Scientific Programming 19 (2011) 253–258
DOI 10.3233/SPR-2011-0330
IOS Press



# Book Review

Dan Nagle
*E-mail: danlnagle@me.com*

**Modern Fortran Explained**, by Michael Metcalf, John Reid and Malcolm Cohen, 488 pp., Oxford University Press, 2011, ISBN: 978-0-19-960142-4.

In the interests of full disclosure, I should explain that I know the authors, I work with two of them on International Standards committees, and I respect and like them all.

*Modern Fortran Explained* is the latest in a long line of books, going back to when what became Fortran 90 was a work in progress in the 1980s. New editions have been published for every revision of the Fortran standard since, to the point where the authors have dropped the year indication from the title. If the year had not been dropped from the title, this book might be *Fortran 95/2003/2008 Explained*. Remnants of this legacy are found in the overall structure of the book, in that it discusses Fortran 95 first, then adds a Fortran 2003 layer, and lastly a Fortran 2008 layer. Forward pointers to changed items, largely lifted restrictions, are found where needed.

*Modern Fortran Explained* has 488 pages, and includes a Preface, 20 Chapters, 7 Appendices and an Index. In the preface, we are told that Fortran is a principal language for science and engineering calculations and that the authors intend to fully describe modern Fortran. Fortran certainly has been around for a while. Since the Fortran 2003 standard was published, Fortran celebrated its 50th anniversary. After originating with IBM in the late '50s, what I call Archaic Fortran was the first standardized programming language with Fortran 66. Modern Fortran begins, I reckon, with Fortran 90.

While old-timers might be astounded to see what has become of a language they perhaps last used on a CDC 6600, what does Fortran offer a younger programmer? I can think of a number of items: first, a very high degree of portability, without disciplining oneself to a subset of the language or low-level tricks with conditional compilation. This requires understanding the numeric requirements of your programming task and using the kind mechanism as it was intended. Next, the availability of first-class arrays, together with the well-designed array syntax, allows a concise statement of many numerical and scientific problems. Additionally, Fortran has substantial libraries for evaluation of mathematical functions, bit manipulation, string manipulation, and also in support of the array features. Many libraries beyond the standard-specified ones are well-trusted due to long years of service, some free and some commercial. And starting with Fortran 2008, there is a clear and concise, easily understood and highly scalable parallelism in the form of the coarray features, which is based on the coarray features Cray has supported for over a decade now. The parallel programming model is intuitive, and the implementation avoids clutter. One needn't forego full support for object-oriented programming to obtain these advantages, either. With 20 Chapters to cover, we had best get started.

Chapter One "Whence Fortran" gives an overview of the history of the standards process that has produced modern Fortran. We read an overview of the structure of the standards committee hierarchy who write the Fortran standards. An international hierarchy of committees is responsible for programming languages (not that every programming language is specified by an International Standard!), with a more-or-less parallel committee hierarchy in each participating country. The main new features introduced with each revision of the Fortran standard are described, up to the production of the latest, Fortran 2008. The description of the making of Fortran 2008 is by far the shortest, surprisingly. But then, one of the authors has retired from the committees, and the other two were on opposite sides of the Coarray Wars, so it's likely best all around to keep that description brief. (Note the indication of the congenial atmosphere of the Fortran committees.) In short, Fortran has emerged from its long history as a fully modern programming language, easily capable of producing highly portable programs via the kind mechanism, with full language support for modern programming techniques (encapsulation, overloading, inheritance, polymorphism), with versa-

tile and expressive array features, and with special emphasis given to the concise and clear expression of highly scalable parallelism.

Chapter Two "Language Elements" starts with the character set, first that of Fortran 95 and later as expanded by Fortran 2003. It moves on to tokens and language source form (the modern free form rather than the archaic card-image fixed form). The next idea is the concept of type, leading to the form of the literals for the intrinsic types. Next, we encounter names. So we can now declare scalar variables of the intrinsic types. Defining derived types comes next, and how to declare variables of whatever derived types we have defined. Having scalar variables, arrays of intrinsic types cannot be far behind. This leads us to the concept of storage order. Hence, we move to character variables and the substring notation. With arrays, and derived types containing arrays, a discussion of objects and subobjects is in order. A discussion of pointers concludes this chapter.

Chapter Three "Expressions and Assignments" comes next. The basic idea is that expressions are evaluated according to the operands and the connecting operators without regard to context, and only then converted as needed for assignment. We start with scalar numeric expressions, including the exponentiation operator, and the results of mixing operand types. The usual truncation of integer division is fully described. Here we find full tables of the promotion of operands of mixed-mode operations. With assignment, we encounter defined and undefined variables. Extending arithmetic operators, we find the relational operators giving logical results. These, in turn, lead to logical operators and their expressions. Extending the discussion to character type, we read of character assignment and the concatenation operator. Having the literals for intrinsic types, we next find the derived type constructors. As soon as variables of derived types have values, we want to combine them using programmer-defined operators. From defined operators, defined assignment is not far away. At this point, we are ready for array expressions and assignments. How pointers are used to build data structures and used in expressions is explained before nullifying a pointer concludes this chapter.

Chapter Four "Control Constructs" allows us to modify the execution sequences seen in the code snippets encountered so far. The choice constructs include the if construct (in Fortran spelled `if(...) then ... end if`) and the case construct (in Fortran spelled `select case(...) case(...)... end select`). The iterative construct is the multi-formed do construct (spelled `do ... end do`). The flexibility given by the exit statement is shown. The go to statement must be mentioned, with the usual admonition on its misuse. This concludes the chapter.

Chapter Five "Program Units and Procedures" shows us how to combine our control constructs into complete programs. The first program unit, of course, is the (main) program, which may be named. A program may contain internal procedures. If a program is to cease execution at more than one place, a `stop` statement is useful. External subprograms, external subroutines and external functions, can also contain internal procedures. Modules provide a means of sharing names, and their attendant attributes (variable, array, type, procedure and so on), among program units. Modules can also contain module procedures which may also be shared. Procedures may, of course, have arguments, and this leads to the concept of the procedure's interface. Fortran does not require pass-by-reference, but rather has a set of rules governing argument association, and these are well described. To return from a procedure at more than one place, a `return` statement is useful. Among the argument attributes are the intents, which restrict how an argument may be used within the procedure. The restrictions on what side effects are allowed to function references appearing in expressions are described. An interface may be explicit or implicit. Procedures may be passed as arguments, a classic example being the function to be minimized being passed to the procedure to do the minimization. Arguments to procedures may use keywords or be optional. With the simple scope of labels, we encounter scope, which leads to a full discussion of scope in general. Next comes recursion, both direct and indirect. With procedure interfaces, we can define generic sets of procedures. Assumed length characters follow. Finally, descriptions of the `function` and `subroutine` statements completing the chapter.

Chapter Six "Array Features" describes one of Fortran's most useful features, its array handling capability. One key here is to treat zero-sized arrays as a useful end-case for array handling generally. Next, we see how to declare an array argument so as to automatically pass its shape. This leads to automatic objects. Another possibility for arrays of variable size is the allocatable array, which is described next. How allocatable arrays are treated as arguments follows, along with allocatable function results. Of course, a derived type may have an allocatable component. This leads to

a discussion of allocatable objects versus pointer objects. After discussing elemental operations and array-valued functions, we come to masked assignment with the `where` statement and block and the `forall` statement and block. This, in turn, leads to the ideas of pure procedures and elemental procedures. This is followed by a detailed discussion of array elements and the distinctions between array components and arrays of derived types. Alternative data structures may be constructed using pointer components. The Fortran semantics of pointers, as aliases of targets, is examined before moving to array constructors. The use of mask arrays completes this chapter.

Chapter Seven "Specification Statements" covers in full detail the specification of entities in Fortran. Implicit typing is mentioned and fully described, only to be disavowed via the standard `implicit none` statement. First, one may declare constants and use constant expressions. These may give initial values to variables. Initial values for pointers, and default initialization of derived type components is next. We encounter the public and private attributes for module data. The pointer, target and allocatable attribute are followed by the intent and optional attributes for dummy arguments. We next encounter the save attribute (and its relationship to initial values) before turning to the description of the `use` statement with its only and renaming clauses. The effects of these on hierarchies of modules is discussed before turning our attention to an expanded discussion of derived type definitions. The standard `namelist` statement completes this chapter (namelist input/output is covered in the input/output chapter further on).

Chapter Eight "Intrinsic Procedures" show us the set of intrinsic procedures available in Fortran 95 (to be expanded in later revisions of the standard). First, we meet the use of keywords in calling procedures. The intrinsic statement is explained, it is useful when extending the use of an intrinsic to new cases, typically, to handle a derived type. Next, we examine Fortran's extensive library of mathematical and string processing procedures. Before moving to the numeric inquiry procedures, we pause to examine the numeric model, upon which the results are defined. Likewise, the bit manipulation procedures are prefaced by a discussion of the bit model. Having seen the elemental procedures, we move to the transformational procedures which are very useful when reducing or producing arrays. The effects of the dim argument are fully covered. Pointer procedures, clock procedures, and random numbers complete the chapter.

Chapter Nine "Data Transfer" starts by mentioning numeric conversion, input/output lists and formatting. This introduction is followed by descriptions of unit numbers and internal files. Formatted input and output, list directed input/output, and namelist input/output follow in order. More advanced topics include non-advancing input/output before encountering the individual edit descriptors. Unformatted input/output and direct access files follow. The standard sequence of action of a transfer statement completes this chapter.

Chapter Ten "Operations on External Files" begins by stating what cannot be standardized: the allowed names for files, and whether sequential access and direct access may be used with the same external file. The `rewind`, `backspace` and `endfile` statements follow. The `open` (including the changeable modes), `close` and `inquire` statements complete this short chapter.

Chapter Eleven ''Floating-Point Exception Handling'' begins with a description of how this feature came to be in Fortran, the development having missed the deadline for inclusion in Fortran 95, it was issued as a Technical Report and included in Fortran 2003. Floating-point exception handling in Fortran is based on the IEEE 754 standard, so we start with a short review. Access to the features of 754 is via intrinsic modules, which are a (then) new feature to Fortran. The discussion follows the exception-enabling flags, halting modes and rounding modes. Opaque derived types manipulated by procedures accessed from the intrinsic modules is the general scheme. If a processor does not support 754, it simply fails to provide the intrinsic module, and the compilation of a program requiring 754 fails when the module is not found. If a processor supports only portions of 754, inquiry procedures exist and the resulting values may be tested.

Chapter Twelve "Interoperability with C" describes the standard means of Fortran and C cooperation within a single program. The intention is to allow standard-defined access by Fortran programs to the many libraries defined for use by C language programs, mainly operating system services and the like, and to allow the contrary so C programs may use the many libraries written in Fortran, mainly numeric libraries. The key idea is to define a mapping between a C prototype and a Fortran interface. Again, an intrinsic module is the mechanism. The intrinsic module defines kind values for the various C types for a Fortran program to use. Access to C pointers is via a set of intrinsic module procedures. Derived types and variables interoperate via a bind attribute. The C entity must have external

visibility. A value attribute is added to Fortran to enable use of C pass-by-value formal parameters. A limited enumeration facility is added to Fortran for use with the corresponding C feature. Examples are provided. This feature was mostly written by the Fortran committees, but interested outside parties, including the C standards committees, and the Fortran binding group of the MPI Forum, have contributed. (A Further Interoperability with C Technical Specification is in the works as this is written. It should be published shortly.) The liaison among the committees in support of these features is excellent.

Chapter Thirteen "Type Parameters and Procedure Pointers" discusses two rather unrelated topics. Fortran's type system relies upon kind type parameters to distinguish among different kinds of types. For example, various sizes of integers are all of type integer, but with differing kind type parameter values. Starting with Fortran 2003, derived types may be defined in such a way that the kind type parameter values need not be specified until the type is used to declare a variable. Thus, one might define a type to model rational numbers, with integers for the numerator and denominator, and not specify the sizes of the integers until declaring a variable using the rational type. The discussion of procedure pointers starts with the abstract interface, followed by the procedure pointer. Next, we learn how to use a procedure pointer as a component in a derived type definition. A discussion of the pass attribute of dummy arguments completes this chapter.

Chapter Fourteen "Object-Oriented Programming" discusses Fortran features for implementing object-oriented programming. We start with type extension before moving to class declarations of polymorphic entities. The rules for establishing a polymorphic entity's dynamic type are explained. Next comes the concept of the unlimited polymorphic entity. To simplify what might become cumbersome notation, the associate construct allows a shorthand to be used for a lengthy fully-qualified data entity's name. The `select type` construct operates analogously to the `select case` construct, but chooses an execution path based on the dynamic type. Abstract types and deferred bindings are described. Type-bound procedures, and their relationship to constructors and finalization are discussed. The type inquiry functions complete this chapter.

Chapter Fifteen "Establishing and Moving Data", and the two chapters that follow, discuss some of the many minor enhancements and programming conveniences defined in Fortran 2003. These include enhancements to structure constructors, to the allocate statement (involving deferred types and polymorphic variables), allocatable entities, the `move_alloc` intrinsic, better control of access from modules including enhancements to the renaming abilities of the only clause on the use statement, and the allowance of numeric intrinsics to give variables their initial values. Chapter Sixteen "Miscellaneous Enhancements" continues this line with pointer intents for pointer dummy arguments, the volatile attribute, the import statement, an intrinsic module providing definitions of more aspects of the program's environment, and support for internationalization including various character sets. Access is provided to error messages, and one may define public entities of private type. Chapter Seventeen "Input/Output Enhancements" discusses how the programmer may customize input/output for entities of derived type (allowing, for example, transfer of linked lists of an unknown number of elements) and asynchronous input/output transfers including the asynchronous attribute for data to be so transferred. The appearance of exceptional values of IEEE 754 arithmetic in formatted input/output transfers is explained, along with the stream access method. Recursive input/output transfers and the `flush` statement are next. This chapter concludes with discussion of intrinsic module procedures, further specifiers on input/output statements, and namelist enhancements.

Chapter Eighteen "Enhanced Module Facilities" discusses the improvements to modules made between Fortran 2003 and Fortran 2008. The feature, informally called submodules, was not quite ready in time for the release of the Fortran 2003 revision. But it was held to be of such importance that it was released as a Technical Report before Fortran 2008 was completed. In short, a module may have submodules, which inherit entities from the module via host association (as a contained procedure would from its containing procedure). A submodule may, in turn, have further submodules. A most useful part is that the interface to a procedure may reside in the module and the implementation of the procedure may reside in a submodule. Thus, when the implementation is changed but the interface not, no recompilation of program units that use the module is necessary. This avoids needless compilation cascades. The feature is also helpful in simplifying the task of organizing data into modules, as the authors clearly explain. (It also encourages implementors to put one module procedure into one object file, rather than writing the whole module into one large object file.)

Chapter Nineteen "Coarrays" is, to me, the main event in a description of Fortran 2008. Coarrays use

a Partitioned Global Address Space (PGAS) approach to provide a succinct notation for Single Program Multiple Data (SPMD) programming. Coarray Fortran might be compared to Unified Parallel C (UPC) or Titanium (an extension of Java). Briefly, a coarray program is executed by replicating the program some number of times. An entity declared as a coarray can be used to fetch or define its value on the current image (each replication of the process is called an image), of course, or the value on another image. What remains, largely, is to synchronize the images, and to ensure that each image's memory correctly reflects the program state. That is the role of the synchronization statements, and of the argument passing restrictions that are to prevent copy-in/copy-out argument passing. Statements that imply synchronization are called image control statements. The current set of coarray features constitutes a simple, functional, core implementation. Work is ongoing on the Fortran standards committees to add amenities, such as a means to group images (which might be called teams), intrinsic functions that apply across images (the collective functions), and parallel input/output. This work is in its formative stages, but is currently intended to be released as a Technical Specification (the new ISO-speak for what was formally called a Technical Report) prior to any future revision of the Fortran standard.

Chapter Twenty "Other Fortran 2008 Enhancements" is the concluding chapter. It describes several small enhancements intended to ease the programmer's task. Perhaps among the most important is the ability to use an `exit` statement to exit from nearly any construct (exiting was previously restricted to loops), and the requirement for support of 64-bit (or larger) integers. Two new performance enhancements are the concurrent loop control of the do loop, which allows an iteration space to be tiled by a single block structure when the iterations may be executed in any order, and the contiguous attribute, which signals to the compiler that a pointer or a dummy argument is associated with a contiguous set of memory locations. Numerous efficiencies are possible when this assertion is true. A variable of complex type may now have its real and imaginary parts referenced easily by a component-like notation. Pointer functions may indicate the storage to receive the value in an assignment statement, and an elemental procedure may be declared impure to enable diagnostics, or a count of references, or similar activity. With larger memories, the kind of an iteration variable may now be declared on the iteration statement, perhaps to ensure a 64-bit integer is used without changing other portions of the code. Some formatting enhancements aimed towards easing the writing of CSV files have been made. This enables easier communication with workbench programs and spreadsheet programs that can read CSV files (and perhaps easily produce quick plots). Most of the rest of the new items in Fortran 2008 are in the intrinsic library. These are mainly concentrated in the mathematical functions (expanded trig for complex, Bessel, error functions, gamma) and in bit manipulation (unsigned comparison, leading and training zero counts, population counts, and a whole slew of reductions). Analogous with minloc and maxloc (location in an array having the minimum or maximum value), a new findloc (location of a given value) has been specified. The execution of external programs is standardized, if not the command line to be executed. Another new feature is standardized access to the name of the program translation phase (that is, the compiler) and the options used. This is helpful when documenting bug reports, benchmarks, and verification/certification runs. It is also useful when computing "on the cloud" and the compiler and its options may not be known beforehand (or even be the same on each node!), but should be included in the output.

The book has seven appendices. The include a list of the intrinsic procedure names. The authors list their own preferences for deprecated features, and I generally agree. They next list the standard-defined deprecated features list and deleted features list. An extended example of object oriented programming using Fortran facilities is shown, to make clear how it all fits together. Next we find a glossary, useful since the authors use standard-defined terms for clarity throughout. And finally, an appendix has the solutions to the exercises.

That was quite a trip. Modern Fortran is a language with many features, and the authors have explained the whole of it very well. I especially like the rationales given for why certain restrictions exist. Usually, it's to favor efficiency. This has been true ever since the original Formula Translator, but the restrictions still leave applications programmers wanting explanations. Some restrictions may not make sense unless one develops compilers for a living. Most applications programmers do not, so it's helpful to see the whys and wherefores. Also, this book is very thorough. Together, these factors account for its length. But every sentence conveys a useful fact. This book is not light reading, the reader will likely not be taking this book to the beach, it will stay beside the workstation where it can be easily consulted.

I have used earlier versions of this book as a textbook for graduate-level programming classes in Computational Science. I will use this one, given the chance. This book is suitable as a textbook (or supplementary reading) for a scientific programming course, at either the upper-division undergraduate level or the first-year graduate level. Most of the chapters have a least a few problems (solutions to the exercises are given in an appendix). I would choose another language for a first course in programming, perhaps python. Array operations and parallel problem solving in a floating-point environment are sophisticated topics that might not do so well in an introductory presentation. Fortran really is a tool that fully supports large-scale, high-performance number crunching.

I would prefer a different order of topics, where related features are treated together rather than spread over the revisions of the language, with the increment added by each revision described with other, perhaps unrelated features of the same revision. But, given the state of the compilers today, the current order of topics is not so bad. Perhaps in time for the next edition of this book, compilers will have more closely matched the then-current standard (several support coarrays today), so at that time such a reordering will be more useful. In the mean time, an applications programmer knows which revision is in use, more importantly, which features are in use. Modern Fortran, a fully-capable yet easily readable, very portable, highly efficient object-oriented language, with first class support for arrays, and an intuitive, highly scalable parallelism, needs some explanation to be used to best advantage. And that is exactly what these authors with this book have accomplished.