The ADA Programming Language

I.C. Pyle

The ADA Programming Language

A GUIDE FOR PROGRAMMERS

I.C. PYLE
Department of Computer Science
University of York, England

Prentice Hall PHI International

ENGLEWOOD CLIFFS, NEW JERSEY LONDON NEW DELHI
SINGAPORE SIDNEY TOKYO TORONTO WELLINGTON

British Library Cataloguing in Publication Data

Pyle, Ian C
The Ada programming language.
1. Ada (Computer program language).
I. Title
001.6'42 QA76.73.A35
ISBN 0-13-003921-7

Library of Congress Cataloguing in Publication Data

Pyle, Ian C 1934-
The Ada Programming Language
1. Ada (Computer program language)
I. Title.
QA76.73.A35P94 001.64'24 80-25034
ISBN 0-13-003921-7 (pbk)

ISBN 0-13-003921-7

PRENTICE-HALL INTERNATIONAL, INC. *London*
PRENTICE-HALL OF AUSTRALIA PTY. LTD., *Sydney*
PRENTICE-HALL OF CANADA, LTD., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA PTE., LTD. *Singapore*
PRENTICE-HALL INC., *Englewood Cliffs, New Jersey*
WHITEHALL BOOKS LIMITED, *Wellington, New Zealand*

10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

To Margaret

Preface

Ada is a new programming language, sponsored by the United States Department of Defense, designed under the leadership of Jean D. Ichbiah. The language is a major advance in programming technology, bringing together the best ideas on the subject in a coherent way designed to meet the real needs of practical programmers. It is the first result of a substantial effort to identify the requirements for programming and satisfy them effectively.

This book is written primarily for practicing programmers of embedded computer systems, giving a full presentation of the power of Ada to those whose working environment will be greatly changed by it during the next few years. Other readers will include programmers of non-embedded systems, for whom most of the facilities will be relevant, and teachers of programming who will benefit from the breadth and coherence of Ada's facilities. In addition, the book should be of value to managers of programming projects, since Ada strongly assists the development of large programs.

In keeping with the primary aim of the book, the style of presentation presumes a knowledge of programming. Topics are introduced in the context of embedded computer systems, in an order which reflects the normal pattern used in programming. This is not necessarily the best order for introductory teaching of the skill of programming itself.

Chapters 1 to 5 cover the basic features of Ada, which any programmer needs to know. The subsequent chapters deal with more advanced features, which should only be studied after the basic features are thoroughly understood. Chapters 6 to 8 deal with particular programming concepts in Ada which will probably be new to most programmers. Chapters 9 and 10 cover the issues of program structure, which take traditional ideas as the start, but make significant extensions. Chapter 11 deals with machine specific issues, and shows how they can be expressed in a machine independent language. Chapters 12 and 13 give an advanced treatment and more details on topics introduced previously. The appendices contain certain notes and definitions of various particular items in Ada.

The official definition of Ada is the Language Reference Manual and Formal Definition, not this book. It is possible that the language may be changed in the light of experience and discovered problems; also, it is to be submitted for formal standardization, and this may require changes at some time in the future. This book differs from an official definition in that the official definition of a language must satisfy both programmers and compiler-writers, two very different kinds of reader. Unfortunately, this usually means that the definition goes barely far enough to satisfy the compiler-writers, and tends to have complications of notation and formality which make it unsatisfactory for programmers. This book is intended to explain the language to programmers who wish to learn Ada without having to become compiler specialists.

Most of the chapters finish with some programming exercises, which readers may use to test their understanding of the ideas presented. The solutions to these exercises are given elsewhere in the book, as examples of other aspects of Ada programs.

As most readers will know, the language is named after a real person, Augusta Ada, Countess of Lovelace, who first programmed a computer, before either computers or programming had been recognized as such.

I wish to thank the many people who have helped me during the course of preparing this book, particularly Ian Wand and Brian Wichmann. The syntax diagrams were produced by a program written by Colin Runciman and David Keeffe. Most of the typing was done by Val Fry.

Most of all, I thank my family for their encouragement and support while Ada has been living with us.

March 1981 I.C.P.

Contents

CHAPTER 1

Introduction

Ada is for programming embedded computer systems - that is, systems in which a computer is directly connected to some apparatus or plant which it monitors and/or controls. This means that Ada can be used for conventional programming (which actually accounts for the majority of embedded computer system programming) and also for the special technical requirements concerning input/output, timing relationships, contingency programming to cope with errors, and long-term maintenance.

Embedded computer systems range from intelligent terminals and smart instrumentation to air traffic control or factory automation, via laboratory data monitoring, numerically controlled machine tools, navigation and guidance systems, stored program controlled telephone exchanges, batch and continuous production control, environmental monitoring, and future domestic products containing microcomputers. The computer involved may be large or small, single or a collection of many processors, or part of a computer network.

It is expected, however, that the program concerned in each system would have a lifetime of several years, and consequently that people other than the original programmers would be involved in maintaining it. This concern for maintenance underlies much of the style of Ada.

Ada gives special attention to the ease of reading and understanding programs - it is based on the realisation that it is more important to be able to read a program and understand it clearly than to be able to write it quickly or briefly. We therefore tend to use fairly long names and identifiers in an Ada program, and state the assumptions which the design of the program implies. The reason for this is that the writer of the program does his job once, but maintainers of the program may have to read the program many times throughout its life.

1.1 An Ada program

Programs in Ada specify not only the actions inside computers, but also the interactions between the computers and the environment in which they are embedded. Since the interactions with the environment can be quite tricky to program, it is usual to design separate pieces of program to deal with the various kinds of input/output devices, and the resulting pieces of program are kept in libraries. For a simple Ada program, we use an existing library package to handle the input/output, and specify the particular actions we want by calling on facilities made available by the package.

In this first example we show a trivial program in Ada. The program is written as a procedure, which specifies the actions to be carried out. In an Ada development environment, many procedures will be held in a library, where they are available for use in other programs. In practice, all programs are likely to refer to the library for units defining many commonly required actions such as input/output and mathematical functions. A package called STANDARD is always available; it is specified in Appendix A. For this example, we use a library package called TEXT_IO. (Its definition is also given in Appendix A. It is significant that the package specification is itself written in Ada.) A program must begin by listing the units it needs; these will be extracted from the library by the translator.

```
with TEXT_IO; use TEXT_IO;
procedure EX_1 is
  pragma MAIN;
begin
  NEW_LINE;
  PUT ("Hello");
end EX_1;
```

The program is written using special key-words such as **with**, **procedure**, **begin** and **end**, together with other words such as TEXT_IO, PUT. The keywords are fixed for all Ada programs, and show the structure of the program. The other words are called identifiers, and are invented by the programmers to denote the particular entities concerned in the program.

This program needs the unit TEXT_IO (and no other), which defines the procedures NEW_LINE and PUT (among others), and sets up input/output files on suitable devices. The program prints the message

```
Hello
```

on a new line on the output device. Notice that the program has a name EX_1 which is given at the beginning and end, so that the body of the program is clearly delimited.

1.2 Another program

This is a slightly more complicated program, but still very trivial and unrealistic. It adds together two simple integers.

```
with TEXT_IO; use TEXT_IO;
procedure EX_2 is
  pragma MAIN;
  A, B : INTEGER range 0 .. 999;
begin
  GET (A); GET (B);
  NEW_LINE;
  PUT ("The sum of");
  PUT (A); PUT ("and"); PUT (B);
  PUT ("is"); PUT (A + B);
end EX_2;
```

Notice that the names A and B are used to hold the values which are read in by the procedure GET (also defined in TEXT_IO). In order that the program can know what to expect for the values before they are used, the type and range for each must be declared at the head of the program. The package STANDARD includes the definition of INTEGER and "+". This package is automatically available for every program unit.

1.3 A real program

Any program for a genuine embedded computer system will be quite large, and may be written as a collection of separate units in order that it can be maintained effectively. Here is one unit of a program to control a filtration plant (see figure 1a). River water is pumped through a filter and clean water is delivered. After a time, the filter gets clogged with debris and has to be cleaned by blowing air through and draining out the sump; filtering can then resume. Occasionally a fault in the valves may make it necessary to close down the whole plant.

```
with MAJOR_PHASES; use MAJOR_PHASES;
procedure SINGLE_FILTER is
begin
  START_UP;
```

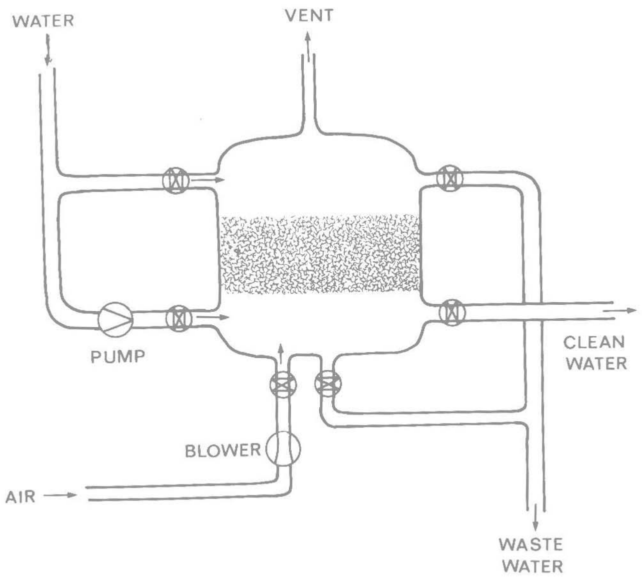

Figure 1a: Filtration Unit

```
      loop
        DELIVER_WATER;
        CLEAN_FILTER;
      end loop;

    exception
      when others =>        -- FAULT or other trouble
        CLOSE_DOWN;
    end SINGLE_FILTER;
```

Further details of the filtration unit are given in later examples.

Notice that this program unit shows how the action of SINGLE_FILTER is achieved in terms of application-specific procedures. These would be expressed in a separate unit that contains the specifications of the other procedures etc.:

```
    package MAJOR_PHASES is
      procedure START_UP;
      procedure DELIVER_WATER;
      procedure CLEAN_FILTER;
      procedure CLOSE_DOWN;
      FAULT : exception;
    end MAJOR_PHASES;
```

The details (bodies) of the procedures such as START_UP are written in a corresponding package body.

1.4 Form of an Ada program

The text of the Ada program consists mainly of two kinds of words and various punctuation marks. Words like **procedure**, **is**, **separate**, **end** (conventionally written in small letters) are reserved for special uses in Ada, and determine the main structure of the program. These are known as keywords. The other words, like SINGLE_FILTER, START_UP, FAULT, CLOSE_DOWN (conventially written in capital letters) are invented by the programmer, to denote the various entities in the program; these words are technically called identifiers. They must always be different from the Ada keywords.

As well as the main text of the program, whose structure is prescribed by the Ada language, there may be comments on any line, introduced by a double hyphen. Comments may contain any characters without restriction, for the rest of the line. They are used by the programmer to give additional information to the reader of the program, but this is not checked in any way by the compiler.

Another special construct in an Ada program is called a **pragma**: this is a phrase used to give information to the compiler about translating the program. A main program is written in Ada as a procedure, but marked:

```
pragma MAIN;
```

Pragmas do not affect the meaning of a program, but may affect the way it is implemented (e.g. choice of optimisation). The possible pragmas in Ada are listed Appendix D.

1.5 Identifiers and naming

Identifiers are the fundamental creation of the programmer: they name the entities which are needed for the particular program he is designing. An identifier is made up using letters and digits (linked by underline characters) - which must be different from the Ada key words, disregarding the case of the letters. Identifiers may not contain spaces, and may not spread over from one line to another.

The words used to make an identifier should be carefully chosen to be a suitable name for the entity concerned: for example a verb (or verb clause) for a procedure (which denotes an action), and a noun (or a noun clause) for a data object (variable or constant). A type may be named by a suitable abstract noun. Choosing appropriate names is an important aspect of programming. These suggestions are of course not enforced by Ada, and do not constrain the creativity of the programmer. Examples of names are given throughout the book, as we introduce the various kinds of entity that can occur in an Ada program.

1.6 The environment of Ada programs

Ada programs are intended for execution in embedded computer systems - implying significant differences from the usage of conventional programming languages. The differences concern the way Ada programs are developed, and their operational environment. The main consequence is that Ada programs are usually cross-compiled on a host computer, distinct from the target computer for operational use; another important consequence is that an Ada program is likely to be the only program in a computer, with complete responsibility for its activities, not sharing facilities or implying an "operating system" of the conventional kind. The Ada programmer may specify how the various parts of the program interact with one another and with the equipment connected to it.

Some specific differences between Ada and other programming languages are noted in Appendices B (Fortran) and C (Pascal). The translation of an Ada program into its executable form is more than traditional compilation. It includes also the operations of linkage-editing and library module incorporation (which are done separately with other languages), and the provision of run-time facilities to implement the various semantic features of the language such as inter-task communication and dynamic storage allocation.

1.6.1 Ada Programming Support Environment

For the development of Ada programs, an Ada Programming Support Environment is planned. At the time of writing (1980), the requirements and main outline of this environment have been specified, but the actual details have not yet been settled.

Translation of an Ada program involves compilation of the separate units, with cross-checking of interfaces and provision of other required units from a library. The environment includes all the necessary utility