Sept 1959

THE 704 FORTRAN II AUTOMATIC CODING SYSTEM

Grace E. Mitchell

IBM

RESEARCH CENTER

INTERNATIONAL BUSINESS MACHINES CORPORATION
YORKTOWN HEIGHTS, NEW YORK

RC-136

THE 704 FORTRAN II AUTOMATIC CODING SYSTEM

by

Grace E. Mitchell

International Business Machines Corporation
Research Center
Yorktown Heights, New York

ABSTRACT: This paper discusses the addition made in the FORTRAN I translator to produce the FORTRAN II translator. The new source language statements, debugging facilities and loader are described.

Research Report
RC-136
September 4, 1959

I–Introduction

After the field distribution of FORTRAN I, it became clear that two principle areas needed improvement, namely facilities for debugging source programs and facilities for creating and using subprograms. The FORTRAN II project was initiated as an attempt to solve the above problems.

The main improvement in the debugging facilities is the incorporation of a general and expandable diagnostic procedure in the translator. The purpose of this procedure is to find and to print a description of every detectable error in a source program. Thus stops in the translator have been eliminated or reduced to a minimum and the ambiguity of the reason for a failure to translate reduced. Routines which are helpful in debugging object programs during execution can be added to the library of FORTRAN II and included in an object program during translation by making use of the improved subprogram facilities.

In FORTRAN I the only available subprograms were the function statement, which allowed a programmer to define a function in one FORTRAN statement, the library tape functions, and the built-in functions. In FORTRAN II the number of built-in functions has been expanded. The expanded FORTRAN source language includes a powerful new function facility which enables a programmer to write library tape functions in the FORTRAN language instead of in the machine language as previously required.

Even greater flexibility has been added to the FORTRAN language by incorporating a provision for the calling and writing of subroutines such as matrix inversion. In fact any FORTRAN program may be written and called as a subroutine. A large program can be written as several subroutines that can be independently compiled

and debugged. This considerably reduces recompile time and allows for easy replacement of sections of the program. Machine language subroutines (e.g., double precision routines which cannot be written in the FORTRAN language) may be added to the FORTRAN library and called by any FORTRAN program.

The FORTRAN II language is compatible "upwards" with the FORTRAN I language, i.e., any FORTRAN I program may be compiled using the FORTRAN II translator with the possible exceptions listed below:

1) Some tables in the FORTRAN II translator have been reduced in size; therefore a table overflow is possible.
2) FORTRAN II object programs in general are somewhat larger than FORTRAN I object programs; therefore it is conceivable that an extremely large FORTRAN I program could exceed memory capacity when compiled by the FORTRAN II translator.

It is assumed in this paper that the reader is familiar with the 704 FORTRAN I language and 704 SAP language.

II—Subprogram Facilities

A. New statements in the language

1. SUBROUTINE NAME (X, Y, I, ...)

This statement, if it appears, must be the first statement in the program. The SUBROUTINE statement defines the entire program to be a closed subroutine, the name of which is NAME (any alpha-numeric characters, not exceeding six, the first of which is alphabetic). The arguments, if any, are to be listed inside a pair of parentheses and separated by commas. An argument name is a non-subscripted variable name (six or fewer alpha-numeric characters,

first of which is alphabetic). Any alpha-numeric variable name which occurs in an executable statement in the subroutine may be listed as an argument of the subroutine.

The program which follows the SUBROUTINE statement may be any FORTRAN II program containing any FORTRAN statements except a SUBROUTINE or FUNCTION statement.

2. FUNCTION NAME (X, Y, I, ...)

This statement occurring as the first statement to a FORTRAN II program defines the program to be a function subprogram. In the case of a function subprogram the name NAME has two distinct meanings.

a) NAME is the name of the subprogram.

b) NAME is the name of the single-valued function being evaluated.

The function name (any alpha-numeric characters, not exceeding six, the first of which is alphabetic), unlike the name of a subroutine, has certain restrictions imposed upon the name.

a) If the function to be evaluated is fixed point, the first character of the name must be I, J, K, L, M, or N.

b) The class of names reserved for the library tape functions (i.e., names having 4 to 7 characters, the last character being F) must not be used as names of function subprograms.

c) Any name appearing in a DIMENSION statement in the function subprogram or in a DIMENSION statement in any program using the subprogram is excluded from use as a name of the function subprogram.

The function to be evaluated, by definition, must have at least one argument. The argument list is identical with that of the subroutine.

As in the case of the subroutine, the program which follows the FUNCTION statement may be any FORTRAN II program containing

any FORTRAN statements except a FUNCTION or SUBROUTINE statement.

3. RETURN

This statement will normally occur in a subroutine or function subprogram. RETURN is a flow statement and causes control to be returned to the calling routine. In the case of a subroutine, normal return of control is to the main routine statement immediately following the CALL statement. For a function subprogram, the RETURN statement will place the value of the function in the accumulator and will return to the arithmetic statement in which the function name appears.

4. CALL NAME (X,Y,I,....)

The CALL statement consists of a name (six or less alpha-numeric characters, the first of which is alphabetic) and may be followed by a string of arguments (as defined below) enclosed in parentheses. The statement is construed to be a call-in of the subroutine named, and a calling sequence is set up transferring control to that subroutine and presenting it with the arguments, if any, which follow the name.

The order of the arguments is taken from the list that appears in the CALL statement, reading from left to right. There must be agreement in number, order, and mode in the argument list here and the argument list in the SUBROUTINE statement if the subroutine was compiled by FORTRAN. Control is returned from the subroutine to the statement immediately following the CALL statement (normal sequencing).

An argument in a subroutine calling statement must be one of the following:

a) any FORTRAN II arithmetic expression such as,

X ** 2 + SINF (X).

b) the name of an array, without subscripts

c) and argument of the following form: , nHx $_1$ x_2

----------x_n,

The interpretation of c) is identically the interpretation of a Hollerith field in a FORMAT statement. This is not the name of a variable, but, as with constants, is itself the data under consideration. It will be stored as follows: the "nH character string" (where n is equal to the number of consecutive Hollerith characters in the 'string' immediately following the 'H') is dropped, the first character (x_1) and the remaining characters, including blanks, are stored as successive characters (six to a word) in successive words, and if the last word is not full, it is filled out with blanks.

5. FUNCTION EVALUATION

When the name of a function subprogram appears in any FORTRAN arithmetic expression, it is construed to be a call-in of the subprogram to evaluate the named function. A calling sequence is set up to transfer control to the subprogram and to present it with the arguments which follow the name of the function. As in the case of a subroutine, an argument may be any FORTRAN II expression, the unsubscripted name of an array, and/or a string of Hollerith characters.

6. COMMON A, B, ...

In FORTRAN II, data storage has been moved down in memory so that a gap no longer exists between program and data. The arrangement of such data is in the normal manner. However, data can be located at the top of memory by listing such data in a COMMON statement. The ordering of storage will be that of the COMMON list, except as it is modified by EQUIVALENCE statements.

The chief purpose of the COMMON statement is to permit data communication between routines which are independently compiled.

7. END (I_1, I_2, I_3, I_4, I_5)

The FORTRAN translator interprets the END statement as an end-of-file condition, thereby permitting the stacking of FORTRAN source programs when compiling. It was deemed advantageous to have programable sense-switch control when batch compiling; hence, the list of I's following the END are interpreted as settings of Sense Switches 1 through 5 on the 704 console as follows:

I = 0 means UP position.

I = 1 means DOWN position.

I = 2 means interrogate the console sense switch.

B. Examples

1. FUNCTION SUBPROGRAM

The following program is an example of a function subprogram.

```
      FUNCTION SUM ( Y, Z, N)
      DIMENSION Y (20), Z (20)
      SUM = 0.0
      DO 5 I = 1, N
5     SUM = Y (I)* Z (I) + SUM
      RETURN
      END (2, 2, 2, 2, 2)
```

CLA NAME is always compiled prior to the return. In the above program the CLA SUM is not necessary; however, in many problems the CLA is required since the value of the function must be left in the accumulator. The arguments Y, Z, and N which appear in the FUNCTION statements are dummy variables. The dummy variables in any subprogram, either function

or subroutine, are assigned no memory locations regardless of the DIMENSION statement since they function solely as name forms within the subroutine. A subscripted dummy variable must be listed in a DIMENSION statement or the translator will interpret the subscripted variable name as a function name. In the case of two or three dimensional arrays, the DIMENSION statement is used by the translator to compile the instructions which compute the absolute memory location of a reference to a subscripted variable. It is necessary to assure compatibility in data storage between the subprogram which uses the data and the calling routine which provides data. Therefore, the dimension information for the two or three dimensional dummy variables must be the same as the dimension information for those actual variables which the dummy variables represent.

2. SORTING SUBROUTINE

The following program is an example of a sorting subroutine (statements 1-5) followed by a skeleton main routine (statements 10-20).

```
1       SUBROUTINE SORT (A, N)
        DIMENSION A (150)
        DO 4 J = 2, N
        SMALL = A(J - 1)
        DO 4 I = J, N
        IF (A (I) - SMALL) 3, 4, 4
3       SMALL = A (I)
        A (I) = A (J -1)
        A (J - 1) = SMALL
4       CONTINUE
        RETURN
5       END (2, 2, 2, 2, 2, )
10      DIMENSION ALPHA (150), BETA (150)
```

```
                    CALL SORT ( ALPHA, M )

                    CALL SORT ( BETA, K )

20                  END ( 2, 2, 2, 2, 2 )
```

These are two distinct programs which give rise to two separate and complete compilations, producing a relocatable binary deck for the subroutine and a relocatable binary deck for the main routine.

3. The following skeleton routine calls for a subroutine which compares Hollerith data.

```
                    CALL FILE ( IHA, IHB, IHC )

                    END ( 2, 2, 2, 2, 2, )
                    SUBROUTINE FILE (X, Y, Z )

                    CALL RECORD (D)
                    IF (D-X) 1, 10, 1
1                   IF (D-Y) 2, 20, 2
2                   IF (D-Z) 3, 30, 3
3                   PAUSE
                    RETURN
10                  CALL PROGI (D)
                    RETURN
20                  CALL PROG 2 (D)
```

```
      RETURN
30    CALL PROG 3 (D)
      RETURN
      END ( 2, 2, 2, 2, 2 )
```

A subroutine may refer to as many other subroutines as desired and at any depth. In the last example the subroutine FILE refers to other subroutines namely RECORD, PROG 1, PROG 2, and PROG 3. In this example, the routine RECORD is assumed to be a SAP coded program which reads an identifying character from a tape file into a location D in such a way that decisions can be made to identify and process the file of data. Notice that SAP coded programs can now be combined with FORTRAN programs in a convenient manner.

III—Subprogram Linkage and the BSS Loader

All routines produced by FORTRAN II, both master routines and subprograms, are loadable by the Binary Symbolic Subroutine Loader. This BSS Loader enables assembled programs in relocatable binary to retain symbolic references to subprograms at lower levels. As a result a routine and all its subprograms can be compiled or assembled independently. At execution time the relocatable binary decks of the main routine and all subprograms (all starting at relocatable 0) may be stacked in the card reader in any order, loaded by the BSS Loader, and run.

The BSS Loader will load absolute and relocatable binary cards and accept the usual control cards plus the program card which will give the symbolic name of the routine and various other information. The first n words of a routine in relocatable binary form which makes symbolic reference to n other routines will contain the BCD symbolic names of the routines referred to. These n

words are called the transfer vector. Reference to a routine B is made from routine A by transferring to that word of routine A which contains the BCD name of the routine B. The BSS Loader will replace the symbolic name of B within the routine A by a transfer to the proper entry point in B after B has been assigned a location. Since FORTRAN will compile routines which make symbolic reference to other routines, each routine in a program may be compiled or recompiled independently of the others. In fact, routines may be compiled that refer to subroutines yet to be written. The construction of a FORTRAN subroutine that uses, say, the sine function and a FORTRAN subroutine called by the statement CALL NAME (A, B, C) can be outlined as follows:

```
SINbbb      BCD     SINbbb          transfer vector
NAMEb       BCD     NAMEb
entry point SXD     $ , 1
            SXD     $+1,2           prologue for
            SXD     $+2,4           this subroutine
            CLA     1, 4
            STA
            etc.
            CLA
function    TSX     SINbbb,4
            STO

FORTRAN
subroutine  TSX     NAMEb, 4
            TSX     A
            TSX     B
            TSX     C
            LXD     $ , 1
return      LXD     $+1,2
```

```
LXD     $+2,4
TRA     n+1,4       where this subroutine
                    has n arguments
```

The operation of the loader will cause the location of the transfer vector, and the number of the entries in it, to be placed in the symbol table. When the program control cards for the routines SIN and NAME are encountered, their names and absolute entry points will be placed in the symbol table. When a transfer card is encountered, the "second pass" will begin, and the two BCD words in the example will be replaced by trap transfers to the appropriate locations.

IV–DIAGNOSTIC

The Section I diagnostic program in the FORTRAN I translator has been left intact in the FORTRAN II translator with the addition of detecting punctuation errors. An additional diagnostic program for detecting source level programming errors has been added to Section I prime. This new diagnostic checks for duplicate statement numbers, transfers to non-executable or non-existent statements, parts of the program which cannot be reached, a non-executable or a conditional transfer statement as the last statement within the range of a DO, and a DO statement at α followed by a non-executable statement at $\alpha + 1$.

A general diagnostic program has also been incorporated in the FORTRAN II translator. If a machine error or a programming error (in Sections 2 through 6) is detected, the translator transfers control to the general diagnostic which prints information as to the Section of the translator in which the stop occurred, the record number of the system tape whose contents were in memory at the time of the stop, the reason for the stop, and instructions to the machine operator. The general diagnostic program was designed not only as an aid to programmers in debugging their

programs, but also as an aid to the customer engineer in case of machine trouble.

V—Changes in the FORTRAN Translator

Major revisions were made in Sections 1 and 6. A new Section Pre-6 was added. Section 1 was modified to classify the new statements, compile the instructions for RETURN, CALL, and function evaluation. Section Pre-6 compiles the prologue for the functions and subroutine subprograms. Section 6 was modified to handle the COMMON statement and the new arrangement of data storage.

Section 1 changes were programmed by Mr. P. B. Sheridan and Mr. L. M. May. Changes to the loader, diagnostic editor and the diagnostic programs were programmed by Miss B.A. Brady. Section Pre-6 and modifications to Section 6 were programmed by the author.

References

1. Reference Manual FORTRAN II for the IBM 704 Data Processing System, International Business Machines.

2. Programmer's Primer for FORTRAN Automatic Coding System for the IBM 704, International Business Machines.

3. J. W. Backus, et al, "The FORTRAN Automatic Coding System", Proceedings of the Western Joint Computer Conference (February, 1957).