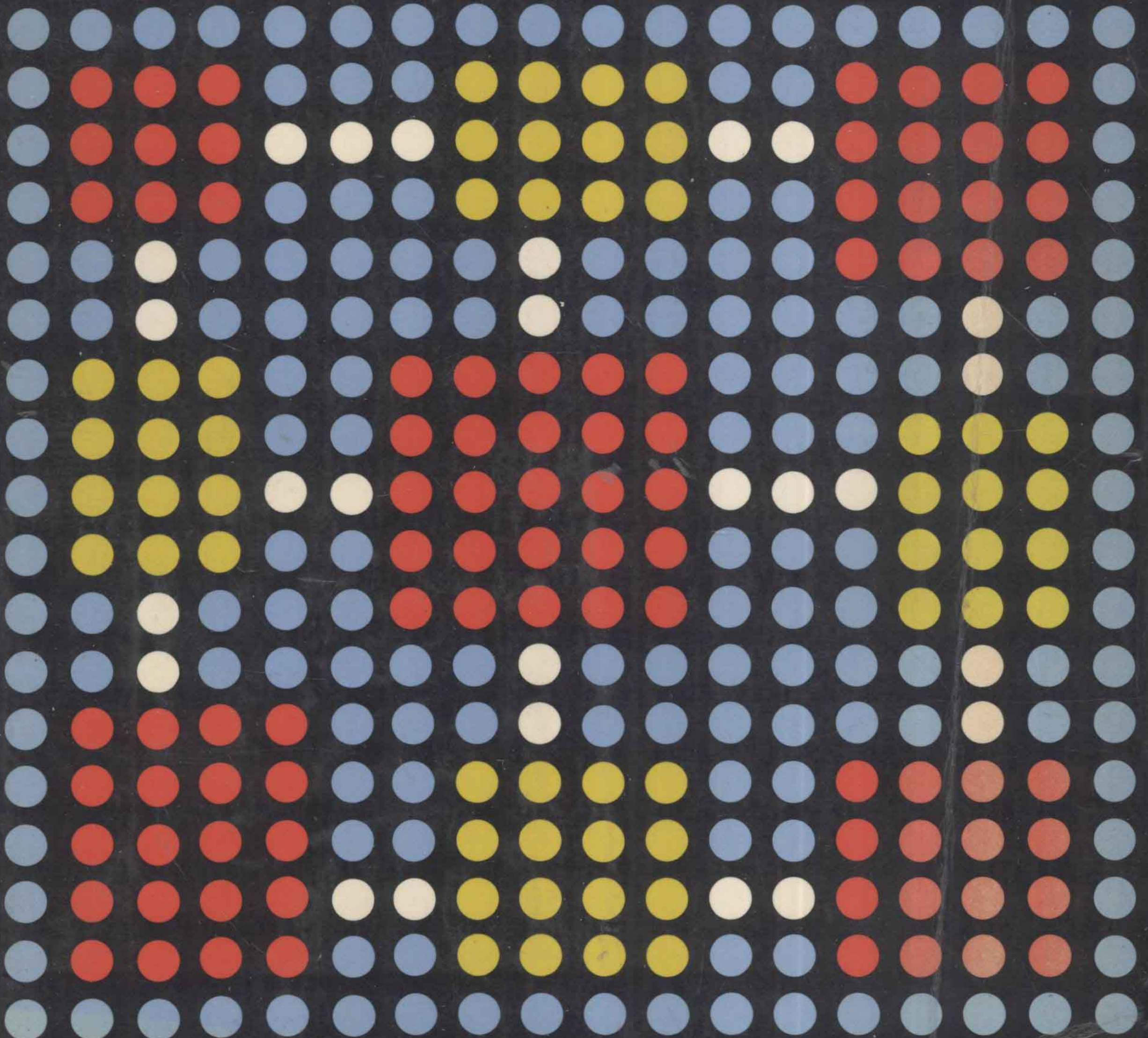
THE ABC'S OF FORTRAN PROGRAMMING

Michael J. Merchant



THE ABC'S OF FORTRAN PROGRAMMING

Mıcnael J. Merchant

Wadsworth Publishing Company, A Division of Wadsworth, Inc., Belmont, California

Computer Science Editor: H. Michael Snell
Editorial Associate: Jenny Sill
Production Editor: Anne Kelly
Art Director: Kate Michels
Designer: Rick Chafian
Cover Designer: Albert Burkhardt

© 1979 by Wadsworth, Inc. All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transcribed, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher, Wadsworth Publishing Company, Belmont, California 94002, a division of Wadsworth, Inc.

Printed in the United States of America

1 2 3 4 5 6 7 8 9 10—83 82 81 80 79

Library of Congress Cataloging in Publication Data

Merchant, Michael J
The ABC's of FORTRAN programming.

Includes index.
1. FORTRAN (Computer program language)
I. Title.
QA76.73.F25M45 001.6'424 78-24198
ISBN 0-534-00634-5

Preface to the Instructor

FORTRAN is an easy language to learn. Most students can learn to write a FORTRAN program in a few weeks. Learning to write a good FORTRAN program, however, is another matter. Too often the student leaves a first course in programming unprepared to tackle a real programming problem.

The ABC's of FORTRAN Programming has two goals. The first is to teach the basics of the FORTRAN computer language, plainly and simply. The second is to help the student put this knowledge to work in solving problems with a computer.

In addressing the first goal, the text concentrates on the mainstream of the FORTRAN language. A few topics, such as P scaling factors and BLOCK DATA subprograms, have been omitted entirely because the beginning student has almost no use for them. Other topics, such as computed and assigned GO TO statements, are of very limited use to a beginning student and so are included in Appendix B. The philosophy of this text is that it is better to give the student a basic set of tools with instructions for using them well, than to provide every conceivable tool and leave the student bewildered by such an excessive variety.

In addressing the second goal, the book provides numerous examples of complete programs. These are all short and easily understood, but they illustrate essential principles in realistic applications. There are no examples concocted to illustrate a syntactic point in a context that would never be used by a good programmer. Many of the exercises also ask the student to write a complete program and run it on the computer. Such active participation is the only way to really grasp the subject.

This book can be used either for a short course on FORTRAN or for a full semester's introduction to programming. Material that is not essential for a basic understanding of the FORTRAN language is included in *optional* sections (indicated by small boxes in the margin), which you can cover selectively, depending on the time available and the interests of the students. Two optional modules on programming style and structured programming give practical advice on how to write a program.

The text does not assume that students have any previous computer experience. Chapter 1 introduces the concepts of an algorithm and a program, and explains how an algorithm is represented by a flowchart. Students familiar with these topics can omit Chapter 1 or use it as a quick review. Chapter 2 gives an overview of the FORTRAN language. Chapter 3 explains how programs are run on a computer and describes how to set up a FORTRAN job. Finally, Chapters 4–10 present a gradual, thorough introduction to FORTRAN programming.

No knowledge of mathematics beyond basic algebra is required. FORTRAN programming can be useful to anyone, not just to mathematicians and scientists; all the examples in this book are nontechnical and self-explanatory.

List-directed input and output statements are used in the early chapters. Most FORTRAN compilers (and WATFOR and WATFIV in particular) allow these forms, and they greatly simplify the details of reading and writing data. If you prefer, however, students can study the optional section in Chapter 2 on formatted PRINT and READ statements and use these forms throughout the book.

I would like to thank Mike Snell, of Wadsworth Publishing Company, for his encouragement. While the responsibility for any deficiencies in this text is my own, I am sincerely grateful to the following for their many helpful comments on the manuscript: Henry A. Etlinger, Rochester Institute of Technology; William L. Harrison, Oregon State University; Lansing Hatfield, Lawrence Livermore Laboratory; Charles E. Moulton, Beaver College; Linda T. Moulton, Montgomery County Community College; Ronald G. Sykora, E. G. & G., Inc.; and Collin J. Watson, University of Oklahoma. I also thank Janet Churchey for her excellent typing. Finally, to my wife Nancy, for her help, and my daughters, Kristina and Vanessa, for their patience, my special thanks.

试读结束：需要全本请在线购买： www.ertongbook.com

Preface to the Student

You are about to begin an exciting experience: learning to program, to command a general-purpose machine capable of any kind of calculation or symbolic manipulation and of following your instructions at the rate of a million operations per second. This machine is a computer. The trick is getting the computer to do what you want, and that is what programming is about.

Of course, you can use a computer without being a programmer. Credit card transactions, checking accounts, automated payroll systems—these are familiar computerized activities that touch our lives daily. Directly or indirectly, most of us deal with computers hundreds of times each year. But the fun part of dealing with computers is writing your own programs. When you can do that, the computer becomes *your* tool, doing what *you* want, and you will be able to use it to do things you could not possibly do alone.

A word of encouragement: Programming is not difficult. It can be learned by anyone who has the patience to do so. I have heard the opinion that programming is only for people with mathematical minds. Nonsense. Anyone who wants to learn and has the patience to practice the skill can become a good programmer. Computers *are* useful for solving mathematical problems; but a knowledge of mathematics is not essential for understanding programming. Computers are also useful in business data processing, social sciences, applied engineering, and dozens of other fields, including just having fun. This book has examples drawn from many fields. However, the text does not assume any special knowledge of the fields, and no mathematics beyond basic algebra is required.

A word of caution, however: Programming is not easy. This is not a subject to be learned the night before an examination. Nor do you learn programming by passively reading and memorizing. What you will be learning is not just a collection of facts but a way of thinking, a way of approaching a problem, a way of inventing a program to achieve a desired result through using the FORTRAN language. This requires practice and active involvement. By the time you have finished Chapter 3, you will be running programs on the computer. From then on, writing programs of increasing difficulty and complexity and running them on the computer will be the most important things you do to learn FORTRAN programming.

Do not expect this skill to come effortlessly. Programs rarely work on the first try. Careful thought, analysis, and perseverance are required. You can expect some frustration, but you can also look forward to an immense pride and satisfaction when your program works.

A word about attitude: There is a tendency for beginners to personify the computer—jokingly, but only half so, to think of it as an antagonist who is out to thwart the program-

mer. Students often remark to each other, "Do you know what that stupid computer did to me?" What they mean is that their program did not run due to a programming error. They rarely say, "I made the most amusing mistake." Instead, they say, "That darn computer did it again." Students sometimes regard programming as a contest between themselves and the computer. Be assured that the computer is quite impersonal and derives no satisfaction from your errors. It is neither intelligent nor malevolent and has neither creativity nor common sense. It does exactly what your program says to do, nothing more. You cannot casually dash off a successful program any more than you can hastily pen a great poem.

Learning to analyze and correct errors requires logical thinking, which can be fun. But if programming is regarded as a hit-or-miss enterprise, it is bound to be difficult and frustrating. Some students try to get a program to run successfully by making random changes in the program and submitting dozens of variations to the computer, hoping that one will work. Even if this method succeeds in getting their homework done, the students are missing the point.

The real enjoyment of programming is in knowing that your program works because you understand it. Answers to selected exercises are provided at the end of this book, and you will profit from comparing your solution to a problem with the answer given here (although there is never just one solution to a programming problem). Ultimately, though, you must be the judge of your own work. When you are applying FORTRAN to your own problems, there will be no answer in the back of the book. The only way you can be sure of your program is to understand thoroughly the FORTRAN language and the basic programming principles you are using. Write your program carefully, making sure you know the reason for each step. Then test the program with trial computer runs to ensure that you have not made any mistakes. Your goal should be to understand the subject so well you do not need to look in the answer section to see if your program is correct. When you can program with that confidence, the computer will truly be your servant.

In many ways, programming is a science. It requires a precise expression of a procedure in unambiguous language, subject to strict rules. But there is also an element of art, for programming is an intensely creative process. To start with a problem statement and build a program requires imagination and intuition. Sometimes, a clever, well thought-out program is a truly beautiful expression of intellect and understanding. The excitement and satisfaction of inventing such a program is no less than that felt by the artist who paints a beautiful picture or chisels a sculpture from stone.

This is what you can look forward to. You will be learning neither pure art nor pure science. I like to think of it as a craft; like the artist's brush, the computer is a tool. The more skilled you become in the craft of programming, the more intricate and polished your creations will be. With these thoughts in mind, let us begin.

Contents

Preface to the Instructor vi

Preface to the Student viii

1 Algorithms, Programs, and Flowcharts 1
- Computers 1
- Algorithms and Programs 4
- Flowcharts 5
- Variables 11
- Decisions 14
- ■ Flowchart Style 23
- Exercises 31

2 Introduction to FORTRAN 42
- The Assignment Statement 43
- The PRINT and READ Statements 45
- The STOP and END Statements 47
- The GO TO and IF Statements 49
- ■ Introduction to Formatted PRINT and READ Statements 55
- Exercises 57

3 Running a Program 59
- Preparing Your Program 59
- Preparing Your Data 64
- ■ Setting Up Your Job Deck 65
- ■ Running Your Program 66
- ■ Errors and Diagnostics 68
- Exercises 69

4 Arithmetic 73
- Numbers 73
- Variables 76

Note: Small boxes before section titles indicate optional sections.

Expressions 80
Real and Integer Arithmetic 84
Type Conversion with an Assignment Statement 87
■ FORTRAN-Supplied Functions 88
Exercises 89

5 Control Statements 92
The GO TO Statement 92
The IF Statement 93
Logical Operators 98
Exercises 105

MODULE I Programming Style 111
Writing a Program Loop 112
Writing a Decision 129
Writing a Chain of Decisions 138
Reliability 142
Exercises 142

6 Formatted Input and Output 150
The Formatted PRINT Statement 150
The FORMAT Statement 151
The I Field Descriptor (Output) 153
The F Field Descriptor (Output) 154
■ The Quote Field Descriptor 158
■ The X Field Descriptor (Output) 159
■ Repeat Factors 160
■ Carriage Control Characters 161
The Formatted READ Statement 164
The I Field Descriptor (Input) 165
The F Field Descriptor (Input) 167
■ The X Field Descriptor (Input) 168
■ The Slash in the FORMAT Statement 173
■ The T Field Descriptor 175
■ Other Input and Output Statements 176
Exercises 177

7 The DO Statement 182
The CONTINUE Statement 185
■ A More General Form of the DO Statement 189
■ Rules for DO Loops 189
Exercises 193

8 Arrays 197
Procedures That Use Arrays 198
Arrays in FORTRAN 199
The DIMENSION Statement 200
■ Two-Dimensional and Three-Dimensional Arrays 219
■ Implied DO Loops in Input and Output Lists 222
Exercises 225

9 Subprograms 231
Subprograms in a Flowchart 231
Subroutines 234
■ Rules for Subroutines 243
Function Subprograms 244
■ Rules for Functions 248
■ Adjustable Dimensions 249
■ Common Storage 256
Exercises 258

MODULE II Structured Programming 267
Exercises 276

10 Character Data 278
The A Field Descriptor 278
Character Constants 282
Assignment and Comparison 283
■ Character Data Type 288
Exercises 297

Appendix A FORTRAN-Supplied Functions 299

Appendix B Additional FORTRAN Features 303

Appendix C Suggested Projects 314

Appendix D Answers to Selected Exercises 327

Index 354

1

Algorithms, Programs, and Flowcharts

You will learn in this book to program a computer in the FORTRAN language. This chapter explains what a program is and how it is represented by a diagram called a flowchart.

Computers

If you have access to a computer system, have someone explain its parts and show you how to run programs on it. The computer may seem complicated, but you can program it without knowing a great deal about its inner workings, just as you can operate a television set without knowing how it is built.

In some ways, a computer is like a hand calculator. The program instructions are like the keys you press on a calculator: They tell the machine what to do. Unlike a hand calculator, however, a computer can process nonnumeric data, such as symbols and alphabetic characters, as well as numbers. For example, in compiling a telephone directory, a properly programmed computer sorts all the names and prints them in alphabetical order. We can define a **computer** as *a machine that can process data by following a list of program instructions*.

What kind of instructions can you give in a program? It would be convenient just to say, "Computer, solve the following problem!" and watch the machine obey. Unfortunately, computers can obey only very simple commands. Every computer has an **instruction set**, consisting typically of one or two hundred commands calling for operations it can perform

试读结束：需要全本请在线购买： www.ertongbook.com

electronically. All these commands fall into one of seven categories: computation, input, output, storing in memory, recalling from memory, comparison, and program control.

COMPUTATION. A computer can do arithmetic. You can write a program instruction telling the machine to multiply, for example. As with a calculator, the actual multiplication is done electronically. A typical large computer can multiply two numbers in one millionth of a second.

INPUT AND OUTPUT. You can also instruct a computer to read in data. Data to be read by a program is called **input data**, or simply **input**. To tell the machine to read, you use an **input instruction** in your program.

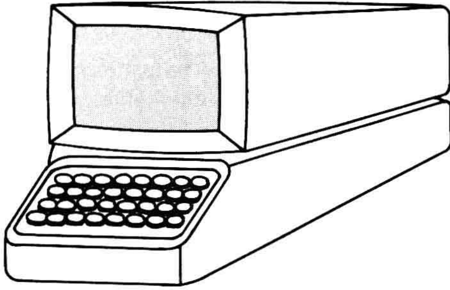
On a hand calculator, you enter numbers by pressing the keys. On a computer system, you might use punch cards instead. You first type the data on a keypunch machine, which has a typewriterlike keyboard. The keypunch is not part of the computer itself, but the punch cards it produces are put in the computer's **card reader**. An input instruction in your program tells the card reader to read a punch card.

When you want your program to write out results, you use an **output instruction**. For instance, you can instruct the machine to print results on a **line printer**. Data written by a program is called **output**.

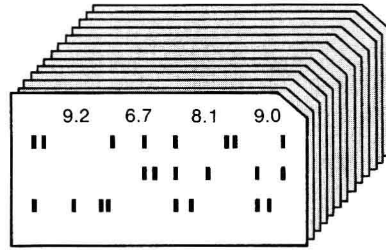
You will probably use only the card reader and line printer at first, but several other devices are used for input and output. Each type has some advantages that make it suitable for certain applications. A computer **terminal** has a keyboard that you can use to enter data directly without using punch cards. Some terminals have a **cathode ray tube (CRT)** display similar to a television screen. Instead of printing results on paper, you can have your program write on the CRT screen. **Magnetic tape** is used for both input and output. Large volumes of data can be recorded very compactly on magnetic tape. Although the data on tape cannot be read directly by a person, it can be read by a computer program for further processing. For example, a commercial bank might run a program each day to record the day's transactions on a magnetic tape. At the end of the month, another program could read the tapes and print statements for the customers. Other common media are **magnetic disk**, **paper tape**, and **microfilm**.

STORING AND RECALLING FROM MEMORY. You can also instruct a computer to *store* or *recall* data from its **memory**. The computer memory is nothing like a human memory; it is just a device for storing data electronically. Many hand calculators have a memory key that lets you save a number. For instance, in a constant percentage problem, you might store a percent figure in memory. Each time you need to use the figure, you can recall it from memory rather than reentering it on the keyboard. This is exactly how a computer memory works. The difference is that instead of a single number, the computer memory can store thousands of numbers.

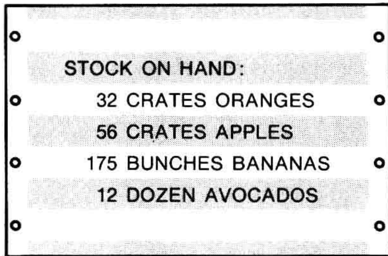
Each memory location has a unique **address**, which is a number used to distinguish it from other memory locations. Think of computer memory as a sequence of mailboxes with a different address on each one. Each mailbox can contain one item of data. A program instruction might say, "Store the number 5.2 in address 1001." Later in the same program, an instruction might say, "Recall the contents of address 1001."



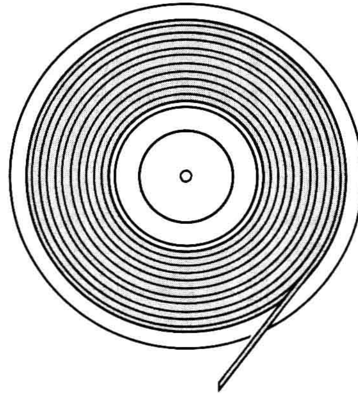
A computer terminal has a keyboard for input and a display screen for output.



Punch card input is prepared on a keypunch and read on a card reader.



Printed output is produced on a line printer.



Magnetic tape is used for both input and output.

Figure 1.1 Some devices used for input and output.

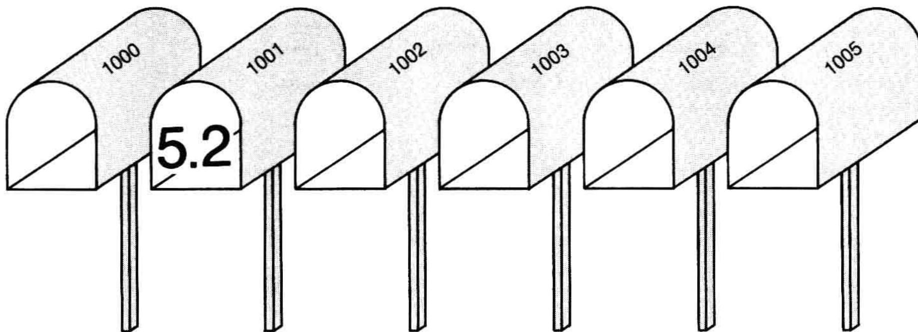


Figure 1.2 You can think of computer memory as a sequence of mailboxes, each with its own address. The instruction "Store 5.2 at address 1001" stores the data 5.2 as the contents of mailbox number 1001.

COMPARISON. Another type of instruction is a **comparison**. This instruction tells the computer to determine whether two numbers stored in memory are the same, or whether one number is greater or less than the other.

PROGRAM CONTROL. The last type of instruction is used for **program control**, which means telling the computer what to do next. This is like an instruction on a tax form that says, "If losses are shown on both lines 12 and 13, go to Part IV." You can use a program control instruction, for example, to tell the machine to repeat a procedure over and over again.

Program control instructions are often combined with comparison instructions. For example, some computers may have a single instruction meaning "Go to Step 1 if the number in address 1010 is greater than zero." Together, comparison and control instructions give a computer the capability to make elementary decisions.

A few more specialized types of instructions exist, such as determining when a card reader is empty. Basically, however, the operations summarized in Table 1.1 are all a computer can do. Writing a program means using these seven kinds of instructions to tell the computer how to solve your problem.

Table 1.1 Types of Program Instructions*

Instruction Type	*Function*	*Example*
Computation	Add, subtract, multiply, etc.	"Add 1 and 2."
Input	Transfer data from some device into memory	"Read a punch card."
Output	Transfer data from memory to some device	"Print the contents of memory address 2000."
Storing in Memory	Save data for later use	"Store the number 5.2 in address 1001."
Recalling from Memory	Retrieve stored data	"Recall the contents of address 1001."
Comparison	Determine whether one number is less than, greater than, or equal to another	"Is the number in address 1001 greater than zero?
Program Control**	Specify which instruction to do next	"Go to Step 1."

* Every computer has an instruction set, consisting typically of one or two hundred basic instructions. These are the building blocks of which all programs are composed. There are seven general types of instructions.

** Program control instructions are often combined with comparison instructions.

Algorithms and Programs

Before you can write a program to solve a problem, you must devise a method of solution expressed as a step-by-step procedure. Such a procedure is called an **algorithm**.* For

* The word *algorithm* comes from the name of a Persian textbook author, al-Khowârizmî (c. 825).

example, to determine whether an integer number is even or odd, you can use the following method.

Algorithm A

1. Divide the number by 2, obtaining a quotient and a remainder.
2. If the remainder is 0, then the number is even.
3. If the remainder is not 0, then the number is odd.

A different algorithm for the same purpose is the following.

Algorithm B

1. Look at the last digit of the number.
2. If the digit is 0, 2, 4, 6, or 8, then the number is even.
3. If the digit is 1, 3, 5, 7, or 9, then the number is odd.

Having decided on the method to be used, you need to express the procedure in a computer language such as FORTRAN. *A program is an algorithm written in a computer language.*

In Chapter 2 we begin our study of the FORTRAN language itself, and in Chapter 3 you will learn how to run a program on a computer. In this chapter we are concerned with how to write an algorithm.

People are usually imprecise in giving instructions to each other. A simple instruction such as "Go to the store for a loaf of bread" requires hundreds of decisions to carry out: Should you go out the front door or the back door? Should you turn left or right? Where is the store? Where is the bread? Should you buy a large loaf, or small? Whole wheat, sourdough, or caraway rye? When you write an algorithm for a computer, you must include every such detail.

Suppose you want to solve a math problem using a calculator. You do not have a calculator, but you have a friend who does, so you call her on the phone to ask for help. She is not at home, but her young brother, who knows very little math, offers to work the calculator if you will tell him exactly what to do. Now you must specify how to solve your problem using instructions that are so precise and unambiguous that he cannot possibly misinterpret them. You might say, "Enter the number 56.2, press the plus key, enter the number 475.3, press the plus key, enter the number 11.63, press the equals key, and tell me the answer." This is an algorithm, expressed in English.

You can imagine how hard it would be to tell someone on the phone how to do a really complicated computation. It is hard in English (or any natural language) to convey complex algorithms that involve many decisions. A useful way of expressing such algorithms is to draw a flowchart.

Flowcharts

A **flowchart** is an easy-to-read diagram of an algorithm. You will probably find it useful to make a flowchart as a first step in writing a program. When the flowchart is done, it will serve as a guide for writing FORTRAN statements.

Each instruction in a flowchart is enclosed in a box, and you use different shapes for different types of instructions. A parallelogram means an input or output operation

(Figure 1.3), and a rectangle means a computation (Figure 1.4). To indicate the order in which the instructions are performed, you connect the boxes with arrows. For example, Figure 1.5 shows a flowchart for a procedure to compute and print the value of 2 + 2. The "Start" oval shows where the procedure begins, and the "Stop" oval shows where it ends.

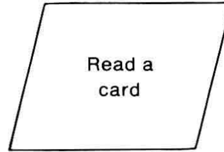


Figure 1.3 A parallelogram represents an input or output instruction.

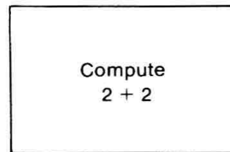


Figure 1.4 A rectangular box represents computation.

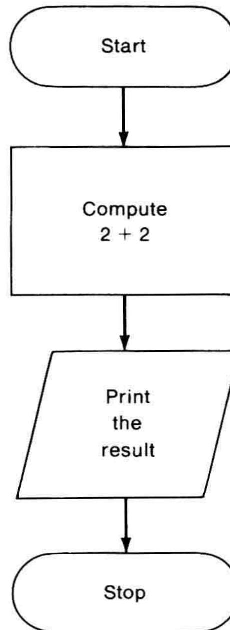


Figure 1.5 Arrows connect the boxes to show the order in which to carry out the instructions.

EXAMPLE 1. A friend has offered to bake cookies for your birthday if you will explain how to make them. Draw a flowchart to describe the cookie-making process. Figure 1.6 shows a flowchart for this procedure. Of course, this is not a computer algorithm, but it illustrates the use of flowcharts in representing processes.

Since the recipe in this example consists of a sequence of instructions that are performed in order, you might just as well have written the instruction in a list, as in a cook book.

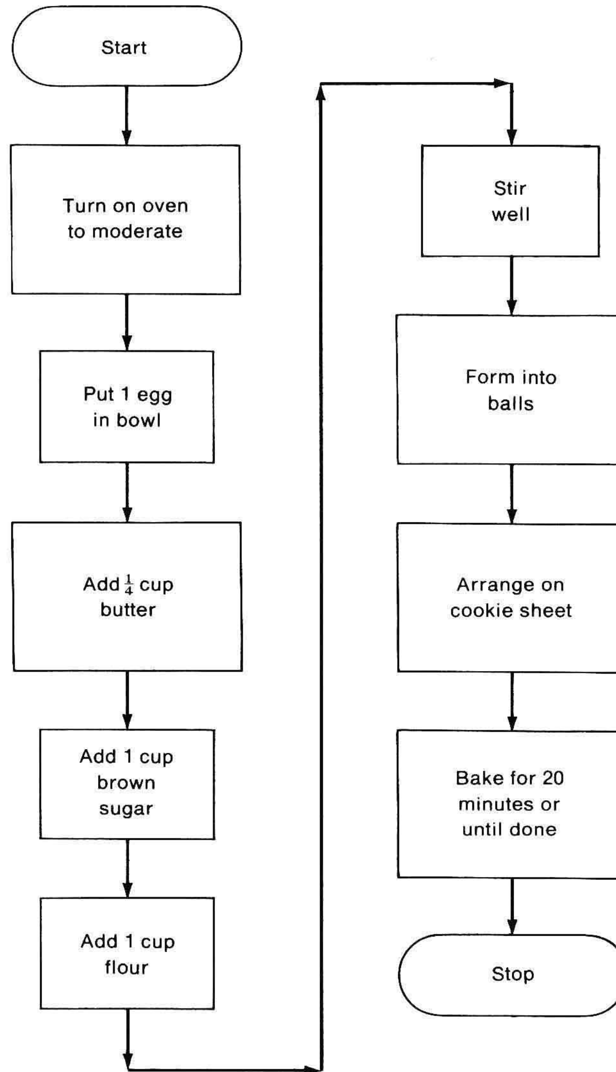


Figure 1.6 Recipe flowchart.

试读结束：需要全本请在线购买： www.ertongbook.com