The Fortran Story Retold

Selected Reprints 1968-2011

Compiled by Loren Meissner, 2016

CONTENTS

As computers doubled in power every few years, and became able to perform calculations on many numbers simultaneously, by the use of processors running in parallel, and as the problems to be solved became ever more complex, the question arose as to whether Fortran 77 was still adequate. (And there were a large number of user requests left over that it had not been possible to include in it.) Programs of over a million lines became commonplace, and managing their complexity and having the means to write them reliably and understandably – so that they produce correct results and could later be modified – were desperately required.

Thus began the battle over Fortran 90 [and a change to lower case spelling]. Fortran had been attacked by computer scientists on two grounds. One was because of its positively dangerous aspects, for instance the lack of any inherent protection against overwriting the contents of memory in the computer, including the program instructions themselves! The other was its lack of indispensible language features, such as the ability to control the logical flow through a program in a clearly structured manner. On the other hand, Fortran had always been a relatively easy language to learn and that, combined with its emphasis on efficient, high-speed processing, had kept it attractive to many busy scientists. Thus, the standards committees were faced with the almost impossible task of modernising the language and making it safer to use, whilst at the same time keeping it ‘Fortran-like’ and efficient. Fortran 90 was the answer.

There were other justifications for continuing to revise the definition of the language. As well as standardizing vendor extensions, there was a need to respond to the developments in language design that had been exploited in other languages, such as APL, Algol 68, Pascal, Ada, C and C++. Here, X3J3 could draw on the obvious benefits of concepts like data hiding. In the same vein was the need to begin to provide an alternative to dangerous storage association, to

abolish the rigidity of the outmoded source form, and to improve further on the regularity of the language, as well as to increase the safety of programming in the language and to tighten the conformance requirements. To preserve the vast investment in Fortran 77 codes, the whole of Fortran 77 was retained as a subset. However, unlike the previous standard, which resulted almost entirely from an effort to standardize *existing practices*, the Fortran 90 standard was much more a *development* of the language, introducing features that were new to Fortran, although based on experience in other languages. This tactic, in fact, proved to be highly controversial, both within the committee and with the wider community. Vested interests got in on the act, determined, depending on their persuasion, and in particular on whether they were users or vendors, either to extend Fortran to cope better with new computers and new problem domains or to stop the whole process in its tracks. The technical and political infighting reached legendary proportions. It was not until 1991, after much vigorous debate and thirteen years' work, that Fortran 90 was finally published by ISO.

It introduced a new notation that allows arrays of numbers, for instance matrices, to be handled in a natural and clear way, and added many new built-in facilities for manipulating such arrays, for example, to add together all the numbers in an array, a single command (`sum`) is all that is required. The use of the array-handling facilities made scientific programming simpler, less error prone and, on the most powerful computers whose hardware can handle vectors of numbers, potentially more efficient than ever.

To make programs more reliable, the language introduced a wealth of features designed to catch programming errors during the early phase of compilation, when they can be quickly and cheaply corrected. These features included new ways of structuring programs and the ability to ensure that the components of a program, the subprograms, 'fit together' properly. For instance, Fortran 90 makes it simple to ensure that an argument mismatch can never arise as it enables programmers to construct verifiable interfaces between subprograms.

In summary, the main features of Fortran 90 were, first and foremost, the array language and data abstraction. The former is built on whole array operations and assignments, array sections, intrinsic procedures for arrays, and dynamic storage. It was designed with optimization in mind. The latter is built on modules and module procedures, derived data types, operator overloading and generic interfaces, together with pointers. Also important were the new facilities for numerical computation, including a set of numeric inquiry functions, the parameterization of the intrinsic types, new control constructs – `select`

`case` and new forms of `do`, internal and recursive procedures and optional and keyword arguments, improved I/O facilities, and many new intrinsic procedures. Last but not least were the new free source form, an improved style of attribute-oriented specifications, the `implicit none` statement, and a mechanism for identifying redundant features for subsequent removal from the language. The requirement on compilers to be able to identify syntax extensions, and to report why a program had been rejected, was also significant. The resulting language was not only a far more powerful tool than its predecessor, but a safer and more reliable one too. Storage association, with its attendant dangers, was not abolished, but rendered unnecessary. Indeed, experience showed that compilers detected errors far more frequently than before, resulting in a faster development cycle. The array syntax and recursion also allowed quite compact code to be written, a further aid to safe programming. Fortran 90 also allowed programmers to tailor data types to their exact needs. Another advance was the language's new ability to structure program data into arbitrarily complex patterns – lists, graphs, trees, etc. – and to manipulate these structures conveniently. This is achieved through the use of pointers. A related feature was the ability to allocate storage for program data dynamically.

After this revision, Fortran became, it must be admitted, a different language, as the entire issue of the journal [Computer Standards & Interfaces, Vol. 18 (1996)], which is devoted to various aspects of the development of Fortran 90, shows.