Study guide for the book: Stephen J. Chapman “Fortran 90/95 for Scientists and Engineers”

May 8, 2006

The book describes the full Fortran 95 language. However, in this course we only need material from the first nine chapters and Section 11.1. The more advanced features of Fortran 95 can be very useful for scientific programming and it is recommended to study more of the language before starting a serious programming project. In particular the concept of derived types (Ch 12) and the advanced features of modules (Ch. 13) are very useful.

The book has lots of pages. Therefore: study the examples as detailed as you wish, you can always read them again for details. Get the big picture first. In a way this is true for the main text as well. In the following I will try to indicate what is essential for a first read to start programming useful things. Also don’t forget that the examples in the book can be downloaded form the website of this course. In a second read, after having already produced some useful programs, you can read the other parts more carefully.

Chapter 1

Give some general background and history of the Fortran language. A quick read.

Chapter 2

This chapter should be studied completely: we need all of it. Read about mixed-mode arithmetic (Sec. 2.6.4) but *never us it*. Forget about default typing: *always use the* `implicit none` *statement*. The author of the book has a preference for typing all the keywords of the language using capitals, like `PROGRAM` and `WRITE`. This is not a Fortran 95 requirement and for the compiler it is identical to `program` and `write`, respectively. Actually most programs are written using mainly lower-case characters nowadays which leads to programs that are much better to read. Syntax-highlighting in the editor can be used to distinguish between keywords and variables.

Chapter 3

For those who are not familiar with programming in general Secs. 3.1–3.2 are an interesting read. Learn only about the logical operators `.and.`, `.or.` and `.not.`. Forget the others (for now). The sections 3.4.1 and 3.4.2 on `if`, `else if` etc. are the most important sections of this chapter. To a lesser extent the `select case` construct of

Sec. 3.4.7 can be also useful. Naming of blocks (Secs. 3.4.5–6) can be useful for large programs, but is not required.

Chapter 4

Only the sections 4.1.1, 4.1.3 and the first part of 4.1.6 (nesting loops) are essential on first read. The rest (named loops, `cycle` and `exit` statement, section 4.2 on characters) can be studied later. Read the informative section 4.3 on debugging code.

Chapter 5

Fortran 95 has many ways of performing I/O. In this project we only need very little of that. Essential reading for this project: Secs. 5.1, 5.3.1, 5.3.4, 5.3.6 (I prefer `ES` over `E` format for floating point date), 5.5 (intro), 5.5.1–5.5.4.

Chapter 6

This chapter introduces rank-1 (one-dimensional) arrays. Since Fortran 95 is primarily a language for efficient numerical computations of scientific and technical problems, this is a very important chapter and should be studied completely. The array syntax for whole arrays and array sections of section 6.3 (similar to Matlab) is a very important part of the language.

Chapter 7

This chapter should have been called 'Introduction to Modules and Procedures' stressing the importance of modules in Fortran 95. In fact, this chapter is about modular programming, i.e. splitting the program into smaller parts with a well defined interface. Traditionally you would use procedures (subroutines and functions) for that. Fortran 95 has a higher-level concept, called a *module*. Modules (can) contain definitions, interfaces, data and procedures and form a logical unit that can perform a well-defined task with a well-defined interface to other modules and the main program. For example, in a finite-element program one could define a module `assemble` that contains everything related to assembling the system. Modern Fortran programming is about devising modules with a well-defined task and choosing the correct layout for that.

On first read: only Sec. 7.5 can be skipped. *Always use* the `intent` attribute for dummy arguments (Sec. 7.1.2). In Sec. 7.2 modules are introduced as a means of sharing global data among subroutines. This is the least important use of modules. In fact this practice should be reduced to a minimum. Data should, as much as possible, reside in the main program or being a functional part of a module that performs a well-defined task. Section 7.3 is very important. It introduces the concept of *module procedures*, which have a so-called *explicit* interface. This means that when you call such a procedure the actual arguments are checked on type etc. at *compile time*. The importance of this *cannot be overestimated*. Therefore: *always use explicit interfaces*. Even old Fortran 77 routines can be made explicit (see interface blocks in Chapter 13).

Chapter 8

The first two sections of this chapter are very important and extend the array syntax of Ch6 to arrays of rank 2 and higher. Intrinsic functions and the `where` constructs also form an important part of the array manipulation infrastructure of Fortran 95. Section 8.5 on the `forall` statement is less important and can be skipped on first read. Section 8.6 is again very important, because it introduces a way of defining the size of an array at runtime.

Chapter 9

Section 9.1 shows how to pass multi-dimensional arrays to subroutines and functions and it is required reading. Section 9.2 introduces the `save` statement to save the value of local variables between different calls of the subroutine and can be skipped on first read. Sec. 9.3 describes the important automatic arrays variant and also contains an overview of all the various type of arrays available in Fortran 95. Remember the page of this overview (403) so that you can read it over and over again, because the naming of the various arrays in Fortran 95 is highly confusing! The elemental functions of Sec 9.4 are important, because you can define your own functions that act on whole arrays at once! Internal procedures, described in Sec. 9.5, always come in handy when performing small separate tasks local to a routine or when defining a local function.

Chapter 11

Sec. 11.1 introduces the `real` data type of higher precision and is very important for numerical programming. This should have been part of Chapter 2. Always use the higher precision ($p = 15$) for computation where possible. Use standard precision ($p = 6$) only when you absolutely need to save disk space when using binary data files or you lack the computer memory to store the data in core.