SLAC Computation Group
Stanford, California

CGTM NO. 136.1
November 1972

USERS GUIDE FOR THE FORTRAN H SYMBOLIC DEBUGGING PACKAGE

John P. Steffani

1. INTRODUCTION

The FORTRAN H Symbolic Debugging Package allows the user to view his FORTRAN H program's variables and their contents upon demand (via subroutine call) or upon program termination, either normal or abnormal.

The FORTRAN H Symbolic Debugging Package consists of a modified FORTRAN H compiler, a modified linkage editor and an execution time supervisor. A cataloged procedure for compile, linkage editing, and execution (FORTXCLG) has been added to SYS1.PROCLIB. The structure of this debugging system is such that a user need only use the cataloged procedure FORTXCLG in place of the standard FORTHCLG. The executable code produced by the modified compiler is identical to what will be produced by the standard FORTRAN H compiler. Due to the initialization of uninitialized program variables by the linkage editor and the execution time supervisor, consistent results will be the rule, not only for debugging but for production runs as well. User options are provided for control of the execution time supervisor. Provision has been made for explicit invocation of the symbolic print package by means of a subroutine call. The subroutine used for invocation acts as a null (or do nothing) subroutine in the event the user isn't running under the control of the debugging package. Printouts of trace backs include the value of the parameters passed. The contents of each variable are shown in character, integer, hex, and floating-point values.

In order to use the debugging facility, change FORTHCLG to FORTXCLG on the EXEC JCL card and submit the job as normal.

2. NEW COMPILER FEATURES

A. Production of Symbolic Debug Information

The compiler is modified to preserve the variable names, type and location, statement labels, etc., in the form of SYM records. No alteration is made to the executable code produced by the compiler.

B. Recognition of %INCLUDE Statement

The compiler now recognizes the %INCLUDE statement. This statement is used to include (incorporate) strings of text from external data sets (source statements) into the source program being scanned. Maximum block size of included data sets is 400 bytes. Logical record length is assumed to be 80 bytes. The format of the %INCLUDE statement:

%INCLUDE ddname (member-name), or

%INCLUDE member-name - implies DDname of SYSLIB by default.

The % must appear as the first non-blank character in columns 7 through 72.

The string 'INCLUDE' must follow the %. Blanks between the % and string 'INCLUDE' are ignored.

After the string 'INCLUDE' may be the member name, in which case the member is assumed to reside in a data set defined by the ddname SYSLIB. The alternate form is explicit specification of the ddname followed by the member name enclosed in parentheses. Only one member name within the parentheses is allowed.

The first non-blank encountered after the string 'INCLUDE' is assumed to begin the specification of the ddname or the member name. The ddname specifies the ddname occurring in the name field of the appropriate DD statement. Its associated "member name" specifies the name of the data set member to be incorporated as source text until the end of the member is reached, or another %INCLUDE is encountered. Up to four %INCLUDE statements may be nested. END statements may appear in the included source.

The %INCLUDE statement is treated as a comment by the compiler. Error messages resulting from error conditions, occurring in the INCLUDE processor, are printed as comments by the compiler. These messages are:

Message	Condition
BAD REQUEST	Illegal command in call to INCLUDE processor.
NO MEMBER	Member not found in data set.
CAN'T OPEN DS	Data set can't be opened.
READ ERROR	I/O error on read.
ZERO LENGTH ERROR	No data transferred from data set.
LENGTH ERROR	Data set blocksize not multiple of 80.
NOT OPEN	Attempt to read from included data set not currently open.

In all the above situations, the INCLUDE processor terminates the current INCLUDE member process and attempts to pop up to the next level INCLUDE if nesting is indicated, or to resume normal input stream process.

3. ADDITIONAL COMPILER DATA SET REQUIREMENTS

The modified compiler makes use of a temporary data set ddname SYSUT5 (this is included in the FORTXCLG procedure). Attempts to use the compiler without this data set will produce abend codes of 613 and 713.

4. INCREASE IN COMPILE TIME

A fractional increase in compile time is to be expected. Once again, this increase will be a function of the number of variables in the program/routine.

5. INCREASE IN COMPILER SIZE

The increase in the size of the root segment of the compiler is 3590 bytes.

6. DECK OR OBJECT MODULE PRODUCED BY COMPILER

Based upon the deck or load option selected, the object module produced by the compiler will contain additional information. The additions consist of SYM records containing the program variables and related information. On the average, three variables will be contained on each 80 character SYM record.

The SYM records themselves lend nothing whatsoever to the executability of the user program and are only informative in function.

When planning on saving the object module produced by the FORTXCLG compiler, remember that it will be larger than a corresponding object module produced by the standard FORTRAN H compiler. The increase in size is a function of the number of variables in the program/routine, etc.

7. ADDING JCL CARDS FOR %INCLUDE DATA SETS

When using the FORTXCLG procedure and the %INCLUDE source statement, the referenced ddnames must be included in the JCL for the compile or FORT step. This may best be illustrated by an example of just such a usage.

```
//ANYJOB  JOB XYZ,CLASS=A
//STEP1 EXEC FORTXCLG
//FORT.SCOMON DD DSN=GROUP.X.COMON,DISP=SHR
//FORT.SYSIN DD *
     REAL*8 ABLE(2)
     %INCLUDE XCOMON(DOUBL)
     %INCLUDE XCOMON(SGLE)
     I = 2
     . . .
```

In the above example, source statements from member DOUBL of data set GROUP.X.COMON and then member SGLE will be processed before the I = 2 statement.

```
//ANYJOB  JOB  XYZ,CLASS=A
//STEP1  EXEC  FORTXCLG
//FORT.SYSLIB DD DSN=GROUP.X.COMON,DISP=SHR
//FORT.X DD DSN=ABC,DISP=SHR
//FORT.SYSIN DD *
     REAL*8 ABLE(2)
     %INCLUDE DOUBL
     %INCLUDE SGLE
     %INCLUDE X(CMNT)
```

In the above example, source statements from data set GROUP.X.COMON members DOUBL and SGLE will be processed. Source input will then be directed to member CMNT of data set ABC.

8. UNINITIALIZED VARIABLES PRESET BY LINKAGE EDITOR

The area reserved for uninitialized variables that becomes part of the load module is preset to a bit pattern of 80808080... . This feature, when used with the VARDUMP1 program in the GO step, insures that all uninitialized variables, common blocks, etc., are preset to the above mentioned bit pattern.

9. RUN TIME OPTIONS

The GO step allows the user several options which may be specified by use of the step parameter passing facility of the EXEC JCL statement. The options and default settings are:

OPTION	DEFAULT	DESCRIPTION
ONLY/ALWAYS	ALWAYS	ONLY - will only produce symbol table printout in the event of an explicit call (i.e., call VARDMP) or abnormal termination of step. ALWAYS - will always produce symbol table printout, even if there is a normal GO step termination.
NOCOMMON/COMMON	COMMON	NOCOMMON - do not print variables in blank or labeled common blocks. COMMON - print all common variables.
SHORT/LONG	LONG	SHORT - print only map, trace back and registers. LONG - print map, trace back, registers, and all program variables.
TERSE/VERBOSE	VERBOSE	TERSE - each variable's value is expressed in its known type only. Four variables appear on each line. VERBOSE - the variable's value is expressed in character, hex, integer, and floating point formats and the variable's location (both relative and absolute) is printed.
NOTIMEOUT/TIMEOUT=n	TIMEOUT=1	NOTIMEOUT - will allow the program to run until CPU time is exceeded. However, no symbol print processing is then possible. TIMEOUT=n - when < n CPU seconds remain, terminate the program and produce a symbol table printout. The program will terminate with a 33E completion code.

upper limit = 20

OPTION	DEFAULT	DESCRIPTION
SIZE=X ARRAYSIZE=X	SIZE=20000	Limit the number of array elements to X for print purposes.

An example of the use of the option expression follows:

```
//ANYJOB  JOB  XYZ,CLASS=A
//CLG EXEC FORTXCLG,PARM.GO='ONLY,TIMEOUT=5"
//FORT.SYSIN DD*
     integer A(20)
     . . .
```

In the example given above, a printout will be produced only if the GO step abnormally terminates. Also, if the program is still executing and the CPU time remaining has been reduced to 5 seconds or less, then abort the user program with a completion code of 33E and produce a trace back and program variable printout.

10. CORE REQUIREMENTS OF THE GO STEP

In addition to the user problem program, approximately 13K bytes are required for program VARDUMP1.

11. ADDITIONAL DATA SET REQUIREMENTS FOR THE GO STEP

Two data set declarations (DD statements) are required for the GO step. These are included in the FORTXCLG procedure. The ddnames are USERLMOD and SYMBOLPR. USERLMOD points to the load module produced by the linkedit step and SYMBOLPR is the ddname for the output data set containing the symbol table printout.

12. OBJECT/LOAD MODULE COMPATIBILITY

Any valid OS/360 object modules and/or load modules may be included at link-edit time. However, program variables can only be printed for those routines that have been processed by the modified FORTRAN H compiler.

13. OUTPUT OF THE FORTXCLG DEBUGGING PACKAGE

The EXEC step or GO step of the FORTXCLG procedure produces a listing of the program's variables and their contents. The listing is directed to the data set defined by the DDname SYMBOLPR. A sample printout is found in CGTM No. 136.

The load map produced by the execution time supervisor lists both relative and absolute core addresses.

The date and time on the heading of each page are obtained when the execution time supervisor obtains control, either by explicit call or termination, of the user problem program.

The order in which a program's variables appear in the listing is alphabetical by symbol length. That is, all one character variables, A through Z, then all two character variables, AA etc.

The structure of the listing for a program's variables consists of the variable name, the variable type, T or F indicating TRUE or FLASE for logical varaibles, R or I indicating real or imaginary for complex variables, relative and absolute core locations, and the value of the variable expressed in character format, integer, hex and floating-point. Uninitialized variables are identified by their contents. The value 808080... is used to indicate uninitialized variables by the linkage editor and the execution time supervisor. Common blocks are printed upon the first occurrence of the common block unless the NOCOMMON option was expressed in the PARM.GO field. In the event that the common block is extended in some ensuing routine, then the listing for that common block begins where the previous list ended.

If the length of the common block used in the particular routine is less than the length of the common block loaded for the program, then this is indicated by the message 'SHORTENED COMMON BLOCK'. If the common block is longer than its last occurrence in a previous routine, the message 'EXTENDED COMMON BLOCK' is printed.

Arrays are treated as single dimensioned. The maximum number of elements to be printed can be specified by the SIZE=n or ARRAYSIZE=n parameter in the PARM.GO field. Contiguous elements of an array containing an identical value are not printed save for the first element. The range of the elements containing the same value is printed on the next line.

14. ABNORMAL TERMINATION IN THE EXEC STEP DUE TO THE SYMBOLIC DEBUGGING PACKAGE

The following abend codes are issued by the module SYMPRINT in the GO step of the FORTXCLG debugging package:

ABEND CODE	DESCRIPTION
747	No TCB address resulted from the attach of the user problem program.
749	Invalid input from the load module produced by the LKED step.
750	Address of allocated core is invalid.
751	Member 'MAIN' can't be found in the data set defined by ddname 'USERLMOD'.
752	Open was unsuccessful for data set(s) defined by either USERLMOD or SYMBOLPR ddnames.

15. ROUTINE'S VARIABLES PRINTED BY CALL FROM USER PROGRAM

The user may explicitly invoke the symbolic print function by including the FORTRAN source statement "CALL VARDMP" in the routine whose variables are desired to be printed. Every execution of this statement will cause the routine's variables and their contents to be printed. This process is in addition to printing the program's variables at termination time.

16. USE OF VARDMP WITH EXTENDED ERROR MESSAGE FACILITY

The subroutine VARDMP may be used as the routine to be executed upon occurrence of the specified error. The symbolic debugging package will then attempt to identify the user routine in which the error occurred, and to print out the variables of that routine. As an example, if the following two source statements were included in a portion of the user program that is executed, then upon any occurrence of

```
external VARDMP
call errset(207,0,0,0,VARDMP,208)
```

error numbers 207 or 208 (exponent overflow and exponent underflow), the variables in the routines in which the overflow or underflow occurred will be printed out along with their current contents.