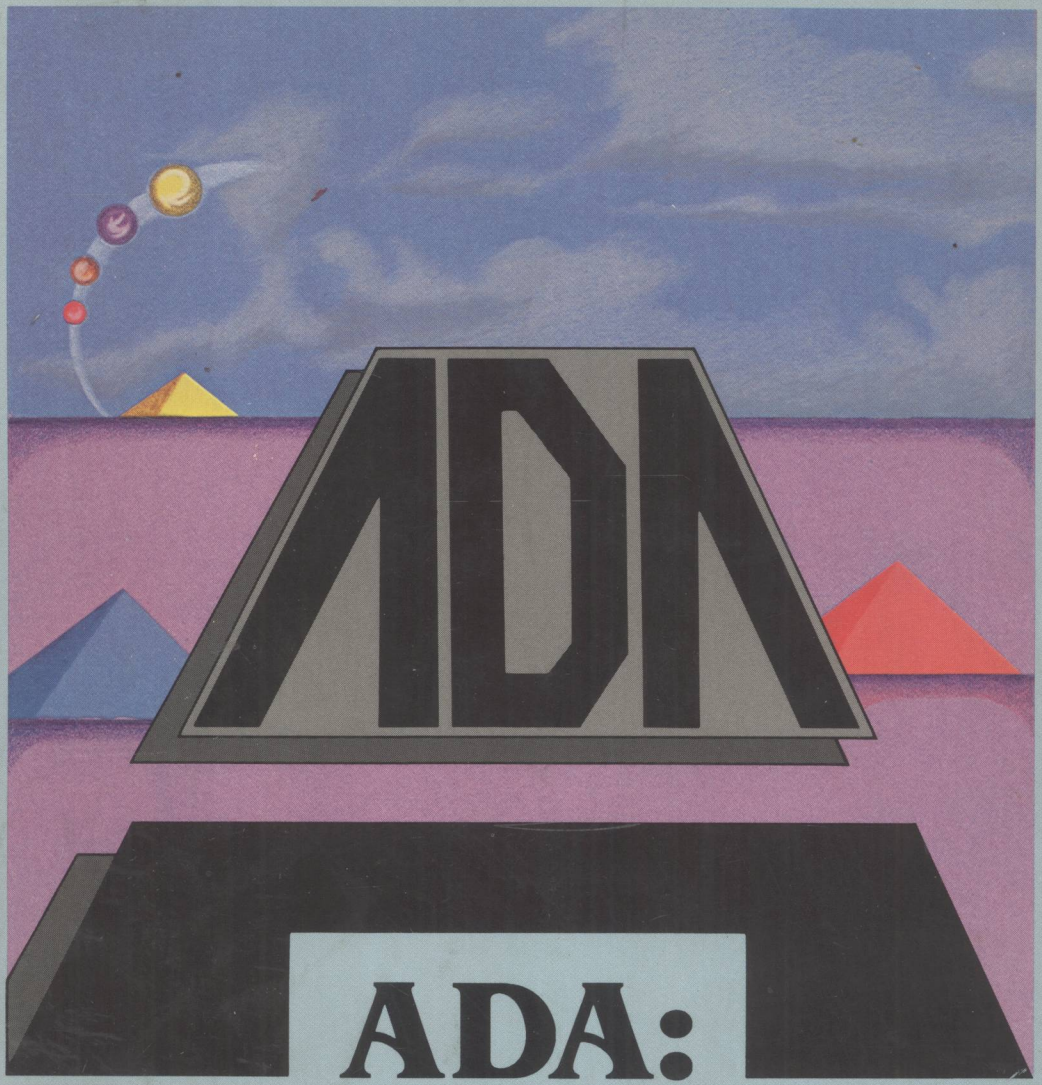
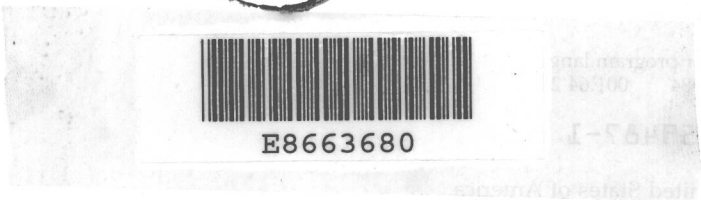


ADA: AN INTRODUCTION

SABINA SAIB

ADA: AN INTRODUCTION

Sabina Saib



HOLT, RINEHART AND WINSTON
New York Chicago San Francisco Philadelphia
Montreal Toronto London Sydney Tokyo
Mexico City Rio de Janeiro Madrid

To my husband, Ihsan,
and my sons,
David and Joseph

The frontispiece photograph is courtesy of National Physical Laboratory, Teddington, England.

Address correspondence to:
383 Madison Avenue, New York, NY 10017

Library of Congress Cataloging in Publication Data
Saib, Sabina.
Ada : an introduction.
Includes index.
1. Ada (Computer program language) I. Title.
QA76.73.A35S25 1984 001.64'24 84-27987

ISBN 0-03-59487-1

Printed in the United States of America

Published simultaneously in Canada

5 6 7 8 039 9 8 7 6 5 4 3 2 1

CBS COLLEGE PUBLISHING
Holt, Rinehart and Winston
The Dryden Press
Saunders College Publishing

Preface

This book provides an introduction to Ada, a modern computer programming language. The book assumes some background and familiarity with computers but not with Ada itself. Central to this text are the many examples and problems. Besides being helpful in learning Ada, these illustrations and exercises can serve as models for you to create your own programs. The examples have all been tested using an Ada compiler. An introductory class may omit the later chapters on more advanced programming techniques while an intermediate programmer may prefer to skim the first chapters.

Some Background on Ada

The programming language Ada was named after Countess Ada Augusta Lovelace (1815–1852) of England. The frontispiece shows a picture painted when she was nineteen, shortly after her marriage to Lord King.

Ada was born into a notable English family. Her father was Lord Byron, the poet, who spent much of his time away from his family. Her mother, Lady Noel Byron, encouraged Ada to develop her talents, one of which was an aptitude for mathematics. Ada studied mathematics and became a friend and collaborator of Charles Babbage (1792–1871), who designed and attempted to build a mechanical digital computer or what he called an "analytical engine."

Ada is best known for her translation of an article about Babbage's computer. She not only translated the article but added her own notes to the paper that included the instructions for a computer program to calculate a table of Bernoulli numbers used in the solution of differential equations. As a result, Ada is known as the world's first computer programmer.

Ada is a high-level programming language with roots in a research language called LIS. Certain features were also borrowed from such modern programming languages as Pascal. The first Ada compiler was developed for a Digital Equipment Corporation computer, the VAX 11/780, by the Courant Institute, New York University in 1983. This compiler, called Ada/ED, was used to test the programs in this book. There are Ada compilers available today for mainframe and minicomputers and also for the most popular microcomputers.

A well-supported language, Ada has been designated by the U.S. Department of Defense as its official computer language. The U.S. government intends to make Ada compilers available on many computers at a nominal charge through the National Technical Information Service. The Ada compiler will be supported with numerous tools to aid in the design, preparation, testing, and maintenance of computer programs written in Ada.

Ada is an American National Standards Institute language, and there is a reference manual available from the U.S. Printing Office. After you have mastered Ada, you may want to order this manual to help you in writing your programs.

To learn more about Countess Ada Augusta Lovelace:

Moore, Doris Langley. *Ada, Countess of Lovelace* New York: Harper & Row, 1977.

Ada's program:

Menabrea, L. F. "Sketch of the Analytical Engine Invented by Charles Babbage (with notes by the translator Ada Augusta, Countess of Lovelace)," in *Babbage's Calculating Engines,* New York: Dover, 1961.

The official Ada reference manual:

U.S. Department of Defense, *Reference Manual for the Ada Programming Language,* ANSI/MIL-STD-1815A (February 1983).

More about LIS:

Ichbiah, J. D., and G. Ferran "Separate Definition and Compilation in LIS and its Implementation." In *Lecture Notes in Computer Science, Cornell Symposium on the Design of Higher Order Languages,* New York: Springer Verlag, 1977.

CONTENTS

CHAPTER 1

INTRODUCTION

Ada is a high-level programming language used to communicate instructions to computers. It would be ideal if we could communicate with computers in English, but computers are not yet as smart as we are, so we need special languages to talk to them. Ada is one of many such special languages.

We call Ada a high-level programming language—this implies that there must also be some lower level programming languages. And indeed there are. While many programmers work with these lower level languages, they have several drawbacks.

One drawback is that working with them is tedious, time-consuming, and error-prone; the languages themselves are often not very meaningful unless you are a programmer trained in one computer's specific *machine language,* the lowest level programming language and the language that speaks directly to the computer's hardware. Programmers communicate with a computer in machine language with a stream of numbers,

```
1000 10 07 00002
1001 20 03 02002
1003 04 03 02003
```

which the computer interprets as an instruction to add, print, store, or so forth.

Another drawback to machine language and other low level languages is that they are machine-specific; a program written for one kind of computer

cannot be used on any other. This also holds true for assembly language, which is the next level above machine language.

Assembly language is a step removed from machine language in that you write your commands using numbers, letters, and short words:

```
ORG 1000
LDA 2
ADD C
STA X
```

Each assembly language command translates into one machine language command, which is made up of the stream of numbers we saw before. The computer program that performs this translation is called an *assembler*.

While an assembly language command such as ADD is more meaningful than the machine number 20, which might also mean *add*, the program is still far from readable. Another drawback to assembly language is that each computer has its own assembler program and machine language. Programs written in assembly language cannot be moved from one kind of computer to another.

There are many higher level languages such as FORTRAN, the forerunner of most of these languages. These languages use commands that are more readable than assembly language, looking something like English and mathematical notation. As with assembly language, these languages also need translators to turn their commands into machine language; these translators are called *compilers*.

Ada is a higher level language similar to FORTRAN. However, because Ada is an American National Standards Institute standard and is designed to be used on a variety of computers, it will be the same across all these computers. There will be no omissions or enhancements in Ada compilers, which

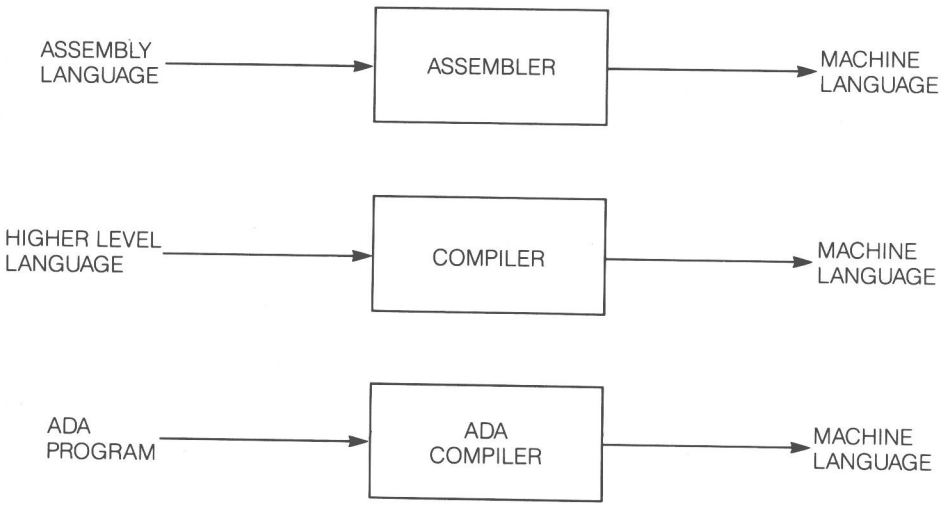


Figure 1.1 Levels of Languages

is not the case with most other programming languages. These enhancements, sometimes called *supersets*, and the omissions, called *subsets*, have made transporting programs across computers difficult.

Another Ada advantage over other high-level programs is that Ada programs use complete English words instead of cryptic abbreviations. This means that programmers can use meaningful names in writing their programs instead of being limited to using very short names. It's still up to programmers to make the programs comprehensible to their colleagues, but Ada at least provides a means for them to do so.

Ada can also detect errors very early in the preparation and testing of a computer program, a process known as *error-checking*. It also possesses powerful constructs for writing programs that programmers previously could write only in assembly language.

Even higher level languages than Ada exist. However, these languages are designed for special application areas such as surveying or structural analysis—Ada was designed for general rather than special applications.

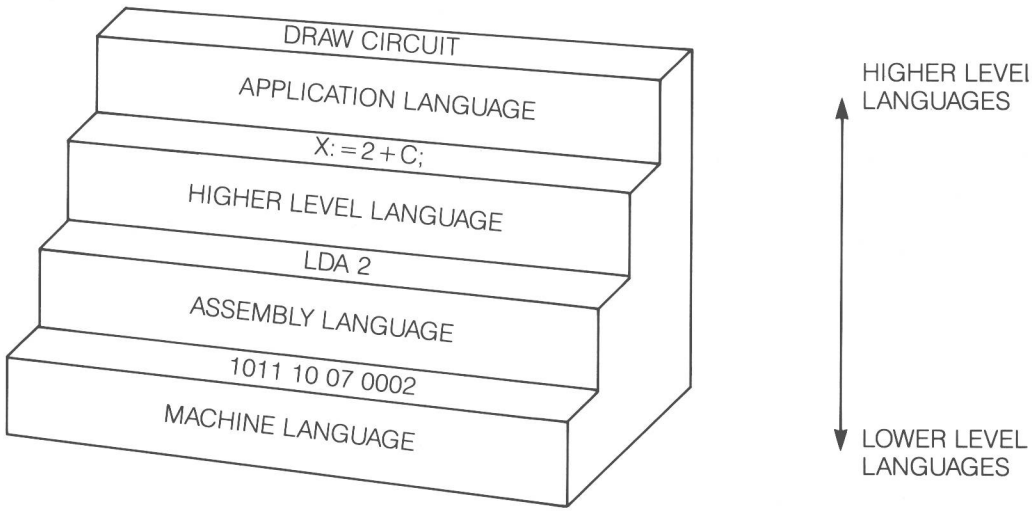


Figure 1.2 Computer Programming Languages

COMPUTER PROGRAMS

A computer program is made up of a series of commands written in a language understandable by a computer. Analogies to computer programs are the instructions you see on the back of a shampoo bottle, the steps in a recipe, or the directions to a friend's house. The shampoo analogy is a good one because the instructions on a shampoo bottle usually ask you to repeat all the steps.

The English instructions that you see on the shampoo bottle make up what is called an *algorithm*, a specific sequence of steps you use to solve a

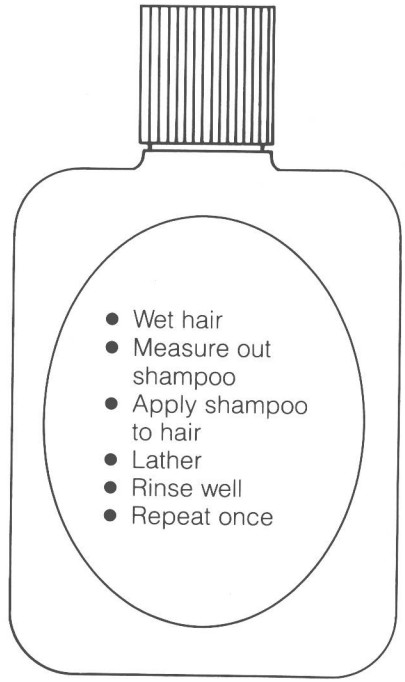


Figure 1.3 Algorithms

problem. You can think of an algorithm as a computer program written in English.

Creating an algorithm is considered more difficult than actually writing the program once the algorithm is given. To design an algorithm, the problem is first divided into relatively large steps, each of which is further subdivided into smaller steps. This subdivision or refinement of the algorithm is very important in developing large programs—this way individual programmers can each work on a small part of the problem and their efforts are then combined. In most algorithms for computer programs, there is a great deal of repetition in the instructions—repetition is something that computers are able to handle very nicely.

The data for the program are also a part of programming. You will have many choices as to how to represent, store, and operate on the information needed by a computer program—the way data are handled in the computer is very important. We will see that decisions on how data are stored and operated on using computer algorithms can make a program easier to understand, can cause a program to compute faster, and can help prevent programming errors, especially when the program is changed.

SIMPLE ADA PROGRAMS

Although you may not be using the same compiler as the one used in the preparation of this text, the programs given in this chapter and throughout the book should word on your computer just as they did when they were prepared. Enter the programs exactly as they are shown. Your computer terminal probably has a TVlike display and a typewriterlike keyboard. You will prepare your Ada program using an editor and the Ada compiler and then execute or run your program and view your results on the display.

A very simple Ada program that you should type into your terminal, translate with the Ada compiler, and execute on your computer is shown below:

```
with TEXT_IO;
use TEXT_IO;
procedure SIMPLE_ADA is
--This is a simple Ada program
begin
--It displays the message within quotes on the terminal
     PUT_LINE('' Hi there, I am a very friendly computer! '');
end SIMPLE_ADA;
```

When compiled and executed this simple program will cause the computer to display, "Hi there, I am a very friendly computer!" on your terminal. Out of the eight lines in the program, only one line causes this to happen. Lines that cause the computer to do something are called *executable lines*. The line that started with PUT_LINE is the single executable line or statement in the program. The remaining lines set up the program.

You'll note that the program was written using a combination of upper- and lowercase letters. Ada does not differentiate between upper- and lowercase in interpreting the program commands. A bold typeface differentiates some important words to Ada called *keywords*. Since you do not have boldface on your terminal, you will type the keywords just like all the other words in your program. The keywords are sometimes called *reserved words;* they have a special meaning to Ada and must be used according to its rules. There are only 63 keywords in Ada, this program used six of them: **with, use, procedure, is, begin,** and **end.**

The first two lines in the program, **with** TEXT_IO and **use** TEXT_IO, appear so often in programs that some Ada compilers assume they exist even though you have not typed them in. These lines tell the compiler that you want to input some information into the computer and receive some output from the program. Almost any program you write will input some information and output some results. This program's output was the message displayed on the terminal, "Hi there, I am a very friendly computer!"

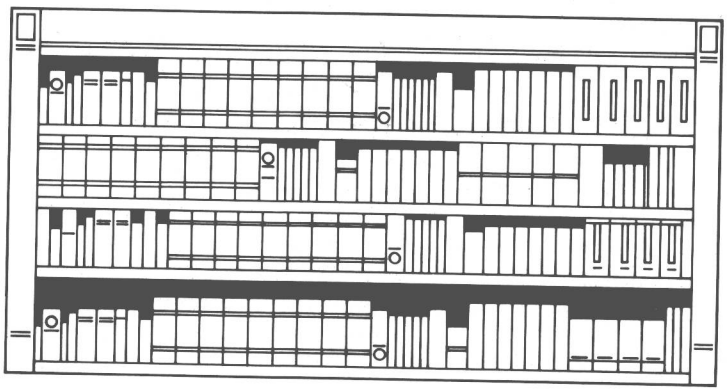
Ada uses programs that have been written and stored in a *program library* for input and output. A program library is made up of library units, each of which is a previously prepared program or, more likely, a collection of previously prepared programs. The **with** statement (the first line in our example)

names one or more program library units—we can use these programs in our program merely by naming them. The name of the program library unit in our example is TEXT_IO. The semicolon following TEXT_IO ends the statement. If a program has a **with** statement, it will always be the first statement to appear in the program. The simple form of the **with** statement is:

with *program library units;*

where the italicized portion of the statement stands for something that must be filled in. We filled it in with one of the standard program library units available with Ada compilers, TEXT_IO. We will learn more about the **with** statement in chapter 10.

The program's next line is the **use** statement, which names a package of programs in the program library unit named in the preceding **with** statement. A package is a collection of related programs and data; packages can also contain packages for organizing programs together in a large collection. For example, think of a law library, which might be part of a larger library system, as a package. A law book from that library would be considered a package in the library. As with the books in a law library, the packages that are part of a larger package should all be related.



```
with a_library;--Ada library of packages
use a_book;   --Ada package in program library

with LAW_LIBRARY;
use LAW_LIBRARY.TORTS;
```

Figure 1.4 Law Library

In our example, we want to access all the packages in the library, so we name the entire package in the **use** statement. The **use** statement is not confined to the beginning of the program and can appear in several places. Until we learn more about it in chapter 10, however, we will employ **use** for input and output and keep it separate in front of the program. The simple form of the use statement is:

use *package name;*

As we saw, semicolons end statements. You can type more than one statement on a line if you wish, but most programs look better if you just type one statement per line. You can also spread out statements across several lines; for example, the first two lines in our program could have been written on one line:

```
with TEXT_IO; use TEXT_IO;
```

But again your program will be more readable if you keep statements to just one line wherever possible.

The third line in our example names the program SIMPLE_ADA. We could have given it any name as long as we started the name with a letter and used only letters, digits, and the underscore in the name. For example, we could have named the program EXAMPLE_1 or FRIENDLY_COMPUTER_PROGRAM or any other name conforming to Ada's rules. It is better to pick meaningful names for your programs than names such as PROGRAM_1 or TODAYS_JOB. The name of your program can be as long as you like so long as it fits on one line of your terminal (usually 80 characters).

In Ada, the instructions that tell the computer what to do and the data used in the program make up a computer program. The instruction or executable part of the program is contained after the keyword **begin** and before the keyword **end.** The data declaration is contained after the keyword **is** and before the keyword **begin.** Thus the usual layout of any Ada program after the statements that tell it which library and packages to use is:

```
procedure program name is
   declaration part
begin
   executable statements
end program name ;
```

The program we wrote is also called a *main program,* a *procedure,* or a *main procedure.* When we write a single procedure, it is always the main procedure. When we write a collection of procedures, we will place them in a package and have a main procedure that will use the package. It is good practice to line up the keywords **procedure, begin,** and **end** as shown in the example. An Ada compiler will still run your program if you don't line them up, but people who read your program will rely on your nice layout. In chapter 8, we will see how to write procedures that work together.

The top part of our simple example had no data declaration statements. We had, instead, a *comment*, special text set off with two hyphens. Comments can contain English text to explain what the program does—such information is called *program documentation*. Comments in Ada do not need to end with a semicolon because the end of the line ends a comment. Some good things to put in a comment are your name, the date, the version of the program, and of course what the program does. Comments can be placed anywhere in an Ada program, even on the same line as a statement, but that makes the rest of the line a comment. A well-commented program will be more understandable because you can explain what you are doing in the program in plain English. The general form of a comment is:

```
-- English text
```

Note that the second comment in our example tells what the program does—that it displays the message within quotes on the terminal.

The only output statement in this program—PUT_LINE (" Hi there, I am a very friendly computer! ");—causes output to a terminal, or for those computers without terminals, causes output to a printer. PUT_LINE is actually another program prepared for your use and put in the TEXT_IO package. To use PUT_LINE, you give its name and place what you want to be displayed within parentheses. Text messages such as were used in the example require quotes around the message. The statement ends with a semicolon. The general form of the PUT_LINE statement for messages is:

PUT_LINE (" *message* ") ;

The message must be able to fit on one line. If your message takes more than one line, all you have to do is use more than one PUT_LINE statement.

In our second example, we will use several PUT_LINE statements to form a picture.

```
with TEXT_IO;
use TEXT_IO;
procedure VALENTINE is
-- A program to draw a heart
begin
  PUT_LINE("   xxx   xxx   ");
  PUT_LINE("  x   x x   x  ");
  PUT_LINE("  x    x    x  ");
  PUT_LINE("  x         x  ");
  PUT_LINE("   x       x   ");
  PUT_LINE("    x     x    ");
  PUT_LINE("     x   x     ");
  PUT_LINE("      x x      ");
  PUT_LINE("       x       ");

end VALENTINE;
```