ADA® AS A SECOND LANGUAGE

Norman H. Cohen

ADA® AS A SECOND LANGUAGE

Norman H. Cohen
SofTech, Inc.

McGraw-Hill Book Company
New York St. Louis San Francisco Auckland Bogotá Hamburg Johannesburg
London Madrid Mexico Montreal New Delhi Panama Paris
São Paulo Singapore Sydney Tokyo Toronto

This book was set in Times Roman by Publication Services.
The editor was Gerald A. Gleason;
the production supervisor was Leroy A. Young;
the cover was designed by Mark Wieboldt.
Project supervision was done by Publication Services.
Arcata Graphics/Halliday was printer and binder.

ADA® AS A SECOND LANGUAGE

Copyright © 1986 by McGraw-Hill, Inc. All rights reserved.
Printed in the United States of America. Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the prior written permission of the publisher.

1 2 3 4 5 6 7 8 9 0 HALHAL 8 9 8 7 6

Library of Congress Cataloging-in-Publication Data
Cohen, Norman H.
ADA® as a second language.
1. Ada (Computer program language) I. Title.
QA76.73.A35C64 1986 005.13'3 85-2946
ISBN 0-07-011589-3

Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

ABOUT THE AUTHOR

Norman H. Cohen is a system consultant for SofTech, Inc. and author of numerous articles on the Ada language. He led the development of major parts of the U.S. Army Ada Curriculum and teaches Ada language and software engineering classes. Dr. Cohen was a principal contributor to a series of case studies on the use of the Ada language, chief designer of an Ada-based program design language, and leader of a research project on the formal verification of Ada programs. He has chaired the Philadelphia area chapter of SIGAda (the Association for Computing Machinery's Special Interest Group on Ada) and a government-sponsored working group on Ada formal semantics. He received his B.A. in computer science from Cornell University and his M.S. and Ph.D. in computer science from Harvard University.

To Dianne; and to David,
Ilana, and Aviva

此为试读,需要完整PDF请访问: www.ertongbook.com

PREFACE

Ada as a Second Language is designed to fulfill a wide range of needs. The only prerequisite is an introductory programming course or equivalent practical experience. The book may be used to obtain a reading knowledge of the Ada language, to obtain a writing knowledge of the Ada language, or to serve as a programming reference for someone who is already writing Ada programs. It may be used as a textbook in a course or to learn the Ada language on one's own.

Ada as a Second Language is at once a tutorial introduction to the Ada language and a complete reference. In teaching the language, the book takes time to explain complicated matters in a patient, reassuring manner, with generous explanations. The exposition emphasizes the concerns of the practicing programmer, not theoretical principles of programming languages. As a reference, the book contains a complete description of the nitty-gritty details a programmer needs to write practical, working Ada programs, not just the general principles needed to convey the "flavor" of the language. The blemishes of the language, and their practical implications for the programmer, are described along with the language's graceful contours. There are abundant cross-references to the sections in which concepts are introduced.

WHAT IS THE FIRST LANGUAGE?

The book is entitled *Ada as a Second Language* because it is an introduction to the Ada programming language, but not an introduction to programming. We assume the reader is familiar with certain fundamental programming notions—variables, arrays, loops, conditional statements, and subprograms, for example. The first language can be any statement-oriented high-level language. However, we pay particular attention to FORTRAN, PL/I, and Pascal.

The Ada language is based on programming and software-engineering concepts that may be new to a FORTRAN, PL/I, or Pascal programmer. We take care to make the reader comfortable with programming concepts like programmer-defined types, recursion, pointers, exception handling, and concurrency before the corresponding Ada language feature is introduced. Furthermore, software-engineering concepts like abstract data types, information hiding, and loose coupling, which do not find direct expression in the FORTRAN, PL/I, and Pascal programming languages, must be well understood in order to use the Ada programming language properly. We explain such software-engineering concepts in depth before considering the associated Ada features.

Ada as a Second Language frequently compares the Ada language's features to similar features of FORTRAN, PL/I, and Pascal, to help readers recognize those instances where the Ada language provides a new notation for a familiar concept. For example, the Ada language's FOR loop is compared to the DO loops of FORTRAN and PL/I and the **for** loop of Pascal. Of course the exposition of an Ada feature never relies on a reader's familiarity with another language.

We also point out circumstances in which familiarity with FORTRAN, PL/I, or Pascal may confuse students of the Ada language or prejudice them to program in a manner that is not appropriate for the Ada language. For example, there is a subtle difference between the treatment of loop control variables in the Ada language and their treatment in the other three programming languages, and this difference can lead to enigmatic errors; entities known as *constants* in FORTRAN and PL/I are called *literals* in the Ada language, and the term *constant* has a quite different meaning; packages can be used in the Ada language in the same way as COMMON blocks are used in FORTRAN, but this is rarely appropriate; the exceptions of the Ada language bear strong similarity to the ON-conditions of PL/I, but there are important differences that make PL/I condition handling approaches inappropriate in Ada programs.

While the comparative study of programming languages is fascinating, it is not the subject of this book. Our emphasis is overwhelmingly on the Ada language. Other languages are discussed only to the extent that the discussion is likely to help the reader learn the Ada language.

APPROACH

A programming language cannot be taught simply by enumerating rules. Therefore, *Ada as a Second Language* is replete with realistic examples of Ada programming. One-line examples have their place, but extended examples are also necessary, to illustrate how features fit together and the contexts in which they ought to be used. The reader will benefit from over 200 complete Ada compilation units that have been compiled by a validated Ada compiler to verify their legality. Examples have been carefully crafted to be neither too simplistic nor too complicated. The examples are substantial enough to give a realistic idea of how features should be used, but not so involved that they detract from the point being made. Our primary goal, after all, is to teach the Ada language, not particular algorithms or applications.

Just as one cannot learn to play bridge well simply by learning the rules of the game, one cannot learn to program well simply by learning the rules of the programming language. *Ada as a Second Language* explains the basic concepts underlying the features of the Ada language—data abstraction (which it has recently become fashionable to call "object-oriented design"), modularity, the distinction between interfaces and implementations, information hiding, portability, and concurrency, for example—before explaining the features themselves. The book offers specific practical advice on how and why to use each feature, and warns about pitfalls to be avoided.

Good programming style is emphasized throughout the text. Guidelines are provided, for example, on the use of long, descriptive identifiers; the formatting of program text; the appropriate use of exceptions; and the minimization of recompilation costs. The examples consistently practice what the text preaches.

STRUCTURE OF THE BOOK

The typical chapter begins with a discussion of the programming and software-engineering concepts underlying a language feature, then goes on to discuss the essential aspects of the feature itself. Detailed rules about the feature follow later in a separate section. These details must be learned to write correct Ada programs but are best deferred until the reader has become comfortable with general principles. These details can be skipped entirely by the reader who seeks only a general familiarity with the Ada language or the ability to understand Ada programs written by others. Each chapter ends with a summary of the major points and a set of exercises. The exercises include drills on fundamental concepts as well as programming problems.

Chapters 1 and 2 contain introductory material. Chapter 1 briefly recounts the concerns and events that led to the creation of the Ada programming language. This chapter is not essential for learning to read or write Ada programs, but it does provide perspective on the motivation for various language features. Chapter 2 provides a high-level overview of the entire language. In addition, the Ada language's lexical rules and the notation we use for describing Ada syntax are presented there. It is the most important chapter in the book, providing an overall framework within which the specific information in subsequent chapters can be understood. In particular, this chapter introduces the crucial Ada notions of separate compilation units and packages.

Chapters 3 through 6 are concerned with the predominant issue in Ada programs, data types. Chapter 3 introduces the notion of an abstract data type as a set of abstract values plus operations on those values, independent of the underlying physical representation; and describes how the FORTRAN, PL/I, Pascal, and Ada languages provide increasingly comprehensive support for data abstraction. This chapter provides the background needed to appreciate the role of data types in the Ada language, and particularly the role of private types. Chapter 4 describes object and type declarations and discusses six classes of types in detail—integer, floating-point, fixed-point, enumeration, array, and record types. Access types are deferred until Chapter 5 to permit a thorough discussion of the underlying concepts, which will be new to FORTRAN programmers. Chapter 6 introduces the notion of subtypes. The distinction between subtypes and types, a common stumbling block for newcomers to the Ada language, is carefully explained. Derived types are introduced in this chapter to illustrate the concept of strong typing.

Chapters 7 through 9 describe the traditional algorithmic features found in all procedural languages. These features are used in earlier chapters and the reader will already be intuitively familiar with them, but specific rules, caveats, and guidelines are provided here. Chapter 7 describes in detail the statements used in nonconcurrent programming. Chapter 8 gives detailed rules about expressions. Chapter 9 treats subprograms.

The discussion on record types in Chapter 4 is confined to record types without discriminants. Record types with discriminants are introduced in Chapter 10. We deliberately separate this discussion from the earlier coverage of data types. The reader can then master the basic principles of data types before tackling the more intricate topic of discriminants. Furthermore, the thorough treatment of subprograms in Chapter 9 permits more complete and realistic illustrations of how discriminants are used.

The first ten chapters contain all the information necessary for writing small Ada programs and individual subprograms. For some programmers, this may be sufficient. However, it is the features described in the second half of the book—particularly support for programming in the large, exceptions, generic program units, and concurrency—that make the Ada language unique.

Chapters 11 through 13 describe the features that make it possible to write large Ada programs while managing their complexity. Though packages are introduced in Chapter 2 and used in subsequent chapters, Chapter 11 covers them in detail, relating packages to the notions of modularity and information hiding. Chapter 12 describes private and limited private types and relates them to the discussion of data abstraction in Chapter 3. Chapter 13 discusses separate compilation. Each of these chapters includes abundant advice on the appropriate use of language features, for example, when it is appropriate to make a private type limited and how to reduce potential recompilation costs.

Nested program units are introduced as early as Chapter 2, and by this point the reader should have a strong intuitive notion of the Ada language's scope and visibility rules. Chapter 14 builds on this intuition to provide a formal set of rules. The idea is to solidify the reader's understanding of scope and visibility and to enable him to answer scope and visibility questions in unusual situations where the answer might not be intuitively obvious. The rules are applied to specific examples to illustrate how they lead to the conclusions that the reader expects. This will help the reader apply the rules in other situations.

Chapter 15 deals with exceptions. This chapter includes an optional section for PL/I programmers, comparing Ada exceptions to PL/I ON-conditions. A considerable portion of the chapter is devoted to guidelines for the appropriate use of exceptions.

Chapter 16 covers generic units. The need for generic units is described by example and the basic notions of template and instance are then discussed. The mechanics of writing and using generic units are illustrated by examples that convey the power of generic units and the variety of ways in which they can be used. A section on generalization provides the thoughtful reader with insights on the appropriate role for generic units in the design process.

Chapter 17 discusses the predefined file input and output facilities of the Ada language. Until this point, all input and output is performed using a simplified package, `Basic_IO`, whose text is given in the Appendix. `Basic_IO` provides a subset of the predefined facilities, consisting of the rudimentary facilities required for interesting examples and exercises. The Ada language's predefined input and output facilities consist not of additional language features, but rather program units written using the features already in the language. A description of these facilities is really a description of a particular set of Ada software components. Therefore, while Chapter 17 contains many details provided for the programmer's reference, it also serves as an extended illustration of the use of packages, generic units, and exceptions.

Chapters 18 and 19 deal with Ada tasks. The reader interested in only a general overview of the language's multitasking features can confine his attention to Chapter 18. That chapter discusses fundamental notions of concurrency and asynchronism, the concept of task objects and task types, and simple rendezvous that do not involve selective waits. Two examples of multitask programming are provided—excerpts from a video game program, to illustrate fundamental real-time programming techniques, and a text processing example, to show how multitasking can simplify apparently sequential problems. Chapter 19 deals with more advanced aspects of tasking, including activation and termination of tasks, selective waits, priorities, entry families, abortion of tasks, and variables shared by multiple tasks. The interaction between tasks and exceptions is also discussed in this chapter.

Chapter 20 deals with low-level and implementation-dependent programming. The approach is necessarily general, since details vary from one Ada compiler to another. Rather than focusing on a particular compiler, we carefully define a variety of hypothetical Ada implementations to illustrate implementation-dependent features and the ways they may differ under different compilers. The hypothetical compilers are for real machines and devices, so the examples are authentic.

NOTES ON STYLE

Ada as a Second Language capitalizes reserved words and uses a mixture of upper and lower case for other identifiers. This differs from the style found in most publications about the Ada language, but it is a more appropriate style for the practicing programmer. The habit of writing identifiers entirely in upper case is a vestige of the day when all programs were entered by keypunch. Given modern-day keyboards, it makes sense to reserve the use of upper case for highlighting abbreviations and the beginnings of words. By mixing upper and lower case, we make our identifier names more expressive and informative. Reserved words should be written in a way that distinguishes them from other identifiers and makes them stand out, so that the underlying program structure is apparent. In many publications, this is done by setting the reserved words in boldface type. For the many programmers who do not have boldface available to them, however, this effect is best achieved by writing reserved words entirely in upper case.

The text uses masculine pronouns in a generic sense, in accordance with conventional English practice. For example, we speak of how a programmer can hide the data in *his* package from other programmers. I am confident that no reader will seriously interpret this as an assertion that women are not programmers. Indeed, as Chapter 1 explains in greater detail, the Ada language is named after the woman generally regarded as the world's first programmer.

ACKNOWLEDGMENTS

This book has benefited from the detailed attention that several devoted friends have paid to it. The comments of Christine Ausnit, Fredric Cohen, and Frank Pappas led to significant improvements. I am further honored to have had the manuscript reviewed by two of the Distinguished Reviewers who officially reviewed the design of the Ada language itself—John Goodenough and Nico Lomuto. Nico deserves special thanks, both for excellent pedagogical and technical advice and for his enthusiastic encouragement. Finally, I am indebted to Jorge Rodriguez for making the facilities of SofTech, Inc., available to me to compile the book's examples.

Ada as a Second Language is the outgrowth of a twenty-session continuing education course on the Ada language. I told my wife and son that, with a few months' effort, I could transform my lecture notes into a textbook. As soon as I sat down and started writing, taking care to introduce underlying concepts and to describe language features completely, it became obvious that I had grossly underestimated the effort involved. Perhaps a textbook on the Ada language can be written in so short a time (in fact I suspect some have been), but not a textbook worth reading. Three years and two daughters later, as I complete this "few months' effort," I thank Dianne for her patience when I needed patience, for her impatience when I needed prodding, and for her loving support throughout. The role of author's spouse is not an enviable one, but Dianne accepted the burdens, above and beyond her already awesome responsibilities, with composure and valor.

Norman H. Cohen

此为试读，需要完整PDF请访问：www.ertongbook.com

CONTENTS

Preface xvii

Chapter 1 The Development of the Ada Language 1

1.1 The Defense Department's Software Crisis 1
1.2 The Effort to Find a Standard Language 2
1.3 The Design of the Ada Language 4
1.4 Other Aspects of the Ada Effort 6
1.5 Summary 7

Chapter 2 An Introduction to the Ada Language 9

2.1 The Text of an Ada Program 9
2.2 Notation for Ada Syntax 16
2.3 An Overview of the Ada Language 16
2.3.1 Compilation Units 16
2.3.2 Subprograms and Main Programs 18
2.3.3 Statements 19
2.3.4 A Closer Look at Subprograms 22
2.3.5 Defining New Types 25
2.3.6 Extending the Meanings of Operators 28
2.3.7 Handling Unexpected Situations 29
2.3.8 Packages 31
2.3.9 Private Types 36
2.3.10 Generic Program Units 38
2.3.11 Concurrent Computations 42
2.3.12 Low-Level Interfaces 44
2.4 Summary 45

Chapter 3 Programmer-Defined Data Types 50

3.1 What Is a Data Type? 50
3.2 Treatment of Data Types in the FORTRAN, PL/I, Pascal, and Ada Languages 54

3.3 Classes of Ada Data Types 59
3.3.1 Integer Types 60
3.3.2 Floating-Point Types 60
3.3.3 Fixed-Point Types 61
3.3.4 Enumeration Types 61
3.3.5 Array Types 62
3.3.6 Record Types 62
3.3.7 Access Types 62
3.3.8 Task Types 62
3.3.9 Private Types 63
3.4 Summary 63

Chapter 4 Object and Type Declarations 65
4.1 Object Declarations 66
4.2 Type Declarations 68
4.2.1 Integer Type Declarations 69
4.2.2 Floating-Point Type Declarations 71
4.2.3 Fixed-Point Type Declarations 73
4.2.4 Enumeration Type Declarations 74
4.2.4.1 Manipulation of Enumeration Types 74
4.2.4.2 Overloading Enumeration Literals 75
4.2.4.3 Predefined Enumeration Types 76
4.2.5 Discrete Types 77
4.2.6 Array Type Declarations 79
4.2.6.1 Constrained Array Types 79
4.2.6.2 Unconstrained Array Types 82
4.2.7 Record Type Declarations 86
4.3 Details 90
4.3.1 Static and Dynamic Expressions 90
4.3.2 Numeric Literals and Universal Expressions 91
4.3.3 Expressions Allowed in Declarations 92
4.3.4 Predefined Numeric Types and Portability 95
4.3.5 Enumeration Types with Character Literals 96
4.3.6 More on Array Types 98
4.3.6.1 Null Arrays 98
4.3.6.2 Multidimensional Arrays Versus Arrays of Arrays 99
4.3.6.3 "One of a Kind" Arrays 99
4.4 Summary 101

Chapter 5 Access Types 109
5.1 Access Type Declarations 110
5.2 Dynamic Allocation 111
5.3 Naming Dynamically Allocated Variables 113
5.4 The Use of Access Types 117
5.4.1 Variables with Changing Roles 117
5.4.2 Shared Data 119
5.4.3 Reducing Copying of Data 122
5.4.4 Pointing to Arrays of Various Sizes 124
5.4.5 Recursive Data Types 126

5.5 Access Types and Index Constraints 130
5.6 Summary 131

Chapter 6 Subtypes and Type Equivalence 135
6.1 Type Restrictions on the Use of Data 135
6.2 Subtypes 136
6.2.1 Subtype Declarations 138
6.2.2 Compatibility Between Typemarks and Constraints 139
6.2.2.1 Range Constraints 140
6.2.2.2 Accuracy Constraints 142
6.2.2.3 Index Constraints 142
6.2.3 Discrete Ranges of Subtypes 143
6.2.4 Subtypes of Array and Record Components 145
6.2.5 Predefined Subtypes 146
6.3 Differences Between Types and Subtypes 146
6.4 Derived Types 151
6.5 Summary 152

Chapter 7 Statements 156
7.1 Assignment Statements 157
7.2 Procedure Call Statements, Input, and Output 158
7.2.1 Procedure Call Statements 158
7.2.2 Input and Output 158
7.3 IF Statements and CASE Statements 160
7.3.1 IF Statements 161
7.3.2 CASE Statements 163
7.4 NULL Statements 166
7.5 LOOP Statements 167
7.5.1 WHILE Loops 167
7.5.2 FOR Loops 168
7.5.3 Basic Loops 170
7.6 EXIT Statements 171
7.7 GOTO Statements 177
7.8 DELAY Statements 180
7.9 Block Statements 180
7.10 Summary 183

Chapter 8 Expressions 188
8.1 Elementary Expressions 189
8.1.1 Literals 190
8.1.1.1 Numeric Literals 190
8.1.1.2 String Literals 190
8.1.1.3 Enumeration Literals 191
8.1.1.4 The Literal NULL 191
8.1.2 Names 191
8.1.3 Attributes 192
8.1.4 An Extended Example 197
8.1.5 Aggregates 202
8.1.5.1 Array Aggregates 202
8.1.5.2 Record Aggregates 206

8.1.6 Qualified Expressions 208
8.1.7 Type Conversions 209
8.1.8 Allocators 211
8.1.9 Function Calls 211
8.2 Compound Expressions 212
8.2.1 Exponentiation 213
8.2.2 Absolute Value 214
8.2.3 Multiplication and Division 214
8.2.4 Remainder 215
8.2.5 Plus and Minus 215
8.2.6 Catenation 216
8.2.7 Relational Operators and Membership Tests 217
8.2.8 Logical Operators and Short-Circuit Control Forms 221
8.2.8.1 Logical Operators 222
8.2.8.2 Short-Circuit Control Forms 222
8.2.8.3 An Example: Insertion Sort 223
8.2.8.4 Parentheses in Logical Expressions 225
8.2.8.5 Extensions of the Logical Operators 225
8.3 Details 226
8.3.1 Order of Evaluation of Operands 226
8.3.2 Nondecimal Numeric Literals 226
8.3.3 Accuracy of Fixed-Point and Floating-Point Arithmetic 227
8.3.3.1 Model Numbers of a Fixed-Point Subtype 229
8.3.3.2 Model Numbers of a Floating-Point Subtype 230
8.3.4 Attributes of Fixed-Point and Floating-Point Subtypes 232
8.3.5 Static Expressions and Static Subtypes 233
8.3.6 Universal Expressions 237
8.3.7 More On Attributes 238
8.3.8 Base Types 238
8.3.9 More on Aggregates 242
8.4 Summary 243

Chapter 9 Subprograms 253
9.1 Procedure Bodies 253
9.2 Function Bodies 261
9.3 Subprogram Calls 263
9.4 Recursive Subprograms 266
9.5 Overloading Subprograms 279
9.6 Defining New Meanings for Operators 284
9.7 The `Inline` Pragma 291
9.8 Details 293
9.8.1 Nested Subprograms and Global Variables 293
9.8.2 The Process of Passing Parameters 295
9.8.3 More on Overloading 297
9.9 Summary 299

Chapter 10 Record Types with Discriminants 306
10.1 Records with Varying Structure 306
10.2 Declaring Record Types with Discriminants 309
10.2.1 Declaring Record Types with Variants 311

10.2.2 Declaring Record Types with Variable-Length Components 314
10.2.3 Declaring Types Whose Components Have Discriminants 315
10.2.4 Allowable Uses of Discriminants 316
10.3 Discriminant Constraints 317
10.3.1 Initial Values of Discriminants 317
10.3.2 Constrained and Unconstrained Records 318
10.3.3 Subtypes of Record Types 321
10.4 Dynamically Allocated Records with Discriminants 322
10.4.1 Allocators for Records with Discriminants 323
10.4.2 Discriminant Constraints for Access Types 325
10.5 Subprogram Parameters with Discriminants 326
10.6 Details 327
10.6.1 Full Form of a Discriminant Constraint 327
10.6.2 Position of Discriminants Within Records 328
10.6.3 When Expressions Are Evaluated 328
10.7 Summary 330

Chapter 11 Packages 337
11.1 Package Specifications and Package Bodies 338
11.2 The Syntax of Packages 340
11.3 The Placement and Lifetime of a Package 343
11.4 Fractions Revisited 351
11.5 USE Clauses 357
11.6 Renaming Declarations 362
11.7 Derived Subprograms 368
11.8 Packages Without Bodies 370
11.9 Predefined Packages 372
11.10 Summary 375

Chapter 12 Private and Limited Private Types 382
12.1 Abstract Data Types Versus Internal Representations 382
12.2 Declaration of Private Types 384
12.3 The Predefined Package `Calendar` 389
12.4 Deferred Constants 395
12.5 A Private Type for Sets of Integers 397
12.6 Limited Private Types 404
12.7 Private Types with Discriminants 411
12.8 Summary 418

Chapter 13 Separate Compilation 421
13.1 Program Libraries 422
13.2 Compilation Units and Compilations 423
13.3 Subunits 432
13.4 Order of Compilation 441
13.4.1 Top-Down Development and Independent Development 444
13.4.2 Recompilation 446
13.5 Order of Elaboration 449
13.6 Summary 451

Chapter 14 Scope and Visibility 457
14.1 Scope Rules 459
14.2 Visibility Rules 464
14.2.1 Visibility and Overloading 464
14.2.2 Nesting and Hiding 465
14.2.3 Direct Visibility and Visibility by Selection 467
14.2.4 The Effect of USE Clauses 472
14.3 The Package Standard 473
14.4 Summary 474

Chapter 15 Exceptions 479
15.1 Handling Exceptions 480
15.1.1 Exceptions in Sequences of Statements 481
15.1.2 Exceptions Raised by Declarations 486
15.1.3 Exceptions Raised in Handlers 486
15.2 Propagation of Exceptions 486
15.2.1 Propagation from Subprogram Bodies 486
15.2.2 Propagation from Block Statements 487
15.2.3 Propagation from Packages 488
15.3 Predefined Exceptions 488
15.4 Programmer-Defined Exceptions 494
15.5 The Suppress Pragma 500
15.6 A Comparison With PL/I Conditions 505
15.7 Guidelines for the Use of Exceptions 507
15.8 Summary 514

Chapter 16 Generic Units 521
16.1 Templates and Instances 521
16.2 Generic Formal Parameters 532
16.2.1 Generic Formal Objects 533
16.2.2 Generic Formal Subprograms 535
16.2.3 Generic Formal Types 539
16.2.3.1 Generic Formal Parameters for Numeric Types 540
16.2.3.2 Generic Formal Discrete Types 543
16.2.3.3 Generic Formal Array Types 546
16.2.3.4 Generic Formal Access Types 551
16.2.3.5 Generic Formal Private Types 552
16.2.3.6 Generic Formal Types With Discriminants 555
16.2.3.7 Generic Formal Parameters for Arbitrary Numeric Types 556
16.3 Generic Instantiations 558
16.4 Defaults for Generic Parameters 560
16.5 Generalization 564
16.6 Predefined Generic Units 569
16.7 Summary 573

Chapter 17 Predefined Input and Output 582
17.1 Basic Concepts 583
17.1.1 External and Internal Files 583

此为试读，需要完整PDF请访问：www.ertongbook.com