Programming in Ada 2012

Programming in Ada 2012

JOHN BARNES

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of
education, learning and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781107424814

First published 2014
Reprinted with corrections and additions 2021

Typeset By John Barnes Informatics

A catalogue record for this publication is available from the British Library

Library of Congress Cataloguing in Publication data

ISBN 978-1-107-42481-4 Paperback

To Barbara

Contents

Foreword

Programming Languages and Software practice are always engaged in a game of leapfrog: a forward looking programming language introduces new ways of thinking about software development, and its constructs shape the way programmers think about their craft; creative programmers invent new idioms and patterns to tackle ever more complex programming tasks, and these idioms become incorporated in the next generation of programming languages.

The latest version of Ada, whose description we owe once again to the inimitable expository talents of John Barnes, has exemplified this dynamic repeatedly over the last 30 years.

- Ada 83 showed programmers how programming in the large should be organized (packages, strong typing, privacy) and convinced them that indices out of range were not a common pitfall of programming but elementary errors that could be controlled with proper declarations and constraint checking. Ada 83 also put concurrent programming in a mainstream programming language.
- Ada 95 benefited from a decade-long development in object-oriented programming techniques, and successfully grafted the ideas of polymorphism and dynamic dispatching onto a strongly-typed language with concurrency. It enhanced programming-in-the large capabilities with child units and their generic incarnations.
- Ada 2005 showed how data-based synchronization (protected types) and concurrency (task types) could be unified through a novel use of interface inheritance, and adopted a conservative model of multiple inheritance of interfaces that has proved more robust than the more unrestricted models of MI. Ada 2005 also introduced into the language an extensive container library, following here the example of other established languages and many earlier experimental high-level languages that showed the usefulness of reasoning over data aggregates.

And now – Ada 2012, the latest version of the language whose description you are holding, reflects both aspects of this dialectic process: it introduces new ways of thinking about program construction, and it reflects developments in software practice that hark back to the earlier days of our profession but that have seldom, if ever, found their way into well-established programming languages.

The general rubric for these new/old ideas is Programming by Contract. The term became well-known through the pioneering work of Bertrand Meyer and the design of Eiffel, but it probably found more significant use in the SPARK community, in the design of critical software for applications that require a real measure of formal verification for their deployment.

Ada 2012 offers the programmer a wealth of new tools for specifying the intent of a program: preconditions, postconditions, type invariants, subtype predicates. All of these allow the software architect to present more clearly the intent of a piece of software, and they allow the compiler and/or the run-time system to verify that the software behaves as intended. The use of pre- and postconditions was proposed a generation ago by E. Dijkstra and C.A.R. Hoare, but their pioneering efforts were not widely adopted by the software community, among other things because good language support for these mechanisms was lacking. Their introduction in a language whose user community is particularly concerned with mission-critical software reflects the fact that concerns about safety and security are more urgent than ever. We can expect that these techniques will be adopted early and enthusiastically by the aerospace and automotive software development community, as they have been in the small and dedicated SPARK community.

Preconditions, postconditions, type invariants and type predicates are logical assertions about the behavior of a given construct. When these assertions involve data aggregates (vectors, sets, and other container types) it is particularly convenient to use notations from first-order logic, namely quantified expressions.

An important syntactic innovation of Ada 2012 is the introduction of quantified expressions both in their universal form (all elements of this set are French Cheeses) and their existential form (some element of this vector is purple). As a result, the language includes the new keyword **some**. These quantified expressions are of course implicit loops over data aggregates, and in parallel with their introduction, Ada 2012 has extended considerably the syntax of iterators over containers. A generalized notion of indexing now allows the programmer to define their own iterable constructs, as well as mapping between arbitrary types.

Contracts, quantified expressions, and generalized indexing may appear to be miscellaneous additions to an already large language; in fact they are elegantly unified under the umbrella of a new construct: the Aspect Specification, which also generalizes and unifies the earlier notions of attributes and pragmas. The coherence of the language has thus been enhanced as well.

Programming languages must also respond to developments in Computer Hardware. The most significant development of the last decade has been the appearance of multicore architectures, which provide abundant parallelism on a single chip. Making efficient use of the computer power now available on a single processor has been the goal of much development in language design. Ada 2012 provides tools for describing multicore architectures, and for mapping computing activities onto specific cores or sets of them.These are novel capabilities for a general-purpose programming language, and we can expect them to have a profound impact on the practice of parallel programming.

This thumbnail description of the high points of the new version of the language is intended to whet your appetite for the pages that follow. Once again, John Barnes has provided a wonderfully lucid, learned, and insightful description of the latest version of Ada. He has been the tireless explicator of the design and evolution of the language over more than three decades, and the Ada community has acquired its understanding and love of the language from his prose. A programming language is a tool for thought, and most Ada users have learned how to think about programs from John Barnes's books. I can only hope that the widest possible audience will learn to think straight from the exciting descriptions that follow.

The design of Ada 2012 is once again the result of the collective effort of the Ada Rapporteur group, an extremely talented group of language designers who combine deep industrial experience with an equally deep knowledge of programming language semantics and theoretical computer science. The ARG, of which John Barnes has been an invaluable member from the inception of Ada, has once again created a modern and elegant programming language that addresses the needs of a new generation of software designers. It has been an enormous privilege to work with them. I trust the reader will enjoy the result of their work for years to come. Happy Programming!

Ed Schonberg
AdaCore
Chairman, Ada Rapporteur Group
New York, March 2014

Preface

Welcome to *Programming in Ada 2012* which has been triggered by the recent ISO standardization of Ada 2012.

The original language, devised in the 1980s, is known as Ada 83 and was followed by Ada 95, Ada 2005, and now Ada 2012. Ada has gained a reputation as being the language of choice when software needs to be correct. And as software pervades into more areas of society so that ever more software is safety critical or security critical, it is clear that the future for Ada is bright. One observes, for example, the growth in use of SPARK, the Ada based high integrity language widely used in areas such as avionics and signalling.

Ada 83 was a relatively simple but highly reliable language with emphasis on abstraction and information hiding. It was also notable for being perhaps the first practical language to include multitasking within the language itself.

Ada 95 added extra flexibility to the strongly typed and secure foundation provided by the Software Engineering approach of Ada 83. In particular it added the full dynamic features of Object Oriented Programming (OOP) and in fact was the first such language to become an ISO standard. Ada 95 also made important structural enhancements to visibility control by the addition of child units, it greatly improved multitasking by the addition of protected types, and added important basic material to the standard library.

Ada 2005 then made improvements in two key areas. It added more flexibility in the OOP area by the addition of multiple inheritance via interfaces and it added more facilities in the real-time area concerning scheduling algorithms, timing and other matters of great importance for embedded systems. It also added further facilities to the standard library such as the ability to manipulate containers.

Ada 2012 makes further important enhancements. These include features for contracts such as pre- and postconditions, tasking facilities to recognize multicore architectures, and major additions and improvements to the container library.

In more detail, the changes include

- Contracts – pre- and postconditions, type invariants, and subtype predicates are perhaps the most dramatic new features. The introduction of these features prompted a rethink regarding the specification of various properties of entities in general. As a consequence the use of pragmas has largely been replaced by the elegant new syntax of aspect specifications which enables the properties to be given with the declaration of the entities concerned.

- Expressions – the introduction of the contract material showed a need for more flexible expressions. Accordingly, Ada now includes conditional expressions, case expressions, quantified expressions and more flexible forms of membership tests. A new form of function is also introduced in which the body is essentially given by a single expression.
- Structure and visibility – perhaps the most startling change in this area is allowing functions to have parameters of all modes. This removes the need for a number of obscure techniques (dirty tricks really) which had been used. Other important improvements concern incomplete types.
- Tasking – Ada 2012 has new features describing the allocation of tasks to individual processors and sets of processors; these additions were prompted by the rapid growth in the use of multicore architectures.
- Generally – new flexible forms of iterators and dereferencing are introduced mainly for use with containers. Better control of storage pools is permitted by the introduction of subpools.
- Predefined library – some improvements are made concerning directories and a feature is added for the identification of locale. However, the most important improvement is the addition of many new forms of containers. These include multiway trees and task-safe queues. There are also bounded forms of all containers which are important for high integrity systems where dynamic storage management is often not permitted.

This book follows the tradition of its predecessors. It presents an overall description of Ada 2012 as a language. Some knowledge of the principles of programming is assumed but an acquaintance with specific other languages is by no means necessary.

The book comprises 27 chapters grouped into four parts as follows

- Chapters 1 to 4 provide an overview which should give the reader an understanding of the overall scope of the language as well as the ability to run significant programs as examples – this is particularly for newcomers to Ada.
- Chapters 5 to 11 cover the small-scale aspects such as the lexical details, scalar, array and simple record types, control and expression structures, subprograms and access types.
- Chapters 12 to 22 discuss the large-scale aspects including packages and private types, contracts, separate compilation, abstraction, OOP and tasking as well as exceptions and the details of numerics.
- Chapters 23 to 27 complete the story by discussing the predefined library, interfacing to the outside world and the specialized annexes; there is then a finale concluding with some ruminations over correctness and a brief introduction to SPARK.

The finale includes, as in its predecessors, with the fantasy customer in the shop trying to buy reusable software components and whose dream now seems as far away or indeed as near at hand as it did many years ago when I first toiled at this book. The discussion continues to take a galactic view of life and perhaps echoes the cool cover of the book which depicts the Ice Comet.

Those familiar with *Programming in Ada 2005* might find the following summary of key changes helpful

- The number of chapters has grown from 25 to 27. The new chapter 8 covers the new forms of expressions such as if expressions, case expressions, and quantified expressions. The new chapter 16 discusses the material on contracts. The chapter on containers (now chapter 24) has grown because of the introduction of containers for multiway trees, single indefinite objects, and various forms of queues. A number of existing chapters have additional sections such as that on aliasing.
- The revisions to produce Ada 2012 have impacted to a greater or lesser extent on many aspects of the language. Most chapters conclude with a checklist summarizing important points to remember and listing the main additions in Ada 2012.
- As a consequence the book is now some 120 pages longer. It would have been even longer had I not decided that it was unnecessary to include the answer to every exercise. Accordingly, the printed answers cover just the introductory chapters (for the benefit of those entirely new to Ada) and those exercises that are referred to elsewhere in the book. But all the answers are on the associated website.

The website also includes the six sample programs both in text form and as executable programs, some material from earlier versions of this book which now seem of lesser importance but which I nevertheless was reluctant to lose completely. More details of the website will be found below.

And now I must thank all those who have helped with this new book. The reviewers included Janet Barnes, Alan Burns, Rod Chapman, Jeff Cousins, Bob Duff, Stuart Matthews, Ed Schonberg, Tucker Taft, and Andy Wellings. Their much valued comments enabled me to improve the presentation and to eliminate a number of errors. Some of the new material is based on parts of the *Ada 2012 Rationale* and I must express my special gratitude to Randy Brukardt for his painstaking help in reviewing that document.

Finally, many thanks to my wife Barbara for help in typesetting and proof-reading and to friends at Cambridge University Press for their continued guidance and help.

John Barnes
Caversham, England
March 2014

The 2016 update

Whenever a new standard appears and is put into use, it is almost inevitable that various imperfections are soon discovered. Ada 2012 was no exception and a number of corrections and improvements were deemed to be necessary. These alterations were processed in the usual manner by the Ada Rapporteur Group and resulted in a Corrigendum which was approved and published by ISO in February

2016. This updated reprint of Programming in Ada 2012 accordingly covers this revised 2016 standard.

The revisions are described by Ada Issues (AIs) and a number of these are mentioned in the Index. A few early issues were actually captured in the original printing and some form the material in Section 16.5 regarding the output of messages arising from the violation of predicates. Another early change concerned the package Ada.Locales whose original form quite frankly did not work at all.

Many of the recent issues relate to improving the performance and description of contracts which were perhaps the most important addition in Ada 2012. Another improvement is that I have taken this opportunity to replace the use of library pragmas such as Pure and Preelaborate by the corresponding aspect clauses. This rearrangement saves space and clutter and will be particularly beneficial when a number of other matters are addressed by what I hope will be Ada 2022.

As well as including new material, I have taken the opportunity to correct all errors that have been discovered. I am particularly grateful to a number of readers who have pointed out errors and I must especially thank Pascal Pignard and Tama McGlinn for their many messages and to Jeff Cousins for his help in making corrections. As well as the technical errors there were a lot of unfortunate errors in the cross references which have also been corrected.

John Barnes
Caversham, England
February 2021

Notes on the website

The website is www.cambridge.org/barnes. It contains three main things: the full answers to all the exercises, some obscure or obsolete material on exceptions, discriminants, and iterators which were in previous versions of the book, and additional material on the six sample programs.

I do hope that readers will find the sample programs on the website of interest. I am aware that they are a bit intricate. But this seems almost inevitable in order to illustrate a broad range of features of Ada in a reasonably concise manner. However, in most cases they build on examples in preceding chapters and so should not be difficult to follow.

Each example commences with some remarks about its purpose and overall structure. This is followed by the text of the program and then some notes on specific details. A desire to keep the program text short means that comments are at a minimum. However, the corresponding source text on the website includes much additional commentary. The website also includes further discussion and explanation and suggestions for enhancement. In general the programs use only those features of the language explained in detail by that point in the book.

The first program, Magic Moments, illustrates type extension and dispatching. It shows how the existence of common components and common operations enable dispatching to compute various geometrical properties almost by magic.

The Sylvan Sorter is an exercise in access types and basic algorithmic techniques including recursion.

The Rational Reckoner provides two examples of abstract data types – the rational numbers themselves and the stack which is the basis of the calculator part of the program.

The Super Sieve illustrates multitasking and communication between tasks both directly through entry calls and indirectly through protected objects. For added interest it is made generic so that more general primes than the familiar integers may be found. This provides the opportunity to use a discriminated record type and a modular type to represent binary polynomials.

The program Wild Words is probably the hardest to follow because it is not based on any particular example described in the preceding chapters. It illustrates many of the facilities of the character and string handling packages as well as the generation of random numbers.

The final program, Playing Pools, shows how users might write their own storage allocation package for the control of storage pools. The example shown enables the user to monitor the state of the pool and it is exercised by running the familiar Tower of Hanoi program which moves a tower of discs between three poles. Variety is provided by implementing the stack structures representing the three poles (and defined by an interface) in two different ways and dispatching to the particular implementation. The website includes an extended version which uses three different ways.

Information on many aspects of Ada such as vendors, standards, books and so on can be obtained from the websites listed in the Bibliography.

The website also includes a list of corrections and additions made by this reprint.