



Fortran95 for Fortran77 Programmers

Ian Bush

Computational Science and Engineering Department,
CLRC Daresbury Laboratory, Warrington, Cheshire, WA4 4AD.

Revision: 1.0

Abstract

Fortran 95 is the current international standard for the Fortran programming language. In itself it is a small revision of the previous Fortran 90 standard, which was a large extension of FORTRAN 77.

This is a Technical Note of the UKHEC Collaboration.

Report available from <http://www.ukhec.ac.uk/publications/notes/fortran95.pdf>

© UKHEC 2002.

Neither the UKHEC Collaboration nor its members separately accept any responsibility for loss or damage arising from the use of information contained in any of their reports or in any communication about their tests or investigations.

On moving from 77 to Fortran 95 the first thing is not to worry! The VAST majority of standard conforming FORTRAN 77 programs are also Fortran 95 programs. One of the prime concerns of the standards committee was to ensure backward compatibility, and as such very little has been removed from the language. What has been removed is generally rarely used and poor practice e.g. The ASSIGN statement, real variables for do loops (If you don't know what these are please don't find out, if you do, please forget them!). In fact the main cause for incompatibility is if the name of a variable or constant clashes with the name of one the new intrinsic subprograms, and even this, in my experience, is rare.

So why should you wish to move to Fortran 95? Well apart from it being the new standard, and apart from it being a modern, powerful language which allows you to express your programs in a much clearer way, and apart from the language itself providing facilities to aid checking the correctness of your code, some specific examples are described in the following sections.

Symbol length

Symbols may be up to 32 characters long, allowing such things as variable names to be much clearer (the old method of cutting out the vowels and truncating to six characters is no longer needed). The underscore is also permitted.

Source form

Fortran 95 has a new, much more flexible source form. You are no longer stuck in the 72-column straitjacket which, when coupled with the extended symbol length above, again improves code clarity.

Implicit none

Use it. NOW!

Parameterised data types and constants

These allow you to portably select the precision for your whole code. For instance in Fortran 77, a Real on many machines occupied 32 bits, while on Cray systems it was 64 bits, which often caused problems. The use of parameterised data types avoids this entirely.

An extended range of control constructs

This not only includes such statements as Do While and Select Case, but also Cycle which allows you to skip to the next iteration of a do loop, and Exit to leave a loop entirely. Further, the Forall and Where constructs facilitate parallel programming.

Array syntax

Array syntax allows operations on a whole array in one statement. For instance if a, b and c are arrays of the same size

a = 0.0 will set the whole of a to zero

a = 2.0*a will double each element of a

$a = b+c$ will set element i of a to $b(i)+c(i)$

and you can even perform complex expressions such as

$a = \text{sqrt}(b*b+c*c)$

It is also possible to select subsections of arrays, e.g.

$a(3:9) = 2.0*b(1:7)$ will set $a(3)$ to $2.0*b(1)$, $a(4)$ to $2.0*b(2)$ etc.

and even to perform strided operations

$a(1:7:2) = 0.0$ will set elements 1, 3, 5 and 7 of a to zero.

Dynamic memory allocation

Allocatable arrays allow the program to request the appropriate amount of memory for that array for the current run. This has two advantages

- No longer do you have to adjust dozens of parameters for a new run of a different size.
- If a run does not need one of the arrays you need not allocate it, potentially allowing larger sized jobs for a given amount of memory.

Interface block

If you know C these are similar to, but more powerful than, prototypes. An interface block tells the calling routine what the correct arguments are for a routine it wants to call. Apart from allowing the compiler to recognise when there is an argument mismatch and so alert the programmer to an error, this is also the basis for some of the more powerful facilities of the language:

Through use of the Intent facility it is possible to tell the compiler how the variable will be used in the subprogram allowing extra optimisation.

In Fortran 77 it was usually necessary to have to pass not only an array but also its dimensions. Interface blocks make this latter argument unnecessary for it is possible to use some of the new intrinsic functions to find out the size of the array in the called routine.

Interface blocks are how the code implements overloading, and even allows you to create your own operators.

A number of other features, such as Optional arguments and keyword argument lists have also been added.

Modules

There is not enough space here to explain how wonderful modules are! Basically they allow you to encapsulate data and the routines that act upon that data thus helping code organisation and clarity. In fact in their simplest form they can even be a glorified common block. However they reach their full glory when combined with data hiding using Private and Public, interface blocks and, if required, derived types (see below). They allow an implementer to provide an interface (e.g. a set of routine

calls) for a given operation to the rest of the program whilst entirely hiding how that operation is carried out. This may not sound very much, but consider:

- You can avoid entirely the possibility of name clashes e.g. between a local variable and a common block
- Say a new and better algorithm for the operation is developed. Modules greatly facilitate the incorporation of this into the code by simply providing a new module with the same interface and NO other changes.

To realise how good modules are you really have to use them!

Derived data types

(Struct for C programmers) Derived data types allow you to create your own data types that are more convenient or natural for your code. Further the language has facilities, as mentioned above, to define operations on those types, which when combined with modules allows some extremely elegant code.

Intrinsics

Fortran 95 has a greatly extended range of intrinsic subprograms. For instance these include portable timers (System_clock, Cpu_time, Date_and_Time), a random number generator, improved character handling facilities, matrix multiply, dot product, bit handling routines, reduction routines (e.g. Maxval) and various inquiry functions to support the new features of the language (e.g. Allocated to check whether an allocatable array is allocated).

Other features

A number of other features, e.g. Improved I/O facilities, more edit descriptors, pointers (but they are quite unlike C pointers), recursive subprograms, automatic arrays, array valued functions and zero sized arrays (yes, they do have their uses)

Conclusions

All in all, Fortran 95, when compared to Fortran 77, greatly helps code clarity, maintainability and portability whilst being almost entirely backward compatible. What are you waiting for?

For more information a good place to start is

<http://www.kcl.ac.uk/kis/support/cit//fortran/f90home.html>

This site has links detailing not only the language but also what books, on line tutorials, utilities and compilers are available.

A comprehensive tutorial offered by the University of Liverpool is located at

<http://www.liv.ac.uk/HPC/HTMLFrontPageF90.html>

As for books, I personally like Metcalf and Reid, Fortran 90/95 Explained, Oxford University Press, 2nd edition, 1999, ISBN 0-19-850558-2 though some people consider it best as a reference manual. I have also heard good reports about Kerrigan, Migrating to Fortran 90, O'Reilly and Associates, 2nd ed. Sept.94, ISBN 1-56592-049-X

There is also the news group `comp.lang.fortran`, which is a forum with a good signal to noise ratio and the regulars include a number of people from the standards committee. You might even get a reply from me!