Study guide for the book: Stephen J. Chapman “Fortran 95/2003 for Scientists and Engineers”

January 7, 2009

The book describes the full Fortran 95 language[1]. However, in this course we only need material from the first nine chapters, Section 11.1, Chapter 12 and Section 13.1. 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 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 the book[2] . 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`. Actually most programs are written using mainly lower-case characters nowadays and I find that much better to read.

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

[1]The book also describes the latest standard Fortran 2003. The Silverfrost compiler is only capable of compiling Fortran 95. Therefore, the Fortran 2003 extensions will not be used in this project.

[2]`www.mhhe.com/chapman3e` under the link ‘Book files’.

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. Read the informative section 3.5 on debugging code.

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.

In Section 7.1 subroutines are introduced. *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).

Section 7.4 introduces functions and Section 7.5 describes how procedures (functions and subroutines) can be put in the argument list.

Chapter 8

The first three sections of this chapter are very important and extend the array syntax of Ch6 to arrays of rank 2 and higher. The `where` constructs also form an important part of the array manipulation infrastructure of Fortran 95, but do not need to be studied in the first read. Section 8.5 on the `forall` statement is less important and can also be skipped on first read. Section 8.6 (only for Fortran 95, so skip Section 8.6.2) 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 extends allocatable arrays to procedures and Sec. 9.4 describes the important automatic arrays variant and also contains an overview of all the various type of arrays available in Fortran 95. The rest of the chapter is optional and is not needed for this project.

Chapter 10

In this chapter more features of character variables are introduced. This is an optional read, since we do not need it for this project.

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.

The remaining part of the chapter is optional and not needed for this project.

Chapter 12

This chapter introduces the very important *derived data types*. Although the example used is the unavoidable `person` data type, the derived data type can be used to group together all kind of information in numerical programs as well. For example, in a finite-element program all the information of a mesh (coordinates, topology, element type, ...) can be stored in one single structure. The data in the structure can be passed to subroutines as a whole. Only the Sections 12.1–12.5 should be read. Skip the rest (which is only for Fortran 2003).

A natural definition for derived types containing 'dynamic' arrays would be something like

```
type :: mesh_t
  ...
  real, dimension(:,:), allocatable :: coor
  ...
end type mesh_t
```

however in Fortran 95 the `allocatable` atribute is not allowed for components of derived types, unfortunately[3]. The solution to this is defining the array `coor` as a pointer (see Chapter 14):

```
type :: mesh_t
  ...
  real, dimension(:,:), pointer :: coor => null()
  ...
end type mesh_t
```

The array `coor` can now be allocated and deallocated as usual and the data in the array can also be used as usual:

```
type ( mesh_t) :: mesh
...
allocate ( mesh%coor(100,2) )
...
mesh%coor(1:10,1) = 1.0
...
deallocate ( mesh%coor )
```

Chapter 13

Section 13.1 discusses the important subject of the *scope* of variables, i.e. where do variables live and are they available to a program unit? Especially with respect to variables defined in a `module`, this is an important subject.

The remaining part of this chapter is optional and will not be needed in this project.

[3]It is available in the new standard Fortran 2003, and many (but not all, including the Silverfrost compiler) compilers offer it as an extension to Fortran 95. We try to avoid it for now.