

AN APPROACH TO DEBUGGING

by

613-8301

CHING-NEU HONG

B. A., Fu-Jen Catholic University, 1969

A MASTER'S REPORT

submitted in partial fulfillment of the
requirements for the degree

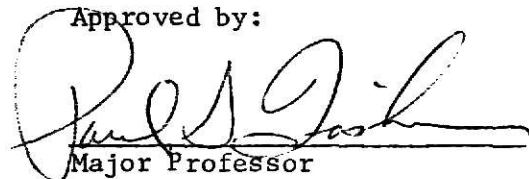
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1973

Approved by:



Major Professor

LD
2668
R4
1973
A65
2.2
Docu-
ments

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION	1
What is debugging	1
Aids to debugging	2
Objective of the study	6
II. PROGRAM	7
Description of the PL/1 program	14
Limitation and further suggestions	16
III. SUMMARY	22
IV. ACKNOWLEDGMENTS	23
V. BIBLIOGRAPHY	24
VI. APPENDICES	25
Appendix A. REPORT program variable list	25
Appendix B. PL/I program	29
Appendix C. Original FORTRAN program	36
Appendix D. Output of the PL/1 program	38
Appendix E. Output of the modified WATFIV program	40
Appendix F. Output of the PL/1 program (different data)	43

LIST OF FIGURES

Figure	Page
I. Flowchart for PL/1 program	19
II. Flowchart for subroutine program	21

CHAPTER I

INTRODUCTION

I. WHAT IS DEBUGGING?

Debugging is the technique of detecting, diagnosing, and correcting errors which may occur in programs or systems.¹ Debugging may also be applied to the process of testing the performance of hardware systems and to the testing of a complete data processing system, but for this paper it relates to detecting, diagnosing, and correcting errors which may occur in programs.

"Syntax errors" and "Logic errors" are two different types of program errors. Syntax errors are errors due to incorrect coding, and logic errors are those due to incorrect appreciation of the problem.

A program written in a symbolic language requires a compilation, or a translation, into the machine language so that it can be understood by the processor. During the process of compilation, syntax errors involving incorrect handling of the symbolic language are detected. Most compilers reject incorrect statements and print out some indication of the type of errors. Normally programs containing compilation errors cannot be run, and the errors must be corrected before the program can be tested. Usually, syntax errors are comparatively easy to find. Here is an example of a syntax error of a DO statement in PL/I. Suppose we

¹Chandor, A., Granham, J. and Williamson, R. A Dictionary of Computers, Australis: Penguin Books, 1970.

write "DO 100 I=1,10". This is a FORTRAN statement, not a PL/I statement. It should be changed to "DO I=1 TO 10;" in the PL/I statement.

An example of a logic error would be to calculate the average score of students. The result is obtained by adding the scores of the students taking the exam and dividing by the number of students taking the exam, not by having the combined total score divided by the number of students in the class. The calculation would be performed correctly by the program, but the result would not be that which was desired. If the program fails to achieve the expected results, or if an unexpected halt occurs within the program, a logic error diagnosis is required. Logic errors can only be detected by testing the program with sample data. The system which will be described in this paper illustrates techniques which could be used in a debugging system to aid the programmer in finding these types of errors.

Frequently, a large part of debugging involves determining what the error is and where it occurred. After finding the error, a correction can be checked by hand or by submitting the corrected program to be run again. In this way, it is possible to determine the essential information which possibly was obscured by the error determination type, or simply by lack of visible information.

II. AIDS TO DEBUGGING

We can say debugging is possibly the most important aspect of programming,² as well as the most costly portion of writing a program in

²IBM Corporation, IBM System 360 Operation System Programmer's Guide to Debugging, IBM Form GC28-6670-3.

terms of both human and machine effort.

When an error occurs in a program, frequently even a very experienced programmer can not identify the cause of the error without considerable effort.

Jacob T. Schwartz in "An Overview of Bugs"³ discusses debugging and the debugging tools. These tools trace the program at different levels such as the program transfer level, or the subroutine level. Also it is necessary to use different kinds of traps, different kinds of dumps and different kinds of program-termination summaries, such as the number of statements executed in a program run because of the various types of errors which occur in the program. The late-stage debugging tools include a certain number of ideas in the area of systematic program testing. He also mentions a type of debugging aid suggested by Flod's correctness-proof methods.⁴ This idea is as follows: one can implement a system by which a programmer, during the process of program elaboration, could formally state those assumptions (concerning the moment-to-moment state of his data) which led him to write his program as he did. However, this method is difficult at best for most programmers.

R. M. Balzer in "EXDAMS-Extendable Debugging and Monitoring System"⁵ point out that the ability to debug programs has not progressed as much as the increased use of the higher-level languages has. EXDAMS provides

³Schwartz, Jacob T. "An Overview of Bugs," Debugging Techniques in Large Systems, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1971.

⁴ibid. p 10.

⁵Balzer, R. M. "EXDAMS-Extendable Debugging and Monitoring System," AFIPS Conference Proceedings Spring Joint Computer Conference, 1969, pp. 567-580.

a single environment in which the users can easily add new on-line debugging aids to the system, one at a time, without further modifying the source level compilers, EXDAMS, or other relevant programs. EXDAMS was designed to provide two capabilities:

- 1) An extendable facility to which new debugging and monitoring aids could be easily added, then tested; and
- 2) A system providing some measure of independence of the machine.

This means independence not only of the particular machine on which it is being run and the particular implementation of the language being debugged and/or monitored, but also of several source languages in which users' programs could be written and debugged and/or monitored.

EXDAMS contains two types of debugging and monitoring aids, "static" and "motion-picture." The static aids display information that is extant at a given point in execution time, such as the value of variables at the time an error occurred. The motion-picture aids are execution-time sensitive, that is, they display data as it changes during execution time.

EXDAMS is a four-phase system the four phases are program analysis compilation, run-time history-gathering, and debug-time history-playback. The program analysis produces a symbol table and a random-access file of a model of the user's source program for the history-playback. As it analyses the program and builds a model of the program, it also inserts debugging statements into the program to implement the history-gathering. Each model entry includes an indicator of the type of model entry, a pointer to the associated source statement, and an index to an entry in either the model or the symbol table. The compilation phase compiles

the source program, as updated during program analysis. The compiled version of the updated program is run with a set of run-time routines. These routines gather dynamic information about the program's behavior. This information is collected in a buffer that is written out when complete. It is the history tape of the programs's behavior which, together with the symbol table and model, is sufficient to recreate the program's behavior in either the forward or backward direction of execution time. The debug-time history-playback phase contains the debugging and monitoring aids which present the history information to the user in a usable form on his display screen. It also interprets the user's commands for alternative displays and/or execution-time variations. It provides an editing capability for modifying discovered bugs and for returning this modified program to the system for another debugging run.

Another debugging technique, called the QUIKTRAN System,⁶ allows the program undergoing debugging to be changed by the insertion and deletion of statements. A form of unconditional breakpoint capability is included in this technique so that a statement can be inserted at any point in the program. When the point is reached, this has the effect of transferring control to the user. For example, in a FORTRAN program, every time when an "assignment" statement is encountered, a print statement is inserted. This print statement indicates the value of the variable on the left-hand side of the assignment. This debugging technique modifies the FORTRAN program in a semi-automatic fashion to provide the necessary information.

⁶Evans, Thomas G. and Darley, D. Lucille "On-line Debugging Techniques: A Survey," AFIPS Conference Proceedings Fall Joint Computer Conference, 1966, p. 43.

Unfortunately, the usual way to debug a program is to check it step by step by hand to discover the location of the logic error. The next step is to modify it and then submit it again. This process continues until the correct output is obtained. The primary reason for a large number of iterations in this process is the lack of definite information available to the programmer. This paper is an attempt to provide further suggestions for more usable techniques in debugging systems.

III. OBJECTIVE OF THE STUDY

In order to provide information to the programmer concerning the state of the program at the time execution terminated, a PL/I program has been written to aid in providing this information.

This paper describes this program and its use as a debugging tool to discover logic errors which occur in FORTRAN programs. In other words, a FORTRAN program will be the input data to the program described herein. The PL/I program will process the input program statement by statement and a new FORTRAN program will be punched out. This program can then be run through WATFIV. This second program will consist of the original FORTRAN program and the inserted statements which monitor the calculated intermediate results. The program shown here is not to be considered as a finished tool, due to the assumptions made concerning the input (FORTRAN) program. However, the program does indicate a direction of our thought with respect to the type of information a debugging system should provide to a typical user in this a typical environment.

CHAPTER II

PROGRAM

To illustrate the technique of debugging as performed by this program, an example is given in Appendix B. The language used in the processing program is PL/I and the input data is a program written in a sufficiently restricted version of FORTRAN IV. The program is designed as stated for helping programmers find the logic errors in FORTRAN programs. The PL/I program processes the input FORTRAN program, statement by statement. After reading a FORTRAN statement, the PL/I program examines the type of FORTRAN statement read and identifies it as (1) ASSIGNMENT, (2) READ or WRITE, (3) DO, (4) IF, (5) GO TO, (6) INTEGER or REAL. Depending on the type of statement it is, then immediately after each statement it may be necessary to insert FORTRAN "WRITE" statements into the original FORTRAN program. These inserted "WRITE" statements are to print out the intermediate results of the FORTRAN program. The modified FORTRAN program, with its inserted "WRITE" statements is printed and then punched. The output of the new program consists of that of the original FORTRAN statements plus the calculated intermediate results. These results can be checked to see whether the correct answer is obtained at various stages in the program. Since each result is printed out immediately after the calculation is done, the error may be directly traced to the FORTRAN statement which performs the calculation. The effects of the debugging system on the user program statements are as follows:

1. If it is an "ASSIGNMENT" statement, then insert a print and a format statement to output the value of the variables on the left-hand side of the assignments in order to check whether the correct answer or computation is obtained. For example, if statement 5 is

$$Y = (A**3 + B*C - 2.*B/A)*C - B$$
then immediately after this statement the following two statements are inserted

```
PRINT 33333,Y
```

```
33333 FORMAT(1H , 10X,'5Y',F10.2)
```

2. If it is a "READ" or a "WRITE" statement, do nothing.
3. If it is a "DO" statement, check whether the program has specified the EXPAND or NO EXPAND option. If the EXPAND option is in force then continue to check types of all the statements within the DO LOOP. Conversely if NO EXPAND option is in effect then continue to read the input data until a "CONTINUE" statement is found, not checking types. The NO EXPAND option is an alternative to providing all the information associated with the execution of a loop. We feel that for debugging a typical loop the NO EXPAND option, which would normally be default, is sufficient. As the loop executes keep all the information concerning the DO LOOP such as loop range, loop parameter, initial loop value, terminal loop value, the first value set and the last value set for all variables within the loop. This information is retrieved after the DO LOOP is executed the first time, and after the DO LOOP has completed. In addition, search for any "GO TO" statement in the DO LOOP. If there is any "GO TO" statement, check whether there is an exit point to go out of the loop. The history

gathering information is inserted within the DO LOOP and then printed and punched out.

In order to specify the EXPAND and NO EXPAND option the programmer inserts a "NO EXPAND" statement or an "EXPAND" statement or both of them. These two statements are punched from the 7th column into a card, and inserted into the FORTRAN program before the "DO" statement, this insertion has to be done before the FORTRAN program is processed by the debugging program. After the type of statement read-in by the PL/I program is checked, if it is a "NO EXPAND" statement then it will not expand the immediately following DO statement i. e. it will not insert "WRITE" statements within the DO LOOP. Similarly if it is an "EXPAND" statement, the normal write statement insertion is done in the immediately following DO LOOP until either the end of the program is found or a "NO EXPAND" statement is encountered.

To differentiate these two conditions in the PL/I processing program, the variable NE is set to 1 for the "NO EXPAND" statement and otherwise. Whenever a DO statement is met, a check on the value of NE determines the condition of expansion. For example, if the statement "DO 10 I=1,10" is read in, then check NE; if "NO EXPAND" prevails, then continue to read the data until the "CONTINUE" statement is found without checking statement types. All the loop control information previously specified such as loop range, loop parameter, initial loop value, terminal loop value, the first value set and the last value set for all variables within the loop are kept as used, except for checking whether there are exit points within the loop.

Considering the following example of a FORTRAN program segment.

```

      DO 10 I=1,10

      II=I

      NUM3=NUM1+NUM2

      IF (I.EQ.10) GO TO 2

      GO TO 3

2  PRINT,II

      IF (.TRUE.) GO TO 1

3  NUM1=NUM2

      NUM2=NUM3

10  CONTINUE

      GO TO 1

      STOP

      END

```

If a "NO EXPAND" statement is found, the output will be as follows:

```

      DO 10 I=1,10

      II=I

      NUM3=NUM1+NUM2

      IF (I.EQ.10) GO TO 2

      GO TO 3

2  PRINT,II

      IF (.TRUE.) GO TO 55555

3  NUM1=NUM2

      NUM2=NUM3

      IF (I.EQ.1) PRINT 77777,II,NUM3,NUM1,NUM2

10  CONTINUE

      IF (I.EQ.10) PRINT 88888,II,NUM3,NUM1,NUM2

```

```

77777 FORMAT(1H ,10X,'LOOP RANGE=10 TO 19'/11X,'LOOP PARAMETER=I'/11X,
1'INITIAL LOOP VALUE=1',5X,'TERMINAL LOOP VALUE=10'/11X,
2'FIRST VALUE SET II,NUM3,NUM1,NUM2',4(I10))
88888 FORMAT(1H ,10X,'LAST VALUE SET II,NUM3,NUM1,NUM2',4(I10))

      GO TO 1

55555 PRINT 66666

66666 FORMAT(1H ,10X,'EXIT POINT=16 TR 1)

      GO TO 1

      STOP

      END

```

If before the "DO" statement it had an "EXPAND" statement instead of "NO EXPAND" statement, then the normal write statement insertion is done, and the above example would be as follows (shown in Appendix F):

```

      DO 10 I=1,10

      II=I

      PRINT 33336,II

33336 FORMAT(1H ,10X,'11 II',I10)

      NUM3=NUM1+NUM2

      PRINT 33337,NUM3

33337 FORMAT(1H ,10X,'12 NUM3',I10)

      IF (I.EQ.10) GO TO 2

      PRINT 11112

11112 FORMAT(1H ,10X,'13 NT')

      PRINT 22222

22222 FORMAT(1H ,10X,'14 TR 3')

      GO TO 3

2 PRINT,II

```

```

        IF (.TRUE.) GO TO 1
        PRINT 11113
11113 FORMAT(1H ,10X, '16 NT')
        3 NUM1=NUM2
        PRINT 33338,NUM1
33338 FORMAT(1H ,10X,'17 NUM1',I10)
        NUM2=NUM3
        PRINT 33339,NUM2
33339 FORMAT(1H ,10X,'18 NUM2',I10)
        10 CONTINUE

```

4. If an "IF" statement is read in, then insert a print statement and a format of this print statement. This print statement is to specify if the "IF" statement condition is not met, then no transfer is made. The print statement prints out "NT", which stands for no transfer. For example, consider the statement

```
IF (A.EQ.0.0) GO TO 5
```

then immediately after this statement, insert the following two statements:

```

PRINT 11111
11111 FORMAT(1H ,10X,'4 NT')

```

5. If a "GO TO" statement is read, then before this "GO TO" statement a print statement will be inserted which will print out "TR", meaning transfer. This specifies that the program transferred to the given statement. For example, consider the statement as

```
GO TO 1
```

then before this statement, the following statements will be inserted

PRINT 22222

22222 FORMAT (1H ,10X,'21 TR 1')

6. If an "INTEGER" or a "REAL" statement is read, this will specify identification of the FORTRAN variables, use a subroutine "SUB" program to separate each identification and store it. This is to define the format of "ASSIGNMENT" statement's identification. For example, an "INTEGER AX,BY,CZ,HD" statement is read in, then call the subroutine "SUB". This subroutine will separate AX,BY,CZ,HD and store them individually, then in this input FORTRAN program AX, BY,CZ and HD are to stand for integer number, so before the format of the "ASSIGNMENT" statement's identification is to be defined, the identification should be checked, if it happened to be AX,BY,CZ or HD, then define it's format to I10, otherwise F10.2.

Ideally when all the information is assembled, the logic errors can be traced to the location where they occurred. When the error is located, it can be corrected, and thus the aim of this program is achieved.

In the PL/I program output as shown in Appendix D, the "NUMBER GIVEN" indicated the FORTRAN program's original statement number. The "ASSIGNED" is the read-in statement count but not including the inserted statements.

The flow chart for the PL/I program (shown in Appendix B) is given in Figure 1 and Figure 2 is the flow chart of subroutine program.

The original FORTRAN program is shown in Appendix C, the output from the PL/I program is presented in Appendix D, while the output of the modified WATFIV program is shown in Appendix E.

DESCRIPTION OF THE PL/I PROGRAM

The PL/I is shown in Appendix A and the flowchart is given in Figure 1. The flow of the PL/I program is outlined in the following steps.

- P1. Declare variable used.
- P2. Initialize some values.
- P3. Update data.
- P4. Test if it is a "READ" statement. If yes, go to P13. If not, go to P5.
- P5. Test if it is a "NO EXPAND" statement. If yes, set NE equal to 1, then go to P14. If not, go to P6.
- P6. Test if it is a "EXPAND" statement. If yes, set NE equal to 0, then go to P14. If not, go to P7.
- P7. Test if it is a "DO" statement. If yes, go to D1. If not, go to P8.
- D1. Check whether the program has speckfied the EXPAND or NO EXPAND option. If NE equal to 1, then go to P14. If NE equal to 0, then go to D2.
- D2. Find loop initial range, loop parameter, initial loop value, terminal loop value.
- D3. Keep all the loop control information which are found in D2.
- D4. Store the statement.
- D5. Read in data.

- D6. Test if it is a "CONTINUE" statement. If yes, go to
- D7. If not, find the index of equal sign, then go to D4.
- D7. Test if there is a "GO TO" statement to exit from a DO loop. If yes, go to D8. If not, go to D9.
- D8. Find the exit point.
- D9. Find the loop terminal range.
- D10. Insert a "IF" statement, this is to test FORTRAN program to see if it is the first iteration for this DO LOOP or not. This is to print out the first value set for all variables within the loop.
- D11. Print out and punch out the "CONTINUE" statement.
- D12. Insert a "IF" statement, this is for test FORTRAN program whether it is the last iteration for this DO LOOP or not, and this is for print out the last value set for all variables within the loop.
- D13. Print out and punch out all the information for this DO LOOP.
- D14. Go to P3.
- P8. Test if it is a "INTEGER" statement. If yes, call the subroutine "SUB", then go to P13. If not, go to P9.
- P9. Test if it is a "REAL" statement. If yes, call the subroutine "SUB", then go to P13. If not, go to P10.
- P10. Test if it is a "IF" statement. If yes, then insert a print statement and a format of this print statement, then go to P13. If not, go to P11.

- P11. Test if it is a "GO TO" statement. If yes, insert a print statement and a format of this print statement, then go to P13. If not, go to P12.
- P12. Find the index of equal sign, and format this assignment statement.
- P13. Punch out the statement.
- P14. Print out the statement.
- P15. Go to P4.

The subroutine "SUB" program is to separate each INTEGER or REAL statement's identification and store them, the steps are as follows:

- S1. Declare variable used.
- S2. Do I=II to 72. (II is the beginning column of the define INTEGER or REAL's identification.)
- S3. Test if I=','. If yes, store all the character before ', ' mark, and go to S4. If not, go to S5.
- S4. Set II=I+1, go to S2.
- S5. Test if I=' '. If yes, store this last identification. If not, go to S2.
- S6. Return to the main program.

LIMITATION AND FURTHER SUGGESTIONS

There are some restrictions for the input FORTRAN program:

- 1) Every statement should begin in column 7 and reserved words must not include blanks.
- 2) For the "ASSIGNMENT" statement, its variable name cannot begin with READ, WRITE, IF, DO, REAL and EXPAND. For example, if

an ASSIGNMENT statement is:

READR=X+Y

then for the design in this paper it will treat it as a READ statement, and the same is true for WRITE, IF, DO, REAL and EXPAND.

- 3) The input data can not use READ, WRITE, IF, DO, REAL and EXPAND as an array name. For example, if a statement is:

READ(5)=10

the debugging program will treat it as a READ statement, not as an ASSIGNMENT.

- 4) Only un-nested DO LOOPS are accomodated by this debugging package.
5) The "DO" statement form should be as follows:

DO 10 I=1,100

or DO 10 I=1,100,2

one blank before and after the statement number is necessary.

If the "DO" statement is DO10I=1,100 then it won't work in the debugging program.

- 6) Within a DO LOOP, there can be no more than 20 statements.
7) The user can not use some 5 digit statement numbers in the program, such as 11111, 22222, 33333 to 55555, 66666, 77777 and 88888, as they are reserved for special use.
8) User can not have the "GO TO" statement as following form:

GO TO (n_1, n_2, \dots, n_k), NAME

where n_1, n_2, \dots, n_k is a list of statement numbers and NAME is a variable name.

- 9) The user can not have a statement more than one card in length, except a continued FORMAT statement.
- 10) The user can not have the equal sign in his FORMAT statement.
- 11) If statements are assumed to be followed by GO TO statements, not ASSIGNMENTS.

However improvement in terms of operation could be made through elimination of the above considerations. Improvements, in terms of providing more information can only be made after extensive use of such a facility as this program currently represents our 'best' guess.

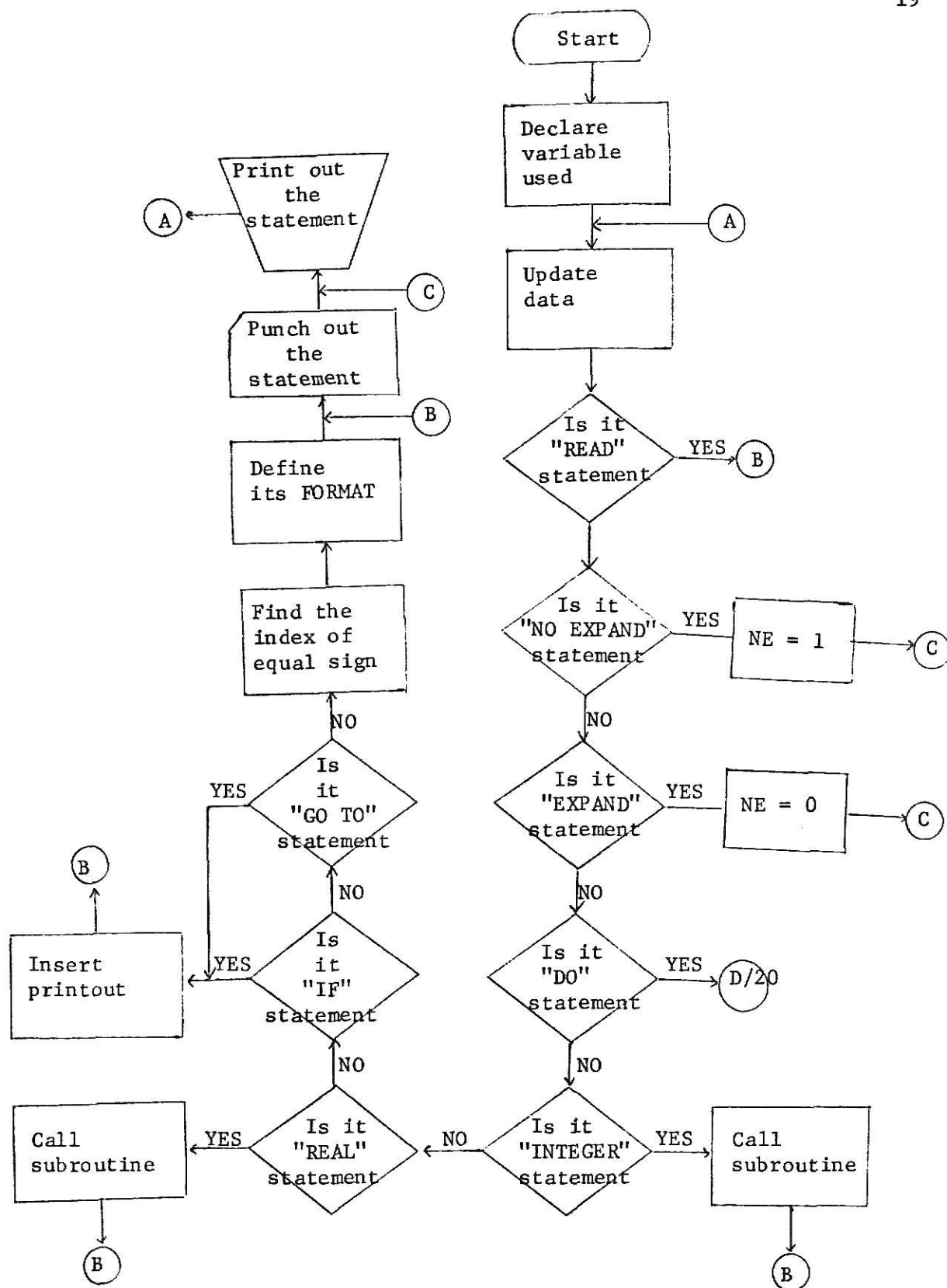
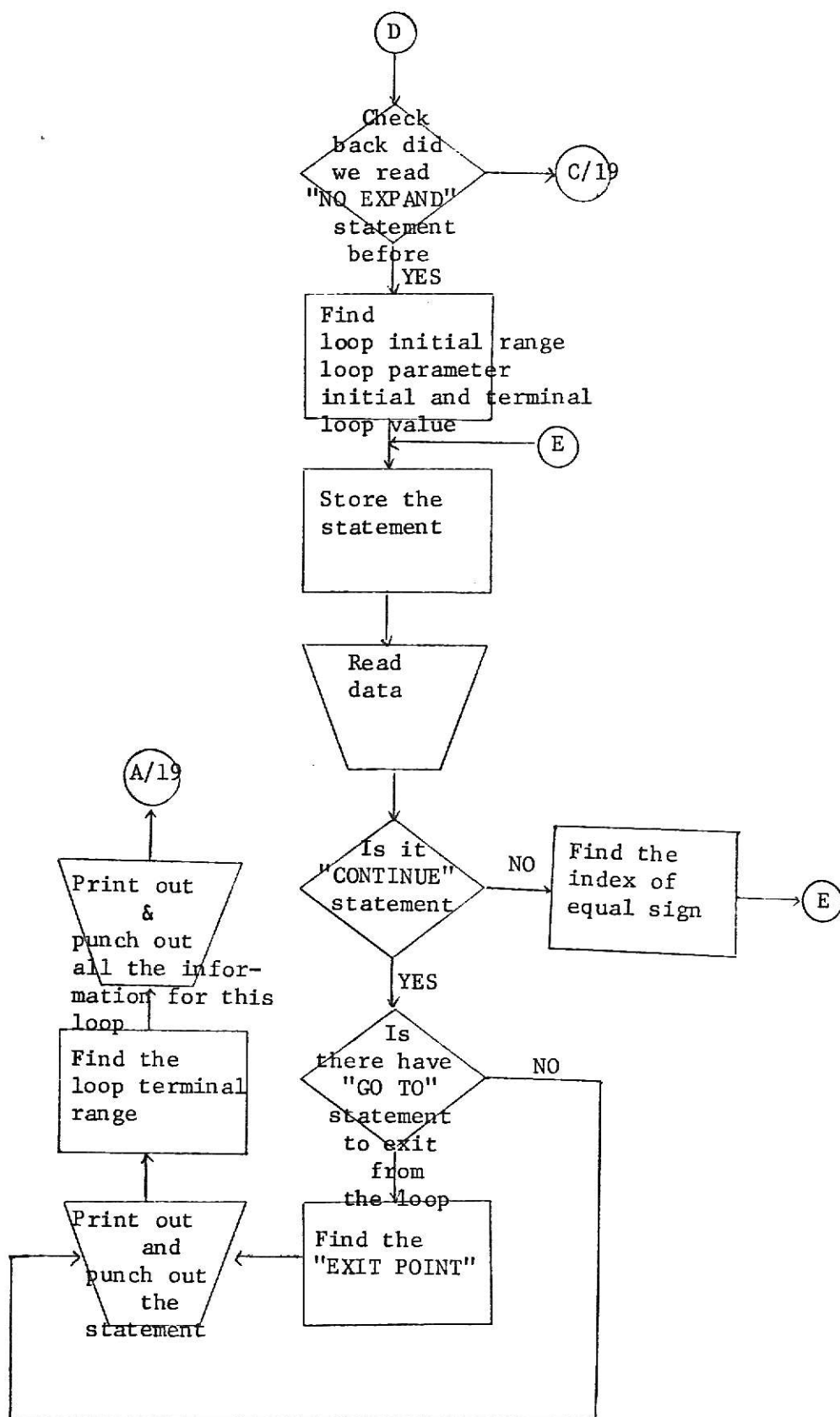


Figure 1. Flowchart for PL/I program



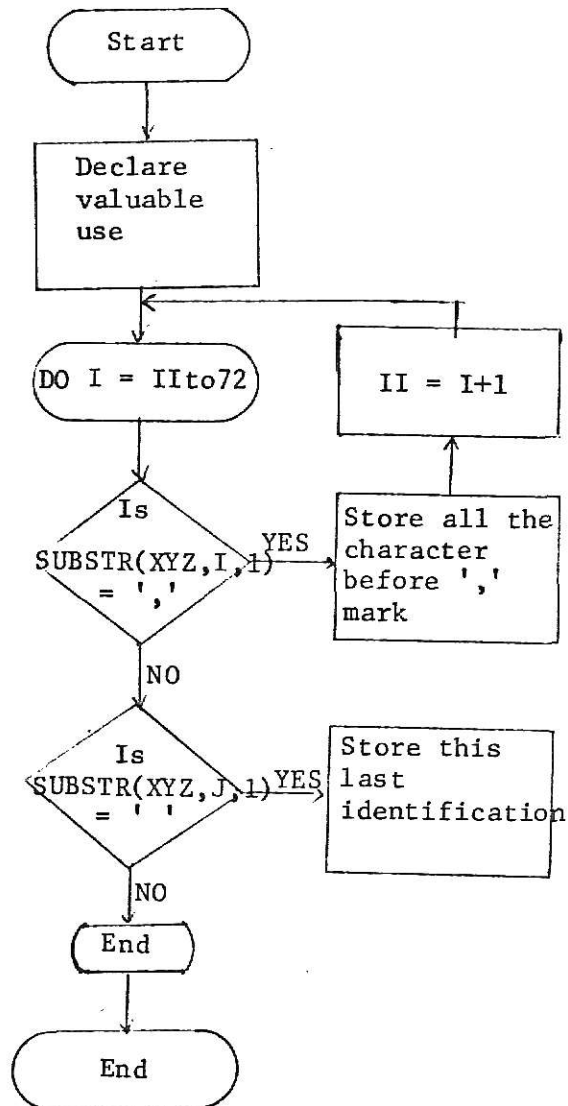


Figure 2. Flowchart for subroutine program

SUMMARY

This paper has presented in Chapter I a brief description of debugging and the debugging aids. It emphasizes that the debugging system is a process to aid the FORTRAN programmers in finding the logic errors.

An example using PL/I is presented in Appendix A (Chapter II), the purpose of the PL/I program is to aid students in finding the logic errors in a FORTRAN program. To debug the program, the system automatically inserts print statements to obtain intermediate results and information. Information is printed out to aid in showing the incorrect statements or the unexpected result, so that the error can be easily corrected.

Although logic errors are almost inevitable in programming, after using this debugging scheme, most logic errors will be easily identified by the programmer. Thus such a system can save a great deal of time in searching for errors as well as indicate flow of control under various input can computation possibilities.

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to Dr. Paul Fisher, Major Professor, for his enthusiastic guidance and constructive suggestions throughout this report and the Master program.

Also, I express my appreciation to Dr. Kenneth Conrow and Dr. Myron Calhoun for their assistance and advice during the course of the program and to the Computing Center which provided the computation time.

BIBLIOGRAPHY

- Balzer, R. M. "EXDAMS-Extendable Debugging and Monitoring System," AFIPS Conference Proceedings Spring Joint Computer Conference, 1969, pp. 567-580.
- Chandor, A., Granham, J. and Williamson, R. A Dictionary of Computers, Australia: Penguin Books, 1970.
- Evans, Thomas G. and Darley, D. Lucille "On-Line Debugging Techniques: A Survey," AFIPS Conference Proceedings Fall Joint Computer Conference, 1966, p. 43.
- IBM Corporation, IBM System 360 Operation System Programmer's Guide to Debugging, IBM Form GC28-6670-3.
- Schwartz, Jacob T. "An Overview of Bugs," Debugging Techniques in Large Systems, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1971.

APPENDIX A

REPORT Program Variable List

<u>VARIABLE</u>	<u>MODE</u>	<u>DESCRIPTION</u>
XYZ	CHAR(80)	Read in FORTRAN statement
LABEL	CHAR(5)	Label of FORTRAN statement
ABC	CHAR(75)	FORTRAN statement from col. 7 to col. 80
IDTF	CHAR(6)	FORTRAN variable name
FORMAT	CHAR(75)	FORMAT for the print statement
I	FIXED	Scalar variable
N	FIXED	Number of read in statement
K	FIXED	Scalar variable
KK	FIXED	Label of format statement
L	FIXED	Label of format statement
LL	CHAR(75)	Format statement
M	FIXED	Label of format statement
MM	CHAR(75)	Format statement
FCHAR	CHAR(1)	The first character of the identifier name
PHRASE	CHAR(5)	Mode of the identifier
I1	CHAR(1)	INIT('I')
J1	CHAR(1)	INIT('J')
K1	CHAR(1)	INIT('K')
L1	CHAR(1)	INIT('L')
M1	CHAR(1)	INIT('M')
N1	CHAR(1)	INIT('N')
VARI(20)	CHAR(6)	The integer variable name
VARR(20)	char(6)	The real variable name

<u>VARIABLE</u>	<u>MODE</u>	<u>DESCRIPTION</u>
J1	FIXED	Scalar variable
JR	FIXED	Scalar variable
II	FIXED	Scalar variable
LR	FIXED	Loop initial range
LTR	FIXED	Loop terminal range
LE	FIXED	Scalar variable
LC	FIXED	Scalar variable
C	FIXED	Scalar variable
D	FIXED	Scalar variable
BBB(20)	FIXED	The number of DO LOOP statement
A	FIXED	Scalar to label in the DO LOOP
B	FIXED	Scalar to count statement in the DO LOOP
D	FIXED	Scalar to count the ASSIGNMENT statement in the DO LOOP
NE	FIXED	EXPAND or NO EXPAND specified
ILV	CHAR(6)	Initial loop value
TLV	CHAR(6)	Terminal loop value
PAR	CHAR(6)	Loop parameter
EE(10)	CHAR(5)	Store the ASSIGNMENT statement's identifier which within the DO LOOP
AA(10)	CHAR(5)	Store FORTRAN statement's label which within the DO LOOP
CC(10)	CHAR(5)	Store the GO TO statement which within the DO LOOP
BB(10)	CHAR(80)	Store all the statements which within the DO LOOP

<u>VARIABLE</u>	<u>MODE</u>	<u>DESCRIPTION</u>
TR(20)	CHAR(5)	Transfer's name
EXIT(20)	FIXED	Exit point
F	PIC'99999'	Scalar variable
H	PIC'99999'	Scalar variable
J	PIC'99999'	Scalar variable
FF(20)	CHAR(80)	Print statement
HH(20)	CHAR(80)	Format statement
GG(20)	CHAR(80)	GO TO statement

ILLEGIBLE DOCUMENT

**THE FOLLOWING
DOCUMENT(S) IS OF
POOR LEGIBILITY IN
THE ORIGINAL**

**THIS IS THE BEST
COPY AVAILABLE**

ILLEGIBLE

**THE FOLLOWING
DOCUMENT (S) IS
ILLEGIBLE DUE
TO THE
PRINTING ON
THE ORIGINAL
BEING CUT OFF**

ILLEGIBLE

APPENDIX B

```

REPORT: PROCEDURE OPTIONS(MAIN);
/* THIS PROGRAM IS USE PL/1 TO DEBUGGING A FORTRAN PROGRAM */
/* AND PUNCH IT OUT */
ON ENDFILE(SYSIN)
GO TO QUIT;
/* DECLARE VARIABLES USED */
DCL XYZ CHAR(80);
DCL LABEL CHAR(5), ABC CHAR(75);
DCL IDTF CHAR(6) VAR, FORMAT CHAR(75) VAR;
DCL (I,N,K,KK) FIXED;
DCL L FIXED INIT(22221), LL CHAR(75) VAR;
DCL M FIXED INIT(11110), MM CHAR(75) VAR;
DCL FCHAR CHAR(1), PHRASE CHAR(5) VAR;
DCL I1 CHAR(1) INIT('I'),
    J1 CHAR(1) INIT('J'),
    K1 CHAR(1) INIT('K'),
    L1 CHAR(1) INIT('L'),
    M1 CHAR(1) INIT('M'),
    N1 CHAR(1) INIT('N');
DCL (VARI,VARR)(20) CHAR(6) VAR;
DCL (JI,JR) FIXED INIT(0), I1 FIXED;
DCL (LR,LTR,LE,LC,C,D,BBB(20)) FIXED,
    (A,B,E,NE) FIXED INIT(0),
    (ILV,TLV,PAR,EE(10)) CHAR(6) VAR,
    (AA,CC)(10) CHAR(5) VAR, BB(20) CHAR(80);
DCL TR(20) CHAR(5), EXIT(20) FIXED,
    F PICTURE '99999' INIT(55554),
    H PICTURE '99999' INIT(66665),
    J PICTURE '99999' INIT(0),
    (FF,HH,GG)(20) CHAR(80);
/* PRINT LABELS */
PUT SKIP EDIT('NUMBER GIVEN','ASSIGNED')(A,X(5),A);
/* INITIALIZE SOME VALUES */
N=0;
KK=33332;
READ: GET EDIT(XYZ)(A(80));
/* COUNT THE ORIGINAL READ IN STATEMENTS */
N=N+1;
LABEL=SUBSTR(XYZ,1,5);
ABC=SUBSTR(XYZ,6,75);
IF SUBSTR(XYZ,7,4)='READ' THEN
GO TO OUTPUT;
IF SUBSTR(XYZ,7,9)='NO EXPAND' THEN
DO;
NE=1;
GO TO PRINT;
END;
IF SUBSTR(XYZ,7,6)='EXPAND' THEN
DO;
NE=0;
GO TO PRINT;
END;
IF SUBSTR(XYZ,7,2)='DO' THEN
DO;
/* IF IT IS THE 'DO' STATEMENT THEN CHECK BACK, DID WE READ THE */
/* 'NO EXPAND' STATEMENT BEFORE, IF YES THEN WE ARE NOT GOING TO */
/* EXPAND THE FOLLOWING DO LOOP */

```

```

/* IN THE FOLLOWING LR AND LTR INDICATED THE LOOP RANGE,          */
/* PAR:LOOP PARAMETER, ILV:INITIAL LOOP VALUE, TLV:TERMINAL LOOP  */
/* VALUE, EXIT:EXIT POINT                                         */
/* IF NE=1 THEN                                                    */
/* GO TO OUTPUT;                                                  */
LR=N;
LE=INDEX(XYZ,'=');
DO I=LE TO 10 BY-1;
  IF SUBSTR(XYZ,I,1)=' ' THEN
    PAR=SUBSTR(XYZ,I+1,LE-I-1);
  END;
LC=INDEX(XYZ,',');
DO I=LC+1 TO LC+6;
  IF SUBSTR(XYZ,I,1)=',' THEN
    GO TO V;
  IF SUBSTR(XYZ,I,1)=' ' THEN
    GO TO V;
  END;
V:
  ILV=SUBSTR(XYZ,LE+1,LC-LE-1);
  TLV=SUBSTR(XYZ,LC+1,I-LC-1);
D1:
  R=B+1;
/* KEEP ALL THE STATEMENTS WHICH WITHIN THIS DO LOOP */
  BB(B)=XYZ;
  BBB(B)=N;
  GET EDIT(XYZ)(A(80));
  N=N+1;
  LABEL=SUBSTR(XYZ,1,5);
  ABC=SUBSTR(XYZ,6,75);
  I=VERIFY(LABEL,' ');
  IF I=0 THEN
    DO;
      A=A+1;
/* KEEP THE LABELS WHICH WITHIN THIS DO LOOP */
      AA(A)=SUBSTR(LABEL,I,5-I+1);
      END;
      IF SUBSTR(XYZ,7,8)='CONTINUE' THEN
        DO;
          I=INDEX(E(E),',');
          EE(E)=SUBSTR(E(E),1,I-1);
/* TEST IN THE DO LOOP DOES THERE HAVE 'GO TO' STATEMENT TO EXIT */
/* FROM THE LOOP */
          DO I=1 TO B;
            C=INDEX(BB(I),'GO TO ');
            IF C=0 THEN
              GO TO D2;
            D=INDEX(BB(I),',');
            CC(I)=SUBSTR(BB(I),C+6,D-C-6);
            DO K=1 TO A;
              IF CC(I)=AA(K) THEN
                GO TO D2;
            END;
            J=J+1;
            EXIT(J)=BBB(I);
            TR(J)=CC(I);
            F=F+1;
            BB(I)=SUBSTR(BB(I),1,C+5)||F;
            H=H+1;

```

```

FF(J)=F||' PRINT '||H;
HH(J)=H||' FORMAT(1H ,10X,'EXIT POINT='||EXIT(
)||' TR '||TR(J)||''');
GG(J)='GO TO '||CC(I);
D2: PUT SKIP EDIT(SUBSTR(BB(1),1,5),BBB(1),SUBSTR(BB
(1),6,75))(X(2),A(5),X(12),F(4),X(7),A);
PUT FILE(PUNCH)EDIT(BR(1))(COL(1),A);
END;
D3: LTR=N;
PUT SKIP EDIT('IF(',PAR,'.EQ.',ILV,')PRINT 77777,
(EF(1)DO I=1 TO E))(X(35),A,A,A,A,A,10(A));
PUT FILE(PUNCH)EDIT('IF(',PAR,'.EQ.',ILV,')PRINT 7
777,',(EF(1)DO I=1 TO E))(COL(7),A,A,A,A,A,(E)A);
PUT SKIP EDIT(LABEL,N,ABC)(X(2),A(5),X(12),F(4),X(
),A);
PUT FILE(PUNCH)EDIT(LABEL,ABC)(COL(1),A(5),A(75));
PUT SKIP EDIT('IF(',PAR,'.EQ.',TLV,') PRINT 88888,
,(EF(1)DO I=1 TO E))(X(35),A,A,A,A,A,(E)A);
PUT FILE(PUNCH)EDIT('IF(',PAR,'.EQ.',TLV,') PRINT
8888,',(EF(1)DO I=1 TO E))(COL(7),A,A,A,A,A,(E)A);
PUT EDIT('77777 FORMAT(1H ,10X,'LOOP RANGE=',LR,
TO ',LTR,','/11X,'LOOP PARAMETER=',PAR,','/11X,',',1''INITIAL LOOP VALU
E=',ILV,','/5X,'TERMINAL LOOP VALUE=',TLV,','/11X,',',2''FIRST VALUES
ET ',(EF(1)DO I=1 TO E),','/E,'(I10))')(SKIP,X(35),A,F(2),A,F(2),A,
,A,SKIP,X(40),A,A,A,A,A,SKIP,X(40),A,(E)A,A,F(2),A);
PUT FILE(PUNCH)EDIT('77777 FORMAT(1H ,10X,'LOOP R
ANGE=',LR,' TO ',LTR,','/11X,'LOOP PARAMETER=',PAR,','/11X,',',1''INITIA
L LOOP VALUE=',ILV,','/5X,'TERMINAL LOOP VALUE=',TLV,','/11X,',',2''F
RST VALUES SET ',(EF(1)DO I=1 TO E),','/E,'(I10))')(COL(1),A,F(2),A
F(2),A,A,A,SKIP,COL(6),A,A,A,A,A,SKIP,COL(6),A,(E)A,A,F(2),A);
PUT SKIP EDIT('88888 FORMAT(1H ,10X,'LAST VALUES
ET ',(EF(1)DO I=1 TO E),','/E,'(I10))')(X(35),A,(E)A,A,F(2),A);
PUT FILE(PUNCH)EDIT('88888 FORMAT(1H ,10X,'LAST V
ALUES SET ',(EF(1)DO I=1 TO E),','/E,'(I10))')(COL(1),A,(E)A,A,F(2),
);
GO TO READ;
END;
I=INDEX(XYZ,'=');
IF I=0 THEN
GO TO D1;
E=E+1;
EE(E)=SUBSTR(XYZ,7,I-7)||',';
GO TO D1;
/* END OF DO LOOP */
END;
IF SUBSTR(XYZ,7,7)='INTEGER' THEN
DO;
DO I=14,16;
IF SUBSTR(XYZ,I,1)=' ' THEN
DO;
II=I+1;
CALL SUB(XYZ,II,VARI,JI);
GO TO OUTPUT;
END;
END;
END;
IF SUBSTR(XYZ,7,4)='REAL' THEN

```

```

DO;
  DO I=11,13;
    IF SUBSTR(XYZ,I,1)=' ' THEN
      DO;
        II=I+1;
        CALL SUB(XYZ,II,VARR,JR);
        GO TO OUTPUT;
      END;
    END;
  END;
  IF SUBSTR(XYZ,7,2)='IF' THEN
    DO;
      PUT SKIP EDIT(LABEL,N,ABC)(X(2),A(5),X(12),F(4),X(7),A(75)
    );
      M=M+1;
      MM=' FORMAT(1H ,10X, '||N||' NT')';
      PUT SKIP EDIT('PRINT ',M)(X(35),A,F(5));
      PUT SKIP EDIT(M,MM)(X(35),F(5),A);
      PUT FILE(PUNCH)EDIT(LABEL,ABC)(COL(1),A(5),A(75));
      PUT FILE(PUNCH)EDIT('PRINT ',M)(COL(7),A,F(5));
      PUT FILE(PUNCH)EDIT(M,MM)(COL(1),F(5),A);
      GO TO READ;
    END;
    IF SUBSTR(XYZ,7,5)='GO TO' THEN
      DO;
        L=L+1;
        LL=' FORMAT(1H ,10X, '||N||' TR'||SUBSTR(XYZ,12,5)||
        '||)';
        PUT SKIP EDIT('PRINT ',L)(X(35),A,F(5));
        PUT SKIP EDIT(L,LL)(X(35),F(5),A);
        PUT FILE(PUNCH)EDIT('PRINT ',L)(COL(7),A,F(5));
        PUT FILE(PUNCH)EDIT(L,LL)(COL(1),F(5),A);
        PUT FILE(PUNCH)EDIT(LABEL,ABC)(COL(1),A(5),A(75));
        PUT SKIP EDIT(LABEL,N,ABC)(X(2),A(5),X(12),F(4),X(7),A);
        IF J=0 THEN
          DO;
            DO I=1 TO J;
              PUT EDIT(FF(I),HH(I),GG(I))(SKIP,X(35),A,SKIP,X
              35),A,SKIP,X(35),A);
              PUT FILE(PUNCH)EDIT(FF(I),HH(I),GG(I))(COL(1),A
              COL(1),A,COL(7),A(66));
            END;
            J=0;
            GO TO READ;
          END;
          GO TO READ;
        END;
      K=7;
      /* IN THIS PROGRAM WANT PRINT ALL HAVE EQUAL SIGN STATEMENTS' VALUE *
      /* BEFORE PRINT IT OUT SHOULD CHECK IS IT ALL READY DEFINE TO A *
      /* INTEGER NUMBER, REAL NUMBER OR NOT, IF NOT THEN DEFINE IT *
      I=INDEX(XYZ,'=');
      IF I=0 THEN
        GO TO OUTPUT;
      K=I-K;
      IDTF=SUBSTR(XYZ,7,K);
      KK=KK+1;

```

```

      FCHAR=SUBSTR(XYZ,7,1);
      IF JI=0&&JR=0 THEN
        GO TO T2;
      IF JI=0 THEN
        GO TO T1;
      DO I=1 TO JI;
        IF IDTF=VARI(I) THEN
          DO;
          PHRASE='I10';
          GO TO FORM;
          END;
        END;
T1:    IF JR=0 THEN
        GO TO T2;
      DO I=1 TO JR;
        IF IDTF=VARR(I) THEN
          DO;
          PHRASE='F10.2';
          GO TO FORM;
          END;
        END;
T2:    IF INDEX('IJKLMN',FCHAR)=-0 THEN
        PHRASE='I10';
      ELSE
        PHRASE='F10.2';
FORM:  FORMAT=' FORMAT(1H ,10X, '||N||'      '||IDTF||', '||PHRASE||'
)';

      PUT SKIP EDIT(LABEL,N,ABC)(X(2),A(5),X(12),F(4),X(7),A);
      PUT SKIP EDIT('PRINT ',KK,',',IDTF)(X(35),A,F(5),A,A);
      PUT SKIP EDIT(KK,FOPMAT)(X(35),F(5),A);
      PUT FILE(PUNCH)EDIT(LABEL,ABC)(COL(1),A(5),A(75));
      PUT FILE(PUNCH)EDIT('PRINT ',KK,',',IDTF)(COL(7),A,F(5),A,A(K)
;
      PUT FILE(PUNCH)EDIT(KK,FORMAT)(COL(1),F(5),A);
      GO TO READ;
OUTPUT: PUT FILE(PUNCH)EDIT(LABEL,ABC)(COL(1),A(5),A(75));
PRINT:  PUT SKIP EDIT(LABEL,N,ABC)(X(2),A(5),X(12),F(4),X(7),A);
      GO TO READ;
/* SUBROUTINE 'SUB' TO SEPERATE THE INTEGER & REAL'S IDENTIFICATION *
/* AND STORE IT *
SUB:    PROCEDURE(XYZ,II,IDEN,J);
        DCL XYZ CHAR(80), IDEN(20) CHAR(6) VAR;
        DCL (II,J) FIXED;
XX:     DO I=II TO 72;
        IF SUBSTR(XYZ,I,1)='.' THEN
          GO TO YY;
        IF SUBSTR(XYZ,I,1)=' ' THEN
          DO;
          J=J+1;
          IDEN(J)=SUBSTR(XYZ,II,I-II);
          PUT SKIP EDIT(IDEN(J))(A);
          GO TO S1;
        END;
      END;
YY:     J=J+1;
        IDEN(J)=SUBSTR(XYZ,II,I-II);
        PUT SKIP EDIT(IDEN(J))(A);

```

```
      II=I+1;  
      GO TO XX;  
S1:    RETURN;  
      END SUB;  
QUIT:  END;
```


APPENDIX C

```
      INTEGER AX,BY,CZ,HD  
1  READ,A,B,C  
   PRINT,A,B,C  
   IF (A.EQ.0.0) STOP  
   Y=(A**3+B*C-2.*B/A)*C-B  
   PRINT,A,B,C,Y  
   NUM1=0  
   NUM2=1  
   NO EXPAND  
   DO 10 I=1,10  
     II=I  
     NUM3=NUM1+NUM2  
     IF (I.EQ.10) GO TO 2  
     GO TO 3  
2  PRINT,II  
   IF (.TRUE.) GO TO 1  
3  NUM1=NUM2  
   NUM2=NUM3  
10 CONTINUE  
   EXPAND  
   GO TO 1  
   STOP  
   END
```

APPENDIX D

NUMBER GIVEN ASSIGNED

AX
BY
CZ
HD

	1	INTEGER AX,BY,CZ,HD		
	2	READ A,B,C		
	3	PRINT A,B,C		
1	4	IF (A.EQ.0.0) STOP		
		PRINT 11111		
		11111 FORMAT(1H ,10X, *	4	NT*)
	5	Y=(A**3+B*C-2.*B/A)*C-8		
		PRINT 33333,Y		
		33333 FORMAT(1H ,10X, *	5	Y',F10.2)
	6	PRINT A,B,C,Y		
	7	NUM1=0		
		PRINT 33334,NUM1		
		33334 FORMAT(1H ,10X, *	7	NUM1',I10)
	8	NUM2=1		
		PRINT 33335,NUM2		
		33335 FORMAT(1H ,10X, *	8	NUM2',I10)
	9	NO EXPAND		
	10	DO 10 I=1,10		
	11	II=*		
	12	NUM3=NUM1+NUM2		
	13	IF (I.EQ.10) GO TO 2		
	14	GO TO 3		
2	15	PRINT,II		
	16	IF (.TRUE.) GO TO 55555		
3	17	NUM1=NUM2		
	18	NUM2=NUM3		
10	19	IF(I.EQ.1)PRINT 77777,II,NUM3,NUM1,NUM2		
		CONTINUE		
		IF(I.EQ.10) PRINT 33338,II,NUM3,NUM1,NUM2		
		77777 FORMAT(1H ,10X, *LOOP RANGE=10 TO 10*/11X, *LOOP PARAMETER=1*/11X,		
		1*INITIAL LOOP VALUE=1*,5X, *TERMINAL LOOP VALUE=10*/11X,		
		2*FIRST VALUES SET II,NUM3,NUM1,NUM2', 4(110))		
		33338 FORMAT(1H ,10X, *LAST VALUES SET II,NUM3,NUM1,NUM2', 4(110))		
	20	EXPAND		
		PRINT 22222		
		22222 FORMAT(1H ,10X, *	21	TR 1 ')
	21	GO TO 1		
		55555 PRINT 66666		
		66666 FORMAT(1H ,10X, *EXIT POINT=	16	TR 1 ')
		GO TO 1		
	22	STOP		
	23	END		

APPENDIX E

```

1  1J 00
2  1  READ,A,B,C
3  PRINT,A,B,C
4  IF (A.D.O.O) STOP
5  PRINT 11111
6  11111 FORMAT(1H,10X,'      4      NT')
7  Y=(3372+84C-2.78/2)*C-D
8  PRINT 33333,Y
9  33333 FORMAT(1H,10X,'      5      Y',F10.2)
10 PRINT,A,B,C,Y
11 NUM1=0
12 PRINT 33334,NUM1
13 33334 FORMAT(1H,10X,'      7      NUM1',110)
14 NUM2=1
15 PRINT 33335,NUM2
16 33335 FORMAT(1H,10X,'      8      NUM2',110)
17 DO 10 I=1,10
18   II=I
19   NUM3=NUM1+NUM2
20   IF (I.EQ.10) GO TO 2
21   GO TO 3
22 2 PRINT,II
23 IF (.TRUE.) GO TO 55555
24 1 NUM1=NUM2
25 NUM2=NUM3
26 IF(I.EQ.1)PRINT 77777,II,NUM3,NUM1,NUM2
27 10 CONTINUE
28 IF(I.EQ.10) PRINT 88888,II,NUM3,NUM1,NUM2
29 77777 FORMAT(1H,10X,'LOOP RANGE=1 TO 10',11X,'LOOP PARAMETER=1',11X,
30 'INITIAL LOOP VALUE=1',5X,'TERMINAL LOOP VALUE=10',11X,
31 'FIRST VALUES SET II,NUM3,NUM1,NUM2',4(110))
30 88888 FORMAT(1H,10X,'LAST VALUES SET II,NUM3,NUM1,NUM2',4(110))
31 PRINT 22222
32 22222 FORMAT(1H,10X,'      21      TR 1  ')
33 GO TO 1
34 55555 PRINT 66666
35 66666 FORMAT(1H,10X,'EXIT POINT=      16      TR 1  ')
36 GO TO 1
37 STOP
**WARNING** UNNUMBERED EXECUTABLE STATEMENT FOLLOWS A TRANSFER
38 END

```

```

ENTRY
0.1000000E 01 0.2000000E 01 0.3000000E 01
4 NT
5 Y 7.00
0.1000000E 01 0.3000000E 01 0.3000000E 01 0.7000000E 01
7 NUM1 0
9 NUM2 1
LOOP RANGE=10 TO 19
LOOP PARAMETER=1
INITIAL LOOP VALUE=1 TERMINAL LOOP VALUE=10
FIRST VALUES SET II,NUM3,NUM1,NUM2 1 1 1 1
10
EXIT POINT= 16 TR 1
0.4000000E 01 0.5000000E 01 0.6000000E 01
4 NT
5 Y 544.00
0.4000000E 01 0.5000000E 01 0.6000000E 01 0.5440000E 03

```

```

      7      NUM1      0
      0      NUM2      1
LOOP RANGE=1 TO 10
LOOP PARAMETER=1
INITIAL LOOP VALUE=1      TERMINAL LOOP VALUE=10
FIRST VALUES SET 11,NUM3,NUM1,NUM2      1      1      1
10
EXIT POINT=      16      TO 1
0.00000000 00      0.70000000 01      0.80000000 01
CORE USAGE      OBJECT CODE=      1616 BYTES,ARRAY AREA=      0 BYTES,TOTAL AREA AVAILABLE=      45312 BYTES
DIAGNOSTICS      NUMBER OF ERRORS=      0, NUMBER OF WARNINGS=      1, NUMBER OF EXTENSIONS=      0
COMPILE TIME=      1.20 SEC,EXECUTION TIME=      0.74 SEC, WATFIV - VERSION 1 LEVEL 3 MARCH 1971      DATE= 73/053

```

APPENDIX F

NUMBER GIVEN	ASSIGNED			
AX				
BY				
CZ				
HD				
1	1	INTEGER AX, BY, CZ, HD		
	2	READ, A, B, C		
	3	PRINT, A, B, C		
	4	IF (A.EQ.0.0) STOP		
		.PRINT 11111		
		11111 FORMAT(1H, 10X, '	4	NT')
	5	Y=(A**3+B*C-2.*B/A)*C-B		
		PPRINT 33333, Y		
		33333 FORMAT(1H, 10X, '	5	Y', F10.2)
	6	PRINT, A, B, C, Y		
	7	NUM1=C		
		PRINT 33334, NUM1		
		33334 FORMAT(1H, 10X, '	7	NUM1', I10)
	8	NUM2=1		
		PRINT 33335, NUM2		
		33335 FORMAT(1H, 10X, '	8	NUM2', I10)
	9	EXPAND		
	10	DO 10 I=1, 10		
	11	II=1		
		PRINT 33336, II		
		33336 FORMAT(1H, 10X, '	11	II', I10)
	12	NUM3=NUM1+NUM2		
		PRINT 33337, NUM3		
		33337 FORMAT(1H, 10X, '	12	NUM3', I10)
	13	IF (I.EQ.10) GO TO 2		
		PRINT 11112		
		11112 FORMAT(1H, 10X, '	13	NT')
		PPRINT 22222		
		22222 FORMAT(1H, 10X, '	14	TR 3 ')
	14	GO TO 3		
2	15	PRINT, II		
	16	IF (.TRUE.) GO TO 1		
		PRINT 11113		
		11113 FORMAT(1H, 10X, '	16	NT')
3	17	NUM1=NUM2		
		PRINT 33338, NUM1		
		33338 FORMAT(1H, 10X, '	17	NUM1', I10)
	18	NUM2=NUM3		
		PRINT 33339, NUM2		
		33339 FORMAT(1H, 10X, '	18	NUM2', I10)
10	19	CONTINUE		
		PRINT 22223		
		22223 FORMAT(1H, 10X, '	20	TP 1 ')
	20	GO TO 1		
	21	STOP		
	22	END		

AN APPROACH TO DEBUGGING

by

CHING-NEU HONG

B. A., Fu-Jen Catholic University, 1969

AN ABSTRACT OF A MASTER'S REPORT

Submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1973

ABSTRACT

The purpose of this paper is to introduce the topic of debugging and debugging aids with respect to those parts of a program which are not amendable to ordinary trace techniques. Debugging is the technique of detecting, diagnosing and correcting errors which may occur in programs or systems. For this study, the design of some techniques to aid the programmer in finding logic errors which may occur in programs is shown.

For illustration, a program written in PL/I is used as a debugging tool to debug logic errors which occur in FORTRAN programs. The PL/I program processes the input FORTRAN program statement by statement, in the while the system automatically insert print statements to obtain intermediate results and information. These results can be checked to see whether the correct answer is obtained at various stages in the program. After using this debugging scheme most logic error could be easily identified by the programmer.