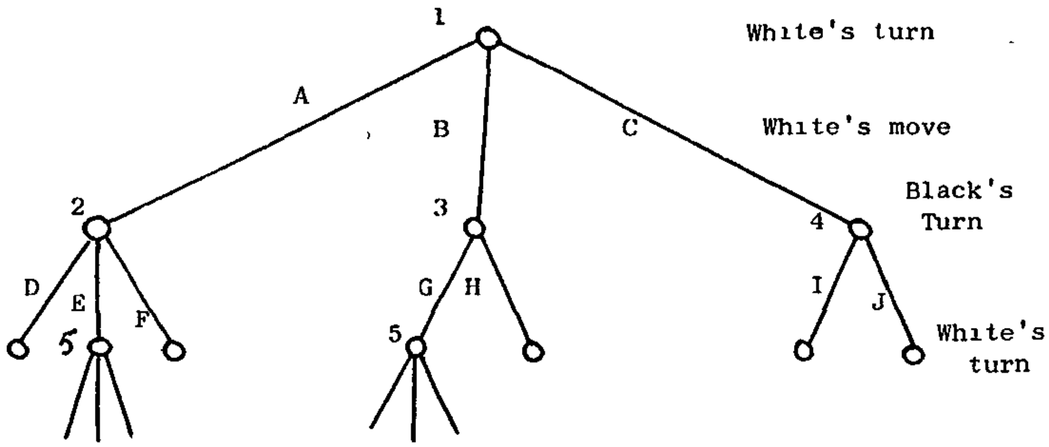
1) Various legal moves are discovered (and continuations of them)
2) The merits of the possible moves are determined
3) The final decision on choice of move is made and implemented.

The various problems involved in each of these steps will now be considered separately and the use of the ALGOL 68-R features described in the previous section explained.

1. Discovery of legal moves

Various methods have been employed to discover possible moves. The net result of all the methods is that moves are found one by one. In general, it is not known at the start of this step how many moves will be found and it is therefore convenient to allocate storage space for the information associated with each move using generators.

A tree-like representation may be used to discuss this problem. The nodes of the tree are equivalent to positions and the branches from a node are possible moves:



From the above diagram, it can therefore be seen that, at position 1, there are three moves A, B and C leading to positions 2, 3 and 4 respectively. For the purposes of simplicity only a small number of possible moves are shown at each position. In many board games, there are considerably more - in chess, 30 on average. Note also that move E from position 2 and move G from position 3 both lead to position 5. It is obvious that it is undesirable to duplicate all the information associated with a position since much space is wasted. A computer representation which avoids this problem as well as retaining the above structure is therefore desired.

If each position is considered as a place where:

1) information is stored concerning locations of pieces, evaluations etc.,

2) a list of possible moves is stored,

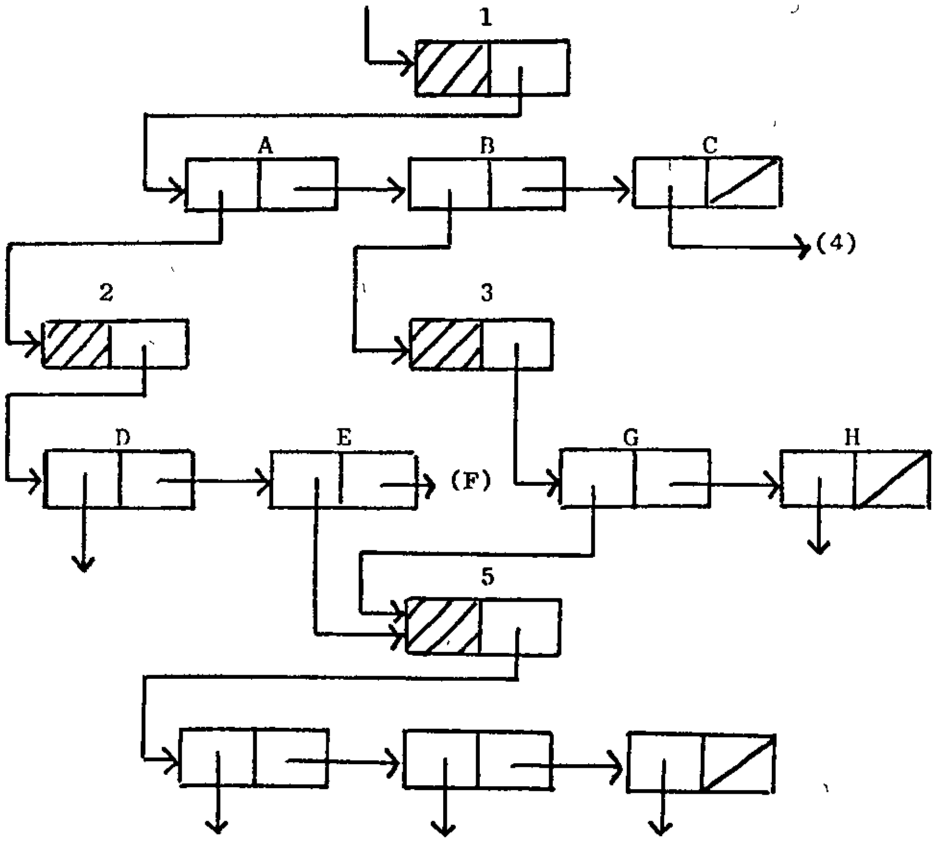
then the following mode declarations are easily understood:

```
'MØDE''MØVE', 'C' this mode is mentioned 'C'
'MØDE''DATA'' = 'STRUCT'( 〈 whatever is required〉 );
'MØDE''PØSITIØN' = 'STRUCT'('DATA' INFØRMATIØN,
                                    'REF''MØVE' LIST);
'MØDE''MØVE' = 'STRUCT'('REF''PØSITIØN'PLACE,
                                 'REF''MØVE' NEXT);
```

Mode "MØVE" is mentioned before it is defined. This is necessary since it REFers to mode "PØSITIØN" and vice-versa.

Note that the use of REFerences allows position 5 to be stored only once. When a new "PØSITIØN" is generated, its "LIST" field will contain "NIL". As moves from that position are discovered, they are added one by one to the "LIST".

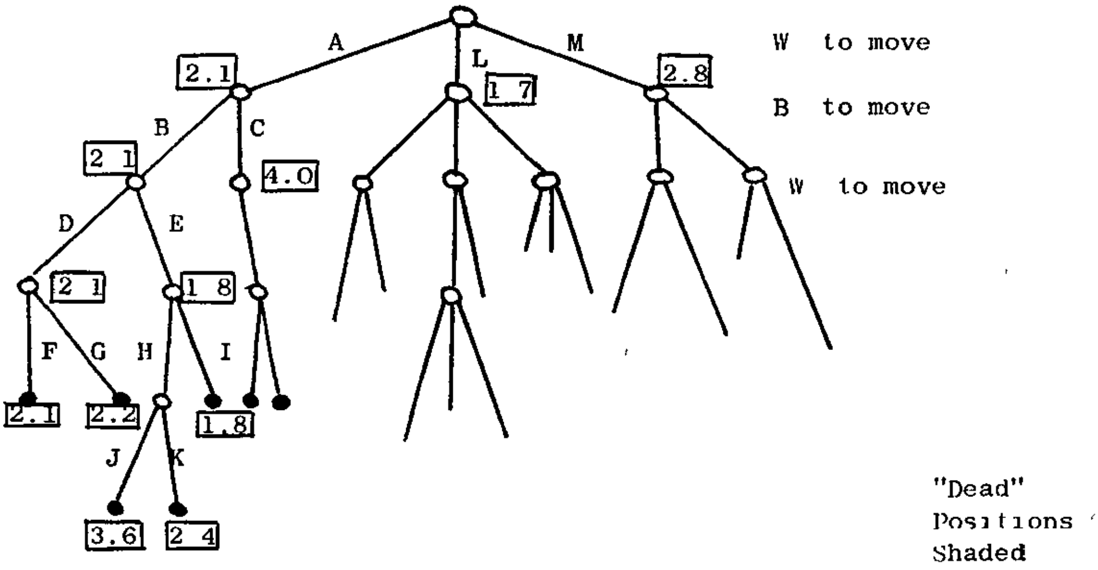
This may be shown diagrammatically as:



Each shaded area represents all the information associated with one position. This is the "INFØRMATIØN" field of mode "PØSITIØN" and will probably contain quite a large number of items of data.

2. Move evaluation

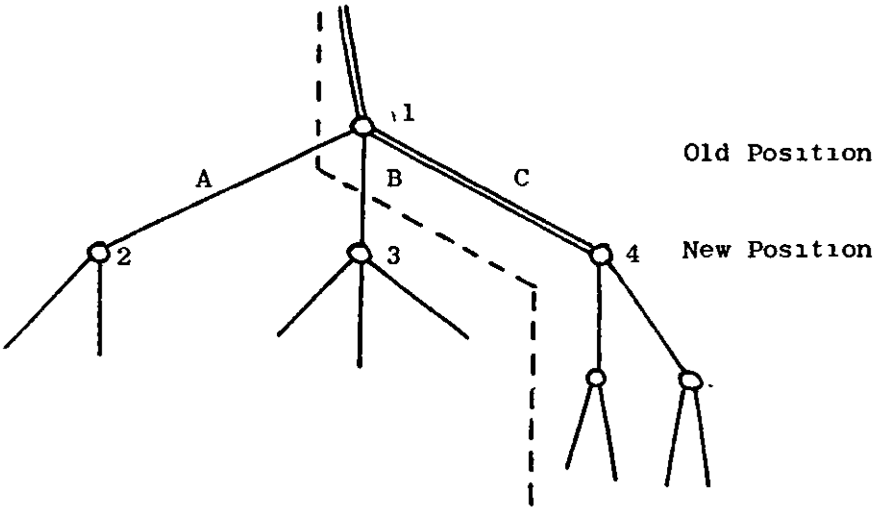
It has been found that, for the class of games considered, a satisfactory method of evaluation depends on working out the continuations of a move to a position that is "dead". By this is meant a position where further exchanges of pieces are unlikely. For the purpose of this discussion, it will be assumed that the evaluation of such a position will give a single number as its result. Looking ahead 2 moves by each player and then exploring non-dead continuations might lead to a tree of possibilities as below:



The values enclosed in rectangles are evaluations made by W (white). High values are desirable. Consider the sequence of moves starting with move A. Initially, only the dead positions are evaluated. The analysis must start at these positions and move gradually to the head of the tree. Consider White's choice between moves J and K. White will naturally chose move J, because this has the maximum value. The position at which this choice is made therefore has the value 3.6. Now consider Black's choice between H and I. Black will choose the minimum value (I) and hence, at this position, the value will be 1.8. This procedure may be repeated until all the positions immediately below the present position have values. On the diagram, White has the choice of move A(2.1), move L(1.7) and move M (2.8) and move M is thus selected. This procedure is known as minimaxing. The point of interest to this discussion is that the value of a position can only be determined with reference to its place in the tree. The machine representation of the problem is therefore superior if it depicts the place in the tree in a simple manner. The branching list structure chosen as the example satisfies this criteria.

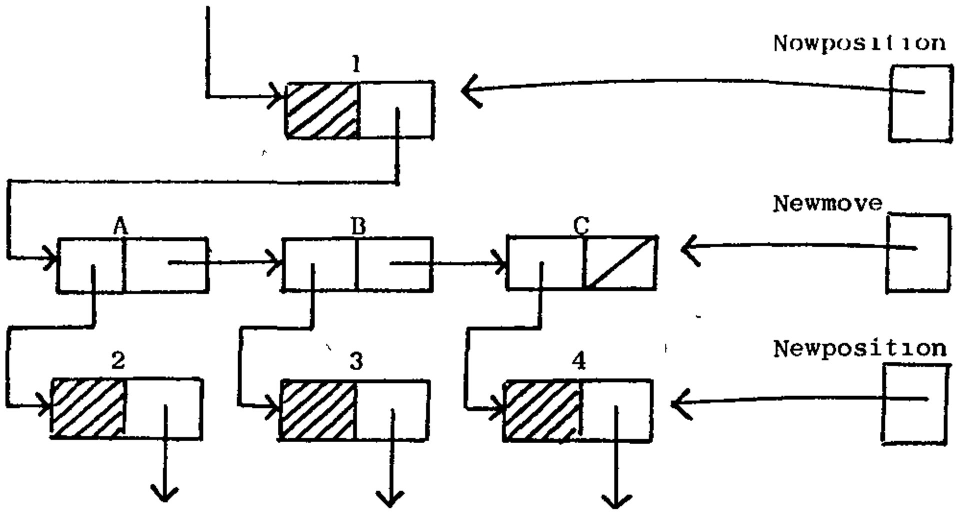
3. Move made

After the evaluation and analysis, the move is made. Using the tree representation as before may lead to the situation below:



Double line indicates actual moves made.
Dotted line separates useful part of
tree from garbage.

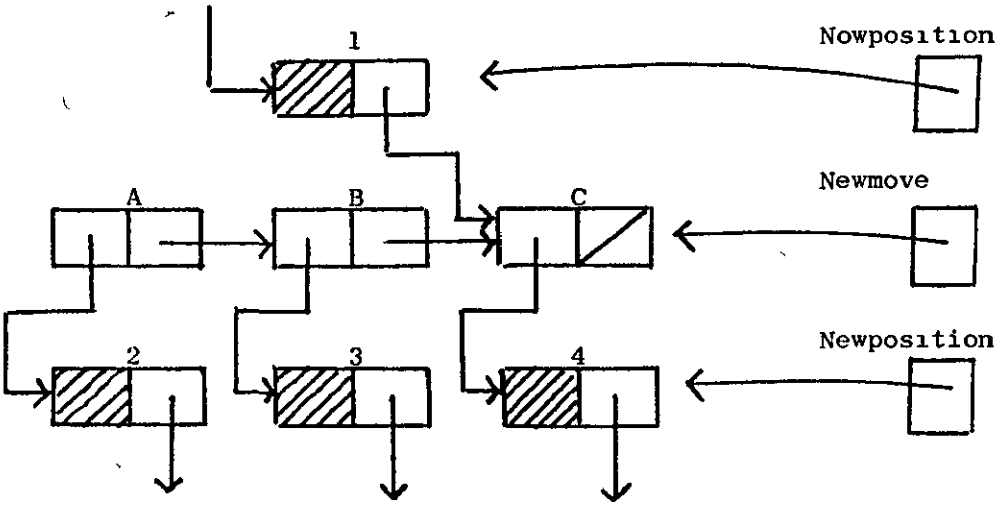
Now all of the tree on the L.H.S. of the dotted line is useless. Since a large amount of space will be wasted, this part of the tree must be abandoned. Consider the computer representation of positions 1, 2, 3 and 4 before the move is made:



Three pointers are shown, "NOWPOSITION", "NEWPOSITION" and "NEWMOVE". The following piece of program makes the unwanted part of the tree completely inaccessible to the pointers and therefore ready for garbage collection.

```
1.          LIST'ØF'NØWPØSITIØN ⟵ NEWMØVE;
2.          NEXT'ØF'NEWMØVE ⟵ 'NIL';
```

Line 2 is not necessary in this case but is provided in casc moves A or B had been selected. The computer representation is now:



Since the garbage collector is completely automatic, the programmer has no problems "marking" which pieces of the tree that he wishes to keep. He is now ready to "make" the move:

```
NØWPØSITIØN ⟵ NEWPØSITIØN;
```

The game now proceeds with the player inputting his move to the machine which then starts the process again (with the proviso always that part of the tree already exists from his previous move <u>if</u> the player made the expected move J.

In summary, the use of a list processing approach has made possible the easy addition of moves, the easy deletion of moves and simplified evaluation due to the preservation of the natural form of the problem in the machine.

A.1.2.2. <u>List Structures used for the Present Study</u>

The work done for the present study involved considerable use of list-processing techniques. The MODE's used to build the structure are shown in Appendix 3, LISTING 8. TABLE A.1.1. contains brief description of the function of each MODE. The complete plant model used is one large structure containing a large number of pointers to other parts of the model. Some pointers used are redundant, in a strict sense, since they only contain information that may be obtained indirectly from two or more other pointers. Such pointers are only useful when it is convenient to save computation time at the expense of core store space.

FIG. A.1.2. shows the form of the basic structures used for the study. In these diagrams each "box" in the structure corresponds to 1 field in the MODE declaration. If the field is itself a structure, the box contains the number of the MODE in TABLE A.1.1. REFerences to other modes are shown by an arrow drawn from the middle of the appropriate box. If the MODE referred to in this way is not shown, the MODE identification number for TABLE A.1.1. is shown next to the arrow. Each different type of MODE shown in the structure is drawn as a horizontal rectangle and is labelled once. Arrays are shown as vertical rectangles and are not labelled.

No.	General Notes and MODE name	Field	Function
1	NAMEVAL (Model variable)	1	Stores one variable name from model
		2	Stores variable value
		3	REFerence to alarm data. NIL if variable not measured.
2	UNIT (Equivalent to unit module)	1	Unit module name
		2	REFerence to list of input streams
		3	REFerence to list of output streams
		4	REFerence to next unit in list of units
		5	REFerence to list of equations
		6	REFerence to stored fault data
		7	REFerence to list of variables not in unit streams
		8	REFerence to list of lists of variables common to several units
3	POINTER (One stream on module)	1	Name of input or output stream
		2	Name of adjacent unit
		3	Name of corresponding stream or adjacent unit
		4	REFerence to next stream in list
		5	REFerence to corresponding stream on adjacent unit
		6	REFerence to array of stream variables
		7	REFerence to unit of which stream is part
4	STANDARDS (Data for measured variable)	1	Alarm type identification number
		2-6	Normal value and alarm limit data
		7	REFerence to unit in which variable is measured
		8	REFerence to list of deduced alarms in which variable is high
		9	REFerence to list of deduced alarms in which variable is low
		10	Alarm message for this variable
5	CONSTANTS (List item for holding variable local to unit)	1	REFerence to next entry in list
		2	Nameval field as specified in 1 above
6	EFFECT (List item used to REFer to deduced alarms)	1	REFerence to next entry in list
		2	REFerence to deduced alarm data

TABLE A.1.1.

Brief Description of Functions of MODES used for this study

No.	General notes and MODE name	Field	Function
7	CONLIST (List of lists)	1	REFerence to list of variables common to several units
		2	REFerence to next entry in list of lists of common variables
8	FAULTVAL (List used to store values)	1	REFerence to next entry in list
		2-3	Old and new variable values
		4	REFerence to variable in question
9	EXPRESSION (Equivalent to one model equation)	1	REFerence to next entry in list of equations
		2	REFerence to variable on LHS of equation
		3	REFerence to normal RHS of equation
		4	REFerence to abnormal RHS of equation (used for faults)
		5	REFerence to real number which may be used to store gradient for differential equations. Contains NIL for algebraic equations
10	EQUATION (List used to construct equation in core store)	1	REFerence to next entry in list
		2	UNION of REFerence to operand in equation REFerence to store denoting a function Operator used in equation
11	CHECK (Head of Deduced Alarm List)	1	REFerence to list of symptoms
		2	Deduced Alarm message
		3	Unique Alarm number
12	CHECKLIST (Deduced Alarm Symptom)	1	REFerence to next entry in list of symptoms
		2	REFerence to variable to which this symptom applies
		3	Code for alarm condition in this list of symptoms
		4	Elapsed time after fault starts before this symptom occurs
		5	REFerence to effect list in which this symptom occurs
13	NODE (Node in Alarm Tree)	1	REFerence to variable corresponding to this node
		2	Identification number
		3	Store for result of last use of scanning program
		4	REFerence to list of possible causes of abnormal value
		5	REFerence to entry in current analysis structure
		6	REFerence to next node in list of nodes

TABLE A.1.1. (Cont.)

No.	General Notes and MODE name	Field	Function
14	BRANCH (List item to store possible causes of abnormality)	1	REFerence to possible cause of abnormality
		2	Factor used to store directional information
		3	Factor used to store magnitude of directional effects
		4	REFerence to next entry in list of branches
15	FAULT (List of off-normal causes)	1	REFerence to list of effects. First entry is prime cause of others
		2	REFerence to next entry in list of faults
16	ALARM (List of effects)	1	REFerence to next entry in list of effects
		2	UNION of {REFerence to node concerned REFerence to deduced alarm concerned
		3	Time at which variable passed alarm limit
		4	Boolean to denote if alarm still active
		5	Boolean to denote if this alarm possible effect of several causes
		6	Complete alarm message for this variable
17	TEMP (List item used to evaluate equations)	1	REFerence to next entry in list
		2	REFerence to last entry in list
		3	Value stored for use in evaluation of expression
18	OPSTACK (Used to store equations)	1	Operator stored
		2	REFerence to next entry
19	SYSLIST (Used to store a particular list of units)	1	REFerence to unit in list of units
		2	REFerence to array used to store steady-state values
		3	Boolean denotation used for "housekeeping" purposes
		4	REFerence to next item in this list
20	ALG (Used for algebraic equations)	1	REFerence to next item in this list
		2	REFerence to variable on LHS of equation
		3	REFerence to RHS of equation
21	SAVE (Used for differential equations)	1	REFerence to next item in this list
		2	REFerence to variable on LHS of equation
		3	REFerence to RHS of equation
		4-10	Real numbers used for integration routine

TABLE A.1.1. (Cont.)

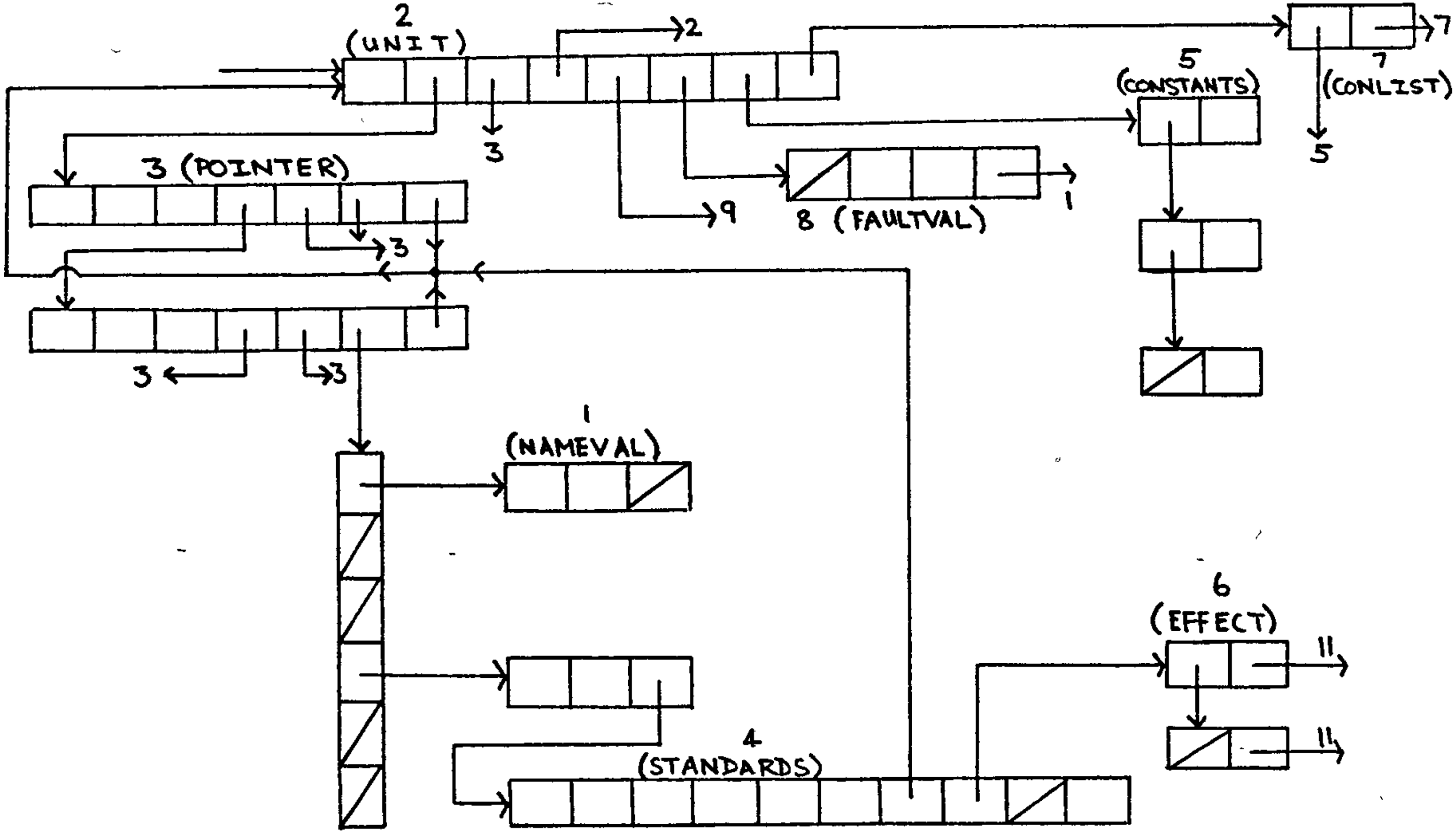


FIG. A.1.2.(a)

Basic Structure of Unit Module Representation used for this Study

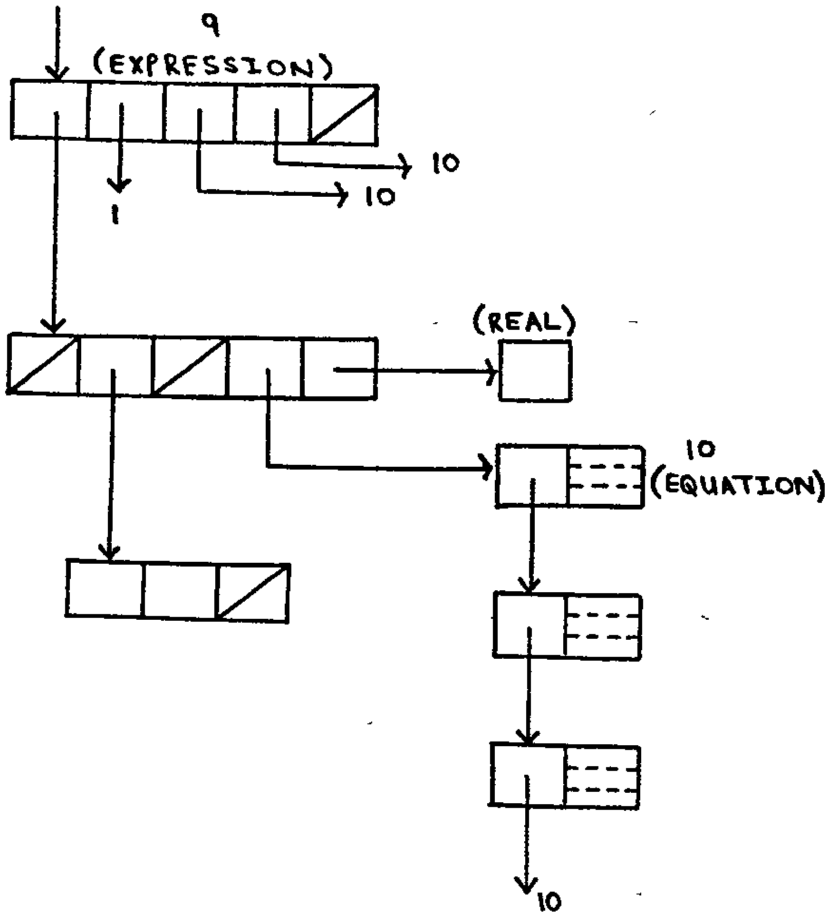


FIG. A.1.2.(b)

Structure of Lıst used for
Equatıons of Model

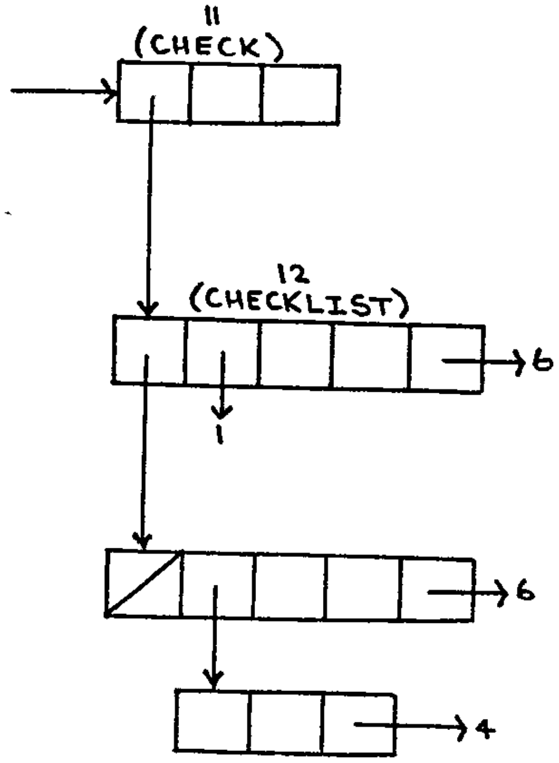


FIG. A.1.2.(c)

Structure used for Storage of
Deduced Alarm Data

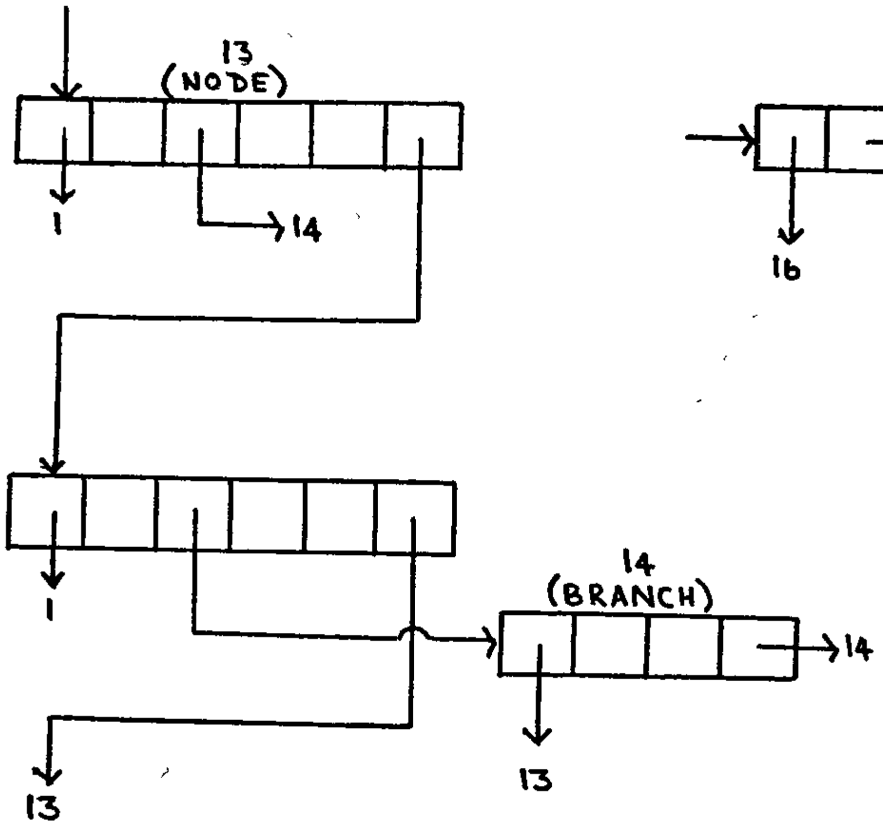


FIG. A.1.2.(d)

Structure used to Represent Alarm Tree

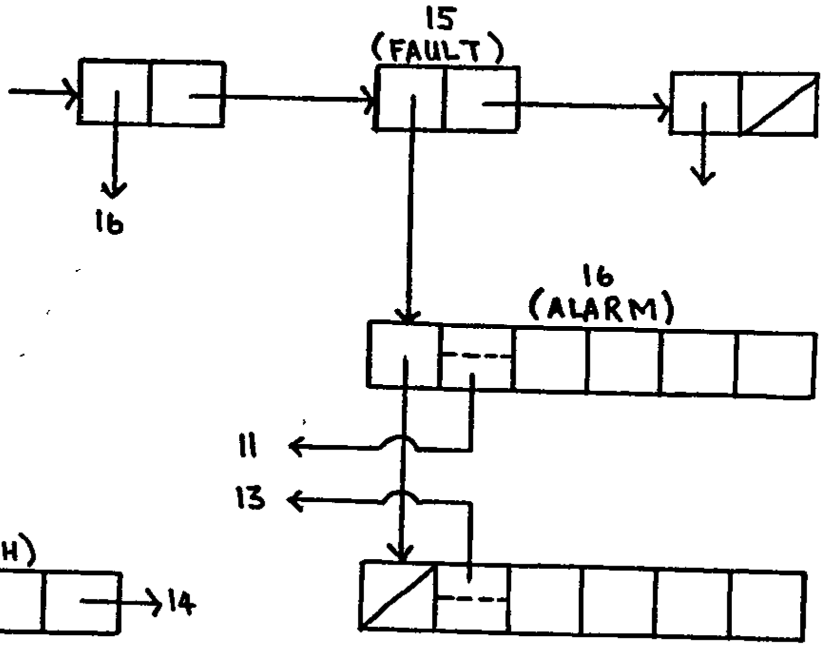


FIG. A.1.2.(e)

Structure used to Store Result of Analysis

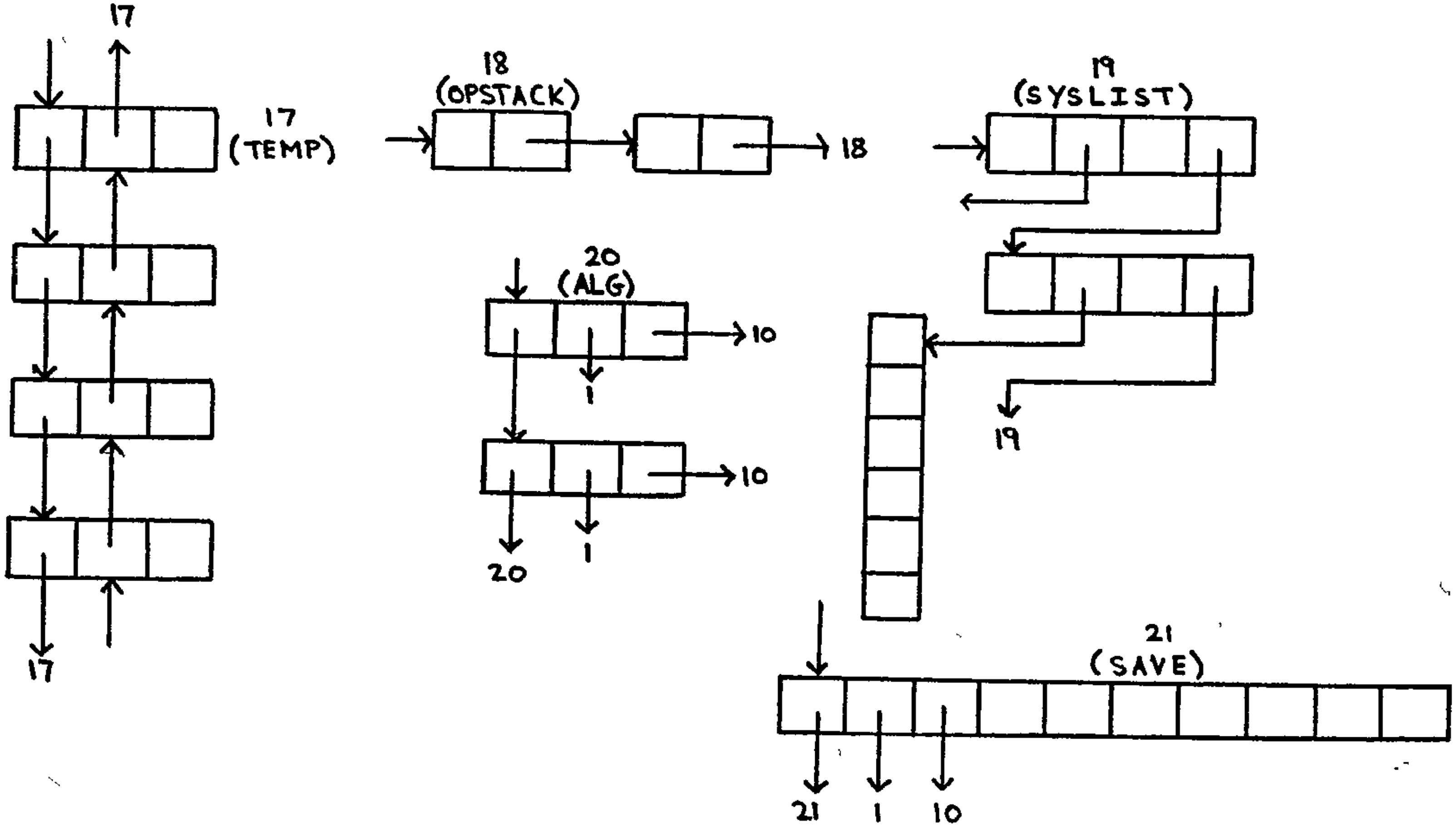


FIG. A.1.2.(f)

Representations of Miscellaneous MODES

A.1.2.3. Simple Program

This subsection contains, as an example of a simple program, a procedure to convert the right-hand side of an expression, written in ordinary algebraic notation, into Reverse Polish notation. Routines of this type are used in compilers in order to translate equations in the program into a form more acceptable to the machine. For the present work, a routine of this type was needed to be supplied by the programmer since the model equations to be used were supplied as data.

Consider the problem of evaluating the right hand side of an expression

$$\mathrm{RHS} = A + B*((C + D/E) - F) \uparrow G$$

Using the rules of algebra, it is simple to show that the order in which the arithmetic operations are performed is not the same as that in which the operators themselves are discovered in a left-to-right scan of the expression:

$$\mathrm{RHS} = A + B*((C + D/E) - F) \uparrow G$$

```
Order of use        6  5   2  1   3    4
of operators
```

Now consider the same expression written in Reverse Polish notation:

$$\mathrm{RHS} = A\ B\ C\ D\ E\ \underset{1}{/}\ \underset{2}{+}\ F\ \underset{3}{-}\ G\ \underset{4}{\uparrow}\ \underset{5}{*}\ \underset{6}{+}$$

Note that i) the operators are now in the correct order

ii) the operands are in the same order as before

iii) the brackets have disappeared entirely.

In order to evaluate this expression, a scan is made from left to right. When an operator is encountered, the previous two operands are used as its arguments. This is shown in the following table:

Operator	First Argument	Second Argument
/	D	E
+	C	D/E
-	(C + D/E)	F
↑	((C + D/E) - F)	G
*	B	((C + D/E) - F)↑ G
+	A	B * ((C + D/E) - F)↑G

The transformation of an expression into its Reverse Polish form may be effected by obeying the following rules as the expression is scanned from left to right:

1) All operands are placed in the Reverse Polish expression as they are encountered,

2) All operators and brackets have priorities associated with them according to the following table:

Operator	Priority
)	0
(	1
+	2
-	2
/	3
*	3
↑	4

An operator "stack" is established which will initially be empty. Operators and left-hand brackets are placed on and removed from the stack, as detailed below, on a first-in, last-out basis.

3) Left-hand brackets are placed on the stack immediately after they are encountered,

4) When an arithmetic operator is encountered, the top of the stack is examined to see if the operator there has a priority higher than or equal to the arithmetic operator. If such an operator is on the top of the stack, then it is transferred to the Polish expression. This process is repeated until no such operator exists on the stack. The operator in the source expression is then placed on the stack.

5) When a right-hand bracket is encountered in the source expression, operators are successively removed from the stack and placed in the Polish expression until a left-hand bracket is left on top of the stack. Both brackets are now discarded.

6) When the scan of the source expression is complete, the contents of the operator stack are added one by one to the Polish expression.

Fig. A.1.3 shows various stages in this process diagrammatically for the example given earlier.

Fig. A.1.4. shows a flowchart showing the logic of the program and TABLE A.1.2 shows some expressions and their Polish notation equivalents produced by the program. Note that the variable names used in this table all consist of 2 characters. This is standard for the form of the program used for the present study. A listing of the program used for the study is included in Appendix 3, LISTING 9.

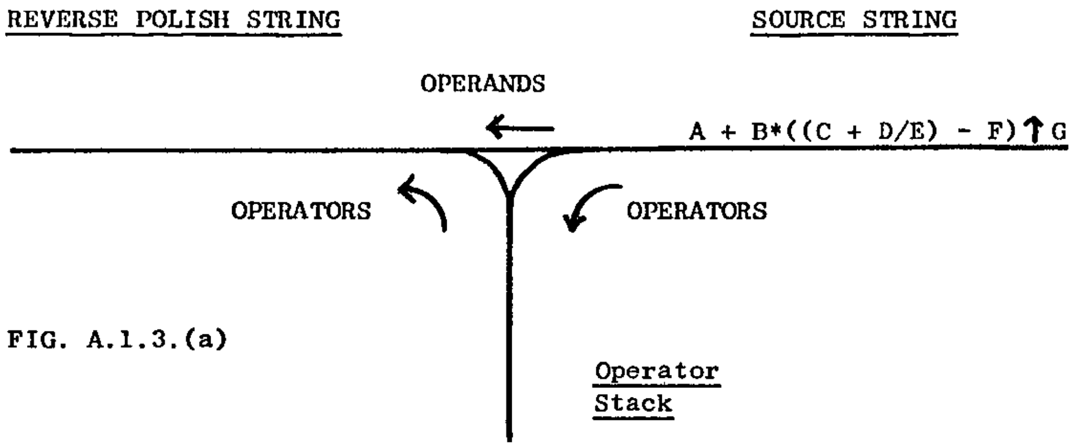


FIG. A.1.3.(a)

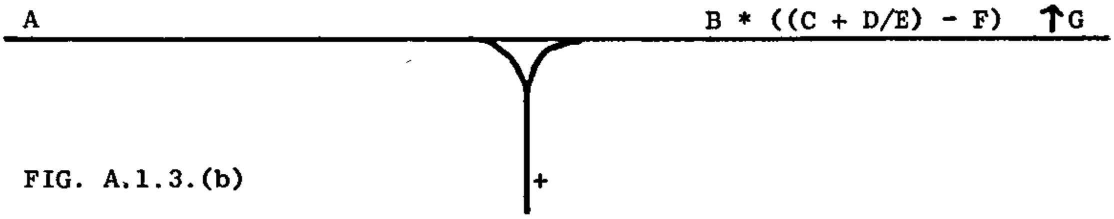


FIG. A.1.3.(b)

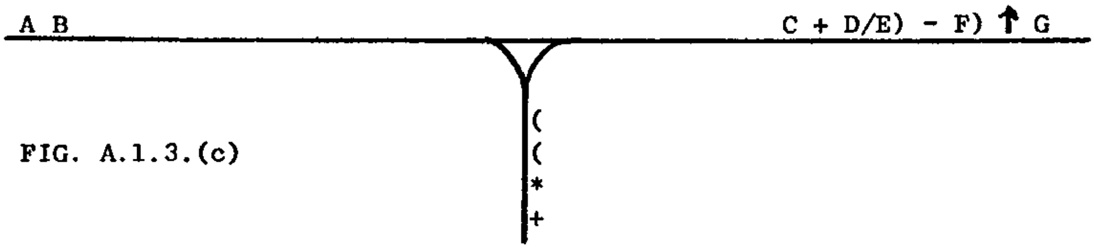


FIG. A.1.3.(c)

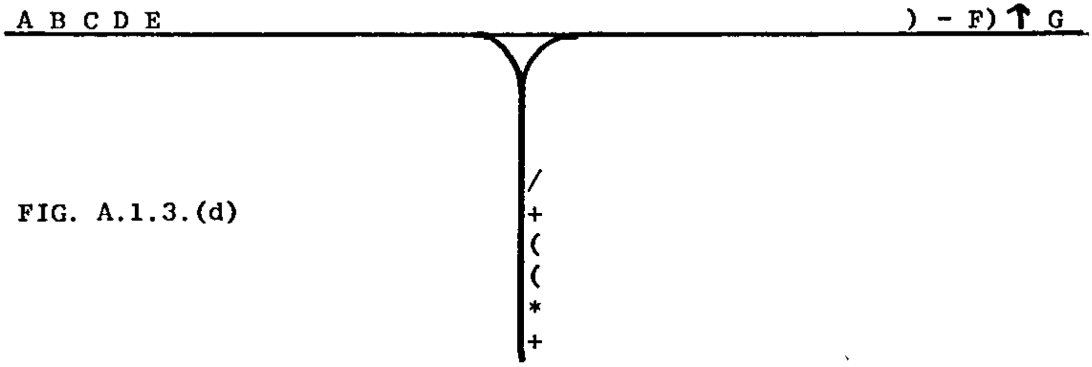


FIG. A.1.3.(d)

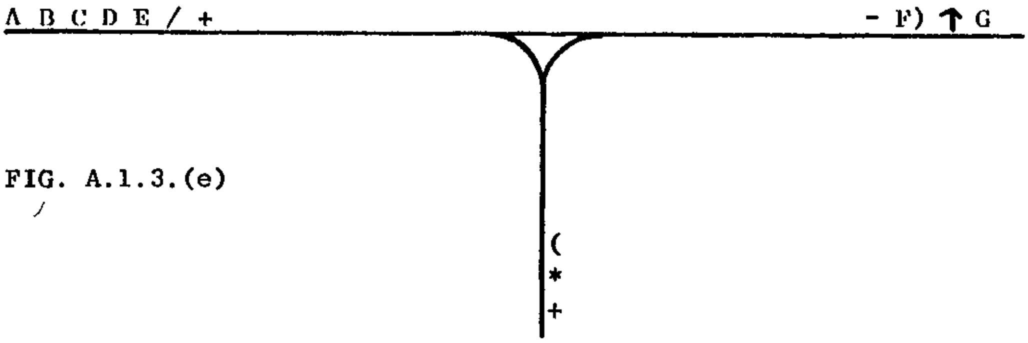


FIG. A.1.3.(e)

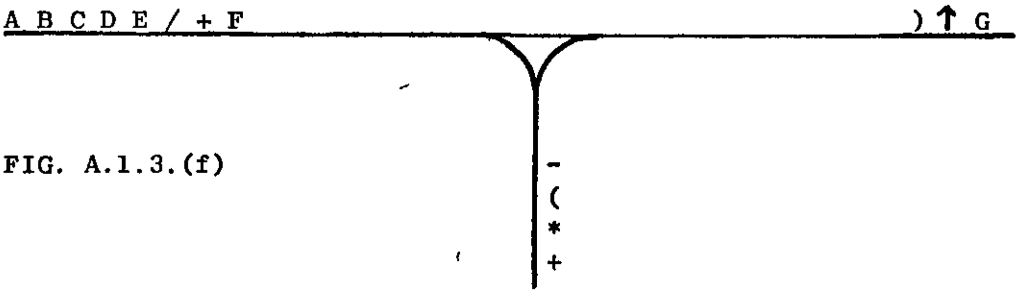


FIG. A.1.3.(f)

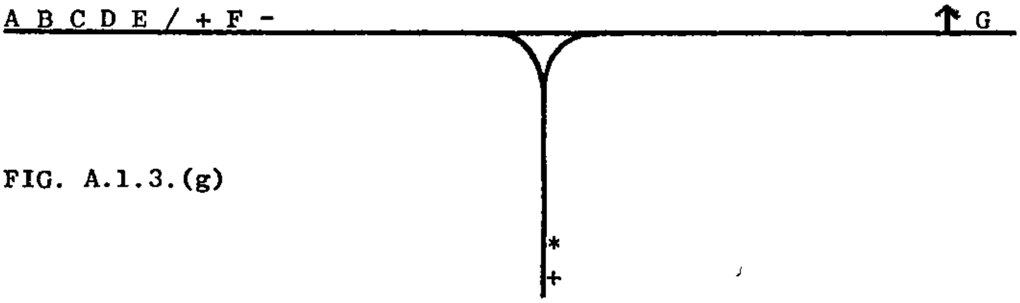


FIG. A.1.3.(g)

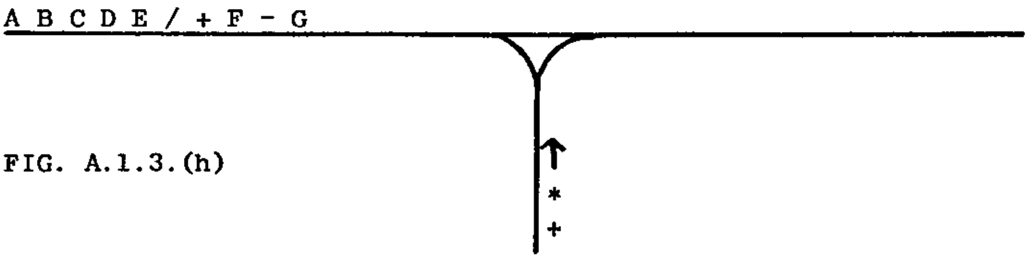


FIG. A.1.3.(h)

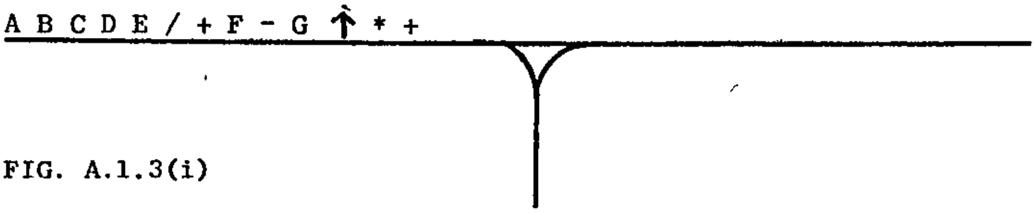


FIG. A.1.3(i)

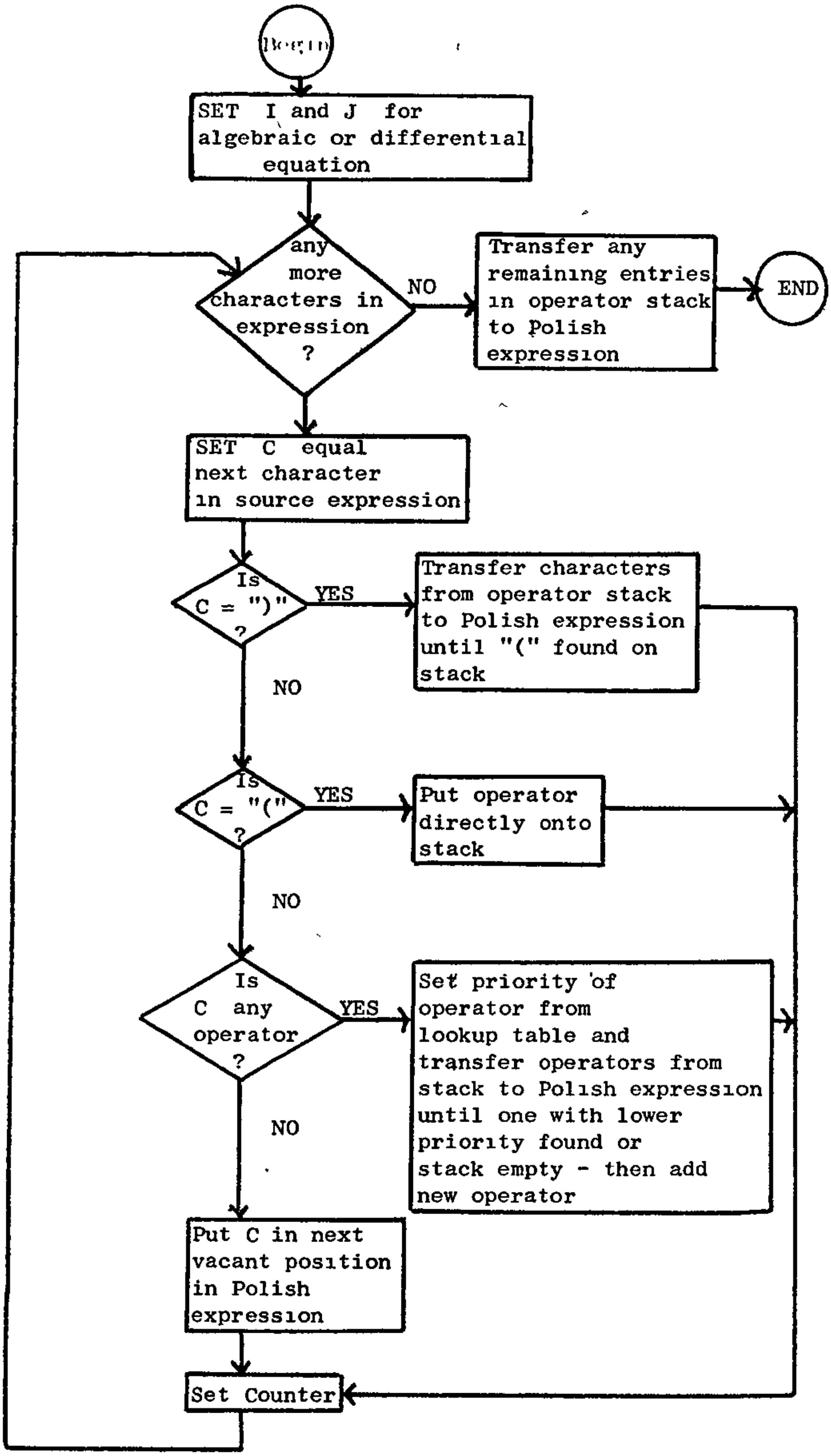

Begin
SET I and J for algebraic or differentıal equation
any more characters in expression ?
NO
Transfer any remainıng entries ın operator stack to Polish expressıon
END
SET C equal next character ın source expression
Is C = ")" ?
YES
Transfer characters from operator stack to Polish expression until "(" found on stack
NO
Is C = "(" ?
YES
Put operator directly onto stack
NO
Is C any operator ?
YES
Set priority of operator from lookup table and transfer operators from stack to Polısh expressıon until one with lower priorıty found or stack empty - then add new operator
NO
Put C in next vacant posıtion in Polish expressıon
Set Counter

FIG. A.1.4.

Logıc of Program used to Convert Expressıons written ın Standard Algebraic Form into Reverse Polish Notatıon

D(XC)/DT ← (QD*(XD-XC)+QA*(XA-XC)-QB*(XB-XC))/HT	D(XC)/DT ← QDXDXC-*QAXAXC-*+QBXBXC-*-HT/
D(QC)/DT ← (QA-QC)/WL	D(QC)/DT ← QAQC-WL/
TC ← A0+XC*(A1+XC*(A2+XC*(A3+XC*A4)))	TC ← A0XCA1XCA2XCA3XCA4*+*+*+*+
XB ← B0+XC*(B1+XC*(B2+XC*(B3+XC*B4)))	XB ← B0XCB1XCB2XCB3XCB4*+*+*+*+
ZS ← XC*SF	ZS ← XCSF*
QB ← QD	QB ← QD
TB ← TC	TB ← TC

TABLE A.1.2

Expressions converted from standard form into Reverse Polish Notation

REFERENCES

1. McCarthy, J., "Recursive functions of symbolic expressions and their computation by machine" (The LISP Programming System). Quarterly Progress Report No. 53, Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, Massachusetts, April 15, (1959).

2. Newell, A., Tonge, F.M., Feigenbaurn, E.A., Mealy, G.H., Saber, N., Green, B.F., and Wolf, A.K. "Information Processing Language V Manual, Section I: The elements of I P L Programming", The Rand Corporation Paper P-1897,(1960)

3. Woodward, P.M., and Bond, S.G., "Users Guide to ALGOL 68-R", 1st edition (1971), Royal Radar Establishment, Ministry of Defence, Malvern, Worcs.

4. Van Wijngaarden, A., Mailloux, B.J., Peck, J.E.L., and Koster, C.H.A., "Report on the Algorithmic Language ALGOL 68", Numerische Mathematik, 14, 79-218, (1969)

5. Foster, J.M., "List Processing", MacDonald, London, (1967)

6 Prentice, J.A., "The use of ALGOL 68-R under George 2 L", Loughborough University of Technology Computer Centre Document AL/RE/242, (1972)

APPENDIX 2

INSTRUCTIONS FOR PROGRAM USE AND DATA FORMAT

TABLE OF CONTENTS

APPENDIX 2

INSTRUCTIONS FOR PROGRAM USE AND DATA FORMAT

It was noted in Appendix 1 that ALGOL 68-R programs may be assembled from previously-compiled segments stored in an album. The data space available to the user depends on the program size since, for a "normal" run under the Loughborough University Computer Centre version of GEORGE 2, the total amount of core store available is 48 K words. In most cases, a particular run of an alarm-analysis program required only a limited number of the program facilities available. It was therefore decided to organise the program into a form that would allow a particular set of facilities to be obtained by assembling the appropriate segments. The contents of the segments were chosen so that the final program was of a minimum size.

All user-supplied information is supplied in the form of data. The program facilities used for a particular run are controlled by means of "keywords" in the data. Each keyword causes the program to read in data or perform some operation on the data already read in or both. After the program has performed the task corresponding to a keyword, a further keyword is read in. In order to operate this system, a small control segment is required. The only function that this segment performs is that of reading in keywords and translating each keyword into calls to the appropriate procedures. Each program segment has one or more keywords associated with it. The segments provided are each described in the sections which follow. Each section is subdivided to deal with the individual keywords. Table A.2.1 lists the keywords applicable to each segment, their default values and modes. Each keyword nominally consists of 12 characters. No blanks must appear before or in the middle of keywords but the requisite number of trailing blanks must be used when data input follows on the same card as a keyword consisting of less than 12 non-space characters.

The complete program required for a particular alarm analysis job is assembled from the required segments as shown in Appendix 1.

Segment	Keyword	For keywords that correspond to variables		Section in this Appendix
		Mode	Default Value	
MODES	DEBUG	Boolean	True	A.2.1.1.
	LOOKATNO	Integer	100	A.2.1.2.
	RESET	Boolean	True	A.2.1.3.
	CHECKNO	Integer	1	A.2.1.4.
	SMALLSTEP	Real	$1.0 * 10^{-8}$	A.2.1.5.
	NO	Integer	10	A.2.1.6.
	MAXLEVEL	Integer	2	A.2.1.7.
	HOWMANY	Integer	2	A.2.1.8.
	TOLERANCE	Real	$1.0 * 10^{-6}$	A.2.1.9.
	PRINTNO	Integer	100	A.2.1.10.
	NODENO	Integer	1	A.2.1.11.
	MOREUNITS	Boolean	True	A.2.1.12.
	TARGET	Integer	10	A.2.1.13.
	FUNCMODELS	Boolean	False	A.2.1 14.
SEG1	MASTERMODEL	-	-	A.2.2.1.
	SYSTEMDATA	-	-	A.2.2.2.
SEG2	ADDALARMS	-	-	A.2.3.1.
	INPUTCHECKS	-	-	A.2.3.2.
	RUNSYSTEM	-	-	A.2.3.3.
SEG3	MASTEREFFECT	-	-	A.2.4.1.
SEG4	SETUPTREE	-	-	A.2.5.1.
	ALTERTREE	-	-	A.2.5.2.
	PRINTTREE	-	-	A.2.5.3.
SEG5	TESTREE	-	-	A.2.6.1.
PSEG	PRINTMODELS	-	-	A.2.7.1.
	PRINTUNIEFS	-	-	A.2.7.2.

TABLE A.2.1.

Keywords for Use with Program Segments

A.2.1 Segmont MODES

This segment contains:

1. The declarations for all non-standard modes and operators used ın the program.

2. The declarations of all "global" variables. (Those that are used non-locally ın the program).

The keywords applicable to thıs segment are all used to alter the values of global varıables. Each keyword consısts of the same characters as the identıfıer of the varıable concerned (e.g. keyword DEBUG is used to alter variable DEBUG). The subsectıons which follow treat the keywords in turn. Examples of the use of each keyword applicable to this segment are given in Appendix 3, LISTING 1.

A.2.1.1 Keyword DEBUG

Thıs keyword ıs used to set the value of variable DEBUG to one of the boolean values TRUE or FALSE. When the value ıs TRUE the integration routine gıves extra output desıgned to assist ın tracing errors ın the approximate "steady state" data previously input to the program. The default value of the varıable is TRUE. As with all keywords which are used to define values of global varıables, the input value should appear on the same card as the keyword starting after column 12.

A.2.1.2 Keyword LOOKATNO

This allows the number of columns on the line-printer used for output to be specified in the input data. The default value is 100.

A.2.1.3 Keyword RESET

If the global variable corresponding to this keyword is set to FALSE the model variables will not be reset to their initial values after the integration routine is used. This facility allows the program to be used to find the true steady-state from the approximate steady-state values fed in by the user. The default value is TRUE.

A.2.1.4 Keyword CHECKNO

For convenience, a unique reference number may be assigned to each set of deduced alarm symptoms. These symptoms may be generated in different program runs and a facility is therefore needed to allow the user to input the next number to be used during a program run. After each set of symptoms is found the program updates the value of the variable CHECKNO by 1. The default value is 1.

A.2.1.5 Keyword SMALLSTEP

This keyword allows the input of the initial stepsize to be used by the integration routine. The default value of the variable SMALLSTEP is 10^{-8}.

A.2.1.6 Keyword NO

This keyword allows the user to set the number of calls to the integration routine by the procedure TESTREE. The default value of the variable NO is 10.

A.2.1.7 Keyword MAXLEVEL

This keyword controls the printing of the alarm tree as performed by PRINTREE. The value of the variable MAXLEVEL determines the number of levels of nodes printed out as possible cause nodes for each entry in the list of nodes. The default value is 2. The amount of output produced by the program increases geometrically with the value of variable MAXLEVEL.

A.2.1.8 Keyword HOWMANY

When considering faults whose effects are unlikely to propagate very far from the source of the disturbance during the simulation, the program selects a system of equations that apply to units in the vicinity of the fault. The number of units selected will depend on the value of the variable HOWMANY. With HOWMANY set to 0, only the units adjacent to that in which the disturbance occurs will be considered. Higher values of the variable cause the program to search for loops formed by the links between units. As an example with HOWMANY set to 1 any unit which is directly connected to at least 1 output of a unit already included and 1 input of a unit already included will itself be included in the system. With HOWMANY set to 2, loops with 2 adjacent units not already in the system will be included. During the integration of the set of equations, the program checks that variables in the process streams to and from units outside the system are approximately at their normal values. If deviations from the normal values are found, the integration is stopped, the system of equations enlarged and the integration is automatically restarted from the initial conditions. The default value is 2.

A.2.1.9 Keyword TOLERANCE

This keyword allows the user to set the maximum error allowed before convergence is assumed in the integration procedure MASTERINTEG. The default value is $1.0 * 10^{-6}$. Both the relative error and the absolute error are used during convergence testing.

A.2.1.10 Keyword PRINTNO

This keyword allows the user to set the number of iterations of the integration procedure allowed between output of all variable values. The default value is 100.

A.2.1.11 Keyword NODENO

This keyword allows the user to set the number of the first node in the tree to be created. After each node is created the value of the variable NODENO is increased by unity. The default value is 1.

A.2.1.12 Keyword MOREUNITS

When the boolean variable MOREUNITS is set to 'FALSE' then no extra units will be added to the system during integration even if the variables in units outside the system change significantly. This facility is intended for use when MASTEREFFECT is used to find controller settings. The default value is 'TRUE'.

A.2.1.13 Keyword TARGET

This keyword is used to control the number of calls to the integration routine during use of MASTEREFFECT. The symptoms found by this procedure are stored and the integration terminated if the following condition is true:

$$\begin{array}{c}\text{Number of symptoms}\\ \text{found}\end{array} \times \begin{array}{c}\text{Number of calls of}\\ \text{integration program}\end{array} > \text{TARGET}$$

The default value is 10.

A.2.1.14 Keyword FUNCMODELS

This keyword allows the user to specify that a model supplied is in functional form. With the boolean variable FUNCMODELS set to 'TRUE' a different method will be used for assembling the alarm tree in procedure SETUPTREE. The default value is 'FALSE'.

A.2.2 Segment SEG1

This segment contains:

1. Utility procedures used throughout the program,
2. The procedures used to set up the plant model in core store and initialise variable values.

The keywords used to initiate calls to procedures in this segment are MASTERMODEL and SYSTEMDATA. In both cases, the procedure called has an identifier consisting of the same characters as the keyword.

A.2.2.1 Keyword MASTERMODEL

As might be expected from the name, this keyword controls the setting-up of the plant model to be used by the program.
FIG. A.2.1 shows the main routines used by the program for this purpose.
Routines LISTUNIT, READMODEL and PREPSOURCE are all used to input data to the program and a series of data cards is therefore expected following any card with this keyword on it.

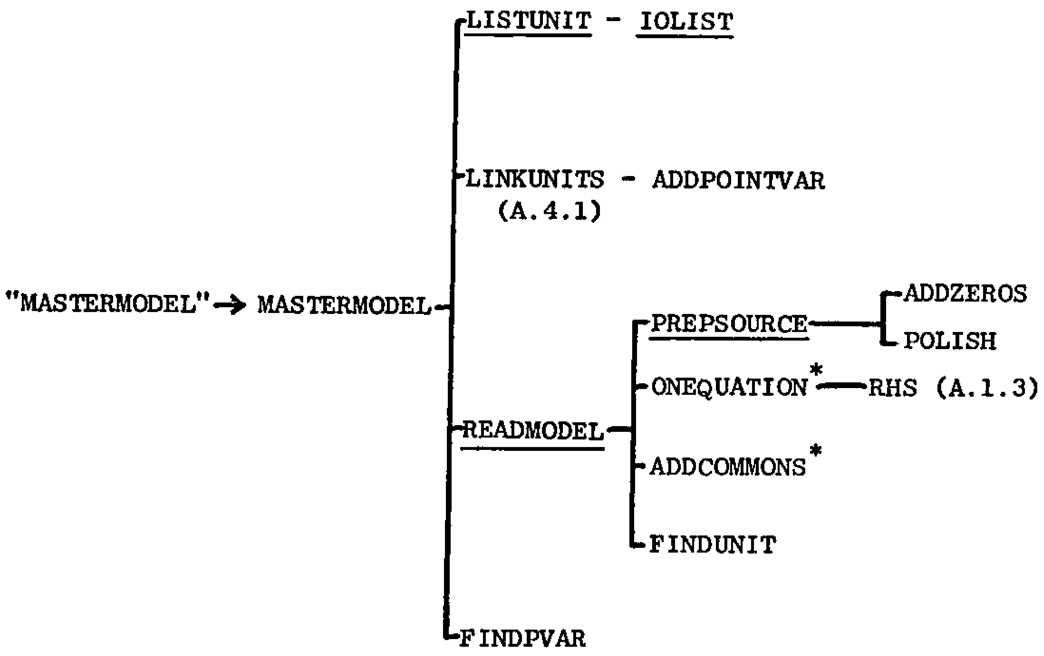


FIG. A.2.1

Main Procedures called when keyword MASTERMODEL is used

The nomenclature used for the above dıagram is as follows:

1. The keyword is shown ın quotes. An arrow shows the call initıally made after use of the keyword.

2. Procedures that require input data are underlined.

3. Recursive procedures are marked with an asterisk.

4. If a flowchart for the procedure is gıven in Appendıx 4 (or elsewhere), then the section number is given underneath the procedure name.

This nomenclature will be used throughout this Appendix.

The cards in the data that follow the keyword may be divided into 2 sets:

1. The set that supplies the data for procedure LISTUNIT.

2. The set that supplies the data for procedure READMODEL. (The input of data by PREPSOURCE is controlled by READMODEL).

The procedure LISTUNIT translates the input data giving details of unit names used in the model and the links between the units into the skeleton plant model. Four "keyletters" are used to control the input of this information. The keyletters are U, I, O and E corresponding to Unit, Input, Output, and End. Each keyletter is the only character that is read on its respective card and must be in column 1. The use of the four keyletters may now be considered in turn.

a) Keyletter U

This keyletter tells the program that the following card contains a module unit name in columns 1-4. Any of the 64 characters in the ALGOL 68-R character set may be used for the name. The unit name must always precede the data on inputs and outputs. The first card read by LISTUNIT must therefore contain this keyletter. Subsequent use of the keyletter will cause the program to assume that all information on inputs and outputs of the preceding unit has been read in. The unit so defined will then be created in core store.

b) Keyletter I

This keyletter is used to inform the program that one or more cards follow, each containing details of one input stream on the unit module currently considered. The format of these cards is given below.

Column(s)	Contents
1	Identification character of input stream
2	Blank
3-6	The name of the unit to which the stream is linked
7	Blank
8	Identification character of corresponding output stream of unit in cols. 3-6.
9	Blank
10	Contains a "," if further inputs follow on succeeding cards. Any other character (including "space") causes the program to expect a keyletter on the next card.

c) Keyletter O

This keyletter causes the program to expect details of one or more output streams of the unit currently considered. The format of the data cards is the same as those used for keyletter I except that when reading details of column contents, the word "output" should be substituted for "input" wherever encountered and vice-versa.

d) Keyletter E

This keyletter causes the program to create the unit module previously considered and to then return control to procedure MASTERMODEL.

During execution of LISTUNIT, the details of the skeleton plant model as read in are printed out.

The procedure READMODEL is responsible for reading in the model equations corresponding to each unit and assembling them in a form convenient for use in the computer core-store. The input of this data is controlled by means of "dollarwords". Each dollarword nominally consists of 12 characters (including the requisite number of trailing blanks as with keywords). The dollarword starts in column 2. Column 1 must contain a $ sign. The functions of the various dollarwords are given below.

a) Dollarword UNITNAMES

This dollarword causes the program to expect a list of names of units on the following card(s). The program stores this list of names until required by one of the procedures calls initiated by another dollarword. The list of names must start in column 1 of the following card. Each unit name consists of 4 characters and is followed by a comma if further names follow it.

b) Dollarword EQUATIONS

This dollarword causes the program to read in one or more equations. Each equation must start in column 1 of a data card. The program continues reading in equations until a $ character in column 1 of a card is encountered. The type of equations that may be read in is limited by the following rules:

1. All equations must be either algebraic or first order ordinary differential equations (with time as the independent variable).

2. All variables and constants must consist of two characters. The two characters must be alphanumeric. Operator signs and other symbols (including "space") must not be used as variable or constant names.

3. The operators that may be used are +, -, *, /, ↑, @ and £. (The last two operators may be defined in the program if required).

4. Parentheses "(" and ")" are used in the usual way to define the order of operations on the right hand side of equations. On the left hand side, they are used as in the example below to denote a derivative:

D(XC)/DT ← QA(XA - XC)/LC/AT

5. The backarrow symbol " ← " must be used between the left- and right-hand sides of expressions.

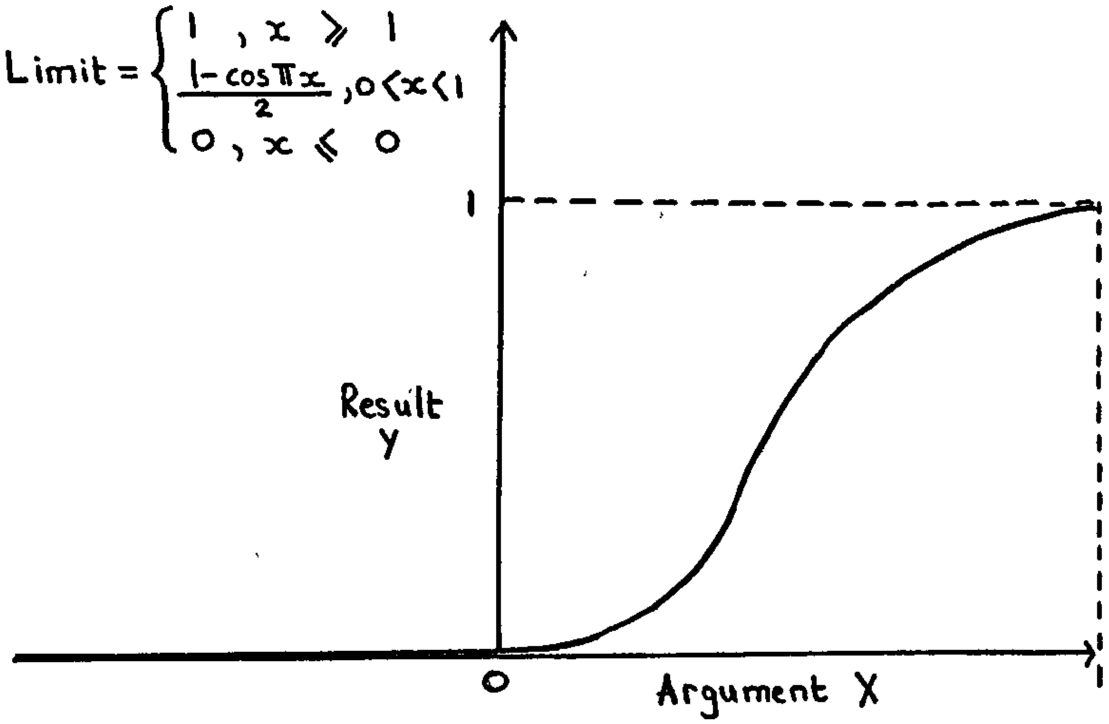
6. A "$" symbol followed by a 3-character name and an operand enclosed in square brackets denotes a function and its argument as below for the use of the trigonometric function sin:

```
PA ← KP + $ SIN [ X3 ]*  K4
```

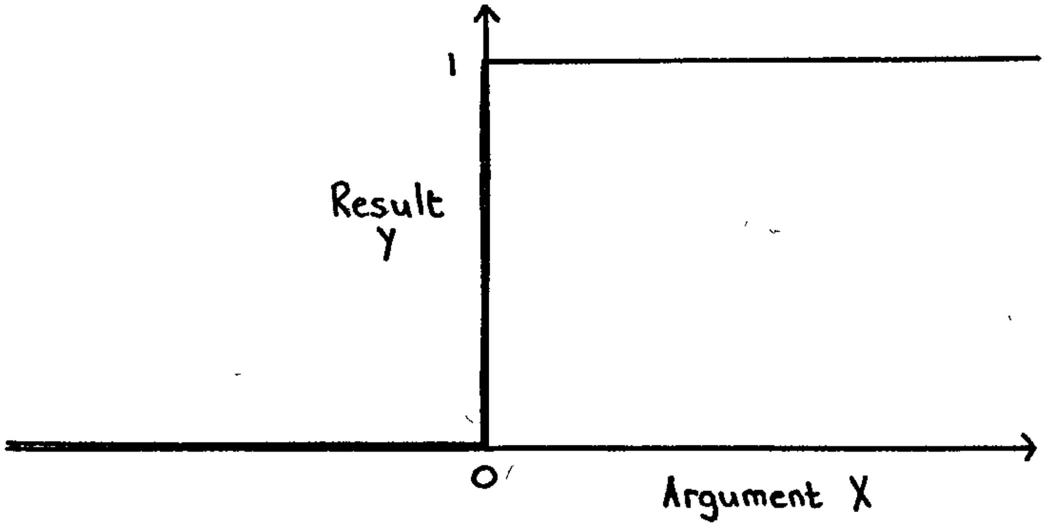
The functions available are:

square root	SQR
exponential	EXP
$\log_e$	LIN
sin	SIN
cos	COS
tan	TAN
arcsin	ASN
arccos	ACS
arctan	ATN
limit	LIM
random	RAN
heavyside	HEV

The arguments of the trigometric functions are in radians. "Limit" and "heavyside" are defined as in FIG. A.2.2. "Random" gives pseudo-random numbers uniformly distributed in the range 0 to 1. The last two functions were only included for completeness.



Function "Limit"



Function "Heavysıde"

FIG. A.2.2.

Defınıtion of the Functions "Lımit" and "Heavysıde"

7. No blanks to be left in equations.

Examples of equations written in this form are shown in Appendix 3, LISTING 2.

After all the equations have been read in and put into Reverse Polish Notation by the program, they are each added to the model of all the units read in after the proceeding occurence of dollarword UNITNAMES.

c) Dollarword COMMONVARS

This facility allows the user to specify that a list of variable names (and their associated values) are common to a list of units. The first variable name must be in columns 1 and 2 of the following card followed by 2 blanks. If further variable names are common, a comma is punched in column 5. Up to 16 variables may be specified using the same format repeated in succeeding 5-column fields across the card. The program creates a list containing each of the variables and sets pointers in core store to allow this list to be accessed from each of the units in the last use of dollarword UNITNAMES.

d) Dollarword LONGEST

This facility allows the user to specify the number of characters in the longest equation to be read in during succeeding uses of dollarword EQUATIONS. The default value is 59.

e) Dollarword ENDREADMODEL

As might be expected, this dollarword causes the program to exit from procedure READMODEL. This facility may only be used once for each occurence of keyword MASTERMODEL in the input data.

An example showing the use of keyword MASTERMODEL is given in Appendix 3, LISTING 2.

A.2.2.2 Keyword SYSTEMDATA

This keyword is used to initialise variable values. The variable names and values are supplied on separate cards. A set of cards are supplied for each unit required during each use of the keyword. The first card in each set must be punched to the format below:

Col(s)	Contents
1-4	Unit name
5	Blank
6	Contains a comma "," if more sets of data follow the set for this unit.

Subsequent cards in each set are punched according to the format below:

Col(s)	Contents
1-2	Variable name
3-5	Blanks
6-N	Value
(N+1)	Blank
(N+2)	Contains a comma if further variables follow in this set

An example showing the use of this keyword is given in Appendix 3, LISTING 3.

A.2.3 Segment SEG2

This segment contains:

1. The integration procedures,
2. The procedures used to define alarms, check variables for alarm conditions and input predefined sets of deduced alarm symptoms,
3. Other miscellaneous procedures that are conveniently stored in this segment.

The keywords applicable to this segment, ADDALARMS, INPUTCHECKS and RUNSYSTEM are described in the subsections which follow.

A.2.3.1 Keyword ADDALARMS

This keyword causes the program to read in the details of the alarms used for the system. The data is arranged in the form of sets corresponding to the unit modules of the model. The first card in each set defines the unit in question and is punched according to the format below:

Col(s)	Contents
1-4	Unit name
5	Contains a comma if more sets follow this one.

The remaining cards in each set define the alarms for particular variables. Each variable is defined by two cards. The first contains the alarm message to be used on the output display. The second must be punched to the format below:

Col(s)	Contents
1-2	Variable name
3-4	Blank
5	A number
6	Blank
7-69	Up to 4 real numbers separated by spaces (see note below)
70	Blank
71	A comma if more variables follow in this set

The contents of columns 7-69 depend on the value of the number in column 5 as defined by TABLE A.2.2.

LISTING 4 in Appendix 3 shows the use of this keyword.

Integer Value	Interpretation of Real Numbers in Columns 7 – 69				Type of Alarm
	1st	2nd	3rd	4th	
1	Blank	Blank	Blank	Blank	Absolute (5% above and below norm)
2	Blank	Blank	Blank	Blank	Absolute (10% above and below norm)
3	Blank	Blank	Blank	Blank	Absolute (20% above and below norm)
4	Low Limit as fraction of normal value	Prelow limit as fraction of normal value	Prehigh limit as fraction of normal value	High limit as fraction of normal value	Deviation
5	Low Limit	Prelow limit	Prehigh limit	High limit	Deviation
6	Normal value	Must be zero	Maximum allowed negative rate of change	Maximum allowed positive rate of change	Rate
7	Low limit	High limit	Blank	Blank	Absolute/ Deviation
8	SPARE				
9	SPARE				

TABLE A.2.2

Definition of Numbers fed in as Alarm Limit Data in Procedure ADDALARMS

A.2.3.2 Keyword INPUTCHECKS

This keyword initiates a call to the procedure of the same name that organises input of predefined sets of deduced alarm symptoms. The data is arranged in sets, each set corresponding to one deduced alarm. The first card in each set contains the deduced alarm message. The second card contains details of the whole set and is punched as below:

Col(s)	Contents
1-5	Integer number used to identify this alarm
6	Blank
7	Number of symptoms in set
8	Blank
9	A comma if there are any further sets

The remaining cards in each set contain details of the symptoms. Each symptom occupies one card and is punched as below:

Col(s)	Contents
1-4	Name of unit in which variable is measured
5	Blank
6-7	Name of variable
8-9	Blank
10-11	Signed Integer denoting alarm condition: -2 for low alarm -1 for pre-low alarm 0 for normal value +1 for pre-high alarm +2 for high alarm
12	Blank
13-N	Real number giving elapsed time after fault deemed to have occured at which symptom appeared.

A.2.3.3 Keyword RUNSYSTEM

This keyword causes the complete plant model to be run using the integration routines controlled by the procedure MASTERINTEG. Only 1 data card is required in addition to the card containing the keyword. This card contains five numbers separated by spaces. The first is the time at which the integration starts, the second the initial steplength, the third is the maximum integration time, the fourth must be a "1" and the fifth sets the number of iterations of the integration routine between printings of all variable values. An example is shown in Appendix 3, LISTING 5.

A.2.4 Segment SEG3

This segment contains the procedures that are used to find the symptom sets for deduced alarms. Only 1 keyword, MASTEREFFECT, is used to control these procedures. The complex call sequence generated by this keyword is shown in FIG. A.2.3.

A.2.4.1 Keyword MASTEREFFECT

The data used with this keyword is supplied in sets. The format of each set is given below. Cards 1, 2 and 4 are mandatory whilst card 3 is only needed if column 6 of card 2 contains the boolean denotation F. Each set of cards defines the abnormal conditions at the start of the program run for the current fault considered. Each occurence of card 2 in a set corresponds to one abnormal condition. If the condition is specified by a variablo valuo change, this information also appears on card 2. If tho condition is specified by a different equation for one of the model variables, then the new equation is given on card 3.

Card	Col(s)	Contents
1	1 - 80	Deduced Alarm message
2	1 - 4	Unit name
	5	Blank
	6	Boolean denotation "T" if condition corresponds to a variable value change or "F" if correlation corresponds to a new equation
	7	Blank
	8	A "T" if any more card 2's in this set If not, an"F".
	9	Blank
	10 - 11	Variable name if "T" in column 6 otherwise these columns and remainder of card blank
	12 - 14	Blank
	15 - N	Real number for new variable value
3	1 - M	New equation if "F" in column 6 of preceding card 2.
4	1	Contains a "T" if more sets of data follow or "F" for no more sets.

An example is shown in Appendix 3, LISTING 6.

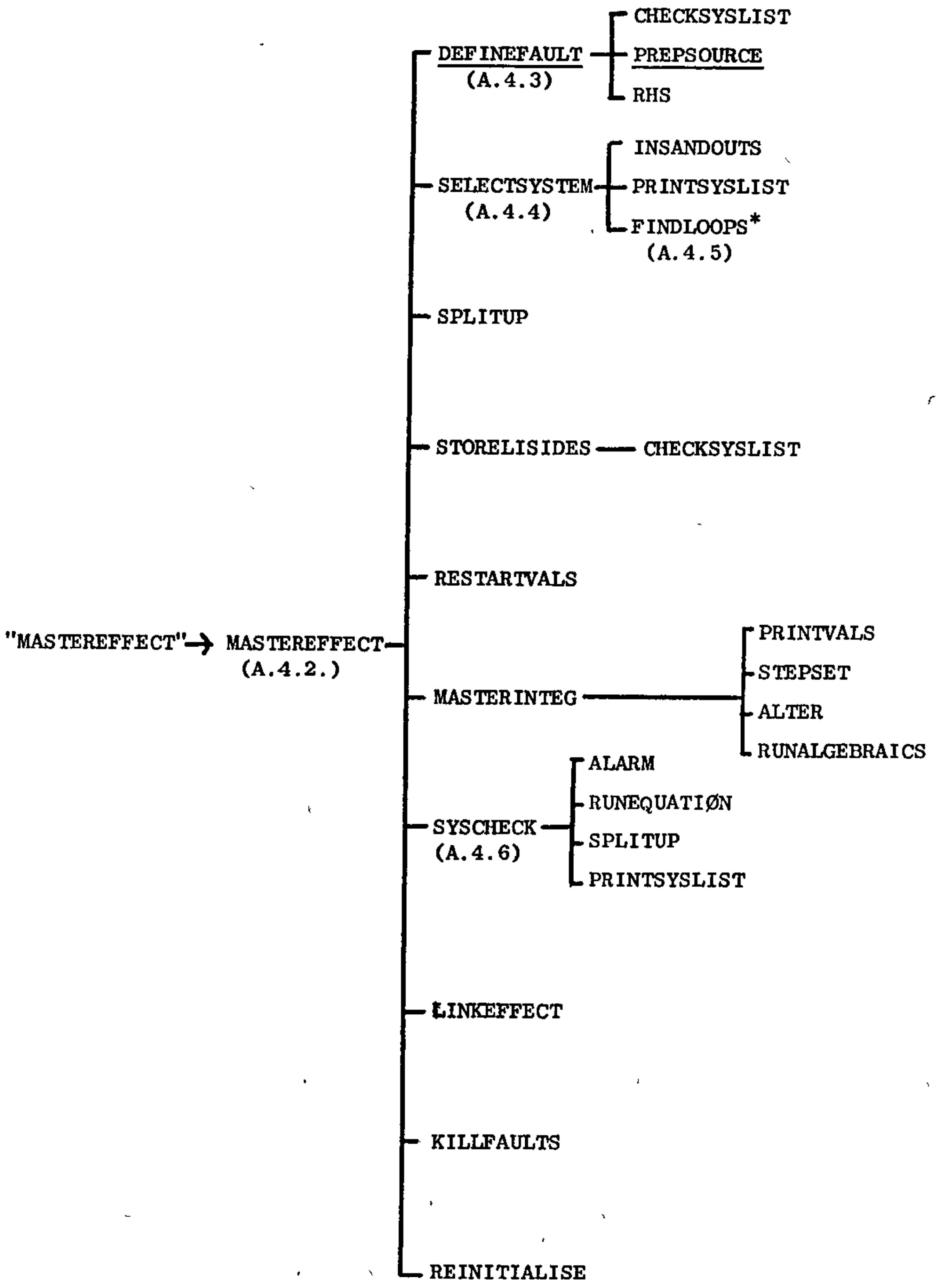


FIG. A.2.3

Main Procedure Calls initiated by Keyword MASTEREFFECT

A.2.5 Segment SEG4

This segment contains the procedures for setting up, modifying and printing out the alarm tree. The three keywords applicable to this segment, SETUPTREE, ALTERTREE and PRINTREE require no data. Examples of the use of these keywords are shown in Appendix 3, LISTING 7.

A.2.5.1 Keyword SETUPTREE

This keyword causes the program to set up the information flow diagram in the form of a branched tree. Directional information is stored with each branch. This tree contains all of the variables in the model.

A.2.5.2 Keyword ALTERTREE

This keyword automatically converts the information flow diagram, set up using SETUPTREE, into the alarm tree. The directional information is modified for use in the alarm tree if necessary.

A.2.5.3 Keyword PRINTREE

This keyword allows the tree produced either by SETUPTREE or ALTERTREE to be listed on the line printer. The number of levels of possible cause nodes given depends on the setting of variable MAXLEVEL. This variable may be set using keyword MAXLEVEL as described in subsection A.2.1.7.

A.2.6 Segment SEG5

This segment contains the procedures to carry out simple tests on the alarm tree produced by the program. Only 1 keyword, TESTREE, is used for this purpose. An example of the use of the keyword is shown in Appendix 3, LISTING 7.

A.2.6.1 Keyword TESTREE

The use of this keyword causes the program to assemble all the model-equations and run the model. Periodically, the simulation is interrupted and the process variables scanned for off-normal values. An analysis is carried out and results printed on the line printer. The series of calls generated by the use of this keyword is shown in FIG. A.2.4.

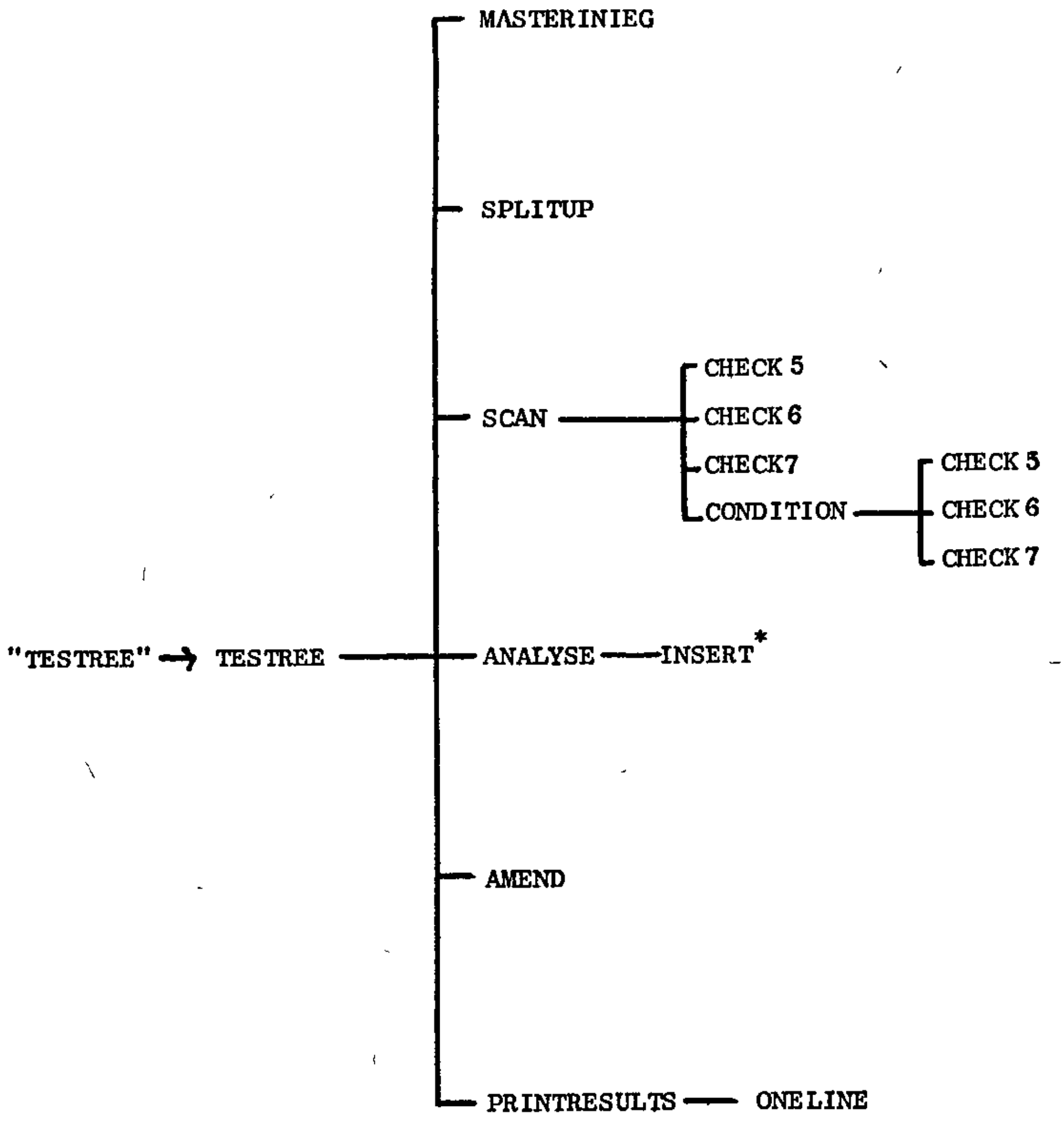


FIG. A.2.4

Procedure Calls generated by Use of Keyword TESTREE

A.2.7 Segment PSEG

This segment contains the output procedures PRINTMODELS and PRINTUNIEFS which are called by the use of keywords of the same name. An example showing the use of these keywords is shown in Appendix 3, LISTING 7. No data is needed with either of these keywords.

A.2.7.1 Keyword PRINTMODELS

This keyword causes the program to print out a list of the units used in the plant model with details of the variables used in input and output streams and the variables that are local to the unit.

A.2.7.2 Keyword PRINTUNIEFS

This keyword causes the program to print out details of all deduced alarm symptoms stored by the program for each of the units in the model.

APPENDIX 3

EXAMPLES OF DATA INPUT AND SEGMENT LISTINGS

TABLE OF CONTENTS

```
DEBUG           T
LOOKATNO        75
RESET           F
CHECKNO         13
SMALLSTEP       1.0 & -2
NO              30
MAXLEVEL        3
HOWMANY         1
TOLERANCE       1.0 & -3
PRINTNO         200
NODENO          70
MOREUNITS       F
TARGET          6
FUNCMODELS      T
```

LISTING 3.1

Examples of Keywords Used with Segment MODES

LISTING 3.2

Example of Use of Keyword MASTERMODEL

```
MASTERMODEL
U
INPT
Ø
A TAN1 1
U
TAN1
I
1 INPT A ,
4 COL1 D
Ø
2 OUTP Z ,
3 PUMP C
U
OUTP
I
Z TAN1 2
U
PUMP
I
C TAN1 3
Ø
D COL1 C
U
COL1
I
C PUMP D
O
D TAN1 4
E
```

```
$LONGEST     35
$UNITNAMES
TAN1
$EQUATIONS
D(L3)/DT←(Q1+Q4-Q2-Q3)/AT
P1← K1*TB/(LM-L3)
Q3←K3*(P1+K4*L3-P3)↑ K5
$UNITNAMES
INPT,TAN1,COL1,PUMP,OUTP
$COMMONVARS
M1   ,M2   ,RO   ,CP
$ENDREADMODEL
```

LISTING 3.2 (contd.)

```
SYSTEMDATA
REAl ,
LA   1.5 ,
QB   15.7 ,
W2   4.943 ,
K1   0.002 ,
P4   1.08
CVO2
QB   15.7 ,
PB   3.12 ,
KB   0.96 ,
KH   0.5
```

LISTING 3.3

Example of Use of Keyword SYSTEMDATA

```
ADDALARMS
VALV,
A101 EXIT PRESSURE ON VALV
PB  1
A102 EXIT FLOWRATE ON VALV
QB  4   0.90   0.95    1.05   1.10
TAN4
A103 LEVEL IN TAN4
LB  7   1.40     1.80
```

LISTING 3.4

Example of Use of Keyword ADDALARMS

LISTING 3.5

Example of Use of Keyword INPUTCHECKS

```
INPUTCHECKS
D403    LOSS OF VACUUM
13     3
COL7 PB  -2 30.0
COND Q4  +2 10.0
COL5 P7  -2 30.0




RUNSYSTEM
0.0     1.0 & -4     2.00.0      1      100
```

```
MASTEREFFECT
D107   COOLING WATER TEMP RISE
INPT T F TW   297.0
F
```

LISTING 3.6

Example of Use of Keyword MASTEREFFECT

```
SETUPTREE
PRINTREE
ALTERTREE
PRINTREE
TESTREE
PRINTMODELS
PRINTUNIEFS
```

LISTING 3.7

Example of Use of Keywords SETUPTREE, ALTERTREE
PRINTREE, TESTREE, PRINTMODELS and PRINTUNIEFS

```
'BFGIN'
'MODE''UNIT';
'MODE''CHFCK';
'MODE''STANDARDS';
'MODE''POINTER';
'MODE' 'EFFECT';
'MODE''ALARM';
'MODE''BRANCH';
'MODE''NAMEVAL' = 'STRUCT'('BYTES' NAME,
                           'REAL' VALUE,
                           'REF''STANDARDS' STAND);
'MODE''CHFCKLIST' = 'STRUCT'('REF''CHECKLIST' NEXCH,
                             'REF''NAMEVAL' NAVAL,
                             'INT' COUNT,
                             'REAL' TIME,
                             'REF''EFFECT' EFFECT);
'MODE''EFFFCT' = 'STRUCT'('PEF''EFFECT' NEXEF,
                          'REF''CHECK' CHECK);
'MODE''CHECK' = 'STRUCT'('REF''CHECKLIST' NEXCH,
                         'STRING' MESSAGE,
                         'INT' NO);
'MODE''POINTER' = 'STRUCT'('CHAR' NAME,
                           'BYTES' LIST,
                           'CHAR' CAR,
                           'REF''POINTER' NEXP,LINK,
                           'REF'[]'REF''NAMEVAL' VARI,
                           'REF''UNIT' UNIT);
'MODE''CONSTANTS' = 'STRUCT'('REF''CONSTANTS' NEXCON,
                             'NAMEVAL' NAVAL);
'MODE''OPSTACK' = 'STRUCT'('CHAR' CHAR,
                           'REF''OPSTACK' NEXT);
'MODE''FUNCTION' = 'STRUCT'('INT' NO,
```

LISTING 3.8

Segment MODES

```
                                  'REF''NAMEVAL' ARG);
'MODE''EQUATION' = 'STRUCT'('REF''EQUATION' NEXT,
                            'UNION'('REF''NAMEVAL',
                                    'REF''FUNCTION',
                                    'CHAR')CONTENTS);
'MODE''EXPRESSION' = 'STRUCT'('REF''EXPRESSION' NEXTEX,
                              'REF''NAMEVAL' LHS,
                              'REF''EQUATION' NORMAL,ALTER,
                              'REF''REAL' GRADIENT);
'MODE''CONLIST' = 'STRUCT'('REF''CONSTANTS' CON,
                           'REF''CONLIST' NEXLIST);
'MODE''FAULTVAL' = 'STRUCT'('REF''FAULTVAL' NEXT,
                            'REAL' OLD,NEW,
                            'REF''NAMEVAL' NV);
'MODE''UNIT' = 'STRUCT'('BYTES' UNAME,
                        'REF''POINTER' INLIST,OUTLIST;
                        'REF''UNIT' NEXU,
                        'REF''EXPRESSION' RELATIONS,
                        'REF''FAULTVAL' FAULTVALS,
                        'REF''CONSTANTS' CON,
                        'REF''CONLIST' COMMONS);
'MODE''STANDARDS' = 'STRUCT'('INT' TYPE,
                             'REAL' A,B,C;D,NORM,
                             'REF''UNIT' UNIT,
                             'REF''EFFECT' HIEFLIST;LOEFLIST,
                             'STRING' ME);
'MODE''TEMP' = 'STRUCT'('REF''TEMP' NEXT,
                        'REF''TEMP' LAST,
                        'REAL' VALUE);
'MODE''SYSLIST' = 'STRUCT'('REF''UNIT' SYSU,
                           'REF'[ ]'REAL' STORE;
                           'BOOL' INSYSTEM,
                           'REF''SYSLIST' NEXS);
```

LISTING . 3.8 (contd.)

LISTING 3.8 (contd.)

```
'MODE''ALG' = 'STRUCT'('REF''ALG' NEXALG,
                       'REF''NAMEVAL' LHS,
                       'REF''EQUATION' RHS);
'MODE''SAVE' = 'STRUCT'('REF''SAVE' NEXSAVE,
                        'REF''NAMEVAL' VARIABLE,
                        'REF''EQUATION' RHSIDE,
                        'REAL' K1,K2,K3,K4,K5,E,YS);
'MODE''NODE' = 'STRUCT'('REF''NAMEVAL' NAVAL,
                        'INT' COUNT,
                        'BOOL' NORMAL,
                        'REF''BRANCH' CAUSES,
                        'REF''ALARM' ENTRY,
                        'REF''NODE' STRAIGHT);
'MODE''ALARM' = 'STRUCT'('REF''ALARM' NEXT,
                         'UNION'('REF''NODE','REF''CHECK')TYPE,
                         'REAL' TIME,
                         'BOOL' ACTIVE,SHARED,
                         'STRING' MESSAGE);
'MODE''FAULT' = 'STRUCT'('REF''ALARM' CAUSE,
                         'REF''FAULT' NEXT);
'MODE''BRANCH' = 'STRUCT'('REF''NODE' NEXNODE,
                          'REAL' ADD,SUB,
                          'REF''BRANCH' NEXBRAN);
'C'***************************************************************************'C'
'BOOL' NOSTANDARDS←'TRUE',DEBUG←'TRUE',MOREUNITS←'TRUE',RESET←'TRUE';
'INT' NO ← 10,MAXLEVEL ← 2,HOWMANY ← 2,TARGET ← 10;
'INT' CHECKNO ← 1,LOOKATNO ← 100,NODENO ← 1,PRINTNO ← 100;
'REAL' SMALLSTEP ← 1.0&-8,TOLERANCE ← 1.0&-6;
'BOOL' FUNCMODELS ← 'FALSE';
'PRIORITY' 'UP' = 1,£ = 5,@ = 5;
'C'---------------------------------------------------------------------------'C'
'OP' @ = ('REF''REAL'A,'REAL'B)'REAL';
'BEGIN'
```

```
          'BOOL' R;
          'IF' A > B
          'THEN' A ← B
          'FI';
          A
'END';
'C'-------------------------------------------------------------------------'C'
'OP' £ = ('PEF''REAL'A,'REAL'B)'REAL';
'BEGIN'
          'BOOL' R;
          'IF' A < B
          'THEN' A ← B
          'FI';
          A
'END';
'C'-------------------------------------------------------------------------'C'
'OP' = = ('REAL' A,B)'BOOL';
'BEGIN'
          'BOOL' R ← 'FALSE';
          'IF' 'ABS'(A-B) < 'ABS'(B) * TOLERANCE 'OR' 'ABS'(A-B) < TOLERANCE
          'THEN' R ← 'TRUE'
          'FI';
          R
'END';
'C'-------------------------------------------------------------------------'C'
'OP' # = ('REAL' A,B)'BOOL';
'BEGIN'
          'BOOL' T ← 'FALSE';
          'IF' 'ABS'(A-B) > 'ABS'(0.025*(A+B))
          'THEN' T ← 'TRUE'
          'FI';
          T
'END';
```

LISTING 3.8 (contd.)

LISTING 3.8 (contd.)

```
'C'----------------------------------------------------------------------------'C'
'OP' 'UP' = ('REF''REAL' A,'REAL' B):
'BEGIN'
        'REAL' RESULT;
        'IF' A <= 0.0
        'THEN' A ← 0.0
        'ELSE' A ← EXP(B*LN(A))
        'FI'
'END';
'C'----------------------------------------------------------------------------'C'
[1]'INT' PTAB = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,1,0,0,4,0,0,0,5,2,6,0,0,0,0,0);
[1]'INT' OPTAB = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,6,0,0,0,0,0,3,1,0,2,0,4
                ,7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,5,0);
'REF''TEMP' USERLIST ← 'TEMP' ← ('NIL','NIL',0);
'FOR' I 'TO' 30 'DO'
(   USERLIST ← 'TEMP' ← (USERLIST,'NIL',0);
    LAST'OF'NEXT'OF'USERLIST ← USERLIST  )
'END'
'KEEP''NAMEVAL','CHECKLIST','EFFECT','CHECK','POINTER','CONSTANTS','OPSTACK',
      'FUNCTION','EQUATION','EXPRESSION','CONLIST','FAULTVAL','UNIT',
      'STANDARDS','TEMP','FAULT','ALARM','SYSLIST','ALG','SAVE','NODE',
      'BRANCH',
       NOSTANDARDS,TARGET,FUNCMODELS,HOWMANY,MAXLEVEL,NO,
                 LOOKATNO,SMALLSTEP,MOREUNITS,DEBUG,NODENO,TOLERANCE,PRINTNO,
      PTAB,OPTAB,USERLIST,RESET,CHECKNO,
      'UP',@,£,=,*
'FINISH'
```

LISTING 3.9

Segment SEG1

```
'BEGIN'
'PROC' LIN = ('REAL' X)'REAL';
(     'IF' X <=0 'THEN' 0 'ELSE' X > 1 'THEN' 1 'ELSE' 0.5*(1-COS(PI*X))'FI' );
'C'----------------------------------------------------------------------------'C'
'PROC' RAN = ('REAL' X)'REAL';
'BEGIN'  RANDOM  'END';
'C'----------------------------------------------------------------------------'C'
'PROC' HEV = ('REAL' X)'REAL';
(      'IF' X <=0 'THEN' 0 'ELSE' 1 'FI'  );
'C'----------------------------------------------------------------------------'C'
'PROC' TWONLY = ('REF''NODE' N);
'BEGIN'
        'BYTES' B ← NAME'OF'NAVAL'OF'N;
        'CASE' (COUNT'OF'N)'/'1000
        'IN' PRINT(B),
             PRINT((1'ELEM'B,2'ELEM'B,"  ")),
             PRINT((3'ELEM'B,4'ELEM'B,"  "))
        'OUT' PRINT(B)
        'ESAC'
'END';
'C'----------------------------------------------------------------------------'C'
'PROC' CHOPSTRING = ('REF'[]'CHAR' M);
'BEGIN'
        'INT' I ← 'UPB'M;
        'WHILE' I > 1 'DO'
        (    'IF' M[I] # " "
             'THEN' 'GOTO' CHOP
             'ELSE' I'MINUS'1
             'FI' );
        CHOP:M ← M[1:I+1]
'END';
'C'----------------------------------------------------------------------------'C'
```

```
'PROC' LOOKAT = ('REF''CHARPUT' V);
'BEGIN'
     'INT' I ← CHARNUMBER(V);
     'IF' I >= LOOKATNO
     'THEN' PUTC(V,NEWLINE)
     'FI'
'END';
'C'------------------------------------------------------------------------'C'
'PROC' ADDZEROS = ('REF'[ ]'CHAR' SOURCE);
'BEGIN'
     'INT' L ← 1,K ← 1;
     [1:'UPB' SOURCE + 10]'CHAR' NEW;
     'CLEAR' NEW;
     'WHILE' SOURCE[L] # " " 'DO'
     (    L1:NEW[K] ← SOURCE[L];
             L'PLUS'1;
             K'PLUS'1;
          L2:'IF' SOURCE[L] = "(" 'OR' SOURCE[L] = "+"
             'THEN' NEW[K] ← SOURCE[L];
                    L'PLUS'1;
                    K'PLUS'1;
                    'IF' SOURCE[L] = "-"
                    'THEN' NEW[K] ← "Z";
                           NEW[K+1] ← "0";
                           K'PLUS'2;
                           'GOTO' L1
                    'ELSE' 'GOTO' L2
                    'FI'
             'FI');
    SOURCE ← NEW[1:K]
'END';
'C'------------------------------------------------------------------------'C'
'PROC' POLISH = ('REF'[]'CHAR' SOURCE,[]'INT' T);
```

LISTING 3.9 (contd.)

```
'BEGIN'
    'INT' I,J,N,K ← 'UPB'SOURCE;
    'REF''OPSTACK' P,S ← 'NIL';
    'CHAR' C;
    'IF' SOURCE[2] = "(" 'AND' SOURCE[5] = ")"
    'THEN' I ← J ← 10
    'ELSE' I ← J ← 4
    'FI';
    'WHILE' I < K 'DO'
    (   C ← SOURCE[I];
        P ← S;
        'CASE' T['ABS'C] + 1
        'IN' 'WHILE' S 'ISNT' ('REF''OPSTACK''VAL''NIL') 'DO'
             (    S ← NEXT'OF'S;
                  'IF' CHAR'OF'P = "("
                  'THEN' 'GOTO' LEND
                  'ELSE' SOURCE[J] ← CHAR'OF'P;P ← S;J'PLUS' 1
                  'FI'  ),
             S ← 'LOC''OPSTACK' ← (C,S);'GOTO' LEND,
             N ← 2,
             N ← 3,
             N ← 4,
             N ← 5,
             SOURCE[J] ← C; J'PLUS'1;'GOTO' LEND
        'OUT' PRINT((NEWLINE,"ERROR IN PRIORITY TABLE",NEWLINE))
        'ESAC';
        'WHILE' S 'ISNT' ('REF''OPSTACK''VAL''NIL') 'DO'
        (    'IF' T['ABS'(CHAR'OF'S)] < N
             'THEN' S ← 'LOC''OPSTACK' ← (C,S);'GOTO' LEND
             'ELSE' SOURCE[J]←CHAR'OF'S;S←NEXT'OF'S;J'PLUS'1
             'FI'  );
        S ← 'LOC''OPSTACK' ← (C,S);
        LEND:I'PLUS'1  );
```

LISTING 3.9 (contd.)

LISTING 3.9 (contd.)

```
               'WHILE' S 'ISNT' ('REF''OPSTACK''VAL''NIL') 'DO'
               (     SOURCE[J] ← CHAR'OF'S;
                     J 'PLUS' 1;
                     S ← NEXT'OF'S   );
               SOURCE[J] ← " ";
               SOURCE ← SOURCE[1:J];
               'IF' DEBUG
               'THEN' PRINT(SOURCE)
               'FI'
'END';
'C'-------------------------------------------------------------------------'C'
'PROC' PREPSOURCE = ('INT'LONGEST,'REF'[]'CHAR'SOURCE,[]'INT' T);
'BEGIN'
               'INT' L ← 1;
               [1:LONGEST]'CHAR' TEMP;
               'CLEAR' TEMP;
               'CHAR' CAR ← "$";
               'WHILE' CAR * " " 'DO'
               (     READ(CAR);
                     TEMP[L] ← CAR;
                     L'PLUS'1;
                     'IF' L > LONGEST
                     'THEN' PRINT((NEWLINE,"***** ERROR IN PREPSOURCE *****",NEWLINE))
                     'FI'   );
               SOURCE ← TEMP[1:L];
               ADDZEROS(SOURCE);
               PRINT((NEWLINE,SOURCE));
               'IF' LONGEST >= 60
               'THEN' PRINT(NEWLINE)
               'ELSE' SETCHARNUMBER(STANDOUT,65)
               'FI';
               POLISH(SOURCE,T)
'END';
```

```
'C'-----------------------------------------------------------------------'C'
'PROC' FINDCON = ('BYTES' BITE,'REF''REF''NAMEVAL'N,'REF''CONSTANTS'CON)'BOOL':
'BEGIN'
        'BOOL' OKAY ← 'FALSE';
        'REF''CONSTANTS' C ← CON;
        'WHILE' C 'ISNT' ('REF''CONSTANTS''VAL''NIL') 'DO'
        (    'IF' NAME'OF'NAVAL'OF'C = BITE
             'THEN' OKAY ← 'TRUE';
                    N ← NAVAL'OF'C;
                    C ← 'NIL'
             'ELSE' C ← NEXCON'OF'C
             'FI' );
        OKAY
'END';
'C'-----------------------------------------------------------------------'C'
'PROC' SEARCHFORVAR = ('BYTES' BITE,'REF''REF''NAMEVAL' N,
                        'REF''UNIT' UNIT,'BOOL' NOSTAND)'BOOL':
'BEGIN'
        'BOOL' OKAY ← 'FALSE',IN ← 'TRUE';
        'REF''CONLIST' L;
        'INT' I;
        'REF''POINTER' P ← INLIST 'OF' UNIT;
        'CHAR' ONE;
        'BYTES' B;
        'REF''NAMEVAL' NV;
        [1:4]'CHAR' CC;
        PLOOP:'WHILE' P 'ISNT' ('REF''POINTER''VAL''NIL') 'DO'
              (    'IF' 2'ELEM'BITE ≠ NAME'OF'P
                   'THEN' P ← NEXP'OF'P
                   'ELSE' ONE ← 1'ELEM' BITE;
                          I ← PTAB['ABS'ONE + 1];
                          'IF' I = 0
                          'THEN' 'GOTO' LCONS
```

LISTING 3.9 (contd.)

```
       'ELSE' NV ← (VARI'OF'P)[I];
              B ← NAME'OF'NV;
              CC ← 'IF' IN
                   'THEN' (1'ELEM'B,2'ELEM'B," "," ")
                   'ELSE' (3'ELEM'B,4'ELEM'B," "," ")
                   'FI';
              'IF' 'CTB'CC = BITE
              'THEN' OKAY ← 'TRUE';
                    N ← NV;
                    'IF' 'NOT'NOSTAND
                    'THEF' STAND'OF'NV'IS'
                          ('REF''STANDARDS''VAL''NIL')
                    'THEN' STAND'OF'NV ← 'STANDARDS' ←
                   (0,0,0,0,0,0,UNIT,'NIL','NIL',"SKIP")
                    'FI';
                    'GOTO' LEND
              'ELSE' 'GOTO' LCONS
              'FI'
       'FI'
   'FI'    );
'IF' IN
'THEN' P ← OUTLIST'OF'UNIT;
    IN ← 'FALSE';
    'GOTO' PLOOP
'FI';
LCONS:L ← COMMONS'OF'UNIT;
   'WHILE' L 'ISNT' ('REF''CONLIST''VAL''NIL') 'DO'
   (   'IF' FINDCON(BITE,N,CON'OF'L)
       'THEN' OKAY ← 'TRUE';'GOTO' LEND
       'ELSE' L ← NEXLIST'OF'L
       'FI' );
   'IF' 'NOT'FINDCON(BITE,N,CON'OF'UNIT)
   'THEN' CON'OF'UNIT ← 'CONSTANTS' ←
```

LISTING 3.9 (contd.)

```
                                          (CON'OF'UNIT,(BITE,0,'NIL'));
                           N ← NAVAL'OF'CON'OF'UNIT
                    'ELSE' OKAY ← 'TRUE'
                    'FI';
        LEND;OKAY
'END';
'C'-----------------------------------------------------------------------------'C'
'PROC' RHS = ('REF''INT' L,'REF''UNIT' UNIT,[]'CHAR' CAR,
                                            'BOOL' NOSTAND)'REF''EQUATION':
'BEGIN'
        'REF''NAMEVAL' PAL;
        []'CHAR' MM = "***** ERROR IN FUNCTION NAME *****";
        'REF''EQUATION' E ← 'NIL';
        'INT' I ← 'UPB'CAR - 1;
        [1:4]'CHAR' CC;
        'BYTES' BITE;
        'BOOL' B;
        'CHAR' D;
        'WHILE' I > L 'DO'
        (    D ← CAR[I];
             I 'MINUS' 1;
             'IF' OPTAB['ABS'D + 1] # 0
             'THEN' E ← 'EQUATION' ← (E,D)
             'ELSE' 'IF' D = "]"
                    'THEN' CC ← (CAR[I-1],CAR[I]," "," ");
                           BITE ← 'CTB'CC;
                           SEARCHFORVAR(BITE,PAL,UNIT,NOSTAND);
                           I'MINUS'7;
                           CC ← CAR[I+1:I+4];
                           BITE ← 'CTR'CC;
                           E ← 'EQUATION' ← (E,'FUNCTION' ← ('IF'   BITE="$SQR"
                                                             'THEN' 1
                                                             'ELSF' BITE="$EXP"
```

LISTING 3.9 (contd.)

```
                                                              'THEN' 2
                                                              'FLSF' BITF="$LIN"
                                                              'THEN' 3
                                                              'ELSF' BITF="SSIN"
                                                              'THEN' 4
                                                              'FLSF' BITE="SCOS"
                                                              'THEN' 5
                                                              'ELSF' BITE="$TAN"
                                                              'THEN' 6
                                                              'FLSF' BITF="$ASN"
                                                              'THEN' 7
                                                              'ELSF' BITF="$ACS"
                                                              'THEN' 8
                                                              'FLSF' BITE="$ATN"
                                                              'THEN' 9'
                                                              'ELSF' BITE="SLIM"
                                                              'THEN' 10
                                                              'FLSF' BITE="$RAN"
                                                              'THEN' 11
                                                              'ELSF' BITF="$HEV"
                                                              'THEN' 12
                                                              'FLSE' PRINT(MM);0
                                                              'FI',PAL))
                    'ELSE' CC ← (CAP[I];D," "," ");
                           I 'MINUS' 1;
                           BITE ← 'CTB'CC;
                           SEARCHFOPVAR(BITE,PAL,UNIT,NOSTAND);
                           E ← 'EQUATION' ← (E,PAL)
                    'FI'
             'FI'   );
             E
'END';
'C'----------------------------------------------------------------------------'C'
```

LISTING 3.9 (contd.)

LISTING 3.9 (contd.)

```
'PROC' ONEQUATION = ('REF''SYSLIST' SAME,[]'CHAR' CAR,'BOOL' NORMAL):
'BEGIN'
        'INT' L;
        'BOOL' B;
        'REF''UNIT' UNIT ← SYSU'OF'SAME;
        'REF''EXPRESSION' EX ← 'EXPRESSION' ← (RELATIONS'OF'UNIT,
                                                'NIL','NIL','NIL','NIL');
        RELATIONS'OF'UNIT ← EX;
        [1:4]'CHAR' CC;
        'BYTES' BITE;
        'IF' CAR[2] = "(" 'AND' CAR[5] = ")"
        'THEN' CC ← (CAR[3],CAR[4]," "," ");
               GRADIENT'OF'EX ← 'REAL' ← 0;
               L ← 9
        'ELSE' CC ← (CAR[1],CAR[2]," "," ");
               L ← 3
        'FI';
        BITE ← 'CTB' CC;
        B ← SEARCHFORVAR(BITE,LHS'OF'EX,UNIT,'FALSE');
        'IF' NORMAL
        'THEN' NORMAL'OF'EX
        'ELSE' ALTER'OF'EX
        'FI' ← RHS(L,UNIT,CAR,'FALSE');
        'IF' NEXS'OF'SAME 'ISNT' ('REF''SYSLIST''VAL''NIL')
        'THEN' ONEQUATION(NEXS'OF'SAME,CAR,NORMAL)
        'FI'
'END';
'C'---------------------------------------------------------------------'C'
'PROC'FINDUNIT=('REF''UNIT'UNITLIST,'REF''BYTES'UNITNAME)'REF''UNIT':
'BEGIN'
        'REF''UNIT' U ← UNITLIST;
        'WHILE' U 'ISNT' ('REF''UNIT''VAL''NIL') 'DO'
        (     'IF' UNAME'OF'U = UNITNAME
             'THEN' 'GOTO' LEND
             'ELSE' U ← NEXU'OF'U
             'FI'     );
        PRINT((NEWLINE,"*** FINDUNIT FAILS *** ",UNITNAME,NEWLINE));
        LEND:U
'END';
'C'-----------------------------------------------------------------------------'C'
'PROC' ADDCOMMONS = ('REF''SYSLIST' SAME,'REF''CONSTANTS' C);
'BEGIN'
        'REF''UNIT' U ← SYSU'OF'SAME;
        COMMONS'OF'U ← 'CONLIST' ← (C,COMMONS'OF'U);
        'IF' NEXS'OF'SAME'ISNT' ('REF''SYSLIST''VAL''NIL')
        'THEN' ADDCOMMONS(NEXS'OF'SAME,C)
        'FI'
'END';
'C'-----------------------------------------------------------------------------'C'
'PROC' READMODEL = ('REF''UNIT'UNITLIST,[]'INT' TABLE);
'BEGIN'
        'CHAR' MORE,FIRST;
        [1:12]'CHAR' C;
        'REF''CONSTANTS' V ← 'NIL';
        'REF''SYSLIST' SAME ← 'NIL';
        'BOOL' NORMAL ← 'TRUE';
        'BYTES' UNITNAME,VARNAME;
        'INT' LONGEST ← 59;
        PRINT(NEWPAGE);
        PRINT((NEWLINE,NEWLINE,"READMODEL PRINTOUT",NEWLINE,
                               "******************",NEWLINE));
        READ((NEWLINE,FIRST));
        'IF' FIRST ≠ "S"
        'THEN' PRINT((NEWLINE,"***** ERROR IN READMODEL *****"));
               'GOTO' LEND
        'FI';
```

LISTING 3.9 (contd.)

LISTING 3.9 (contd.)

```
LOOP:RFAD(C);
     'IF'   C = "EQUATIONS    "
     'THEN' READ(NEWLINE);
            N:'BEGIN'
                      'REF'[]'CHAR' CAR = [1:0'FLEX']'CHAR';
                      PREPSOURCE(LONGEST,CAR,TABLE);
                      ONEQUATION(SAME,CAR,NORMAL)
              'END';
              READ((NEWLINE,FIRST));
              'IF' FIRST = "$"
              'THEN' 'GOTO' LOOP
              'ELSE' READ(BACKSPACE);
                     'GOTO' N
              'FI'
     'ELSF' C = "UNITNAMES    "
     'THEN' READ((NFWLINE,UNITNAME,MORE));
     PRINT((NEWLINE,NEWLINE));
            SAME ← 'NIL';
            L:SAME ← 'SYSLIST' ← (FINDUNIT(UNITLIST,UNITNAME),
                                                   'NIL','FALSE',SAME);
     PRINT((NEWLINE,"      ",UNITNAME,NEWLINE));
              'IF' MORE = ","
              'THEN' READ((UNITNAME,MORF));
                     'GOTO' L
              'FI'
     'ELSF' C = "COMMONVARS   "
     'THEN' PRINT((NFWLINE,"VARIABLES COMMON TO ABOVE UNITS:",
                             NEWLINE));
            READ((NEWLINE,VARNAME,MORE));
            PRINT((NEWLINE,VARNAME,MORE));
            M:V ←'CONSTANTS' ← (V,(VARNAME,0,'NIL'));
              'IF' MORE = ","
              'THEN' READ((VARNAME,MORE));
```

```
                                   PRINT((VARNAME,MORE));
                                   'GOTO' M
                              'FI';
                              ADDCOMMONS(SAME,V);
                              PRINT((NEWLINE,NEWLINE));
                              V ← 'NIL'
                    'ELSF' C = "LONGEST     "
                    'THEN' READ(LONGEST)
                    'ELSF' C = "ENDREADMODEL"
                    'THEN' 'GOTO' LEND
                    'ELSE' PRINT((NEWLINE,"***** ERR IN READMODEL *****"));
                           'GOTO' LEND
                    'FI';
                    READ((NEWLINE,FIRST));
                    'IF' FIRST = "$"
                    'THEN' 'GOTO' LOOP
                    'ELSE' PRINT((NEWLINE,"***** READMODEL ERROR *****"))
                    'FI';
               LEND:PRINT(NEWPAGE)
'END';
'C'-----------------------------------------------------------------------'C'
'PROC' ADDPOINTVAR = ('CHAR' I,O)'REF'[]'REF''NAMEVAL':
'BEGIN'
          'REF'[]'REF''NAMEVAL' N ← [1:6]'REF''NAMEVAL';
          'CHAR' V;
          [1:4]'CHAR' CC;
          'FOR' J 'TO' 6 'DO'
          (    V ← 'CASE' J 'IN' "Q","Y","P","T","X","Z"  'ESAC';
               CC ← (V,I,V,O);
               N[J] ← 'NAMEVAL' ← ('CTB'CC,0,'NIL')    ));
          N
'END';
'C'-----------------------------------------------------------------------'C'
```

LISTING 3.9 (contd.)

LISTING 3.9 (contd.)

```
'PROC' ICLIST = 'REF' 'POINTER';
'BEGIN'
        'BYTES' LIST:
        'CHAR' MORE ← ",",CAR,NAME;
        'REF' 'POINTER' NEXP ← 'NIL';
        'WHILE' MORE = "," 'DO'
        ( .   READ((NEWLINE,NAME,SPACE,LIST,SPACE,CAR,SPACE,MORE));
              SETCHARNUMBER(STANDOUT,30);
              PRINT((NAME,"          ",LIST,"     ",CAR,NEWLINE));
              NEXP ← 'POINTER' ← (NAME,LIST,CAR,NEXP,'NIL','NIL','NIL')  );
        NEXP
'END';
'C'-------------------------------------------------------------------'C'
'PROC' LISTUNIT = 'REF''UNIT':
'BEGIN'
        'BOOL' OK ← 'FALSE';
        'REF''UNIT' U ← 'NIL';
        'REF''POINTER' IN ← 'NIL';
        'REF''POINTER' OUT ← 'NIL';
        'CHAR' CAR;
        'BYTES' UNITNAME,B ← "GT  ";
        'REF''CONSTANTS' C ← 'CONSTANTS' ← ('NIL',(B,0,'NIL')):
        C ← 'CONSTANTS' ← (C,("WN  ",1,'NIL'));
        C ← 'CONSTANTS' ← (C,("HH  ",0,'NIL')):
        C ← 'CONSTANTS' ← (C,("EX  ",2.71828,'NIL'));
        C ← 'CONSTANTS' ← (C,("KH  ",0.5,'NIL'));
        C ← 'CONSTANTS' ← (C,("ZO  ",0,'NIL'));
        B ← "LT  ";
        PRINT(NEWPAGE):
        SETCHARNUMBER(STANDOUT,13);
        PRINT(("LISTUNIT OUTPUT",NEWLINE,NEWLINE,
              "    UNIT     INPUT OR     POINT ON   LINKED WITH",NEWLINE,
              "    NAME      OUTPUT        UNIT     UNIT  POINT",NEWLINE));
```

LISTING 3.9 (contd.)

```
        LOOP:READ((NEWLINE,CAR));
             'IF'   CAR = "I"
             'THEN' SETCHARNUMBER(STANDOUT,15);PRINT("INPUT");
                    IN ← IOLIST
             'ELSF' CAR = "O"
             'THEN' SETCHARNUMBER(STANDOUT,15);PRINT("OUTPUT");
                    OUT ← IOLIST
             'ELSF' CAR = "U"
             'THEN' 'IF' OK
                    'THEN' U ←'UNIT' ← (UNITNAME,IN,OUT,U,'NIL','NIL',
                                     'CONSTANTS' ← ('NIL',(B,0,'NIL')),
                                     'CONLIST' ← (C,'NIL'));
                           OUT ← IN ← 'NIL'
                    'ELSE' OK ← 'TRUE'
                    'FI';
                    READ((NEWLINE,UNITNAME));
                    PRINT((NEWLINE,"    ",UNITNAME,"       "))
             'ELSF' CAR = "F"
             'THEN' U ←'UNIT' ← (UNITNAME,IN,OUT,U,'NIL','NIL',
                               'CONSTANTS' ← ('NIL',(B,0,'NIL')),
                               'CONLIST' ← (C,'NIL'));
                    PRINT((NEWLINE,NEWLINE));
                    'GOTO' LEND
             'ELSF' PRINT((NEWLINE,"***** ERROR IN LISTUNIT *****",CAR));
                    'GOTO' LEND
             'FI';
             'GOTO' LOOP;
        LEND:U
'END';
'C'---------------------------------------------------------------------'C'
'PROC' LINK'UNITS = ('REF''UNIT' ULIST):
'BEGIN'
        'REF''UNIT' UI ← ULIST,UO;
        'REF''POINTER' PI,PO;
        'WHILE' UI 'ISNT' ('REF''UNIT''VAL''NIL') 'DO'
        (    PI ← INLIST'OF'UI;
             'WHILE' PI 'ISNT' ('REF''POINTER''VAL''NIL') 'DO'
             (    UO ← FINDUNIT(ULIST,LIST'OF'PI);
                  PO ← OUTLIST'OF'UO;
                  'WHILE' PO 'ISNT' ('REF''POINTER''VAL''NIL') 'DO'
                  (    'IF'(NAME'OF'PI = CAR'OF'PO 'AND'
                           CAR'OF'PI = NAME'OF'PO 'AND'
                           LIST'OF'PO=UNAME'OF'UI )
                       'THEN' LINK'OF'PO ← PI;
                              LINK'OF'PI ← PO;
                              VARI'OF'PI ← VARI'OF'PO +
                                    ADDPOINTVAR(NAME'OF'PI,NAME'OF'PO);
                              UNIT'OF'PI ← UI;
                              UNIT'OF'PO ← UO;
                              'GOTO' LABEL
                       'ELSE' PO ← NEXP'OF'PO
                       'FI' );
                  PRINT((NEWLINE,"ERROR IN MODEL DATA  ",NAME'OF'PI,SPACE,
                         LIST'OF'PI,SPACE,CAR'OF'PI," OF UNIT ",UNAME'OF'UI,
                         "  DOES NOT HAVE COUNTERPART IN DATA SUPPLIED",
                                                                NEWLINE));
                  LABEL:PI ← NEXP'OF'PI );
             UI ← NEXU'OF'UI )
'END';
'C'--------------------------------------------------------------------------'C'
'PROC' FINDPVAR = ('REF''UNIT' UNITLIST):
'BEGIN'
        'REF''UNIT' U ← UNITLIST;
        'REF''POINTER' P;
        'REF'[]'REF''NAMEVAL' V;
        'WHILE' U 'ISNT' ('REF''UNIT''VAL''NIL') 'DO'
```

LISTING 3.9 (contd.)

LISTING 3.9 (contd.)

```
        (     P ← INLIST'OF'U;
              'WHILE' P 'ISNT' ('REF''POINTER''VAL''NIL') 'DO'
              (      V ← VARI'OF'P;
                     'FOR' J 'TO' 6 'DO'
                     'IF' STAND'OF'V[J] 'IS' ('REF''STANDARDS''VAL''NIL')
                     'THEN' V[J] ← 'NIL'
                     'ELSE' STAND'OF'V[J] ← 'NIL'
                     'FI';
                     P ← NEXP'OF'P     );
           U ← NEXU'OF'U     )
'END';
'C'-----------------------------------------------------------------------------'C'
'PROC' UNITDATA = ('REF''UNIT'UNIT);
'BEGIN'
         'CHAR' MORE ← ",";
         'BYTES'VARNAME;
         'REAL'VARVAL;
         'REF''NAMEVAL' PAL;
         'BOOL' OKAY,FOUND;
         PRINT((NEWLINE,NEWLINE,NEWLINE,"UNIT ",UNAME 'OF' UNIT,
                NEWLINE,"*********",NEWLINE,NEWLINE));
         'WHILE' MORE = "," 'DO'
         (      READ((NEWLINE,VARNAME,SPACE,VARVAL,SPACE,MORE));
                PRINT((LOOKAT,SPACE,VARNAME,VARVAL));
                'IF' SEARCHFORVAR(VARNAME,PAL,UNIT,NOSTANDARDS)
                'THEN' PRINT(" ")
                'ELSE' PRINT("*")
                'FI';
                VALUE'OF'PAL ← VARVAL     );
        PRINT((NEWLINE,NEWLINE))
'END';
'C'-----------------------------------------------------------------------------'C'
'PROC' SYSTEMDATA = ('REF''UNIT' UNITLIST);
```

```
'BEGIN'
        'REF''UNIT' U;
        'CHAR' MOREU ← ",";
        'BYTES' UNITNAME;
        PRINT((NEWPAGE,NEWLINE,NEWLINE));
        PRINT(("SYSTEMDATA PRINTOUT",NEWLINE));
        PRINT(("*******************",NEWLINE));
        'WHILE' MOREU = "," 'DO'
        (    READ((NEWLINE,UNITNAME,SPACE,MOREU));
             U ← FINDUNIT(UNITLIST,UNITNAME);
             UNITDATA(U)     );
        PRINT(("*  INDICATES PRECEEDING NAME NOT FOUND IN EQUATIONS",
               NEWLINE,NEWLINE,NEWLINE))
'END';
'C'------------------------------------------------------------------------'C'
'PROC' MASTERMODEL = ([]'INT' TABLE)'REF''UNIT';
'BEGIN'
        'REF''UNIT' U ← LISTUNIT;
        LINKUNITS(U);
        READMODEL(U,TABLE);
        FINDPVAR(U);
        U
'END';
'C'------------------------------------------------------------------------'C'
'PROC' LINKEFFECT = ('REF''CHECKLIST' C,'REF''CHECK' CK);
'BEGIN'
        'REF''CHECKLIST' C1 ← NEXCH'OF'CK,C2;
        'REF''NAMEVAL' N;
        'REF''STANDARDS' S;
        NEXCH'OF'CK ← 'NIL';
        'WHILE' C1 'ISNT' ('REF''CHECKLIST''VAL''NIL') 'DO'
        (    C2 ← NEXCH'OF'C1;
             NEXCH'OF'C1 ← NEXCH'OF'CK;
```

LISTING 3.9 (contd.)

```
             NEXCH'OF'CK ← C1;
             C1 ← C2    );
        C1 ← NEXCH'OF'CK;
        'WHILE' C1 'ISNT' ('REF''CHECKLIST''VAL''NIL') 'DO'
        (    S ← STAND'OF'NAVAL'OF'C1;
             'IF' COUNT'OF'C1 > 0
            'THEN' HIEFLIST'OF'S ← 'EFFECT' ← (HIEFLIST'OF'S,CK)
            'ELSE' LOEFLIST'OF'S ← 'EFFECT' ← (LOEFLIST'OF'S,CK)
            'FI';
             C1 ← NEXCH'OF'C1   )
'END';
'C'------------------------------------------------------------------------'C'
'PROC' FUNC = ('INT' I)'PROC'('REAL')'REAL';
(  'CASE' I
  'IN' SQRT,EXP,LN,SIN,COS,TAN,ARCSIN,ARCCOS,ARCTAN,LIM,RAN,HEV
  'ESAC'  );
'C'------------------------------------------------------------------------'C'
'PROC' RUNEQUATION = ('REF''EQUATION' EQ)'REAL';
'BEGIN'
       'REF''TEMP' T ← USERLIST;
       'REF''EQUATION' E ← EQ;
       'REF''FUNCTION' F;
       'CHAR' C;
       'REF''NAMEVAL' N;
       'WHILE' F 'ISNT' ('REF''EQUATION''VAL''NIL') 'DO'
       ( 'CASE' (N,F,C) ::= CONTENTS'OF'E
         'IN' VALUE'OF'T ← VALUE'OF'N;T ← NEXT'OF'T,
              VALUE'OF'T ← FUNC(NO'OF'F)(VALUE'OF'ARG'OF'F);T ← NEXT'OF'T,
              T ← LAST'OF'T;
              'BEGIN'
                    'REF''REAL' R = VALUE'OF'LAST'OF'T;
                    'REF''REAL'S = VALUE'OF'T;
                    'CASE' OPTAB['ABS'C + 1]
```

LISTING 3.9 (contd.)

```
                      'IN'  R 'PLUS' S,
                            R 'MINUS'S,
                            R 'TIMES'S,
                            R  'DIV' S,
                            R  'UP'  S,
                            R   £    S,
                            R   @    S
                       'OUT' PRINT((NEWLINE,"*****ERROR IN RUNEQUATION*****"))
                     'ESAC'
               'END'
          'ESAC';
          E ← NEXT'OF'E );
        VALUE'OF'LAST'OF'T
'END';
'SKIP'
'END'
'KEEP' LIN.LOOKAT,PREPSOURCE,SEARCHFORVAR,RHS,ONEQUATION,FINDUNIT,
       TWONLY,CHOPSTRING,
       READMODEL,SYSTEMDATA,MASTERMODEL.LINKEFFECT,RUNEQUATION
'FINISH'
```

LISTING 3.9 (contd.)

```
'BEGIN'
'PROC' RUNALG = ('REF''ALG' ALGEBRAEICS,'REF''BOOL' CHANGED);
'BEGIN'
        'REF''ALG' A + ALGEBRAEICS;
        'REAL' STORE;
        'WHILE' A 'ISNT' ('REF''ALG''VAL''NIL') 'DO'
        (     STORE + VALUE'OF'LHS'OF'A;
             VALUE'OF'LHS'OF'A + RUNEQUATION(RHS'OF'A);
             'IF''NOT'(STORE=VALUE'OF'LHS'OF'A)
             'THEN' CHANGED + 'TRUE';
                    'IF' DEBUG
                    'THEN' PRINT((NEWLINE,NAME'OF'LHS'OF'A,SPACE,
                                        VALUE'OF'LHS'OF'A,SPACE,STORE))
                    'FI'
             'FI';
             A + NEXALG'OF'A    )
'END';
'C'-------------------------------------------------------------------------------'C'
'PROC' STEPUP = ('REF''SAVE' SET,'REF''REAL' H,'REF''INT' COUNTER);
'BEGIN'
        'REF''SAVE' S + SET;
        H'TIMES'1.5;
        [1:6]'REAL' A;
        COUNTER + 8;
        'WHILE' S 'ISNT' ('REF''SAVE''VAL''NIL') 'DO'
        ( A + (YS'OF'S,F'OF'S,K5'OF'S,K4'OF'S,K3'OF'S,K2'OF'S);
          K3'OF'S + K2'OF'S;
          K4'OF'S + EQUIPOL(A,3.75);
          K5'OF'S + EQUIPOL(A,2.50);
          F 'OF'S + EQUIPOL(A,1.25);
          S + NEXSAVE'OF'S )
'END';
```

LISTING 3.10 (contd.)

```
'C'--------------------------------------------------------------------------'C'
'PROC' STEPDOWN = ('REF''SAVE' SET,'REF''REAL' H);
'BEGIN'
        'REF''SAVE' S ← SET;
        [1:6]'REAL' A;
        H'TIMES'0.5;
        'WHILE' S 'ISNT' ('REF''SAVE''VAL''NIL') 'DO'
        ( A ← (YS'OF'S,E'OF'S,K5'OF'S,K4'OF'S,K3'OF'S,K2'OF'S);
          YS'OF'S ← K4'OF'S;
          E 'OF'S ← EQUIPOL(A,3.5);
          K5'OF'S ← K3'OF'S;
          K4'OF'S ← EQUIPOL(A,4.5);
          K3'OF'S ← K2'OF'S;
          S ← NEXSAVE'OF'S )
'END';
'C'--------------------------------------------------------------------------'C'
'PROC' UPDATE = ('REF''SAVE' SET);
'BEGIN'
        'REF''SAVE' S ← SET;
        'WHILE' S 'ISNT' ('REF''SAVE''VAL''NIL') 'DO'
        ( YS'OF'S ← E 'OF'S;
          E 'OF'S ← K5'OF'S;
          K5'OF'S ← K4'OF'S;
          K4'OF'S ← K3'OF'S;
          K3'OF'S ← K2'OF'S;
          S ← NEXSAVE'OF'S )
'END';
'C'--------------------------------------------------------------------------'C'
'PROC' ALTER = ('REF''SAVE' SET,'REF''REAL' H,'REF''INT' COUNTER);
'BEGIN'
        [1:5]'REAL' A;
        'REF''SAVE' S ← SET;
        H'TIMES'0.25;
```

LISTING 3.10 (contd.)

```
        COUNTER ← 5;
        PRINT(NEWLINE);
        'WHILE' S 'ISNT' ('REF''SAVE''VAL''NIL') 'DO'
        ( A ← (YS'OF'S,F'OF'S,K5'OF'S,K4'OF'S,K3'OF'S);
          VALUE'OF'VARIABLE'OF'S ← K1'OF'S;
          YS'OF'S ← K4'OF'S;
          F 'OF'S ← EQUIPOL(A,3.25);
          K5'OF'S ← EQUIPOL(A,3.5);
          K4'OF'S ← EQUIPOL(A,3.75);
          S ← NEXSAVE'OF'S    )
'END';
'C'-----------------------------------------------------------------------------'C'
'PROC' STEPSET = ('REF''SAVE'SET,'REF''REAL'H,'INT'ITERC,ITERA,
                                                  'REF''INT'COUNTER):

'BEGIN'
        'IF' COUNTER < 10
        'THEN' UPDATE(SET);COUNTER'PLUS'1
        'ELSE' 'CASE' ITERC
               'IN' STEPUP(SET,H,COUNTER),
                    STEPUP(SET,H,COUNTER),
                    UPDATE(SET),
                    UPDATE(SET),
                    UPDATE(SET)
               'OUT' STEPDOWN(SET,H)
               'ESAC'
        'FI'
'END';
'C'-----------------------------------------------------------------------------'C'
'PROC' PRINTVALS = ('REF''ALG' ALGEBRAEICS,'REF''SAVE' SET,'REAL' T):
'BEGIN'
        'REF''ALG' A ← ALGEBRAEICS;
        'REF''SAVE' S ← SET;
        'REF''NAMEVAL' N;
```

LISTING 3.10 (contd.)

```
          AFTER'OF'NUMBERSTYLE ← 5;
          PRINT((NEWLINE,NEWLINE,"TIME = ",T,NEWLINE,NEWLINE,
                 "VALUES SET BY DIFFERENTIAL EQUATIONS:",NEWLINE,NEWLINE));
          'WHILE' S 'ISNT' ('REF''SAVE''VAL''NIL') 'DO'
          ( N ← VARIABLE'OF'S;
            PRINT((LOOKAT,NAME'OF'N,VALUE'OF'N,SPACE));
            S ← NEXSAVE'OF'S );
          PRINT((NEWLINE,NEWLINE,"VALUES SET BY ALGEBRAIC EQUATIONS:",
                                                            NEWLINE,NEWLINE));
          'WHILE' A 'ISNT' ('REF''ALG''VAL''NIL') 'DO'
          ( N ← LHS'OF'A;
            PRINT((LOOKAT,NAME'OF'N,VALUE'OF'N,SPACE));
            A ← NEXALG'OF'A );
          AFTER'OF'NUMBERSTYLE←10
'END';
'C'-----------------------------------------------------------------------'C'
'PROC' MASTERINTEG = ('REF''SAVE' SET,'REF''ALG' ALGEBRAEICS,
                      'REF''REAL' TMAX,H,T,'REF''INT' METHODNO,PRINTNO,COLS):
'BEGIN'
          'REF''SAVE' S ← SET;
          'REF''ALG' A ← ALGEBRAEICS;
          'BOOL' CHANGED;
          'INT' COUNTER,ITERA,ITERC,PRINTCOUNT ← 0;
          'REAL' STORE,MAXSTEP ← 0;
          'IF' METHODNO = 1
          'THEN' PRINTVALS(A,S,T);
                 COUNTER ← 0;
                 METHODNO ← 2
          'ELSE' COUNTER ← 10
          'FI';
          PRINT(NEWLINE);
          'WHILE' T < TMAX 'DO'
          ( PRINTCOUNT 'PLUS' 1;
START:S ← SET;
      'WHILE' S 'ISNT' ('REF''SAVE''VAL''NIL') 'DO'
      ( K1'OF'S ← VALUE'OF'VARIABLF'OF'S+0.08333*H*(23.0*K3'OF'S-
                                                     16.0*K4'OF'S+
                                                      5.0*K5'OF'S);

        S ← NEXSAVE'OF'S );
      S ← SET;
      'WHILE' S 'ISNT' ('REF''SAVE''VAL''NIL') 'DO'
      ( STORE ← VALUE'OF'VARIABLE'OF'S;
        VALUE'OF'VARIABLE'OF'S ← K1'OF'S;
        K1'OF'S ← STORE;
        S ← NEXSAVE'OF'S  );
      CHANGED ← 'TRUE';
      ITERA ← 0;
      'WHILE' CHANGED 'DO'
      ( CHANGED ← 'FALSE';
        ITERA'PLUS'1;
        'IF' ITERA > 7 'AND' DEBUG
        'THEN' PRINT((NEWLINE,"ITERA = ",ITERA));
               ITERA ← 0;
               PRINTVALS(ALGEBRAEICS,SET,T)
        'FI';
        RUNALG(ALGEBRAEICS,CHANGED) );
   LC:ITERC ← 0;
      MAXSTEP Ξ H;
      CHANGED ← 'TRUE';
      'WHILE' (CHANGED'AND'(ITERC<10)) 'DO'
      ( S ← SET;
        CHANGED ← 'FALSF';
        ITERC'PLUS'1;
        'WHILE' S 'ISNT' ('REF''SAVE''VAL''NIL') 'DO'
        ( K2'OF'S ← RUNEQUATION(RHSIDE'OF'S);
          S ← NEXSAVE'OF'S );
```

LISTING 3.10 (contd.)

```
            S + SET;
            'WHILE' S 'ISNT' ('REF''SAVE''VAL''NIL') 'DO'
            ( STORE + VALUE'OF'VARIABLE'OF'S;
            VALUE'OF'VARIABLE'OF'S + K1'OF'S +0.0416667*H*( 9.0*K2'OF'S+
                                                            19.0*K3'OF'S-
                                                             5.0*K4'OF'S+
                                                                 K5'OF'S);
              'IF' 'NOT'(STORE=VALUE'OF'VARIABLE'OF'S)
              'THEN' CHANGED + 'TRUE'
              'FI';
              S + NEXSAVE'OF'S );
            RUNALG(ALGEBRAEICS,CHANGED) );
            'IF' ITERC = 10
            'THEN' ALTER(SET,H,COUNTER);'GOTO' START
            'FI';
            T 'PLUS' H;
            STEPSET(SET,H,ITERC,ITERA,COUNTER);
            'IF' PRINTCOUNT = PRINTNO
            'THEN' PRINTVALS(ALGEBRAEICS,SET,T);
                   PRINTCOUNT + 0
            'FI' );
        PRINT((NEWLINE,NEWLINE,"MAXIMUM STEPSIZE =",MAXSTEP,NEWLINE));
        PRINTVALS(ALGEBRAEICS,SET,T)
'END';
'C'-----------------------------------------------------------------------'C'
'PROC' SPLITUP = ('REF''SYSLIST' SYSTEM,'REF''REF''SAVE' SET,'REF''REF''ALG'
                                                                  ALGEBRAEICS):
'BEGIN'
        'REF''SYSLIST' SYS + SYSTEM;
        PRINT((NEWLINE,NEWLINE,"UNITS AND VARIABLES TO BE USED FOR INTEGRATION",
              NEWLINE,"A AND D DENOTE ALG. OR DIF. EQUATION",NEWLINE,NEWLINE));
        'WHILE' SYS 'ISNT' ('REF''SYSLIST''VAL''NIL') 'DO'
        (   'REF''EXPRESSION' E + RELATIONS'OF'SYSU'OF'SYS;
            'INT' ICOUNT;
            PRINT((NEWLINE,NEWLINE,UNAME'OF'SYSU'OF'SYS,NEWLINE,"****",
                                                                 NEWLINE));
            ICOUNT ← 1;
            'WHILE' E 'ISNT' ('REF''EXPRESSION''VAL''NIL') 'DO'
            (   'CHAR' C;
                'IF' GRADIENT'OF'E 'IS' ('REF''REAL''VAL''NIL')
                'THEN' ALGEBRAEICS ← 'ALG' ← (ALGEBRAEICS,LHS'OF'E,NORMAL'OF'E);
                       C ← "A"
                'ELSE' SET ← 'SAVE' ← (SET,LHS'OF'E,NORMAL'OF'E,0,0,0,0,0,0,0);
                       C ← "D"
                'FI';
                'IF' ICOUNT * 9 >= LOOKATNO + 10
                'THEN' PRINT(NEWLINE);ICOUNT ← 1
                'FI';
                PRINT(("   ",C," ",NAME'OF'LHS'OF'E));
                ICOUNT'PLUS'1;
                E ← NEXTEX'OF'E  );
            SYS ← NEXS'OF'SYS  )
'END':
'C'-------------------------------------------------------------------------'C'
'PROC' ADDALARMS = ('REF''UNIT' ULIST):
'BEGIN'
        'CHAR' MOREC,MOREI,MOREU ← ",";
        'BYTES' UNAME,VNAME;
        'REF''UNIT' U;
        'REF''NAMEVAL' N;
        'STRING' NE;
        'INT' TYPENO;
        'REAL' A,B,C,D,PERCENT,V;
        'REF''STANDARDS' S;
        PRINT((NEWPAGE,NEWLINE,"ADDALARMS OUTPUT",NEWLINE,
                              "*****************",NEWLINE));
```

LISTING 3.10 (contd.)

```
'WHILE' MOREU = "," 'DO'
(    MOREI ← ",";
     READ((NEWLINE,UNAME,MOREU));
     PRINT((NEWLINE,NEWLINE,"UNIT ",UNAME,NEWLINE,"**********",NEWLINE));
     U ← FINDUNIT(ULIST,UNAME);
     'WHILE' MOREI = "," 'DO'
     (    READ((NEWLINE,ME));
          CHOPSTRING(ME);
          PRINT((NEWLINE,ME));
          READ((NEWLINE,VNAME,TYPENO));
          PRINT((NEWLINE,VNAME));
          SEARCHFORVAR(VNAME,N,U,'TRUE');
          'CASE' TYPENO
          'IN' PERCENT ← 5.0;
               PERCENT ←10.0;
               PERCENT ←20.0;
               READ((A,B,C,D));V ← VALUE'OF'N;
           )      A'TIMES'V;B'TIMES'V;C'TIMES'V;D'TIMES'V;'GOTO'L1,
               READ((A,B,C,D));'GOTO'L1,
               READ((A,B,C,D));'GOTO'L1,
            READ((A,B));C ← 0.0;D ← 0.0;'GOTO' L1
          'OUT' PRINT((NEWLINE,"***** ERROR IN TYPENO *****"))
          'ESAC';
          V ← VALUE'OF'N;
          A ← (1.0 - PERCENT*0.01 )*V;
          B ← (1.0 - PERCENT*0.005)*V;
          C ← (1.0 + PERCENT*0.005)*V;
          D ← (1.0 + PERCENT*0.01 )*V;
       L1:S ← STAND'OF'N ← 'STANDARDS' ← (TYPENO,A,B,C,D,VALUE'OF'N,
                                                          U,'NIL','NIL',ME);
          PRINT((A,B,C,D,NEWLINE));
          SETCHARNUMBER(STANDIN,71);
          READ(MOREI)  )  )
```

```
'END';
'C'----------------------------------------------------------------------------'C'
'PROC' INPUTCHECKS = ('REF''UNIT' ULIST);
'BEGIN'
        'STRING' ME;
        'BOOL' MORE ← 'TRUE',NOSTAND ← 'TRUE';
        'INT' NO,NUM,COUNT;
        'REF''NAMEVAL' N;
        'BYTES' VNAME,UNAME;
        'REF''UNIT' U;
        'REF''CHECK' CK;
        'REF''CHECKLIST' CL;
        'REAL' TIME;
        PRINT((NEWPAGE,"INPUTCHECKS PRINTOUT",NEWLINE));
        'WHILE' MORE 'DO'
        (    READ((NEWLINE,ME));
             READ((NEWLINE,NO,SPACE,NUM,SPACE,MORE));
             CHOPSTRING(ME);
             PRINT((NEWLINE,NEWLINE,ME));
             PRINT((NEWLINE,NO,SPACE,NUM));
             CK ← 'CHECK' ← ('NIL',ME,CHECKNO);
             CHECKNO'PLUS'1;
             'FOR' J 'TO' NUM 'DO'
             (    READ((NEWLINE,UNAME,SPACE,VNAME,COUNT,SPACE,TIME));
                  PRINT((NEWLINE,UNAME,SPACE,VNAME,COUNT,SPACE,TIME));
                  U ← FINDUNIT(ULIST,UNAME);
                  'IF' SEARCHFORVAR(VNAME,N,U,'TRUE')
                  'THEN' NEXCH'OF'CK ← 'CHECKLIST'  ← (NEXCH'OF'CK,
                                                      N,COUNT,TIME,'NIL')
                  'ELSE' PRINT((NEWLINE,"ERROR IN INPUTCHECKS",NEWLINE))
                  'FI' );
             LINKEFFECT(NEXCH'OF'CK,CK)    );
        PRINT(NEWPAGE)
```

LISTING 3.10 (contd.)

```
'END';
'C'-------------------------------------------------------------------'C'
'PROC' CHECK5 = ('REF''STANDARDS'AL,'REF''NAMEVAL'N,'INT'I,'REAL'TIME)'REAL':
'BEGIN'
        'REAL' ERROR ← 0.0,V ← VALUF'OF'N;
        'IF' V < B'OF'AL
        'THEN' ERROR ← -1.0
        'ELSF'V> C'OF'AL
        'THEN' ERROR ←  1.0
        'FI';
        'IF' ( V <= A'OF'AL 'OR' V >= D 'OF'AL )
        'THEN' ERROR'TIMES'2
        'FI';
        ERROR ← ((ERROR - I)/4.0);
        ERROR
'END';
'C'-------------------------------------------------------------------'C'
'PROC' CHECK6 = ('REF''STANDARDS'AL,'REF''NAMEVAL'N,'INT'I,'REAL'TIME)'REAL':
'BEGIN'
        'REAL' ERROR ← 0.0,NOW ← VALUE'OF'N,RATE,DT ← TIME - B'OF'AL;
        'IF' DT <= 0
        'THEN' DT ← TIME
        'FI';
        RATE ← (NOW - A'OF'AL)/DT;
        'IF' RATE < C'OF'AL
        'THEN' ERROR ← -1.0
        'ELSF' RATE > D'OF'AL
        'THEN' ERROR ←  1.0
        'FI';
        ERROR ← ((ERROR - I)/2.0);
        A'OF'AL ← NOW;
        B'OF'AL ← TIME;
        ERROR
```

LISTING 3.10 (contd.)

```
'END';
'C'------------------------------------------------------------------------'C'
'PROC' CHECK7 = ('REF''STANDARDS'AL,'REF''NAMEVAL'N,'INT'I,'REAL'TIME)'REAL':
'BEGIN'
        'REAL' ERROR ← 0.0,V ← VALUE'OF'N;
        'IF' V <= A'OF'AL
        'THEN' ERROR ← -1.0
        'ELSF'V>= B'OF'AL
        'THEN' ERROR ←  1.0
        'FI';
        ERROR ← ((ERROR - I)/2.0);
        ERROR
'END';
'C'------------------------------------------------------------------------'C'
'PROC' RUNSYSTEM = ('REF''UNIT' ULIST)'REAL';
'BEGIN'
        'REAL' TMAX;
        'INT' METHODNO,PRINTNO,COLS;
        'REF''NAMEVAL' N;
        SEARCHFORVAR("HP   ",N,ULIST,'TRUE');
        'REF''REAL' H = VALUE'OF'N;
        SEARCHFORVAR("GT   ",N,ULIST,'TRUE');
        'REF''REAL' TIME = VALUE'OF'N;
        'REF''UNIT' U ← ULIST;
        'REF''SYSLIST' S ← 'NIL';
        'REF''ALG' ALG ← 'NIL';
        'REF''SAVE' SET ← 'NIL';
        PRINT((NEWPAGE,"RUNSYSTEM PRINTOUT",NEWLINE,"*******************",
                                                       NEWLINE,NEWLINE));
        'WHILE' U 'ISNT' ('REF''UNIT''VAL''NIL') 'DO'
        (     S ← 'SYSLIST' ← (U,'NIL','FALSE',S);
              U ← NEXU'OF'U     );
        SPLITUP(S,SET,ALG);
        READ((NEWLINE,TIME,H,TMAX,METHODNO,PRINTNO));
        MASTERINTEG(SET,ALG,TMAX,H,TIME,METHODNO,PRINTNO,COLS);
        TIME
'END';
'SKIP'
'END'
'KEEP' RUNALG,PRINTVALS,MASTERINTEG,SPLITUP,ADDALARMS,INPUTCHECKS,
       CHECK5,CHECK6,CHECK7,RUNSYSTEM
'FINISH'
```

LISTING 3.10 (contd.)

```
'BEGIN'
'PROC' CHECKSYSLIST = ('REF''SYSLIST' LIST,'REF''UNIT' UNIT)'BOOL':
'BEGIN'
        'REF''SYSLIST' S ← LIST;
        'BOOL' RESULT ← 'FALSE';
        'WHILE' S 'ISNT' ('REF''SYSLIST''VAL''NIL') 'DO'
        (    'IF' SYSU'OF'S 'IS' ('REF''UNIT''VAL'UNIT)
             'THEN' RESULT ← 'TRUE';'GOTO' LEND
             'ELSE' S ← NEXS'OF'S
             'FI'     );
        LEND:RESULT
'END';
'C'-----------------------------------------------------------------------'C'
'PROC' PRINTSYSLIST = ('REF''SYSLIST' LIST):
'BEGIN'
        'REF''SYSLIST' S ← LIST;
        PRINT(NEWLINE);
        'WHILE' S 'ISNT' ('REF''SYSLIST''VAL''NIL') 'DO'
        ( PRINT((LOOKAT;UNAME'OF'SYSU'OF'S,SPACE));
          S ← NEXS'OF'S );
        PRINT(NEWLINE)
'END';
'C'-----------------------------------------------------------------------'C'
'PROC' FINDLOOPS = ('REF''REF''SYSLIST' SYSTEM,OTHERS,'INT' LEVEL,
                                                     'REF''UNIT' UNIT)'BOOL':

'BEGIN'
        'INT' NEXTLEVEL ← LEVEL - 1;
        'BOOL' RESULT ← 'FALSE';
        'REF''UNIT' U;
        'REF''POINTER' P ← OUTLIST'OF'UNIT;
        'WHILE' P 'ISNT' ('REF''POINTER''VAL''NIL') 'DO'
        (     U ← UNIT'OF'LINK'OF'P;
                'IF' CHECKSYSLIST(SYSTEM,U)
                'THEN' RESULT ← 'TRUE'
                'ELSE' 'IF' 'NOT'CHECKSYSLIST(OTHERS,U)
                       'THEN' 'IF' LEVEL > 0
                              'THEN' OTHERS ← 'SYSLIST' ← (U,'NIL','FALSE',OTHERS);
                                     'IF' FINDLOOPS(SYSTEM,OTHERS,NEXTLEVEL,U)
                                     'THEN' RESULT ← 'TRUE';
                                            SYSTEM ← 'SYSLIST' ← (U,'NIL','TRUE',
                                                                           SYSTEM)
                                     'FI'
                              'FI'
                       'FI'
                'FI';
                P ← NEXP'OF'P  ):
           RESULT
'END';
'C'-------------------------------------------------------------------------------'C'
'PROC' INSANDOUTS = ('REF''UNIT' UNIT,'REF''REF''SYSLIST' SYSTEM):
'BEGIN'
        'REF''POINTER' P ← INLIST'OF'UNIT;
        'REF''UNIT' U;
        'BOOL' IN ← 'TRUE';
        LOOP:'WHILE' P 'ISNT' ('REF''POINTER''VAL''NIL') 'DO'
             ( U ← UNIT'OF'LINK'OF'P;
               'IF' 'NOT'CHECKSYSLIST(SYSTEM,U)
               'THEN' SYSTEM ←  'SYSLIST' ← (U,'NIL','TRUE',SYSTEM)
               'FI';
               P ← NEXP'OF'P );
             'IF' IN
             'THEN' IN ← 'FALSE';
                    P ← OUTLIST'OF'UNIT;
                    'GOTO' LOOP
             'FI'
```

LISTING 3.11 (contd.)

```
'END';
'C'---------------------------------------------------------------------------'C'
'PROC' SELECTSYSTEM = ('REF''REF''SYSLIST' SYSTEM,'INT' HOWMANY);
'BEGIN'
        'REF''SYSLIST' S ← SYSTEM;
        'REF''SYSLIST' OTHERS ← 'NIL';
        'INT' LEVEL ← HOWMANY;
        'BOOL' RESULT2 ← 'TRUE';
        'FOR' K 'TO' 3 'DO'
        (S ← SYSTEM;
        'WHILE' S 'ISNT' ('REF''SYSLIST''VAL''NIL') 'DO'
        (   INSANDOUTS(SYSU'OF'S,SYSTEM);
            S ← NEXS'OF'S     )     );
        'FOR' J 'WHILE' RESULT2'DO'
        ( S ← SYSTEM;
          RESULT2 ← 'FALSE';
          'WHILE' S 'ISNT' ('REF''SYSLIST''VAL''NIL') 'DO'
          ( 'IF' LEVEL > 1
            'THEN' 'IF' FINDLOOPS(SYSTEM,OTHERS,LEVEL,SYSU'OF'S)
                   'THEN' RESULT2 ← 'TRUE'
                   'FI'
            'FI';
            S ← NEXS'OF'S );
          PRINTSYSLIST(SYSTEM);
          LEVEL ← HOWMANY - J  )
'END';
'C'---------------------------------------------------------------------------'C'
'PROC' REINITIALISE = ('REF''SYSLIST' WHOLESET);
'BEGIN'
        'REF''SYSLIST' S ← WHOLESET;
        'REF''EXPRESSION' E;
        'REF'[ ]'REAL' R;
        'WHILE' S 'ISNT' ('REF''SYSLIST''VAL''NIL') 'DO'
        ( E ← RELATIONS'OF'SYSU'OF'S;
          P ← STORE 'OF' S;
             'FOR' J 'WHILE' E 'ISNT' ('REF''EXPRESSION''VAL''NIL') 'DO'
             ( VALUE'OF'LHS'OF'E ← R[J];
               E ← NEXTEX 'OF' E );
          S ← NEXS'OF'S )
'END';
'C'-------------------------------------------------------------------------'C'
'PROC'  STORELHSIDES = ('REF''UNIT' ULIST,'REF''SYSLIST' SYSTEM)'REF''SYSLIST';
'BEGIN'
        'REF''SYSLIST' WHOLESET ← 'NIL';
        'REF''UNIT' U ← ULIST;
        'REF''EXPRESSION' E;
        'INT' N;
        'REF'[ ]'REAL' R;
        'WHILE' U 'ISNT'('REF''UNIT''VAL''NIL')'DO'
        ( WHOLESET ← 'SYSLIST' ← (U,'NIL','IF' CHECKSYSLIST(SYSTEM,U)
                                          'THEN' 'TRUE'
                                          'ELSE' 'FALSE'
                                          'FI',WHOLESET);
          E ← RELATIONS'OF'U;
          N ← 0;
          'WHILE' E 'ISNT' ('REF''EXPRESSION''VAL''NIL') 'DO'
          (N 'PLUS' 1;
           E ← NEXTEX'OF'E );
          R ← STORE'OF'WHOLESET ← [1:N]'REAL';
          E ← RELATIONS'OF'U;
          'FOR' J 'TO' N 'DO'
          ( R[J] ← VALUE'OF'LHS'OF'E;
            E ← NEXTEX'OF'E );
          U ← NEXU 'OF' U );
        WHOLESET
'END';
```

```
'C'-------------------------------------------------------------------------'C'
'PROC' ALARM = ('REF''CHECK'CK,'REF''NAMEVAL'N,'REAL'TIME,'REF''INT'NUMBER);
'BEGIN'
        'REF''CHECKLIST' C ← NEXCH'OF'CK;
        'REF''STANDARDS' S ← STAND'OF'N;
        'INT'I ← 'ROUND''CASE' (TYPE'OF'S  - 5)
                        'IN' 2.0*CHECK6(S ,N,0,TIME),
                             2.0*CHECK7(S ,N,0,TIME)
                        'OUT'4.0*CHECK5(S ,N,0,TIME)
                        'ESAC';
        'BOOL' RESULT ← 'IF' I = 0
                        'THEN' 'TRUF'
                        'ELSE' 'FALSE'
                        'FI';
        'WHILE' C 'ISNT' ('REF''CHECKLIST''VAL''NIL') 'DO'
        ( 'IF' NAVAL'OF'C 'IS' ('REF''NAMEVAL''VAL'N)
          'THEN' 'IF' I = COUNT'OF'C
                 'THEN' RESULT ← 'TRUE'
                 'ELSE' RESULT ← 'FALSE'
                 'FI';
                 C ← 'NIL'
          'ELSE' C ← NEXCH'OF'C
          'FI'   );
        'IF''NOT'RESULT
        'THEN' NEXCH'OF'CK←'CHECKLIST'←(NEXCH'OF'CK,N,I,TIME,'NIL');
               PRINT((NEWLINE,NEWLINE,UNAME'OF'UNIT'OF'S,"    ",NAME'OF'N,"  ",
                       VALUE'OF'N,NORM'OF'S,I,"  ",ME'OF'S,NEWLINE));
               NUMBER'PLUS'1
        'FI'
'END';
'C'-------------------------------------------------------------------------'C'
'PROC' SYSCHECK = ('REF''SYSLIST' WHOLESET,'REF''REF''SYSLIST' SYSTEM,
                   'REF''REF''ALG' ALGEBRAEICS,'REF''REF''SAVE' SET,
```

LISTING 3.11 (contd.)

```
                    'REAL' TIME,'REF''INT'NUMBER,'REF''CHECK'CK)'BOOL':
'BEGIN'
        'REF''EXPRESSION' E;
        'BOOL' ALTERED ← 'FALSE';
        'REF''SYSLIST' S ← WHOLESET,NEW;
        'REF''NAMEVAL' N;
        'REF'[ ]'REAL' R;
        'REAL' NEWVAL;
        'BOOL' OK ← 'FALSE';
        AFTER'OF'NUMBERSTYLE ← 5;
        'WHILE' S 'ISNT' ('REF''SYSLIST''VAL''NIL') 'DO'
        ( E ← RELATIONS'OF'SYSU'OF'S;
          R ← STORE'OF'S;
        'FOR' I 'WHILE' E 'ISNT'('REF''EXPRESSION''VAL''NIL') 'DO'
        (     N ← LHS'OF'E;
              'IF' INSYSTEM'OF'S
              'THEN' 'IF' STAND'OF'N'ISNT'('REF''STANDARDS''VAL''NIL')
                     'THEN' ALARM(CK,N,TIME,NUMBER)
                     'FI'
              'ELSE' NEWVAL ← RUNEQUATION(NORMAL'OF'E);
              OK ← 'IF' GRADIENT'OF'E'IS'('REF''REAL''VAL''NIL')
                   'THEN' (R[I] = NEWVAL)
                   'ELSE' (R[I] = VALUE'OF'N +NEWVAL*TIME)
                   'FI';
                   'IF'('NOT'OK 'AND' MOREUNITS)
                   'THEN' NEW←'SYSLIST'←(SYSU'OF'S,'NIL','TRUE','NIL');
                          SPLITUP(NEW,SET,ALGEBRAEICS);
                          INSYSTEM'OF'S ← 'TRUE';
                          NEXS'OF'NEW ← SYSTEM;
                          SYSTEM ← NEW;
                          PRINT((NEWLINE,NEWLINE,"UNIT  ",UNAME'OF'SYSU'OF'S,
                                 "  ADDED TO SYSTEM",NEWLINE));
                          PRINTSYSLIST(SYSTEM);
```

LISTING 3.11 (contd.)

```
                          OK ← 'FALSE';
                          ALTERED ← 'TRUE'
                    'FI'
              'FI';
              F ← NEXTEX'OF'F     );
          S ← NEXS 'OF' S );
        AFTER'OF'NUMBERSTYLE ← 10;
        ALTERED
'END';
'C'-----------------------------------------------------------------------------'C'
'PROC' RESTARTVALS = ('REF''SYSLIST' SYSTEM);
'BEGIN'
        'REF''SYSLIST' S ← SYSTEM;
        'REF''FAULTVAL' F;
        'WHILE' S 'ISNT' ('REF''SYSLIST''VAL''NIL') 'DO'
        (    F ← FAULTVALS'OF'SYSU'OF'S;
             'WHILE' F 'ISNT' ('REF''FAULTVAL''VAL''NIL') 'DO'
             (    VALUE'OF'NV'OF'F ← NEW'OF'F;
                  F ← NEXT'OF'F  );
             S ← NEXS'OF'S  );
        PRINT((NEWLINE,NEWLINE))
'END';
'C'-----------------------------------------------------------------------------'C'
'PROC' KILLFAULTS = ('REF''SYSLIST' SYSTEM);
'BEGIN'
        'REF''SYSLIST' S ← SYSTEM;
        'REF''EXPRESSION' E;
        'REF''FAULTVAL' F;
        'WHILE' S 'ISNT' ('REF''SYSLIST''VAL''NIL') 'DO'
        (     F ← FAULTVALS'OF'SYSU'OF'S;
              'WHILE' F 'ISNT' ('REF''FAULTVAL''VAL''NIL') 'DO'
              (    VALUE'OF'NV'OF'F ← OLD'OF'F;
              FAULTVALS'OF'SYSU'OF'S ← 'NIL';
```

```
                  F ← NEXT'OF'F  );
             E ← RELATIONS'OF'SYSU'OF'S;
             'WHILE' E 'ISNT' ('REF''EXPRESSION''VAL''NIL') 'DO'
             (    'IF' ALTER'OF'E 'ISNT' ('REF''EQUATION''VAL''NIL')
                  'THEN' NORMAL'OF'E ← ALTER'OF'E;
                         ALTER'OF'E ← 'NIL'
                  'FI';
                  E ← NEXTEX'OF'E  );
             S ← NEXS'OF'S  );
        PRINT((NEWLINE,NEWLINE))
'END';
'C'-----------------------------------------------------------------------------'C'
'PROC' DEFINEFAULT = ('REF''REF''SYSLIST' SYSTEM,'REF''UNIT' ULIST,
                      'REF''STRING' ME,'REF''BOOL' MOREFAULTS,[]'INT' TABLE);
'BEGIN'
        'REF''UNIT' U;
        'BYTES' UNAME,VNAME;
        'REAL' VAL;
        'BOOL' MORE ← 'TRUE',ALG,VAR,EXALG;
        'REF''NAMEVAL' N,NV;
        'REF''EXPRESSION' EX;
        [1:4]'CHAR' CC;
        'INT' L;
        READ((NEWLINE,ME));
        CHOPSTRING(ME);
        PRINT((NEWLINE,NEWLINE,"FAULT MESSAGE:",NEWLINE,ME,NEWLINE));
        'WHILE' MORE 'DO'
        (    READ((NEWLINE,UNAME,SPACE,VAR,SPACE,MORE));
             PRINT((NEWLINE,UNAME));
             U ← FINDUNIT(ULIST,UNAME);
             'IF' 'NOT'CHECKSYSLIST(SYSTEM,U)
             'THEN' SYSTEM ← 'SYSLIST' ← (U,'NIL','TRUE',SYSTEM)
             'FI';
```

```
'IF' VAR
'THEN' READ((SPACE,VNAME,SPACE,VAL));
       PRINT(("   VALUE OF ",VNAME, "IS NOW =",VAL));
       'IF' SEARCHFORVAR(VNAME,N,U,'TRUE')
       'THEN' FAULTVALS'OF'U ← 'FAULTVAL' ← (FAULTVALS'OF'U,
                                              VALUE'OF'N,VAL,N)
       'ELSE' PRINT((NEWLINE,"ERROR IN DEFINEFAULT",NEWLINE))
       'FI'
'ELSE' EX ← RELATIONS'OF'U;
       PRINT(("   NEW EQUATION:",NEWLINE));
       'BEGIN'
               'REF'[]'CHAR' C = [1:0'FLEX']'CHAR';
               READ(NEWLINE);
               PREPSOURCE(80,C,TABLE);
               'IF' C[2] = "(" 'AND' C[5] = ")"
               'THEN' CC ← (C[3],C[4]," "," ");
                      ALG ← 'FALSE';
                      L ← 9
               'ELSE' CC ← (C[1],C[2]," "," ");
                      ALG ← 'TRUE';
                      L ← 3
               'FI';
               VNAME ← 'CTR'CC;
               SEARCHFORVAR(VNAME,NV,U,'TRUE');
               'WHILE' EX 'ISNT' ('REF''EXPRESSION''VAL''NIL')'DO'
               (    'IF' LHS'OF'EX'IS'('REF''NAMEVAL''VAL'NV)
                    'THEN' 'IF' GRADIENT'OF'EX'IS'
                                   ('REF''REAL''VAL''NIL')
                           'THEN' EXALG ← 'TRUE'
                           'ELSE' EXALG ← 'FALSE'
                           'FI';
                           'IF' ( (EXALG'AND'ALG) 'OR'
                                  ('NOT'EXALG'AND''NOT'ALG) )
```

LISTING. 3.11 (contd.)

```
                                        'THEN'ALTER'OF'EX ← NORMAL'OF'EX;
                                              NORMAL'OF'EX ← RHS(L,U,C,'TRUE');
                                              'GOTO' LABEL
                                        'FI'
                                'FI';
                                EX ← NEXTEX'OF'EX    );
                            LABEL:'SKIP'
                    'END'
             'FI' );
        READ((NEWLINE,MOREFAULTS))
'END';
'C'--------------------------------------------------------------------------'C'
'PROC' MASTEREFFECT = ('REF''UNIT'ULIST,'INT'HOWMANY,[]'INT' TABLE):
'BEGIN'
        'BOOL' MOREFAULTS ← 'TRUE';
        'REF''SYSLIST' SYSTEM ← 'NIL',WHOLESET ← 'NIL';
        'INT' LOOPCOUNT,NUMBER,METHODNO,COLS;
        'REF''CHECK' CK;
        'REAL' TMAX;
        'REF''NAMEVAL' N;
        SEARCHFORVAR("GT  ",N,ULIST,'TRUE');
        'REF''REAL' T = VALUE'OF'N;
        SEARCHFORVAR("HH  ",N,ULIST,'TRUE');
        'REF''REAL' H = VALUE'OF'N;
        'REF''SAVE' SET ← 'NIL';
        'REF''ALG' ALG ← 'NIL';
        'STRING' ME;
        PRINT((NEWPAGE,"MASTEREFFECT OUTPUT",NEWLINE,
                       "*******************",NEWLINE));
        'WHILE' MOREFAULTS 'DO'
        (     DEFINEFAULT(SYSTEM,ULIST,ME,MOREFAULTS,TABLE);
              SELECTSYSTEM(SYSTEM,HOWMANY);
              SPLITUP(SYSTEM,SET,ALG);
```

```
WHOLESET ← STORELHSIDES(ULIST,SYSTEM);
RESTART:RESTARTVALS(SYSTEM);
        H ← SMALLSTEP;
        CK ← 'CHECK' ← ('NIL',ME,CHECKNO);
        CHECKNO'PLUS'1;
        TMAX ← 1000.0 * H;
        METHODNO ← 1;
        T ← NUMBER ← LOOPCOUNT ← 0;
INLABEL:MASTEPINTEG(SET,ALG,TMAX,H,T,METHODNO,PRINTNO,COLS);
        LOOPCOUNT'PLUS'1;
        'IF' 'NOT'SYSCHECK(WHOLESET,SYSTEM,ALG,SET,T,NUMBER,CK)
        'THEN' 'IF' NUMBER*LOOPCOUNT > TARGET
               'THEN' LINKEFFECT(NEXCH'OF'CK,CK)
               'ELSE' 'IF' LOOPCOUNT < TARGET + 10
                      'THEN' TMAX ← T + 100.0*H;
                             METHODNO ← 2;
                             'GOTO' INLABEL
                      'ELSE' 'IF' NUMBER > 0
                             'THEN' LINKEFFECT(NEXCH'OF'CK,CK)
                             'ELSE' PRINT((NEWLINE,
                                          "***NO EFFECTS***"))
                             'FI'
                      'FI'
               'FI';
               'IF' RESET 'THEN' REINITIALISE(WHOLESET) 'FI';
               KILLFAULTS(SYSTEM);
               SET ← 'NIL';
               ALG ← 'NIL';
               SYSTEM ← 'NIL';
               WHOLESET ← 'NIL'
        'ELSE' REINITIALISE(WHOLESET);
               'GOTO' RESTART
        'FI' )
```

LISTING 3.11 (contd.)

```
'END';
'SKIP'
'END';
'KEEP' MASTEREFFECT
'FINISH'
```

LISTING 3.11 (contd.)

```
'BEGIN'
'PROC' FINDNODE = ('REF''NODE' NODELIST,'REF''NAMEVAL'NV)'REF''NODE';
'BEGIN'
        'REF''NODE' N ← NODELIST;
        'WHILE' N 'ISNT' ('REF''NODE''VAL''NIL') 'DO'
        (     'IF' NAVAL'OF'N 'IS' ('REF''NAMEVAL''VAL'NV)
              'THEN' 'GOTO' LEND
              'ELSE' N ← STRAIGHT'OF'N
              'FI'  );
        PRINT((NEWLINE,"**** ERROR - NO NODE FOUND ****",NEWLINE));
   LEND:N
'END';
'C'-------------------------------------------------------------'C'
'PROC' ADDBRANCH = ('REF''NODE'NODELIST,'REF''NAMEVAL'FROM,TO,'REAL'ADD,SUB):
'BEGIN'
        'REF''NODE' NF ← FINDNODE(NODELIST,FROM),NT ← FINDNODE(NODELIST,TO);
        CAUSES 'OF'NT ← 'BRANCH' ← (NF,ADD,SUB,CAUSES 'OF'NT)
'END';
'C'-------------------------------------------------------------'C'
'PROC' CHECKVAR = ('REF''FAULTVAL' LIST,'REF''NAMEVAL' N)'BOOL';
'BEGIN'
        'REF''FAULTVAL' F ← LIST;
        'BOOL' FOUND ← 'FALSE';
        'WHILE' F 'ISNT' ('REF''FAULTVAL''VAL''NIL')'DO'
        (    'IF' NV'OF'F  'IS' ('REF''NAMEVAL''VAL'N)
             'THEN' FOUND ← 'TRUE';'GOTO' LAST
             'ELSE' F ← NEXT'OF'F
             'FI'  ));
   LAST:FOUND
'END';
'C'-------------------------------------------------------------'C'
'PROC' ARITHDIRE = ('REF''EQUATION' EQ,'REF''FAULTVAL' LIST,'REF''NODE'NODELIST,
                                                           'REF''NAMEVAL' TO,'INT' SIZE);
'BEGIN'
      'REF''FAULTVAL' FV ← NEXT'OF'LIST,FW;
      'INT' SIZESQ ← SIZE*(SIZE+2),PLUSNO,MINUSNO;
      'REAL' RANRES,RANVAL,RES,VAL,RATIO,PLUS,MINUS;
      'REF''NAMEVAL' FROM;
      'FOR' J 'TO' SIZE 'DO'
      (   PLUS ← MINUS ← PLUSNO ← MINUSNO ← 0;
          FROM ← NV'OF'FV;
          'FOR' K 'TO' SIZESQ 'DO'
          (   FW ← LIST;
              'FOR' L 'FROM' 0 'TO' SIZE 'DO'
              (   VALUE'OF'NV'OF'FW 'PLUS' ( 0.05*(RANDOM - 0.5) *
                                                         VALUE'OF'NV'OF'FW);
                  FW ← NEXT'OF'FW );
              RANRES ← RUNEQUATION(EQ);
              RANVAL ← VALUE'OF'FROM;
              VALUE'OF'FROM 'TIMES' 1.01;
              RES ← RUNEQUATION(EQ);
              VAL ← VALUE'OF'FROM;
              'IF' 'ABS'(VAL - RANVAL) > 1.0R-10
              'THEN' RATIO ← (RANRES - RES)/(RANVAL - VAL);
                     'IF' RATIO > 0
                     'THEN' PLUS'PLUS'RATIO; PLUSNO'PLUS'1
                     'ELSE' MINUS'PLUS'RATIO;MINUSNO'PLUS'1
                     'FI'
              'FI';
              FW ← LIST;
              'FOR' L 'FROM' 0 'TO' SIZE 'DO'
              (   VALUE'OF'NV'OF'FW ← OLD'OF'FW;
                  FW ← NEXT'OF'FW  ) );
              'IF' SIZESQ ≠ 0
              'THEN' MINUS ← (PLUS+MINUS)/SIZESQ;
```

LISTING 3.12 (contd.)

```
                       PLUS ← (2.0 * PLUSNO/SIZESQ - 1);
                       ADDBRANCH(NODELIST,FROM,TO,PLUS,MINUS)
                'FI';
                FV ← NEXT'OF'FV  )
'END';
'C'--------------------------------------------------------------------------'C'
'PROC' FUNCTDIRE = ('REF''EQUATION' EQ,'REF''FAULTVAL' LIST,'REF''NODE'NODELIST,
                                                   'REF''NAMEVAL' TO,'INT' SIZE);
'BEGIN'
        'REF''EQUATION' E;
        'REF''FAULTVAL' F ← NEXT'OF'LIST;
        'REF''NAMEVAL' N,A;
        'REF''FUNCTION' FU;
        'CHAR' C;
        'FOR' J 'TO' SIZE 'DO'
        (   N ← NV'OF'F;
            E ← EQ;
            'FOR' I 'DO'
            (   'CASE' (A,FU,C) ::= CONTENTS'OF'E
                'IN' 'IF' A  'IS' ('REF''NAMEVAL''VAL'N)
                     'THEN' 'IF' A ::= CONTENTS'OF'NEXT'OF'E
                            'THEN' ADDBRANCH(NODELIST,N,TO,VALUE'OF'A,0);
                                  'GOTO' L1
                            'ELSE' PRINT((NEWLINE,"**** ERROR IN FUNC ****"))
                            'FI'
                     'FI',
                     'SKIP',
                     'SKIP'
                'ESAC';
                E ← NEXT'OF'E  );
            L1:F ← NEXT'OF'F  )
'END';
'C'--------------------------------------------------------------------------'C'
```

LISTING 3.12 (contd.)

```
'PROC' SETUPTREE = ('REF''UNIT' UNITLIST)'REF''NODE':
'BEGIN'
        'REF''FAULTVAL' VARIAPLFS ← 'NIL';
        'INT' X,VARNO ← 1;
        'REF''POINTER' P;
        'REF''UNIT' U ← UNITLIST;
        'REF''EXPRESSION' FX,EQ;
        'PEF''NODE' NODELIST ← 'NIL';
        'PEF''FUNCTION' F;
        'REF''NAMEVAL' N,TO,FROM;
        'REF''FAULTVAL' LIST,FV;
        'INT' SIZE;
        'PEF''EQUATION' Q;
        'CHAR' C:
        'WHILE' U 'ISNT' ('REF''UNIT''VAL''NIL') 'DO'
        (    EQ ← RELATIONS'OF'U;
             'WHILE' EQ 'ISNT' ('REF''FXPRESSION''VAL''NIL') 'DO'
             (    'IF' CHECKVAR(VARIABLES,LHS'OF'EQ)
                  'THEN' PRINT((NEWLINE,"VARIABLE SET TWICE",NAME'OF'LHS'OF'EQ,
                                NEWLINF,"************************",NEWLINE))
                  'ELSE' TO ← LHS'OF'EQ;
                         'IF' STAND'OF'TO'ISNT'('REF''STANDARDS''VAL''NIL')
                         'THEN' 'IF' 3'FLEM'(NAME'OF'TO) = " "
                                'THEN' X ← 1
                                'ELSE' P ← INLIST'OF'U;
                                       C ← 2'ELEM'(NAMF'OF'TO);
                                       'WHILE' P 'ISNT'('REF''POINTER''VAL'
                                                                  'NIL') 'DO'
                                       (  'IF' NAME'OF'P = C
                                          'THEN' 'FOR' J 'TO' 6 'DO'
                                                 ( 'IF' (VAPI'OF'P)[J] 'IS'
                                                        ('REF''NAMEVAL''VAL'TO)
                                                   'THEN' X ← 2;
                                                   'GOTO' CREATE
                                            'FI')
                                  'FI';
                                  P ← NEXP'OF'P );X ← 3;
                               CREATE:'SKIP'
                        'FI'
                 'ELSE' X ← 0
                 'FI';
                 VARIABLES ← 'FAULTVAL' ← (VARIABLES,
                                           VALUE'OF'TO,VARNO,TO);
                 NODELIST ← 'NODE' ← (TO,NODENO+X*1000,'TRUE',
                                              'NIL','NIL',NODELIST);
                 NODENO'PLUS'1;
                 VARNO 'PLUS'1
          'FI';
          EQ ← NEXTEX'OF'EQ );
     U ← NEXU'OF'U  );
U ← UNITLIST;
'WHILE' U 'ISNT' ('REF''UNIT''VAL''NIL') 'DO'
(    EQ ← RELATIONS'OF'U;
    'WHILE' EQ 'ISNT' ('REF''EXPRESSION''VAL''NIL') 'DO'
    (    SIZE ← 0;
         Q ← NORMAL'OF'EQ;LIST ←'FAULTVAL'←('NIL',VALUE'OF'LHS'OF'EQ,0,
                                                             LHS'OF'EQ);
         'WHILE' Q 'ISNT' ('REF''EQUATION''VAL''NIL') 'DO'
         (    'CASE' (N,F,C) ::= CONTENTS'OF'Q
              'IN' 'SKIP',
                   N ← ARG'OF'F,
                   'GOTO' L1
              'ESAC';
              'IF' (CHECKVAR(VARIABLES,N) 'AND''NOT'CHECKVAR(LIST,N))
              'THEN' NEXT'OF'LIST ← 'FAULTVAL' ← (NEXT'OF'LIST,
                                                   VALUE'OF'N,0,N);
```

LISTING 3.12 (contd.)

LISTING 3.12 (contd.)

```
                            SIZE'PLUS'1
                     'FI';
                     L1:Q ← NEXT'OF'Q    );
                TO ← LHS'OF' EQ;
                'IF' FUNCMODELS
                'THEN' FUNCTDIRE(NORMAL'OF'EQ,LIST,NODELIST,TO,SIZE)
                'ELSE' ARITHDIRE(NORMAL'OF'EQ,LIST,NODELIST,TO,SIZE)
                'FI';
                EQ ← NEXTEX'OF'EQ   );
            U ← NEXU'OF'U   );
        NODELIST
'END';
'C'-------------------------------------------------------------------------'C'
'PROC' TREEPRINTER = ('REF''NODE' NL,'INT' LEVEL,MAXLEVEL):
'BEGIN'
        'INT' NEXTLEVEL ← LEVEL - 1;
        'REF''NODE' F ← NL;
        [1:120]'CHAR' INSET;
        'CLEAR' INSET;
        'REF''BRANCH' B;
        'REF''NAMEVAL' N;
        'INT' J ← (MAXLEVEL - LEVEL) * 15;
        PRINT((NEWLINE,INSET[1:J]));
             'IF' CHARNUMBER(STANDOUT) > 100
             'THEN' PRINT((NEWLINE,INSET[1:J]))
             'FI';
             N ← NAVAL'OF'F;
             TJONLY(F);
             'IF' STAND'OF'N'ISNT' ('REF''STANDARDS''VAL''NIL')
             'THEN' PRINT(("(",UNAME'OF'UNIT'OF'STAND'OF'N,")   "))
             'ELSE' PRINT("          ")
             'FI';
             'IF' LEVEL ≠ 0
             'THEN' B ← CAUSES'OF'F;
                       'WHILE' B 'ISNT' ('REF''BRANCH''VAL''NIL') 'DO'
                       (    PPINT((ADD'OF'B,"  ",SUB'OF'B,"  "));
                            TREFPRINTER(NEXNODF'OF'B,NEXTLEVEL,MAXLEVEL);
                            PRINT((NEWLINE,INSET[1:I+12]));
                            B ← NEXBRAN'OF'B    )
             'FI'
'END';
'C'--------------------------------------------------------------------------'C'
'PROC' SCRAP BRANCH = ('REF''REF''BRANCH' B,'REF''BRANCH' THISONE):
'BEGIN'
        'IF' B 'IS' ('REF''BRANCH''VAL'THISONE)
        'THEN' B ← NEXBRAN'OF'B
        'ELSE' SCRAP BRANCH(NEXBRAN'OF'B,THISONE)
        'FI'
'END';
'C'--------------------------------------------------------------------------'C'
'PROC' ALTER TREE = ('REF''REF''NODE' NL):
'BEGIN'
        'REF''NODE' N1;
        'REF''BRANCH' B1,B2,B3;
        'REAL' ADD,SUB;
        'INT' I;
        'BOOL' CHANGED ← 'TRUE';
        'WHILE' CHANGED 'DO'
        (  N1 ← NL;
           CHANGED ← 'FALSE';
           'WHILE' N1 'ISNT' ('REF''NODE''VAL''NIL') 'DO'
           (  B1 ← CAUSES'OF'N1;
              'WHILE' B1 'ISNT' ('REF''BRANCH''VAL''NIL') 'DO'
              (   'IF' STAND'OF'NAVAL'OF'NEXNODF'OF'B1
                                              'IS' ('REF''STANDARDS''VAL''NIL')
                  'THEN' B2 ← CAUSES'OF'NEXNODE'OF'B1;
```

LISTING 3.12 (contd.)

```
           ADD ← ADD'OF'B1;
           SUB ← SUB'OF'B1;
           I ← COUNT'OF'NEXNODE'OF'B1;
           SCRAP BRANCH(CAUSES'OF'N1,B1);
           CHANGED ← 'TRUE';
           'WHILE' B2 'ISNT' ('REF''BRANCH''VAL''NIL') 'DO'
           (  'IF' NEXNODE'OF'B2 'ISNT' ('REF''NODE''VAL'N1)
              'THEN' B3 ← CAUSES'OF'N1;
                     'WHILE' B3 'ISNT' ('REF''BRANCH'
                                                 'VAL''NIL') 'DO'
                     (  'IF' NEXNODE'OF'B3 'IS' ('REF''NODE''VAL'
                                                  NEXNODE'OF'B2)
                        'THEN' 'GOTO' LABEL
                        'ELSE' B3 ← NEXBRAN'OF'B3
                        'FI' );
                     CAUSES'OF'N1 ← 'BRANCH' ← (NEXNODE'OF'B2,
                                                 ADD'OF'B2+ADD,
                                                 SUB'OF'B2+SUB,
                                                 CAUSES'OF'N1)
              'FI';
              LABEL:B2 ← NEXBRAN'OF'B2  )
    'FI';
    B1 ← NEXBRAN'OF'B1 );
N1 ← STRAIGHT'OF'N1  );
PRINT((NEWLINE,"CHECK FOR GARBAGE"));
NL ← N1 ← 'NODE' ← ('NIL',0,'FALSE','NIL','NIL',NL);
'WHILE' STRAIGHT'OF'N1 'ISNT' ('REF''NODE''VAL''NIL') 'DO'
(   'REF''NODE' N2 ← STRAIGHT'OF'NL;
    'IF' STAND'OF'NAVAL'OF'STRAIGHT'OF'N1'ISNT'
                         ('REF''STANDARDS''VAL''NIL')
    'THEN' N1←STRAIGHT'OF'N1;'GOTO' LON
    'FI';
    'WHILE' N2 'ISNT' ('REF''NODE''VAL''NIL') 'DO'
```

LISTING. 3.12 (contd.)

```
                  (   B1 ← CAUSES'OF'N2;
                      'WHILE' B1 'ISNT' ('REF''BRANCH''VAL''NIL') 'DO'
                      (   'IF' NEXNODE'OF'B1'IS'('REF''NODE''VAL'STRAIGHT'OF'N1)
                          'THEN' N1 ← STRAIGHT'OF'N1;
                                 'GOTO' LON
                          'ELSE' B1 ← NEXBRAN'OF'B1
                          'FI'  );
                      N2 ← STRAIGHT'OF'N2  );
                  STRAIGHT'OF'N1 ← STRAIGHT'OF'STRAIGHT'OF'N1;
                  LON:'SKIP'    );
              NL ← STRAIGHT'OF'NL   )
'END';
'C'--------------------------------------------------------------------------------'C'
'PROC' PRINTREE = ('REF''NODE'N,'INT'MAXLEVEL):
'BEGIN'
        'REF''NODE' F ← N;
        AFTER'OF'NUMBERSTYLE ← 4;
        PRINT((NEWPAGE,"PRINTREE OUTPUT",NEWLINE,"****************",NEWLINE));
        'WHILE' F 'ISNT' ('REF''NODE''VAL''NIL')'DO'
        (    TREEPRINTER(F,MAXLEVEL,MAXLEVEL);
             F ← STRAIGHT'OF'F    );
        AFTER'OF'NUMBERSTYLE ← 10
'END';
'SKIP'
'END'
'KEEP' SETUPTREE,PRINTREE,ALTERTREE
'FINISH'
```

LISTING 3.12 (contd.)

```
'BEGIN'
'PROC' INSERT = ('REF''ALARM' CAUSE,EFFECT)'BOOL':
'BEGIN'
        'BOOL' ADDED ← 'FALSE';
        'IF' NEXT'OF'CAUSE 'IS' ('REF''ALARM''VAL''NIL')
        'THEN' 'IF' CAUSE 'ISNT' ('REF''ALARM''VAL'EFFECT)
               'THEN' NEXT'OF'CAUSE ← EFFECT;
                      ADDED ← 'TRUE'
               'FI'
        'ELSE' ADDED ← INSERT(NEXT'OF'CAUSE,EFFECT)
        'FI';
        ADDED
'END';
'C'------------------------------------------------------------------------'C'
'PROC' CONDITION = ('REF''CHECKLIST' CL,'REAL' TIME)'BOOL':
'BEGIN'
        'REF''NAMEVAL' N ← NAVAL'OF'CL;
        'REF''STANDARDS' S ← STAND'OF'N;
        'INT' I ← COUNT'OF'CL;
        'IF' 0 = 'ROUND''CASE' TYPE'OF'S - 5
                        'IN' 2.0*CHECK6(S,N,I,TIME),
                             2.0*CHECK7(S,N,I,TIME)
                        'OUT'4.0*CHECK5(S,N,I,TIME)
                        'ESAC'
        'THEN' 'TRUE'
        'ELSE' 'FALSE'
        'FI'
'END';
'C'------------------------------------------------------------------------'C'
'PROC' SCAN = ('REF''NODE'NL,'REAL'TIME)'REF''ALARM':
'BEGIN'
        'REF''ALARM' HEAD ← 'ALARM' ← ('NIL',NL,0,'FALSE','FALSE',"SKIP");
```

```
'REF''ALARM' AL ← HEAD;
'REF''NODE' N ← NL;
'REF''NAMEVAL' VAR;
'REF''EFFECT' E;
'REF''CHECKLIST' C;
'REF''STANDARDS' S;
'WHILE' N 'ISNT' ('REF''NODE''VAL''NIL') 'DO'
(    VAR ← NAVAL'OF'N;
     S ← STAND'OF'VAR;
     'IF' 0 = 'ROUND''CASE' (TYPE'OF'S - 5)
                       'IN'  2.0*CHECK6(S,VAR,0,TIME),
                             2.0*CHECK7(S,VAR,0,TIME)
                       'OUT' 4.0*CHECK5(S,VAR,0,TIME)
                       'ESAC'
     'THEN' 'IF' 'NOT'(NORMAL'OF'N)
            'THEN' PRINT((NEWLINE,NAME'OF'VAR,"  O.K."));
                   NORMAL'OF'N ← 'TRUE';
                   ACTIVE'OF'ENTRY'OF'N ← 'FALSE'
            'FI'
     'ELSE' 'IF' NORMAL'OF'N
            'THEN' PRINT((NEWLINE,NAME'OF'VAR,"  ",VALUE'OF'VAR));
                   NORMAL 'OF'N ← 'FALSE';
                   AL ← NEXT'OF'AL ← 'ALARM' ← ('NIL',N,TIME,
                                                'TRUE','FALSE',"SKIP");
                   ENTRY'OF'N ← AL
            'FI'
     'FI';
     N ← STRAIGHT'OF'N  );
N ← NL;
'WHILE' N 'ISNT' ('REF''NODE''VAL''NIL') 'DO'
(    S ← STAND'OF'NAVAL'OF'N;
     'FOR' J 'TO' 3 'DO'
      (     E ← 'CASE' J
```

LISTING 3.13 (contd.)

LISTING 3.13 (contd.)

```
                      'IN' HIEFLIST'OF'S,
                           LOEFLIST'OF'S
                      'ESAC';
                  'WHILE' E 'ISNT' ('REF''EFFECT''VAL''NIL') 'DO'
                  (    C ← NEXCH'OF'CHECK'OF'E;
                       'WHILE' C 'ISNT' ('REF''CHECKLIST''VAL''NIL') 'DO'
                       (    'IF' CONDITION(C,TIME)
                            'THEN' C ← NEXCH'OF'C
                            'ELSE' 'GOTO' LABEL
                            'FI'  );
                       AL ← NEXT'OF'AL ← 'ALARM' ← ('NIL',CHECK'OF'E,TIME,
                                                         'TRUE','FALSE',"SKIP");
                       LABEL:E ← NEXEF'OF'E )  );
            N ← STRAIGHT'OF'N   );
        NEXT'OF'HEAD
'END';
'C'-------------------------------------------------------------------------'C'
'PROC' ANALYSE = ('REF''REF''FAULT'LIST,'REF''ALARM'NEWONES,'REF''NODE'NL);
'BEGIN'
        'REF''ALARM' DED,AL ← NEWONES;
        'REF''FAULT' F;
        'REF''CHECK' CK,CD;
        'REF''BRANCH' B;
        'REF''NODE' N,M;
        'BOOL' CAUSEFOUND;
        PRINT((NEWLINE,NEWLINE,NEWLINE));
        'WHILE' AL 'ISNT' ('REF''ALARM''VAL''NIL') 'DO'
        (   CAUSEFOUND ← 'FALSE';
            'CASE' (N,CK) ::= TYPE'OF'AL
            'IN' B ← CAUSES'OF'N;
                 'WHILE' B 'ISNT' ('REF''BRANCH''VAL''NIL') 'DO'
                 (  M ← NEXNODE'OF'B;
                    'IF' 'NOT'(NORMAL'OF'M)
```

```
   'THEN' 'IF' INSERT(ENTRY'OF'M,AL)
          'THEN' 'IF' CAUSEFOUND
                 'THEN' SHARED'OF'AL ← 'TRUE'
                 'ELSE' CAUSEFOUND ← 'TRUE'
                 'FI'
          'FI'
   'FI';
   B ← NEXBRAN'OF'B  );
'IF' CAUSEFOUND
'THEN' PRINT((NEWLINE,"CAUSE FOUND FOR  "))
'ELSE' NEXT'OF'LIST ← 'FAULT' ← (AL,NEXT'OF'LIST);
        PRINT((NEWLINE,"NEW CAUSE  ----  "))
'FI';
 TWONLY(N);
 PRINT(("(",UNAME'OF'UNIT'OF'STAND'OF'NAVAL'OF'N,")"));
AL ← NEXT'OF'AL;
NEXT'OF'ENTRY'OF'N ← 'NIL',
F ← LIST;
'IF' CAUSE'OF'F 'IS' ('REF''ALARM''VAL''NIL')
'THEN' CAUSE'OF'F ← AL;
       AL ← NEXT'OF'AL;
       NEXT'OF'CAUSE'OF'F ← 'NIL'
'ELSE' DED ← 'ALARM' ← (CAUSE'OF'F,N,0,'TRUE','FALSE',"SKIP");
       'WHILE' NEXT'OF'DED'ISNT'('REF''ALARM''VAL''NIL') 'DO'
       (   'CASE' (N,CD) ::= TYPE'OF'NEXT'OF'DED
           'IN' 'SKIP',
                'IF' CD 'IS' ('REF''CHECK''VAL'CK)
                'THEN' AL ← NEXT'OF'AL;
                        'GOTO' LEND
                'ELSE' DED ← NEXT'OF'DED
                'FI'
           'ESAC' );
       NEXT'OF'DED ← AL;
```

LISTING 3.13 (contd.)

```
                                AL ← NEXT'OF'AL;
                                NEXT'OF'NEXT'OF'DED ← 'NIL';
                                LEND;'SKIP'
                           'FI'
                     'ESAC'    );
         PRINT((NEWLINE,NEWLINE,"END OF ANALYSE",NEWLINE,NEWLINE))
'END';
'C'-------------------------------------------------------------------------------'C'
'PROC' ONELINE = ('REF''NAMEVAL' VAR,'INT' COUNT,'REAL' TIME):
'BEGIN'
         'REF''STANDARDS' S ← STAND'OF'VAR;
         'INT' I ← 'ROUND''CASE' TYPE'OF'S - 5
                                 'IN'  PRINT("R ");2.0*CHECK6(S,VAR,COUNT,TIME),
                                       PRINT("  ");2.0*CHECK7(S,VAR,COUNT,TIME)
                                 'OUT' PRINT("M ");4.0*CHECK5(S,VAR,COUNT,TIME)
                                 'ESAC';
         'CASE' I+3
         'IN' PRINT("VL"),
              PRINT(" L"),
              PRINT("OK"),
              PRINT(" H"),
              PRINT("VH")
         'ESAC';
         PRINT((SPACE,ME'OF'S))
'END';
'C'-------------------------------------------------------------------------------'C'
'PROC' PRINTRESULTS = ('REF''FAULT' LIST,'REF''NODE' NL,'REAL' TIME):
'BEGIN'
         'INT' KEEPAFTER ← AFTER'OF'NUMBERSTYLE;
         AFTER'OF'NUMBERSTYLE ← 4;
         'REF''FAULT' F ← LIST;
         'REF''NODE'N;
         'REF''CHECK'CK;
```

```
        'REF''STANDARDS' S;
        'REF''ALARM' A;
        'REF''NAMEVAL' VAR;
        'WHILE' F 'ISNT' ('REF''FAULT''VAL''NIL') 'DO'
        (    PRINT((NEWLINE,NEWLINE));
             A ← CAUSE'OF'F;
             'WHILE' A 'ISNT' ('REF''ALARM''VAL''NIL') 'DO'
             (    'IF' ACTIVE'OF'A
                  'THEN' PRINT((NEWLINE,"*"))
                  'ELSE' PRINT((NEWLINE," "))
                  'FI';
                  'CASE' (N,CK) ::= TYPE'OF'A
                  'IN' VAR ← NAVAL'OF'N;
                       S ← STAND'OF'VAR;
                        TWONLY(N);
                        PRINT(("(",UNAME'OF'UNIT'OF'S,") ",
                              VALUE'OF'VAR,NORM'OF'S,"  "));ONELINE(VAR,0,TIME),
                        PRINT((NEWLINE,MESSAGE'OF'CK," --- DEDUCED ALARM "))
                  'ESAC';
                  A ← NEXT'OF'A );
             F ← NEXT'OF'F  )
'END';
'C'------------------------------------------------------------------------'C'
'PROC' AMEND = ('REF''REF''FAULT' LIST,'REAL' TIME);
'BEGIN'
        'REF''CHECKLIST' C;
        'REF''FAULT' F ← LIST;
        'REF''ALARM' A,H;
        'REF''NODE' N;
        'REF''CHECK' CK;
        'REF''BRANCH' B;
        'BOOL' CHANGED;
        H ← A ← 'ALARM' ← (CAUSE'OF'F,N,0,'FALSE','FALSE',"SKIP");
```

LISTING 3.13 (contd.)

```
'WHILE' NEXT'OF'A 'ISNT' ('REF''ALARM''VAL''NIL') 'DO'
(    'CASE' (N,CK) ::= TYPE'OF'NEXT'OF'A
     'IN' A ← NEXT'OF'A,
          C ← NEXCH'OF'CK;
          'WHILE' C 'ISNT' ('REF''CHECKLIST''VAL''NIL') 'DO'
          (    'IF' CONDITION(C,TIME)
               'THEN' C ← NEXCH'OF'C
               'ELSE' NEXT'OF'A ← NEXT'OF'NEXT'OF'A;
                      'GOTO' LABEL
               'FI' );
          A ← NEXT'OF'A;
          LABEL:'SKIP'
     'ESAC'  );
CAUSE'OF'F ← NEXT'OF'M;
'WHILE' NEXT'OF'F'ISNT'('REF''FAULT''VAL''NIL') 'DO'
(   A ← CAUSE'OF'NEXT'OF'F;
    M ← 'NIL';
    'WHILE' A 'ISNT' ('REF''ALARM''VAL''NIL') 'DO'
    (    'IF' ACTIVE'OF'A
         'THEN' M ← A
         'FI';
         A ← NEXT'OF'A  );
    'IF' M 'IS' ('REF''ALARM''VAL''NIL')
    'THEN' A ← CAUSE'OF'NEXT'OF'F
    'ELSE' A ← NEXT'OF'M
    'FI';
    'WHILE' A 'ISNT' ('REF''ALARM''VAL''NIL') 'DO'
    (   'CASE' (N,CK) ::= TYPE'OF'A
        'IN' ENTRY'OF'N ← 'NIL',
             'SKIP'
      'ESAC';
        A ← NEXT'OF'A  );
    'IF' M'IS' ('REF''ALARM''VAL''NIL')
```

LISTING 3.13 (contd)

```
    'THEN' NEXT'OF'F ← NEXT'OF'NEXT'OF'F
    'ELSE' NEXT'OF'M ← 'NIL';
           F ← NEXT'OF'F
    'FI'  );
F ← LIST;
'WHILE' NEXT'OF'F 'ISNT' ('REF''FAULT''VAL''NIL') 'DO'
(    A ← CAUSE'OF'NEXT'OF'F;
     CHANGED ← 'FALSE';
     'CASE' (N,CK) ::= TYPE'OF'A
     'IN' B ← CAUSES'OF'N;
          'WHILE' B 'ISNT' ('REF''BRANCH''VAL''NIL') 'DO'
          (    'IF' 'NOT'(NORMAL'OF'NEXNODE'OF'B)
               'THEN' 'WHILE' A 'ISNT' ('REF''ALARM''VAL''NIL') 'DO'
                      (  'IF' A 'IS'('REF''ALARM''VAL'
                                               ENTRY'OF'NEXNODE'OF'B)
                         'THEN' 'GOTO' RESET
                         'ELSE' A ← NEXT'OF'A
                         'FI' );
                      A ← CAUSE'OF'NEXT'OF'F;
                     PRINT((NEWLINE,"MISSING EFFECT FOUND "));
                     TWONLY(NEXNODE'OF'B);
                     TWONLY(N);
                     PRINT(NEWLINE);
                      CHANGED ← 'TRUE';
                      M ← ENTRY'OF'NEXNODE'OF'B;
                      'WHILE'NEXT'OF'M'ISNT'('REF''ALARM''VAL''NIL')'DO'
                      M ← NEXT'OF'M;
                      NEXT'OF'M ← A;
                      RESET:A ← CAUSE'OF'NEXT'OF'F
               'FI';
               B ← NEXBRAN'OF'B  ),
          'SKIP'
     'ESAC';
```

LISTING 3.13 (contd.)

```
         'IF' CHANGED
         'THEN' NEXT'OF'F ← NEXT'OF'NEXT'OF'F
         'ELSE' F ← NEXT'OF'F
         'FI'  )
'END';
'C'-----------------------------------------------------------------------------'C'
'PROC' TESTREE = ('REF''NODE'NL,'REF''UNIT'ULIST,'INT'NO):
'BEGIN'
     'REF''FAULT' LIST ← 'FAULT' ← ('NIL','NIL');
     'REAL' TMAX ← 0;
     'REF''FAULT' C ← 'FAULT' ← ('NIL','NIL');
     NEXT'OF'C ← 'FAULT' ← ('NIL','FAULT' ← ('NIL',C));
     'INT' METHODNO ← 1,PRINTNO ← 50,COLS;
     'REF''NAMEVAL' N;
     SEARCHFORVAR("HH  ",N,ULIST,'TRUE');
     'REF''REAL' H = VALUE'OF'N;
     H ← SMALLSTEP;
     SEARCHFORVAR("GT  ",N,ULIST,'TRUE');
     'REF''REAL' TIME = VALUE'OF'N;
     TIME ← 0.0;
     'REF''SYSLIST' S ← 'NIL';
     'REF''ALG' ALG ← 'NIL';
     'REF''SAVE' SET ← 'NIL';
     'REF''UNIT' U ← ULIST;
     'WHILE' U 'ISNT' ('REF''UNIT''VAL''NIL') 'DO'
     (   S ← 'SYSLIST' ← (U,'NIL','FALSE',S);
         U ← NEXU'OF'U    );
     SPLITUP(S,SET,ALG);
     'FOR' J 'TO' NO 'DO'
     (   TMAX ← TMAX + 20.0*H;
         MASTERINTEG(SET,ALG,TMAX,H,TIME,METHODNO,PRINTNO,COLS);
         CAUSE'OF'C ← SCAN(NL,TIME);
         C ← NEXT'OF'C;
```

```
                    ANALYSE(LIST,CAUSE'OF'C,NL);
                    AMEND(LIST,TIME);
                    PRINTRESULTS(LIST,NL,TIME)    )
'END';
'SKIP'
'END'
'KEEP' TESTREE
'FINISH'
```

LISTING 3.13 (contd.)

LISTING 3.14

Segment PSEG

```
'BEGIN'
'PROC' PRINTSYMPTOM = ('REF''EFFECT' EFLIST);
'BEGIN'
          'INT' KEEPAFTER ← AFTER'OF' NUMBERSTYLE;
          'REF''EFFECT' E ← EFLIST;
          'REF''CHECK' C;
          'REF''CHECKLIST' CL;
          'REF''NAMEVAL' N;
          'REF''STANDARDS' S;
          AFTER'OF'NUMBERSTYLE ← 5;
          'WHILE' E 'ISNT' ('REF''EFFECT''VAL''NIL') 'DO'
          (    C ← CHECK'OF'E;
               PRINT((NEWLINE,NEWLINE,MESSAGE'OF'C,NEWLINE));
               CL ← NEXCH'OF'C;
               'WHILE' CL 'ISNT' ('REF''CHECKLIST''VAL''NIL') 'DO'
               ( N ← NAVAL'OF'CL;
                 S ← STAND'OF'N;
                 PRINT((NEWLINE,UNAME'OF'UNIT'OF'S,SPACE,NAME'OF'N,
                       SPACE,COUNT'OF'CL," BEFORE ",TIME'OF'CL));
                'CASE' TYPE'OF'S
                'IN' PRINT("  ABSOLUTE   "),
                     PRINT("  RATE       "),
                    PRINT("  FULL-SCALE ")
                'OUT'PRINT("  DEVIATION  ")
                'ESAC';
               PRINT((ME'OF'S,NEWLINE));
                PRINT((A'OF'S,B'OF'S,C'OF'S,D'OF'S,NORM'OF'S));
                 CL ← NEXCH'OF'CL );
                E ← NEXEF'OF'E  );
          PRINT(NEWLINE);
          AFTER'OF'NUMBERSTYLE ← KEEPAFTER
'END';
```

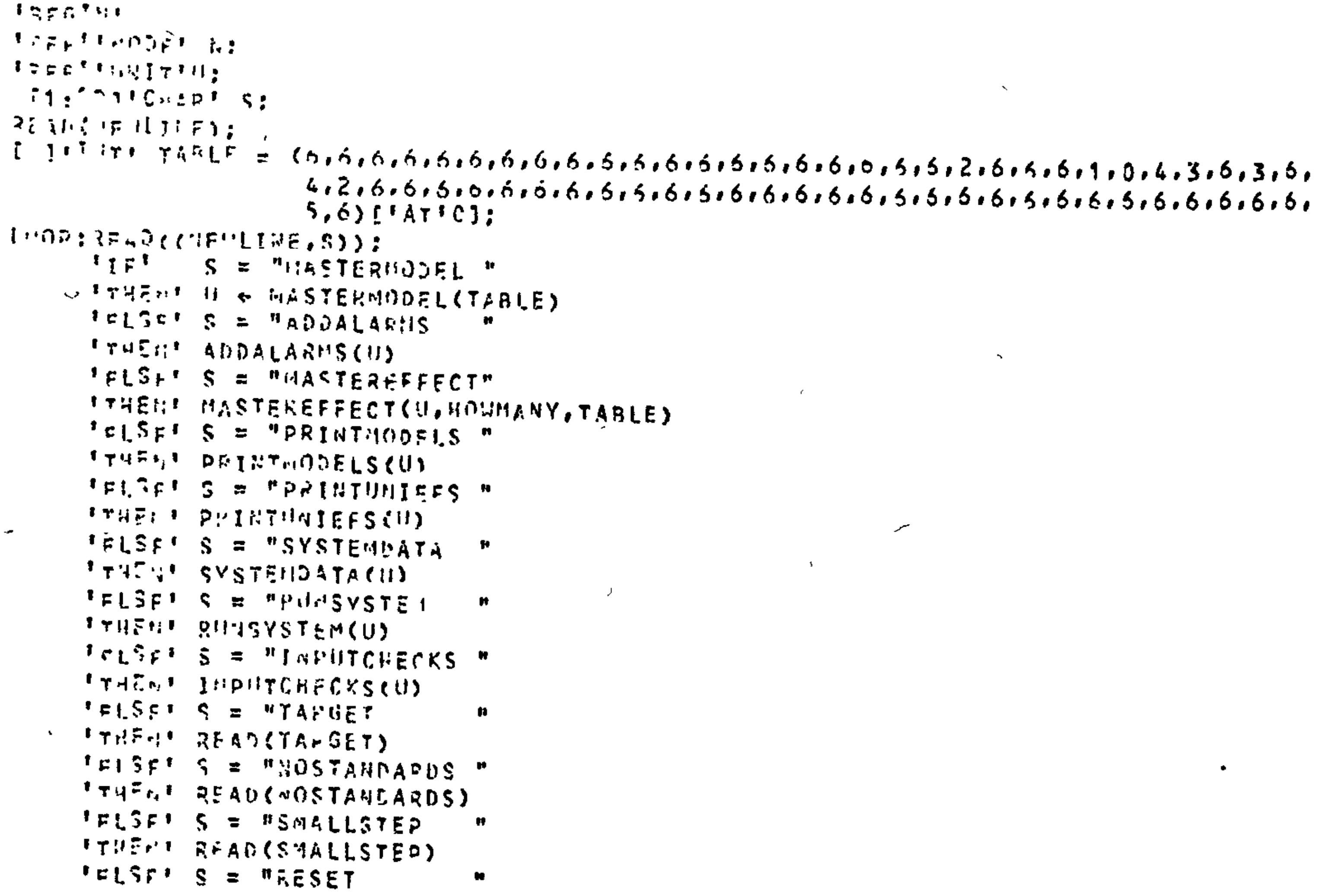
```
'C'------------------------------------------------------------------------------'C'
'PROC' PRINTUNIEFS = ('REF''UNIT' ULIST);
'BEGIN'
        'REF''UNIT' U ← ULIST;
        'REF''EXPRESSION' E;
        'REF''STANDARDS' S;
        'REF''NAMEVAL' N;
        PRINT((NEWLINE,NEWLINE,"SYMPTOM LISTS",NEWLINE,"*************"));
        'WHILE' U 'ISNT' ('REF''UNIT''VAL''NIL') 'DO'
        (     PRINT((NEWLINE,NEWLINE,UNAME'OF'U,NEWLINE,"*****",NEWLINE));
              E ← RELATIONS'OF'U;
              'WHILE' E 'ISNT' ('REF''EXPRESSION''VAL''NIL') 'DO'
              (     N ← LHS'OF'E;
                    S ← STAND'OF'N;
                    'IF' S 'ISNT' ('REF''STANDARDS''VAL''NIL')
                    'THEN' 'IF' UNIT'OF'S'IS'('REF''UNIT''VAL'U)
                           'THEN' PRINT((NEWLINE,NAME'OF'N,NEWLINE));
                                  PRINTSYMPTOM(LOEFLIST'OF'S);
                                  PRINTSYMPTOM(HIEFLIST'OF'S)
                           'ELSE' PRINT((NEWLINE,NAME'OF'N," - OTHER UNIT"))
                           'FI'
                    'FI';
                    E ← NEXTEX'OF'E  );
              U ← NEXU'OF'U  )
'END';
'C'------------------------------------------------------------------------------'C'
'PROC' PRINTCONS = ('REF''CONSTANTS' CON,'BYTES' BITE);
'BEGIN'
        'REF''CONSTANTS' C ← CON;
        PRINT((NEWLINE,NEWLINE));
        'WHILE' C 'ISNT' ('REF''CONSTANTS''VAL''NIL') 'DO'
        (     PRINT((LOOKAT,NAME'OF'NAVAL'OF'C,SPACE,
                            VALUE'OF'NAVAL'OF'C,SPACE));
```

LISTING 3.14 (contd.)

LISTING 3.14 (contd.)

```
                C ← NEXCON'OF'C       )
'END';
'C'------------------------------------------------------------------------'C'
'PROC' PRINTPOINTS = ('REF''UNIT' U);
'BEGIN'
        'FOR' J 'TO' 2 'DO'
        (   'REF''POINTER' P←'CASE' J
                                  'IN'PRINT((NEWLINE,NEWLINE,"INPUTS"));INLIST'OF'U,
                                      PRINT((NEWLINE,NEWLINE,"OUTPUTS"));OUTLIST'OF'U
                                  'ESAC';
            'WHILE' P 'ISNT' ('REF''POINTER''VAL''NIL') 'DO'
            (   PRINT((NEWLINE,NEWLINE,NAME'OF'P," ",LIST'OF'P," ",
                                                            CAR'OF'P,NEWLINE));
                'FOR' I 'TO' 6 'DO'
                (   'REF''NAMEVAL' N ←(VARI'OF'P)[I];
                    'IF' N 'ISNT' ('REF''NAMEVAL''VAL''NIL')
                    'THEN' PRINT((NEWLINE,NAME'OF'N,SPACE,VALUE'OF'N));
                           (   'REF''STANDARDS' S ← STAND'OF'N;
                               'IF' S 'ISNT' ('REF''STANDARDS''VAL''NIL')
                               'THEN' PRINT((SPACE,NORM'OF'S,SPACE,ME'OF'S))
                               'FI'  )
                    'FI'  );
                P ← NEXP'OF'P ) );
        PRINT((NEWLINE,NEWLINE))
'END';
'C'------------------------------------------------------------------------'C'
'PROC' PRINTU = ('REF''UNIT' UNIT);
'BEGIN'
        'REF''EXPRESSION' E ← RELATIONS'OF'UNIT;
        'BYTES' N;
        'REF''CONLIST' CLIST;
        'BOOL' B ← 'TRUE';
        'BYTES' NAME ← UNAME 'OF' UNIT;
        PRINT((NEWLINE,NEWLINE,"MODEL OF UNIT ", NAME,
                               NEWLINE,"******************"));
        PRINTPOINTS(UNIT);
        PRINT((NEWLINE,NEWLINE));
        PRINTCONS(CON'OF'UNIT,NAME);
        CLIST ← COMMONS'OF'UNIT;
        'WHILE' CLIST 'ISNT' ('REF''CONLIST''VAL''NIL') 'DO'
        (     PRINTCONS(CON'OF'CLIST,NAME);
              CLIST ← NEXLIST'OF'CLIST  );
              PRINT((NEWLINE,NEWLINE))
'END';
'C'---------------------------------------------------------------------------'C'
'PROC' PRINTMODELS = ('REF''UNIT' ULIST):
'BEGIN'
        'REF''UNIT' U ← ULIST;
        PRINT((NEWLINE,NEWLINE,NEWLINE,"PRINTMODELS OUTPUT",
                                 NEWLINE,"*******************",NEWLINE));
        'WHILE' U 'ISNT' ('REF''UNIT''VAL''NIL') 'DO'
        (     PRINTU(U);
              U ← NEXU'OF'U  )
'END';
'SKIP'
'END'
'KEEP' PRINTMODELS,PRINTSYMPTOM,PRINTUNIEFS
'FINISH'
```

LISTING 3.14 (contd.)



```
'BEGIN'
'REF''MODEL' M;
'REF''UNIT'U;
 [1:20]'CHAR' S;
READ(FILE);
[ ]'INT' TABLE = (6,6,6,6,6,6,6,6,6.5,6,6,6,6,6,6.6,0,5,5,2,6,6,6,1,0,4,3,6,3,6,
                  4,2,6,6,6,0,6,6,6,6,5,6,5,6,6,6,6,6,5,5,6,6,5,6,6,5,6,6,6,6,6,
                  5,6)['AT'C];
LOOP:READ((NEWLINE,S));
     'IF'    S = "MASTERMODEL "
     'THEN'  U ← MASTERMODEL(TABLE)
     'ELSE'  S = "ADDALARMS    "
     'THEN'  ADDALARMS(U)
     'ELSE'  S = "MASTEREFFECT"
     'THEN'  MASTEREFFECT(U,HOWMANY,TABLE)
     'ELSE'  S = "PRINTMODELS "
     'THEN'  PRINTMODELS(U)
     'ELSE'  S = "PRINTUNIEFS "
     'THEN'  PRINTUNIEFS(U)
     'ELSE'  S = "SYSTEMDATA  "
     'THEN'  SYSTEMDATA(U)
     'ELSE'  S = "RUNSYSTEM   "
     'THEN'  RUNSYSTEM(U)
     'ELSE'  S = "INPUTCHECKS "
     'THEN'  INPUTCHECKS(U)
     'ELSE'  S = "TARGET      "
     'THEN'  READ(TARGET)
     'ELSE'  S = "NOSTANDARDS "
     'THEN'  READ(NOSTANDARDS)
     'ELSE'  S = "SMALLSTEP   "
     'THEN'  READ(SMALLSTEP)
     'ELSE'  S = "RESET       "
```

LISTING 3.15

Example of Main Programme Segment

```
      'THEN' READ(RESET)
      'ELSE' S = "MAXLEVEL    "
      'THEN' READ(MAXLEVEL)
      'ELSE' S = "DEBUG       "
      'THEN' READ(DEBUG)
      'ELSE' S = "LOOKATNO    "
      'THEN' READ(LOOKATNO)
      'ELSE' S = "HOWMANY     "
      'THEN' READ(HOWMANY)
      'ELSE' S = "NO          "
      'THEN' READ(NO)
      'ELSE' S = "PRINTNO     "
      'THEN' READ(PRINTNO)
      'ELSE' S = "TOLERANCE   "
      'THEN' READ(TOLERANCE)
      'ELSE' S = "MOREUNITS   "
      'THEN' READ(MOREUNITS)
      'ELSE' S = "ENDOFDATA   "
      'THEN' 'GOTO' LEND
      'ELSE' PRINT((NEWLINE,"***** ERROR IN COMMAND DATA *****"));
             'GOTO' LEND
      'FI';
      'GOTO' LOOP;
LEND:PRINT((NEWLINE,NEWLINE,NEWLINE,"END OF PROGRAM",NEWLINE,
             "**************"))
'END'
'FINISH'
```

LISTING 3.15 (contd.)

APPENDIX 4

FLOWCHARTS OF IMPORTANT PROCEDURES

TABLE OF CONTENTS

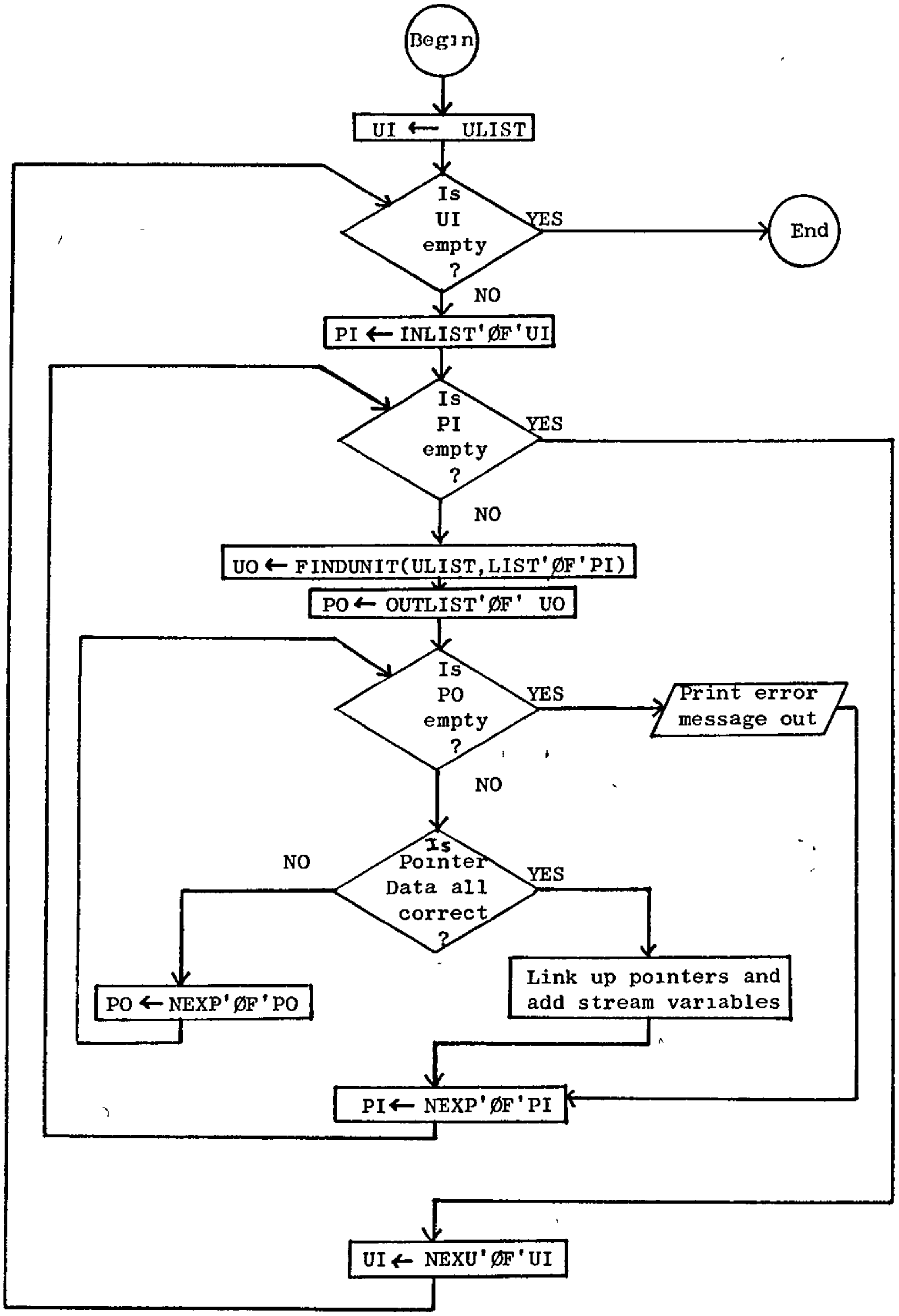


FIG. A.4.1

Flowchart of Procedure LINKUNITS

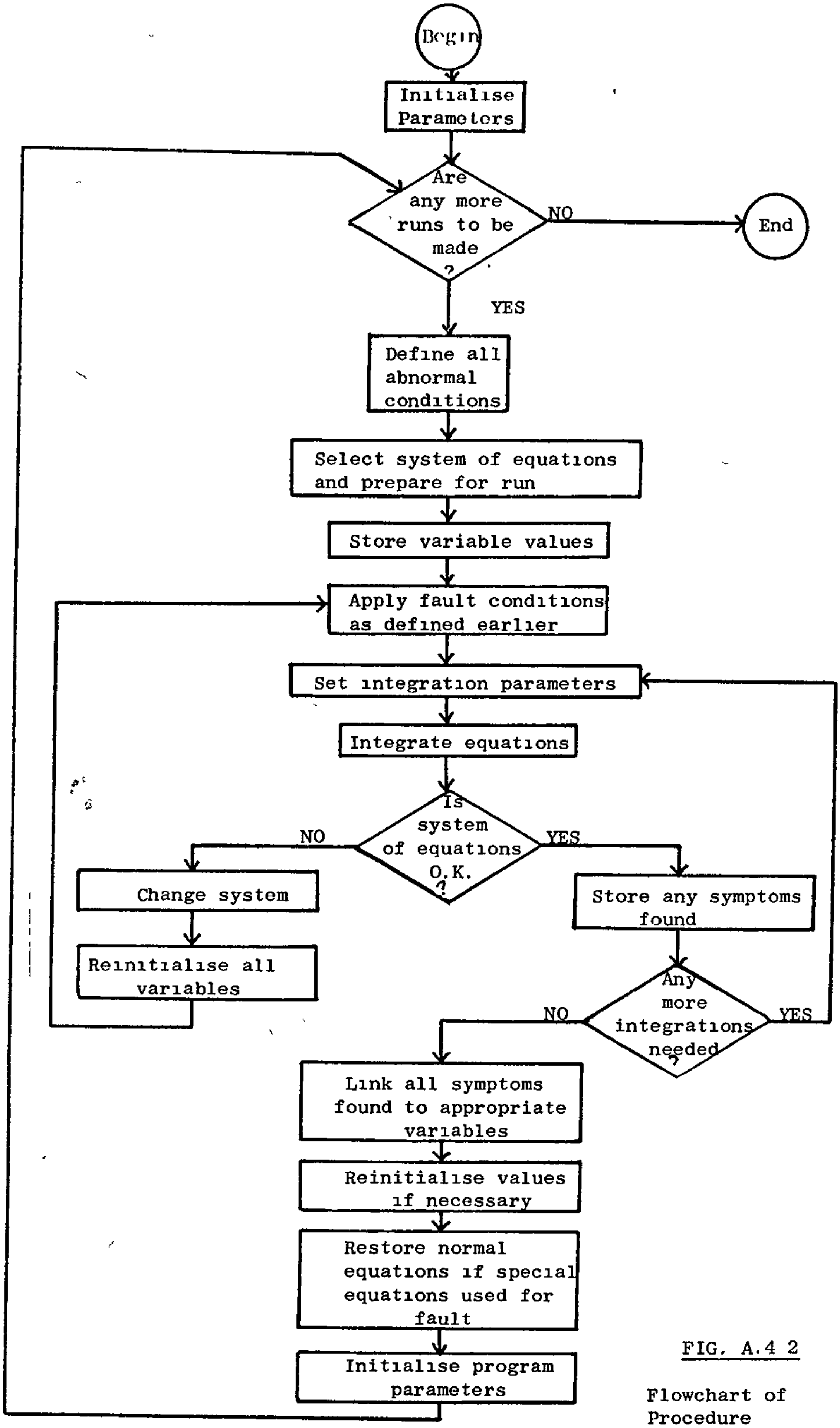

Begin
Initialise Parameters
Are any more runs to be made ?
NO
End
YES
Define all abnormal conditions
Select system of equations and prepare for run
Store variable values
Apply fault conditions as defined earlier
Set integration parameters
Integrate equations
Is system of equations O.K. ?
NO
YES
Change system
Reinitialise all variables
Store any symptoms found
Any more integrations needed ?
NO
YES
Link all symptoms found to appropriate variables
Reinitialise values if necessary
Restore normal equations if special equations used for fault
Initialise program parameters

FIG. A.4 2

Flowchart of Procedure MASTEREFFECT

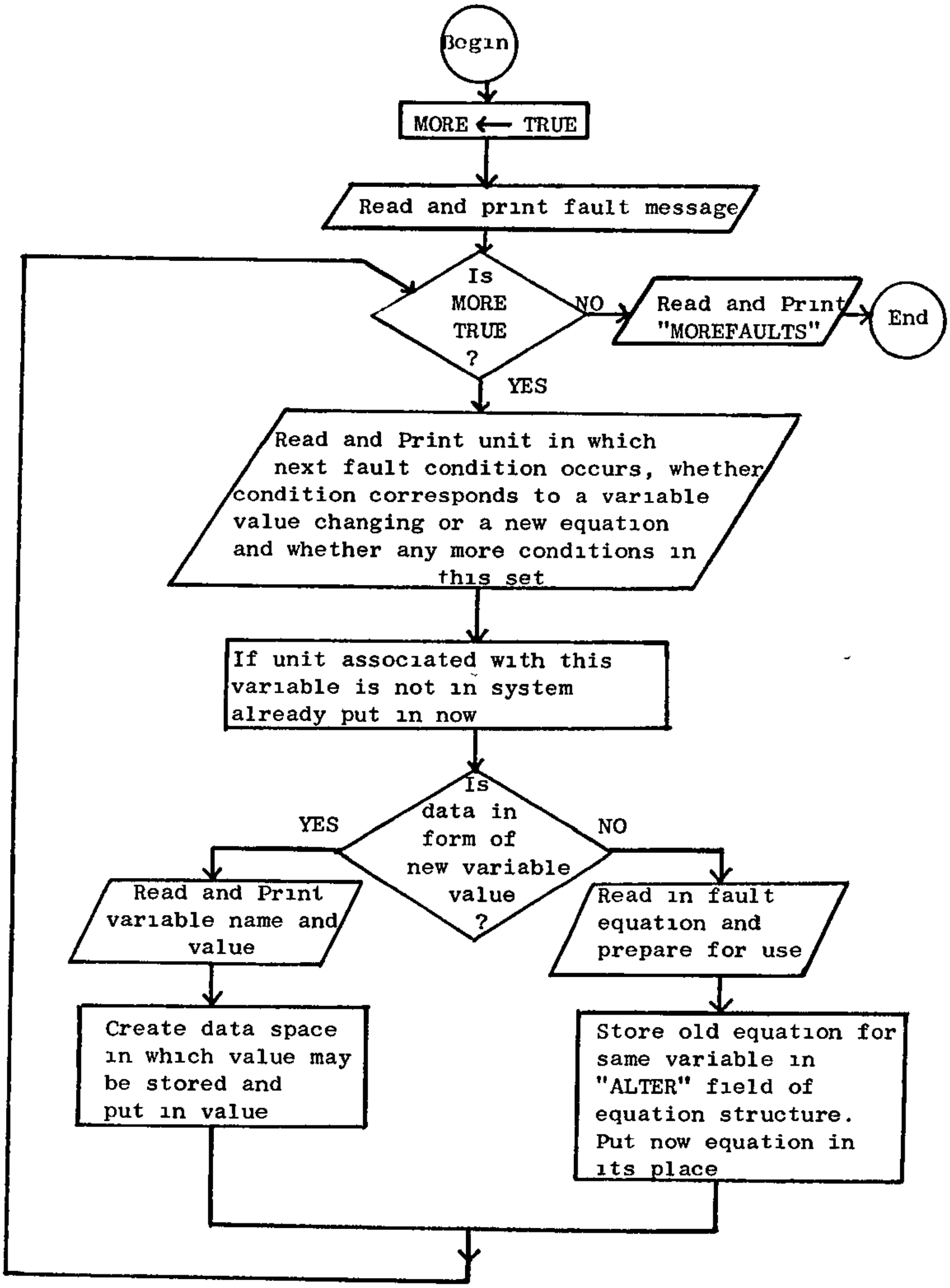

Begin
MORE ← TRUE
Read and print fault message
Is MORE TRUE ?
NO
Read and Print "MOREFAULTS"
End
YES
Read and Print unit in which next fault condition occurs, whether condition corresponds to a variable value changing or a new equation and whether any more conditions in this set
If unit associated with this variable is not in system already put in now
Is data in form of new variable value ?
YES
NO
Read and Print variable name and value
Read in fault equation and prepare for use
Create data space in which value may be stored and put in value
Store old equation for same variable in "ALTER" field of equation structure. Put now equation in its place

FIG. A.4.3

Simple Flowchart of Procedure DEFINEFAULT

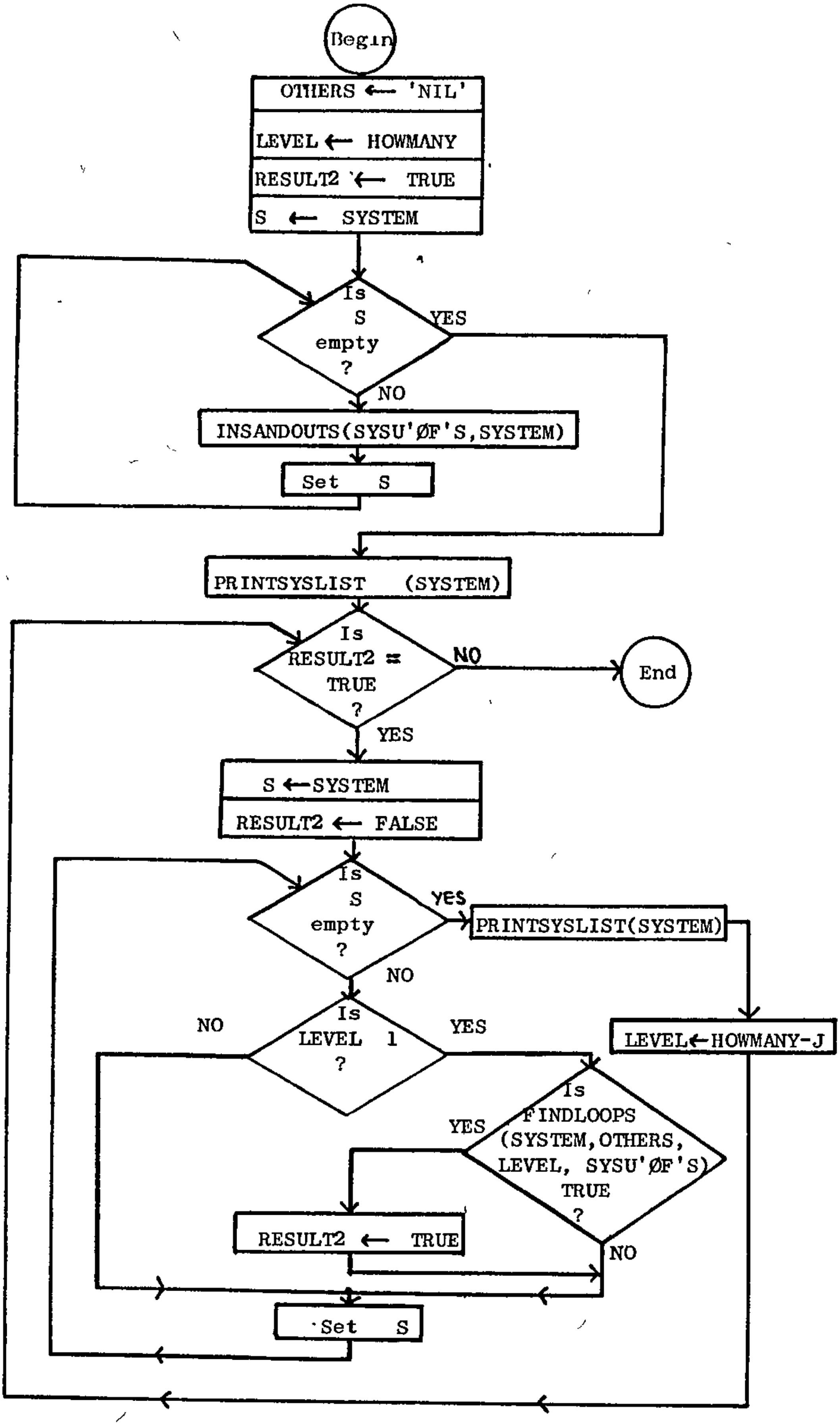


FIG. A.4.4

Flowchard of Procedure SELECTSYSTEM

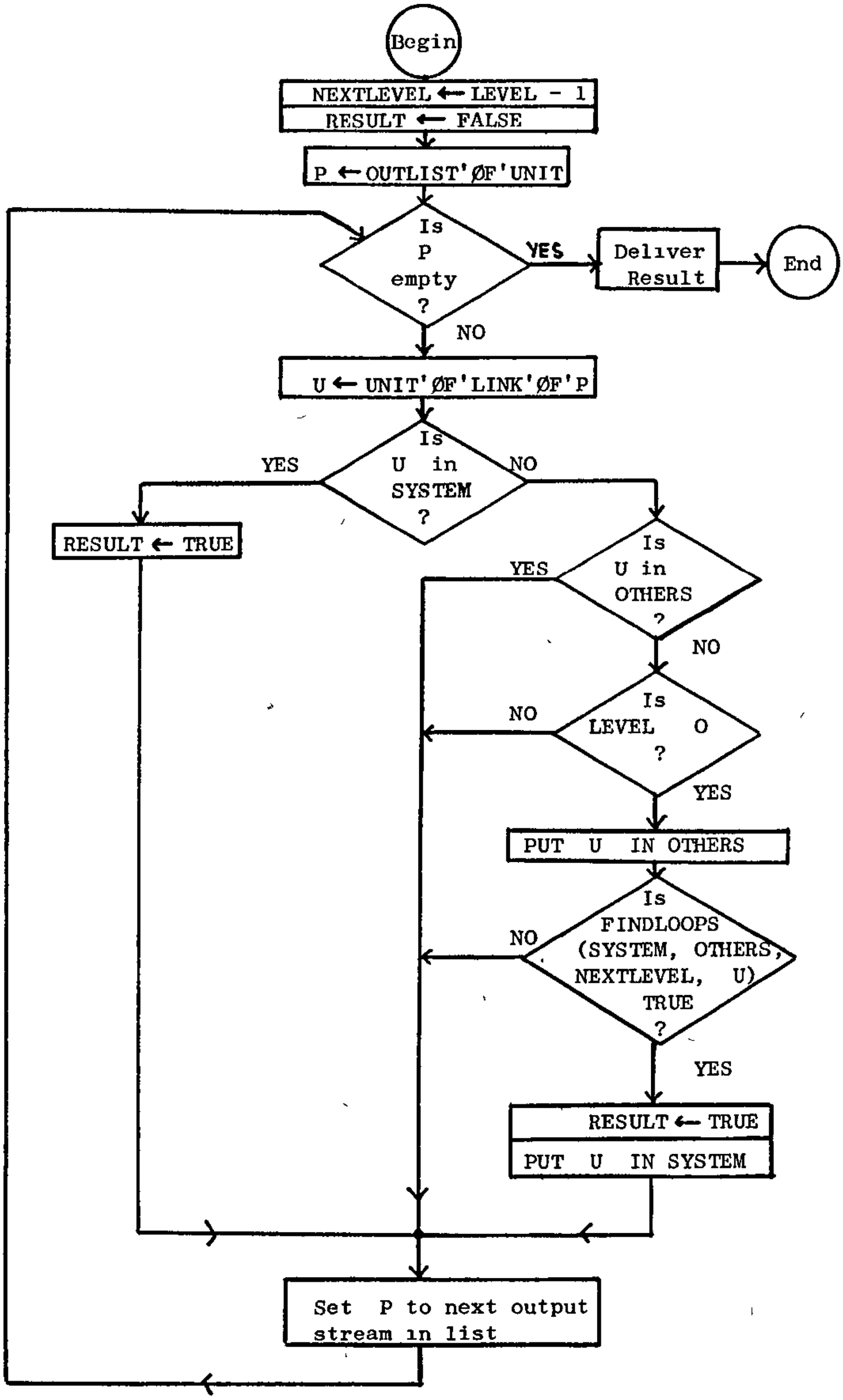


FIG. A.4.5

Flowchart of Procedure FINDLOOPS

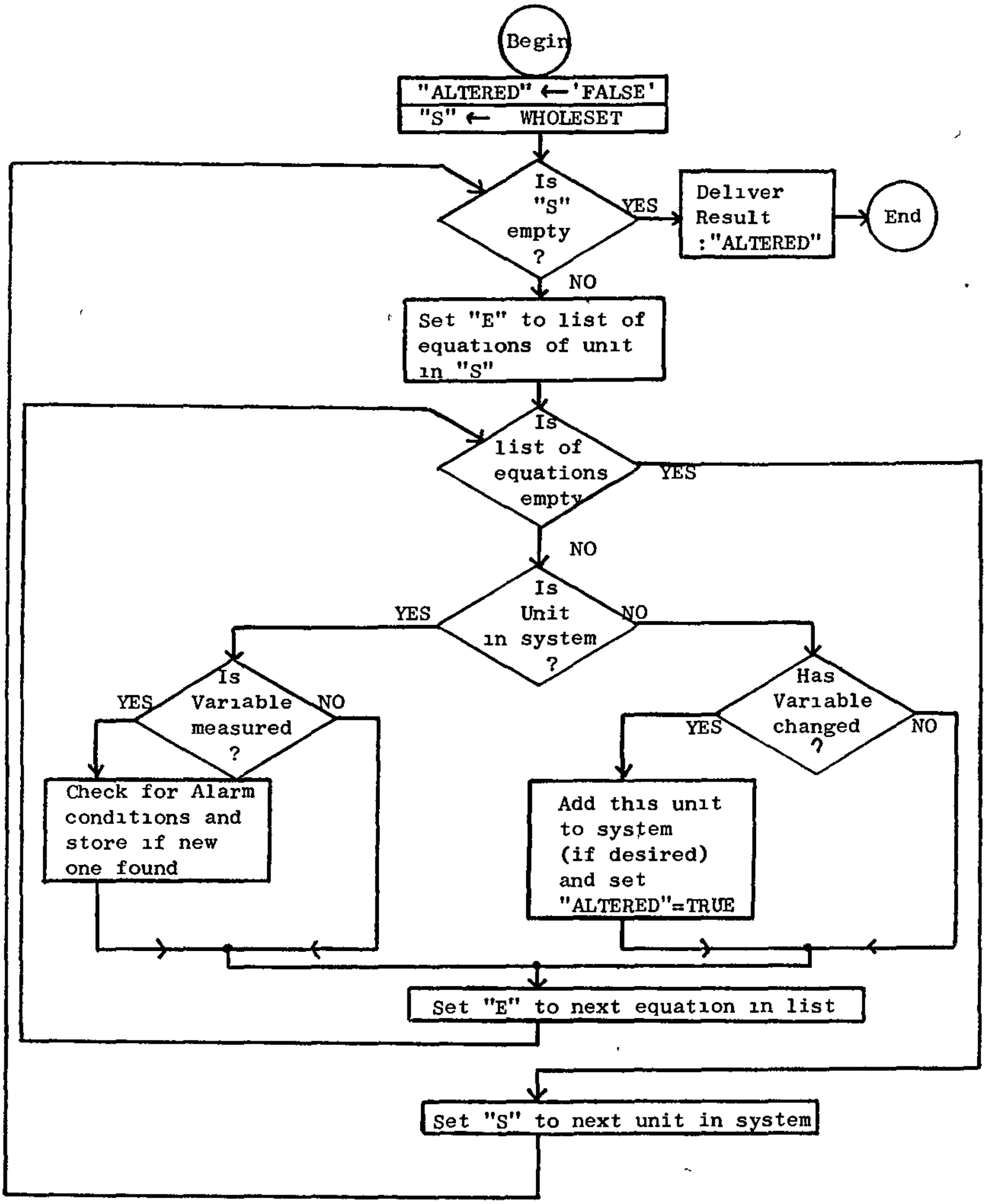

Begin
"ALTERED" ← 'FALSE'
"S" ← WHOLESET
Is "S" empty ?
YES
NO
Deliver Result :"ALTERED"
End
Set "E" to list of equations of unit in "S"
Is list of equations empty
YES
NO
Is Unit in system ?
YES
NO
Is Variable measured ?
YES
NO
Check for Alarm conditions and store if new one found
Has Variable changed ?
YES
NO
Add this unit to system (if desired) and set "ALTERED"=TRUE
Set "E" to next equation in list
Set "S" to next unit in system

FIG. A.4.6

Simple Flowchart of Procedure SYSCHECK

APPENDIX 5

SIMPLE MODELS OF PROCESS EQUIPMENT FOR BOTH NORMAL AND FAULT CONDITIONS

TABLE OF CONTENTS

APPENDIX 5

SIMPLE MODELS OF PROCESS EQUIPMENT FOR BOTH NORMAL AND FAULT CONDITIONS

A.5.1 Introduction

The simple models shown in this section are intended to be representative of the class of models that may be conveniently used for alarm analysis purposes. In order to be consistent with the conventions adopted for the program, the variables in the models will generally consist of 2 characters. The first character will be used to denote the particular quantity considered and the second character will denote the stream or point on the model where the quantity is considered (see Chapter 4). The nomenclature which follows defines the characters used to denote common quantities. Any exceptions to the nomenclature will be defined on the model concerned. A second convention adopted is that all model streams are based on mass or volume units rather than molar units except where the model is marked with an asterisk. Models marked in this way contain notes on the stream units employed. Such models are used if the resulting set of equations is simpler than the corresponding set based on mass or volume units.

The expressions shown are in the form required for the model - no rearrangement or substitution is necessary. In order to comply with the convention adopted for the program, it is assumed that, unless otherwise noted, all stream properties except pressure are set in the unit outputs. Pressure is normally set in the unit input streams. In some cases, the expressions used are only applicable within certain ranges of process variables. In order to allow for these situations, provision is made in the program to put upper and lower limits on variable values.

In order to preserve continuity in process streams passing through several units, high-gain differential equations are used to set intermediate stream pressures. The need for this type of equation arises because conventional equations for liquid flow generally assume an incompressible fluid. If this assumption is not made, the equations become considerably more complicated and it is therefore convenient to use high-gain equations. The need for this type of equation is discussed by Franks(50).

Functional models are not considered explicitly in this Appendix although, as seen in Chapter 6, the conversion of the equations shown in this Appendix to functional form is trivial.

When choosing variables names for use in model equations, the user should avoid the names below which are automatically available to every unit module and are supplied by the program:

Name	Significance	Value
L_T	local time	-
G_T	global time	-
H_H	step length of integration	-
E_X	e	2.71828
K_H	Half	0.5
Z_O	Zero	0.0
W_N	Unity	1.0

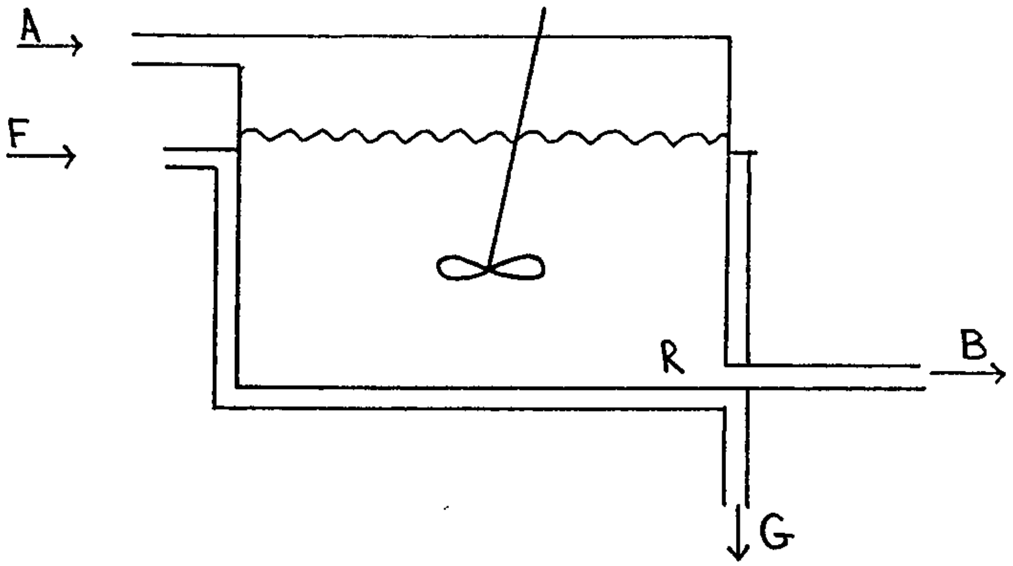
When a stream has been defined for a unit, for example stream A, then QA, XA, YA, TA, PA and ZA are assumed by the program to be the stream properties and will be used as such if encountered in the unit equations.

A.5.2 Standard Models

Nomenclature

Symbol	Quantity	Typical Unit
A	Area	m^2
C	Specific heat	J/kg ^oK
D	Temperature difference	^oK
E	Heat flux	W/m^2
G	Gain constant (for high-gain equations)	-
H	Latent heat	J/kg
K	Constant	-
L	Level	m
M	Mass	kg
P	Pressure	N/m^2
Q	Volumetric flowrate	m^3/s
T	Temperature	^oK
U	Overall heat transfer coefficient	W/m^2 ^oK
V	Volume	m^3
W	Hydraulic delay	s
X	Mass fraction	-
Z	Voltage	V
ρ	Density	kg/m^3

C. S. T. R.



Notes:

1. Lıquıd phase, constant volume reactıon
2. Well-mıxed vessel and coolıng ȷacket
3. Closed vessel
4. No wall resıstance to heat transfer
5. No phase change in jacket

Mass balance:

$$A_R \frac{dL_R}{dt} = Q_A - Q_B$$

Component balance:

$$A_R L_R \frac{dX_R}{dt} = Q_A(X_A - X_R) - A_R L_R \; f_1(X_R)$$

Heat balance on reactants:

$$A_R L_R \frac{dT_R}{dt} = Q_A(T_A - T_R) - \frac{\Delta H \; . \; A_R L_R \; f_1(X_R)}{C_p} + \frac{E_c}{C_p \; \rho \; L}$$

where: $f_1(X_R)$ is specified for the particular reaction considered.

Pressure in vapour:

$$P_A = \frac{K_G T_R}{V_M - A_R L_R}$$

where: V_M is the maximum volume of vapour space.

Pressure at exit of reactor:

$$P_R = P_A + K_1 L_R$$

Outlet stream:

$$Q_B = K_B(P_R - P_B)^{\frac{1}{2}}$$

$$T_B = T_R$$

$$X_B = X_R$$

For cooling jacket:

$$\frac{dP_F}{dt} = G_F(Q_F - Q_G)$$

$$T_G = \frac{Q_F \rho_F C_F T_F + E_c}{Q_G \rho_F C_F}$$

$$Q_G = K_G(P_F - P_G)^{\frac{1}{2}}$$

Heat flux to coolant: $E_c = U_c A_c (T_G - T_R)$

Tank

Notes:

1. Perfectly mixed
2. No heat losses
3. Closed vessel
4. No volume changes
5. No phase changes

$$\frac{d(L_c)}{dt} = (Q_A - Q_B)/A_T$$

$$\frac{d(T_c)}{dt} = (T_A - T_C) * Q_A/L_C/A_T$$

$$\frac{d(X_C)}{dt} = Q_A(X_A - X_C) \quad /L_C/A_T$$

$$Q_B = K_B \left[(P_A + K_1\ L_C) - P_B \right]^{\frac{1}{2}}$$

$$T_B = T_C$$

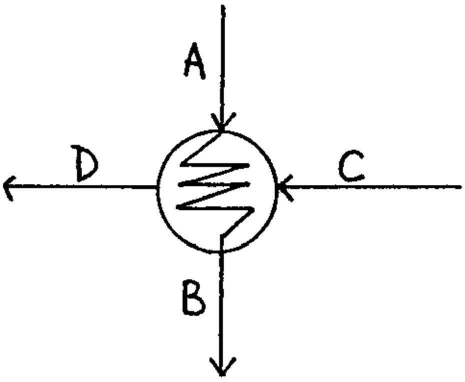
$$X_B = X_C$$

Condenser

(Very simple model)

Notes:

1. No holdup for either stream
2. No temp. change in tube-side stream
3. Tube-side flow enters as vapor and is completely condensed
4. No phase change on shell-side
5. No heat losses
6. No internal resistance to heat transfer



$$Q_B = Q_A$$

$$T_B = T_A$$

$$X_B = X_A$$

$$E_C = H_L Q_A$$

$$\frac{dP_C}{dt} = G_C(Q_C - Q_D)$$

$$Q_D = K_D(P_C - P_D)^{\frac{1}{2}}$$

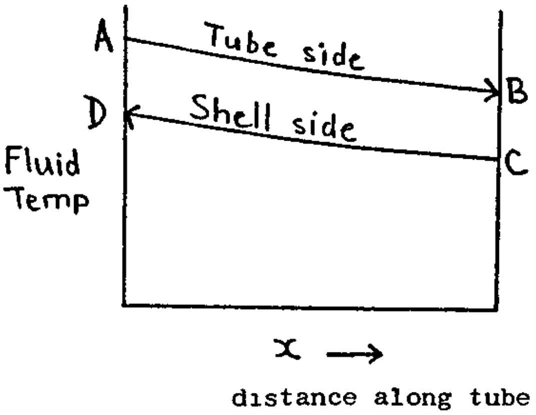
$$T_D = \frac{Q_C \rho_C C_{P_C} T_C + E_C}{Q_D \rho_C C_{P_C}}$$

Heat Exchanger

Notes:

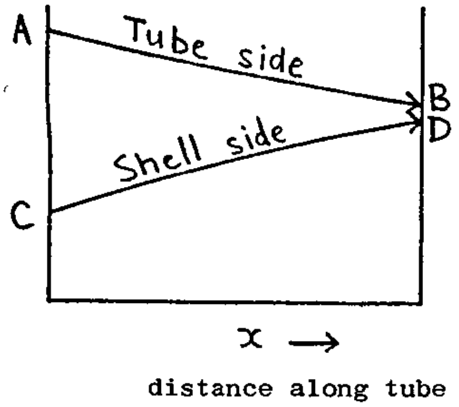
1. Shell and Tube design
2. Shell perfectly mixed
3. Plug flow in tube
4. No phase change
5. No wall resistance
6. No density changes

Counter-current



$$D_M = \frac{(T_A - T_D) - (T_B - T_C)}{\ln\left[\dfrac{T_A - T_D}{T_B - T_C}\right]}$$

Co-current



$$D_M = \frac{(T_A - T_C) - (T_B - T_D)}{\ln\left[\dfrac{T_A - T_C}{T_B - T_D}\right]}$$

$$E_C = A_C U_C D_M$$

$$\frac{dP_A}{dt} = G_A(Q_A - Q_B)$$

$$Q_B = K_B(P_A - P_B)^{\frac{1}{2}}$$

$$\frac{dP_C}{dt} = C_C(P_C - P_D)$$

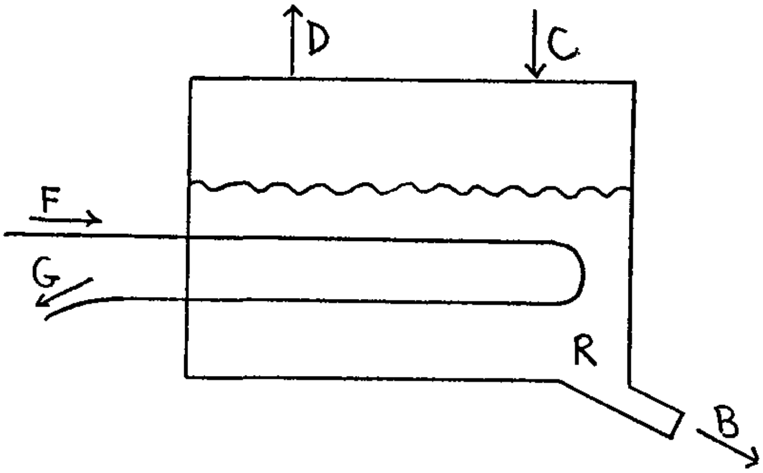
$$Q_D = K_D(P_C - P_D)^{\frac{1}{2}}$$

$$T_B = \frac{Q_A \rho_S C_S T_A - E}{Q_B \rho_S C_S}$$

$$T_D = \frac{Q_C \rho_C C_T T_C + E}{Q_D \rho_T C_T}$$

Reboiler (1)* (or Vaporiser)

Notes:
1. Binary distillation
2. All flows and concentrations in molar units except bottom stream and heating stream (mass units)
3. Perfect mixing in sump
4. Plug flow in heating coil
5. Equilibrium between phases
6. Constant pressure in column
7. No wall resistance for heat flux from heating coil
8. No phase change in heating coil (i.e. hot oil used)



$$\frac{dX_R}{dt} = \left[Q_C(X_C - X_R) - Q_D(X_D - X_R)\right] / H_R$$

$$\frac{d\ H_R}{dt} = (Q_C - Q_D - Q_R)$$

$$L_R = H_R/A_R/\rho_R/V_F$$

$$T_R = f_1\ (X_R)$$

$$Q_R = K_R \left[\ (K_p + K_1\ L_R) - P_B\right]^{\frac{1}{2}}$$

$$E_C = U_C A_C \cdot \frac{(T_F - T_R) - (T_C - T_R)}{\ln\left[\frac{T_F - T_R}{T_C - T_R}\right]}$$

$$X_D = f_2\ (X_R)$$

$$Q_D = E_C/H_L$$

$$T_D = T_R$$

$$\bar{Q}_B = Q_R\ \left[M_1\ X_R + (1 - X_R)\ M_2\right]$$

$$T_B = T_R$$

$$\bar{X}_B = X_R \cdot \frac{M_1}{M_1\ X_R + M_2(1 - X_R)}$$

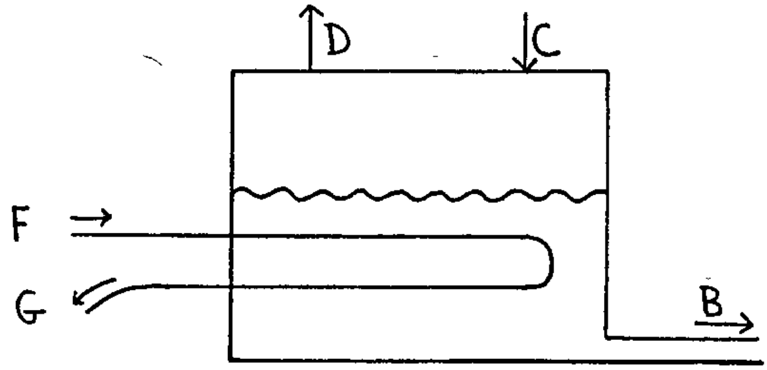
Heating medium side:

$$\frac{dP_F}{dt} = G_F(\bar{Q}_F - \bar{Q}_G)$$

$$\bar{Q}_G = K_G(P_F - P_G)^{\frac{1}{2}}$$

$$T_G = \frac{\bar{Q}_F\ \rho_F\ C_F\ T_F - E_C}{\bar{Q}_G\ \rho_F\ C_F}$$

Reboiler (2)*



Notes:

1. Binary distillation
2. Streams C and D in molar units
3. Perfect mixing assumed
4. Equilibrium between plases
5. Constant pressure in column
6. No wall resistance to heat transfer
7. Heating jacket equipped with condensate trap
8. Vaporisation rate in sump fixed entirely by heat transfer rate from coil
9. Condensed stream at boiling point

For steam in coil:

$$P_F = \frac{K_F M_V T_B}{V_V}$$

$$T_G = f(P_F)$$

$$\frac{d M_G}{dt} = \bar{Q}_A \rho_A - \bar{Q}_L \rho_L$$

M_G is mass of vapour phase

$$E_C = A_C \, U_C(T_G - T_R)$$

$$\bar{Q}_L = \frac{E_C}{L_L \quad {}_L}$$

$$A_J \frac{d\,L_G}{dt} = \bar{Q}_L - \bar{Q}_G$$

$$V_V = V_M - A_J \, L_G$$

where: V_M is the maximum volume

$$P_J = P_F + K_L \, L_G$$

Outflow:

$$\bar{Q}_G = K_V \, A_V(P_J - P_G)^{\frac{1}{2}}$$

where: $A_V = K_B . \text{ØLIM}\left[L_V\right]$ ØLIM is a function that defines the valve area

and: $L_V = L_G - L_S$

In sump:

$$\frac{dX_R}{dt} = \left[Q_C(X_C - X_R) - Q_D(X_D - X_R)\right] / H_R$$

$$\frac{dH_R}{dt} = Q_C - Q_D - Q_R$$

$$L_R = \frac{H_R}{A_R \, \rho_R \, V_F}$$

$$T_R = f_1(X_R)$$

$$Q_R = K_R \left[(K_P + K_1 \; L_R) - P_B \right]^{\frac{1}{2}}$$

For vapour flow:

$$X_D = f_2(X_R)$$

$$Q_D = \frac{E_C}{L_H \; \rho \; H}$$

$$T_D = T_R$$

For bottoms flow:

$$\bar{Q}_B = Q_R \left[M_1 X_R + (1 - X_R) \; M_2 \right]$$

$$T_B = T_R$$

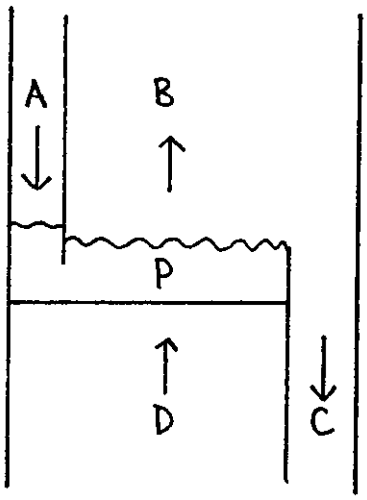
$$\bar{X}_B = X_R \cdot \frac{M_1}{M_1 X_R + M_2 \; (1 - X_R)}$$

where: M_1 and M_2 are the molecular weights of the two components.

Distillation Column Plate*

Notes:

1. Binary Distillation
2. All flows and concentrations in molar units
3. Perfect mixing
4. Equilibrium between phases
5. Constant molar overflow
6. Constant pressure
7. Vapour flow changes transmitted instantaneously



Comments:

T_P and X_P are included if temperature or concentration measurements are required.

$$\frac{dX_P}{dt} = \left[Q_D(X_D - X_C) + Q_A(X_A - X_C) - Q_B(X_B - X_C)\right] / H_T$$

where: H_T is the molar holdup

$$\frac{dQ_C}{dt} = (Q_A - Q_C)/W_L$$

$$T_C = f_1(X_C)$$

$$T_P = T_C$$

$$X_P = X_C$$

$$Q_B = Q_D$$

$$T_B = T_C$$

$$X_B = f_2(X_C)$$

Feed Platc*

Notes: 1. Binary dıstillation

2. Feed stream (F) ın mass units, all others in molar units

3. Feed is luqıd at B. pt.

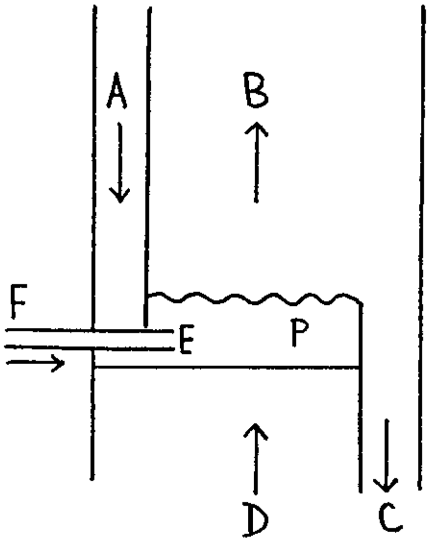
4. Perfect mıxing on plate

5. Equılibrıum between liquid and vapour

6. No heat losses

7. Constant pressure

8. Constant molar overflow



$$\frac{dX_C}{dt} = \Big[Q_D(X_D - X_C) + Q_E(X_E - X_C) - Q_B(X_B - X_C) + Q_A(X_A - X_C)\Big] /H_T$$

where: H_T ıs the molar holdup

$$\frac{dQ_C}{dt} = (Q_A + Q_E - Q_C)/W_L$$

$$T_C = f_1(X_C)$$

$$X_B = f_2(X_C)$$

$$Q_B = Q_D$$

$$T_B = T_C$$

$$X_E = \frac{M_2\bar{X}_F}{M_2\bar{X}_F + M_1(1 - \bar{X}_F)}$$

$$Q_E = \bar{Q}_F\left[\frac{X_F}{M_1} + \frac{(1 - X_F)}{M_2}\right] \qquad P_F = K_p$$

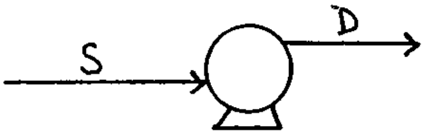
K_p is column pressure

If measurements are required:

$$T_P = T_C$$

$$X_P = X_C$$

Centrifugal Pump



Notes:
1. Isothermal operation assumed
2. Constant rotational speed of impeller.

Basic Relation:

$$N_S = \frac{n_R \; Q}{\left(H\right)^{3/4}}$$

where: N_S is the specific speed of the pump

n_R is the rotation speed

Full equation in terms of discharge flow:

$$Q_D = A_D \left\{ 2 \left[\left(\frac{n_R \; Q_S^{\frac{1}{2}}}{N_S}\right)^{4/3} - K_L(P_D - P_S) \right] + \frac{Q_S^{\;2}}{A_S^{\;2}} \right\}^{\frac{1}{2}}$$

Simplified equation:

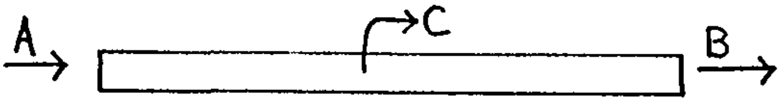
$$Q_D = \left[K_1 \; Q_S^{\;2/3} - K_2(P_D - P_S) + K_3 \; Q_S^{\;2} \right]^{\frac{1}{2}}$$

$$\frac{dP_B}{dt} = K_S(Q_S - Q_D)$$

$$X_D = X_S$$

$$T_D = T_S$$

Flow Meter or Pipe



Notes: 1. Isothermal Flow

2. No density change

3. No stream C for pipe

Outflow:

$$Q_B = K_B(P_A - P_B)^{\frac{1}{2}}$$

Continuity:

$$\frac{dP_A}{dt} = G_A(Q_A - Q_B)$$

Exit temp:

$$T_B = T_A$$

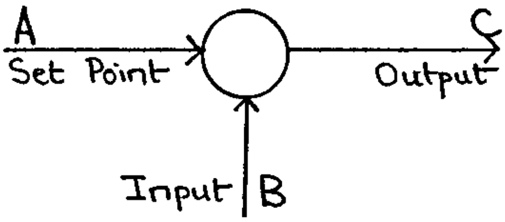
Exit concentration:

$$X_B = X_A$$

Output signal (for flow meter only):

$$Z_C = S_F\, Q_B$$

P - I Controller



Notes:

1. Ideal action - if it is desired to take into account saturation effects, then limits must be specified on the output pressure.
2. Integral action represented by approximation
3. Contrary to convention of Chapter 4, the output pressure is specified

Basic equation: $$P_C = P_O + K_C \left[E + \frac{1}{T_I} \int_o^t E \, dt \right]$$

where the error, E, is defined by:

$$E = Z_A - Z_B$$

and P_O is the steady-state output.

The integral may be approximated by defining a "local time", L_T, for the controller, a "global time", G_T, using the steplength H_H (used for the integration of the differential equations in the model) and storing a running sum, S_U, as the current value of the error integral. The following relations are then used in the order shown:

$$S_U \leftarrow S_U + E(L_T + H_H - G_T)$$

$$L_T \leftarrow G_T + H_H$$

At the start of each step of the integration L_T and G_T have the same value. S_U is incremented by the quantity $E.H_H$ and L_T increased by H_H. On subsequent iterations during the same step of the integration (corresponding to the corrector iterations of the integration program) S_U and L_T will not be altered since G_T is not incremented by the quantity H_H until the end of the step.
G_T is automatically incremented by the program and no expression is therefore needed in any of the unit modules to perform this task.

The model output is given by:

$$P_C = P_O + K_C(E + K_I S_U)$$

where: P_O is the steady-state output.

The function LIM allows the model to incorporate the effect of saturation. If the minimum and maximum controller outputs are P_L and P_H, then the model becomes:

$$E \leftarrow Z_A - Z_B$$

$$S_U \leftarrow S_U + E(L_T + H_H - G_T)$$

$$P_I \leftarrow P_O + K_C(E + K_I S_U)$$

where: P_I is the ideal output pressure

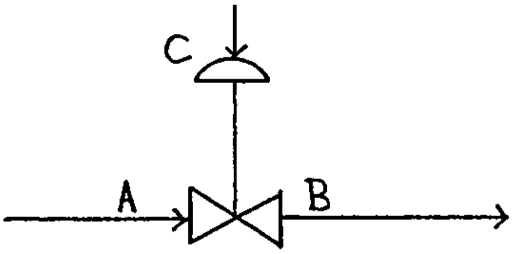
$$P_C \leftarrow P_L + (P_H - P_L) . \text{LIM}(P_I)$$

Function LIM is defined as follows:

Argument X	Value of Function LIM(X)
$X < 0$	0
$0 < X < 1$	$\frac{1}{2}(1 - \cos \pi X)$
$X > 1$	1

This function is available as a standard program facility as described in Section A.2.2.1.

Control Valve



Notes: 1. Isothermal flow through slidıng-steam valve

2. Contrary to conventıon of Chapter 4, pressure in stream C ıs assumed to be fixed at the controller.

$$Q_B = A_V \cdot K_V \left[\frac{P_A - P_B}{\rho} \right]^{\frac{1}{2}}$$

A_V area available for flow

K_V valve constant

where: $A_V = f_1 (P_{BON})$

For quıck actıng valves:

$$P_{BON} = P_C$$

P_{BON} is bonnet pressure

For velocıty lımited valves:

$$\frac{dA_V}{dt} < V_{MAX}$$

V_{MAX} maximum valve velocıty

$$K_V{}' = \frac{K_V}{\frac{1}{2}}$$

For large valves, the change in bonnet pressure with respect to controller output, P_C, may be 1st (or higher) order:

$$T_V \frac{dP_{BON}}{dt} = P_C - P_{BON}$$

T_V valve bonnet time constant

Full equations for general use:

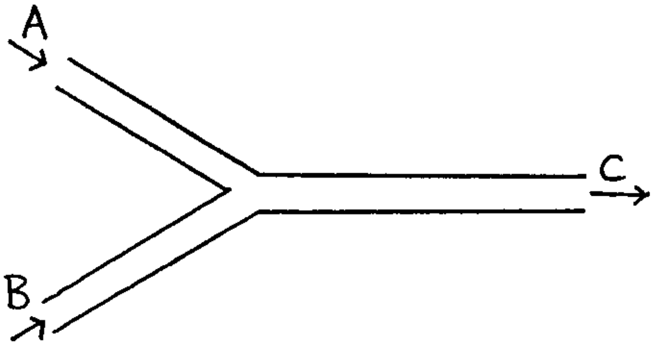
$$Q_B = K_V{}' f_1(P_C)(P_A - P_B)^{\frac{1}{2}}$$

$$\frac{dP_A}{dt} = G_A(Q_A - Q_B)$$

$$T_B = T_A$$

$$X_B = X_A$$

Stroam Mixor



Notes:

1. Identical stream properties (C_P , ρ)
2. No heat of mixing
3. No heat losses
4. No holdup
5. No volume charge

$$X_C = \frac{Q_A X_A + Q_B X_B}{Q_A + Q_B}$$

$$Q_C = K_A(P_A - P_C)^{\frac{1}{2}}$$

$$\frac{dP_A}{dt} = G_A(Q_A + Q_B - Q_C)$$

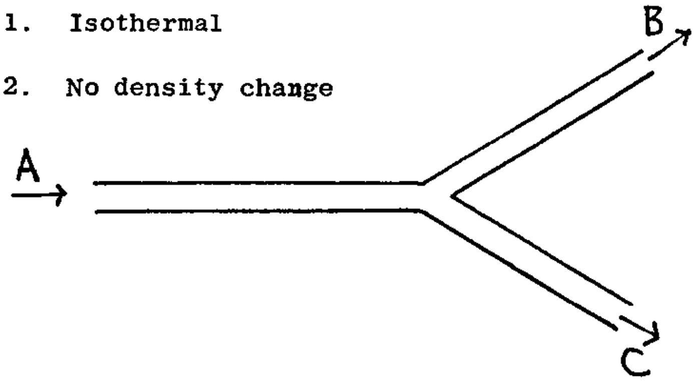
$$P_B = P_A$$

$$T_C = \frac{Q_B T_B + Q_A T_A}{Q_A + Q_B}$$

Stream Divider

Notes: 1. Isothermal

2. No density change



$$Q_B = K_B(P_A - P_B)^{\frac{1}{2}}$$

$$Q_C = K_C(P_A - P_C)^{\frac{1}{2}}$$

$$\frac{dP_A}{dt} = G_A(Q_A - Q_B - Q_C)$$

$$T_C = T_A$$

$$T_B = T_A$$

$$X_B = X_A$$

$$X_C = X_A$$

A.5.3 Representation of Process Plant Faults

As noted in Section 5.3.3, the faults to be considered are represented by changes in variable values or equations or combinations of both. This section will be concerned with demonstrating how the models used for the present study may be changed in order to simulate simple fault conditions.

The example to be used is a control valve. The "normal" equations used are:

Outflow: $Q_B = K_V P_C (P_A - P_B)^{\frac{1}{2}}$

Continuity: $\dfrac{dP_A}{dt} = G_A (Q_A - Q_B)$

Temperature: $T_B = T_A$

Concentration: $X_B = X_A$

For a valve jammed shut, the outflow is zero and therefore a new equation may be used:

$$Q_B = Z_O$$

where: Z_O has the value 0.0.

This expression temporarily replaces that normally used to define the outflow. The use of keyword MASTEREFFECT, described in Chapter 5 and Appendix 2, Section 2.4.1, allows the user to input an expression to replace the normal one. The program automatically replaces the old expression in the model and stores the old expression for automatic reinsertion after the current fault is finished.

For the case of a valve jamming in the open position, the simulation is slightly more difficult. The simplest solution is to change the value of Q_B directly by inputting the maximum flow rate and then supplying the dummy expression.

$$Q_B = Q_B$$

to maintain the flow at this value.

The same dummy expression may be used to simulate a valve that has jammed in the normal position although no prior change in variable value is used for this case. Such a condition will not cause any symptoms to appear during simulation unless a further change is imposed on the model at the same time.

The simple model shown makes no provision for a difference between the line pressure, P_C, set by the controller, and the valve bonnet pressure. (Note that the "full" valve model shown in Fig. 4.1(a) of Chapter 4 does include this feature). If it is desired to consider a leaking valve bonnet this may be overcome by reducing the value of K_V.

For a valve where the process stream is leaking, a change must be made in the high gain equation used to preserve continuity. Since the leak has no corresponding stream on the model used, a mass "sink" must be created. This is done by including a further constant in the high gain equation:

$$\frac{dP_A}{dt} = G_A(K_A Q_A - Q_B)$$

where K_A is less than unity.

Some faults may be time-dependent. As an example, consider a compressor failure in the process air system. The decrease in pressure might be considered to follow a simple decaying exponential:

$$P_C = P_N e^{-K_1 t}$$

where: P_N is the normal pressure

P_C is the actual pressure at time t

K_1 is the decay constant

Whilst this failure might be considered a little unlikely, it serves to illustrate the point that such faults are plausible. More complicated time-dependent faults may be simulated by use of the trigonometric and logarithmic functions given in Section A.2.2.1.