THE DAVE SYSTEM USER'S MANUAL*

by

Lloyd D. Fosdick
Carol Miesse Drey
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado 80309

#CU-CS-106-77

March, 1977

Manuscript typed by:
Dorothy Foerst

* This work was supported in part by the National Science Foundation under Grant No. MCS75-09972.

THE DAVE SYSTEM USER's MANUAL

CONTENTS

		Page
I.	INTRODUCTION	1
	1. Overview	1
	2. Basic definitions	1
	3. Discussion of undefinition of a variable	4
	4. Fundamental assumptions of DAVE	4
II.	USAGE	5
	1. How to execute DAVE	5
	2. How to interpret the DAVE output	6
	2.1 Overview	6
	2.2 Output prologue	8
	2.3 Error diagnostics	16
	2.4 Warning diagnostics	43
	2.5 Message diagnostics	119
	2.6 Diagnostics	128
III.	INTERNAL STRUCTURE	130
	1. Major software components	130
	2. Tables generated	134
	3. File organization	144
	4. System dependencies	146
	5. Size alterations	150
	5.1 Maximum data base size	150
	5.2 Other size limitations	151
	6. Recovery	155
IV.	INSTALLATION	156
V.	INTERNAL DIAGNOSTICS	162
	REFERENCES	190
	APPENDIX A	A1--A45
	APPENDIX B	B1--B3
	APPENDIX C	C1

DAVE USER's MANUAL

Chapter I: INTRODUCTION

1. Overview

DAVE is a software tool for gathering information about global data flow in FORTRAN programs, and for identifying the anomalous use of data in these programs. In the usual terminology DAVE is characterized as a static analysis tool, meaning that DAVE gathers information about the subject program from scanning it as, for example, a compiler would to construct the object code. DAVE does not require any modification of the subject program, nor does it require any intervention by the user during execution. Only some initial setting of parameters which control the output is required.

The DAVE system has been described in an article in Software Practice & Experience [1]. We will assume familiarity with this reference. A survey article on the use of data flow analysis to detect software errors has also appeared [2]. The philosophy of our approach is described in this reference but the implementation details are different.

This manual has four chapters in addition to the introduction. Chapter 2 is of particular interest to the general user; Chapter 3 is of interest to those who may wish to know some details of the inner working of DAVE. Chapter 4 contains instructions for installing DAVE on a computer, and Chapter 5 contains explanations of DAVE's internal diagnostics.

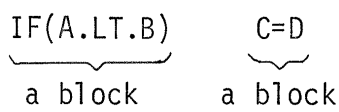
This manual also contains in an appendix a complete listing of output obtained from a DAVE run on a program designed to generate all DAVE diagnostics. Although the circumstances of this DAVE execution are exceptional, perusal of the output in this appendix should help the reader gain a further understanding of the kinds of program constructs which will lead to certain diagnostic messages from DAVE.

2. Basic definitions

For ease of reference we list these definitions in alphabetic order. The words defined here have other meanings too; the intended meaning should be clear from context.

available COMMON block: A COMMON block is said to be available to a program unit if it is declared in the program unit or if it is declared in some program unit which is an ancestor of it, as defined by the call graph.

block: A FORTRAN statement, a sequence of FORTRAN statements or in the case of a logical IF, either of the two components of the logical IF; e.g.



In DAVE output blocks are numbered for reference purposes.

call graph: The graph structure representing the order in which program units are invoked. The nodes are program units, and an edge (b,c) represents the fact that execution of b may invoke subprogram c.

defined: A variable is said to be defined if it has a specific value according to the FORTRAN standard; e.g. execution of the statement

```
A=B+C
```

assigns a specific value to A and thus at subsequent points in the execution history A is defined until a point is reached where it becomes undefined (see below).

executable path: A path is executable if there exists input data for the program containing this path which would force execution of the blocks in the order defined by the path.

input: A variable is said to be an input variable for a block, or a subprogram, if execution of the block, or subprogram may require that the variable be defined at the time of entry to the block or subprogram; e.g. the variable C is an input variable for the block

```
A=C+1
```

and for the subprogram

```
SUBROUTINE EXAMPL(C,D)
IF(D.LT.0,0) RETURN
C=C+1
RETURN
END
```

output: A variable is said to be an output variable for a block, or a subprogram, if execution of the block, or subprogram, may result in the assignment of a value to the variable and at the time of exit from the block or subprogram the variable remains defined as a consequence of the assignment; e.g. C is an output variable for the block

```
C=C+1
```

and it is an output variable for the subprogram shown above in the definition of input. In the second case notice that the variable C will not be assigned a value if D<0, but it is an output variable because it will be assigned a value if D≥0.

path: A sequence of blocks $b_i, b_{i+1}, b_{i+2}, \ldots, b_{i+k}$ in a program, such that execution of b_j may be followed by execution of b_{j+1}.

program unit: Main program or subprogram.

referenced: Similar to input (see above) and normally used when talking about the use of a variable in a single statement; thus we say, C is referenced in the statement

```
A=C+1.
```

strict input: This phrase has the same meaning as input except the underlined word "may" in the definition of input is replaced by must; e.g. referring again to the definition of input, the variable D is strict input for the subroutine EXAMPL, but C is not strict input for this subroutine since there is a path through the subroutine which does not require a value of C.

strict output: This phrase has the same meaning as output except the underlined word "may" in the definition of output is replaced by must; e.g. in the subroutine below NUMB is strict output

```
      SUBROUTINE COUNT (A,B,N,NUMB)
      DIMENSION A(N)
      NUMB=0
      DO 10 K=1,N
        IF(A(K).EQ.B) NUMB=NUMB+1
   10 CONTINUE
      RETURN
      END
```

but in the subroutine EXAMPL (cf. definition of input) C is not strict output since there is a path through the subroutine which does not assign a value to C.

subject program: The program analyzed by DAVE.

3. Discussion of undefinition of a variable

DAVE, in accordance with ANS Fortran, assumes that upon entry to a program unit, all local variables for the unit are undefined unless initialized in a DATA statement. An undefined variable becomes defined when a value is assigned to it. Once a variable is defined it can become undefined in several ways. When a RETURN statement or a STOP statement is reached all local variables for this program unit become undefined. When a DO-loop is satisfied, the control variable for the DO becomes undefined. When a local variable, initialized in a DATA statement, is assigned a value by execution of some statement in the program unit, then upon subsequent entry to the program unit the local variable is undefined. When a COMMON variable is defined in a program unit, it will become undefined upon exit from the unit if the variable's COMMON block is not available to the calling program unit.

4. Fundamental assumptions of DAVE.

DAVE assumes that the subject program is a syntactically correct ANS FORTRAN program. If the subject program does not satisfy this requirement the results are unpredictable. Therefore it is essential that the subject program be run through a good diagnostic compiler or through a tool such as PFORT [4] which will detect ANS FORTRAN syntax errors. A few deviations from ANS FORTRAN are permitted and are listed in Appendix C.

In addition, for DAVE's analysis to be correct, there must be at most one non-trivial connected component in the subject program's call graph (where a trivial component is a graph consisting of one node) and the non-trivial component is acyclic with a unique entry node.

Chapter II: DAVE USAGE

1. How to execute DAVE

In the following discussion, we will assume that there is a procedure file called DAVE, set up when DAVE was installed (see Chapter IV), which when called controls DAVE's execution. The necessary inputs to DAVE are:

1. The Fortran program to be analyzed. This may consist of a single subprogram, a group of subprograms, or a main program and subprograms, with the restrictions outlined in Chapter I, Section 4. The limits on the size of the subject program are: a maximum of 100 subprograms run through together, each of which has no more than 500 blocks. All declarations are counted as one block; otherwise each statement is a block, with logical IF's counting as two blocks; comments and formats are excluded from the count. Although the limit is 100 program units run together, the larger the group, the more strain placed on all internal arrays and overflow may occur. Detailed information on all size limitations appears in Chapter III, Section 5.

2. An options file. This allows the user to specify certain instructions for DAVE. The format is as follows:

Keyword	Parameters	Default	Description
SI=	ON OFF	OFF	Simulation of unsatisfied externals. If SI=ON, missing subprograms will be simulated by DAVE to permit analysis to continue. If SI=OFF and there are unsatisfied externals, DAVE will terminate analysis.
SU=	d_i,d_j,...,d_m-d_r ALL E or ERRORS ALL W or WARNINGS ALL M or MESSAGES OFF	OFF(i.e. all diagnostics will be printed)	Suppression of diagnostics, d_i indicates a diagnostic number, d_m-d_r a range of diagnostic numbers to be suppressed. The use of ALL will suppress an entire category. The parameters may occur in any order.

The options may be expressed in any order, may be separated by blanks or commas, and may extend over any number of records in the file.

The options file is read first during DAVE's execution so that if it and the subject program are on cards, it must precede the subject program

and be separated from it by an end-of-file. If all defaults are desired, the options file may be empty.

The method of calling the procedure file DAVE will depend upon the particular installation. For example, on the CDC 6400 under KRONOS 2.1, the following command is used:

```
CALL(DAVE(OPTIONS=file1,INPUT=file2)
where file1 = options file
      file2 = subject program
```

The maximum core required for any phase of DAVE's execution is 135000 octal words on the CDC 6400. An upper bound for the time requirement is approximately 1 CPU sec./ line of subject program.

2. How to interpret the DAVE output.

2.1 Overview

Sections 2.1 through 2.5 in this chapter discuss the output produced when DAVE completes execution of PHASE3 (standard termination). If execution is terminated by PHASE0, there is a different output format, which is discussed in Section 2.6.

In a normal DAVE run the first line of the output file will read

```
DAVE LEVEL ....
```

where the dots stand for a level number which actually appears in their place. At the time of this writing the level number is 8.0. After this line the following message will appear

```
DAVE TERMINATION NORMAL
```

Next in order of appearance is a message describing the options selected by the user on the data card. If the subject program had missing subprograms and the user requested a simulation of the I/O behavior then a note concerning this is printed before the list of user options specified.

The list of user options specified is an echo of the values (ON or OFF) assigned to the three parameters SI (simulate I/O behavior for missing subprograms), RE (restart of previous run), SU

(suppress diagnostics). The RE option may be used only under special circumstances; see Chapter III section 6 for a discussion of the recovery capability.

If the DAVE run failed because of a fatal error, and thus did not terminate normally, the following message appears in the output file:

FATAL ERROR(S) -- DAVE ANALYSIS CANNOT CONTINUE

Diagnostic messages related to the fatal error(s) follow this. A tabulation of possible error messages is presented later. We assume now that the DAVE run termination is normal, and proceed with the general description of the organization of the output file.

Next in order of appearance on the output file is the line:

DIAGNOSTIC SUMMARY -- PART 1

Following this line is a tabulation of the number of occurrences of error diagnostics, warning diagnostics, and message diagnostics issued for each of the program units of the subject program.

The next section on the output file begins with the line

DIAGNOSTIC SUMMARY -- PART 2

Following this line is a tabulation of the number of occurrences of each diagnostic in the entire program.

After the diagnostic summaries there is a listing of the call graph, headed by the line

CALL GRAPH

This is printed in a three column format. The first column contains a list of all of the program units in the subject program. Associated with each program unit, S, in column 1, there is a list in column 2 of those which call S, and there is a list in column 3 of those which are called by S. The program units are listed in column 1 in the order of their appearance in the program.

At this point the general, summary information printed by DAVE comes to an end and the more detailed information concerning the results of the analysis of each subprogram begins. The general layout is as follows. There is a listing of each subprogram and following the listing of the subprogram the error diagnostics,

warning diagnostics, and message diagnostics pertinent to that subprogram are printed. The listing of the subprogram is annotated on the left by block numbers. These block numbers are used to reference paths and individual blocks in the diagnostics following the subprogram. The various types of diagnostics are numbered for easy reference: errors are numbered in the one hundreds; warnings are numbered in the two hundreds; messages are numbered in the three hundreds. There are twelve error diagnostics, thirty-seven warning diagnostics, and four message diagnostics. As explained earlier, the user can suppress the printing of any of these diagnostics by specifications on the data card.

2.2 Output Prologue

This is the first part of the DAVE output which under normal conditions begins with the message:

```
DAVE TERMINATION NORMAL
```

Following this is a message regarding the options selected by the user on the data card. All options selected by the user are listed here (these options are discussed above in connection with the options file (cf. p.5). If all default options are selected, then this segment of the output appears as follows:

```
USER OPTIONS SPECIFIED THIS RUN
-------------------------------

1.  SIMULATE I/O BEHAVIOR FOR MISSING SUBPROGRAMS (SI=OFF).
2.  RE-START OF PREVIOUS RUN   (RE=OFF).
3.  SUPPRESS DIAGNOSTICS (SU=OFF).
```

If the user does want to simulate missing subprograms, uses the default on the restart option, and suppresses diagnostics numbered 105, 202, 209, 303, 304, then this segment of the output will appear as follows:

```
USER OPTIONS SPECIFIED THIS RUN
-------------------------------

1.  SIMULATE I/O BEHAVIOR FOR MISSING SUBPROGRAMS (SI=ON).
2.  THIS IS NOT A RECOVERY RUN (RE=OFF).
3.  SUPPRESS DIAGNOSTICS (SU= )   105  202  209  303  304
```

The list of suppressed diagnostics is printed in order of increasing number. If the user has SU=ALL E on the data card then an explicit listing of all error diagnostic numbers is printed, rather than ALL E and the same is true mutatis mutandis, for ALL W, ALL M. When the option SI=ON is selected and there are missing subprograms, the following note appears in this portion of the output immediately above the list of user options:

NOTE -- FOR MISSING SUBPROGRAMS THE FOLLOWING I/O BEHAVIOR HAS BEEN SIMULATED.

A. FOR FUNCTION SUBPROGRAMS, THE FUNCTION NAME HAS BEEN CLASSIFIED AS STRICT OUTPUT AND ALL ARGUMENTS AS STRICT INPUT, NON OUTPUT.

B. FOR SUBROUTINE SUBPROGRAMS, ALL ARGUMENTS HAVE BEEN CLASSIFIED AS STRICT INPUT, NON OUTPUT.

A SIMULATED SUBPROGRAM IS ASSUMED TO USE NO COMMON VARIABLES. THE NUMBER AND DIMENSIONS OF ITS DUMMY ARGUMENTS HAVE BEEN INFERRED FROM THE FIRST INVOCATION OF THE SUBPROGRAM BY THE PROGRAM UNIT INDICATED BELOW.

SIMULATED SUBPROGRAM	CALLER
---	---

Since the intent of this option has already been discussed in the previous section we will not repeat this discussion here.

An example of the printing of part 1 of a diagnostic summary is shown below.

DIAGNOSTIC SUMMARY -- PART 1

SUBPROGRAM	FREQUENCY ERRORS	WARNINGS	MESSAGES
SYSMAIN	24	48	5
BLKDATA			1
E101	2	5	1
SUB103	1	4	2
SUB302	1	5	2
SUB105		2	1
SUB106	1	1	2
SUB208	2	1	1
W201		4	1
SUB215		1	1
SUB		4	1
FUN	1	2	1
FSIM			1
SUBSIM			1

The interpretation of this is rather obvious. Reading across a line of this table we find the number of error diagnostics, warning diagnostics, and message diagnostics for the subprogram (or main program SYSMAIN or block data BLKDATA) named in the first column of the line. To illustrate, in this example the DAVE analysis of the subprogram E101 resulted in two error diagnostics, five warning diagnostics, and one message diagnostic.

An example of the printing of part 2 of a diagnostic summary is shown below.

DIAGNOSTIC SUMMARY -- PART 2

ERRORS		WARNINGS		MESSAGES	
IDENT.NO.	FREQUENCY	IDENT.NO.	FREQUENCY	IDENT.NO.	FREQUENCY
101	1	201	1	301	2
102	2	202	1	302	2
103	4	203	1	303	3
104	1	204	5	304	14
105	1	205	1		
106	2	206	1		
107	2	207	2		
108	2	208	1		
109	2	209	1		
110	4	210	3		
111	2	211	1		
112	9	212	1		
		213	2		
		214	2		
		215	1		
		216	5		
		217	2		
		218	1		
		219	1		
		220	1		
		221	1		
		222	1		
		223	2		
		224	1		
		225	1		
		226	1		
		227	1		
		228	1		
		229	12		
		230	1		
		231	1		
		232	2		
		233	2		
		234	1		
		235	1		
		236	10		
		237	4		

Here again the interpretation is fairly obvious. Under the heading ERRORS there is a listing of the frequency of occurrence of each error diagnostic detected by DAVE in the subject program. In this example all error diagnostics were generated by DAVE at least once. Similar remarks apply to the columns headed WARNINGS and MESSAGES. In this example we see that warning diagnostic 201 occurred once, that message diagnostic 301 occurred twice, etc. The two examples shown here, DIAGNOSTIC SUMMARY -- PART 1 and DIAGNOSTIC SUMMARY -- PART 2, are part of the same DAVE output (shown in entirety in Appendix A). Note that the corresponding column sums in PART 1 and PART 2 agree, there being a total of 32 errors, 77 warnings, and 21 messages according to column sums in PART 1 and in PART 2. We reiterate that even when diagnostics are suppressed they will still appear as counted in these summaries. In PART 2 if diagnostic 230 had been suppressed, then in the printing of this summary the identification number 230 would have been marked by an asterisk (viz. 230*). In this case, the following note also would appear in the listing:

*DENOTES A SUPPRESSED DIAGNOSTIC WHICH WILL NOT APPEAR IN THE LISTING.

An example of the printing of a call graph appears below.

CALL GRAPH

SUBPROGRAM	CALLED BY	CALLS
SYSMAIN		E101
		SUB103
		SUB105
		SUB106
		SUB208
		W201
		SUB215
		SUB
		FSIM
		SUBSIM
E101	SYSMAIN	
	SUB106	
SUB103	SYSMAIN	SUB302
SUB302	SUB103	SUB106
SUB105	SYSMAIN	SUB106
SUB106	SYSMAIN	E101
	SUB302	
	SUB105	
SUB208	SYSMAIN	
W201	SYSMAIN	
SUB215	SYSMAIN	
SUB	SYSMAIN	
FUN		
FSIM	SYSMAIN	
SUBSIM	SYSMAIN	

2.3 Error Diagnostics

First, a word about notation used in all diagnostics. In order to make identifiers stand out in the diagnostics they appear in "brackets"; for example, the identifier D218 will appear in a DAVE diagnostic as ---*D218*--. Paths of control flow through a program are specified by block numbers. For example, the path specified by 8 9 10 refers to a path in which control passes from block 8, to block 9, to block 10. The path specified by 1 - 6 8 refers to a path in which control passes from block 1 to block 6 through consecutively numbered blocks (i.e. 2 3 4 5) and then to block 8.

Generally speaking an error diagnostic refers to a situation which DAVE detects that is very likely, but not certain, to result in an error when the subject program executes. For example, we include among errors those situations in which execution of the subject program will always cause a reference to an uninitialized variable. A program containing such an error would execute correctly if the user intended the initial value to be zero and the system under which the program executes always initializes the store to zero, but it would not execute correctly under a different system protocol.

Most error diagnostics refer to an anomalous use of data on all paths to some block. If the anomalous use occurs only on some, but not all, paths then a corresponding warning diagnostic instead of an error diagnostic is printed; for example, compare diagnostics 106 and 206. This distinction is connected with our inability to recognize executable paths. We attempt to deal with it as follows. We assume that every block can be reached on some executable path; hence if an anomalous use of data occurs on every path to the block, it must occur on an executable path. On the other hand, if the anomalous use occurs only on some paths to a block then it might only occur on non-executable paths. It is for this reason that we often let an anomalous use on all paths cause an error diagnostic while an anomalous use on some paths only causes a warning diagnostic. Some anomalous uses are judged to be less likely to represent errors than others and may cause warning diagnostics even when they occur on all paths; for example, see diagnostic 222.

When there is an error message and a corresponding warning message, the numbering of the diagnostics reflects the correspond-

ence; e.g. 101 and 201 correspond, 102 and 202 correspond, etc.

There are twelve error diagnostics, numbered 101, 102, ..., 112. They are discussed below in numerical order. In the DAVE output these diagnostics are printed immediately after the listing of the program unit to which they refer. Errors can result from certain combinations of circumstances that occur, or fail to occur, at different points in a program. A diagnostic is associated with one of these points and if the other point or points occur in other program units then the diagnostic is printed for the program unit closest to the root of the call graph.

101 DESCRIPTION **101**

This diagnostic can only appear with a function subprogram. It appears when DAVE detects an attempt to reference an undefined variable on every path through the subprogram and the variable's name is the name of the function subprogram.

101 EXAMPLE **101**

Program segment

```
      BLOCK                          SOURCE

         1           FUNCTION FUNA(X,Y)
         2           DO 10 J =1,50
         3           FUNA = FUNA + X + Y
         4        10 CONTINUE
         0   C       .
         0   C       .
         0   C       .
         5           RETURN
         1           END
```

Diagnostic

```
** 101 **  FUNCTION NAME ---*FUNA*-- IS REFERENCED BEFORE BEING ASSIGNED
           A VALUE ON ALL PATHS.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
              1     2     3
```

102 DESCRIPTION **102**

This diagnostic can only appear with a function subprogram. It appears if the function name is never assigned a value.

102 EXAMPLE **102**

Program segment

```
BLOCK                    SOURCE

    1          FUNCTION FUNB(X,Y)
    2          DO 10 J = 1,50
    3          Y = X + Y
    4       10 CONTINUE
    5          RETURN
    1          END
```

Diagnostic

```
**102 **  FUNCTION NAME ---*FUNB*-- IS NEVER ASSIGNED A VALUE.
```

103 DESCRIPTION **103**

This diagnostic is associated with a subprogram call. It appears if an actual argument in the subprogram call is a constant or an expression (excepting the trivial case where the expression is just a variable) and the following is true: on every path in the called subprogram the dummy argument corresponding to this actual argument is assigned a value.

103 EXAMPLE **103**

Program segment

```
BLOCK                          SOURCE

   1          SUBROUTINE ASUB(R,S)
   2          X = 1.0
   3          CALL BSUB(X,3.0,2.0)
   0   C      .
   0   C      .
   0   C      .
   4          RETURN
   1          END
```

```
BLOCK                          SOURCE

   1          SUBROUTINE BSUB(A,B,C)
   2          B = A + C
   3          RETURN
   1          END
```

Diagnostic (printed after subroutine ASUB)

```
** 103 **  BLOCK NO.     3
          AN ACTUAL ARGUMENT IS AN EXPRESSION OR CONSTANT, YET THE
          CORRESPONDING DUMMY ARGUMENT IS ASSIGNED A VALUE ON ALL PATHS.
                                CALLING SUBPROGRAM  CALLED SUBPROGRAM
                                   ---*ASUB*--         ---*BSUB*--
                ARGUMENT              REAL              ----*B*----
                POSITION               2                    2
```

104 DESCRIPTION **104**

This diagnostic is associated with a subprogram call. It appears if the number of actual arguments in the subprogram call and the number of dummy arguments in the called subprogram are unequal.

104 EXAMPLE **104**

Program segment

```
BLOCK                          SOURCE

   1          SUBROUTINE CSUB(R,S)
   0  C       .
   0  C       .
   0  C       .
   2          CALL DSUB(X,Y,Z)
   0  C       .
   0  C       .
   0  C       .
   3          RETURN
   1          END
```

```
BLOCK                          SOURCE

   1          SUBROUTINE DSUB(A,B)
   0  C       .
   0  C       .
   0  C       .
   2          RETURN
   1          END
```

Diagnostic (printed after subroutine CSUB)

```
** 104 **  BLOCK NO.     2
           THE NUMBER OF DUMMY ARGUMENTS IN CALLED SUBPROGRAM ---*DSUB*--
           DOES NOT AGREE WITH THE NUMBER OF ACTUAL ARGUMENTS SUPPLIED
           BY CALLING SUBPROGRAM ---*CSUB*--.
```

105 DESCRIPTION **105**

This diagnostic is associated with a subprogram call. It appears if both of the following are true: an actual argument in the call is an external procedure name; the corresponding dummy argument in the called procedure is referenced, as if it were a simple variable, on every path.

105 EXAMPLE **105**

Program segment

```
BLOCK                              SOURCE

   1          SUBROUTINE ESUB(R,S)
   1          EXTERNAL FUN
   0   C      .
   0   C      .
   0   C      .
   2          CALL FSUB(X,Y,FUN)
   0   C      .
   0   C      .
   0   C      .
   3          RETURN
   1          END
```

```
BLOCK                              SOURCE

   1          SUBROUTINE FSUB(A,B,C)
   2          A = B + C
   3          RETURN
   1          END
```

Diagnostic (printed after subroutine ESUB)

```
** 105 **  BLOCK NO.     2
          AN ACTUAL ARGUMENT IS A PROCEDURE DECLARED EXTERNAL, YET THE
          CORRESPONDING DUMMY ARGUMENT IS REFERENCED AS A VARIABLE
          ON ALL PATHS.
                                CALLING SUBPROGRAM  CALLED SUBPROGRAM
                                    ---*ESUB*--         ---*FSUB*--
                ARGUMENT            ---*FUN*---         ----*C*----
                POSITION                 3                   3
```

106 DESCRIPTION **106**

This diagnostic is associated with a subprogram call. It appears if both of the following are true: an actual argument in the call is an external procedure name; the corresponding dummy argument in the called procedure is assigned a value, as if it were a simple variable, on every path.

106 EXAMPLE **106**

Program segment

```
BLOCK                          SOURCE

  1          SUBROUTINE GSUB(R,S)
  1          EXTERNAL FUN
  0  C       .
  0  C       .
  0  C       .
  2          CALL HSUB(X,Y,FUN)
  0  C       .
  0  C       .
  0  C       .
  3          RETURN
  1          END
```

```
BLOCK                          SOURCE

  1          SUBROUTINE HSUB(A,B,C)
  2          C = A + B
  3          RETURN
  1          END
```

Diagnostic (printed after subroutine GSUB)

```
** 106 **  BLOCK NO.     2
          AN ACTUAL ARGUMENT IS A PROCEDURE DECLARED EXTERNAL, YET THE
          CORRESPONDING DUMMY ARGUMENT, USED AS A VARIABLE, IS ASSIGNED
          A VALUE ON ALL PATHS.
                              CALLING SUBPROGRAM  CALLED SUBPROGRAM
                                 ---*GSUB*--         ---*HSUB*--
                ARGUMENT         ---*FUN*---         ----*C*----
                POSITION               3                   3
```

107 DESCRIPTION **107**

This diagnostic is associated with a subprogram and the use of a COMMON variable in the subprogram. It appears if a dummy argument for the subprogram and a COMMON variable have the same identifier.

107 EXAMPLE **107**

Program segment

```
BLOCK                    SOURCE

   1          SUBROUTINE ISUB(ABS,X)
   1          COMMON Y,ABS
   0  C       .
   0  C       .
   0  C       .
   2          RETURN
   1          END
```

Diagnostic

** 107 ** THE NAME ---*ABS*--- IS USED TO REPRESENT BOTH A DUMMY ARGUMENT AND A COMMON VARIABLE IN THIS SUBPROGRAM.

108 DESCRIPTION **108**

This diagnostic is associated with a subprogram call. It appears if a dummy argument is bound to a COMMON variable because of the subprogram call and execution of the subprogram always causes an assignment of a value to the variable.

108 EXAMPLE **108**

Program segment

```
BLOCK                          SOURCE

   1          SUBROUTINE JSUB(R,S)
   1          COMMON A,B,C
   0  C       .
   0  C       .
   0  C       .
   2          CALL KSUB(C,D,E)
   0  C       .
   0  C       .
   0  C       .
   3          RETURN
   1          END

BLOCK                          SOURCE

   1          SUBROUTINE KSUB(T,U,V)
   1          COMMON F,G,H
   2          T = T + 1
   0  C       .
   0  C       .
   0  C       .
   3          RETURN
   1          END
```

Diagnostic (printed after subroutine JSUB)

```
** 108 **  BLOCK NO.     2
           A SUBPROGRAM REFERENCE CAUSES DUMMY ARGUMENT ----*T*----
           TO BECOME ASSOCIATED WITH A COMMON VARIABLE IN THE CALLED
           SUBPROGRAM.  ----*T*---- IS ASSIGNED A VALUE ON ALL PATHS.
                              CALLING SUBPROGRAM  CALLED SUBPROGRAM
                                 ---*JSUB*--         ---*KSUB*--
                 ARGUMENT        ----*C*----         ----*T*----
           COMMON VARIABLE       ----*C*----         ----*H*----
```

109 DESCRIPTION **109**

This diagnostic is associated with a main program and the use of a COMMON variable. It appears if on every path in the main program an undefined variable is referenced. The reference of the undefined variable might occur within a procedure called directly or indirectly by the main program.

109 EXAMPLE **109**

Program segment

```
BLOCK                          SOURCE

   0  C--MAIN PROGRAM
   1         COMMON /LAB/ J1,J2
   2         X = 1.0
   3         CALL SUB(X,Y)
   0  C      .
   0  C      .
   0  C      .
   4         STOP
   1         END

BLOCK                          SOURCE

   1         BLOCK DATA
   1         COMMON /LAB/ K1,K2
   1         DATA K1/3/
   1         END

BLOCK                          SOURCE

   1         SUBROUTINE SUB(A,B)
   1         COMMON /LAB/ L1,L2
   2         IF (L2 .LT. 2)
   3        $ RETURN
   0  C      .
   0  C      .
   0  C      .
   4         RETURN
   1         END
```

Diagnostic (printed after main program)

** 109 ** COMMON VARIABLE ----*J2*--- IN COMMON BLOCK ---*LAB*--- IS REFERENCED ON ALL PATHS IN THE MAIN PROGRAM, YET IT HAS NOT PREVIOUSLY BEEN ASSIGNED A VALUE, NOR HAS IT BEEN INITIALIZED IN BLOCK DATA. (SEE NOTE 1)

NOTE 1 ---- ALTHOUGH DETECTED IN THIS SUBPROGRAM, THE CAUSE FOR THIS DIAGNOSTIC MAY HAVE OCCURRED AT A DEEPER LEVEL OF SUBPROGRAM REFERENCES AND BEEN PROPAGATED UP TO THIS ONE.

110 DESCRIPTION **110**

This diagnostic is associated with a subprogram and the use of a COMMON variable. It appears if on every path in the subprogram an undefined variable is referenced. The reference of the undefined variable might occur within a procedure called directly or indirectly by the subprogram.

110 EXAMPLE **110**

Program segment

```
BLOCK                          SOURCE

   0  C--MAIN PROGRAM
   2         X=1.0
   3         CALL ASUB(X,Y)
   0  C      .
   0  C      .
   0  C      .
   4         STOP
   1         END
BLOCK                          SOURCE

   1         BLOCK DATA
   1         COMMON /LAB/K1,K2
   1         DATA K1/3/
   1         END
BLOCK                          SOURCE

   1         SUBROUTINE ASUB(A,B)
   1         COMMON /LAB/ L1,L2
   2         CALL BSUB(A)
   0  C      .
   0  C      .
   0  C      .
   3         RETURN
   1         END
BLOCK                          SOURCE

   1         SUBROUTINE BSUB(Z)
   1         COMMON /LAB/ M1,M2
   2         IF (M2 .LT. 2)
   3        $ RETURN
   0  C      .
   0  C      .
   0  C      .
   4         RETURN
   1         END
```

Diagnostic (printed after main program)

** 110 ** COMMON VARIABLE ----*L2*--- IS REFERENCED ON ALL PATHS IN CALLED SUBPROGRAM ---*ASUB*--, YET IS NOT INITIALIZED. IT DOES NOT APPEAR IN BLOCK DATA, AND ITS COMMON BLOCK ---*LAB*--- IS NOT AVAILABLE TO CALLING SUBPROGRAM -*SYSMAIN*-. (SEE NOTE 1)

NOTE 1 ---- - ALTHOUGH DETECTED IN THIS SUBPROGRAM, THE CAUSE FOR THIS DIAGNOSTIC MAY HAVE OCCURRED AT A DEEPER LEVEL OF SUBPROGRAM REFERENCES AND BEEN PROPAGATED UP TO THIS ONE.

111 DESCRIPTION **111**

This diagnostic is associated with the control variable of a DO loop. It appears if on all paths from the closing statement of the DO loop the control variable is referenced and between the closing statement and the reference there is no definition of the variable. (The control variable is undefined when the DO loop is satisfied and an exit is made from the closing statement of the DO loop.)

111 EXAMPLE **111**

Program segment

```
BLOCK          SOURCE

  0   C
  0   C--MAIN PROGRAM
  0   C       .
  0   C       .
  0   C       .
  7           DO 10 K = 1,20
  0   C       .
  0   C       .
  0   C       .
  8       10  CONTINUE
  9       20  X = A(K)
  0   C       .
  0   C       .
  0   C       .
  1           END
```

Diagnostic

```
** 111 **  CONTROL VARIABLE ----*K*---- BECOMES UNDEFINED UPON SATISFACTION
           OF ITS DO LOOP AT BLOCK NO.    8, YET IS REFERENCED ON ALL
           PATHS THEREAFTER.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
              8     9
```

112 DESCRIPTION **112**

This diagnostic is associated with the use of a local variable in a main program or subprogram. It appears if the local variable is referenced and on all paths from the entry point to the point of reference it is undefined. The reference might occur within a subprogram called directly or indirectly from this program unit.

112 EXAMPLE **112**

Program segment

```
BLOCK                    SOURCE

   1          SUBROUTINE MSUB(X,Y)
   2          IF (K .LT. 0)
   3         $ X = X + Y
   0  C       .
   0  C       .
   0  C       .
   4          RETURN
   1          END
```

Diagnostic

```
** 112 **  LOCAL VARIABLE ----*K*---- IS REFERENCED BEFORE BEING ASSIGNED
           A VALUE ON ALL PATHS.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
              1     2
```

2.4 Warning Diagnostics

There are thirty-seven different warning diagnostics. They are discussed below in numerical order. In the DAVE output these diagnostics are printed immediately after the listing of the program unit to which they refer. They are printed after the error diagnostics.

201 DESCRIPTION **201**

This diagnostic can only appear with a function subprogram. It appears when DAVE detects an attempt to reference an undefined variable on some, but not all, paths through the subprogram and the variable's name is the name of the function subprogram.

201 EXAMPLE **201**

Program segment

```
BLOCK                                SOURCE

   1          FUNCTION FUNC(X,Y)
   2          IF (X .LT. Y)
   3        $ GO TO 10
   4          FUNC = 0.0
   5          RETURN
   6       10 DO 20 J = 1,50
   7          FUNC = FUNC + X + Y
   8       20 CONTINUE
   0  C       .
   0  C       .
   0  C       .
   9          RETURN
   1          END
```

Diagnostic

```
** 201 **  FUNCTION NAME ---*FUNC*-- IS REFERENCED BEFORE BEING
           ASSIGNED A VALUE ON SOME PATHS.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
              1     2     3     6     7
```

202 DESCRIPTION **202**

This diagnostic can only appear with a function subprogram. It appears if the function name is undefined on some but not all paths to a RETURN statement. Normally this means that the function name is not always assigned a value on the path but it could appear if the name is assigned a value and then is undefined.

202 EXAMPLE **202**

Program segment

```
BLOCK                    SOURCE

  1          FUNCTION FUND(X,Y)
  2          IF (X .LT. Y)
  3        $ FUND = 0.0
  4          DO 10 J = 1,50
  5          Y = Y + X
  6       10 CONTINUE
  7          RETURN
  1          END
```

Diagnostic

** 202 ** FUNCTION NAME ---*FUND*-- IS NOT ASSIGNED A VALUE ON SOME PATHS.

203 DESCRIPTION **203**

This diagnostic is associated with a subprogram call. It appears if an actual argument in the subprogram call is a constant or an expression (excepting the trivial case where the expression is just a variable) and the following is true: on some, but not all, paths in the called subprogram the dummy argument corresponding to this actual argument is assigned a value.

203 EXAMPLE **203**

Program segment

```
BLOCK                          SOURCE

   1          SUBROUTINE NSUB(R,S)
   2          X = 1.0
   3          CALL OSUB(X,3.0,2.0)
   0  C       .
   0  C       .
   0  C       .
   4          RETURN
   1          END
```

```
BLOCK                          SOURCE

   1          SUBROUTINE OSUB(A,B,C)
   2          IF (A .GT. B)
   3         $ B = A + C
   4          A = A + C
   5          RETURN
   1          END
```

Diagnostic (printed after subroutine NSUB)

```
** 203 **  BLOCK NO.     3
          AN ACTUAL ARGUMENT IS AN EXPRESSION OR CONSTANT, YET THE
          CORRESPONDING DUMMY ARGUMENT IS ASSIGNED A VALUE ON SOME PATHS.
                                CALLING SUBPROGRAM  CALLED SUBPROGRAM
                                    ---*NSUB*--         ---*OSUB*--
                ARGUMENT                REAL             ----*B*----
                POSITION                  2                    2
```

204 DESCRIPTION **204**

This diagnostic is associated with a subprogram call. It appears if an actual argument in the call is an expression, but not just a variable, and the corresponding dummy argument in the called subprogram is never referenced.

204 EXAMPLE **204**

Program segment

```
BLOCK                SOURCE

  0  C--MAIN PROGRAM
  0  C      .
  0  C      .
  0  C      .
  4         CALL BSUB(X,X+Y,Z)
  0  C      .
  0  C      .
  0  C      .
  5         STOP
  1         END
```

```
BLOCK                      SOURCE

  1          SUBROUTINE BSUB(A,B,C)
  2          IF (C .LT. 0.0)
  3         $ C = -C
  4          A = A + C
  5          RETURN
  1          END
```

Diagnostic (printed after main program)

```
** 204 **  BLOCK NO.     4
          AN ACTUAL ARGUMENT IS AN EXPRESSION OR CONSTANT, YET THE
          CORRESPONDING DUMMY ARGUMENT IS NEVER REFERENCED.
                                CALLING SUBPROGRAM  CALLED SUBPROGRAM
                                   -*SYSMAIN*-          ---*BSUB*--
                ARGUMENT             EXPRESSION          ----*B*----
                POSITION                  2                    2
```

205 DESCRIPTION **205**

This diagnostic is associated with a subprogram call. It appears if both of the following are true: an actual argument in the call is an external procedure name; the corresponding dummy argument in the called procedure is referenced as if it were a simple variable on some, but not all, paths.

205 EXAMPLE **205**

Program segment

```
BLOCK                          SOURCE

   1          SUBROUTINE CSUB(R,S)
   1          EXTERNAL FUN
   0  C       .
   0  C       .
   0  C       .
   2          CALL DSUB(X,Y,FUN)
   0  C       .
   0  C       .
   0  C       .
   3          RETURN
   1          END

BLOCK                          SOURCE

   1          SUBROUTINE DSUB(A,B,C)
   2          IF(A .LT. B)
   3         $A = B + C
   4          B = B + C(3)
   5          RETURN
   1          END
```

Diagnostic (printed after subroutine CSUB)

```
** 205 **  BLOCK NO.     2
          AN ACTUAL ARGUMENT IS A PROCEDURE DECLARED EXTERNAL, YET THE
          CORRESPONDING DUMMY ARGUMENT IS REFERENCED AS A VARIABLE ON
          SOME PATHS.
                                CALLING SUBPROGRAM  CALLED SUBPROGRAM
                                   ---*CSUB*--          ---*DSUB*--
                ARGUMENT           ---*FUN*---          ----*C*----
                POSITION                 3                    3
```

206 DESCRIPTION **206**

This diagnostic is associated with a subprogram call. It appears if both of the following are true: an actual argument in the call is an external procedure name; the corresponding dummy argument in the called procedure is assigned a value as if it were a simple variable on some, but not all, paths.

206 EXAMPLE **206**

Program segment

```
BLOCK                          SOURCE

   1          SUBROUTINE RSUB(R,S)
   1          EXTERNAL FUN
   0  C       .
   0  C       .
   0  C       .
   2          CALL SSUB(X,Y,FUN)
   0  C       .
   0  C       .
   0  C       .
   3          RETURN
   1          END

BLOCK                          SOURCE

   1          SUBROUTINE SSUB(A,B,C)
   2          IF (A .LT. B)
   3        $ C = A + B
   4          RETURN
   1          END
```

Diagnostic (printed after subroutine RSUB)

```
** 206 **  BLOCK NO.     2
          AN ACTUAL ARGUMENT IS A PROCEDURE DECLARED EXTERNAL, YET THE
          CORRESPONDING DUMMY ARGUMENT, USED AS A VARIABLE, IS ASSIGNED
          A VALUE ON SOME PATHS.
                              CALLING SUBPROGRAM  CALLED SUBPROGRAM
                                 ---*RSUB*--         ---*SSUB*--
                ARGUMENT         ---*FUN*---         ----*C*----
                POSITION               3                   3
```

207 DESCRIPTION **207**

This diagnostic is associated with a subprogram. It appears if a dummy argument in the subprogram declaration is never used.

207 EXAMPLE **207**

Program segment

```
BLOCK                    SOURCE

   1          SUBROUTINE TSUB(X,I)
   2          X = X + 3.0
   3          CALL SUB(X,J)
   4          RETURN
   1          END
```

Diagnostic

```
** 207 **  DUMMY ARGUMENT ----*I*---- IS NEVER USED.
```

208 DESCRIPTION **208**

This diagnostic is associated with a subprogram call. It appears if a dummy argument is bound to a COMMON variable because of the subprogram call and execution of the subprogram causes an assignment of a value to the variable on some, but not all, paths.

208 EXAMPLE **208**

Program segment

```
BLOCK                    SOURCE

   1          SUBROUTINE USUB(R,S)
   1          COMMON A,B,C
   0  C       .
   0  C       .
   0  C       .
   2          CALL VSUB(C,D,E)
   0  C       .
   0  C       .
   0  C       .
   3          RETURN
   1          END

BLOCK                    SOURCE

   1          SUBROUTINE VSUB(T,U,V)
   1          COMMON F,G,H
   2          IF (U .LT. V)
   3         $ T = T + 1
   4          RETURN
   1          END
```

Diagnostic (printed after subroutine USUB)

```
** 208 **  BLOCK NO.     2
           A SUBPROGRAM REFERENCE CAUSES DUMMY ARGUMENT ----*T*----
           TO BECOME ASSOCIATED WITH A COMMON VARIABLE IN THE CALLED
           SUBPROGRAM.  ----*T*---- IS ASSIGNED A VALUE ON SOME PATHS.
                                CALLING SUBPROGRAM  CALLED SUBPROGRAM
                                   ---*USUB*--         ---*VSUB*--
                 ARGUMENT          ----*C*----         ----*T*----
           COMMON VARIABLE         ----*C*----         ----*H*----
```

209 DESCRIPTION **209**

This diagnostic is associated with a main program and the use of a COMMON variable. It appears if on some, but not all, paths in the main program an undefined variable is referenced. The reference of the undefined variable might occur within a procedure called directly or indirectly by the main program.

209 EXAMPLE **209**

Program segment

```
BLOCK                          SOURCE

   0  C--MAIN PROGRAM
   1         COMMON /LAB/ J1,J2
   2         X = 1.0
   3         CALL ASUB(X,Y)
   0  C      .
   0  C      .
   0  C      .
   1         END

BLOCK                          SOURCE

   1         BLOCK DATA
   1         COMMON /LAB/ K1,K2
   1         DATA K1/3/
   1         END

BLOCK                          SOURCE

   1         SUBROUTINE ASUB(A,B)
   1         COMMON /LAB/ L1,L2
   2         IF(L1 .LT. 1)
   3        $ RETURN
   4         L1 = L2-1
   0  C      .
   0  C      .
   0  C      .
   5         RETURN
   1         END
```

Diagnostic (printed after main program)

** 209 ** COMMON VARIABLE ----*J2*--- IN COMMON BLOCK ---*LAB*--- IS REFERENCED ON SOME PATHS IN THE MAIN PROGRAM, YET IT HAS NOT PREVIOUSLY BEEN ASSIGNED A VALUE, NOR HAS IT BEEN INITIALIZED IN BLOCK DATA. (SEE NOTE 1)

NOTE 1 ALTHOUGH DETECTED IN THIS SUBPROGRAM, THE CAUSE FOR THIS
---- - DIAGNOSTIC MAY HAVE OCCURRED AT A DEEPER LEVEL OF SUBPROGRAM REFERENCES AND BEEN PROPAGATED UP TO THIS ONE.

210 DESCRIPTION **210**

This diagnostic is associated with a subprogram and the use of a COMMON variable. It appears if on some, but not all, paths in a subprogram an undefined variable is referenced. The reference of the undefined variable might occur within a procedure called directly or indirectly by the subprogram.

211 DESCRIPTION **211**

This diagnostic is associated with the control variable of a DO loop. It appears if on some, but not all, paths from the closing statement of the DO loop the control variable is referenced before being defined. (The control variable is undefined when the DO loop is satisfied and an exit is made from the closing statement of the DO loop.)

210 EXAMPLE **210**

Program segment

```
BLOCK                          SOURCE

   0  C      .
   0  C      .
   0  C      .
   0  C--MAIN PROGRAM
   4         Y = 1.0
   5         CALL BSUB(Y,Z)
   0  C      .
   0  C      .
   0  C      .
   6         STOP
   1         END

BLOCK                          SOURCE

   1         BLOCK DATA
   1         COMMON /LB/ J1,J2
   1         DATA J1/3/
   1         END

BLOCK                          SOURCE

   1         SUBROUTINE BSUB(A,B)
   1         COMMON /LB/ L1,L2
   2         CALL CSUB(A)
   0  C      .
   0  C      .
   0  C      .
   3         RETURN
   1         END

BLOCK                          SOURCE

   1         SUBROUTINE CSUB(Z)
   1         COMMON /LB/ M1,M2
   2         IF (M1 .LT. 1)
   3        $ RETURN
   4         M1 = M2 - 1
   0  C      .
   0  C      .
   0  C      .
   5         RETURN
   1         END
```

Diagnostic (printed after main program)

```
** 210 ** COMMON VARIABLE ----*L2*--- IS REFERENCED ON SOME PATHS IN
         CALLED SUBPROGRAM ---*BSUB*--, YET IS NOT INITIALIZED.
         IT DOES NOT APPEAR IN BLOCK DATA, AND ITS COMMON BLOCK
         ----*LB*--- IS NOT AVAILABLE TO CALLING SUBPROGRAM
         -*SYSMAIN*-.  (SEE NOTE 1)

NOTE 1   ALTHOUGH DETECTED IN THIS SUBPROGRAM, THE CAUSE FOR THIS
         DIAGNOSTIC MAY HAVE OCCURRED AT A DEEPER LEVEL OF SUBPROGRAM
         REFERENCES AND BEEN PROPAGATED UP TO THIS ONE.
```

211 EXAMPLE **211**

Program segment

```
BLOCK                SOURCE
  0   C--MAIN PROGRAM
  0   C      .
  0   C      .
  0   C      .
 12          DO 100 I = 1,20
  0   C      .
  0   C      .
  0   C      .
 13     100 CONTINUE
 14          IF (X .LT. Y)
 15         $ STOP
 16          X = A(I)
  0   C      .
  0   C      .
  0   C      .
  1          END
```

Diagnostic

```
** 211 **  CONTROL VARIABLE ----*I*---- BECOMES UNDEFINED UPON SATISFACTION
           OF ITS DO LOOP AT BLOCK NO.   13, YET IS REFERENCED ON SOME
           PATHS THEREAFTER.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
             13    14    16
```

212 DESCRIPTION **212**

This diagnostic is associated with the use of a local variable in a main program or subprogram. It appears if the local variable is referenced while undefined on some, but not all, paths from the entry point. The reference might occur within a subprogram called directly or indirectly from this program unit.

212 EXAMPLE **212**

Program segment

```
BLOCK                                SOURCE

   1          SUBROUTINE WSUB(X,Y)
   2          K = 1
   3          IF (X .LT. Y)
   4        $ CALL XSUB(J,K)
   0  C       .
   0  C       .
   0  C       .
   5          RETURN
   1          END
```

```
BLOCK                                SOURCE

   1          SUBROUTINE XSUB(L,M)
   2          IF(M.GT.1)
   3        $ L = L + 1
   4          RETURN
   1          END
```

Diagnostic (printed after subroutine WSUB)

```
** 212 **  LOCAL VARIABLE ----*J*---- IS REFERENCED BEFORE BEING
            ASSIGNED A VALUE ON SOME PATHS.
            ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
               1 -   4
```

213 DESCRIPTION **213**

This diagnostic is associated with a subprogram call. It appears if the data type of an actual argument differs from the data type of the corresponding dummy argument in the called subprogram.

213 EXAMPLE **213**

Program segment

```
BLOCK                    SOURCE

  0  C--MAIN PROGRAM
 17         B = 1.0
 18         CALL ESUB(C,J)
  0  C      .
  0  C      .
  0  C      .
 19         STOP
  1         END

BLOCK                          SOURCE

  1         SUBROUTINE ESUB(X,Y)
  2         X = Y + 1.0
  0  C      .
  0  C      .
  0  C      .
  3         RETURN
  1         END
```

Diagnostic (printed after main program)

```
** 213 **  BLOCK NO.   18
          CORRESPONDING ARGUMENTS HAVE DIFFERENT DATA TYPES.
                                CALLING SUBPROGRAM  CALLED SUBPROGRAM
                                   -*SYSMAIN*-          ---*ESUB*--
                ARGUMENT           ----*J*----          ----*Y*----
                POSITION                2                    2
                DATA TYPE             INTEGER                REAL
```

214 DESCRIPTION **214**

This diagnostic is associated with a subprogram call. It appears if the data type of a COMMON variable in the calling program unit differs from the data type of the corresponding COMMON variable in the called subprogram.

214 EXAMPLE **214**

Program segment

```
BLOCK                                SOURCE

    1          SUBROUTINE DSUB(X,Y)
    1          COMMON /BLK/J,K,F
    2          L = 1
    0   C      .
    0   C      .
    0   C      .
    3          CALL ESUB(A,B)
    0   C      .
    0   C      .
    0   C      .
    4          RETURN
    1          END

BLOCK                                SOURCE

    1          SUBROUTINE ESUB(C,D)
    1          COMMON /BLK/ L,M,N
    2          N = 0
    0   C      .
    0   C      .
    0   C      .
    3          RETURN
    1          END
```

Diagnostic (printed after subroutine DSUB)

```
** 214 **  CORRESPONDING COMMON VARIABLES IN COMMON BLOCK ---*BLK*---
           HAVE DIFFERENT DATA TYPES.
                                CALLING SUBPROGRAM   CALLED SUBPROGRAM
                                   ---*DSUB*--          ---*ESUB*--
                VARIABLE           ----*F*----          ----*N*----
                DATA TYPE             REAL                 INTEGER
```

215 DESCRIPTION **215**

This diagnostic is associated with a subprogram call. It appears if an actual argument in the subprogram call has a dimension different from that of the corresponding dummy argument in the called subprogram.

215 EXAMPLE **215**

Program segment

```
BLOCK                          SOURCE

    0  C--MAIN PROGRAM
    1         DIMENSION A(10,10)
    0  C      .
    0  C      .
    0  C      .
    2         CALL ASUB(A,B)
    0  C      .
    0  C      .
    0  C      .
    1         END

BLOCK                          SOURCE

    1         SUBROUTINE ASUB(C,D)
    1         DIMENSION C(100)
    0  C      .
    0  C      .
    0  C      .
    2         RETURN
    1         END
```

Diagnostic (printed after main program)

```
** 215 **  BLOCK NO.     2
          CORRESPONDING ARGUMENTS HAVE DIFFERENT DIMENSIONALITY.
                                CALLING SUBPROGRAM  CALLED SUBPROGRAM
                                   -*SYSMAIN*-          ---*ASUB*--
                ARGUMENT           ----*A*----          ----*C*----
                POSITION                1                    1
                DIMENSIONS              2                    1
```

216 DESCRIPTION **216**

This diagnostic is associated with a subprogram call and a COMMON variable. It appears if both of the following are true: on every path in the called subprogram the last use of the COMMON variable is a definition of it; the COMMON block for the variable is not available to the calling subprogram.

216 EXAMPLE **216**

Program segment

```
BLOCK                    SOURCE

   0  C--MAIN PROGRAM
   3        X = 1.0
   4        CALL BSUB(X,Y)
   0  C     .
   0  C     .
   0  C     .
   1        END
```

```
BLOCK                          SOURCE

   1        SUBROUTINE BSUB(C,D)
   1        COMMON /BLOCK/ K
   2        K = 1
   3        D = C + 3.0
   4        RETURN
   1        END
```

Diagnostic (printed after main program)

** 216 ** COMMON VARIABLE ----*K*---- IS ASSIGNED A VALUE ON ALL PATHS IN CALLED SUBPROGRAM ---*BSUB*--, YET ITS COMMON BLOCK --*BLOCK*-- IS NOT AVAILABLE TO CALLING SUBPROGRAM -*SYSMAIN*-. HENCE, A COMPUTED VALUE WILL BE LOST. (SEE NOTE 1)

NOTE 1 ALTHOUGH DETECTED IN THIS SUBPROGRAM, THE CAUSE FOR THIS DIAGNOSTIC MAY HAVE OCCURRED AT A DEEPER LEVEL OF SUBPROGRAM REFERENCES AND BEEN PROPAGATED UP TO THIS ONE.

217 DESCRIPTION **217**

This diagnostic is associated with a subprogram call and a COMMON variable. It appears if both of the following are true: on some, but not all, paths in the called subprogram the last use of the COMMON variable is a definition of it; the COMMON block containing this COMMON variable is not available to the calling subprogram.

217 EXAMPLE **217**

Program segment

```
BLOCK                          SOURCE
    0  C--MAIN PROGRAM
    5         Y = 1.0
    6         CALL CSUB(Y,Z)
    0  C      .
    0  C      .
    0  C      .
    1         END
BLOCK                          SOURCE


    1         SUBROUTINE CSUB(C,D)
    1         COMMON /BLK/ K
    2         CALL CCSUB(C)
    3         D = C + 3.0
    4         RETURN
    1         END

BLOCK                          SOURCE


    1         SUBROUTINE CCSUB(E)
    1         COMMON /BLK/ K
    2         IF (E .LT. 0.0)
    3        $ K = 1
    4         RETURN
    1         END
```

Diagnostic (printed after the main program)

** 217 ** COMMON VARIABLE ----*K*---- IS ASSIGNED A VALUE ON SOME PATHS IN CALLED SUBPROGRAM ---*CSUB*--, YET ITS COMMON BLOCK ---*BLK*--- IS NOT AVAILABLE TO CALLING SUBPROGRAM -*SYSMAIN*-. HENCE, A COMPUTED VALUE MAY BE LOST. (SEE NOTE 1)

NOTE 1 ALTHOUGH DETECTED IN THIS SUBPROGRAM, THE CAUSE FOR THIS
---- - DIAGNOSTIC MAY HAVE OCCURRED AT A DEEPER LEVEL OF SUBPROGRAM REFERENCES AND BEEN PROPAGATED UP TO THIS ONE.

218 DESCRIPTION **218**

This diagnostic is associated with a subprogram call and a COMMON variable initialized in BLOCK DATA. It appears if both of the following are true: on all paths in the called subprogram the last use of the COMMON variable is a definition of it; the COMMON block containing this COMMON variable is not available to the calling subprogram.

218 EXAMPLE **218**

Program segment

```
BLOCK                          SOURCE

   0  C
   0  C
   0  C--MAIN PROGRAM
   7         CALL DSUB(U,V)
   0  C      .
   0  C      .
   0  C      .
   1         END

BLOCK                          SOURCE


   1         BLOCK DATA
   1         COMMON /BLOK/ K
   1         DATA K/1/
   1         END

BLOCK                          SOURCE


   1         SUBROUTINE DSUB(A,B)
   1         COMMON /BLOK/ K
   2         K = K + 1
   0  C      .
   0  C      .
   0  C      .
   3         RETURN
   1         END
```

Diagnostic (printed after main program)

** 218 ** COMMON VARIABLE ----*K*---- IS INITIALIZED IN BLOCK DATA. IT IS ASSIGNED A VALUE ON ALL PATHS IN CALLED SUBPROGRAM ---*DSUB*--, YET ITS COMMON BLOCK ---*BLOK*-- IS NOT AVAILABLE TO CALLING SUBPROGRAM -*SYSMAIN*-. HENCE, UNDEFINITION WILL OCCUR UPON EXIT FROM ---*DSUB*--. (SEE NOTE 2)

NOTE 2 ---- - IF MESSAGE 301 CONCERNING THIS VARIABLE APPEARS IN THE OUTPUT, IT MAY PROVIDE ADDITIONAL USEFUL INFORMATION ABOUT THE DATA FLOW AMONG SUBPROGRAMS.

219 DESCRIPTION **219**

This diagnostic is associated with a subprogram call and a COMMON variable initialized in BLOCK DATA. It appears if both of the following are true: on some, but not all, paths in the called subprogram the last use of the COMMON variable is a definition of it; the COMMON block containing this COMMON variable is not available to the calling subprogram. This diagnostic is like 218 except it applies to some, but not all, paths.

219 EXAMPLES **219**

Program segment

```
BLOCK                          SOURCE

      0  C
      0  C--MAIN PROGRAM
      8        CALL ESUB(U,V)
      0  C     .
      0  C     .
      0  C     .
      9        STOP
      1        END

BLOCK                          SOURCE

      1        BLOCK DATA
      1        COMMON /BLCK/ L
      1        DATA L /1/
      1        END

BLOCK                          SOURCE

      1        SUBROUTINE ESUB(A,B)
      1        COMMON /BLCK/ K
      2        READ (5,100) A,B
      3        IF (A .LT. B)
      4       $ K = K + 1
      5        RETURN
      0    100 FORMAT (2E10.0)
      1        END
```

Diagnostic (printed after main program)

** 219 ** COMMON VARIABLE ----*K*---- IS INITIALIZED IN BLOCK DATA. IT IS ASSIGNED A VALUE ON SOME PATHS IN CALLED SUBPROGRAM ---*ESUB*--, YET ITS COMMON BLOCK ---*BLCK*-- IS NOT AVAILABLE TO CALLING SUBPROGRAM -*SYSMAIN*-. HENCE, UNDEFINITION MAY OCCUR UPON EXIT FROM ---*ESUB*--. (SEE NOTE 2)

NOTE 2 ---- - IF MESSAGE 301 CONCERNING THIS VARIABLE APPEARS IN THE OUTPUT, IT MAY PROVIDE ADDITIONAL USEFUL INFORMATION ABOUT THE DATA FLOW AMONG SUBPROGRAMS.

220 DESCRIPTION **220**

This diagnostic is associated with the use of a local variable in a subprogram. It appears if both of the following are true: the local variable is initialized in a DATA statement; on every path in the subprogram this local variable is assigned a value.

220 EXAMPLE **220**

Program segment

```
BLOCK                          SOURCE

   1          SUBROUTINE HSUB(X,Y)
   1          DATA K/0/
   2          K = K + 1
   0  C       .
   0  C       .
   0  C       .
   3          RETURN
   1          END
```

Diagnostic

```
** 220 **  LOCAL VARIABLE ----*K*----, INITIALIZED IN A DATA STATEMENT,
           IS ASSIGNED A VALUE ON ALL PATHS.  UNDEFINITION WILL OCCUR
           UPON EXIT FROM THIS SUBPROGRAM.
```

221 DESCRIPTION **221**

This diagnostic is associated with the use of a local variable in a subprogram. It appears if both of the following are true: the local variable is initialized in a DATA statement; on some, but not all, paths in the subprogram this local variable is assigned a value. This diagnostic is like 220 except it applies to some, but not all, paths.

221 EXAMPLE **221**

Program segment

```
BLOCK                    SOURCE

   1          SUBROUTINE ISUB(X,Y)
   1          DATA K/0/
   2          IF (K .NE. 0)
   3         $ RETURN
   4          K = 1
   0  C       .
   0  C       .
   0  C       .
   5          RETURN
   1          END
```

Diagnostic

** 221 ** LOCAL VARIABLE ----*K*----, INITIALIZED IN A DATA STATEMENT, IS ASSIGNED A VALUE ON SOME PATHS. UNDEFINITION MAY OCCUR UPON EXIT FROM THIS SUBPROGRAM.

222 DESCRIPTION **222**

This diagnostic appears after a subprogram if some dummy argument for the subprogram is defined in a statement and on all paths from the point of definition both of the following are true: there is another definition of the dummy argument; between the two points of definition there is no reference of the dummy argument.

222 EXAMPLE **222**

Program segment

```
BLOCK                    SOURCE

  1          SUBROUTINE JSUB(X,Y)
  2          X = 1.0
  3          IF (Y .LT. 0.0)
  4         $ Y = Y + 1.0
  5          X = Y ** 2
  6          RETURN
  1          END
```

Diagnostic

```
** 222 **  DUMMY ARGUMENT ----*X*---- IS ASSIGNED A VALUE IN BLOCK
           NO.    2 AND IS ASSIGNED A VALUE THEREAFTER BEFORE BEING
           REFERENCED, ON ALL PATHS.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
              2 -   5
```

223 DESCRIPTION **223**

This diagnostic appears after a subprogram if some dummy argument for the subprogram is defined in a statement and both of the following are true on some, but not all paths, from the point of definition: (1) there is a second definition of the dummy argument; (2) between the two points of definition there is no reference of the dummy argument. This diagnostic is like 222 except it applies to some, but not all, paths.

Program segment

```
BLOCK                    SOURCE

  1          SUBROUTINE KSUB(X,Y)
  2          X = 1.0
  3          IF (Y .LT. 0.0)
  4         $ RETURN
  5          X = Y**2
  6          RETURN
  1          END
```

Diagnostic

```
** 223 **  DUMMY ARGUMENT ----*X*---- IS ASSIGNED A VALUE IN BLOCK
           NO.    2 AND IS ASSIGNED A VALUE THEREAFTER BEFORE BEING
           REFERENCED, ON SOME PATHS.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
              2     3     5
```

224 DESCRIPTION **224**

This diagnostic appears after a subprogram if some COMMON variable is defined in a statement and on all paths from the point of definition both of the following are true: there is another definition of the COMMON variable; between the two points of definition there is no reference of the COMMON variable.

224 EXAMPLE **224**

Program segment

```
BLOCK                    SOURCE

    1          SUBROUTINE FSUB(Y)
    1          COMMON /BLOC/ X
    2          X = 1.0
    3          IF (Y .LT. 0.0)
    4         $ Y = 1.0
    5          X = Y**2
    6          RETURN
    1          END
```

Diagnostic

```
** 224 **  COMMON VARIABLE ----*X*---- IS ASSIGNED A VALUE IN BLOCK
           NO.    2 AND IS ASSIGNED A VALUE THEREAFTER BEFORE BEING
           REFERENCED, ON ALL PATHS.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
              2 -   5
```

225 DESCRIPTION **225**

This diagnostic appears after a subprogram if some COMMON variable is defined in a statement and on some, but not all, paths from the point of definition both of the following are true: (1) there is another definition of the COMMON variable; (2) between the two points of definition there is no reference of the COMMON variable. This diagnostic is like 224 except it applies to some, but not all, paths.

225 EXAMPLE **225**

Program segment

```
BLOCK                    SOURCE

   1          SUBROUTINE GSUB(Y)
   1          COMMON /CMBLK/ X
   2          X = 1.0
   3          IF (Y .LT. 0.0)
   4         $ RETURN
   5          X = Y**2
   6          RETURN
   1          END
```

Diagnostic

```
** 225 **  COMMON VARIABLE ----*X*---- IS ASSIGNED A VALUE IN BLOCK
           NO.    2 AND IS ASSIGNED A VALUE THEREAFTER BEFORE BEING
           REFERENCED, ON SOME PATHS.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
              2     3     5
```

226 DESCRIPTION **226**

This diagnostic appears after the main program if some COMMON variable is defined in a statement and on all paths from the point of definition both of the following are true: there is another definition of the COMMON variable, or an undefinition of the COMMON variable (a STOP statement does this); between the first definition and the second definition, or undefinition, there is no reference of the COMMON variable.

226 EXAMPLE **226**

Program segment

```
BLOCK                          SOURCE

   0  C--MAIN PROGRAM
   1        COMMON /BLK/ X
   2        X = 1.0
   3        READ (5,100) Y
   4        IF(Y .LT. 0.0)
   5       $ STOP
   6        X = Y**2
   0  C      .
   0  C      .
   0  C      .
   7        STOP
   0    100 FORMAT(F3.0)
   1        END
```

Diagnostic

```
** 226 **  IN THE MAIN PROGRAM, COMMON VARIABLE ----*X*---- IS
           ASSIGNED A VALUE IN BLOCK  NO.    2 AND IS EITHER
           ASSIGNED A VALUE THEREAFTER BEFORE BEING REFERENCED,
           OR IS NOT SUBSEQUENTLY REFERENCED, ON ALL PATHS.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
              2 -   5
```

227 DESCRIPTION **227**

This diagnostic appears after the main program if some COMMON variable is defined in a statement and on some, but not all, paths from the point of definition both of the following are true: there is another definition of the COMMON variable or a STOP statement; between the first definition and the second definition, or STOP statement, there is no reference of the COMMON variable.

Program segment

```
BLOCK                          SOURCE

  0   C--MAIN PROGRAM
  1         COMMON /BLOCK/ X
  2         X = 1.0
  3         READ (5,100) Y
  4         IF (Y .LT. 0.0)
  5        $ STOP
  6         Y = X**2
  0   C      .
  0   C      .
  0   C      .
  0     100 FORMAT (F3.0)
  1         END
```

Diagnostic

```
** 227 **  IN THE MAIN PROGRAM, COMMON VARIABLE ----*X*---- IS
           ASSIGNED A VALUE IN BLOCK  NO.     2 AND IS EITHER
           ASSIGNED A VALUE THEREAFTER BEFORE BEING REFERENCED,
           OR IS NOT SUBSEQUENTLY REFERENCED, ON SOME PATHS.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
              2 -   5
```

228 DESCRIPTION **228**

This diagnostic appears after the main program if an element of an array in COMMON is defined in a statement and on every path from this point of definition there is no reference to any element of the array.

228 EXAMPLE **228**

Program segment

```
BLOCK                    SOURCE

    0  C
    0  C
    0  C--MAIN PROGRAM
    1         COMMON /BLCK/ A(20)
    2         READ (5,150) U,V
    3         DO 10 K = 1,20
    4         A(20) = U
    5      10 CONTINUE
    0  C      .
    0  C      .
    0  C      .
    0  C   (A NOT USED IN ANY STATEMENT)
    0  C      .
    0  C      .
    0  C      .
    0     150 FORMAT (2F3.0)
    1         END
```

Diagnostic

```
** 228 **  IN THE MAIN PROGRAM, AN ELEMENT OF THE COMMON ARRAY
           ----*A*---- IS ASSIGNED A VALUE IN BLOCK NO.    4
          AND THE ARRAY IS NOT SUBSEQUENTLY REFERENCED ON ANY PATH.
```

229 DESCRIPTION **229**

This diagnostic appears after a program unit if a local variable is defined in a statement and on every path from this statement one of the following is true: there is another definition of the variable and between the first definition and the second definition there is no reference of the variable; a RETURN or STOP statement is reached and there is no reference of the variable between the first definition and the RETURN or STOP.

229 EXAMPLE **229**

Program segment

```
BLOCK                    SOURCE

   1          SUBROUTINE HSUB(X,Y)
   2          J = 1
   3          IF (X .LT. 0.0)
   4         $ J = 2
   5          Y = Y + 1
   6          RETURN
   1          END
```

Diagnostic

```
** 229 **  LOCAL VARIABLE ----*J*---- IS ASSIGNED A VALUE IN BLOCK
           NO.    2 AND IS EITHER ASSIGNED A VALUE THEREAFTER BEFORE
           BEING REFERENCED, OR IS NOT SUBSEQUENTLY REFERENCED,
           ON ALL PATHS.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
              2     3     4
```

230 DESCRIPTION **230**

This diagnostic appears after a program unit if a local variable is defined in a statement and on some, but not all, paths from this statement one of the following is true: there is another definition of the variable and between the first definition and the second definition there is no reference of the variable; a RETURN or STOP statement is reached and there is no reference of the variable between the first definition and the RETURN or STOP

230 EXAMPLE **230**

Program segment

```
BLOCK                    SOURCE

   1          SUBROUTINE SUB(X,Y)
   2          J = 1
   3          IF(X .LT. Y)
   4         $ RETURN
   5          Y = J + 1
   6          RETURN
   1          END
```

Diagnostic

```
** 230 **  LOCAL VARIABLE ----*J*---- IS ASSIGNED A VALUE IN BLOCK
           NO.    2 AND IS EITHER ASSIGNED A VALUE THEREAFTER BEFORE
           BEING REFERENCED, OR IS NOT SUBSEQUENTLY REFERENCED,
           ON SOME PATHS.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
              2     3     4
```

231 DESCRIPTION **231**

This diagnostic appears after a program unit if an element of a local array is defined in a statement and on all paths from this statement to a RETURN or STOP statement there is no reference to any element of the local array.

231 EXAMPLE **231**

Program segment

```
BLOCK                    SOURCE

    1          SUBROUTINE JSUB(X,Y)
    1          DIMENSION A(10)
    2          DO 10 K = 1,10
    3          A(K) = 0.0
    4       10 CONTINUE
    5          X = X + Y
    6          RETURN
    1          END
```

Diagnostic

```
** 231 **  AN ELEMENT OF THE LOCAL ARRAY ----*A*---- IS ASSIGNED A VALUE
           IN BLOCK NO.    3 AND THE ARRAY IS NOT SUBSEQUENTLY
           REFERENCED ON ANY PATH.
```

232 DESCRIPTION **232**

This diagnostic is associated with a subprogram call. It appears if both of the following are true: the same argument appears more than once as an actual argument; in one appearance it is strict input for the called subprogram and in the other appearance it is strict output for the called subprogram. This diagnostic appears also when there is more than one subprogram call in a statement as in

```
Y=AFUN(B,D) + BFUN(B,E)
```

where B is strict input for BFUN and strict output for AFUN.

232 EXAMPLE **232**

Program segment

```
BLOCK            SOURCE
    0  C
    0  C
    0  C--MAIN PROGRAM
   11         Z = 1.0
   12         CALL ASUB(Z,Z,W)
    0  C      .
    0  C      .
    0  C      .
    1         END

BLOCK                        SOURCE


    1         SUBROUTINE ASUB(A,B,C)
    2         A = 1.0
    3         C = 1.0
    4         IF (B .LT. 0.0)
    5        $ C = 2.0
    6         RETURN
    1         END
```

Diagnostic (printed after main program)

** 232 ** BLOCK NO. 12
A POSSIBLE ILLEGAL SIDE EFFECT HAS BEEN DETECTED. IT OCCURS VIA A VARIABLE PASSED IN AN ARGUMENT LIST. THIS VARIABLE HAS APPEARED AT LEAST TWICE IN A STATEMENT -- IN ONE APPEARANCE IT IS USED AS STRICT INPUT AND IN THE OTHER AS STRICT OUTPUT.

	CALLING SUBPROGRAM -*SYSMAIN*-	CALLED SUBPROGRAM ---*ASUB*--
ARGUMENT	----*Z*----	----*B*----
POSITION	2	2

233 DESCRIPTION **233**

This diagnostic is associated with a subprogram call and a COMMON variable. It appears if both of the following are true: a COMMON variable is used more than once in execution of the statement; in one case it is used as strict input and in the other it is used as strict output.

233 EXAMPLE **233**

Program segment

```
BLOCK                          SOURCE

    0  C--MAIN PROGRAM
    1         COMMON /BLK/ C,D
    2         C = 1.0
    3         A = 2.0
    4         Y = C + FUN(A)
    0  C      .
    0  C      .
    0  C      .
    5         STOP
    1         END

BLOCK                          SOURCE

    1         FUNCTION FUN(Y)
    1         COMMON /BLK/ C,D
    2         FUN = Y**2 + Y + 1.0
    3         C = Y - 1.0
    0  C      .
    0  C      .
    0  C      .
    4         RETURN
    1         END
```

Diagnostic

```
** 233 **  BLOCK NO.     4
          A POSSIBLE ILLEGAL SIDE EFFECT HAS BEEN DETECTED.  IT OCCURS
          VIA A COMMON VARIABLE WHICH HAS BEEN REFERENCED (POSSIBLY
          INDIRECTLY) AT LEAST TWICE IN A STATEMENT -- IN ONE APPEAR-
          ANCE IT IS USED AS STRICT INPUT AND IN THE OTHER AS STRICT
          OUTPUT.
                                CALLING SUBPROGRAM  CALLED SUBPROGRAM
                                   -*SYSMAIN*-         ---*FUN*---
                    VARIABLE       ----*C*----         ----*C*----
               COMMON BLOCK        ---*BLK*---         ---*BLK*---
```

234 DESCRIPTION **234**

This diagnostic appears after a program unit if a statement references an arithmetic statement function and both of the following are true: a variable is used twice in execution of the statement; in one case it is used as strict input and in the other it is used as strict output.

234 EXAMPLE **234**

Program segment

```
BLOCK                          SOURCE

   0  C--MAIN PROGRAM
   1         AFUN(X,Y) = BFUN(C) + X + Y
   0  C      .
   0  C      .
   0  C      .
   2         Z = AFUN(E,F) + C
   0  C      .
   0  C      .
   0  C      .
   1         END
```

```
BLOCK                          SOURCE

   1         FUNCTION BFUN(D)
   2         D = D + 1
   3         RETURN
   1         END
```

Diagnostic

```
** 234 **  BLOCK NO.     2
     A POSSIBLE ILLEGAL SIDE EFFECT HAS BEEN DETECTED.  IT OCCURS
     VIA A GLOBAL VARIABLE REFERENCED IN AN ARITHMETIC STATEMENT
     FUNCTION.  THIS VARIABLE HAS APPEARED AT LEAST TWICE IN A
     STATEMENT -- IN ONE APPEARANCE IT IS USED AS STRICT INPUT AND
     IN THE OTHER AS STRICT OUTPUT.
                          CALLING SUBPROGRAM  CALLED SUBPROGRAM
                             -*SYSMAIN*-          ---*AFUN*--
               VARIABLE      ----*C*----          ----*C*----
```

235 DESCRIPTION **235**

This diagnostic appears after a program unit if the program unit is never referenced as an external procedure.

235 EXAMPLE **235**

Program (complete)

```
BLOCK                          SOURCE

   0  C--MAIN PROGRAM
   1         READ(5,100) X
   2         READ(6,100) X
   3         STOP
   0     100 FORMAT(E20.10)
   1         END


BLOCK                          SOURCE


   1         SUBROUTINE CSUB(Y)
   2         Y = Y + 1
   3         RETURN
   1         END
```

Diagnostic (printed after subroutine CSUB)

```
** 235 **  SUBPROGRAM ---*CSUB*-- IS NEVER CALLED.
```

236 DESCRIPTION **236**

This diagnostic appears after a program unit if a local variable is not defined in any statement.

236 EXAMPLE **236**

Program segment

```
BLOCK                    SOURCE

   1          SUBROUTINE DSUB(X,Y)
   1          INTEGER Z
   2          IF (X .LT. Y)
   3         $ X = X + 1
   4          RETURN
   1          END
```

Diagnostic

```
** 236 **  LOCAL VARIABLE ----*Z*---- IS NEVER ASSIGNED A VALUE.
```

237 DESCRIPTION **237**

This diagnostic appears after a program unit if the following is true: a variable in COMMON has a data type which is different from the data type of the same variable used in BLOCK DATA.

237 EXAMPLE **237**

Program segment

```
BLOCK                    SOURCE

   1          BLOCK DATA
   1          COMMON /CMBLK/ A,B,C
   1          DATA A,B,C/1.0,2.0,3.0/
   1          END

BLOCK                    SOURCE

   1          SUBROUTINE SUB(X)
   1          COMMON /CMBLK/ N,O,P
   0  C       .
   0  C       .
   0  C       .
   2          RETURN
   1          END
```

Diagnostic (printed after subroutine SUB)

```
** 237 **  CORRESPONDING COMMON VARIABLES IN COMMON BLOCK --*CMBLK*--
           HAVE DIFFERENT DATA TYPES IN SUBPROGRAM ---*SUB*---
           AND BLOCK DATA.

                              SUBPROGRAM          BLOCK DATA
                             ---*SUB*---
                VARIABLE     ----*N*----         ----*A*----
                DATA TYPE      INTEGER             REAL
```

2.5 Message Diagnostics

There are four message diagnostics. They are described below in numerical order. In the DAVE output these diagnostics are printed immediately after the listing of the program unit to which they refer. They follow the warning diagnostics.

301 DESCRIPTION **301**

This diagnostic identifies a COMMON variable initialized in BLOCK DATA. It appears after the program unit, highest in the call graph, in which the COMMON variable is classified as input or strict input.

301 EXAMPLE **301**

Program segment

```
BLOCK                          SOURCE

   0  C
   0  C
   0  C--MAIN PROGRAM
   1         COMMON /BLK/ A,B
   2         B = A + 1
   0  C      .
   0  C      .
   0  C      .
   1         END

BLOCK                          SOURCE

   1         BLOCK DATA
   1         COMMON /BLK/ X,Y
   1         DATA X/1.0/
   1         END
```

Diagnostic

```
** 301 **  COMMON VARIABLE ----*A*---- IN BLOCK ---*BLK*--- OF
           SUBPROGRAM -*SYSMAIN*- IS INITIALIZED IN BLOCK DATA.
```

302 DESCRIPTION **302**

This appears with a program unit if a COMMON block is available to it but is not declared in it.

302 EXAMPLE **302**

Program segment

```
BLOCK                SOURCE
  0   C
  0   C--MAIN PROGRAM
  1         COMMON /BLOCK/ C,D
  0   C     .
  0   C     .
  0   C     .
  6         CALL ASUB(U,V)
  0   C     .
  0   C     .
  0   C     .
  7         STOP
  1         END

BLOCK                          SOURCE

  1         SUBROUTINE ASUB(C,D)
  2         CALL BSUB(D)
  0   C     .
  0   C     .
  0   C     .
  3         RETURN
  1         END

BLOCK                          SOURCE

  1         SUBROUTINE BSUB(F)
  1         COMMON /BLOCK/ U,V
  0   C     .
  0   C     .
  0   C     .
  2         RETURN
  1         END
```

Diagnostic (printed after subroutine ASUB)

```
** 302 **  THE FOLLOWING COMMON BLOCKS, ALTHOUGH NOT EXPLICITLY IN
           SUBPROGRAM ---*ASUB*--, ARE AVAILABLE TO IT.

                COMMON BLOCK          AVAILABILITY

                --*BLOCK*--             ALWAYS
```

303 DESCRIPTION **303**

This diagnostic is associated with COMMON variables. All COMMON variables used are listed by COMMON block.

303 EXAMPLE **303**

Program segment

```
BLOCK                          SOURCE

   1          SUBROUTINE DSUB(A,B)
   1          COMMON /LAB/ L1,L2
   2          CALL DDSUB(A)
   0  C       .
   0  C       .
   0  C       .
   3          RETURN
   1          END

BLOCK                          SOURCE

   1          SUBROUTINE DDSUB(Z)
   1          COMMON /LAB/ M1,M2
   2          IF (M1 .LT. 1)
   3         $ RETURN
   4          M1 = M2 - 1
   0  C       .
   0  C       .
   0  C       .
   5          RETURN
   1          END
```

Diagnostic (printed after subroutine DSUB)

```
** 303 ** THE FOLLOWING DATA FLOW OCCURS THROUGH COMMON WHEN SUBPROGRAM
        --*DDSUB*-- IS CALLED.

   COMMON
    BLOCK        VARIABLE    INPUT CLASSIFICATION   OUTPUT CLASSIFCATION
    -----        --------    --------------------   -------------------

  ---*LAB*---   ----*L2*---        INPUT                   NON
  ---*LAB*---   ----*L1*---        STRICT                  OUTPUT
```

304 DESCRIPTION **304**

This diagnostic is associated with each program unit. It gives the input and output classification for every global variable for the unit.

304 EXAMPLE **304**

Program segment

```
BLOCK                          SOURCE

  1          SUBROUTINE ESUB(X,Y)
  1          COMMON /COMBLK/ J,K
  2          IF (K .LT. 0)
  3         $ X = Y + 1.0
  4          Y = 2.0
  5          J = J + 1
  6          RETURN
  1          END
```

Diagnostic

```
** 304 ** I/O CLASSIFICATION OF ARGUMENTS AND COMMON VARIABLES
          FOR SUBROUTINE ---*ESUB*--

             ARGUMENTS
                  POSITION      NAME      INPUT CLASS      OUTPUT CLASS

                     1       ----*X*----       NON            OUTPUT
                     2       ----*Y*----      INPUT           STRICT

             COMMON BLOCK      --*COMBLK*-
                  AVAILABILITY = ORIGINAL

             ARGUMENTS
                  POSITION      NAME      INPUT CLASS      OUTPUT CLASS

                     1       ----*J*----     STRICT           STRICT
                     2       ----*K*----     STRICT             NON
```

2.6 Diagnostics Emanating from PHASEO

Sections 2.1 through 2.5 discuss the output produced when DAVE terminates after execution of PHASE3 -- the standard termination. However, certain user errors are detected by PHASEO and cause termination at that point. The first action performed by PHASEO is to read the user options file (cf. section 1). If an error in expressing the options is encountered, the following diagnostic is issued and PHASEO aborts the job.

```
ERROR IN USER OPTIONS--CONSULT USER MANUAL
        --USER OPTIONS FILE--
      <listing of options file>
```

The remaining diagnostics produced by PHASEO are mainly due to non-ANSI constructions found in the subject program. PHASEO processes the entire subject program and prints diagnostics as errors are encountered. If any errors are found, PHASEO aborts the job. As in the output for an abnormal termination of a DAVE run discussed in section 2.1, the first line of the output file will read:

```
DAVE LEVEL ....
```

where the dots stand for a level number. After this will appear the line:

```
FATAL ERROR(S)--DAVE ANALYSIS CANNOT CONTINUE
```

Then follow diagnostics of the form:

```
ERROR FOUND BY PHASEO ANALYSIS IN SUBPROGRAM XXXXXX
<statement in error>
*****<error message>*****
```

The possible <error message>'s and their meanings are:

1. MAXIMUM NUMBER OF CONTINUATION LINES EXCEEDED

   19 continuation lines is the maximum allowable.

2. PROGRAM ERROR

   A valid ANSI FORTRAN statement has caused an error within PHASEO; this is a DAVE system error.

3. INVALID CHARACTER ENCOUNTERED

   A character other than a digit or blank was found in column 1-5.

4. LOGICAL IF STATEMENT FOUND AS DO LOOP TERMINATOR

   The user may add a CONTINUE statement to avoid this error.

5. INCORRECT OR UNKNOWN STATEMENT TYPE ENCOUNTERED

   This is a Non-ANSI statement or a statement which should not occur in this position in the code.

6. DO LOOP NESTING HAS REACHED LIMIT

   The current limit on nesting is 30.

7. INVALID DO STATEMENT SYNTAX

   This is an incorrectly formed DO statement.

8. IMPROPERLY FORMED LOGICAL IF STATEMENT

   This is an incorrectly formed IF statement.

9. EXPANDED LOGICAL IF STATEMENT MAY EXCEED MAXIMUM NUMBER OF CONTINUATION LINES

   The expansion of a logical IF statement exceeds 19 continuation lines; the user must rewrite the statement into a series of less complex statements.

10. BLOCK NUMBER HAS EXCEEDED MAXIMUM

    The max block number is 9999. When more blocks are encountered, the block number is reset to 1.

11. AMBIGUOUS PROGRAM ENTRY POINT ENCOUNTERED

    Two or more main programs were found.

12. END STATEMENT IS INCORRECTLY TERMINATING ROUTINE

    Execution may fall through to end statement.

13. DO LOOP TERMINATOR NOT FOUND

    A DO loop terminator was not found.

14. MAXIMUM NUMBER OF DIMENSIONED VARIABLES EXCEEDED

    The maximum allowable is set at 50 dimensioned variables.

15. KEYWORD IN DECLARATION IS MISSPELLED OR UNRECOGNIZABLE

    This statement is interpreted as a declaration statement but is misspelled or unrecognizable.

16. EMPTY SUBJECT PROGRAM FILE

    The file specified as containing the subject program is empty.

Chapter III: DAVE's INTERNAL STRUCTURE

1. Major Software Components

DAVE is composed of four separate phases, which are executed consecutively. These are referred to as PHASE0, PHASE1, PHASE2, and PHASE3. Each phase performs a segment of the analysis of the subject program and passes the information gathered to the next phase for further processing. Briefly, PHASE0 performs preliminary analysis of the subject program, PHASE1 builds a data base for each program unit comprising the subject program, PHASE2 performs the data flow analysis, and PHASE3 outputs the results for the user.

The entity used to create and access the data bases is referred to as the data base system and is composed of an initialization package called DBINIT and a package of library routines called DATAB. DBINIT is usually run only once--at the time DAVE is installed on a particular machine. A major change in the size of the data bases used by DAVE, as explained in section 5.1 of this chapter, would necessitate a re-configuration of the data base system, accomplished by re-running DBINIT. The routines comprising DATAB are used by PHASE1, PHASE2 and PHASE3 to build and access the data bases for the subject program.

A full description of the data base system may be found in [3].

PHASES 0-3 are described in more detail below.

I. PHASE0.

The subprograms comprising PHASE0 are:

BRNANL	EXTRA	INPBUF
BLOCK	FETNAM	PEND
ADARRN	FINDEX	PLCALL
CLASS	GETARR	PRINTL
CNBLKN	IADVNP	PSHDOL
DAVOPT	ICHKAN	PUTNHS
CARD	ICHRCK	RLABEL
DETIFH	ICONVT	RMDOLB
DETIFT	IDOTRM	
ERROR	IDTFUN	
ERRPT	INCBLK	
EXPLIF	INIT	

BRNANL is the driver for PHASE0, which numbers the blocks of the subject program and classifies the statements according to type. All declarations and the END statement are assigned to block 1 and all

executable statements are a single block except for logical IF statements which form two blocks; comments and FORMATs are designated block 0. PHASE0 produces a file, which is passed to PHASE1, containing the subject program with the block number, line number, statement type code, and number of continuation lines for each statement. Non-ANSI constructions are flagged as errors and cause termination of the DAVE run by PHASE0 after it has processed the entire subject program. The user options file is checked by Subroutine DAVOPT at the outset of PHASE0 and causes termination immediately if an error is found there.

II. PHASE1.

The subprograms comprising PHASE1 are:

DRIVE1	SYMTB	GETOK	STMTB	CONTOK
	PROIMP	FNDSM	ADD	FLTCHR
BLOCK	XTRNL	ADDST	SEVEN	NAMCHR
	ONEVR	STRDM	DLOOP	CHR
TOKLST	DATYP	COMPLT	RDWR	INTCHR
SETTLP	SUBFN	EQCHN	FCALL	CHAR
STMTIN	DATAS	EQCHK	PUSH	ICONVT
PHASE1	EQUIV	EQADD	ASIGN	
SKIP	COMN	EQMRG	VARDM	CGRAPH
LUNTIL	FNDCM	EQOFF		GRORDR
EXPRSN	PRORW	COMOFF	MKLBTB	CKGRPH
AC1	PRACI	NOFF	PUTLB	LKCLRS
AC2	PRODM			PRTGR
AC3				UPIT
AC4				CMDOWN
AC5				
AC6				
NEXTNB				
LASTNB				
PUTTOK				
ISTYPE				

DRIVE1 is the driver for PHASE1, which processes each subprogram unit in the subject program line-by-line, building concurrently the Symbol Table, Label Table, Statement Table, Common Block Table for each unit and the call graph. The tables reside in the unit's data base area, which is written out to a file when the end of the subprogram is reached. The in-core data area is then used for the next subprogram unit's tables. Subprograms TOKLST through ISTYPE above build the token list for the current line of the subject program unit being analyzed. Subprograms SYMTB through NOFF build the Symbol Table and Common Block Table. Subprograms STMTB through VARDM build the State-

ment Table and MKLBTB and PUTLB build the Label Table. CONTOK through ICONVT do character-to-numeric conversions. NAMCHR, in particular, converts symbolic names into their corresponding base 37 representations, i.e.

A→1
B→2
⋮
Z→26
0→27
⋮
9→36
blank→0

Subprograms CGRAPH through CMDOWN build the call graph. The data bases, call graph, and other information compiled by PHASE1 are then passed to PHASE2.

III. PHASE2.

The subprograms comprising PHASE2 are:

DRIVE2	INPVAR	BLKTAB
	OUTVAR	FEXST
BLOCK DATA	IODRVR	VARPR
	IOINIT	IOVPNT
DUMYIN	IBBLT	LABBLK
	PACKB	BLKPR
SBMTCH	OKFPUS	ENTBLK
PRMTCH	TYPE2	TRANS
STEPAR	CKUNDF	EAXBLK
IOSTAT	CLEAN	TSTTYP
COMTCH	PATH	LEXST
NXCM	ENPATH	IOVUND
CLSTMT	SAVEIO	SKIPCF
ASFIO	MVSUB	
CONSBD	BSUBTB	NOFF
BDMAIN		
COPYCM		

DRIVE2 is the driver for PHASE2, which produces the diagnostics about the subject program while processing its units in a leafs-up order. Subroutine DUMYIN is called if there are missing subprograms (i.e. unsatisfied externals) in the subject program and the user has specified that he wishes them to be dummied in so that processing may continue. For function subprograms, the function name is assumed to be non-input strict output and any parameters are assumed to be strict input, non-

output. For subroutine subprograms, parameters are also assumed to be strict input, non-output. The number of parameters and their dimensions are inferred from the first invocation of the missing unit in the subject program.

Subprograms SBMTCH through COPYCM perform matching of actual and dummy parameters and COMMON variables in external calls, checking for possible sources of error. The input/output classifications of global variables in a unit are passed up to the calling subprogram at this time.

Subprograms INPVAR through BSUBTB perform the input/output classification of a subprogram unit's variables, checking for possible sources of error, and entering the I/O information in the Subprogramwide Table for the unit.

Subprograms BLKTAB through SKIPCF build the Block Table, which contains the entry and exit blocks, transition codes, and I/O variables for each block in a subprogram unit. The Block Table is used by the INPVAR-BSUBTB group when performing the I/O analysis.

NOFF is a utility routine.

IV. PHASE3.

The subprograms comprising PHASE3 are:

DRIVE3

BLOCK

C37T11
C37TOA
CMPRES
CONVT
DIAGS
ERSTOR
GROUP
GTSYS
NPACKP
NUMCHR
PACKP
PRPATH
PRTGR
PSUBTB
REORD
SORTL
SOURCE
STOUT

DRIVE3 is the driver for PHASE3, which interprets the diagnostics produced by PHASE2 and produces the DAVE output described in Chapter II,

Section 2. The program units of the subject program are printed in the order in which they were submitted to DAVE, with each unit being followed by the errors, warnings and messages pertaining to it. ERSTOR builds a data base to hold the diagnostic information so that it may be easily retrieved. STOUT prints the header information and summary statistics and PRTGR prints the call graph. SOURCE prints each subprogram unit of the subject program, and DIAGS and GROUP print the error, warning, message diagnostics pertaining to the unit.

2. Tables Generated

These tables are built and accessed via the data base system. See the data base report [3] for a description of the data structures employed.

I. SYMBOL TABLE

The name of the symbol table is ISYMTB. It is a sequential table, each node consisting of nine fields:

ISNAME: This field contains the coded representation of the name (base 37 integer)

ISDTYP: This field contains a code number identifying the data type as follows:

0 typeless
1 integer
2 real
3 double precision
4 complex
5 logical

ISNTYP: The field contains a code number identifying the name type as follows:

1 scalar
2 array
3 arithmetic statement function
4 intrinsic or basic external built in function
5 procedure name

ISNDIM: This field contains the header for a linked list of information about the dimensioning of a variable when ISNTYP = 2. Otherwise this field is empty. Details of the linked list are below.

ISPARX: If this symbol is a formal parameter, then its position in the calling sequence (1,2,...) is in this field. If this symbol is a function subprogram name then this field contains 0. Otherwise this field is empty.

ISDATV: If this symbol is assigned a value in a data statement then a pointer to the data statement is in this field.

ISCOMN: If this symbol appears in a common statement then this field contains a pointer to the common table. For example, if this symbol is contained in the named common block XXXX and if this common block occupied the fourth sequential position in the common table then ISCOMN = 4. If this symbol does not appear in a common block then this field is empty.

ISEQUV: If this symbol appears in an equivalence statement, this field contains a pointer to the entry for this symbol in a linked list of variables to which it is equivalenced, called the equivalence chian. If this symbol does not appear in an equivalence statement then this field is empty. The equivalence chain is described below.

ISSTMT: This field is a header for a linked list of the numbers of the statements in which this symbol appears. Details of the linked list are below.

ISVOFF: This field contains the offset in storage locations of a COMMON variable in its block.

The entries in the linked lists of dimension information have node type IDMNØD and contain two fields:

ISDMVL: This is the value field: the first node's value is the number of dimensions {1,2, or 3}; successive nodes have the values appearing in each subscript position. See example below.

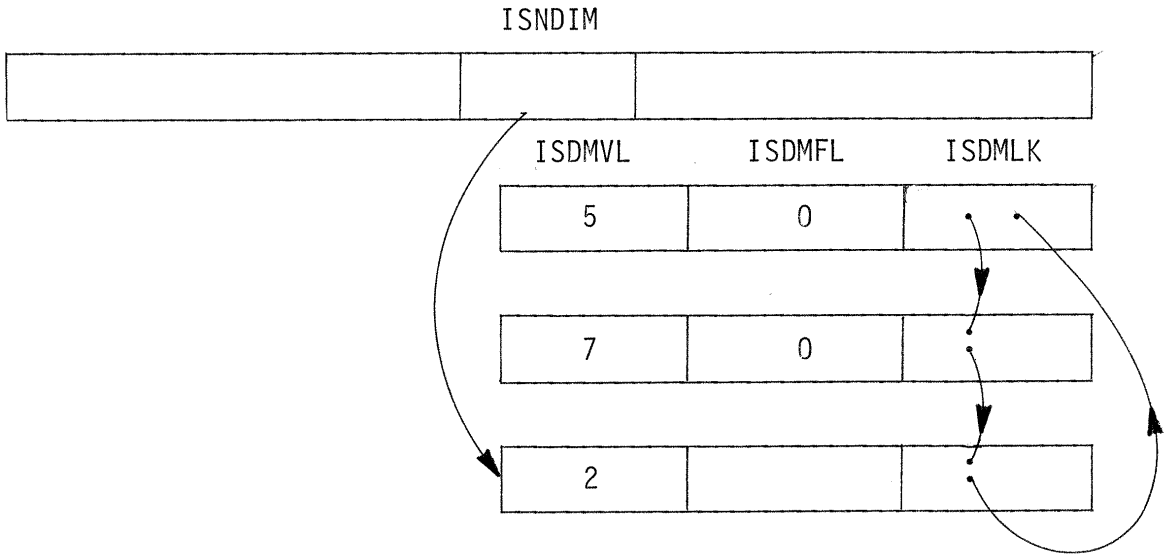
ISDMFL: This is a field containing a flag to differentiate between an integer valued dimension and a variable dimension.

ISDMLK: This is the link field. For example, the dimension statement

DIMENSION XAMPL (5,7)

would lead to this structure

XAMPL's entry in Symbol Table:



The entries in the linked lists of numbers of statements have node type ISTNØD and have two fields:

ISSTVL: This field contains a pointer to the sequential entry in the statement table of a statement containing this symbol.

ISSTLK: This is the link field.

Equivalenced variables are chained together in a linked list. For a variable in an equivalence list, the field ISEQUV of its Symbol Table entry contains a pointer to its entry in its equivalence chain (which is a linked list).

Nodes in the chains are of type ISEQND with the following fields:

ISEQVL: Contains a pointer to the variable's Symbol Table entry.

ISEQOF: Contains the offset number, to indicate the position of the first location of the variable relative to all the others in the chain. The number of storage locations a variable type requires is needed in this calculation.

ISEQCH: Contains the equivalence chain number.

ISEQLK: Link field.

II. STATEMENT TABLE

ISTMTB is the name of the sequential table in the data base which contains information about the statements of a Fortran subprogram.

Nodes in ISTMTB have the following fields:

LINOST: Line number on which statement begins.

IBLKST: Basic block number in which statement occurs.

ITYPST: Type of statement.

IIVHST: Header for a linked list of input variable information for a statement; nodes in this list are of type IIVNOD with fields:

IIVNDX: Variable's index in the Symbol Table.

IIVTYP: Contains an integer in the set {0, 1, 2, 3} where

IIVTYP = { 0 type uncertain
1 type input
2 type strict input
3 type non-input

IIVLNK: Link field in IIVNOD.

IOVHST: Header for a linked list of output variable information for a statement; nodes in this list are of type IOVNOD with fields:

IOVNDX: Variable's index in the Symbol Table.

IOVTYP: Contains an integer in the set {0, 1, 2, 3, 4} where

IOVTYP = { 0 type uncertain
1 type output
2 type strict output
3 type non-output
4 type undefined

IOVLNK: Link field in IOVNOD.

IXTRNL: Headed for a linked list of external references; nodes in this list are of type IXTNOD with fields:

IXTNAM: Contains pointer to name of external in Symbol Table.

IXTLNK: Link field in IXTNOD.

IXTPAR: Contains pointer to linked list of parameters; nodes in this list are of type IPRNOD with fields:

IPRTYP: Contains integer code for the parameter type

Code number	Meaning
1	expression
2	identifier
3	integer constant
4	real constant
5	double precision constant
7	complex constant
9	logical constant
10	Hollerith string
33	procedure declared external
100+Symbol Table pointer to ASF name	argument in arithmetic statement function definition used as parameter in procedure call

IPRNAM: Contains pointer to entry in Symbol Table of parameter name when relevant; when IPRTYP>100, this field contains the position of the argument in the dummy argument list

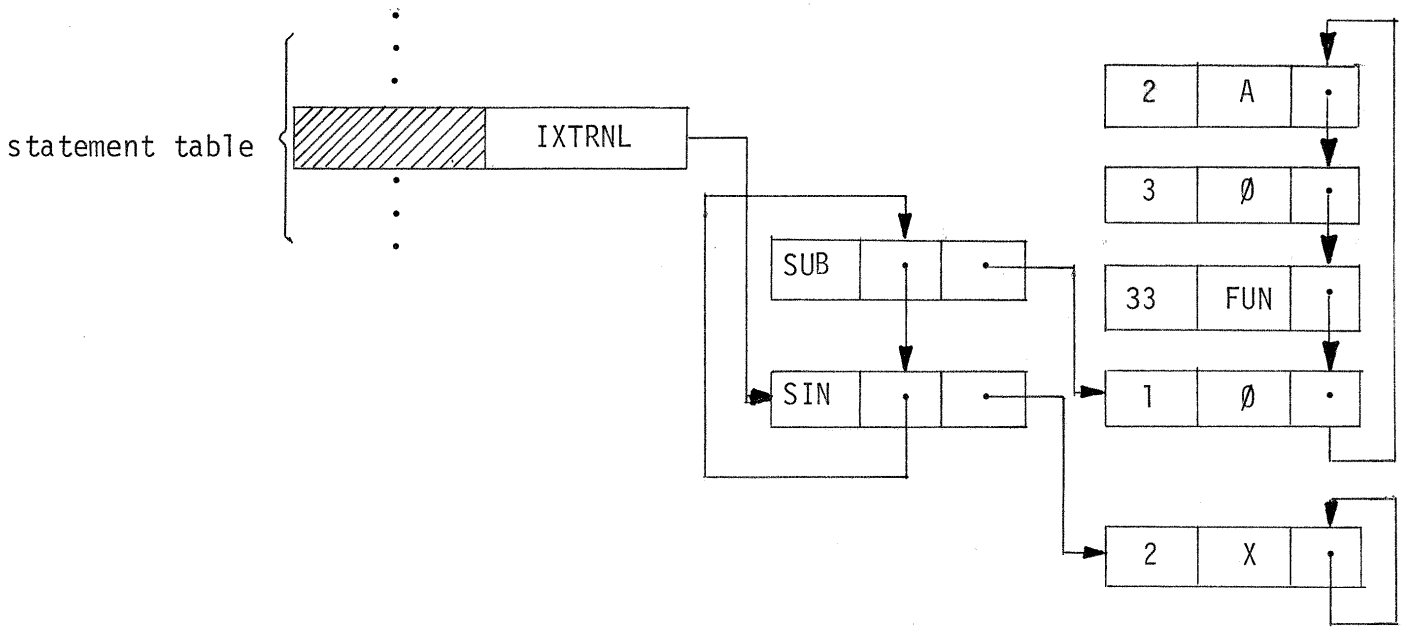
IPRLNK: Link field in IPRNOD

Example of list of external references:

The statement

CALL SUB(A, 5, FUN, SIN(X))

will generate the following structure:



III. COMMON BLOCK TABLE

The Common Block Table (ICOMTB) is a sequential table built in the course of constructing the Symbol Table. For a variable in common, the field ISCOMN of its Symbol Table entry contains a pointer to an entry in the Common Block Table.

Nodes in ICOMTB consist of four fields:

ILABCM: Label of common block as encoded base 37 integer; for blank common, this is the encoding for six blanks.

IVARCM: Header to linked list of pointers to Symbol Table entries which are located in that common block. The entries (node type ICOMND) in this linked list have three fields:

ICOMVL: pointer to the variable's Symbol Table entry

ICOMNM: Contains base 37 name of the symbol for identifiers in blocks added to the table as sometimes or always

available. For identifiers in those blocks contained in the source code, this field will be empty.

ICOMLK: Link field.

ICOMAV: Contains a code in the set {1,2,3} indicating whether the block is:

(1) in the subprogram's source code.

(2) always available to the subprogram.

(3) sometimes available to the subprogram.

ICMCOP: Contains the base 37 name of the subprogram from which the block's description is copied; this is meaningful only for those blocks added to the table as sometimes or always available.

For those identifiers in common blocks which are added to the Common Block Table as sometimes or always available, entries will be made in the Symbol Table. However, field ISNAME will be left empty. Information will be entered in fields ISDTYP, ISNTYP, ISNDIM and ISCOMN as usual.

IV. BASIC BLOCK TABLE

The basic block table contains the following information for each block:

1. The first statement in the block;
2. The last statement in the block;
3. List of input variables for the block;
4. List of output variables for the block;
5. List of entry blocks for the block;
6. List of exit blocks for the block.

It is assumed that the basic blocks of the subprogram unit are numbered sequentially (1, 2, ...) and the nodes in the sequential basic block table are in this order: the first entry in the table is for basic block 1, the second for basic block 2, etc.

The name of the basic block table is IBLKTB. It is a sequential table, each node consisting of six fields:

IFRSTM: This field contains a pointer to the statement table at the entry for the first statement of the block.

ILSSTM: This field contains a pointer to the statement table at entry for the last statement of the block.

IIVBLK: This field is a header for the linked list of input variables of the block.

IOVBLK: This field is a header for the linked list of output variables of the block.

IEBBLK: This field is a header for the linked list of entry blocks for the block.

IXBBLK: This field is a header for the linked list of exit blocks for the block.

The linked list of input variables for the block has one node for each input variable. Each node contains three fields. These items are specified as follows:

IIVNOD: Node type name for nodes in this list.

IIVNDX: This is a field containing a pointer to the node in the symbol table for this variable.

IIVTYP: This is a field containing an integer in the set {0,1,2,3,4} where

IIVTYP = 0 means uncertain if input
IIVTYP = 1 means input
IIVTYP = 2 means strict input
IIVTYP = 3 means non-input
IIVTYP = 4 means undefined input

IIVLNK: This is a link field

The linked list of output variables for the block has one node for each output variable. Each node contains three fields. The names of these items are: IOVNOD, IOVNDX, IOVTYP, IOVLNK with meanings corresponding to those above, mutatis mutandis.

The linked list of entry blocks for the block has one node for each entry block. Each node contains five fields. These items are specified as follows:

IEBNOD: Node type name for nodes in this list.

IEBNDX: This is a field containing a pointer to the node in the basic block table for this entry block.

IETRAN: This field contains the transfer code from this entry block (IEBNDX) to the current block. The transfer codes are listed below.

IEUNDF: This field is a header to a linked list of variables which become undefined during the transfer from this entry block to the current block.

IEBLNK: This is a link field.

The linked list of exit blocks for the block has one node for each exit block. Each node contains five fields. The names of these items are: IXBNOD, IXBNDX, IXTRAN, IXUNDF, IXBLNK with meanings corresponding to those above, mutatis mutandis.

The linked list of DO-parameters which become undefined during a transfer has one node for each parameter. Each node contains two fields.

IDOPAR: Node type name for nodes in the list.

IDOUND: This field contains the Symbol Table pointer for this DO-parameter.

IDOLNK: This is a link field.

The transfer codes for a transfer from block A to block B are:

1. Unconditional Transfer (an unconditional GØ TØ or a normal drop through)
2. Conditional Transfer (true branch of a logical IF)
3. Conditional Transfer (false branch of a logical IF)
4. Do-loop fall through
5. Do-loop iteration
6. Assigned GØ TØ
7. Computed GØ TØ
8. The transfer is undetermined. The last executable statement in block A will be examined.

V. LABEL TABLE

The name of the label table is ILABTB. It is a sequential table, each node consisting of 3 fields:

LABELV: This field contains the label as an integer value

IEXLAB: This field contains the statement number of the statement that uses the label as an external label.

INLABH: This field contains the header to the internal label list. This list contains the statement numbers of all statements that use the label as an internal label.

INTLST is the name of the nodes on the internal label list. Each node consists of 2 fields:

INTLAB: This field contains the statement number of the statement that uses the label as an internal label.

NXTLAB: This field contains a pointer to the next node of the internal label list.

VI. SUBPROGRAMWIDE TABLE

The sequential table ISUBTB contains information about an entire subprogram unit and has one entry, or in the case of the master data base, has an entry for each subprogram unit.

Each entry consists of three fields:

NAMESP: The base 37 representation of the name of a subprogram. This field contains a "0" for the main program and is empty for BLOCK DATA.

ISBERS: A header to a linked list of all external subprograms referenced by this one. The list consists of IERFND nodes.

IPLSTS: A pointer field pointing to a list of lists of parameters and COMMON variables for this subprogram and all of its arithmetic statement functions. IPLSTS points to a linked list whose nodes are of type LSTPRN. The first node of this list is to be assumed to be the header for the list of parameter variables to the entire subprogram. Subsequent nodes are to be the headers for the various COMMON blocks referenced and finally for the parameter lists for the various ASF's in the subprogram.

Nodes:

1. LSTPRN: Contains information about the lists of parameters, common variables and ASF's for a subprogram unit.

   Each node consists of six fields:

   INAMLS: The base 37 representation of the list name,i.e. subprogram name, COMMON block name, or ASF name.

   IAVAIL: Contains a code indicating whether the common block is in the unit's source code, is always available, or is sometimes available {1,2,3}.

   IOSTLP: Pointer to I/O status list composed of IOSTND nodes.

   INTYCD: Contains code (statement type code) to indicate type of name in INAMLS.

   ICMORG: Contains, for common blocks which are sometimes or always available to the subprogram the base 37 name of the subprogram from which the block's description was copied.

   LSTLNK: Link field for LSTPRN nodes.

2. IOSTND: Contains information about the parameters, common variables, and ASF parameters in a particular list.

   Each node consists of seven fields:

   IOVNME: The base 37 representation of the particular variable's name.

   IDATYP: The data type code of this variable using the Symbol Table code values.

   IOVDLP: A pointer field pointing to a linked list containing the number of dimensions and maximum allowable subscript value for each dimension, in turn. This field will contain an empty for scalar variables. Otherwise, the linked list pointed to will consist of IDMNOD nodes, described in the Symbol Table documentation.

   IOVSUN: An integer field whose value will be the total number of storage units occupied by this entity. Thus, for example, the IOVSUN field for an $i \times j \times k$ integer array should contain ijk. A double precision array dimensioned at $i \times j \times k$ will, however, occupy $d \cdot ijk$ storage units.

   INPCLS: The input category for a variable, where

   1 means input
   2 means strict input
   3 means non-input

   IOUCLS: The output category for a variable, where

   1 means output
   2 means strict output
   3 means non-output

   IOLLNK: Link field for IOSTND nodes.

3. IERFND: Contains the names of the external references for this unit.

   Each node consists of two fields:

   IERNME: The base 37 representation of the name of an external subprogram referenced by this program unit.

   IEFLNK: Link field for IERFND nodes.

VII. DIAGNOSTIC TABLE

The name of the table containing the DAVE-generated diagnostics for subprogram units is IERRTB. There is one entry per unit processed, each entry having three fields:

IERPTR: Pointer to a linked list of errors for this unit; the list contains IDIAND nodes.

IWAPTR: Pointer to a linked list of warnings for this unit; the list contains IDIAND nodes.

IMEPTR: Pointer to a linked list of messages for this unit; the list contains IDIAND nodes.

The IDIAND nodes contain five fields:

NUMFLD: Contains the diagnostic number.

IARGNM: May contain a base 37 name concerning this diagnostic.

IBLKNM: May contain a block number for this diagnostic.

LISPTR: Pointer to a linked list of IPATND nodes containing additional information for the diagnostic; last IPATND node contains the number of nodes in the list.

IDIALK: Link field for IDIAND nodes.

The IPATND nodes contain two fields:

IBLNUM: Contains a piece of information concerning the diagnostic; for paths, the block numbers are packed in IPATND nodes, with the next to last node containing the length of the path.

IPATLK: Link field for IPATND nodes.

3. File Organization

The following is a list of the input and output files for each of DAVE's execution phases. See Chapter IV for additional information on the structure of the files employed.

I. PHASE0

INPUT FILES

IN: Contains FORTRAN source program to be analyzed.

OPTIONS: Contains user options.

OUTPUT FILES

SOURCE: Contains subject program and each statement's type, line number, block number, and number of continuation lines.

OUTPUT: Contains error information in case of non-ANSI constructions in subject program.

II. PHASE1

INPUT FILES

COMDAT: Contains data base initialization values (produced by DBINIT run).

SOURCE: Passed from PHASE0.

OUTPUT FILES

CALLGR: Contains call graph for subject program and information on the format of the file of data bases.

ORDER: Contains the processing order for the subprogram units.

ROLL: Contains a data base for each subprogram unit; each data base contains a Symbol Table, Statement Table, Common Block Table, Label Table, Block Table, and Subprogramwide Table.

PAGEI: Paging file used (if necessary) by each data base when it is being built.

OUTPUT: Contains diagnostic information in case of a DAVE execution error; this file is copied to another file ERRS through the job control language after completion of PHASE1 so that its contents will not appear first in the listing of DAVE output.

STATUS: Contains a code indicating the status of completion of PHASE1, i.e. whether it terminated normally, or abnormally through ERRSUB in the case of a DAVE execution error.

DIAG: Contains diagnostic information gathered by DAVE about the subject program. The only diagnostic found by PHASE1 is that of missing subprograms with the simulation option off.

MODSRC: Contains the modified subject program; it is used by PHASE3 for printing the FORTRAN source.

III. PHASE2

INPUT FILES

COMDAT: Contains data base initialization values.

CALLGR: Passed from PHASE1.

ORDER: Passed from PHASE1.

ROLL: Passed from PHASE1.

PAGEI: Paging filed used (if necessary) by each data base when it is in core.

STATUS: Contains a code indicating the status of completion of PHASE1.

DIAG: Passed from PHASE1.

OUTPUT FILES

PAGEI: Same as above.

OUTPUT: Contains diagnostic information in case of a DAVE execution error; this file is copied to file ERRS through the job control language after completion of PHASE2 so that its contents will not appear at this point in the listing of DAVE output.

STATUS: Contains a code indicating the status of completion of PHASE2.

DIAG: Contains diagnostic information gathered by DAVE about the subject program.

DBUTIL: Contains data base utilization information. It may be helpful to list this file in case of abnormal termination of DAVE.

MASTR: Contains the master data base array (copied from its in-core location.)

PAGEM: Paging file for the master data base.

IV. PHASE3

INPUT FILES

COMDAT: Contains data base initialization values.

CALLGR: Passed from PHASE2.

PAGEI: Paging file for diagnostic data base.

STATUS: Contains PHASE1-PHASE2 completion status code.

DIAG: Passed from PHASE2.

MODSRC: Passed from PHASE1.

MASTR: Passed from PHASE2.

PAGEM: Passed from PHASE2.

OUTPUT FILES

OUTPUT: Contains information gathered by DAVE about the subject program; after completion of PHASE3, file ERRS is printed so that any internal DAVE diagnostics appear after what is printed in PHASE3.

4. System Dependencies

DAVE is usually tailored for the particular system on which it is to be installed before it is sent there. The following is a list of those system dependencies which must be altered.

I. DBINIT

A. The main program contains a PROGRAM card.

B. Common block GLOBAL, initialized in BLOCK DATA, contains machine dependent constants:

	CDC(60 bits/word)	IBM(32 bits/word)
MTYPRT	- 2000000B	Z00008000
MTYFUL	- 4000400000000000000B	Z80000000
IPTFLG	- 1000000B	Z40000000
NPTFLG	- 2777777B	Z3FFFFFFF
MSKFLD(1)	- 3777777B	Z0000FFFF
MSKFLD(2)	- 17777774000000B	ZFFFF0000
MSKFLD(3)	- 777777600000000000B	0
NBSPPW	- 20	16
NPWPFW	- 3	2

	DEC (36 bits/word)	UNIVAC (36 bits/word)
MTYPRT	- "400000	Ø400000
MTYFUL	- "400000000000	Ø400000000000
IPTFLG	- "200000	Ø200000
NPTFLG	- "177777	Ø177777
MSKFLD(1)	- "777777	Ø777777
MSKFLD(2)	- "777777000000	Ø777777000000
MSKFLD(3)	- 0	0
NBSPPW	- 18	18
NPWPFW	- 2	2

C. There is an octal format specification in format 99984 in Subroutine FINDAT.

II. DATAB.

A. The Functions IAND, IOR and INOT which perform bit-by-bit logical intersetion, union, and complement operations, have to be modified.

B. A FUNCTION ISHIFT(IWD,NBITS) which shifts word IWD by NBITS: left circular if NBITS>0 and right, end off with sign extension if NBITS<0, must be added.

C. There is an octal format specification in format 99996 in Subroutine GARTST.

D. There is an octal format specification in format 99993 in Subroutine ERRSUB.

E. In Subroutine ERRSUB there appears a call to Subroutine CALTRC, which prints out traceback information and should be replaced by the local equivalent:

```
40 CALL CALTRC
```

F. Subroutine RANRW contains calls to random access routines WRITMS and READMS:

```
CALL WRITMS(IFILE,IAREA(INDX),NUM,NREC,KEY2)
CALL READMS(IFILE,IAREA(INDX),NUM,NREC)
```

where IFILE = random access file number
IAREA(INDX)=starting address of words to be written out

NUM = number of words to be written out
NREC = record number to be written to
KEY2 = 0 if writing in place
= 1 if writing at end of information

III. PHASE0

A. There is a PROGRAM card in the main program.

B. In the main program and Subroutine DAVOPT, there are calls to Subroutine ABORT, which should terminate the DAVE job without returning control to the next control card. ABORT has no parameters.

C. In Subroutine DAVOPT at line 49 there is an end-of-file test.

D. In Subroutine INPBUF at line 46 there is an end-of-file test.

IV. PHASE1

In the main program there is a PROGRAM card. Also there are calls to OPENMS to open the random access files IDAB1 and IPUNIN and calls to CLOSMS to close them:

```
CALL OPENMS(IDAB1,INDEX,MXREC,0)
CALL OPENMS(IPUNIN,INDEXI,MAXPGS+1,0)

CALL CLOSMS(IDAB1,INDEX,MXREC)
CALL CLOSMS(IPUNIN,INDEXI,MAXPGS+1)
```

The variable MXREC=4*IORSZ+1=401 is the maximum number of records for file IDAB1 with record size IRECSZ, and MAXPGS=100 is the maximum number of records for file IPUNIN with record size IPAGSZ. For CDC, the index arrays, INDEX and INDEXI, are accordingly dimensioned INDEX(401) and INDEXI(101). The fourth argument to OPENMS indicates use of a numbered index for the random access file. The records in file IPUNIN are initialized so that henceforth they may be written to in place.

V. PHASE2

A. In the main program there is a PROGRAM card. Also there are calls to OPENMS to open the random access files IDAB1, IPUNIN, and IPUGMS, with number of records MXREC, MAXPGS, and MAXPGS and record sizes IRECSZ, IPAGSZ, IPAGSZ, repectively. For CDC the records of PAGEM must be initialized so that they can later be written to in place. CLOSMS is called to close IPUGMS. The index arrays for IDAB1, IPUNIN and IPUGMS are dimensioned INDEX(401),INDEXI(101) and INDEXM(101).

B. In Block Data the common block BBMASK is initialized:

IBMSK--a fullword mask with IBBIT zeros right justified.

IIMSK--a fullword mask with IBBIT ones right justified.

IBBPW--number of part word fields/word; dependent upon IBBIT and word size.

IBBIT--number of bits/part word; should be 6 bits if possible, else 8.

	for 60-bit word size	for 36-bit word size	for 32-bit word size
IBMSK	7777777777777777700B	Ø777777777700	ZFFFFFF00
IIMSK	77B	Ø77	Z000000FF
IBBPW	10	6	4
IBBIT	6	6	8

VI. PHASE3

A. In the main program there is a PROGRAM card. Also there are calls to OPENMS to open the random access files IPUNIN and IPUGMS, with number of records MAXPGS and record size IPAGSZ. Their index arrays are dimensioned INDEXI(101) and INDEXM(101).

B. In Block Data the common block BBMASK is initialized:

IBMSK--IBBIT zeros right justified.

IIMSK--IBBIT ones right justified.

IBBPW--number of part words/full word; dependent upon IBBIT and word size.

IBBIT--number of bits/part word; must be large enough to hold the maximum block number.

	for 60-bit word size	for 36-bit word size	for 32-bit word size
IBMSK	777777777777776000B	Ø777777777000	ZFFFF0000
IIMSK	1777B	Ø777	Z0000FFFF
IBBPW	6	4	2
IBBIT	10	9	16

Note: These values are different from those used in PHASE2.

C. In Subroutine SOURCE there are two end-of-file tests at lines 38 and 191.

5. Size Alterations that may be made to change DAVE's core requirements and the size of programs DAVE can analyze

5.1 Maximum data base size

The largest size a data base may become (through paging) is determined by the product of the maximum number of pages MAXPGS(=100) and the page size IPAGSZ(=512), whose values are set in DBINIT. If MAXPGS is changed, arrays dependent upon it must also be changed; these are indicated in 5.2 wherever MAXPGS appears as a dimension.

There are also values set in DBINIT which determine how much of this maximum size of 51200 may actually be used, namely the number of part word fields/word, NPWPFW, which is dependent upon the word-size of the machine and the number of bits per part-word, NBSPPW. Obviously, the more part word fields/word, the more economical the use of storage. However, the maximum data base size addressable is $2^{(NBSPPW-2)}$ so that for a 32-bit word size, with NBSPPW=16 and MAXPGS=32 the largest useable data base is 16384 words. In order to use a 51200 word capability the number of bits required for pointer fields is 18. Therefore, for IBM the data base structure would have to be reconfigured with pointers as full-word fields (with the implication of a less economical use of storage). To reconfigure the data base, it is necessary to change the values of the variables in common block GLOBAL, initialized in BLOCK DATA in DBINIT. DBINIT must then be re-run, creating a new COMDAT; INCDAT does not change. Then DAVE may be run as usual, with the new COMDAT. For IBM's 32-bit word size, the values of variables in GLOBAL for pointers occupying full words with MAXPGS=100 and for pointers occupying half words with MAXPGS=32 are:

	MAXPGS=100	MAXPGS=32
MTYPRT	Z00008000	Z00008000
MTYFUL	Z80000000	Z80000000
IPTFLG	Z40000000	Z00004000
NPTFLG	Z3FFFFFFF	Z00003FFF
MSKFLD(1)	Z0000FFFF	Z0000FFFF
MSKFLD(2)	ZFFFF0000	ZFFFF0000
MSKFLD(3)	0	0
NBSPPW	16	16
NPWPFW	2	2

For a data base size larger than 51200, MAXPGS must be changed and caution must be taken that $2^{(NBSPPW-2)} \geq$ data base size. If NBSPPW must be changed to achieve this, then the other variables in GLOBAL must also be changed accordingly.

5.2 Other Size Limitations

Note: ARRAY(ISIZE) indicates that to alter an array's size, you must change its DIMENSION statement and reset ISIZE in a DATA statement. The current sizes are indicated.

I. PHASE1

A. Driver

1. INDDB(INDSZ) - the in-core data base area for a single subprogram; most efficient if it is a multiple of the page size (IPAGSZ=512) and as large as core limitations will allow to avoid costly paging; INDSZ=3584.
2. IGRPH(IGRSZ) - the size of array IGRPH is dependent upon the number of subprograms being processed, the number of common blocks in each, and the number of subprograms called by each. There is an entry for each subprogram consisting of 6 words; each common block in a subprogram requires an entry of 3 words and each subprogram which it calls or is called by also requires a 3-word entry. IGRSZ=3000.
3. IORDR(IORSZ) - array IORDR must be dimensiond to the maximum number of subroutines that can be processed at one time. IORSZ=100.

The space that must be allocated for the random access file IDAB1 must be altered accordingly if a larger number of subroutines are to be processed together in order to accommodate the additional records. See I.A.5.

4. IDBTB(IORSZ,2) - contains information about the formatting of the individual subprogram data bases on file IDAB1.
5. INDEX(MXREC+1) - used by the CDC random access routines for the file of subprogram data bases (IDAB1); the number of records (MXREC) is variable due to the indeterminate sizes of the data bases because of paging; it can be estimated by dividing the average data base size by the record size (see I.B.1) and multiplying the result by the maximum number of subprograms that can be processed together (see I.A.3.) MXREC=4*IORSZ+1.
6. IDBAR(MAXPGS) - holds information, for a paged data base, about which pages are written out to the file of data bases (IDAB1), MAXPGS=100.
7. INDEXI(MAXPGS+1) - used by CDC random access routines for the paging file (IPUNIN) associated with a subprogram's data base (INDDB); MAXPGS=100.

B. Block Data
   1. IRECSZ - record size for file IDAB1; must be ≤ smaller of the in-core data base areas (INDDB and MSTDB) and a multiple of IPAGSZ, the page size; IRECSZ=2048.

C. Subprograms COMPLT, EQCHN, EQADD, EQUIV BLOCK DATA:
   COMMON/EQINF/IQLST(100,4),IQMAX,IQLN
   DATA IQMAX/100/ (set in BLOCK DATA)
   IQLST must have a row for each variable appearing in an EQUIVALENCE list plus an entry between lists. See D below and II.E.

D. Subprograms COMPLT, EQCHN, EQCHK, EQADD, EQMRG, EQOFF, COMOFF:
   COMMON ICHTB(100,4),ICNX
   ICHTB must be dimensioned the same as IQLST, see. I.C. and II.E.

E. Subprograms ASIGN, ADD, FCALL, PRACI

COMMON/ID1/IDUM(20),IDX,IDLOC

IDUM contains the dummy arguments for an arithmetic statement function. Its size is set in DATA statement

DATA IDLMT/20/

in PRACI and ASIGN.

II. PHASE2

A. Driver

1. INDDB(INDSZ
2. IGRPH(IGRSZ)
3. IORDR(IORSZ)
4. IDBTB(IORSZ,2)
5. IDBAR(MAXPGS)
6. INDEX(MXREC+1)
7. INDEXI(MAXPGS+1)

} must be the same as PHASE1

8. MSTDB(MSTSZ) - the in-core data base area for the master data base, which contains information about all the subprograms processed; same constraints as INDDB; MSTSZ=8704.
9. INDEXM(MAXPGS+1) - used by CDC random access routines for the paging file (IPUGMS) associated with the master data base (MSTDB).
10. IBIND(LENGTH),IPATH(LENGTH),ISTK(2*LENGTH) - LENGTH is the maximum number of blocks in any subprogram being processed. IBIND is also used by Subroutine SBMTCH as array IBUF; ISTK is used by Subroutine IOINIT as array LSTBF and by Subroutine SAVEIO as array IBUF. If LENGTH is adequate, these arrays will be also and don't normally need to be taken into consideration. LENGTH=500.
11. INTID(INIDL) - INIDL is the maximum number of global or local variables (each group is done separately) in any subprogram. Also used by Subroutine SBMTCH as array ICOM, which holds the common blocks available to a subprogram and their availability codes. INIDL=300.

B. Block Data

1. IRECSZ - same as PHASE1
2. Common IOBLK(IOBSZ),IOBSZ,IOBPTR,INNOD
   IOBLK appears in blank common in PHASE2 driver, Block Data, IBBLT,PACKB,INPVAR,OUTVAR,IODRVR,IOINIT,TYPE2, CKUNDF,CLEAN.

   It is equivalenced to the master data base MSTDB and has size roughly equal to the product of the number of blocks in a subprogram and (the number of global [also done for local] variables in the unit plus the average number of exit and entry blocks per block). IOBSZ=9000.

C. Subroutine VARPR

1. IBUF(IASIZE),ITEMP(IASIZE),ITYPE(IASIZE) - used by CPLIST for the list of I/O variables for a statement; nodes (one for each variable) consist of 3 part-word fields; in blank common. IASIZE=1200.

D. Subroutines SBMTCH,PRMTCH,STEPAR,IOSTAT,COMTCH,ASFIO,CONSBD, COPYCM contain array INOS in common block STNOS, which is used by CPLIST to hold the list of statements which reference an external; each node consists of two part-word fields; INOS(350).

E. Subroutine IODRVR

   IDISP(IELMT),IEQUV(IELMT) - contain equivalence information, must be same as I.C., I.D.

III. PHASE3

A. Driver

1. MSTDB(MSTSZ) - same as in PHASE2
2. INDEXI(MAXPGS+1) - same as in PHASE2
3. INDEXM(MAXPGS+1) - same as in PHASE2
4. IERDB(IERSZ) - data base containing diagnostics about subject program. IERSZ=8704.
5. IORDR(100) - same as in PHASES 1 and 2
6. IGRPH(3000) - same as in PHASES 1 and 2

7. LIST(500) - in blank common; must be dimensioned to the maximum number of blocks occurring in any subprogram unit processed or the total number of subprograms being processed, whichever is larger. This would have to be changed also in subprograms DIAGS, ERSTOR, PRTGR, REORD, STOUT. See also B.2 below.
8. IBUF(100,3) - in blank common; first dimension must be the same as dimension of IORDR. This would have to be changed also in subprograms DIAGS, ERSTOR, STOUT.

B. Block Data
1. IRECSZ - same as PHASE 1 and 2
2. IBBIT in common block BBMASK must be ≥ log (max. no. of blocks per subprogram + 1). IBMSK, IIMSK, IBBPW must be adjusted accordingly (see p. 149, VI.B).

6. Recovery

It is possible to modify DAVE so that in certain cases if an error occurs, the job may be re-run from the point of failure. These cases involve two types of errors which may be detected during PHASE1 of analysis: (1) syntax errors in the subject program and (2) unsatisfied externals when the simulation option is off. In these cases, the user can recover from the error by re-submitting the job; but this time the input to DAVE consists of only the unprocessed subprograms. In case 1, the revised input file is that portion of the original subject program from the subprogram being processed at the time of failure to the end of the file; and in case 2, it consists of the missing externals. DAVE could, therefore, pick up where it left off.

Please note, however, that an extra price is paid for the recovery capability in the form of additional I/O charges since large files must be saved between the two runs. Also, ANSI violations and unsatisfied externals may be found more economically by using tools such as Bell Laboratories' PFORT Verifier [4] prior to using DAVE. For these reasons, we do not encourage use of the recovery capability but will provide instructions for anyone interested.

Chapter IV: INSTALLING DAVE ON A COMPUTER

The party desiring to install DAVE should first complete and return the questionnaire, shown in Appendix B.

A tape containing DAVE's source code will then be prepared and mailed. This code will have been tailored to the particular system according to specifications obtained from the questionnaire. This tape will contain the following files:

1. DBINIT -- DAVE SOURCE CODE
2. NUDATA -- DATA FILE FOR DBINIT
3. DATAB -- DAVE SOURCE CODE
4. PHASE0 -- DAVE SOURCE CODE
5. PHASE1 -- DAVE SOURCE CODE
6. PHASE2 -- DAVE SOURCE CODE
7. PHASE3 -- DAVE SOURCE CODE
8. TESTFL -- TEST PROGRAM TO RUN THROUGH DAVE

STEP ONE. Read the tape and set up the files with the above names.

STEP TWO. Compile files 1, 3-7 and save the binary modules under the following names:

Source file name	Binary file name
DBINIT	DBINB
DATAB	DBLIB*
PHASE0	PH0B
PHASE1	PH1B
PHASE2	PH2B
PHASE3	PH3B

*File DBLIB should be made a library file.

STEP THREE. Run DBINB, the data base initialization package. This is only done once, its purpose being the creation of file COMDAT, which is used by DAVE during execution. The program requires the file NUDATA, assigned to unit 4, as input. Its output files are the printer, unit 6; INCDAT, unit 8; and COMDAT, unit 9. COMDAT must be saved.

At this point it is judicious to send the printed output from the data base initialization run to us at the University of Colorado for verification before proceeding.

STEP FOUR. EXECUTE DAVE. DAVE consists of four separate phases, which are run consecutively. The files used by each phase are listed below. The octal core requirements listed are for the CDC 6400.

A. PHASE0

INPUT FILES

Name	Unit No.	Contents
IN	5	Fortran Source program to be analyzed; maximum of 80 characters/record
OPTIONS	1	User options file; maximum of 80 characters/record

OUTPUT FILES

Name	Unit No.	Contents	Disposition
SOURCE	9	Formatted with 125-character records; no.of records=no. of lines in file IN	To be passed to PHASE1
OUTPUT	6	Maximum of 80 characters/ record; only written to in case of error	Printer

Core requirement: 40000

B. PHASE1

INPUT FILES

Name	Unit No.	Contents
COMDAT*	1	Unformatted; 8 records with maximum size of 812 words
SOURCE	9	Formatted with 125-character records; no.of records=no.of lines in subject program

*COMDAT was created in STEP THREE

OUTPUT FILES

Name	Unit No.	Contents	Dispoisition
CALLGR	2	Unformatted; 3 records, maximum size 3001 words	To be passed to PHASE2
ORDER	3	Unformatted; maximum of of 101 records, one word/ record	To be passed to PHASE2

Name	Unit No.	Contents	Disposition
ROLL	4	Random access file; maximum of MXREC=401 records (set in DRIVE1), maximum size IRECSZ=2048 (set in BLOCK DATA)	To be passed to PHASE2
PAGEI	5	Random access file; maximum of MAXPGS=100 records, maximum size IPAGSZ=512 (both set by DBINIT)	To be passed to PHASE2
OUTPUT	6	Only written to in case of DAVE execution error, maximum of 80 characters/record	Printer
STATUS	7	Unformatted, one word	To be passed to PHASE2
DIAG	8	Unformatted, no.of records dependent upon no.of diagnostics issued about subject program, maximum size 504 words, average size 50-100 words	To be passed to PHASE2
MODSRC	10	Unformatted; no.of records= no.of statements in subject programs, maximum record size 1331 words, average size 80 words	To be passed to PHASE3

PHASE1 requires the library DBL1B to satisfy external references during loading.

Core requirements 114000

C. PHASE2

INPUT FILES

Name	Unit No.	Contents
COMDAT	1	Same as in B above
CALLGR	2	Same as in B above
ORDER	3	Same as in B above
ROLL	4	Same as in B above
PAGEI	5	Same as in B above
STATUS	7	Same as in B above
DIAG	8	Same as in B above

OUTPUT FILES

Name	Unit No.	Contents	Disposition
PAGEI	5	Same as in B above	To be passed to PHASE3
OUTPUT	6	Same as in B above	Printer
STATUS	7	Same as in B above	To be passed to PHASE3
DIAG	8	Same as in B above	To be passed to PHASE3
DBUTIL*	11	Formatted; 24 records per subprogram analyzed, 83 words/record	To be copied to output in case of abnormal termination
MASTR	12	Unformatted; maximum 25 records, maximum 2048 words/record	To be passed to PHASE3
PAGEM	13	Random access file; maximum of MAXPGS=100 records, maximum size of IPAGSZ=512 (both set by DBINIT)	To be passed to PHASE3

*DBUTIL contains data base utilization information useful only to the DAVE maintainer. It may be helpful in cases of abnormal termination.

PHASE2 requires DBLIB to satisfy externals.

Core requirement: 121000

D. PHASE3

INPUT FILES

Name	Unit No.	Contents
COMDAT	1	Same as in B above
CALLGR	2	Same as in B above
PAGEI	5	Same as in B above
STATUS	7	Same as in B above
DIAG	8	Same as in B above
MODSRC	10	Same as in B above
MASTR	12	Same as in B above
PAGEM	13	Same as in B above

OUTPUT FILES

Name	Unit No.	Contents	Disposition
OUTPUT	6	Maximum record size of 80 characters; all DAVE analysis is printed in PHASE3	Printer

PHASE3 requires DBLIB to satisfy externals.

Core requirement: 135000

STEP FIVE. Set up a procedure file to execute DAVE. The following is a prototype procedure file for the CDC 6400 under KRONOS 2.1, with an explanation of each job control command. This procedure file, named DAVE, is invoked by the statement:

CALL(DAVE(OPTIONS=FILE1,INPUT=FILE2)

where FILE1 is the file containing the user options and FILE2 is the file containing the Fortran subject program (see section on how to execute DAVE). The names OPTIONS and INPUT are formal parameters to the procedure file; FILE1 and FILE2 are the actual parameters.

JOB CONTROL STATEMENT	EXPLANATION
GET,PHOB,PH1B,PH2B,PH3B,DBLIB,COMDAT.	Get the files necessary for DAVE's execution.
RFL,40000.	Set the field length for PHASE0.
PHOB(OPTIONS,INPUT)	Load and execute PHOB. OPTIONS and INPUT are parameters to the procedure file. OPTIONS indicates the name of the file containing the user options for this run and INPUT indicates the name of the file containing the Fortran subject program. These files appear in the first two positions on the program card for PHASE0.
RFL,114000.	Set the field length for PHASE1.
LDSET(LIB=DBLIB)	Specify that library DBLIB is to be searched to satisfy externals in PH1B.
PH1B.	Load and execute PH1B.
REWIND,OUTPUT. COPY,OUTPUT,ERRS. RETURN,OUTPUT.	Copy to file ERRS anything written to file OUTPUT in PHASE1. Return OUTPUT so that it can be re-created as a new file in PHASE2.
RFL,121000.	Set field length for PHASE2.
LDSET(LIB=DBLIB)	Specify that library DBLIB is to be searched to satisfy externals in PH2B.
PH2B.	Load and execute PH2B.
REWIND,OUTPUT. COPY,OUTPUT,ERRS. RETURN,OUTPUT.	Copy to file ERRS anything written to file OUTPUT in PHASE2. Return OUTPUT so that it can be re-created as a new file in PHASE3.
RFL,134000.	Set field length for PHASE3.
LDSET(LIB=DBLIB)	Specify that library DBLIB is to be searched to satisfy externals in PH3B.
PH3B.	Load and execute PH3B.

JOB CONTROL STATEMENT	EXPLANATION
REWIND,ERRS. COPY,ERRS,OUTPUT.	If anything was written on ERRS, copy it to OUTPUT now.
EXIT. REWIND,ERRS. COPY,ERRS,OUTPUT REWIND,DBUTIL. COPY,DBUTIL,OUTPUT.	Control passes here when a system error (such as address out of range) occurs during processing. Files ERRS and DBUTIL are copied to OUTPUT to help the DAVE maintainer determine the cause of the error.

STEP SIX. Run DAVE on the subject program TESTFL (file #8 on the tape) by calling the procedure file:

```
CALL(DAVE(OPTIONS=INPUT,INPUT=TESTFL)
```

where file INPUT contains the record

```
SI=ON
```

which indicates that simulation of missing subprograms is desired.

The output produced should duplicate the listing in Appendex A.

Chapter V: INTERNAL DIAGNOSTICS PRODUCED BY DAVE

This section contains explanations of the internal diagnostics issued by DAVE and their probable causes. Some of these diagnostics are due to ANSI violations in the subject program, which can be corrected by the user. Others are due to overflow of arrays used by DAVE and may be corrected by enlarging array dimensions. DAVE's size limitations are explained in detail in Chapter III, Section 5. Still other diagnostics may be due to an undetected bug in DAVE, in which case we should be notified.

Diagnostics emanating from DBINIT, the data base initialization package, are due to errors in file NUDATA. These should not appear when using the NUDATA supplied with the DAVE system. However, an explanation of these diagnostics appears here in case the user wishes to modify NUDATA, since the data base system may be used independently from DAVE.

Diagnostics emanating from PHASE0 are mainly due to ANSI FORTRAN violations in the subject program and so are discussed in Chapter II, Section 2.6.

For DATAB, PHASE1, PHASE2 and PHASE3, diagnostic packets are issued through Subroutine ERRSUB, contained in DATAB. The ERRSUB packet has the following format:

```
//ERROR///
ERROR NUMBER =       -1
DETECTED IN SUBPROGRAM EXAMPL
ERROR PACKET IS --
INDEX          OCTAL                  ALPHABETIC        INTEGER
  1      0000000000000000044                  9           36
  2      0255555555555555555      B                       R
  3      0000000000000000020                  P           16
```

This is followed by traceback information (ERRSUB is edited beforehand to call the local traceback routine.) A negative error number is fatal and causes termination of the job by ERRSUB after the traceback. A positive error number causes a diagnostic package to be printed but execution continues. There are very few non-fatal errors so that termination may acceptably occur in the traceback routine.

The listing of the diagnostics is organized as follows:

I. DBINIT
II. DATAB
III. PHASE1, PHASE2, PHASE3 combined

I. DIAGNOSTICS ISSUED BY DBINIT (Data Base Initialization Package)

1. $$$ FATAL ERROR $$$ NUMBER OF DATA TYPES REQUESTED EXCEEDS <maximum number of nodes>

The maximum number of nodes allowable (100) has been exceeded. Processing is aborted.

2. $$$ FATAL ERROR $$$ NUMBER OF FIELDS REQUESTED EXCEEDS <maximum number of fields>

The maximum number of fields allowable (200) has been exceeded. Processing is aborted.

3. $$ ERROR $$ BUFFER OVERFLOW ON ENTRY--<entry name>

There are too many field or node names. This condition is dependent upon #1 and #2 above. Processing continues.

4. $$ ERROR $$ ILLEGAL TYPE CODE--<code>
ALPHANUMERIC (A) TYPE CODE SUBSTITUTED

Legal type codes are 'A', 'P', 'R', 'I'. Processing continues.

5. $$ ERROR $$ TOO MANY TOKENS ON REMAINDER OF CARD--<remainder of card>

There is a card on NUDATA whose syntax is in error. Processing continues.

6. $$ ERROR $$ UNCONVERTIBLE TOKEN--<token>

There is a card on NUDATA whose syntax is in error. Processing continues.

7. $$ ERROR $$ BOUNDS SHOULD BE INTEGER, AT LEAST ONE IS NOT

There is an error in the bounds information on NUDATA. Processing continues.

8. $$ ERROR $$ BOUNDS SHOULD BE REAL, AT LEAST ONE IS NOT

There is an error in the bounds information on NUDATA. Processing continues.

9. $$ ERROR $$ NODE SIZE EXCEEDS<maximum part words> PARTWORDS

There are more than 100 part word fields for a particular node. Processing continues.

10. $$ ERROR $$ DUPLICATE REQUEST NAME

    There are duplicate common block names in the explicit INCLUDE blocks. Processing continues.

11. $ WARNING $ UNIDENTIFIED CARD--<card image>

    There is an invalid key on NUDATA. Valid keys are 'T', 'N', 'F', 'E'. Processing continues.

12. $ WARNING $ COMMENT INDICATOR IS NOT A POSITIVE INTEGER

    The number of comments must be indicated by 'C', blank, or integer value. Processing continues.

13. $ WARNING $ NO BOUND INFORMATION IS REQUIRED

    For 'A' and 'P' types, no bound information is needed. Processing continues.

14. $ WARNING $ NO DATA BASE DEFINED

    No data base description was given.

15. $ WARNING $ A NODE WITH NO FIELDS HAS BEEN DEFINED

    Incomplete node description.

II. DIAGNOSTICS ISSUED BY DATAB

Diagnostics emanating from subprograms in DATAB should not appear under normal circumstances. Since they may be difficult to interpret without a working familiarity with the data base system, we at the University of Colorado should be contacted for assistance.

Notes:

1. Some errors which are detected by user level data accessing routines are actually sensed by service routines called by them. The traceback information for such cases will include the name of that service routine sensing the error. In the following description, errors sensed via service routines are denoted as follows:

Denotation	Service Routine
(a)	CHLPAR
(b)	CHTPAR
(c)	LOCLST
(d)	LOCTBL

2. Some elements of a given error packet may not appear for every error or may contain the alphanumeric name associated with the element; these are denoted by *.

3. This section is divided into two sub-sections:
   A) A listing of the error numbers and their packets by subprogram for DATAB.
   B) A description of the errors by number.

A. Error numbers and packets by subprogram.

	ERROR #	MESSAGE
ADLSTL	1(a)	ERRPKT(1)*=IFIELD
	2(a)	(2)*="NIL"
	4(a)	(3)*=NDTYPH
	9(a)	(4)*="NIL"
	10(a)	(5) =LOCH
	-15	(6)*=LNKFLD
	-20	(7)*="NIL"
	21	(8)*=NODTYP
	-23	(9)*="NIL"
		(10) =LOC
		(11)*=node type of node pointed to
ADLSTT	1(a,b)	ERRPKT(1)*=IFIELD
	2(a)	(2)*="NIL"
	3(b)	(3) =NUMBER
	4(a)	(4)*=ITABLE
	6(b)	(5)*="NIL"
	9(a,b)	(6)*=LNKFLD
	10(a)	(7)*="NIL"
	13(b)	(8)*=NODTYP
	-15	(9)*="NIL"
	-19(b)	(10) =LOC
	-20	(11)*=no. of entries
	21	in ITABLE
	-23	or node type of node pointed to

	ERROR #	MESSAGE
CPLIST	1(a)	ERRPKT(1)*=LNKFLD
	2(a)	(2)*="NIL"
	4(a)	(3)*=NODTYP
	9(a)	(4)*="NIL"
	10(a)	(5) =LOC
	-12	(6) =IBUFSZ
	-15	(7)*=node type
	16	of node pointed
	20	to
GARWCK	-17	ERRPKT(1) =NAMNOD(ITABLE,1)
		(2) =NAMNOD(ITABLE,2)
ITMLST	1(a)	ERRPKT(1)*=IFIELD
	2(a)	(2)*="NIL"
	4(a)	(3)*=NODTYP
	-8	(4)*="NIL"
	9(a)	(5) =LOC
	10(a)	(6)*=free list pointer
	-11(c)	(7)*=size of
	-12(c)	data base
	-15	
ITMTBL	1(b)	ERRPKT(1)*=IFIELD
	3(b)	(2)*="NIL"
	6(b)	(3) =NUMBER
	-8	(4)*=ITABLE
	9(b)	(5)*="NIL"
	13(b)	(6)*=pointer to table entry
	-15	(7)*=base of table
	-18(d)	(8)*=top of table
	-19(b)	
LSTPOS	-1	ERRPKT(1)*=ITABLE
	-3	(2)*="NIL"
	-4	(3)*=type of data base
	-9	
	19	

	ERROR #	MESSAGE
LSTSCH	1(a)	ERRPKT(1)*=IFIELD/LNKFLD
	2(a)	(2)*="NIL"
	4(a)	(3)*=NODTYP
	9(a)	(4)*="NIL"
	10(a)	(5) =LOC
	-15	
MKLSTL	1	ERRPKT(1)*=IFIELD
	2	(2)*="NIL"
	4(a)	(3)*=NDTYPH
	9	(4)*="NIL"
	10(a)	(5) =LOCH
	-15	(6)*=LNKFLD
	19	
	20	(7)*="NIL"
	21	
	-22	(8)*=NODTYP
		(9)*="NIL"
		(10)*=node type of node pointed to
MKLSTT	1	ERRPKT(1)*=IFIELD
	2	(2)*="NIL"
	3(b)	(3) =NUMBER
	6(b)	(4)*=ITABLE
	9	(5)*="NIL"
	13(b)	(6)*=LNKFLD
	-15	
	19	(7)*="NIL"
	-19(b)	
	20	(8)*=NODTYP
	21	
	-22	(9)*="NIL"
		(10)*=number of entries in table or node type of node pointed to

	ERROR #	MESSAGE
MTYLST	1(a)	ERRPKT(1)*=NODTYP
	2(a)	(2)*="NIL"
	4(a)	
	9(a)	
	10(a)	
MTYTBL	1(b)	ERRPKT(1)=ITABLE
	3(b)	
	6(b)	
	9(b)	
	13(b)	
	-19(b)	
NEWNOD	1	ERRPKT(1)*=NODTYP
	4	
	9	(2)*="NIL"
	-15	
	-17	
NXTPOS	-16	ERRPKT(1) =ITABLE
PAGE	-25	ERRPKT(1) =IPRFOF
		(2) =IP
PUTLST	1(a)	ERRPKT(1) =INFO
	2(a)	(2)*=IFIELD
	4(a)	(3)*="NIL"
	7	(4)*=NODTYP
	8	(5)*="NIL"
	9(a)	(6) =LOC
	10(a)	(7)*=MINBND(IFIELD)
	-11(c)	or free pointer
	-12(c)	(8)*=MAXBND(IFIELD
	14	or size of data base
	-15	

	ERROR #	MESSAGE
PUTTBL	1(b)	ERRPKT(1) =INFO
	3(b)	(2)*=IFIELD
	6(b)	(3)*="NIL"
	- 7	(4) =NUMBER
	- 8	(5)*=ITABLE
	9(b)	(6)*="NIL"
	13(b)	(7)*=number of entries
	-14	in table or node
	-15	type pointed to or
	-18(d)	MINBND(IFIELD)
	-19(b)	(8)*MAXBND(IFIELD)
RESEQ	-26	ERRPKT(1)=ISTACK(1) ⋮ ⋮ (MAX) =ISTACK(MAX)
STLIST	- 1	ERRPKT(1) =NODTYP
	- 9	(2) ="NIL"
	-16	(3) =IBUFSZ
TOPLSL	1(a)	ERRPKT(1)*=IFIELD
	2(a)	(2)*="NIL"
	4(a)	(3)*=NDTYPH
	9(a)	(4)*="NIL"
	10(a)	(5) =LOCH
	-15	(6)*=LNKFLD
	-20	(7)*="NIL"
	21	(8)*=NODTYP
		(9)*="NIL"

	ERROR #	MESSAGE
TOPLST	1(a,b)	ERRPKT(1)*=IFIELD
	2(a)	(2)*="NIL"
	3(b)	(3) =NUMBER
	4(a)	(4)*=ITABLE
	6(b)	(5)*="NIL"
	9(a,b)	(6)*=LNKFLD
	10(a)	(7)*="NIL"
	13(b)	(8)*=NODTYP
	-15	(9)*="NIL"
	-19(b)	(10) =LOC
	-20	(11)*=number of entries in table
	21	or node type of node pointed to
WRDAB	-35	ERRPKT(1) =IDBTB(1,1) ⋮ (2*ISZ) =IDBTB(ISZ,2)
XPTLST	1(a)	ERRPKT(1) =XINFO
	2(a)	(2)*=IFIELD
	4(a)	(3)*="NIL"
	7	(4)*=NODTYP
	9(a)	(5)*="NIL"
	10(a)	(6) =LOC
	-11(c)	(7)*=XMIN(IFIELD) or
	-12(c)	free pointer
	-15	(8)*=XMAX(IFIELD) or size
	24	of data base

1(b)	ERRPKT(1) =XINFO
3(b)	(2)*=IFIELD
6(b)	(3)*="NIL"
- 7	(4) =NUMBER
9(b)	(5)*=ITABLE
13(b)	(6)*="NIL"
-15	(7)*=number of entries
-18(d)	in table or node type
-19(b)	of node pointed to or
-24	pointer to table entry
	or MINBND(IFIELD)
	(8)*=base of table or MAXBND(IFIELD)
	(9)*=top of table

1(a)	ERRPKT(1)*=IFIELD
2(a)	(2)*="NIL"
4(a)	(3)*=NODTYP
9(a)	(4)*="NIL"
10(a)	(5) =LOC
-11(c)	(6)*=free pointer
-12(c)	(7)*=size of data base
-15	
24	

1(b)	ERRPKT(1)*=IFIELD
3(b)	(2)*="NIL"
6(b)	(3) =NUMBER
9(b)	(4)*=ITABLE
13(b)	(5)*="NIL"
-15	(6)*=number of entries in
-18(d)	table or node type of
-19(d)	of node pointed to
	(7)*=base of table
	(8)*=top of table

B. Errors described by number.

1. The node name for a list or table node is unrecognizable.
2. Field does not correspond with the list node type.
3. Illegal table.
4. Illegal main data base area.
5. Requested information from a field that is empty.
6. Sequence number is out of range.
7. Information to be put in the data base is not within the correct range.
8. Field type must not be any of the real types.
9. Node is a sequential table node when a linked node is required.
10. Illegal pointer--pointer flag missing.
11. Pointer is out of range.
12. Pointer points to a different type node than NODTYP.
13. Field specified is not a component of the sequential table type.
14. Only pointer information may be put in the specified field.
15. Termination due to illegal parameter(s); see preceding diagnostic packets.
16. Overflow in vector.
17. Overflow in data base. The maximum number of pages MAXPGS=100 and the page size IPAGSZ=512 are set in DBINIT.
18. Computed address of a sequential node is not in the data base - check for illegal data base or vector.
19. Node type specified is not in the vector IBUF.
20. LNKFLD must be a pointer field.
21. IFIELD must be a pointer field.
22. The vector does not contain any nodes to be put on the linked list.
23. Node(s) on list is not of the same type as the node to be added to the list.
24. Field must be of real type.

25. In-core data base area is too small for a non-prefix page to reside in core.

26. Overflow of array ISTACK, whose size is MAX=512. ISTACK and MAX appear in common block STACKS in the calling subprogram. The size of ISTACK is set by DBINIT.

27. An error was discovered in the marking of a node.

28. There has been a change in the format of the prefix between the roll-out and roll-in phases.

29. Array IBUF, whose size is MAXBUF, passed in through ROLLIN, is not large enough to hold the old data base description, whose size is stored in IDBTB(ISZ,1).

30. An unpaged data base is being rolled in but the new in-core array is not large enough to hold even the prefix + linked area of the old data base. IDBTB(NUM,2) is the size of the prefix + linked area.

31. The old unpaged data base will not fit into the new data base array. ICON=the unused portion of the old data base (between linked and sequential) is less than old size-new size.

32. The size IASZ of IAREA is less than LIMIT, the number of data base description items to be written out.

33. MAXBUF, the size of IBUF, is less than MAXPGS=100, the maximum number of pages for a data base. IBUF is used by WRDAB to hold information about the pages, which is then written out to IFLNM.

34. During the roll-in process, it has been found that all the linked nodes were not targeted properly.

35. Overflow of the 2-dimensional array IDBTB used to hold information on the format of the file of data bases. It is dimensioned (no. of data bases,2).

ADDENDUM TO V. III, p. 174

(Internal Diagnostics Produced by DAVE)

CKGRPH		
- 2	ERRPKT(1)=IORDR(IORSZ)	Overflow of array IORDR. There are more than IORSZ subprogram units (including those to be simulated). IORSZ must be increased; see Ch. III, Sec. 5.2.
LKCLRS		
- 1	ERRPKT(1)=IGRPH(IPNT)	Overflow of array IGRPH, due to a large number of subprogram units being processed and/or a large number of common blocks. See Ch. III, Sec. 5.2, I.A.2.

III. PHASE1, PHASE2, PHASE3 combined.

NO.	MESSAGE	MEANING
AC1		
-1	ERRPKT(1)=KCHAR (2)=IPONT (3)=ISAVE	An operator or logical constant has not been found as expected. Probable cause: Syntax error
-2	(1)=INSTK(IPONT) (2)=IPONT (3)=IEND	A complete operator or logical constant was not found as expected. Probable cause: Syntax error in the subject program.
AC2		
-1	ERRPKT(1)=INSTK(IPONT) (2)=IPONT (3)=ITYPE	The current character INSTK(IPONT) is not one of those expected, "+" "-" "/" "(" ")" "," "=". Probable cause: Syntax error in subject program.
AC3		
-1	ERRPKT(1)=IBEG (2)=IEND (3)=ISAVE	No non-blank character is found after the asterisk. Probable cause: A syntax problem involving the use of * or ** has been encountered in the subject program.
AC4		
-1	ERRPKT(1)=IBEG (2)=IEND (3)=IPONT	A non-blank character was not found as expected. Probable cause: Syntax error in subject program.
-2	ERRPKT(1)=IBEG (2)=IEND (3)=IPONT	Either .D or .E was found. Alone they constitute an illegal string. Probable cause: Syntax error in subject program.

NO.	MESSAGE	MEANING
AC5		
-1	ERRPKT(1)=INSTK(IPONT) (2)=IPONT	The current character is not an E, D or H as required. Probable cause: Syntax error in subject program
-3	ERRPKT(1)=INSTK(IPONT) (2)=IPONT	The length of the Hollerith string is less than or equal to 0. Probable cause: Syntax error in subject program - an illegal Hollerith specification has been encountered.
AC6		
-1	ERRPKT(1)=IPONT (2)=IEND	Error in imaginary part of complex constant. Probable cause: Syntax error.
ASIGN		
-1	ERRPKT(1)=I (2)=D (3)=X	There are more than 20 dummy arguments in an arithmetic statement function.
BLKTAB		
-1	ERRPKT(1)=NUM (2)=LENGTH	Overflow of array IUNDOV, whose size is LENGTH=500, the maximum allowable number of blocks per subprogram unit in the subject program. BLKTAB is called by DRIVE2 with corresponding actual parameter array IBIND. See Ch. III, Sec. 5.2, II.A.10.

NO.	MESSAGE	MEANING
C37T11		
-1	ERRPKT(1)=ICHAR(1) : (I)=ICHAR(I)	There is a non-ANSI variable name containing more than 6-characters.
CGRAPH		
-1	ERRPKT(1)=A (2)=R (3)=R (4)=F (5)=U (6)=L (7)=L	Overflow of array IGRPH, due to a large number of subprogram units being processed and/or a large number of common blocks. See Ch. III, Sec. 5.2, I.A.2.
CHR		
-1	ERRPKT(1)=C	The character C is not a blank, letter or digit. Probable cause: An error in the subject program's representation of a variable name, or system fault.
CKGRPH		
-1	ERRPKT(1)=INAMS(1) : (400)=INAMS(400)	There are more than 400 unsatisfied externals in the program units being processed. INAMS contains their names (in coded base 37 representation).
CMDOWN		
-1	ERRPKT(1)=IGRPH(IPCMI) (2)=IGRPH(IPCMI+1) (3)=IGRPH(IPCMI+2)	An entry in IGRPH for this common block is missing. Probable cause: System fault.
-2	ERRPKT(1)=IGRPH(IPCMI) (2)=IGRPH(IPCMI+1) (3)=IGRPH(IPCMI+2)	The classification of this common block in the calling subprogram is unknown. Probable cause: System fault.

NO.	MESSAGE	MEANING
COMPLT		
-1	ERRPKT(1)=INA	The first character (INA) of a symbolic name converted from its base 37 representation is not a letter. Probable cause: System fault or syntax error in subject program.
-2	ERRPKT(1)=ISTNO	There is a missing statement number (ISTNO) in the subject program.
COMTCH		
-1	ERRPKT(1)=IEXT (2)=ICMB	There is no entry for common block ICMB (base 37 representation) in the calling subprogram's list of blocks. Probable cause: System fault.
-2	ERRPKT(1)=IEXT (2)=ICMB	A common block is always available in the called subprogram and either sometimes or never available in the calling subprogram. Probable cause: System fault.
CONTOK		
-1	ERRPKT(1)=INTSK(1) : (100)=INSTK(100)	Type of token is out of allowable range. Probable cause: System fault.
-2	ERRPKT(1)=INSTK(1) : (100)=INSTK(100)	Overflow of token stack. Probable cause: System fault or Fortran statement in subject program exceeds 20 lines.
-3	ERRPKT(1)=INSTK(1) : (100)=INSTK(100)	The starting point of the string input exceeds the end. Probable cause: System fault.
-4	ERRPKT(1)=INSTK(1) : (100)=INSTK(100)	The length of array INSTK is less than the index of the last character of string input . Probable cause: System fault.

NO.	MESSAGE	MEANING
CONTOK	(continued)	
-5	ERRPKT(1)=INSTK(1) : : (100)=INSTK(100)	No "T" was found in an ASSIGN__TO statement.
DATYP		
-1	ERRPKT(1)=K (2)=E (3)=Y (4)= (5)=E (6)=R (7)=R	The statement being processed is not a type declaration as expected. Probable cause: System fault
DUMYIN		
-1	ERRPKT(1)=IORDR(1) : : (N)=IORDR(N)	There is no entry in IGRPH for the caller of this subprogram unit (N^{th} in IORDR). IORDR contains the processing order of the units.
-2	ERRPKT(1)=IORDR(1) : : (N)=IORDR(N)	No entry was found for this subprogram unit's name in the caller's Symbol Table. Probable cause: System fault.
-3	ERRPKT(1)=IORDR(1) : : (N)=IORDR(N)	No entry was found in the caller's Statement Table for the call to this external. Probable cause: System fault.
EQADD		
-1	ERRPKT(1)=I (2)=Q (3)=M (4)=A (5)=X	Overflow in array ICHTB. IQMAX=100, the maximum number of variables which may appear in EQUIVALENCE statements in a program unit. See Ch.III, Sec. 5.2, I.C., I.D., and II.E.

NO.	MESSAGE	MEANING
EQMRG		
-1	ERRPKT(1)=E (2)=Q (3)=U (4)=I (5)=V (6)= (7)=E (8)=R (9)=R	An error has occurred in the processing of equivalence chains. Probable cause: System fault or equivalencing error in subject program.
EQUIV		
-1	ERRPKT(1)=I (2)=Q (3)=M (4)=A (5)=X	Array IQLST has overflowed. IQMAX=100, the maximum number of variables which can appear in equivalence lists plus the number of lists in a program unit. See Ch. III, Sec. 5.2, I.C, I.D. and II.E.
EXPRSN		
-1	ERRPKT(1)=IBEG (2)=IEND (3)=1	Source string indices are incorrect: IBEG>IEND Probable cause: System fault.
	ERRPKT(1)=NEXTPT (2)=LENSTT (3)=2	The next transition is in error; NEXTPT no longer points to the transition table. (NEXTPT must be less than the length of the table LENSTT). Probable cause: Syntax error in an expression in the subject program.
	or	
	ERRPKT(1)=NXTST (2)=KRSTA (3)=3	The next state is in error; NXTST must be one of 1, 2, 3, 4, 5, but has been found to be 0. Probable cause: Syntax error in an expression in the subject program. One possibility is the illegal juxtaposition of 2 operators.

NO.	MESSAGE	MEANING
FCALL		
-1	ERRPKT(1)=ISTK(IP,1)	A parameter classification of procedure declared external or arithmetic statement function name was expected but not found. Probable cause: System fault.
-2	ERRPKT(1)=ISTK(1,1) ⋮ ⋮ ⋮ (IP)=ISTK(IP,1)	The token is equal to none of the expected code identifiers. Probable cause: System fault.
-3	ERRPKT(1)=ISTK(1,1) ⋮ ⋮ (10)=ISTK(10,1) or ERRPKT(1)=IPAR(1,1) ⋮ ⋮ (10)=IPAR(10,1)	Underflow in either ISTK or IPAR. Probable cause: System fault, or incorrect parentheses usage in subject program.
FLTCHR		
-1	ERRPKT(1)=I1 (2)=I2 (3)=V(I1)	Either the input segment V(I1),..., V(I2) does not contain a valid representation of a real number or I1>I2. Probable cause: Syntax error in FORTRAN representation of a real number in the subject program, or system fault.
GRORDR		
-1	ERRPKT(1)=B (2)=A (3)=D (4)=N (5)=A (6)=M (7)=E	An entry for the caller of this subprogram unit was not found. Probable cause: System fault.
GROUP		
-1	ERRPKT(1)=NUM	Have an incorrect diagnostic number, NUM. Probable cause: System fault.

NO.	MESSAGE	MEANING
INPVAR		
-1	ERRPKT(1)=ISTK(1) (2)=ISTK(2)	Underflow of array ISTK. Probable cause: System fault.
-2	ERRPKT(1)=IBIND(IBB) ⋮ (IBK)=IBIND(IBL)	Overflow of array ISTK, dimensioned LIM=1000. Probable cause: There are more than LIM/2=500 blocks in the subprogram unit being processed. IBIND contains the start index in IOBLK for each block. See Ch. III, Sec. 5.2, II.A.10.
-3	ERRPKT(1)=IOBLK(INNOD)	Illegal I/O classification code. Probable cause: System fault.
INTCHR		
-1	ERRPKT(1)=I1 (2)=I2 (3)=V(I1)	Either the input segment V(I1,..., V(I2) does not contain a valid representation of an integer, or I1>I2. Probable cause: Syntax error in the subject program or system fault.
IOINIT		
-2	ERRPKT(1)=P (2)=A (3)=S (4)=T (5)=E (6)=N (7)=D	Overflow of array ISTOP, containing the block numbers of exit nodes for a subprogram unit. ISTPL=100, set in Subroutine IODRVR.
-3	ERRPKT(1)=P (2)=A (3)=S (4)=T (5)=E (6)=N (7)=D	Overflow of array IBIND, which contains one entry/block for a subprogram unit. IBNDL (size of IBIND=500), See Ch. III, Sec. 5.2, II.A.10.

NO.	MESSAGE	MEANING
IOINIT	continued	
-4	ERRPKT(1)=U (2)=N (3)=C (4)=L (5)=A (6)=S (7)=S	A variable's I/O status is unclassified. Probable cause: System fault.
-5	ERRPKT(1)=P (2)=A (3)=S (4)=T (5)=E (6)=N (7)=D	Overflow of array IOBLK whose size is IOBSZ=9000. The size is dependent upon the number of blocks and variables in a subprogram. See Ch. III, Sec. 5.2, II.B.2.
-6	ERRPKT(1)=P (2)=A (3)=S (4)=T (5)=E (6)=N (7)=D	Overflow of array IEQND, whose size is IELMT=100. IEQND contains the equivalenced variables for a subprogram unit. See Ch. III, Sec. 5.2, II.E.
-7	ERRPKT(1)=P (2)=A (3)=S (4)=T (5)=E (6)=N (7)=D	Overflow of array INTID, whose size is INIDL=300. INTID contains Symbol Table pointers for the global or local variables in a subprogram unit. See Ch. III, Sec. 5.2, II.A.11.
IOSTAT		
-1	ERRPKT(1)=IPR(1) (2)=IPR(2) (3)=IPR(3) (4)=IPR(4)	I/O classification of parameter is in error. Probable cause: System fault.
IOVUND		
-1	ERRPKT(1)=IEXITB (2)=NUM (3)=LHEAD (4)=KCOUNT	The block IEXITB does not have a linked list of entry blocks. Probable cause; System fault.

NO.	MESSAGE	MEANING
IOVUND	(continued)	
-2	ERRPKT(1)=IEXITB (2)=NUM (3)=LHEAD (4)=KCOUNT (5)=LLNUM	The block NUM is not in the entry block list for block IEXITB. Probable cause: System fault.
-3	ERRPKT(1)=KCOUNT (2)=KSIZE	KCOUNT has exceeded KSIZE=50, the size of array KOVUND, which contains undefined output variables for a block.
ISTYPE		
-1	ERRPKT(1)=ICHAR (2)=IPONT	A non-ANSI character ICHAR has been found. Probable cause: Syntax error in the subject program.
LABBLK		
-1	ERRPKT(1)=K (2)=LHEAD (3)=LSAVE	The linked list of internal references to an external label is in error. Probable cause: System fault.
-2	ERRPKT(1)=NUMDO (2)=KDO	There are more than KDO=20 DO-statements which refer to the same DO-loop closure statement number.
-3	ERRPKT(1)=JJ (2)=INLABX	There are more than INLABX=50 statements which refer to the same external label.
LASTNB		
-1	ERRPKT(1)=IBEG (2)=IEND (3)=IPONT	IPONT is out of range. Probable cause: System fault.

NO.	MESSAGE	MEANING
LEXST		
1	ERRPKT(1)=IERR (2)=J (3)=0 (4)=LLAST (5)=JTYPE (6)=JCODE	The beginning of the program unit was reached without finding an executable statement. Probable cause: System fault.
2	ERRPKT(1)=IERR (2)=J (3)=JBLOCK (4)=LLAST (5)=JTYPE (6)=JCODE	An executable statement was not found before block J. Probable cause: System fault.
3	ERRPKT(1)=IERR (2)=J (3)=0 (4)=LLAST (5)=JTYPE (6)=JCODE	JCODE is out of range, 1-2. Probable cause: System fault.
LUNTIL		
-1	none	A non-blank character could not be found. Probable cause: Syntax error in subject program.
MKLBTB		
-1	ERRPKT(1)=TKSTK(1) ⋮ ⋮ (2*LCLEN+6)=TKSTK(2*LCLEN+6)	There is a labelling error in an ASSIGN, GOTO, or DO statement.
NAMCHR		
-1	ERRPKT(1)=I1 (2)=I2 (3)=V(I1)	Either the input segment V(I1),..., V(I2) does not contain a valid representation of a FORTRAN variable name, or I1>I2. Probable cause: Syntax error in subject program, or system fault.

NO.	MESSAGE	MEANING
NEXTNB		
-1	ERRPKT(1)=IBEG (2)=IEND (3)=IPONT	IPONT is out of range. Probable cause: System fault.
NUMCHR		
-1	ERRPKT(1)=NUM	The array IARRAY of length LEN is too small to hold the converted integer. NUM is the remainder to be converted to characters. Probable cause: System fault.
OUTVAR		
-1	ERRPKT(1)=ISTK(1) (2)=ISTK(2)	Underflow of array ISTK. Probable cause: System fault.
-2	ERRPKT(1)=IBIND(IBB) : (IBL)=IBIND(IBL)	Overflow of array ISTK, dimensioned LIM=1000. Probable cause: There are more than LIM/2=500 blocks in the subprogram unit being processed. IBIND contains the start index in IOBLK for each block. See CH. III, Sec. 5.2, II.A.10.
-3	ERRPKT(1)=ISTOP(1) : (ISTL)=ISTOP(ISTL)	Overflow of array ISTOP, containing the block numbers of exit nodes for a subprogram unit. ISTL=100, set in Subroutine IODRVR.
-4	ERRPKT(1)=IOBLK(INNOD)	Illegal I/O classification code. Probable cause: System fault.
PHASE1		
-1	[none]	Probable cause: Syntax error in subject program. Check for the following: (a) a missing (non-blank) character. (b) an error in an ASSIGN statement. (c) an error in a DO statement.

NO.	MESSAGE	MEANING
PHASE1	(continued)	
-2	ERRPKT(1)=ITYPE	An illegal statement type (ITYPE) has been encountered. Probable cause: Syntax error. Check for a non-ANSI FORTRAN statement.
PRACI		
-1	ERRPKT(1)=first parameter : (20)=20th parameter	There are more than 20 dummy parameters in an arithmetic statement function.
PRODM		
-1	ERRPKT(1)=N (2)=O (3)=T (4)=D (5)=M	This is not a DIMENSION statement as expected. Probable cause: System fault.
-2	ERRPKT(1)=N (2)=O (3)=L (4)=F (5)=T (6)=P (7)=A (8)=R	Missing left parenthesis. Probable cause: Syntax error in subject program.
PROIMP		
-1	ERRPKT(1)=ITOK (2)=IVAL	Incorrect keyword in IMPLICIT statement. Probable cause: Syntax error in subject program.

NO.	MESSAGE	MEANING
PUSH		
-1	ERRPKT(1)=ISTK(1,1) : (IP)=ISTK(IP,1)	Overflow of array ISTK, dimensioned (100,2). PUSH is called by Subroutine FCALL with array ISTK or array IPAR, which are in blank common in FCALL. To enlarge either array, only the blank common in FCALL need be changed and the dimension in PUSH. Probable cause: Very deep nesting of calls within a subroutine or function call or very deep nesting of parentheses within a call.
PUTTOK		
-1	ERRPKT(1)=IALPHA (2)=IBEFA (3)=IFNTK (4)=LNTOK	An overflow condition exists. Probable cause: System fault.
RDWR		
-1	ERRPKT(1)=IND(1) : (50)=IND(50)	Overflow of array IND, which contains the symbol table indices of variables appearing in implied DO loops in a READ or WRITE statement.
SAVEIO		
-2	ERRPKT(1)=V (2)=A (3)=R (4)=D (5)=I (6)=M	There is a variably dimensioned COMMON array in the subject program.

NO.	MESSAGE	MEANING
SBMTCH		
-1	ERRPKT(1)=ICOM(1) : (ICLMT)=ICOM(ICLMT)	There are more than ICLMT=300 entries for ICOM, which contains 2* no. of common blocks sometimes or always available to this subprogram. ICLMT is passed in from DRIVE2 as INIDL.
SKIP		
-1	[none]	A non-blank character could not be found. Probable cause: Syntax error in subject program. Check for a non-ANSI FORTRAN statement.
SORTL		
-1	ERRPKT(1)=LIST(1) : (NUM)=LIST(NUM)	NUM is greater than the size, LENG, of LIST. NUM should fall between 1 and LENG. Probable cause: System fault.
STEPAR		
-1	ERRPKT(1)=IPR(1) (2)=IPR(2) (3)=IPR(3) (4)=IPR(4)	An argument in a subroutine or function call is not an allowable type. IPR(1) is the base 37 coded representation of the called subprogram. Probable cause: System fault.
STOUT		
-1	ERRPKT(1)=NAM(1) : (21)=NAM(21)	Overflow of array NAM, dimensioned 21, which contains an output line for Part II summary. Probable cause: System fault.

NO.	MESSAGE	MEANING
STRDM		
-1	ERRPKT(1)=B (2)=A (3)=D (4)=T (5)=0 (6)=K	The token is not ")", ",", a symbolic name or an integer as expected. Probable cause: Syntax error in subject program
-2	ERRPKT(1)=N (2)=0 (3)=R (4)=T (5)=P (6)=A (7)=R	The right paranthesis is missing for a dimensioned variable.
TRANS		
-1	ERRPKT(1)=ICODE (2)=KA (3)=KB	ICODE is out of the range 1-8. Probable cause: System fault.
-2	ERRPKT(1)=ICODE (2)=KA (3)=KB	No entry is found on KA's exit list for block KB. Probable cause: System fault.
-3	ERRPKT(1)=ICODE =KA =KB	No entry is found on KB's entry list for block KA. Probable cause: System fault.
-4	ERRPKT(1)=ICODE (2)=KA (3)=JTYPE	The statement type JTYPE for the last statement in block KA is incorrect. Probable cause: System fault.
VARPR		
-1	ERRPKT(1)=INPVAR (2)=J	The input variable (INPVAR) is in the output list with type undefined for statement number J. Probable cause: system fault.

REFERENCES

[1] L. J. Osterweil and L. D. Fosdick, "DAVE--A Validation Error Dectection and Documentation System for FORTRAN Programs," Software-Practice and Experience, 6 (1976), 473-486.

[2] L. D. Fosdick and L. J. Osterweil, "Data Flow Analysis in Software Reliability," ACM Computing Surveys 8, 3 (1976), 305-330.

[3] L. J. Osterweil, L. Clarke, C. Miesse, E. Myers, D. W. Smith, "A Flexible FORTRAN Data Base System," (to appear).

[4] B. G. Ryder, "The PFORT Verifier," Software-Practice and Experience, 4 (1974), 359-378.

DAVE LEVEL 8.0

```
**************************************************
*                                                *
*            DAVE  TERMINATION  NORMAL           *
*                                                *
**************************************************
```

NOTE -- FOR MISSING SUBPROGRAMS THE FOLLOWING I/O BEHAVIOR HAS BEEN SIMULATED.

A. FOR FUNCTION SUBPROGRAMS, THE FUNCTION NAME HAS BEEN CLASSIFIED AS STRICT OUTPUT AND ALL ARGUMENTS AS STRICT INPUT, NON-OUTPUT.

B. FOR SUBROUTINE SUBPROGRAMS, ALL ARGUMENTS HAVE BEEN CLASSIFIED AS STRICT INPUT, NON-OUTPUT.

A SIMULATED SUBPROGRAM IS ASSUMED TO USE NO COMMON VARIABLES. THE NUMBER AND DIMENSIONS OF ITS DUMMY ARGUMENTS HAVE BEEN INFERRED FROM THE FIRST INVOCATION OF THE SUBPROGRAM BY THE PROGRAM UNIT INDICATED BELOW.

SIMULATED SUBPROGRAM	CALLER
---*FSIM*--	-*SYSMAIN*-
--*SUBSIM*-	-*SYSMAIN*-

USER OPTIONS SPECIFIED THIS RUN

1. SIMULATE I/O BEHAVIOR FOR MISSING SUBPROGRAMS (SI= ON).
2. RE-START OF PREVIOUS RUN (RE=OFF).
3. SUPPRESS DIAGNOSTICS (SU=OFF).

DIAGNOSTIC SUMMARY -- PART 1

FREQUENCY

SUBPROGRAM	ERRORS	WARNINGS	MESSAGES
SYSMAIN	24	48	5
BLKDATA			1
E101	2	5	1
SUB103	1	4	2
SUB302	1	5	2
SUB105		2	1
SUB106	1	1	2
SUB208	2	1	1
W201		4	1
SUB215		1	1
SUB		4	1
FUN	1	2	1
FSIM			1
SUBSIM			1

DIAGNOSTIC SUMMARY -- PART 2

ERRORS		WARNINGS		MESSAGES	
IDENT.NO.	FREQUENCY	IDENT.NO.	FREQUENCY	IDENT.NO.	FREQUENCY
101	1	201	1	301	2
102	2	202	1	302	2
103	4	203	1	303	3
104	1	204	5	304	14
105	1	205	1		
106	2	206	1		
107	2	207	2		
108	2	208	1		
109	2	209	1		
110	4	210	3		
111	2	211	1		
112	9	212	1		
		213	2		
		214	2		
		215	1		
		216	5		
		217	2		
		218	1		
		219	1		
		220	1		
		221	1		
		222	1		
		223	2		

224	1
225	1
226	1
227	1
228	1
229	12
230	1
231	1
232	2
233	2
234	1
235	1
236	10
237	4

CALL GRAPH

SUBPROGRAM	CALLED BY	CALLS
SYSMAIN		E101
		SUB103
		SUB105
		SUB106
		SUB208
		W201
		SUB215
		SUB
		FSIM
		SUBSIM
E101	SYSMAIN	
	SUB106	
SUB103	SYSMAIN	SUB302
SUB302	SUB103	SUB106
SUB105	SYSMAIN	SUB106
SUB106	SYSMAIN	E101
	SUB302	
	SUB105	
SUB208	SYSMAIN	
W201	SYSMAIN	
SUB215	SYSMAIN	
SUB	SYSMAIN	
FUN		
FSIM	SYSMAIN	
SUBSIM	SYSMAIN	

$ IN THE CONTINUATION FIELD INDICATES THE EXPANSION
OF THE LOGICAL IF STATEMENT ON THE PREVIOUS LINE

```
BLOCK                    SOURCE

    0  C  PROGRAM TO TEST ALL DAVES ERRORS WARNINGS AND MESSAGES.
    0  C  BEFORE EACH PIECE OF CODE IS A SHORT DESCRIPTION OF THE
    0  C   TYPE OF ERROR WARNING OR MESSAGE TO BE OUTPUT.
    0  C
    0  C
    1        COMMON/B1/CA1,BA
    1        COMMON/BLK1/CA,D218,Y228(6)
    1        EXTERNAL SUBEX
    1        DIMENSION XDAT(5),XDT(5,2)
    1        INTEGER I236
    1        DATA I220/1/ ,X221/1./
    1        LASRF(X,Y)=5.*E101(C)+X+Y
    2        C=1.
    3        B=1.
    0  C
    0  C  WARNING 209
    0  C     COMMON VAR REFERENCED BEFORE BEING DEFINED
    0  C
    4        IF(A.EQ.B)
    5       $ X = CA1
    0  C
    0  C  ERROR 109
    0  C     COMMON VAR IS REF BEFORE BEING DEFINED
    0  C
    6        X = M+CA
    0  C
    0  C  GENERATES ERROR 101 INSIDE FUNCTION
    0  C  GENERATES ERROR 102 INSIDE FUNCTION
    0  C
    7        R = E101(1.)
    0  C
    0  C  ERROR 103
    0  C       ACTUAL ARGUMENT IS CONSTANT OR EXPRESSION AND IS ASSIGNED A
    0  C       VALUE ON ALL PATHS IN CALLED SUBPROGRAM
    0  C  WARNING 203
    0  C       SAME AS 103 BUT SOME PATHS
    0  C
    8        CALL SUB103(3,B+C,Y+1)
    0  C
    0  C  ERROR 104
    0  C     NUMBER OF ARGUMENTS DOESNT MATCH
    0  C
    9        CALL SUB103(A)
    0  CC
    0  C  ERROR 105
    0  C     EXTERNAL IS USED AS A VAR IN CALLED ROUTINE
    0  C
    0  C  WARNING 204
    0  C     CONSTANT IS NEVER REF IN SUBPROGRAM
```

```
 0  C
10        CALL SUB105(SUBEX,3)
 0  C
 0  C  ERROR 106
 0  C     EXTERNAL IS ASSIGNED A VALUE ON ALL PATHS IN SUBPROGRAM
 0  C  WARNING 203
 0  C     CONSTANT IS ASSIGNED A VALUE ON SOME PATHS
 0  C  WARNING 213
 0  C       ARGUMENTS HAVE DIFFERENT DATA TYPES
 0  C
11        CALL SUB106(SUBEX,3)
 0  C
 0  C  WARNING 208
 0  C   COMMON VAR (CA) IS USED AS A DUMMY ARG. AND IS COMMON IN SUBPROGRAM
 0  C
12        CALL SUB208(A,CA)
 0  C
 0  C  ERROR 111
 0  C     CONTROL VAR IS REF OUTSIDE OF LOOP
 0  C  ERROR 112
 0  C     LOCAL VAR IS REF BUT NOT DEFINED
 0  C
13        DO 10 I = 1 , 10
14        K = LOC + 1
15     10 CONTINUE
16        K = I + 6
 0  C
 0  C  WARNING 205 AND 206
 0  C     EXTERNAL ,SUBEX, IS REFERENCED AS A VAR ON SOME PATHS
 0  C     AND IS ASSIGNED A VALUE ON SOME PATHS
 0  C
17        CALL SUB(1.,SUBEX)
 0  C
 0  C
 0  C  ERROR 108
 0  C     COMMON VAR IS ASSOCIATED WITH A DUMMY VAR
 0  C  FUNCTION W201 CAUSES WARNING 201 AND 202
 0  C
18        I = W201(CA)
 0  C
 0  C
 0  C  WARNING 215
 0  C       ARGUMENTS HAVE DIFFERENT DIMENSIONALITY
 0  C  WARNINGS 216 AND 217
 0  C       COMMON VARIABLE ASSIGNED A VALUE ON ALL(SOME) PATHS BUT
 0  C       BLOCK NOT AVAILABLE TO CALLER
 0  C  WARNINGS 218 AND 219
 0  C       COMMON VARIABLE INITIALIZED IN BLOCK DATA IS ASSIGNED A VALUE
 0  C       ON ALL(SOME) PATHS BUT BLOCK IS NOT AVAILABLE TO CALLER
19        CALL SUB215(XDAT,B,C)
 0  C  WARNING 226
 0  C       TYPE II ANOMALY ON ALL PATHS, COMMON VAR. IN MAIN PROGRAM
20        CA = 1.
21        CA = 2.
 0  C
 0  C  WARNING 227
 0  C       TYPE II ANOMALY ON SOME PATHS, COMMON VAR. IN MAIN PROGRAM
22        BA=CA
 0  C
23        IF(D219.EQ.0)
24       $ CA=3.+BA
 0  C
```

```
 0  C  WARNING 228
 0  C     COMMON ARRAY IS ASSIGNED A VALUE AND NOT REFERENCED
 0  C
25        Y228(1)=6.
 0  C
 0  C  WARNING 229
 0  C     LOCAL VAR IS ASSIGNED A VALUE AND NOT REFERENCED
 0  C
26        X229=6.
 0  C
 0  C  WARNING 229
 0  C     SAME AS ABOVE BUT ASSIGNED A VALUE AGAIN
 0  C
27        X230 = 1.
28        IF(CA.EQ.L)
29       $X230=3.
 0  C
 0  C  WARNING 231
 0  C     LOCAL ARRAY XDAT IS ASSIGNED A VALUE AND NOT USED
 0  C
30        XDAT(5)=1.
 0  C
 0  C  WARNING 232
 0  C     ILLEGAL SIDE EFFECT . VAR APPEARS TWICE IN BELOW STMT.
 0  C
31        I = W201(X)+X
 0  C
 0  C  WARNING 233
 0  C     SAME AS 232 BUT WITH COMMON VAR
 0  C
32        I = W201(CA)+CA
 0  C
 0  C  WARNING 234
 0  C     GLOBAL VAR C IS USED TWICE
 0  C
33        I = LASRF(2.,5.)+C
 0  C
 0  C  SIMULATION OF FUNCTION CALL
 0  C
34        I = FSIM(CA)
 0  C
 0  C  SIMULATE SUBROUTINE CALL
 0  C
35        CALL SUBSIM(X,A,B,D)
36        STOP
 1        END
```

E R R O R S

ERROR NUMBER	DESCRIPTION

** 103 ** BLOCK NO. 7
AN ACTUAL ARGUMENT IS AN EXPRESSION OR CONSTANT, YET THE
CORRESPONDING DUMMY ARGUMENT IS ASSIGNED A VALUE ON ALL PATHS.
CALLING SUBPROGRAM CALLED SUBPROGRAM

```
                              -*SYSMAIN*-          ---*E101*--
                ARGUMENT          REAL              ----*A*----
                POSITION           1                    1
```

** 103 ** BLOCK NO. 8
AN ACTUAL ARGUMENT IS AN EXPRESSION OR CONSTANT, YET THE CORRESPONDING DUMMY ARGUMENT IS ASSIGNED A VALUE ON ALL PATHS.

```
                              CALLING SUBPROGRAM  CALLED SUBPROGRAM
                                 -*SYSMAIN*-          --*SUB103*-
                ARGUMENT            INTEGER           ----*I*----
                POSITION             1                    1
```

** 103 ** BLOCK NO. 8
AN ACTUAL ARGUMENT IS AN EXPRESSION OR CONSTANT, YET THE CORRESPONDING DUMMY ARGUMENT IS ASSIGNED A VALUE ON ALL PATHS.

```
                              CALLING SUBPROGRAM  CALLED SUBPROGRAM
                                 -*SYSMAIN*-          --*SUB103*-
                ARGUMENT          EXPRESSION          ----*X*----
                POSITION             2                    2
```

** 103 ** BLOCK NO. 11
AN ACTUAL ARGUMENT IS AN EXPRESSION OR CONSTANT, YET THE CORRESPONDING DUMMY ARGUMENT IS ASSIGNED A VALUE ON ALL PATHS.

```
                              CALLING SUBPROGRAM  CALLED SUBPROGRAM
                                 -*SYSMAIN*-          --*SUB106*-
                ARGUMENT            INTEGER           ----*Z*----
                POSITION             2                    2
```

** 104 ** BLOCK NO. 9
THE NUMBER OF DUMMY ARGUMENTS IN CALLED SUBPROGRAM --*SUB103*- DOES NOT AGREE WITH THE NUMBER OF ACTUAL ARGUMENTS SUPPLIED BY CALLING SUBPROGRAM -*SYSMAIN*-.

** 105 ** BLOCK NO. 10
AN ACTUAL ARGUMENT IS A PROCEDURE DECLARED EXTERNAL, YET THE CORRESPONDING DUMMY ARGUMENT IS REFERENCED AS A VARIABLE ON ALL PATHS.

```
                              CALLING SUBPROGRAM  CALLED SUBPROGRAM
                                 -*SYSMAIN*-          --*SUB105*-
                ARGUMENT          --*SUBEX*--         ----*X*----
                POSITION             1                    1
```

** 106 ** BLOCK NO. 10
AN ACTUAL ARGUMENT IS A PROCEDURE DECLARED EXTERNAL, YET THE CORRESPONDING DUMMY ARGUMENT, USED AS A VARIABLE, IS ASSIGNED A VALUE ON ALL PATHS.

```
                              CALLING SUBPROGRAM  CALLED SUBPROGRAM
                                 -*SYSMAIN*-          --*SUB105*-
                ARGUMENT          --*SUBEX*--         ----*X*----
                POSITION             1                    1
```

** 106 ** BLOCK NO. 11
AN ACTUAL ARGUMENT IS A PROCEDURE DECLARED EXTERNAL, YET THE CORRESPONDING DUMMY ARGUMENT, USED AS A VARIABLE, IS ASSIGNED A VALUE ON ALL PATHS.

```
                          CALLING SUBPROGRAM  CALLED SUBPROGRAM
                             -*SYSMAIN*-         --*SUB106*-
                ARGUMENT        --*SUBEX*--         ----*X*----
                POSITION             1                   1
```

** 108 ** BLOCK NO. 18
A SUBPROGRAM REFERENCE CAUSES DUMMY ARGUMENT ----*X*----
TO BECOME ASSOCIATED WITH A COMMON VARIABLE IN THE CALLED
SUBPROGRAM. ----*X*---- IS ASSIGNED A VALUE ON ALL PATHS.

```
                          CALLING SUBPROGRAM  CALLED SUBPROGRAM
                             -*SYSMAIN*-         ---*W201*--
                ARGUMENT        ----*CA*---         ----*X*----
COMMON VARIABLE                 ----*CA*---         ----*CA*---
```

** 108 ** BLOCK NO. 32
A SUBPROGRAM REFERENCE CAUSES DUMMY ARGUMENT ----*X*----
TO BECOME ASSOCIATED WITH A COMMON VARIABLE IN THE CALLED
SUBPROGRAM. ----*X*---- IS ASSIGNED A VALUE ON ALL PATHS.

```
                          CALLING SUBPROGRAM  CALLED SUBPROGRAM
                             -*SYSMAIN*-         ---*W201*--
                ARGUMENT        ----*CA*---         ----*X*----
COMMON VARIABLE                 ----*CA*---         ----*CA*---
```

** 109 ** COMMON VARIABLE ---*Y228*-- IN COMMON BLOCK ---*BLK1*-- IS
REFERENCED ON ALL PATHS IN THE MAIN PROGRAM, YET IT HAS NOT
PREVIOUSLY BEEN ASSIGNED A VALUE, NOR HAS IT BEEN INITIALIZED
IN BLOCK DATA. (SEE NOTE 1)

** 109 ** COMMON VARIABLE ----*CA*--- IN COMMON BLOCK ---*BLK1*-- IS
REFERENCED ON ALL PATHS IN THE MAIN PROGRAM, YET IT HAS NOT
PREVIOUSLY BEEN ASSIGNED A VALUE, NOR HAS IT BEEN INITIALIZED
IN BLOCK DATA. (SEE NOTE 1)

** 110 ** COMMON VARIABLE ----*M*---- IS REFERENCED ON ALL PATHS IN
CALLED SUBPROGRAM ---*E101*--, YET IS NOT INITIALIZED. IT
DOES NOT APPEAR IN BLOCK DATA, AND ITS COMMON BLOCK ---*E110*--
IS NOT AVAILABLE TO CALLING SUBPROGRAM -*SYSMAIN*-. (SEE
NOTE 1)

** 110 ** COMMON VARIABLE ----*B*---- IS REFERENCED ON ALL PATHS IN
CALLED SUBPROGRAM --*SUB103*-, YET IS NOT INITIALIZED. IT
DOES NOT APPEAR IN BLOCK DATA, AND ITS COMMON BLOCK ---*BLK*---
IS NOT AVAILABLE TO CALLING SUBPROGRAM -*SYSMAIN*-. (SEE
NOTE 1)

** 110 ** COMMON VARIABLE ----*D*---- IS REFERENCED ON ALL PATHS IN
CALLED SUBPROGRAM --*SUB208*-, YET IS NOT INITIALIZED. IT
DOES NOT APPEAR IN BLOCK DATA, AND ITS COMMON BLOCK ---*BLK*---
IS NOT AVAILABLE TO CALLING SUBPROGRAM -*SYSMAIN*-. (SEE
NOTE 1)

** 111 ** CONTROL VARIABLE ----*I*---- BECOMES UNDEFINED UPON SATISFACTION
OF ITS DO LOOP AT BLOCK NO. 15, YET IS REFERENCED ON ALL
PATHS THEREAFTER.

```
          ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
            15    16

** 112 **  LOCAL VARIABLE ---*XDAT*-- IS REFERENCED BEFORE BEING ASSIGNED
          A VALUE ON ALL PATHS.
          ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
             1 -  19

** 112 **  LOCAL VARIABLE ----*A*---- IS REFERENCED BEFORE BEING ASSIGNED
          A VALUE ON ALL PATHS.
          ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
             1 -   4

** 112 **  LOCAL VARIABLE ----*M*---- IS REFERENCED BEFORE BEING ASSIGNED
          A VALUE ON ALL PATHS.
          ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
             1 -   6

** 112 **  LOCAL VARIABLE ----*Y*---- IS REFERENCED BEFORE BEING ASSIGNED
          A VALUE ON ALL PATHS.
          ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
             1 -   8

** 112 **  LOCAL VARIABLE ---*LOC*--- IS REFERENCED BEFORE BEING ASSIGNED
          A VALUE ON ALL PATHS.
          ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
             1 -  14

** 112 **  LOCAL VARIABLE ---*D219*-- IS REFERENCED BEFORE BEING ASSIGNED
          A VALUE ON ALL PATHS.
          ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
             1 -  23

** 112 **  LOCAL VARIABLE ----*L*---- IS REFERENCED BEFORE BEING ASSIGNED
          A VALUE ON ALL PATHS.
          ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
             1 -  28

** 112 **  LOCAL VARIABLE ----*D*---- IS REFERENCED BEFORE BEING ASSIGNED
          A VALUE ON ALL PATHS.
          ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
             1 -  35
```

W A R N I N G S

WARNING NUMBER	DESCRIPTION

** 203 ** BLOCK NO. 8
AN ACTUAL ARGUMENT IS AN EXPRESSION OR CONSTANT, YET THE CORRESPONDING DUMMY ARGUMENT IS ASSIGNED A VALUE ON SOME PATHS.

	CALLING SUBPROGRAM -*SYSMAIN*-	CALLED SUBPROGRAM --*SUB103*-
ARGUMENT	EXPRESSION	----*Y*----
POSITION	3	3

** 204 ** BLOCK NO. 7
AN ACTUAL ARGUMENT IS AN EXPRESSION OR CONSTANT, YET THE CORRESPONDING DUMMY ARGUMENT IS NEVER REFERENCED.

	CALLING SUBPROGRAM -*SYSMAIN*-	CALLED SUBPROGRAM ---*E101*--
ARGUMENT	REAL	----*A*----
POSITION	1	1

** 204 ** BLOCK NO. 8
AN ACTUAL ARGUMENT IS AN EXPRESSION OR CONSTANT, YET THE CORRESPONDING DUMMY ARGUMENT IS NEVER REFERENCED.

	CALLING SUBPROGRAM -*SYSMAIN*-	CALLED SUBPROGRAM --*SUB103*-
ARGUMENT	INTEGER	----*I*----
POSITION	1	1

** 204 ** BLOCK NO. 8
AN ACTUAL ARGUMENT IS AN EXPRESSION OR CONSTANT, YET THE CORRESPONDING DUMMY ARGUMENT IS NEVER REFERENCED.

	CALLING SUBPROGRAM -*SYSMAIN*-	CALLED SUBPROGRAM --*SUB103*-
ARGUMENT	EXPRESSION	----*X*----
POSITION	2	2

** 204 ** BLOCK NO. 10
AN ACTUAL ARGUMENT IS AN EXPRESSION OR CONSTANT, YET THE CORRESPONDING DUMMY ARGUMENT IS NEVER REFERENCED.

	CALLING SUBPROGRAM -*SYSMAIN*-	CALLED SUBPROGRAM --*SUB105*-
ARGUMENT	INTEGER	----*I*----
POSITION	2	2

** 204 ** BLOCK NO. 11
AN ACTUAL ARGUMENT IS AN EXPRESSION OR CONSTANT, YET THE CORRESPONDING DUMMY ARGUMENT IS NEVER REFERENCED.

	CALLING SUBPROGRAM -*SYSMAIN*-	CALLED SUBPROGRAM --*SUB106*-
ARGUMENT	INTEGER	----*Z*----
POSITION	2	2

** 205 ** BLOCK NO. 17
AN ACTUAL ARGUMENT IS A PROCEDURE DECLARED EXTERNAL, YET THE CORRESPONDING DUMMY ARGUMENT IS REFERENCED AS A VARIABLE ON SOME PATHS.

	CALLING SUBPROGRAM -*SYSMAIN*-	CALLED SUBPROGRAM ---*SUB*---

	CALLING SUBPROGRAM	CALLED SUBPROGRAM
ARGUMENT	--*SUBEX*--	----*B*----
POSITION	2	2

** 206 ** BLOCK NO. 17
AN ACTUAL ARGUMENT IS A PROCEDURE DECLARED EXTERNAL, YET THE CORRESPONDING DUMMY ARGUMENT, USED AS A VARIABLE, IS ASSIGNED A VALUE ON SOME PATHS.

	CALLING SUBPROGRAM	CALLED SUBPROGRAM
	-*SYSMAIN*-	---*SUB*---
ARGUMENT	--*SUBEX*--	----*B*----
POSITION	2	2

** 208 ** BLOCK NO. 12
A SUBPROGRAM REFERENCE CAUSES DUMMY ARGUMENT ----*X*---- TO BECOME ASSOCIATED WITH A COMMON VARIABLE IN THE CALLED SUBPROGRAM. ----*X*---- IS ASSIGNED A VALUE ON SOME PATHS.

	CALLING SUBPROGRAM	CALLED SUBPROGRAM
	-*SYSMAIN*-	--*SUB208*-
ARGUMENT	----*CA*---	----*X*----
COMMON VARIABLE	----*CA*---	----*CA*---

** 209 ** COMMON VARIABLE ---*CA1*--- IN COMMON BLOCK ----*B1*--- IS REFERENCED ON SOME PATHS IN THE MAIN PROGRAM, YET IT HAS NOT PREVIOUSLY BEEN ASSIGNED A VALUE, NOR HAS IT BEEN INITIALIZED IN BLOCK DATA. (SEE NOTE 1)

** 210 ** COMMON VARIABLE ----*C*---- IS REFERENCED ON SOME PATHS IN CALLED SUBPROGRAM --*SUB103*-, YET IS NOT INITIALIZED. IT DOES NOT APPEAR IN BLOCK DATA, AND ITS COMMON BLOCK ---*BLK*--- IS NOT AVAILABLE TO CALLING SUBPROGRAM -*SYSMAIN*-. (SEE NOTE 1)

** 210 ** COMMON VARIABLE ----*D*---- IS REFERENCED ON SOME PATHS IN CALLED SUBPROGRAM --*SUB103*-, YET IS NOT INITIALIZED. IT DOES NOT APPEAR IN BLOCK DATA, AND ITS COMMON BLOCK ---*BLK*--- IS NOT AVAILABLE TO CALLING SUBPROGRAM -*SYSMAIN*-. (SEE NOTE 1)

** 210 ** COMMON VARIABLE ----*B*---- IS REFERENCED ON SOME PATHS IN CALLED SUBPROGRAM --*SUB208*-, YET IS NOT INITIALIZED. IT DOES NOT APPEAR IN BLOCK DATA, AND ITS COMMON BLOCK ---*BLK*--- IS NOT AVAILABLE TO CALLING SUBPROGRAM -*SYSMAIN*-. (SEE NOTE 1)

** 213 ** BLOCK NO. 9
CORRESPONDING ARGUMENTS HAVE DIFFERENT DATA TYPES.

	CALLING SUBPROGRAM	CALLED SUBPROGRAM
	-*SYSMAIN*-	--*SUB103*-
ARGUMENT	----*A*----	----*I*----
POSITION	1	1
DATA TYPE	REAL	INTEGER

** 213 ** BLOCK NO. 11
CORRESPONDING ARGUMENTS HAVE DIFFERENT DATA TYPES.

	CALLING SUBPROGRAM	CALLED SUBPROGRAM
	-*SYSMAIN*-	--*SUB106*-
ARGUMENT	INTEGER	----*Z*----
POSITION	2	2
DATA TYPE	INTEGER	REAL

** 214 ** CORRESPONDING COMMON VARIABLES IN COMMON BLOCK ---*BLK1*--
HAVE DIFFERENT DATA TYPES.

	CALLING SUBPROGRAM	CALLED SUBPROGRAM
	-*SYSMAIN*-	--*SUB103*-
VARIABLE	----*CA*---	----*J*----
DATA TYPE	REAL	INTEGER

** 215 ** BLOCK NO. 19
CORRESPONDING ARGUMENTS HAVE DIFFERENT DIMENSIONALITY.

	CALLING SUBPROGRAM	CALLED SUBPROGRAM
	-*SYSMAIN*-	--*SUB215*-
ARGUMENT	---*XDAT*--	---*XDAT*--
POSITION	1	1
DIMENSIONS	1	2

** 216 ** COMMON VARIABLE ----*M*---- IS ASSIGNED A VALUE ON ALL PATHS IN CALLED SUBPROGRAM ---*E101*--, YET ITS COMMON BLOCK ---*E110*-- IS NOT AVAILABLE TO CALLING SUBPROGRAM -*SYSMAIN*-. HENCE, A COMPUTED VALUE WILL BE LOST. (SEE NOTE 1)

** 216 ** COMMON VARIABLE ----*B*---- IS ASSIGNED A VALUE ON ALL PATHS IN CALLED SUBPROGRAM --*SUB103*-, YET ITS COMMON BLOCK ---*BLK*--- IS NOT AVAILABLE TO CALLING SUBPROGRAM -*SYSMAIN*-. HENCE, A COMPUTED VALUE WILL BE LOST. (SEE NOTE 1)

** 216 ** COMMON VARIABLE ----*D*---- IS ASSIGNED A VALUE ON ALL PATHS IN CALLED SUBPROGRAM --*SUB103*-, YET ITS COMMON BLOCK ---*BLK*--- IS NOT AVAILABLE TO CALLING SUBPROGRAM -*SYSMAIN*-. HENCE, A COMPUTED VALUE WILL BE LOST. (SEE NOTE 1)

** 216 ** COMMON VARIABLE ----*C*---- IS ASSIGNED A VALUE ON ALL PATHS IN CALLED SUBPROGRAM --*SUB215*-, YET ITS COMMON BLOCK ---*BLK*--- IS NOT AVAILABLE TO CALLING SUBPROGRAM -*SYSMAIN*-. HENCE, A COMPUTED VALUE WILL BE LOST. (SEE NOTE 1)

** 217 ** COMMON VARIABLE ----*C*---- IS ASSIGNED A VALUE ON SOME PATHS IN CALLED SUBPROGRAM --*SUB103*-, YET ITS COMMON BLOCK ---*BLK*--- IS NOT AVAILABLE TO CALLING SUBPROGRAM -*SYSMAIN*-. HENCE, A COMPUTED VALUE MAY BE LOST. (SEE NOTE 1)

** 217 ** COMMON VARIABLE ----*D*---- IS ASSIGNED A VALUE ON SOME PATHS IN CALLED SUBPROGRAM --*SUB215*-, YET ITS COMMON BLOCK ---*BLK*--- IS NOT AVAILABLE TO CALLING SUBPROGRAM -*SYSMAIN*-. HENCE, A COMPUTED VALUE MAY BE LOST. (SEE NOTE 1)

```
** 218 **  COMMON VARIABLE ----*T*---- IS INITIALIZED IN BLOCK DATA.
           IT IS ASSIGNED A VALUE ON ALL PATHS IN CALLED SUBPROGRAM
           --*SUB215*-, YET ITS COMMON BLOCK ---*IBD*--- IS NOT AVAILABLE
           TO CALLING SUBPROGRAM -*SYSMAIN*-.  HENCE, UNDEFINITION WILL
           OCCUR UPON EXIT FROM --*SUB215*-. (SEE NOTE 2)

** 219 **  COMMON VARIABLE ----*W*---- IS INITIALIZED IN BLOCK DATA.
           IT IS ASSIGNED A VALUE ON SOME PATHS IN CALLED SUBPROGRAM
           --*SUB215*-, YET ITS COMMON BLOCK ---*IBD*--- IS NOT AVAILABLE
           TO CALLING SUBPROGRAM -*SYSMAIN*-.  HENCE, UNDEFINITION MAY
           OCCUR UPON EXIT FROM --*SUB215*-. (SEE NOTE 2)

** 226 **  IN THE MAIN PROGRAM, COMMON VARIABLE ----*CA*--- IS
           ASSIGNED A VALUE IN BLOCK  NO.    20 AND IS EITHER
           ASSIGNED A VALUE THEREAFTER BEFORE BEING REFERENCED,
           OR IS NOT SUBSEQUENTLY REFERENCED, ON ALL PATHS.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
             20    21

** 227 **  IN THE MAIN PROGRAM, COMMON VARIABLE ----*BA*--- IS
           ASSIGNED A VALUE IN BLOCK  NO.    22 AND IS EITHER
           ASSIGNED A VALUE THEREAFTER BEFORE BEING REFERENCED,
           OR IS NOT SUBSEQUENTLY REFERENCED, ON SOME PATHS.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
             22    23    25 -  36

** 228 **  IN THE MAIN PROGRAM, AN ELEMENT OF THE COMMON ARRAY
            ---*Y228*-- IS ASSIGNED A VALUE IN BLOCK NO.    25
           AND THE ARRAY IS NOT SUBSEQUENTLY REFERENCED ON ANY PATH.

** 229 **  LOCAL VARIABLE ---*I220*-- IS ASSIGNED A VALUE IN BLOCK
           NO.    1 AND IS EITHER ASSIGNED A VALUE THEREAFTER BEFORE
           BEING REFERENCED, OR IS NOT SUBSEQUENTLY REFERENCED,
           ON ALL PATHS.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
              1 -  36

** 229 **  LOCAL VARIABLE ---*X221*-- IS ASSIGNED A VALUE IN BLOCK
           NO.    1 AND IS EITHER ASSIGNED A VALUE THEREAFTER BEFORE
           BEING REFERENCED, OR IS NOT SUBSEQUENTLY REFERENCED,
           ON ALL PATHS.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
              1 -  36

** 229 **  LOCAL VARIABLE ----*X*---- IS ASSIGNED A VALUE IN BLOCK
           NO.    5 AND IS EITHER ASSIGNED A VALUE THEREAFTER BEFORE
           BEING REFERENCED, OR IS NOT SUBSEQUENTLY REFERENCED,
           ON ALL PATHS.
           ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
              5     6

** 229 **  LOCAL VARIABLE ----*K*---- IS ASSIGNED A VALUE IN BLOCK
           NO.   14 AND IS EITHER ASSIGNED A VALUE THEREAFTER BEFORE
           BEING REFERENCED, OR IS NOT SUBSEQUENTLY REFERENCED,
```

ON ALL PATHS.
ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
14 15 16

** 229 ** LOCAL VARIABLE ---*X229*-- IS ASSIGNED A VALUE IN BLOCK NO. 26 AND IS EITHER ASSIGNED A VALUE THEREAFTER BEFORE BEING REFERENCED, OR IS NOT SUBSEQUENTLY REFERENCED, ON ALL PATHS.
ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
26 - 36

** 229 ** LOCAL VARIABLE ---*X230*-- IS ASSIGNED A VALUE IN BLOCK NO. 27 AND IS EITHER ASSIGNED A VALUE THEREAFTER BEFORE BEING REFERENCED, OR IS NOT SUBSEQUENTLY REFERENCED, ON ALL PATHS.
ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
27 28 29

** 231 ** AN ELEMENT OF THE LOCAL ARRAY ---*XDAT*-- IS ASSIGNED A VALUE IN BLOCK NO. 30 AND THE ARRAY IS NOT SUBSEQUENTLY REFERENCED ON ANY PATH.

** 232 ** BLOCK NO. 31
A POSSIBLE ILLEGAL SIDE EFFECT HAS BEEN DETECTED. IT OCCURS VIA A VARIABLE PASSED IN AN ARGUMENT LIST. THIS VARIABLE HAS APPEARED AT LEAST TWICE IN A STATEMENT -- IN ONE APPEARANCE IT IS USED AS STRICT INPUT AND IN THE OTHER AS STRICT OUTPUT.

	CALLING SUBPROGRAM	CALLED SUBPROGRAM
	-*SYSMAIN*-	---*W201*--
ARGUMENT	----*X*----	----*X*----
POSITION	1	1

** 232 ** BLOCK NO. 32
A POSSIBLE ILLEGAL SIDE EFFECT HAS BEEN DETECTED. IT OCCURS VIA A VARIABLE PASSED IN AN ARGUMENT LIST. THIS VARIABLE HAS APPEARED AT LEAST TWICE IN A STATEMENT -- IN ONE APPEARANCE IT IS USED AS STRICT INPUT AND IN THE OTHER AS STRICT OUTPUT.

	CALLING SUBPROGRAM	CALLED SUBPROGRAM
	-*SYSMAIN*-	---*W201*--
ARGUMENT	----*CA*---	----*X*----
POSITION	1	1

** 233 ** BLOCK NO. 18
A POSSIBLE ILLEGAL SIDE EFFECT HAS BEEN DETECTED. IT OCCURS VIA A COMMON VARIABLE WHICH HAS BEEN REFERENCED (POSSIBLY INDIRECTLY) AT LEAST TWICE IN A STATEMENT -- IN ONE APPEARANCE IT IS USED AS STRICT INPUT AND IN THE OTHER AS STRICT OUTPUT.

	CALLING SUBPROGRAM	CALLED SUBPROGRAM
	-*SYSMAIN*-	---*W201*--
VARIABLE	----*CA*---	----*CA*---
COMMON BLOCK	---*BLK1*--	---*BLK1*--

** 233 ** BLOCK NO. 32

A POSSIBLE ILLEGAL SIDE EFFECT HAS BEEN DETECTED. IT OCCURS VIA A COMMON VARIABLE WHICH HAS BEEN REFERENCED (POSSIBLY INDIRECTLY) AT LEAST TWICE IN A STATEMENT -- IN ONE APPEARANCE IT IS USED AS STRICT INPUT AND IN THE OTHER AS STRICT OUTPUT.

	CALLING SUBPROGRAM	CALLED SUBPROGRAM
	-*SYSMAIN*-	---*W201*--
VARIABLE	----*CA*---	----*CA*---
COMMON BLOCK	---*BLK1*--	---*BLK1*--

** 234 ** BLOCK NO. 33

A POSSIBLE ILLEGAL SIDE EFFECT HAS BEEN DETECTED. IT OCCURS VIA A GLOBAL VARIABLE REFERENCED IN AN ARITHMETIC STATEMENT FUNCTION. THIS VARIABLE HAS APPEARED AT LEAST TWICE IN A STATEMENT -- IN ONE APPEARANCE IT IS USED AS STRICT INPUT AND IN THE OTHER AS STRICT OUTPUT.

	CALLING SUBPROGRAM	CALLED SUBPROGRAM
	-*SYSMAIN*-	--*LASRF*--
VARIABLE	----*C*----	--* *-

** 236 ** LOCAL VARIABLE ---*XDT*--- IS NEVER ASSIGNED A VALUE.

** 236 ** LOCAL VARIABLE ---*I236*-- IS NEVER ASSIGNED A VALUE.

** 236 ** LOCAL VARIABLE ----*M*---- IS NEVER ASSIGNED A VALUE.

** 236 ** LOCAL VARIABLE ----*Y*---- IS NEVER ASSIGNED A VALUE.

** 236 ** LOCAL VARIABLE ---*LOC*--- IS NEVER ASSIGNED A VALUE.

** 236 ** LOCAL VARIABLE ---*D219*-- IS NEVER ASSIGNED A VALUE.

** 236 ** LOCAL VARIABLE ----*L*---- IS NEVER ASSIGNED A VALUE.

** 236 ** LOCAL VARIABLE ----*D*---- IS NEVER ASSIGNED A VALUE.

M E S S A G E S

MESSAGE NUMBER	DESCRIPTION

** 301 ** COMMON VARIABLE ---*D218*-- IN BLOCK ---*BLK1*-- OF SUBPROGRAM -*SYSMAIN*- IS INITIALIZED IN BLOCK DATA.

** 301 ** COMMON VARIABLE ----*W*---- IN BLOCK ---*IBD*--- OF
SUBPROGRAM --*SUB215*- IS INITIALIZED IN BLOCK DATA.

** 303 ** THE FOLLOWING DATA FLOW OCCURS THROUGH COMMON WHEN SUBPROGRAM
--*SUB103*- IS CALLED.

COMMON BLOCK	VARIABLE	INPUT CLASSIFICATION	OUTPUT CLASSIFICATION
---*BLK1*--	----*CA*---	STRICT	NON
---*BLK1*--	---*D218*--	STRICT	OUTPUT
---*BLK1*--	---*Y228*--	STRICT	OUTPUT

** 303 ** THE FOLLOWING DATA FLOW OCCURS THROUGH COMMON WHEN SUBPROGRAM
---*W201*-- IS CALLED.

COMMON BLOCK	VARIABLE	INPUT CLASSIFICATION	OUTPUT CLASSIFICATION
---*BLK1*--	----*CA*---	STRICT	NON

** 304 ** I/O CLASSIFICATION OF ARGUMENTS AND COMMON VARIABLES
FOR -*SYSMAIN*-

COMMON BLOCK ----*B1*---

AVAILABILITY = ORIGINAL

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	---*CA1*---	INPUT	NON
2	----*BA*---	NON	STRICT

COMMON BLOCK ---*BLK1*--

AVAILABILITY = ORIGINAL

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	----*CA*---	STRICT	STRICT
2	---*D218*--	STRICT	OUTPUT
3	---*Y228*--	STRICT	STRICT

--*LASRF*--

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	----*X*----	STRICT	NON
2	----*Y*----	STRICT	NON

NOTE 1 ALTHOUGH DETECTED IN THIS SUBPROGRAM, THE CAUSE FOR THIS DIAGNOSTIC MAY HAVE OCCURRED AT A DEEPER LEVEL OF SUBPROGRAM REFERENCES AND BEEN PROPAGATED UP TO THIS ONE.

NOTE 2 IF MESSAGE 301 CONCERNING THIS VARIABLE APPEARS IN THE OUTPUT, IT MAY PROVIDE ADDITIONAL USEFUL INFORMATION ABOUT THE DATA FLOW AMONG SUBPROGRAMS.

$ IN THE CONTINUATION FIELD INDICATES THE EXPANSION
OF THE LOGICAL IF STATEMENT ON THE PREVIOUS LINE

```
BLOCK                    SOURCE

    1          BLOCK DATA
    1          COMMON/IBD/B,C,D
    1          COMMON /BLK1/CA,D218,Y228(6)
    1          DATA B,C,D/1.,2.,3./
    1          DATA D218/1./
    1          END
```

** N O E R R O R S **
- - - - - - - -

** N O W A R N I N G S **
- - - - - - - - - -

M E S S A G E S
- - - - - - - -

MESSAGE NUMBER	DESCRIPTION

** 304 ** I/O CLASSIFICATION OF ARGUMENTS AND COMMON VARIABLES FOR *BLOCKDATA*

COMMON BLOCK ---*IBD*---

AVAILABILITY = ORIGINAL

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	----*B*----	NON	STRICT
2	----*C*----	NON	STRICT
3	----*D*----	NON	STRICT

COMMON BLOCK ---*BLK1*--

AVAILABILITY = ORIGINAL

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	----*CA*---	NON	NON
2	---*D218*--	NON	STRICT

$ IN THE CONTINUATION FIELD INDICATES THE EXPANSION
OF THE LOGICAL IF STATEMENT ON THE PREVIOUS LINE

```
BLOCK                    SOURCE

   1         FUNCTION E101(A)
   1         COMMON/BLK/B,C,D
   1         COMMON/E110/M
   0  C
   0  C   FUNCTION CAUSES ERROR101
   0  C      FUNCTION NAME REF BEFORE BEING ASSIGNED A VALUE
   0  C   ERROR 102
   0  C      FUNCTION NEVER ASSIGNED A VALUE
   2         A = 1
   0  C
   3         R = E101
   0  C
   0  C   ERROR 110
   0  C      VAR M NOT INITIALIZED
   0  C
   4         X = M+1
   0  C
   0  C   WARNING 212
   0  C      LOCAL VAR REFERENCED BEFORE BEING ASSIGNED ON SOME PATHS
   0  C
   5         IF(X.EQ.R)
   6       $ K=I
   0  C   WARNING 211
   0  C       CONTROL VARIABLE REFERENCED ON SOME PATHS AFTER BECOMING
   0  C       UNDEFINED
   7         DO 5 K=1,10
   8         M=K*M+M
   9       5 CONTINUE
  10         IF(M.GT.100)
  11        $RETURN
  12         A=K
  13         RETURN
   1         END
```

E R R O R S
- - - - - -

ERROR NUMBER	DESCRIPTION
** 101 **	FUNCTION NAME ---*E101*-- IS REFERENCED BEFORE BEING ASSIGNED A VALUE ON ALL PATHS. ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS 1 2 3

** 102 ** FUNCTION NAME ---*E101*-- IS NEVER ASSIGNED A VALUE.

W A R N I N G S

WARNING NUMBER	DESCRIPTION

** 211 ** CONTROL VARIABLE ----*K*---- BECOMES UNDEFINED UPON SATISFACTION OF ITS DO LOOP AT BLOCK NO. 9, YET IS REFERENCED ON SOME PATHS THEREAFTER.
ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
9 10 12

** 212 ** LOCAL VARIABLE ----*I*---- IS REFERENCED BEFORE BEING ASSIGNED A VALUE ON SOME PATHS.
ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
1 - 6

** 223 ** DUMMY ARGUMENT ----*A*---- IS ASSIGNED A VALUE IN BLOCK NO. 2 AND IS ASSIGNED A VALUE THEREAFTER BEFORE BEING REFERENCED, ON SOME PATHS.
ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
2 - 10 12

** 229 ** LOCAL VARIABLE ----*K*---- IS ASSIGNED A VALUE IN BLOCK NO. 6 AND IS EITHER ASSIGNED A VALUE THEREAFTER BEFORE BEING REFERENCED, OR IS NOT SUBSEQUENTLY REFERENCED, ON ALL PATHS.
ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
6 7

** 236 ** LOCAL VARIABLE ----*I*---- IS NEVER ASSIGNED A VALUE.

M E S S A G E S

MESSAGE NUMBER	DESCRIPTION

** 304 ** I/O CLASSIFICATION OF ARGUMENTS AND COMMON VARIABLES FOR FUNCTION ---*E101*--

```
ARGUMENTS
      POSITION      NAME      INPUT CLASS      OUTPUT CLASS

         0       ---*E101*--      STRICT            NON
         1       ----*A*----        NON           STRICT

COMMON BLOCK      ---*BLK*---

      AVAILABILITY = ORIGINAL

ARGUMENTS
      POSITION      NAME      INPUT CLASS      OUTPUT CLASS

         1       ----*B*----        NON             NON
         2       ----*C*----        NON             NON
         3       ----*D*----        NON             NON

COMMON BLOCK      ---*E110*--

      AVAILABILITY = ORIGINAL

ARGUMENTS
      POSITION      NAME      INPUT CLASS      OUTPUT CLASS

         1       ----*M*----      STRICT           STRICT
```

$ IN THE CONTINUATION FIELD INDICATES THE EXPANSION
OF THE LOGICAL IF STATEMENT ON THE PREVIOUS LINE

```
BLOCK                    SOURCE

  1          SUBROUTINE SUB103(I,X,Y)
  1          COMMON/BLK/B,C,D
  1          COMMON/BLK1/J,Y(7)
  2          I=4
  3          X=6.
  4          IF(Y.GT.X)
  5         $Y=X
  6          B=B+1
  0   C
  0   C  CALL IS TO HELP GENERATE MESSAGE 302
  0   C
  7          CALL SUB302
  0   C
  0   C  HELPS GENERATE WARNING 210 IN MAIN PROGRAM
  0   C
  8          IF(B.EQ.1.)
  9         $ C=D
  0   C
  0   C  WARNING 224
  0   C     COMMON REDEFINED ON ALL PATHS BEFORE BEING REFERENCED
  0   C
 10          B=1.
 11          B=2.
  0   C
  0   C  WARNING 225
  0   C     SAME AS 224 BUT ON SOME PATHS
 12          D = B
 13          IF(X.EQ.C)
 14         $ D=B+1
 15          RETURN
  1          END
```

E R R O R S
- - - - - -

ERROR NUMBER	DESCRIPTION
** 107 **	THE NAME ----*Y*---- IS USED TO REPRESENT BOTH A DUMMY ARGUMENT AND A COMMON VARIABLE IN THIS SUBPROGRAM.

W A R N I N G S

WARNING NUMBER | DESCRIPTION

** 214 ** CORRESPONDING COMMON VARIABLES IN COMMON BLOCK ---*BLK1*-- HAVE DIFFERENT DATA TYPES.

	CALLING SUBPROGRAM --*SUB103*-	CALLED SUBPROGRAM --*SUB302*-
VARIABLE	----*Y*----	----*K*----
DATA TYPE	REAL	INTEGER

** 224 ** COMMON VARIABLE ----*B*---- IS ASSIGNED A VALUE IN BLOCK NO. 10 AND IS ASSIGNED A VALUE THEREAFTER BEFORE BEING REFERENCED, ON ALL PATHS.
ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
10 11

** 225 ** COMMON VARIABLE ----*D*---- IS ASSIGNED A VALUE IN BLOCK NO. 12 AND IS ASSIGNED A VALUE THEREAFTER BEFORE BEING REFERENCED, ON SOME PATHS.
ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
12 13 14

** 237 ** CORRESPONDING COMMON VARIABLES IN COMMON BLOCK ---*BLK1*-- HAVE DIFFERENT DATA TYPES IN SUBPROGRAM --*SUB103*- AND BLOCK DATA.

	SUBPROGRAM --*SUB103*-	BLOCK DATA
VARIABLE	----*J*----	----*CA*---
DATA TYPE	INTEGER	REAL

M E S S A G E S

MESSAGE NUMBER | DESCRIPTION

** 303 ** THE FOLLOWING DATA FLOW OCCURS THROUGH COMMON WHEN SUBPROGRAM --*SUB302*- IS CALLED.

COMMON BLOCK	VARIABLE	INPUT CLASSIFICATION	OUTPUT CLASSIFICATION
---*BLK1*--	----*Y*----	STRICT	NON
---*BLK1*--	----*J*----	STRICT	NON

** 304 ** I/O CLASSIFICATION OF ARGUMENTS AND COMMON VARIABLES
FOR SUBROUTINE --*SUB103*-

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	----*I*----	NON	STRICT
2	----*X*----	NON	STRICT
3	----*Y*----	STRICT	OUTPUT

COMMON BLOCK ---*BLK*---

AVAILABILITY = ORIGINAL

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	----*B*----	STRICT	STRICT
2	----*C*----	INPUT	OUTPUT
3	----*D*----	INPUT	STRICT

COMMON BLOCK ---*BLK1*--

AVAILABILITY = ORIGINAL

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	----*J*----	STRICT	NON
2	----*Y*----	STRICT	OUTPUT

$ IN THE CONTINUATION FIELD INDICATES THE EXPANSION
OF THE LOGICAL IF STATEMENT ON THE PREVIOUS LINE

```
BLOCK                    SOURCE

   1         SUBROUTINE SUB302
   0   C   WARNING 214
   0   C    COMMON VARIABLES HAVE DIFFERENT DATA TYPES IN CALLING AND CALLED
   0   C   WARNING 237
   0   C    COMMON VARIABLES HAVE DIFFERENT DATA TYPES HERE AND IN BLOCK DATA
   0   C
   1         COMMON/BLK1/K(8)
   0   C
   0   C   SUBROUTINE WILL HELP GENERATE MESSAGE 302
   0   C
   2         CALL SUB106(X,Y)
   3         I=1+K(1)
   0   C
   0   C   ERROR 111
   0   C      CONTROL VAR IS REFERENCED OUTSIDE OF LOOP
   0   C
   4         DO 100 J = 1 , 5
   5         X = 1
   6     100 CONTINUE
   7         I = J
   8         RETURN
   1         END
```

E R R O R S

ERROR NUMBER	DESCRIPTION

** 111 ** CONTROL VARIABLE ----*J*---- BECOMES UNDEFINED UPON SATISFACTION OF ITS DO LOOP AT BLOCK NO. 6, YET IS REFERENCED ON ALL PATHS THEREAFTER.
ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
6 7

W A R N I N G S

WARNING

** 229 ** LOCAL VARIABLE ----*X*---- IS ASSIGNED A VALUE IN BLOCK NO. 5 AND IS EITHER ASSIGNED A VALUE THEREAFTER BEFORE BEING REFERENCED, OR IS NOT SUBSEQUENTLY REFERENCED, ON ALL PATHS.
ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
5 - 8

** 229 ** LOCAL VARIABLE ----*I*---- IS ASSIGNED A VALUE IN BLOCK NO. 3 AND IS EITHER ASSIGNED A VALUE THEREAFTER BEFORE BEING REFERENCED, OR IS NOT SUBSEQUENTLY REFERENCED, ON ALL PATHS.
ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
3 - 7

** 237 ** CORRESPONDING COMMON VARIABLES IN COMMON BLOCK ---*BLK1*-- HAVE DIFFERENT DATA TYPES IN SUBPROGRAM --*SUB302*- AND BLOCK DATA.

	SUBPROGRAM --*SUB302*-	BLOCK DATA
VARIABLE	----*K*----	---*Y228*--
DATA TYPE	INTEGER	REAL

** 237 ** CORRESPONDING COMMON VARIABLES IN COMMON BLOCK ---*BLK1*-- HAVE DIFFERENT DATA TYPES IN SUBPROGRAM --*SUB302*- AND BLOCK DATA.

	SUBPROGRAM --*SUB302*-	BLOCK DATA
VARIABLE	----*K*----	----*CA*---
DATA TYPE	INTEGER	REAL

** 237 ** CORRESPONDING COMMON VARIABLES IN COMMON BLOCK ---*BLK1*-- HAVE DIFFERENT DATA TYPES IN SUBPROGRAM --*SUB302*- AND BLOCK DATA.

	SUBPROGRAM --*SUB302*-	BLOCK DATA
VARIABLE	----*K*----	---*D218*--
DATA TYPE	INTEGER	REAL

M E S S A G E S

MESSAGE NUMBER — DESCRIPTION

** 302 ** THE FOLLOWING COMMON BLOCKS, ALTHOUGH NOT EXPLICITLY IN SUBPROGRAM --*SUB302*-, ARE AVAILABLE TO IT.

```
                    COMMON BLOCK          AVAILABILITY

                     ---*BLK*---                ALWAYS
** 304 ** I/O CLASSIFICATION OF ARGUMENTS AND COMMON VARIABLES
          FOR SUBROUTINE --*SUB302*-
               THERE ARE NO PARAMETERS OR COMMON BLOCKS

          COMMON BLOCK      ---*BLK1*--

                AVAILABILITY = ORIGINAL

          ARGUMENTS
                POSITION      NAME      INPUT CLASS      OUTPUT CLASS

                   1        ----*K*----      STRICT             NON

          COMMON BLOCK      ---*BLK*---

                AVAILABILITY = ALWAYS

          VARIABLE NAMES TAKEN FROM SUBPROGRAM --*SUB106*-

          ARGUMENTS
                POSITION      NAME      INPUT CLASS      OUTPUT CLASS

                   1        ----*B*----        NON              NON
                   2        ----*C*----        NON              NON
                   3        ----*D*----        NON              NON
```

$ IN THE CONTINUATION FIELD INDICATES THE EXPANSION
OF THE LOGICAL IF STATEMENT ON THE PREVIOUS LINE

```
BLOCK                    SOURCE

   1         SUBROUTINE SUB105(X,I)
   0   C
   0   C   ERROR 105  GENERATED BY THIS ROUTINE IN MAIN PROGRAM
   0   C   ALSO AIDS IN GENERATION OF MESSAGE 302
   2         Y = X+4
   3         CALL SUB106(X,Y)
   4         RETURN
   1         END
```

** N O E R R O R S **

W A R N I N G S

WARNING NUMBER	DESCRIPTION
** 207 **	DUMMY ARGUMENT ----*I*---- IS NEVER USED.
** 229 **	LOCAL VARIABLE ----*Y*---- IS ASSIGNED A VALUE IN BLOCK NO. 2 AND IS EITHER ASSIGNED A VALUE THEREAFTER BEFORE BEING REFERENCED, OR IS NOT SUBSEQUENTLY REFERENCED, ON ALL PATHS. ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS 2 3

M E S S A G E S

MESSAGE NUMBER	DESCRIPTION
** 304 **	I/O CLASSIFICATION OF ARGUMENTS AND COMMON VARIABLES FOR SUBROUTINE --*SUB105*-

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	----*X*----	STRICT	STRICT
2	----*I*----	NON	NON

$ IN THE CONTINUATION FIELD INDICATES THE EXPANSION
OF THE LOGICAL IF STATEMENT ON THE PREVIOUS LINE

```
BLOCK                    SOURCE

   1         SUBROUTINE SUB106(X,Z)
   0   C
   0   C  SUBROUTINE WILL GENERATE ERROR 106 IN MAIN PROGRAM
   0   C
   2         X=6.
   0   C
   0   C  AID WITH MESSAGE 302
   3         Y=E101(Z)
   0   C
   0   C  WILL GENERATE WARNING 203 IN MAIN PROGRAM
   0   C
   4         IF(Y.LT.X)
   5        $Z=3
   6         RETURN
   1         END
```

E R R O R S

ERROR NUMBER	DESCRIPTION
** 110 **	COMMON VARIABLE ----*M*---- IS REFERENCED ON ALL PATHS IN CALLED SUBPROGRAM ---*E101*--, YET IS NOT INITIALIZED. IT DOES NOT APPEAR IN BLOCK DATA, AND ITS COMMON BLOCK ---*E110*-- IS NOT AVAILABLE TO CALLING SUBPROGRAM --*SUB106*-. (SEE NOTE 1)

W A R N I N G S

WARNING NUMBER	DESCRIPTION
** 216 **	COMMON VARIABLE ----*M*---- IS ASSIGNED A VALUE ON ALL PATHS IN CALLED SUBPROGRAM ---*E101*--, YET ITS COMMON BLOCK ---*E110*-- IS NOT AVAILABLE TO CALLING SUBPROGRAM --*SUB106*-.

HENCE, A COMPUTED VALUE WILL BE LOST. (SEE NOTE 1)

M E S S A G E S

MESSAGE NUMBER — DESCRIPTION

** 302 ** THE FOLLOWING COMMON BLOCKS, ALTHOUGH NOT EXPLICITLY IN SUBPROGRAM --*SUB106*-, ARE AVAILABLE TO IT.

COMMON BLOCK	AVAILABILITY
---*BLK*---	SOMETIMES

** 304 ** I/O CLASSIFICATION OF ARGUMENTS AND COMMON VARIABLES FOR SUBROUTINE --*SUB106*-

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	----*X*----	NON	STRICT
2	----*Z*----	NON	STRICT

COMMON BLOCK ---*BLK*---

AVAILABILITY = SOMETIMES

VARIABLE NAMES TAKEN FROM SUBPROGRAM ---*E101*--

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	----*B*----	NON	NON
2	----*C*----	NON	NON
3	----*D*----	NON	NON

NOTES

NOTE 1 ---- - ALTHOUGH DETECTED IN THIS SUBPROGRAM, THE CAUSE FOR THIS DIAGNOSTIC MAY HAVE OCCURRED AT A DEEPER LEVEL OF SUBPROGRAM REFERENCES AND BEEN PROPAGATED UP TO THIS ONE.

NOTE 2 ---- - IF MESSAGE 301 CONCERNING THIS VARIABLE APPEARS IN THE OUTPUT, IT MAY PROVIDE ADDITIONAL USEFUL INFORMATION ABOUT THE DATA FLOW AMONG SUBPROGRAMS.

$ IN THE CONTINUATION FIELD INDICATES THE EXPANSION
OF THE LOGICAL IF STATEMENT ON THE PREVIOUS LINE

```
BLOCK                    SOURCE

   1         SUBROUTINE SUB208(B,X)
   1         COMMON/BLK1/CA
   1         COMMON/BLK/B,C,D
   0   C
   0   C  ERROR 107
   0   C      COMMON VARIABLE IS ALSO USED AS A DUMMY ARGUMENT
   0   C
   2         IF(Y.EQ.D)
   3        $ GO TO 100
   4         X=B
   5     100 RETURN
   1         END
```

E R R O R S

ERROR NUMBER	DESCRIPTION
** 107 **	THE NAME ----*B*---- IS USED TO REPRESENT BOTH A DUMMY ARGUMENT AND A COMMON VARIABLE IN THIS SUBPROGRAM.
** 112 **	LOCAL VARIABLE ----*Y*---- IS REFERENCED BEFORE BEING ASSIGNED A VALUE ON ALL PATHS. ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS 1 2

W A R N I N G S

WARNING NUMBER	DESCRIPTION
** 236 **	LOCAL VARIABLE ----*Y*---- IS NEVER ASSIGNED A VALUE.

M E S S A G E S

MESSAGE NUMBER — DESCRIPTION

** 304 ** I/O CLASSIFICATION OF ARGUMENTS AND COMMON VARIABLES
FOR SUBROUTINE --*SUB208*-

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	----*B*----	INPUT	NON
2	----*X*----	NON	OUTPUT

COMMON BLOCK ---*BLK1*--

AVAILABILITY = ORIGINAL

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	----*CA*---	NON	NON

COMMON BLOCK ---*BLK*---

AVAILABILITY = ORIGINAL

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	----*B*----	INPUT	NON
2	----*C*----	NON	NON
3	----*D*----	STRICT	NON

$ IN THE CONTINUATION FIELD INDICATES THE EXPANSION
OF THE LOGICAL IF STATEMENT ON THE PREVIOUS LINE

```
BLOCK                    SOURCE

  1          FUNCTION W201(X)
  1          COMMON/BLK1/CA
  0    C
  0    C   WARNING 202
  0    C       FUNCTION NAME IS UNDEFINED ON SOME PATHS
  0    C
  2          Y=5
  0    C
  0    C   WARNING 222
  0    C       DUMMY VAR IS CHANGED BEFORE BEING REF ON ALL PATHS
  0    C
  3          X = 1
  4          X=CA
  5          IF(Y.GT.X)
  6         $ GO TO 100
  7          RETURN
  0    C
  0    C   WARNING 201
  0    C       FUNCTION NAME IS REF BEFORE BEING ASSIGNED ON SOME PATHS
  0    C
  8      100 I=W201
  9          W201=1.
 10          RETURN
  1          END
```

** N O E R R O R S **
- - - - - - - -

W A R N I N G S
- - - - - - - -

WARNING NUMBER	DESCRIPTION
** 201 **	FUNCTION NAME ---*W201*-- IS REFERENCED BEFORE BEING ASSIGNED A VALUE ON SOME PATHS. ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS 1 - 6 8
** 202 **	FUNCTION NAME ---*W201*-- IS NOT ASSIGNED A VALUE ON SOME PATHS.

** 222 ** DUMMY ARGUMENT ----*X*---- IS ASSIGNED A VALUE IN BLOCK NO. 3 AND IS ASSIGNED A VALUE THEREAFTER BEFORE BEING REFERENCED, ON ALL PATHS.
ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
3 4

** 229 ** LOCAL VARIABLE ----*I*---- IS ASSIGNED A VALUE IN BLOCK NO. 8 AND IS EITHER ASSIGNED A VALUE THEREAFTER BEFORE BEING REFERENCED, OR IS NOT SUBSEQUENTLY REFERENCED, ON ALL PATHS.
ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
8 9 10

M E S S A G E S

MESSAGE NUMBER	DESCRIPTION

** 304 ** I/O CLASSIFICATION OF ARGUMENTS AND COMMON VARIABLES FOR FUNCTION ---*W201*--

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
0	---*W201*--	INPUT	OUTPUT
1	----*X*----	NON	STRICT

COMMON BLOCK ---*BLK1*--

AVAILABILITY = ORIGINAL

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	----*CA*---	STRICT	NON

$ IN THE CONTINUATION FIELD INDICATES THE EXPANSION
OF THE LOGICAL IF STATEMENT ON THE PREVIOUS LINE

```
BLOCK                    SOURCE

  1         SUBROUTINE SUB215(XDAT,A,B)
  1         COMMON/IBD/W,V,T
  1         COMMON/BLK/B1,C,D
  1         DIMENSION XDAT(5,2)
  0   C  WARNING 223
  0   C       TYPE II ANOMALY, DUMMY ARGUMENT
  0   C
  0   C
  2         B=XDAT(2,1)
  3         A =2.
  4         IF(XDAT(1,1).EQ.2.)
  5        $ B=3.
  6         Y=1
  7         IF(B.EQ.3)
  8        $ B=Y
  9         C=B*Y
 10         T=W*Y+B
 11         IF(C.GT.100)
 12        $GO TO 10
 13         D=A*B
 14         W=B**2
 15      10 RETURN
  1         END
```

** N O E R R O R S **
- - - - - - - -

W A R N I N G S
- - - - - - - - -

WARNING NUMBER	DESCRIPTION

** 223 ** DUMMY ARGUMENT ----*B*---- IS ASSIGNED A VALUE IN BLOCK NO. 2 AND IS ASSIGNED A VALUE THEREAFTER BEFORE BEING REFERENCED, ON SOME PATHS.
ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
2 - 5

MESSAGE NUMBER ------ | DESCRIPTION -----------

** 304 ** I/O CLASSIFICATION OF ARGUMENTS AND COMMON VARIABLES FOR SUBROUTINE --*SUB215*-

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	---*XDAT*--	STRICT	NON
2	----*A*----	NON	STRICT
3	----*B*----	NON	STRICT

COMMON BLOCK ---*IBD*---

AVAILABILITY = ORIGINAL

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	----*W*----	STRICT	OUTPUT
2	----*V*----	NON	NON
3	----*T*----	NON	STRICT

COMMON BLOCK ---*BLK*---

AVAILABILITY = ORIGINAL

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	----*B1*---	NON	NON
2	----*C*----	NON	STRICT
3	----*D*----	NON	OUTPUT

$ IN THE CONTINUATION FIELD INDICATES THE EXPANSION
OF THE LOGICAL IF STATEMENT ON THE PREVIOUS LINE

```
BLOCK                    SOURCE

   1         SUBROUTINE SUB(A,B)
   1         DATA X,Y/1.,2./
   0   C  WARNING 220 AND 221
   0   C       LOCAL VARIABLE INITIALIZED IN DATA STATEMENT IS ASSIGNED
   0   C       A VALUE ON ALL(SOME) PATHS
   2         Y=A
   3         IF(A.LT.0)
   4        $X=B
   5         IF(X.LT.Y)
   6        $ RETURN
   7         B=1
   8         RETURN
   1         END
```

** N O E R R O R S **
- - - - - - - -

W A R N I N G S
- - - - - - - -

WARNING NUMBER	DESCRIPTION
** 220 **	LOCAL VARIABLE ----*Y*----, INITIALIZED IN A DATA STATEMENT, IS ASSIGNED A VALUE ON ALL PATHS. UNDEFINITION WILL OCCUR UPON EXIT FROM THIS SUBPROGRAM.
** 221 **	LOCAL VARIABLE ----*X*----, INITIALIZED IN A DATA STATEMENT, IS ASSIGNED A VALUE ON SOME PATHS. UNDEFINITION MAY OCCUR UPON EXIT FROM THIS SUBPROGRAM.
** 229 **	LOCAL VARIABLE ----*Y*---- IS ASSIGNED A VALUE IN BLOCK NO. 1 AND IS EITHER ASSIGNED A VALUE THEREAFTER BEFORE BEING REFERENCED, OR IS NOT SUBSEQUENTLY REFERENCED, ON ALL PATHS. ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS 1 2
** 230 **	LOCAL VARIABLE ----*X*---- IS ASSIGNED A VALUE IN BLOCK NO. 1 AND IS EITHER ASSIGNED A VALUE THEREAFTER BEFORE

BEING REFERENCED, OR IS NOT SUBSEQUENTLY REFERENCED,
ON SOME PATHS.
ONE SUCH PATH, INDICATED BY BLOCK NUMBERS, IS
1 - 4

M E S S A G E S

MESSAGE NUMBER | DESCRIPTION

** 304 ** I/O CLASSIFICATION OF ARGUMENTS AND COMMON VARIABLES
FOR SUBROUTINE ---*SUB*---

ARGUMENTS POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	----*A*----	STRICT	NON
2	----*B*----	INPUT	OUTPUT

$ IN THE CONTINUATION FIELD INDICATES THE EXPANSION
OF THE LOGICAL IF STATEMENT ON THE PREVIOUS LINE

```
BLOCK                    SOURCE

   1         FUNCTION FUN(X)
   0   C
   0   C  FUNCTION IS NEVER REFERENCED
   0   C
   2         RETURN
   1         END
```

E R R O R S

ERROR NUMBER	DESCRIPTION
** 102 **	FUNCTION NAME ---*FUN*--- IS NEVER ASSIGNED A VALUE.

W A R N I N G S

WARNING NUMBER	DESCRIPTION
** 207 **	DUMMY ARGUMENT ----*X*---- IS NEVER USED.
** 235 **	SUBPROGRAM ---*FUN*--- IS NEVER CALLED.

M E S S A G E S

MESSAGE NUMBER	DESCRIPTION

** 304 ** I/O CLASSIFICATION OF ARGUMENTS AND COMMON VARIABLES
FOR FUNCTION ---*FUN*---

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
0	---*FUN*---	NON	NON
1	----*X*----	NON	NON

** 304 ** I/O CLASSIFICATION OF ARGUMENTS AND COMMON VARIABLES
FOR FUNCTION ---*FSIM*--

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
0	---*FSIM*--	NON	STRICT
1	DUMMY PARM.	STRICT	NON

** 304 ** I/O CLASSIFICATION OF ARGUMENTS AND COMMON VARIABLES
FOR SUBROUTINE --*SUBSIM*-

ARGUMENTS

POSITION	NAME	INPUT CLASS	OUTPUT CLASS
1	DUMMY PARM.	STRICT	NON
2	DUMMY PARM.	STRICT	NON
3	DUMMY PARM.	STRICT	NON
4	DUMMY PARM.	STRICT	NON

Name: _____ Phone: _____

Address: _____

_____

General

I. Desired format of unlabeled magnetic tape copy of DAVE:

_____ 800 cpi, 7-track, even parity.

_____ 800 cpi, 9-track, odd parity.

_____ 1600 cpi, 9-track, odd parity.

Number of characters per block (multiple of 80)**

_____.

Tape character code: _____

Special instructions: _____

_____

_____

_____

II. Kind of machine: _____

_____

_____

III. Operating system: _____

_____

_____

IV. Fortran compiler to be used: _____

V. How much core is available and comments on its cost and accessibility (e.g. amount of core that is easily accessible for normal production): _____

_____

_____

_____

_____

_____

VI. Number of files which may be open at one time: _____

VII. Is there a random access capability for mass storage files? _____

Specific Fortran

I. Is there a local Fortran-callable traceback routine and how is it called?

_____

_____

** A blocking factor of 80 characters necessitates the use of a 2400 foot tape and 600 foot tape; a factor of 1600 characters requires only a 600 foot tape. There will an additional charge of $15.00 for card image tapes.

II. Are there Fortran-callable shift and bit manipulation routines (returning integer or typeless results)? Please describe below.

a. SHIFT (A1,A2) bit positions: left circular if A2 is positive; right with sign extension and end off if A2 is negative.

b. AND(A1,A2): bit-by-bit logical AND of A1 and A2.

c. OR(A1,A2): bit-by-bit logical OR of A1 and A2.

d. COMPL(A): bit-by-bit Boolean complement of A

III. How do you use your random access read and write routines?

IV. What is the Fortran end-of-file test?

V. What is the notation used to represent machine constants (e.g. CDC octal constants are followed by "B", as in 77B)?

VI. Is there a local Fortran-callable routine which will abort a job, i.e. prevent execution of subsequent control cards? How is it called?

For CDC installations only

I. Is there a PROGRAM card? _____

_____

II. Can buffer sizes be set on the PROGRAM card? e.g.

PROGRAM MAIN (IN=101,OUT=1001,TAPE1=IN,TAPE2=OUT)

Are the sizes specified in octal or in decimal? _____

_____

_____

What is the minimum allowable size? Is it different for formatted and unformatted files? _____

_____

_____

What is the default? _____

APPENDIX C

Non-ANS FORTRAN Constructs Accepted by DAVE

1. The CDC PROGRAM card (including file buffer specifications).

2. Two branch logical IF statements of the form

   IF (logical expression) L1, L2

   where L1 is the transfer label if the expression is true and L2 is the transfer label if the expression is false.

3. Double precision and complex constants.

4. IMPLICIT type declaration statements of the form

   IMPLICIT $type_1(ac_1,\ldots,ac_i\text{-}ac_j,\ldots ac_n),..,type_2(ac_1,\ldots,ac_n)$

   where $type_i \in$ {LOGICAL,REAL,INTEGER,DOUBLE PRECISION,DOUBLE,COMPLEX}
   and ac_i is a single alphabetic character
   $ac_i\text{-}ac_j$ is a range of characters.

5. FORTRAN II READ and PRINT statements.

6. DATA statements where data is entered into an array by mention of the array name only (e.g. DATA A/1.0,2.0,3.0/).

7. FORMAT statements which have Hollerith strings delimited by asterisks.

8. Multiple assignment statements (e.g. A=B=C+1).