UNDERSTANDING ADA

WITH ABSTRACT DATA TYPES

SECOND EDITION

KEN SHUMATE

UNDERSTANDING ADA
WITH ABSTRACT DATA TYPES

Second Edition

KEN SHUMATE
TeleSoft

JOHN WILEY & SONS
NEW YORK • CHICHESTER • BRISBANE • TORONTO • SINGAPORE

Library of Congress Cataloging in Publication Data:

Shumate, Kenneth C.
Understanding Ada: with abstract data types. Ken Shumate. 2nd ed.

p. cm.
Includes index.
ISBN 0-471-60520-4
1. Ada (Computer program language) 2. Abstract data types
(Computer science) I. Title.
QA76.73.A35S48 1989
005.13'3—dc19 88-28743
CIP

Printed in the United States of America

10 9 8 7 6 5 4 3 2

Printed and bound by Malloy Lithographing, Inc.

Preface

Ada is a computer programming language developed for the United States Department of Defense (DoD). The objective of this book is to provide an understanding of Ada. It is intended to be a first book on Ada for programmers with experience in at least one other high-level language.

Understanding Ada takes a very simple approach to explaining the language. At the same time, it covers all aspects of Ada needed to begin programming effectively, and provides a foundation for the further study of Ada and its use in constructing large software systems.

The following sections discuss the important features of the book, its organization, basis for overall approach, and potential uses.

IMPORTANT FEATURES

There are four important features of this book:

- Graduated Introduction to the Language
- Abstract Data Types
- Introduction to Programming-in-the-Large
- Exercises

Graduated Introduction to the Language

Understanding Ada provides a graduated introduction to the language, using Pascal as a basis. It is *not* expected that the reader know Pascal. Ada is introduced at a Pascal-like level to provide an easy introduction for the

Pascal concepts adapted for Ada (strong typing, enumeration types, records, pointers, and so on), with which many readers will not be familiar. On the other hand, readers who *do know* Pascal will be able to progress quickly because they will be able to make comparisons and to understand Ada by analogy with Pascal.

How does the book treat the gradual introduction? There is first of all a short but complete overview of the entire language, followed by a discussion of the use of Ada for developing large systems.

It then covers (1) Ada at about the complexity level of Pascal, (2) additional aspects of Ada that go beyond Pascal but retain the nature of an algorithmic language, and (3) the advanced features of Ada—why and how Ada differs from other languages. The final technical chapter provides an introduction to concurrency that is stronger than usual for a first book on Ada.

Abstract Data Types

The second important feature is a strong emphasis on abstract data types. The abstract data type is introduced in a very simple and uncomplicated way, exactly like the use of abstract data types in Pascal. Improved methods of using abstract data types are introduced as Ada's features are explained. The discussion is complementary; the explanation simultaneously shows how to construct data abstractions properly by using Ada's features, while using this explanation to explain the Ada features themselves. By the end of this book-long discussion, abstract data types are presented as secure, reusable software components, for both sequential and concurrent programs.

Introduction to Programming-in-the-Large

The third feature is the introduction of some basic principles and methods of using Ada to design large software systems—an introduction to programming-in-the-large. This is intended to promote understanding of why Ada contains numerous features to support software engineering principles. The discussion of abstract data types supports this understanding, but the primary mechanism for discussing programming-in-the-large is the use of

case studies. Each case study involves a problem statement called a software requirements specification, top-level design, detailed design, and code. Explanation and discussion are provided at each stage. Case studies are used both as examples and as exercises. One of the case studies presents a discussion of some important software engineering principles, and blends these principles into an effective Ada-oriented design methodology.

Exercises

The preceding features are supported by, and well integrated with, the fourth major feature of the book—an extensive set of solved exercises. The best way to understand Ada is to solve problems. This book provides four opportunities for you to do so.

First, there are small problems called examples that capture some major feature of a chapter. They are solved and discussed as an integral part of the text of the chapter.

Second, there are a set of what I call "Immediate Exercises" dealing with abstract data types. They are denoted with a special "stop sign" symbol, encouraging readers to stop and solve them before proceeding. The immediately following text provides a solution and discussion. They are an integral part of the text of the book and are the primary mechanism for explaining abstract data types.

Third, there is an extensive set of end-of-chapter exercises. In addition to the usual sort of exercises that illustrate and provide practice in using individual language features, there is an on-going graduated set of problems dealing with abstract data types.

Fourth and last are problems dealing with programming-in-the-large. To reinforce the issue of using Ada for design, there are five small case studies, each one building on the previous one. Solution of these case studies calls not only for the use of Ada's advanced features but also for the application of data abstraction and other software engineering principles.

Solutions to odd-numbered end-of-chapter exercises and basic solutions to the case studies are presented at the end of the book. The solutions are more than simple code examples; they include discussion of language features and are intended to be an important part of the book. The exercises will increase the sophistication and degree of understanding of Ada.

Solutions to even-numbered end-of-chapter exercises, further elaboration of the case studies, and more discussion of Ada features and how to use them are presented in an available Instructor's Manual.

ORGANIZATION OF THE BOOK

The book is organized into five parts and eight appendixes.

Part 1, Introduction, sets the foundation for the remainder of the book. It provides some simple examples of Ada programs, presents overviews of both the language itself and how it will influence the development of large software systems, and discusses the history/rationale for Ada's creation.

Part 2, Ada as Pascal, presents Ada as a modern programming language using Pascal as a base. It does not attempt a subset of Ada, but instead addresses those features of Ada that are comparable to Pascal in terms of level of complexity. Part 2 introduces the construction of abstract data types by using the features of Pascal.

Part 3, Ada as Ada, presents those features of Ada that are advances over Pascal. Most importantly, Part 3 introduces the basic ideas of the *package*. The discussion, on a chapter-by-chapter basis, parallels the presentation in Part 2. The importance of the package in building data abstractions is stressed.

Parts 2 and 3 present Ada as an algorithmic language, what is sometimes called a classical language. The features of Ada presented thus far do not meet the requirements for building large real-time systems.

Part 4, Advanced Features, presents features of Ada that *will* allow it to be used effectively for construction of large, long-lived, real-time systems. It includes features for secure data abstractions, for building reusable software components, for dealing with errors and other exceptional situations, and for dealing with concurrency and low-level access to hardware.

Part 5, All about Ada, provides an introduction to the Ada milieu. It is nontechnical and could easily be read immediately following Chapter 2.

Appendixes A through E are portions of the Ada Language Reference Manual and are intended to make the book self-sufficient. Appendixes F through H are case studies and discussions that are intended to illustrate programming-in-the-large, including the introduction of an Ada-oriented design methodology. They are referenced in both the chapter text and the exercises.

BASIS FOR FEATURES/ORGANIZATION

The features and organization of *Understanding Ada* are expressly intended to meet two goals:

- Provide an easy introduction to the language.
- Provide a solid foundation for more advanced understanding of Ada.

The first goal is satisfied by the *Graduated Introduction* to the language. Using "Ada as Pascal" to introduce Ada concepts was inspired by my early experience teaching introductory Ada to engineers with a wide variety of programming backgrounds, but often only Fortran. Teaching Ada "all at once" failed to take advantage of their already sound base in conventional languages and made Ada appear to be more complex than it is.

Some in the Ada community have advanced the idea that it is best to learn Pascal before Ada. I do not think this is necessary. I prefer the approach of teaching Ada at the same level of complexity of Pascal, and then adding features—and their related software engineering principles—to that base. This is a very natural approach, particularly since Pascal was the base language for the design of Ada. This approach has been influenced by my happy experiences teaching Pascal both as a first language and to experienced programmers. Pascal is an effective language for teaching, as is "Ada as Pascal."

The second goal is satisfied by the emphasis on the features *Abstract Data Types* and *Introduction to Programming-in-the-Large*. This emphasis is based on my experience teaching advanced courses in Ada. In fact, *Understanding Ada: With Abstract Data Types* is the foundation for a three-course sequence (each the equivalent of an academic semester):

- Introduction to Ada
- Concurrency in Ada
- Designing Large Real-Time Systems with Ada

This book is the text for the first course and is specifically intended and designed to provide programmers with the necessary foundation for the more advanced courses or other advanced reading in Ada.

Data abstraction is central to successful software design and is the basis for maintainable systems and for reusable software components. It is

important to introduce the ideas early and to reinforce them often. It is equally important to provide early introduction of examples of large programs. Solving more complex problems introduces the idea of software design and illustrates correct use of Ada's features. Even a first book on Ada should point to its use for the construction of large systems.

Both goals are supported by the *Exercises*. The specific approach is based on my belief and observation that the best way to learn Ada is to solve problems. In fact, the primary method for teaching abstract data types is the Immediate Exercise. This method challenges you to stop and think about the issues presented—and write some code—before going on. The solution/discussion then completes the exercise, illustrating how to specify or implement the data abstraction and reinforcing the earlier discussion and your grasp of the concept. This approach is derived from the way I teach this material in a hands-on intensive course. At the point of the immediate exercise, the students attempt a solution in the laboratory before we continue the discussion. This has proved effective.

The end-of-chapter exercises are more conventional, but are extensive, particularly well integrated with the text discussion on abstract data types, and are specifically intended to increase sophistication in understanding Ada. The case study exercises continue to provide a foundation for the two more advanced courses, which are largely oriented to a case-study approach to teaching Ada advanced concepts.

The goals of providing an easy introduction to the language while providing a solid foundation for advanced study—and at the same time covering all aspects of Ada needed to begin programming effectively—are somewhat contradictory. Meeting the goals calls for hard judgments of what to include and what to defer. I've tried to balance simplicity with thoroughness in such a way that the material is easy to understand, provides a sound basis for programming in Ada, and establishes the necessary foundation for later courses or reading.

USE OF THE BOOK

This book is suitable for Ada-intensive courses, for corporate training programs, or as a supplemental text for undergraduate courses that require programming: simulation, software engineering, analysis of algorithms, and so on. It has also been used as a text to accompany courses in compiler or

language design. It could also serve as the basic text for teaching students to program, as the foundation for an academic course in the language Ada, or simply be read by an individual engineer. The book is intended to be an easy introduction to the language for the programmer who wants to begin understanding Ada.

Ken Shumate

San Diego
May 1988

Acknowledgements

FOR THE SECOND EDITION

Tom Burger prepared the end-of-chapter exercises. He is the author of the instructor's manual available to assist in teaching the course. Beyond that, he has been important in helping me refine my ideas about Ada, and about how to present the language to both introductory and advanced students. The new examples and the exercises have been developed using the VAX Ada Compilation System and the TeleSoft Second Generation VAX Ada compiler. They have proven to be effective tools.

Thanks to G. Anderson, K. McCann, and students of both introductory and advanced Ada courses for helpful comments on the first edition and for review of the new material for the second edition.

FOR THE FIRST EDITION

Many people made important contributions during the preparation of this book. I particularly wish to thank those who read early versions of the manuscript for understandability: G. Anderson, B. Colborn, R. Fritz, J. Hooper, K. Nielsen, and R. Sauer. A number of reviewers made valuable comments, as did associates at Hughes Aircraft Company. ROLM and Data General were helpful in providing access to their Ada compiler. Special thanks to M. Shumate, who editied the manuscript during its developement. Maureen, John, Karen, and Kelly Shumate were supportive during the entire project.

Contents

PART 1

INTRODUCTION

Ada is the new computer programming language specified by the United States Department of Defense (DoD) for the programming of computers embedded within larger systems. Computers internal to aircraft, ships, radars, or command and control systems are used in different ways than in business or data processing applications. Computers in such embedded computer systems typically interface with human operators and external devices in real time, as events are occurring. They read signals from sensors and send commands to electrical and electromechanical devices. Commercial systems such as process control and data communications have similiar characteristics. Ada will eventually be the single language for programming DoD embedded computer systems, will find similar use among the defense establishments of allies of the United States, and is certain to have widespread commercial use. In fact, the first delivered production Ada software was a payroll and inventory system for a truck manufacturer.

Part 1, "Introduction," presents some simple examples of Ada programs that show the form of the language, summarizes the history of Ada development and its technical requirements, provides an overview of the complete language, including its advanced features, and discusses how software will be developed using Ada. This introduction sets the foundation for the language-specific presentations of Parts 2, 3, and 4.

Part 1 also introduces, in Appendix F, an Ada design methodology called Layered Virtual Machine/Object-Oriented Design. You should read the appendix as an overview of how Ada parts "fit together" to design large software systems. You can then read it in detail in conjunction with learning Ada's advanced features in Part 4.

1

Jumping Right In

OBJECTIVE: to provide some simple examples of Ada programs

1.1 *FIRST ADA PROGRAM*

Let's begin by looking at a small Ada program whose net effect is to write "Hello, World" on a standard output device.

```
with Text_IO; use Text_IO;
procedure Hello_World is
begin
  Put ("Hello, World");
end Hello_World;
```

Hello_World is the name of the procedure, which is an executable block of code. The executable part is delineated by **begin** and **end**; the procedure itself is delineated by **procedure** Hello_World and **end** Hello_World. The *context* clause "**with** Text_IO; **use** Text_IO;" provides access and visibility to input/output facilities, for example, the "Put." We will use lowercase boldface to indicate Ada reserved words that appear in the text of the book other than examples of code.

1.2 *ANOTHER EXAMPLE*

Here is a longer example, with comments. Note that "--" indicates that what follows is a comment (-- This is a comment).

```
with Text_IO; use Text_IO;
procedure Grades is
-- Grades computes the Sum and Maximum of
-- a set of grades
   Grade, Sum, Maximum, Number_of_Students : Integer;
   package IO is new Integer_IO (Integer); use IO;
begin
   Get (Number_of_Students);
   Maximum := 0;
   Sum := 0;
   for Loop_Count in 1 .. Number_of_Students loop
     Get (Grade);
     Sum := Sum + Grade;
     if Grade > Maximum then
        Maximum := Grade;
     end if;
   end loop;
   Put ("Maximum Grade Is : "); Put (Maximum); New_Line;
   Put ("Sum of Grades Is : "); Put (Sum);     New_Line;
end Grades;
```

As you can see by the examples, Ada is a high-level language with a structure similar to other modern languages. (The "**package** IO is **new**..." gets us access to input/output for integer values.) Indeed, as will be shown in Part 2, much of Ada is as easily understandable as Pascal, which was designed to be a simple language to be used in teaching programming.

1.3 *WHY ADA?*

Ada goes beyond Pascal in its ability to define new types of data objects and provides additional measures to help ensure safe programming practices. Certain language features allow more programmer errors to be caught at compile time rather than during run-time testing (or rather than not being caught at all and turning up as bugs in the system). Even further, Ada provides capabilities for concurrent programming, for error detection and