PREFACE

The first edition of this book was conceived as a result of my experience in writing and maintaining large Fortran programs in both the defense and geophysical fields. During my time in industry, it became obvious that the strategies and techniques required to write large, *maintainable* Fortran programs were quite different from what new engineers were learning in their Fortran programming classes at school. The incredible cost of maintaining and modifying large programs once they are placed into service absolutely demands that they be written to be easily understood and modified by people other than their original programmers. My goal for this book is to teach simultaneously both the fundamentals of the Fortran language and a programming style that results in good, maintainable programs. In addition, it is intended to serve as a reference for graduates working in industry.

It is quite difficult to teach undergraduates the importance of taking extra effort during the early stages of the program design process in order to make their programs more maintainable. Class programming assignments must by their very nature be simple enough for one person to complete in a short period of time, and they do not have to be maintained for years. Because the projects are simple, a student can often "wing it" and still produce working code. A student can take a course, perform all of the programming assignments, pass all of the tests, and still not learn the habits that are really needed when working on large projects in industry.

From the very beginning, this book teaches Fortran in a style suitable for use on large projects. It emphasizes the importance of going through a detailed design process before any code is written, using a top-down design technique to break the program up into logical portions that can be implemented separately. It stresses the use of procedures to implement those individual portions, and the importance of unit testing before the procedures are combined into a finished product. Finally, it emphasizes the importance of exhaustively testing the finished program with many different input data sets before it is released for use.

In addition, this book teaches Fortran as it is actually encountered by engineers and scientists working in industry and in laboratories. One fact of life is common in all programming environments: Large amounts of old legacy code that have to be maintained. The legacy code at a particular site may have been originally written in Fortran IV (or an even earlier version!), and it may use programming constructs that are no longer common today. For example, such code may use arithmetic `IF` statements, or computed or assigned `GO TO` statements. Chapter 18 is devoted to those older features of the language that are no longer commonly used, but that are encountered in legacy code.

The chapter emphasizes that these features should *never* be used in a new program, but also prepares the student to handle them when he or she encounters them.

CHANGES IN THIS EDITION

This edition builds directly on the success of *Fortran 95/2003 for Scientists and Engineers*, 3/e. It preserves the structure of the previous edition, while weaving the new Fortran 2008 material (and limited material from the proposed Fortran 2015 standard) throughout the text. It is amazing, but Fortran started life around 1954, and it is *still* evolving.

Most of the additions in Fortran 2008 are logical extensions of existing capabilities of Fortran 2003, and they are integrated into the text in the proper chapters. However, the use of parallel processing and Coarray Fortran is completely new, and Chapter 17 has been added to cover that material.

The vast majority of Fortran courses are limited to one-quarter or one semester, and the student is expected to pick up both the basics of the Fortran language and the concept of how to program. Such a course would cover Chapters 1 through 7 of this text, plus selected topics in Chapters 8 and 9 if there is time. This provides a good foundation for students to build on in their own time as they use the language in practical projects.

Advanced students and practicing scientists and engineers will need the material on COMPLEX numbers, derived data types, and pointers found in Chapters 11 through 15. Practicing scientists and engineers will almost certainly need the material on obsolete, redundant, and deleted Fortran features found in Chapter 18. These materials are rarely taught in the classroom, but they are included here to make the book a useful reference text when the language is actually used to solve real-world problems.

FEATURES OF THIS BOOK

Many features of this book are designed to emphasize the proper way to write reliable Fortran programs. These features should serve a student well as he or she is first learning Fortran, and should also be useful to the practitioner on the job. They include:

1. *Emphasis on Modern Fortran.*
   The book consistently teaches the best current practice in all of its examples. Many modern Fortran 2008 features duplicate and supersede older features of the Fortran language. In those cases, the proper usage of the modern language is presented. Examples of older usage are largely relegated to Chapter 18, where their old/undesirable nature is emphasized. Examples of modern Fortran features that supersede older features are the use of modules to share data instead of `COMMON` blocks, the use of `DO . . . END DO` loops instead of `DO . . . CONTINUE` loops, the use of internal procedures instead of statement functions, and the use of `CASE` constructs instead of computed `GOTO`s.

2. *Emphasis on Strong Typing.*
   The `IMPLICIT NONE` statement is used consistently throughout the book to force the explicit typing of every variable used in every program, and to catch common typographical errors at compilation time. In conjunction with the explicit declaration of every variable in a program, the book emphasizes the importance of creating a data dictionary that describes the purpose of each variable in a program unit.
3. *Emphasis on Top-Down Design Methodology.*
   The book introduces a top-down design methodology in Chapter 3, and then uses it consistently throughout the rest of the book. This methodology encourages a student to think about the proper design of a program *before* beginning to code. It emphasizes the importance of clearly defining the problem to be solved and the required inputs and outputs before any other work is begun. Once the problem is properly defined, it teaches the student to employ stepwise refinement to break the task down into successively smaller subtasks, and to implement the subtasks as separate subroutines or functions. Finally, it teaches the importance of testing at all stages of the process, both unit testing of the component routines and exhaustive testing of the final product. Several examples are given of programs that work properly for some data sets, and then fail for others.

   The formal design process taught by the book may be summarized as follows:

   - *Clearly state the problem that you are trying to solve.*
   - *Define the inputs required by the program and the outputs to be produced by the program.*
   - *Describe the algorithm that you intend to implement in the program. This step involves top-down design and stepwise decomposition, using pseudocode or flow charts.*
   - *Turn the algorithm into Fortran statements.*
   - *Test the Fortran program. This step includes unit testing of specific subprograms, and also exhaustive testing of the final program with many different data sets.*

4. *Emphasis on Procedures.*
   The book emphasizes the use of subroutines and functions to logically decompose tasks into smaller subtasks. It teaches the advantages of procedures for data hiding. It also emphasizes the importance of unit testing procedures before they are combined into the final program. In addition, the book teaches about the common mistakes made with procedures, and how to avoid them (argument type mismatches, array length mismatches, etc.). It emphasizes the advantages associated with explicit interfaces to procedures, which allow the Fortran compiler to catch most common programming errors at compilation time.
5. *Emphasis on Portability and Standard Fortran.*
   The book stresses the importance of writing portable Fortran code, so that a program can easily be moved from one type of computer to another one.

It teaches students to use only standard Fortran statements in their programs, so that they will be as portable as possible. In addition, it teaches the use of features such as the `SELECTED_REAL_KIND` function to avoid precision and kind differences when moving from computer to computer.

The book also teaches students to isolate machine-dependent code (such as code that calls machine-dependent system libraries) into a few specific procedures, so that only those procedures will have to be rewritten when a program is ported between computers.

6. *Good Programming Practice Boxes.*

   These boxes highlight good programming practices when they are introduced for the convenience of the student. In addition, the good programming practices introduced in a chapter are summarized at the end of the chapter. An example Good Programming Practice Box is shown below:

Good Programming Practice

Always indent the body of an `IF` structure by two or more spaces to improve the readability of the code.

7. *Programming Pitfalls Boxes*

   These boxes highlight common errors so that they can be avoided. An example Programming Pitfalls Box is shown below:

Programming Pitfalls

Beware of integer arithmetic. Integer division often gives unexpected results.

8. *Emphasis on Pointers and Dynamic Data Structures.*

   Chapter 15 contains a detailed discussion of Fortran pointers, including possible problems resulting from the incorrect use of pointers such as memory leaks and pointers to deallocated memory. Examples of dynamic data structures in the chapter include linked lists and binary trees.

   Chapter 16 contains a discussion of Fortran objects and object-oriented programming, including the use of dynamic pointers to achieve polymorphic behavior.

9. *Use of Sidebars.*

   A number of sidebars are scattered throughout the book. These sidebars provide additional information of potential interest to the student. Some sidebars are historical in nature. For example, one sidebar in Chapter 1 describes the IBM Model 704, the first computer to ever run Fortran. Other sidebars

reinforce lessons from the main text. For example, Chapter 9 contains a sidebar reviewing and summarizing the many different types of arrays found in modern Fortran.

10. *Completeness.*

    Finally, the book endeavors to be a complete reference to the modern Fortran language, so that a practitioner can locate any required information quickly. Special attention has been paid to the index to make features easy to find. A special effort has also been made to cover such obscure and little understood features as passing procedure names by reference, and defaulting values in list-directed input statements.

PEDAGOGICAL FEATURES

The book includes several features designed to aid student comprehension. Each chapter begins with a list of the objectives that should be achieved in that chapter. A total of 27 quizzes appear scattered throughout the chapters, with answers to all questions included in Appendix F. These quizzes can serve as a useful self-test of comprehension. In addition, there are approximately 360 end-of-chapter exercises. Answers to selected exercises are available at the book's Web site, and of course answers to all exercises are included in the Instructor's Manual. Good programming practices are highlighted in all chapters with special Good Programming Practice boxes, and common errors are highlighted in Programming Pitfalls boxes. End-of-chapter materials include Summaries of Good Programming Practice and Summaries of Fortran Statements and Structures. Finally, a detailed description of every Fortran intrinsic procedure is included in Appendix C, and an extensive Glossary is included in Appendix E.

The book is accompanied by an Instructor's Manual, containing the solutions to all end-of-chapter exercises. Instructors can also download the solutions in the Instructor's Manual from the book's Web site. The source code for all examples in the book, plus other supplemental materials, can be downloaded by anyone from the book's Web site.

A NOTE ABOUT FORTRAN COMPILERS

Two Fortran compilers were used during the preparation of this book: the Intel Visual Fortran Compiler Version 16.0 and the GNU G95 Fortran compiler. Both compilers provide essentially complete implementations of Fortran 2008, with only a very few minor items not yet implemented. They are also both looking to the future, implementing features from the proposed Fortran 2015 standard.

I highly recommend both compilers to potential users. The great advantage of Intel Fortran is the very nice integrated debugging environment, and the great disadvantage is cost. The G95 compiler is free, but it is somewhat harder to debug.