UNIVERSITY OF EDINBURGH
Edinburgh Regional Computing Centre

User Note 57

(December 1984)

Title:

Moving to Fortran 77 from Earlier Dialects of Fortran

Author: Roderick McLeod

Contact: Advisory service

Software Support Category: A

Synopsis

This note describes the main differences between Fortran 77 and earlier dialects of Fortran. It is intended for users who are already conversant with Fortran programming.

Keywords

EMAS-3, FORTE, Fortran, Fortran 77, Programming Languages.

Edinburgh Regional Computing Centre

James Clerk Maxwell Building, The King's Buildings, Mayfield Road, Edinburgh, EH9 3JZ. Telephone 031-667 1081

Introduction

Fortran 77 has been available to users of EMAS 2900 for several years and many users have converted programs written in earlier dialects of Fortran to use it. Further, many new programs have been written in Fortran 77. However, the FORTE compiler has remained available for users who have for a variety of reasons preferred to remain with the older standard. If you are still using FORTE you are encouraged to convert to Fortran 77 as soon as possible in order to:

- take advantage of its new facilities
- simplify the transfer of your work to EMAS-3 and to other systems
- and make use of its optimising facility which offers a significant improvement in runtime performance.

There is a full description of Fortran 77 in the Language Manual (Ref. 1), and the chapter and section numbers given in this note refer to the same manual.

If you are writing new programs or making major alterations to existing programs you should, wherever possible, try to write code which conforms with the new standard. If, however, you are using large, perhaps imported, and fairly static programs then you may choose to make as few changes as possible. The Edinburgh Fortran 77 compiler whilst supporting the latest International standard for Fortran (ANSI X3.9-1978 and ISO 1539-1980) also accepts almost all the features of FORTE. Thus the transfer of programs in this category should be straightforward. User Note 45 will provide further details of how to use Fortran 77 on EMAS 2900.

Character Handling

An important feature of Fortran 77 is an extension to the type CHARACTER and its associated operators and functions. This extension enables you to manipulate character information without the contorted coding which was needed in earlier versions of the language. Whilst the new features do not make Fortran the ideal language for text manipulation applications they do represent a useful and flexible set of facilities which should more than satisfy the character handling requirements of most Fortran programs.

Apart from the facility to declare character variables and arrays, and to perform assignments, comparisons, and data initialisations, Fortran 77 also provides:

- a syntax to define a sub-string within a string (see 4.1.5)
- a new concatenation operator "//" to join two character elements (see 5.2.2).
- a set of functions LGE, LGT, LLE and LLT to compare, in a program portable manner, two strings to determine their lexical (dictionary) order (see Appendix 1)
- an integer function, INDEX, to locate a sub-string within a string (see Appendix 1).

Additional Control Structures

IF <condition> THEN, END IF, ELSE, and ELSE IF are new control statements which can be used to improve the structure and legibility of your programs. They reduce the need for numeric labels, and hence the opportunities for introducing errors when modifying existing code (see 7.2.3).

Arrays

- The restriction that arrays must have a lower bound of one has been removed (see 4.2.2).

- you can use simple integer expressions for array bounds in declarations (see 4.2.2).

- asterisk can be used as the upper bound for the only or final dimension in a dummy array, when the subprogram is likely to be passed arrays of different sizes (see 4.2.2). Alternatively, adjustable dummy arrays can be used in this situation (see 8.3.3.1).

- you can use arbitrary expressions as array subscripts. However, with default checking, each subscript must be within the declared bounds for a particular dimension (see 3.2.4.1).

- when you refer to an array in an EQUIVALENCE statement you must use the same number of dimensions as you used to declare it. A special compiler option relaxes this rule to assist with importing programs.

DO loops

- DO loops are described in Section 7.3

- the DO-variable for a DO loop can be integer, real or double precision. The last two should be used with care (see 7.3.1).

- the parameters can be positive, negative or zero on entry, apart from the incrementation parameter which must not be zero.

- in FORTE the code inside a DO loop was always executed once. Fortran 77 introduces the zero-trip loop. For example:

```
DO 10 I=m1,m2,m3
```

  The code within the DO loop would not be executed at all if m1 > m2 and m3 > 0. Nor would it be executed when m1 < m2 and m3 < 0.

- the extended range feature is no longer allowed. This means that if you leave a DO loop you cannot re-enter it other than at the DO statement. If you CALL a subroutine or make a function reference, from within the DO loop, then you are allowed to RETURN directly back into the DO loop. If the RETURN is to a label outwith the DO loop you cannot subsequently jump back into it. A special compiler option will allow you to use extended range to simplify the task of importing programs.

Implied-DO loops in DATA statements

You can now use an implied-DO loop to initialise items in a DATA statement. This could be useful for initialising selected elements of an array (see 4.3.1.2). For example:

```
      INTEGER TABLE(100)
      DATA (TABLE(I), I=1,99,2) /50*1/ (TABLE(I), I=2,100,2) /50*2/
C Set odd numbered elements of TABLE to 1 and even ones to 2
```

Type Specifications

The only valid type specifications in standard Fortran 77 are INTEGER, REAL, LOGICAL, DOUBLE PRECISION, COMPLEX and CHARACTER*s. The Edinburgh compiler will continue to allow the use of a length specifier, for example INTEGER*4.

A special compiler option will allow you to use data initialisation within type statements. This is not allowed in standard Fortran 77 but has been provided to help you when importing programs.

File Control

OPEN, CLOSE and INQUIRE are new statements which allow you to open and close files dynamically, and to inspect the status of your files (see 10.7). There is a new facility which allows your program to check the outcome of an I/O operation (see 10.3).

NAMELIST, PUNCH, FIND and DEFINE FILE no longer exist.

READ, WRITE and PRINT

You may continue to use short forms of READ and PRINT when accessing the primary input and output channels. For example:

```
      READ 100,NGOES
      PRINT 200,I,J,K
```

You can also include character constants and expressions in output lists:

```
      WRITE(6,110) 'Number of attempts=', NGOES-1
```

The method of specifying the record number in READ and WRITE statements has been changed. For example:

```
      READ (20'IT*4)IS,IT,IU
```

becomes

```
      READ (20,REC=IT*4)IS,IT,IU
```

Formatted Input/Output

Fortran 77 requires that format codes must have the same type as the corresponding item in the input/output list. Literals can no longer be included in format statements which are used with READ statements.

There are several new FORMAT codes, including:

- BN and BZ which are used to control the interpretation of spaces embedded within numeric fields (see 9.3.1.12).

- Ew.dEe may be used for floating point output if you wish to specify the number of digits in the exponent field (see 9.3.1.3).

- Iw.d can be used when you wish to print leading zeros. It can also be used to leave a blank, rather than printing '0', when the field has a value of zero (see 9.3.1.1).

- T, TL an TR are used to specify the location of the next character to be read or written within the current record (see 9.3.1.10).

Order of Statements

Fortran 77 is more restrictive than earlier Fortrans in defining the order in which statements must appear. There is a useful diagram in Section 2.3.5 which tabulates the rules.

Comments

- can begin with 'C' or '*' in column 1, or be completely blank;

- can precede continuation lines.

Constants

There is a change to the way in which double precision constants are evaluated. In Fortran 77 you must specify that a constant is to be evaluated using double precision by using the form 1.88D3. In the case of FORTE double precision was assumed where indicated by the context, e.g. assignment to a DOUBLE PRECISION variable. In the following example:

```
DOUBLE PRECISION A,B,C
A=0.11111
B=2.6E2
C=1.88D4
```

FORTE will assign a double precision constant to A, B and C. Fortran 77 will only do so for C.

The compiler prints a comment if you attempt to specify a number with more precision than can be stored, e.g if you attempt to multiply a number by 1.23456789.

A special compiler option will allow the use of hexadecimal constants: these should be written in the form X'1A00'.

PARAMETER statement

You can use the PARAMETER statement to define symbolic constants. These can be used in place of constants almost anywhere in the program unit (see 3.3.4). For example:

```
PARAMETER(INPUT=1,VRATE=0.15)
.
.
READ(INPUT,100)GOODS
VAT=GOODS*VRATE
.
.
```

The use of constants in this way simplifies program maintenance, and avoids the risk of a value being altered, since the compiler will not allow you to assign a value to such a constant in executable code.

Subprograms

- You cannot pass parameters using reference by location using the form /PARAM/.

- The extended RETURN facility still exists but the CALL statement must use '*' instead of '&' in front of each statement label (see 8.2.2.2). A special compiler option will allow you to continue to use '&', to assist with importing old programs.

Generic Functions

Most of the intrinsic functions now have generic names which should be used in preference to the specific names. For example, for the square root function you should always use the name SQRT, not DSQRT nor CSQRT. The compiler will work out from the context which entry to use.

Language Extensions

Chapter 11 describes a number of extensions allowed by the Edinburgh compiler beyond the strict Fortran 77 standard. Any use of these extensions will cause the compiler to generate warning messages. Such messages can be suppressed by use of PARM(NOWARNINGS). Alternatively PARM(STRICT) can be used if you want the compiler to fault any non-standard usage.

Further extensions are planned to assist users importing programs and packages which have been developed on the VAX-11 compiler.

REFERENCE

1. 'Edinburgh FORTRAN77 Language Manual'.
Edited by N. Hamilton-Smith. Edinburgh Regional Computing Centre.
February 1981.

If you are interested in the wider implications of Fortran 77 portability then you should read:

'Fortran 77 Portability' J. Larmouth. Software-Practice and Experience, Vol. 11, 1071-1117 (1981).

You can find a full list of conflicts between Fortran 77 and the earlier standard, on which FORTE was based, in Appendix A of:

ANS FORTRAN X3.9 - 1978, American National Standards Institute, New York, 1978.