



3 4456 0549284 7



NATIONAL LABORATORY
operated by
UNION CARBIDE CORPORATION
for the
U.S. ATOMIC ENERGY COMMISSION



ORNL-TM- 1176

COPY NO. - /

DATE - June 25, 1965

DIAGNOSE, a Routine to Debug FORTRAN Programs

J. A. Thompson

Abstract

A computer program, DIAGNOSE, to be used in debugging Fortran-63 programs, is available to users of the CDC-1604 at ORNL. DIAGNOSE will detect three common errors which occur during execution of programs:

- 1) erroneous subscripts,
- 2) use of variables which have not had values assigned to them,
- 3) erroneous do-loop parameters.

CENTRAL RESEARCH LIBRARY
DOCUMENT COLLECTION
LIBRARY LOAN COPY
DO NOT TRANSFER TO ANOTHER PERSON
If you wish someone else to see this document, send in name with document and the library will arrange a loan.

NOTICE This document contains information of a preliminary nature and was prepared primarily for internal use at the Oak Ridge National Laboratory. It is subject to revision or correction and therefore does not represent a final report.

LEGAL NOTICE

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

- A. Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or
- B. Assumes any liabilities with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission, or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.

DIAGNOSE, a Routine to Debug FORTRAN Programs

J. A. Thompson

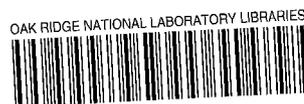
June 25, 1965

The amount of time spent in debugging a program often exceeds that spent in designing and coding. Errors which occur during the execution of the program and which do not immediately halt the execution are especially hard to find. Three of the most common of these errors occur when the programmer uses:

- 1) erroneous subscripts,
- 2) variables which have not had values assigned to them,
- 3) erroneous do-loop parameters.

A program, DIAGNOSE, is designed to detect these errors.

If one of the three types of errors occurs during execution of the program the execution is halted, an error message is written on the standard output unit, and the error return is taken for dumping purposes. The error message consists of statement number identification, variable name, and the type of error. Statement identification is of the form STATEMENT N + n, where "N" is a statement number from the Fortran-63 program and "n" is a relative count forward from this statement. This identification is consistent with Fortran-63 diagnostics except that the first statement of a program, if unnumbered, is considered to be STATEMENT 0 + 1. Further identification is given in the form STATEMENT IS IN NAME where NAME is the name of a program, subroutine, or function.



OAK RIDGE NATIONAL LABORATORY LIBRARIES

3 4456 0549284 7

The following paragraphs explain what conditions are considered as errors and what variable and error information will be written on the standard output unit.

Erroneous Subscripts

Before any operation is performed with a subscripted variable (except input/output operations) the dimensions and subscripts are checked to see if they are undefined, less than or equal to zero, or greater than 32,767. Any one of these conditions is considered an error and one of the following messages is written.

- 1) SUBSCRIPT NO. n IS UNDEFINED ON NAME
- 2) DIMENSION NO. n IS UNDEFINED ON NAME
- 3) SUBSCRIPT NO. n IS .LE. ZERO ON NAME
- 4) DIMENSION NO. n IS .LE. ZERO ON NAME
- 5) SUBSCRIPT NO. n IS TOO LARGE ON NAME
- 6) DIMENSION NO. n IS TOO LARGE ON NAME

where NAME is the name of the subscripted variable and "n" is the position of the subscript.

Examples:

```
1) PROGRAM XAMPLE1
   DIMENSION X(50), Y(50)
   DO 1 I = 1,50
     J = I - 1
     X(J) = 0.
1  Y(J) = 0.
   END
```

The execution of the above program under DIAGNOSE would cause the following message to be written on the standard output unit:

```
STATEMENT 0 + 5, SUBSCRIPT NO. 1 is .LE. ZERO ON X  
STATEMENT IS IN XAMPLE1
```

```
2) PROGRAM XAMPLE2  
COMMON X(50),Y(50),I  
I = 10  
CALL SUB1(X,N)  
END  
SUBROUTINE SUB1(A,N)  
COMMON X(50),Y(50),I  
DIMENSION A(N)  
A(I) = 3.14  
END
```

The above program would result in the following message:

```
STATEMENT 0 + 3, DIMENSION NO. 1 IS UNDEFINED ON A  
STATEMENT IS IN SUB1
```

After the dimensions and subscripts are individually checked the computed subscript is checked against the product of the dimensions.

If an error exists the following message is written:

```
SUBSCRIPT ON NAME EXCEEDS DIMENSIONS
```

Example:

```
PROGRAM XAMPLE3  
DIMENSION X(10,2,4)  
DO 1 I = 1,100  
1 X(I) = 0  
END
```

The following message would be written:

STATEMENT 1 + 0, SUBSCRIPT ON X EXCEEDS DIMENSIONS

STATEMENT IS IN XAMPLE3

Trying to Use Variables Which Have Not Had Values Assigned to Them

DIAGNOSE checks the value of any variable which is involved in an arithmetic, masking, or logical operation not within a subscript. The values of replacement variables (variables on the left hand side of a replacement symbol), variables which are actual parameters of call statements, and variables which stand alone on the right hand side of a replacement symbol are not checked.

Examples

Let FUNCTN be a Fortran function; let SUBRTINE be a Fortran subroutine; let X,Y, and Z be in dimension statements; let A,B,C,I,J,M, and N be simple variables:

1) $A = B * C - \text{FUNCTN}(M, N)$

The values of B and C are checked.

2) $X(I, J) = A * B + \text{FUNCTN}(M, I * J / N)$

The subscript of X is checked.

The values of A, B, I, J, and N are checked.

3) $X(I, A + \text{SINF}(Y(I, J))) = A$

The subscript of X is checked.

4) $B = Y(I, J) - A * \text{SINF}(X(I, J))$

The subscripts of Y and X are checked.

The values of $Y(I, J)$ and A are checked.

5) IF(N-M) 1,2,3 (1,2 and 3 are statement numbers)

The values of N and M are checked

6) CALL SUBRTINE(X(I,J),A,B,Y(I,J)-1.)

The Subscripts of X and Y are checked.

The value of Y(I,J) is checked.

If it is determined that the variable being checked has not had a value assigned to it the following message is written:

NAME IS UNDEFINED

Example:

```
PROGRAM XAMPLE4
READ 1,A,B
1 FORMAT(3F10.7)
ROOT1 = (-B + SQRTF(B*B-4.*A*C))/(2.*A)
END
```

The execution of this program would cause the following message to be printed:

STATEMENT 1 + 1,C IS UNDEFINED

STATEMENT IS IN XAMPLE4

Erroneous Do-loop Parameters

Before the execution of any do-loop each nonconstant parameter is checked to see if it is undefined, negative, or greater than 32,767.

If such is the case one of the following messages will be written:

- 1) NAME IS UNDEFINED
- 2) NAME IS NEGATIVE
- 3) NAME IS TOO LARGE

Example:

```
PROGRAM XAMPLE5  
DIMENSION X(50),Y(50)  
DO 1 I = 1,N  
X(I) = 0.  
1 Y(I) = 0.
```

The following message would be written:

```
STATEMENT 0 + 3, N IS UNDEFINED  
STATEMENT IS IN XAMPLE5
```

Operation of DIAGNOSE

The input for DIAGNOSE consists of Fortran-63 source decks, binary decks, and data. Into each source deck DIAGNOSE inserts calls to certain library subroutines thus producing a new Fortran-63 program. These call statements will not in any way affect the logical flow of the program and the results up to the point at which the error occurs will remain the same.

After the new programs are produced they are compiled and loaded into memory along with the binary decks included as part of the input. Control is then turned over to these programs to operate on any data included in the input.

DIAGNOSE makes two passes at each source program. The first pass compiles four lists and outputs part of the original program along with other information on a scratch tape. The four lists are:

- 1) arrays and their dimensions (VARLIST)
- 2) all statement numbers (IDLIST)

- 3) terminal statement number of do loops (DOLIST)
- 4) statement numbers of replacement and call statements (REPLIST)

The second pass analyzes do statements, replacement statements, call statements, and if statements. For each statement of these types DIAGNOSE may insert call statements to two or more of the following library subroutines: QQQBUG1, QQQBUG2, QQQBUG3, and QQQBUG4. QQQBUG1 handles identification of the statement being checked, QQQBUG2 checks subscripts, QQQBUG3 checks to see if a value has been assigned to a particular variable, and QQQBUG4 checks the variable parameters of a do-loop. The second pass also adjusts statement numbers so that the flow of the user's program remains the same.

A more detailed presentation of the workings of DIAGNOSE may be found in the flow charts following the text. Listings of the complete program are available from the author. The program decks are available from the program librarian, R. B. Bullock.

Submitting a Job to DIAGNOSE

DIAGNOSE may be used in conjunction with the GLORIOUS debugging feature (see ORCID Memo No. 26). When an error is detected the current values of the program variables are dumped.

Caution: If the GLORIOUS debugging feature is called for, tape number 57 is not available to the user.

The following is the necessary ordering of decks submitted to be run under DIAGNOSE:

1) ΔCOOP Card. The debugging package uses logical unit numbers 48 and 49 as scratch tape during the first part of its operation. The user will have access to these tapes, but only as scratch or output tapes. Forty-eight and forty-nine must be defined as scratch tapes on the ΔCOOP card. The G dump option should be selected. Other tapes and other dump options may be selected as the user desires.

2) DIAGNOSE

3) Fortran-63 Source Decks. Not all subroutines, etc., of a program have to be run under the debugging package. Only those to be checked by DIAGNOSE should be submitted as Fortran-63 source decks.

4) FINIS Card. A card with the word FINIS starting in column 10 must appear following the END card of the last Fortran-63 source deck.

5) Binary Decks. These include subroutines, etc., which the user does not want checked by DIAGNOSE. They must include QQQBUG1.

6) ΔCard

7) Data

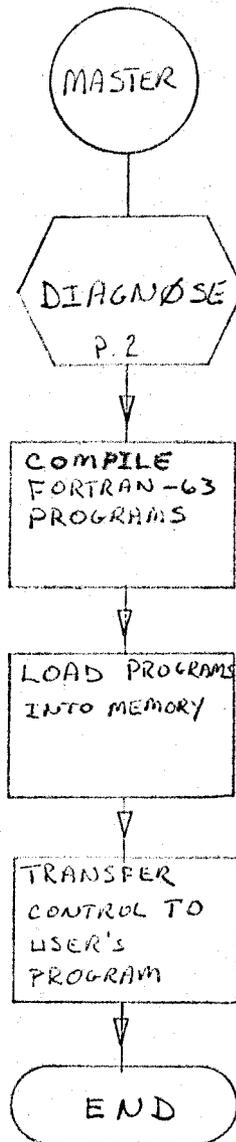
The above ordering is the same as for a regular compilation-execution except that no ΔFTN and ΔEXECUTE card is included and a FINIS card must be included.

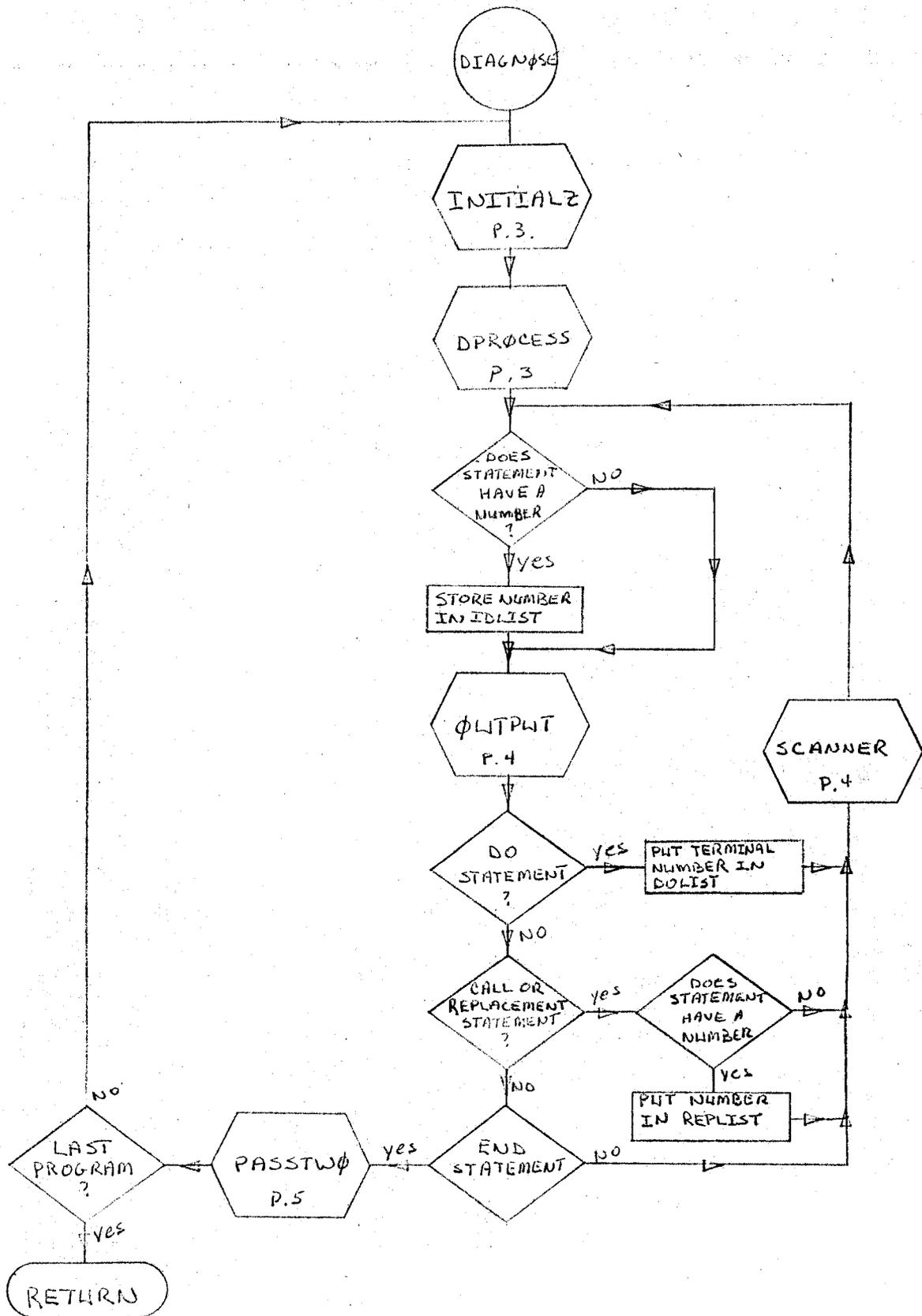
Further Notes on Diagnose

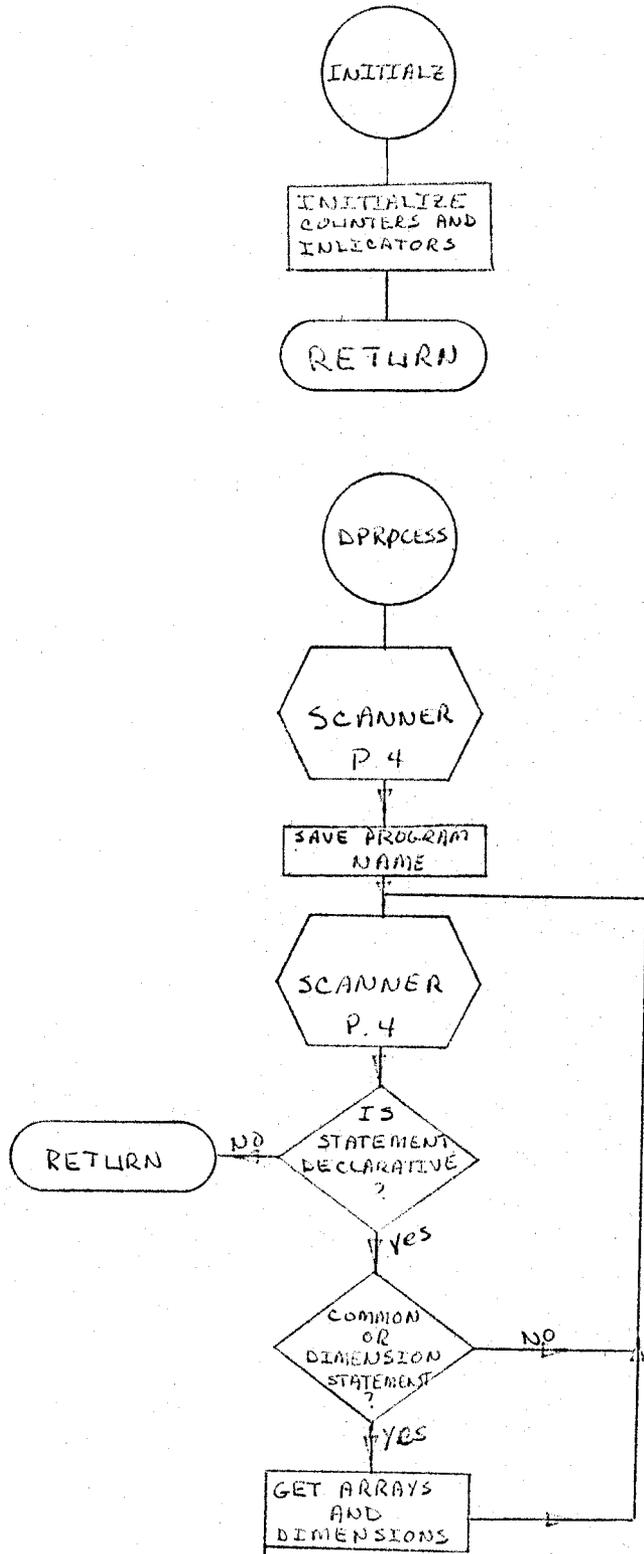
The intermediate programs produced by DIAGNOSE will require approximately twenty per cent more memory than the original program.

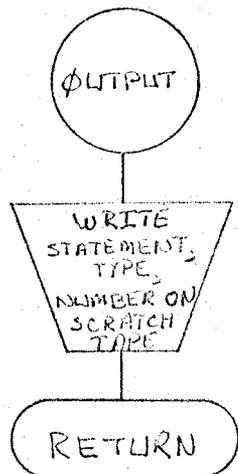
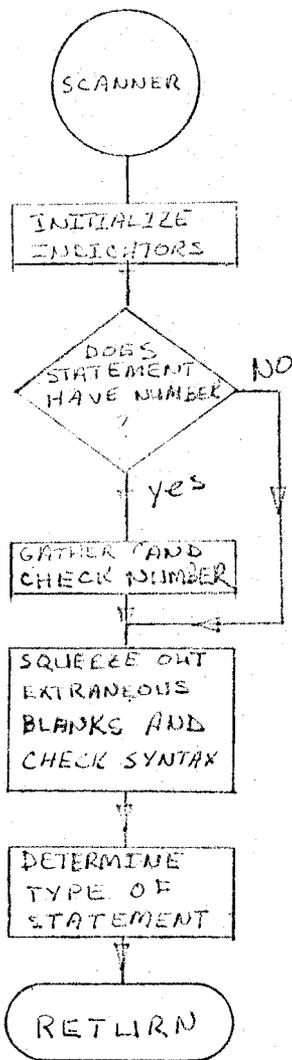
The running times will be on the order of twice as long as that of the original program. For these reasons the debugging package should not be used until trouble actually develops. Further DIAGNOSE expects a compilable Fortran-63 program so that all syntax errors must be corrected before submitting the problem.

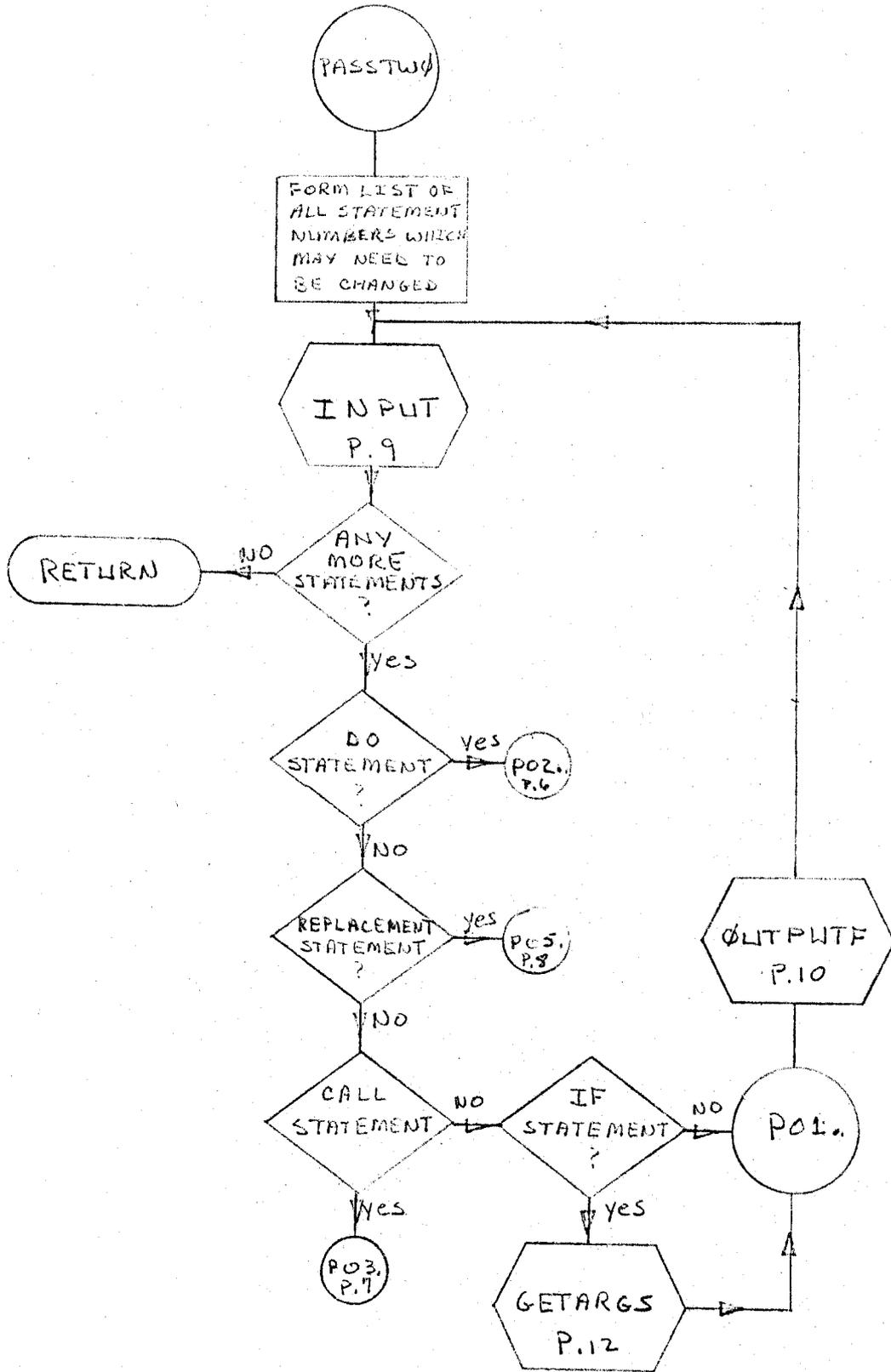
If the user can make an educated guess as to which specific subroutine or group of subroutines is causing the problem he should submit this group as Fortran-63 source decks, and the rest of the program as binary decks. This will cut down on the additional storage and running time needed by DIAGNOSE.

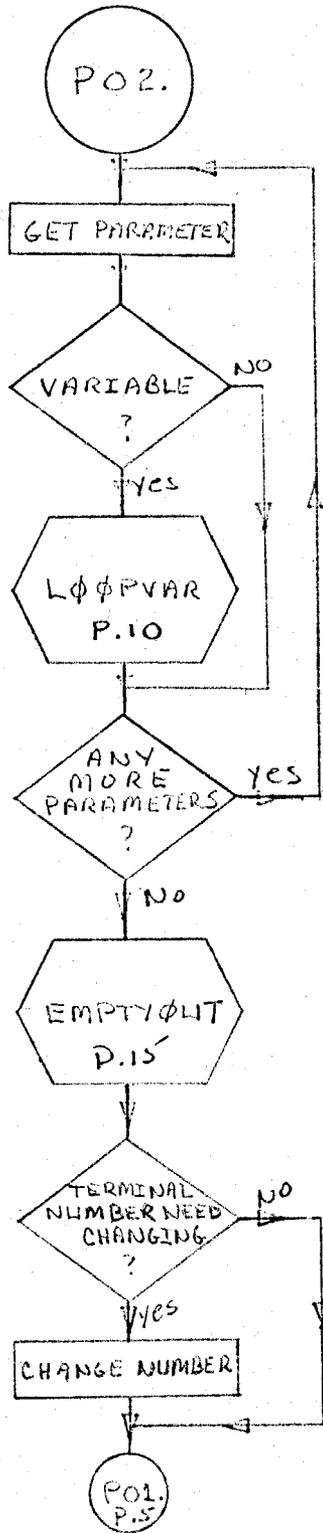


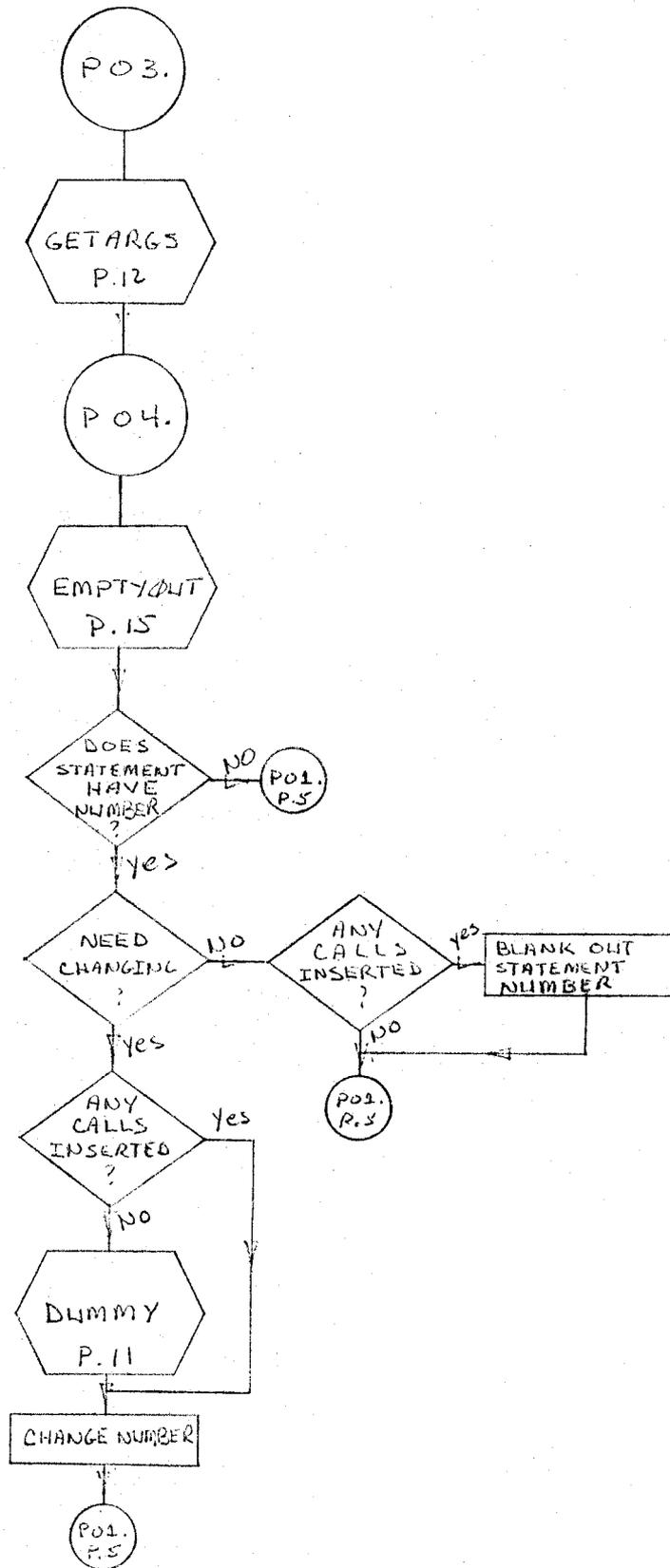


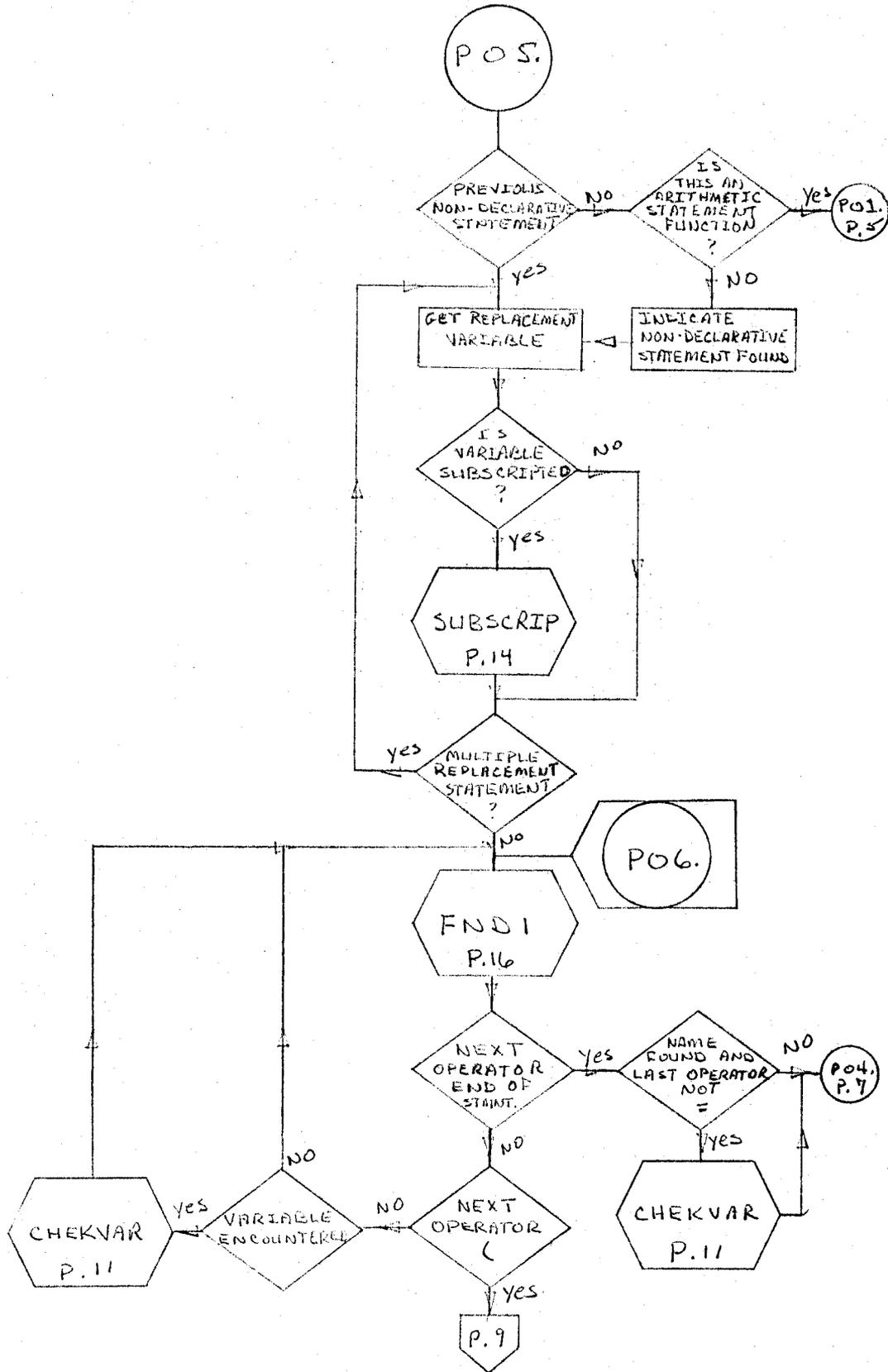


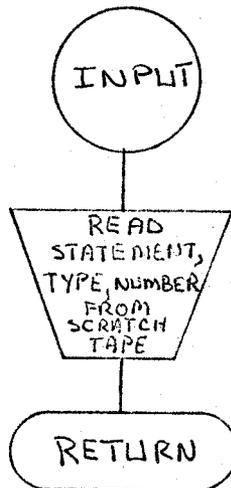
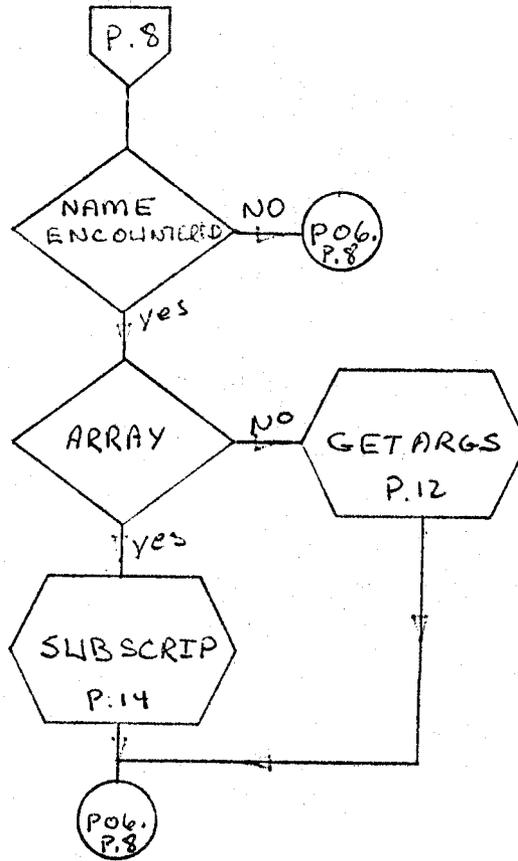


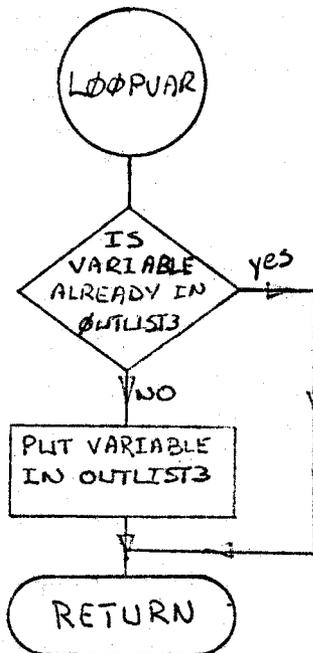
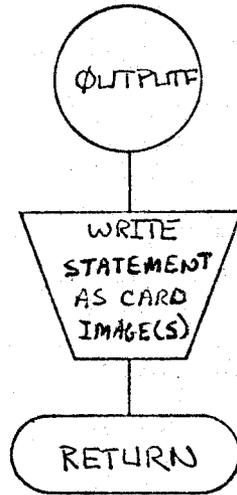


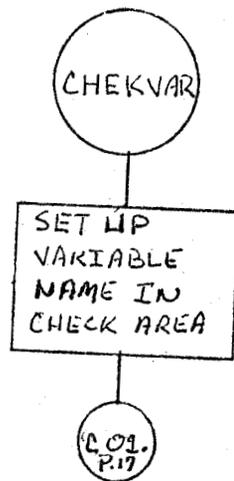
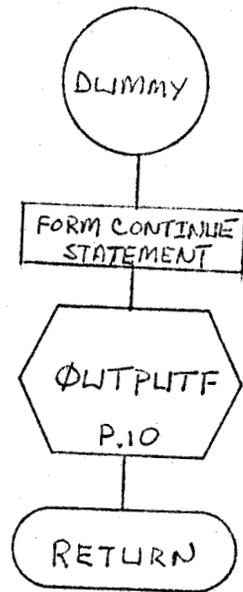


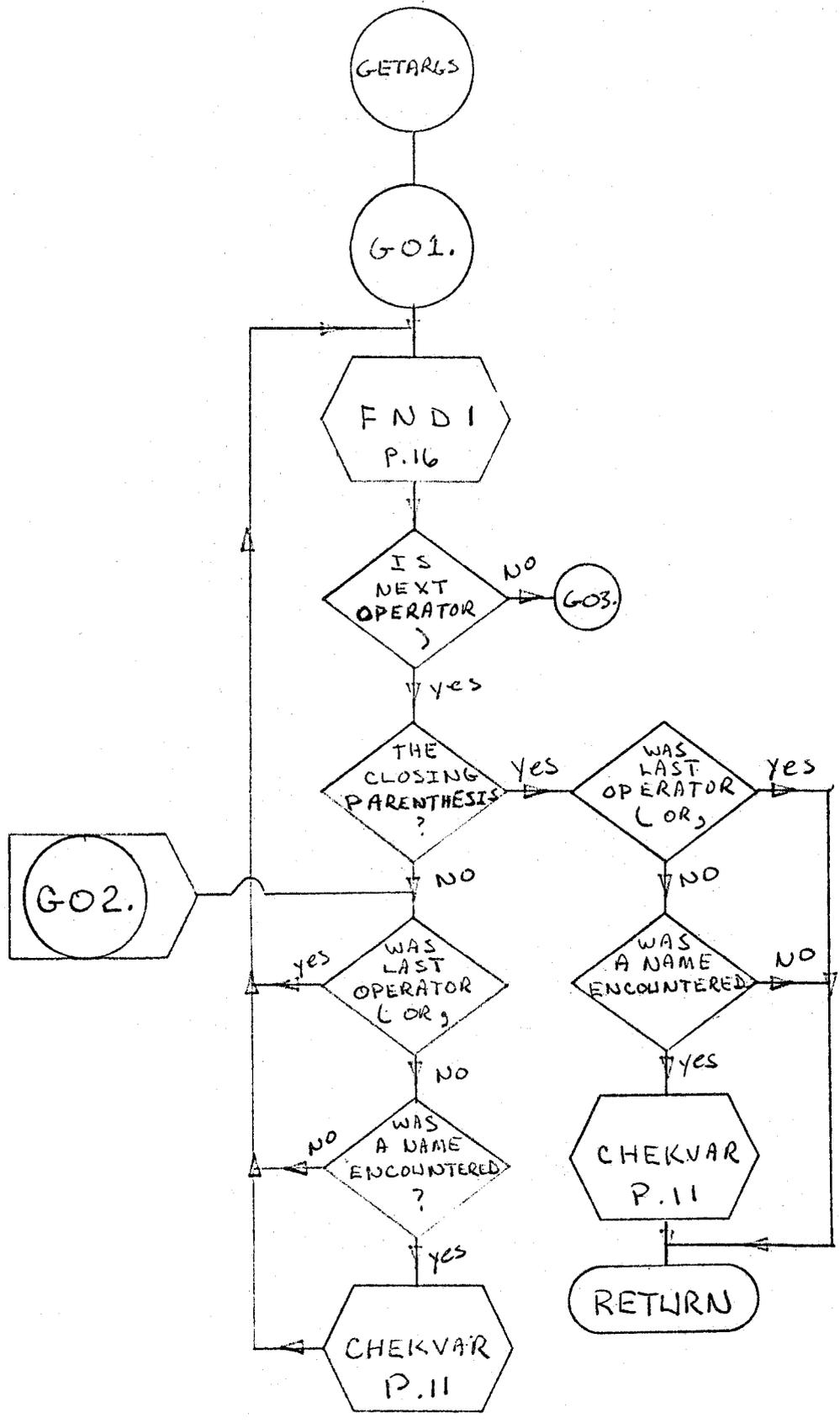


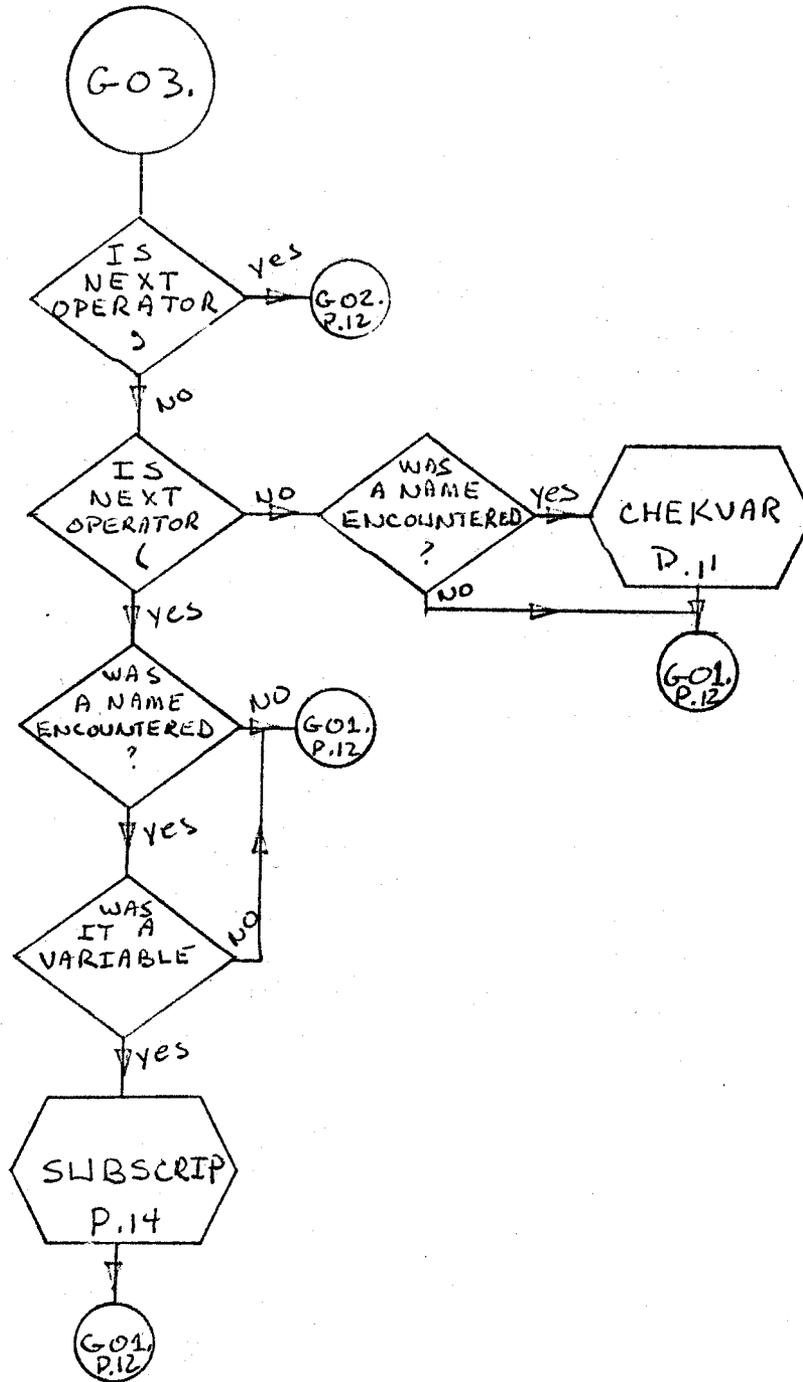


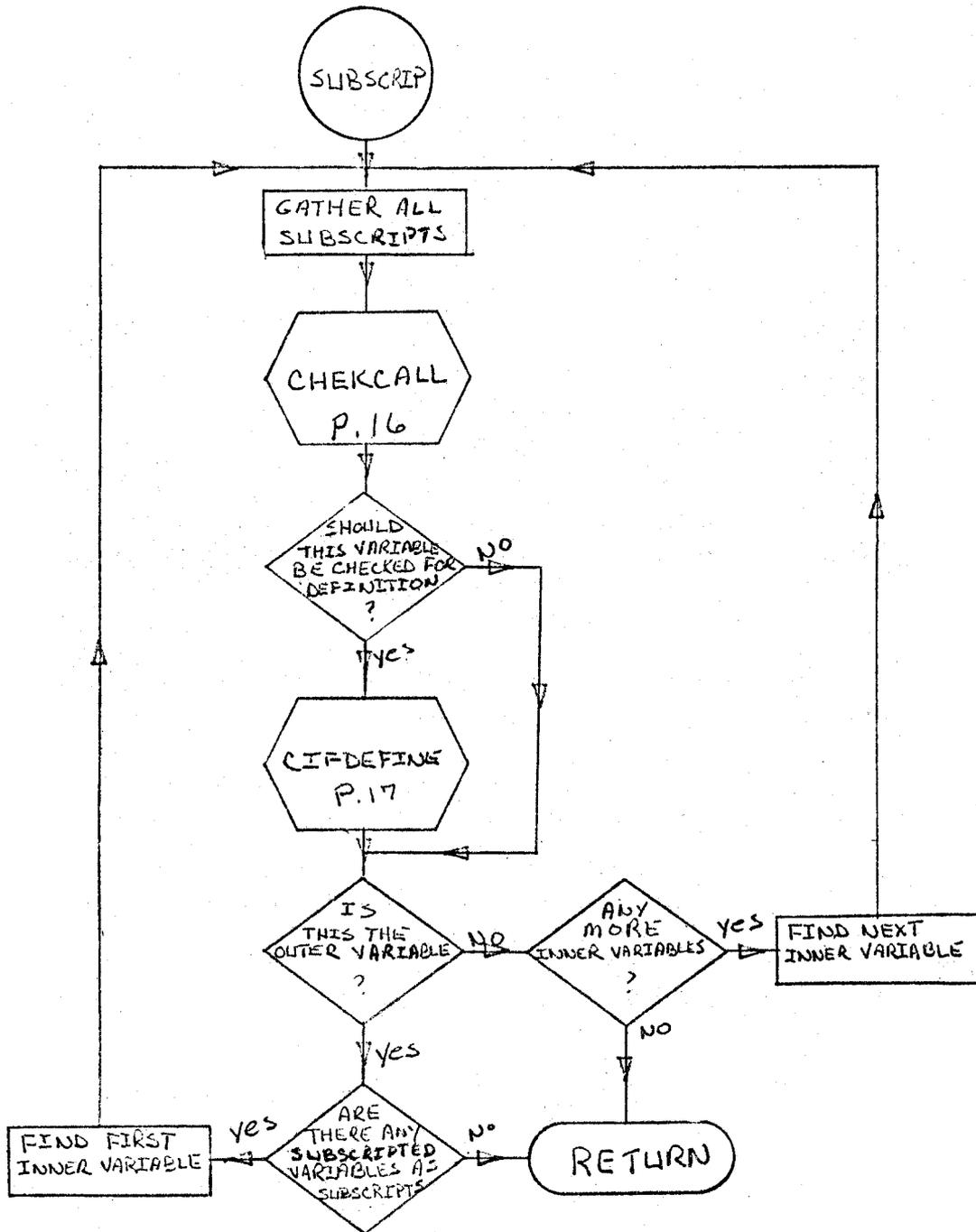


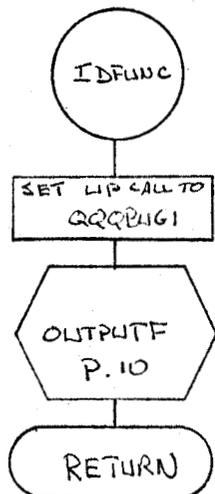
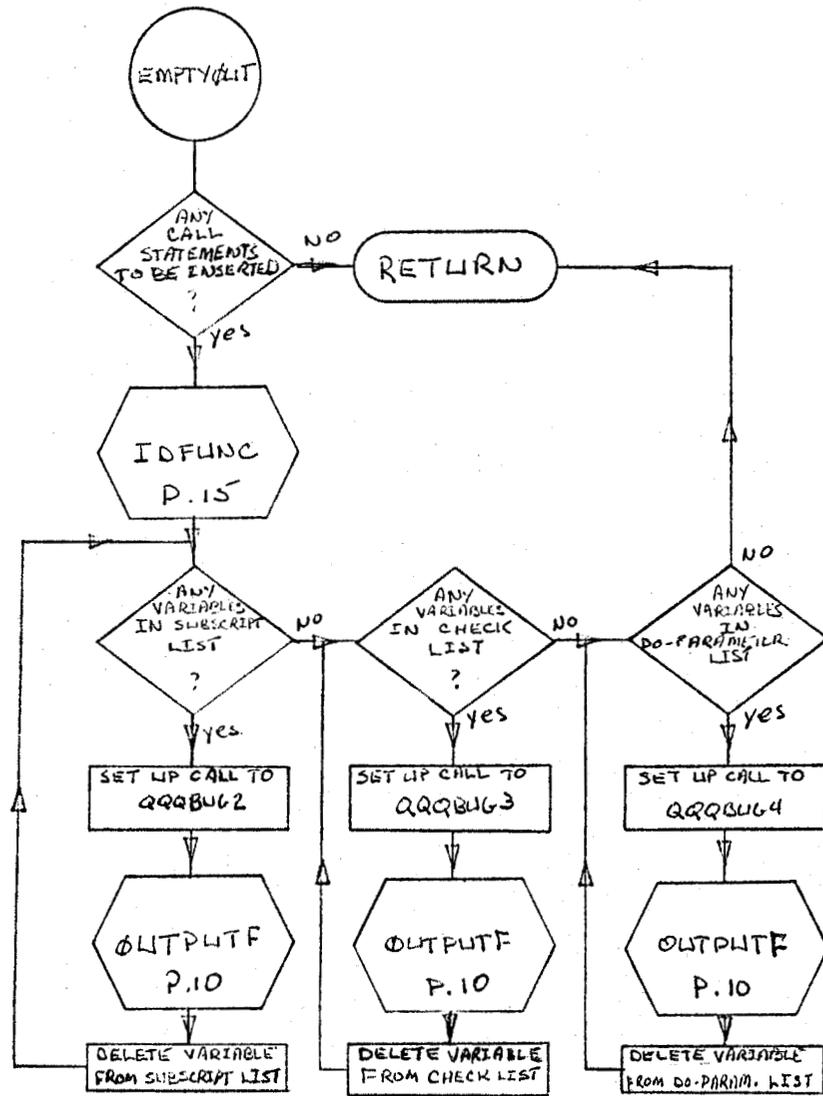


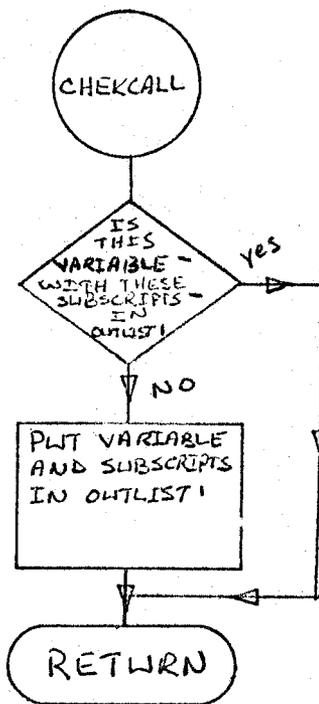


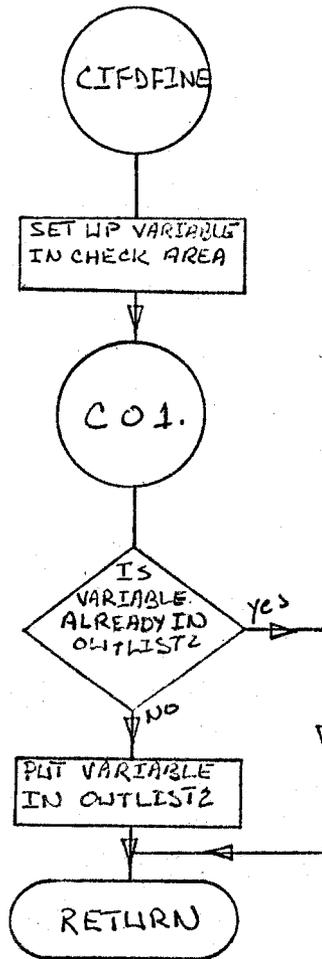












Distribution

- 1-2. Central Research Library
- 3. Document Reference Section
- 4-8. Laboratory Records
- 9. Laboratory Records - Record Copy
- 10-24. Division of Technical Information Extension
- 25-125. Mathematics Division
- 126. Research and Development, ORO
- 127. ORNL Patent Office

