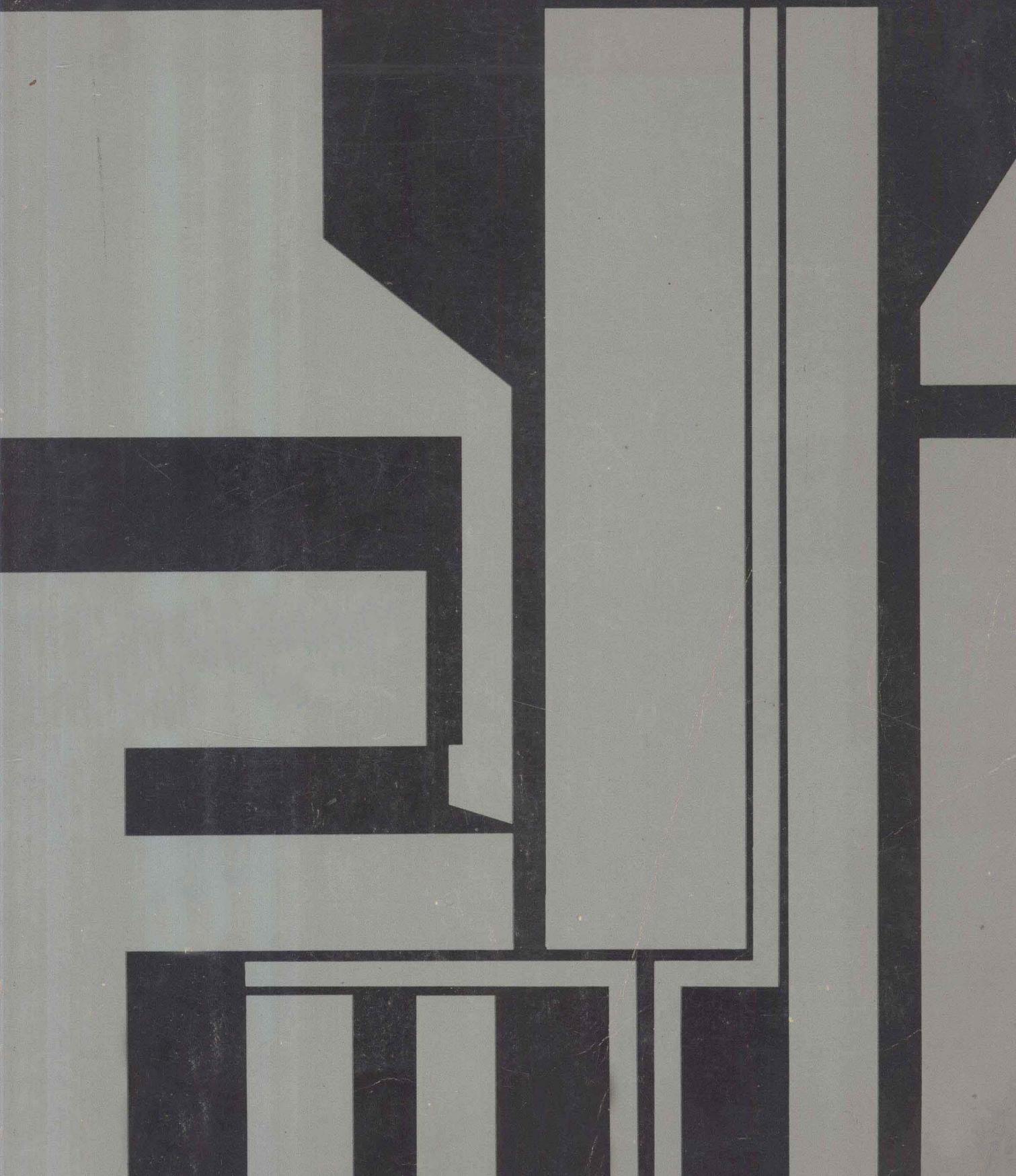
Standard FORTRAN: A Problem-Solving Approach

Laura Cooper | Marilyn Smith



Standard FORTRAN: *A Problem-Solving Approach*

Laura G. Cooper *Florida Agricultural and Mechanical University*
Marilyn Z. Smith *Florida State University*

Houghton Mifflin Company · Boston
Atlanta
Dallas
Geneva, Illinois
Hopewell, New Jersey
Palo Alto

Printed in the U.S.A.

Library of Congress Catalog Card Number: 72–4395

ISBN: 0–395–14028–5

Standard FORTRAN:
A Problem-Solving Approach

Preface

This textbook is the result of five years of teaching experience in the field of computer programming languages at both the undergraduate and graduate levels. We have used several unique teaching concepts:

1. a problem-solving procedure that provides a step-by-step method for writing a program for the computer
2. presentation of the standard ANSI FORTRAN language that the major computer manufacturers follow
3. early presentation of basic material so that the student can write complete programs from the beginning of the course

For each problem in the text, we start the problem-solving procedure with a statement of the problem, then we develop a flowchart, and finally, we write FORTRAN language instructions for the sample problem. Most textbooks give a flowchart only in its final form for a sample problem, but we show the student how to analyze a problem and how to approach the development of the flowchart step-by-step. Then, we guide the student in writing the appropriate FORTRAN instructions corresponding to the steps developed in the flowchart.

Many students shy away from learning FORTRAN because in most books the sample problems are strictly math or science oriented. The numerous sample problems in our text are not restricted to any one area. We feel we have presented a balanced mixture of simple problems from the fields of business, the humanities, and science.

We have adhered strictly to the presentation of standard ANSI FORTRAN, because FORTRAN is a programming language with many dialects. One of its earliest forms was called FORTRAN II. As new commands were added to the language, it became known as FORTRAN IV. Other dialects were developed when various computer manufacturers inserted their own special features into the language. As a result, a program which works on one computer may not work on another one, because the second machine does not know the same dialect of FORTRAN as the first. To bring uniformity to the language, the American National Standards Institute (ANSI) has recommended a standard version of FORTRAN and has encouraged the major computer manufacturers to adopt it. By having a standard language, ANSI hopes to eliminate all the wasted hours spent converting a program for use on different computers. A person learning ANSI FORTRAN can transfer from one computer installation to another and can use the language immediately without studying a new dialect.

One of the main objectives of our text is to teach the student how to use the computer to solve problems. We believe the best way to accomplish this is to have the student write many programs starting with very simple ones. Therefore, we present enough of the basics of problem solving and

the FORTRAN language so that the student should be able to write his first program by the end of the second week of classes, if not sooner. We have organized the book so that the student can learn in stages, always progressing from simple problems to more complex ones.

We approach the teaching of FORTRAN as if it were a foreign language. We introduce new elements of the language in a sample program and later explain the rules of grammar involved. The examples illustrate how to take the different parts of the language and combine them to solve problems. While programming is probably best learned from an instructor, we feel that the ambitious student who has access to a computer could teach himself using our text. We have kept the language of the book simple and free from computer jargon. In addition, we have included many exercises and program assignments to let the student practice using his FORTRAN. Some suggested topics for programs are: an elementary one after Chapter 6, one using single-subscripted variables, one using DØ loops, one on two- or three-dimensional arrays, and one on subprograms. Depending upon the length of the course and the number of credit hours involved, the assignments may be expanded or combined. The entire book can be covered in a one quarter or one semester course. But a two semester course would allow the student to have more practice on the advanced topics.

In our attempt to simplify the material for the beginning student, we have omitted several topics from the main portion of the book and have placed them in Chapter 13. These topics include such items as double precision variables, complex variables, logical variables, the E- and G-field specifications, and Hollerith constants. An individual may use the topics in this chapter whenever a need for them arises. He does not need to finish the first twelve chapters before reading Chapter 13.

Some special features in the text include a brief history of the computer and a discussion of basic computer concepts in Chapter 2. At the end of each chapter we present a list of terms with which the student should be familiar before proceeding to the next chapter. We have included a section on how to use the card punch and the drum control card. To help the student when he writes programs, we have included an appendix designed as a reference to all the ANSI FORTRAN statements discussed in the text. It gives the forms of each statement, the rules for using it, and a reference to the discussion or examples of the statement in the text.

Laura G. Cooper
Marilyn Z. Smith

Contents

Chapter 1

Flowcharts

1.1 Introduction

It is unworthy of excellent men to lose hours like slaves in the labor of calculation which could safely be relegated to anyone else if machines were used.
Leibnitz

Computers can handle large amounts of information compiled by insurance companies, banks, the Census Bureau, and numerous other corporations. They also aid in solving lengthy mathematical problems which involve complex computations. Many familiar situations emphasize the importance of the speed of modern computers. Airline reservations, police investigations, intensive care in hospitals, and space rocketry are just a few examples in which people need an answer immediately and can obtain it from a computer.

Men and women would find some of the tasks mentioned above very tedious and time-consuming, but a computer could do them in a matter of seconds. A computer is similar to an adding machine, because it can do simple arithmetic. Although both a computer and an adding machine are tools and may be used to solve problems, differences do exist between these two machines. In an adding machine the operator enters each number and instruction manually. In order to total a series of numbers, the operator must key in the first number and press the add button, key in the second number and press the add button, key in the third number, and so on. No matter how fast the adding machine, it must wait for the operator to give it a new number and a new instruction (add, subtract, multiply, divide). On the other hand, to have a computer total a series of numbers, an operator submits all the numbers and all the instructions at one time. Then with all the instructions in hand, the computer can begin executing them.

1.2 Why Use a Flowchart?

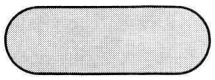
Computers cannot think, but they are capable of performing many arithmetic calculations if they are told what kind of calculation to do and what numbers to use. A computer must receive detailed steps to follow in order to solve a specific problem. It is not enough to command a computer to "calculate my income tax" or "find me a date for Friday night." If a computer is to calculate income tax, it must know which numbers to add, which ones to subtract, and other details a person must know in order to compute the tax by hand. Any problem which a computer is to solve must be broken down into a series of steps that are carried out in some sort of logical order. Usually we first write these steps in the form of a **flowchart**: a diagram which tells from start to finish all the commands which the computer must execute to solve a problem. For the computer to carry out the commands described in the flowchart, we then must convert the commands into some form which the computer understands.

In addition to representing a logical solution to a problem, a flowchart is a means of communication among people who use computers. In understanding another person's method, his flowchart is easier to read than the detailed form which he has prepared for the computer. In each of the next sections we will examine a simple problem and develop a flowchart for solving it.

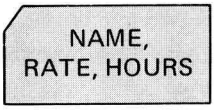
1.3 Flowcharting a Simple Payroll Problem

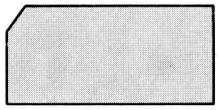
Suppose you work for the payroll department of a large corporation with over 2000 employees. You have a stack of cards containing one card for each employee. The information on the card is the employee's name, his hourly rate of pay, and the number of hours worked per week. Using this information you must compute each person's salary and write him a check. You have an assistant, who may be another person or a computer, write all the checks. Your job is to write down precise instructions on how to do the company's payroll and your assistant will carry out the instructions.

First, we need some place to start. In a flowchart we start by using the symbol

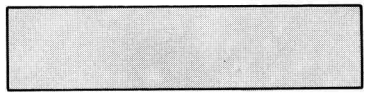


and writing the word START inside. Now, there sits your assistant with a pen and a stack of employee payroll cards in front of him. What should he do next? Compute the salary? But how does he know which numbers to use? Wouldn't he have to get these numbers off the first card in the stack? Yes, he would. Therefore, our first instruction is to read a card and remember the name, the rate of pay, and hours worked. A flowchart has a special symbol for the instruction, "read a card," so that we do not have to write the command "read" inside. This flowchart symbol instructs your assistant to read three items from a single card: NAME, RATE, and HOURS.

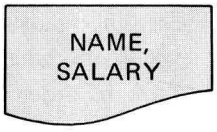


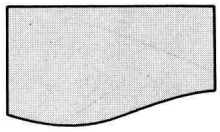


Knowing RATE and HOURS, your assistant can then compute the salary which is RATE × HOURS. We place this command inside the diagram in the flowchart. The arrow (←) states that the value of the product of RATE and HOURS will be labeled SALARY. The use of the arrow is another way of saying that SALARY has just been computed.



What should be the next instruction? To write the check. The items which go on the check are NAME and SALARY. The symbol below signifies the write instruction. We do not need to say "write NAME and SALARY." Instead, the words NAME and SALARY inside this symbol signify that command.





We connect all the boxes to show that we proceed from one box to the next. Our instructions so far have written a check only for the first employee. In order to design our flowchart so that it will write a check for the second employee, we must repeat the instructions, as in Figure 1.1. If this flowchart is to show how to write checks for a company with over 2000 employees, it certainly would be a lengthy one. However, if we examine the flowchart in Figure 1.1, we see that it consists basically of three boxes which we would write repeatedly. We may shorten our flowchart considerably if we use a **loop**—that is, show that we wish to repeat three instructions for each employee. The flowchart in Figure 1.2 illustrates the loop.

Drawing a line from box 4 to just below the START box signifies that after your assistant writes the first check he goes back to the instruction to read the second card, to compute this man's salary, and to write him a check. Then he goes back to read the third card, compute salary, write the check, and continue. What stops the loop? The flowchart does not say. Of course, when no

employee cards remain, there are no more checks to write. We should include something in the flowchart to ask if your assistant has reached the last data card. If he has, then stop; otherwise, continue writing checks. Every flowchart should have a place to start and a way to stop. Figure 1.3 gives the complete flowchart.

Figure 1.1

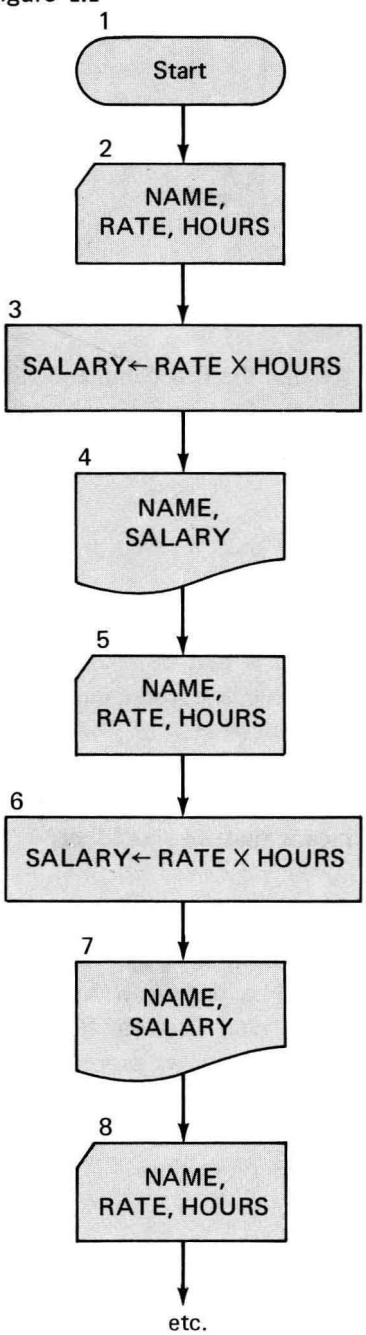


Figure 1.2

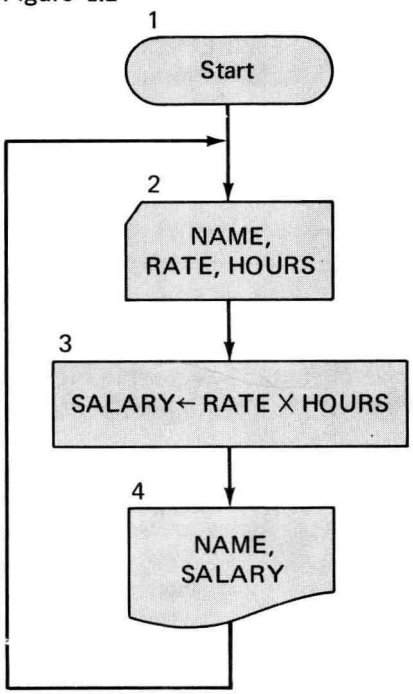
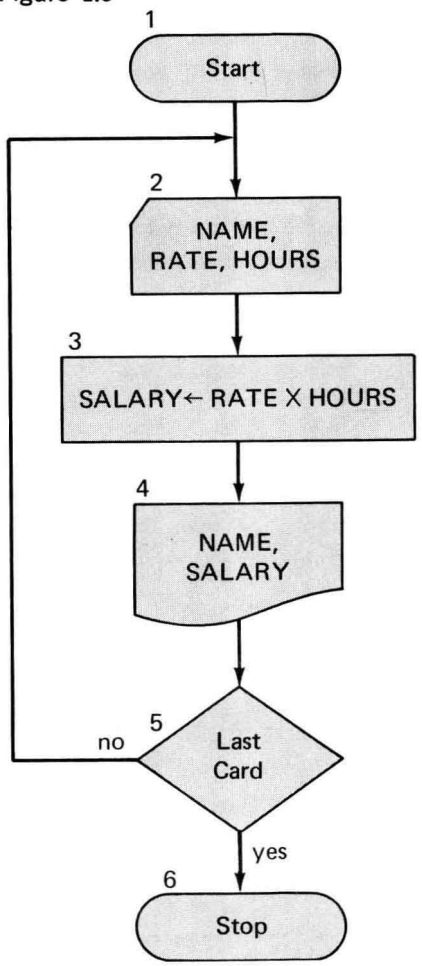


Figure 1.3



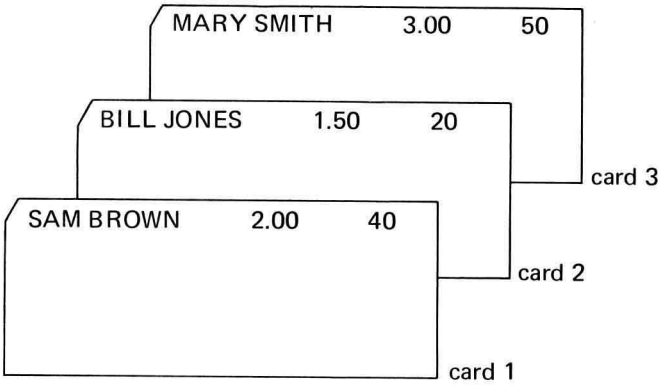
Notice several things about the flowchart in Figure 1.3. Box 5 asks the question: Is this the last card? It is not necessary to include the question mark in the flowchart box, because the diamond-shaped box represents a question. The question has two answers—yes and no—and the flowchart shows which direction to take and what to do whether the answer is yes or no. When we follow the flowchart, we proceed from box 1 to box 2 to box 3 to box 4. Then at box 5 there is a decision to make. We go either to box 2 or to box 6. The possible directions which we may take at a point of decision in a flowchart are called **branches.** The process of following one of these branches is called **branching.** Also note that lines connect all symbols and that the arrows indicate the order of the steps.

Now that we have written the flowchart, how can we be sure that it is correct? Our next step is to take some names and numbers which are employees' rates and hours worked and test the flowchart by hand to see if it will correctly produce the payroll. Is it necessary to use 2000 pieces of test data to check the flowchart? The amount of test data depends upon the problem. In this case, each data card contains the same type of information. Therefore, a flowchart

which works for three or four pieces of data should work for any number of cards.

Suppose we have the three data cards shown in Figure 1.4.

Figure 1.4



Referring to the flowchart in Figure 1.3, we see that the first command is to read NAME, RATE, and HOURS from a card. Take a piece of paper and write

```
 NAME = SAM BROWN
 RATE = 2.00
HOURS = 40
```

With such a simple problem, we could probably remember these values without writing them. However, a good habit to develop is that of writing down everything read in or computed in a flowchart. As problems become more complicated, it is not always possible to remember every number.

After we read one card, we proceed from box 2 to box 3 and calculate the salary. Interpret this box as: multiply RATE by HOURS ($2.00 \times 40 = 80.00$), and call this number SALARY. This means that we should write

```
SALARY = 80.00
```

When the flowchart says to write NAME and SALARY, we may refer to the worksheet to see that the NAME was SAM BROWN and his SALARY was 80.00. Knowing these items, we could write him a check. Then the flowchart asks if we have just processed the last card. Since we haven't, we proceed from flowchart box 5 to box 2 where we find the instruction to read a card. Since we have read the first test card and we cannot read it again, we may wish to cross it off. The card to read now is the second one. Instead of writing

```
 NAME = BILL JONES
 RATE = 1.50
HOURS = 20
```

let us erase or scratch out the old values of NAME, RATE, and HOURS, and replace them with these new values.

```
 NAME = SAM BROWN    BILL JONES
 RATE = 2.00         1.50
HOURS = 40           20
```

Each name may have only one value at a time. Every time there is a new

value, the old one is erased. This concept may not be very intuitive, but, in fact, it is the way computers work. We continue to compute RATE × HOURS ($1.50 \times 20 = 30.00$) and replace the value of SALARY with this new value. The arrow (←) here indicates replacement, that is, replace the previous value of SALARY with 30.00. When we processed the first data card, the arrow told us to *compute* SALARY for the first time. Now that we are processing another data card and SALARY already has a value, the arrows means *replace* the former value of SALARY with a new one. In either case, whether the arrow means compute or replace, the arrow indicates that SALARY is to have a value.

SALARY = ~~80.00~~ 30.00

We continue through the flowchart until we have written the check for MARY SMITH. Then, when the flowchart asks if we have processed the last card, the answer is yes and we stop.

There is an easy and convenient tabular method for analyzing a flowchart. We call this table listing a **trace**. Table 1.1 is a trace diagram of the wage problem. Note the following properties of the table:

1. For each step of the flowchart, one (and only one) line in the table is entered.
2. Since some values remain unchanged during several steps, we do not copy them down each time. In other words, in step 2, the values for NAME, RATE, and HOURS are still: SAM BROWN, 2.00, and 40; but we do not copy them over. However, in step 5 NAME becomes BILL JONES and no longer is SAM BROWN. At this point, when we have a new value for NAME, we may wish to cross off the old value.

The flowchart seems to work because it has produced the correct answers. But remember that a flowchart is the first step toward having the computer solve a problem for us, because the instructions in the flowchart are the same ones which we will give to the computer to tell it how to do the problem.

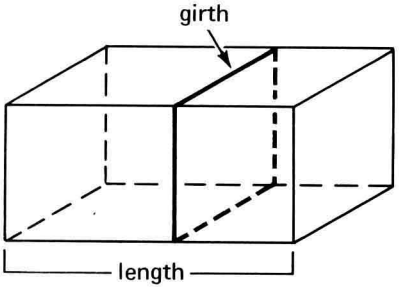
Table 1.1 Trace Diagram of Wage Problem

Step No.	Flow-chart Box No.	Variables				Test	Output (Write)
		Name	Rate	Hours	Salary	Last card	
1	2	Sam Brown	2.00	40			
2	3				80 = (2.00)(40)		
3	4						Sam Brown 80.00
4	5					no	
5	2	Bill Jones	1.50	20			
6	3				30 = (1.50)(20)		
7	4						Bill Jones 30.00
8	5					no	
9	2	Mary Smith	3.00	50			
10	3				150 = (3.00)(50)		
11	4						Mary Smith 150.00
12	5					yes	
13	6						

1.4 A Flowchart to Decide If a Crate May Be Mailed

A warehouse manager has over 1000 crates which he wishes to mail, but the Post Office will accept the crates only if they are not too large. As each crate was filled, the packing manager assigned a number to the crate and measured its length, width, and height. He wrote the information about each crate on a separate card. The Post Office requires that the length of the longest side plus the distance around the other sides (girth) must be less than or equal to 72 inches. We wish to develop a procedure which will enable the warehouse manager to decide which crates he cannot mail.

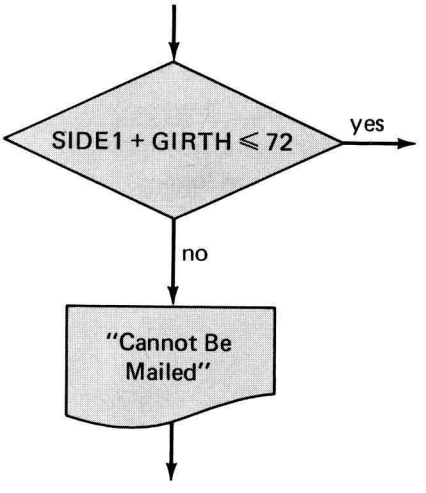
Figure 1.5



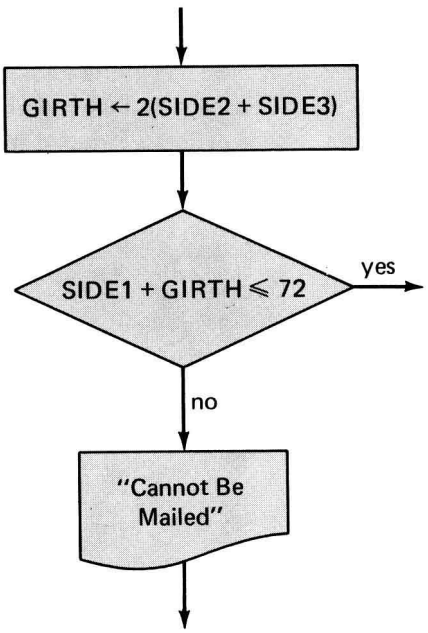
The information which we have consists of the crate code number and the lengths of the three sides. Let us label these items so that we may refer to them easily in the flowchart:

CODE—crate code number
SIDE1—length of the longest side
SIDE2—length of the second longest side
SIDE3—length of the shortest side

The main portion of the problem is to determine if SIDE1 + GIRTH is less than or equal to 72. If it is, the warehouse manager can mail the crate. If not, we should write a message saying that the crate "cannot be mailed." To write this in a flowchart diagram we have the following boxes in which the symbol $\leq$ represents the words "less than or equal to."

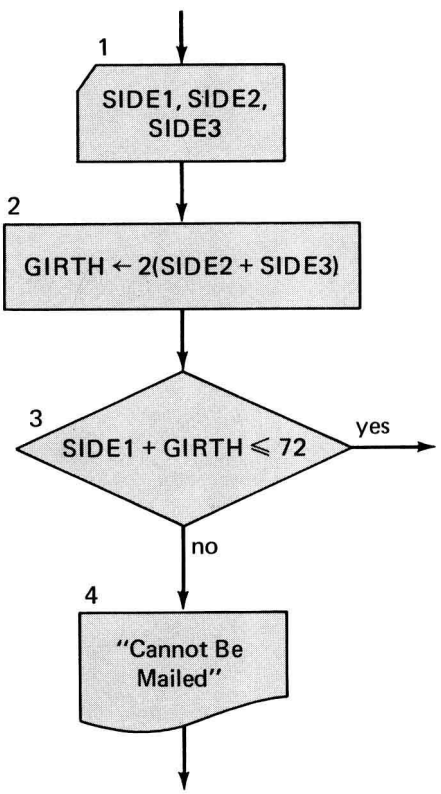


But how does the flowchart obtain a value for SIDE1 and a value for GIRTH? We must compute the GIRTH as 2(SIDE2 + SIDE3) before we may ask the question: Is SIDE1 + GIRTH $\leq$ 72. Let us add this to the flowchart.



So far the flowchart seems to be progressing smoothly, but where do the values for SIDE1, SIDE2, and SIDE3 come from? They must be read from a card before they can be used. Therefore, we add this step to the flowchart.

Figure 1.6



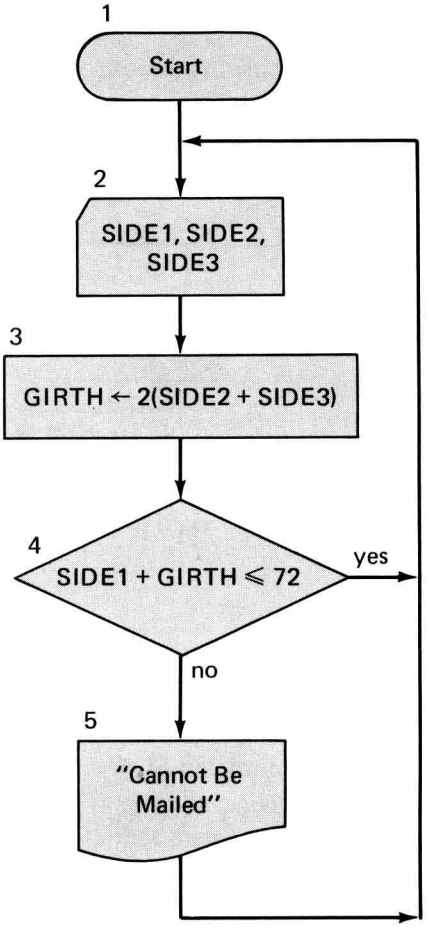
So far, we have at least three loose ends in the flowchart:

1. Where do we START?
2. What do we do if SIDE1 + GIRTH is less than or equal to 72?
3. Where do we proceed after writing "cannot be mailed"?

We want to begin the flowchart by reading a card. If SIDE1 + GIRTH $\leq$ 72, the warehouse manager can mail the crate, and we should proceed to

check the next crate. If he cannot mail the crate, we should write the message and then proceed to check the next crate. By making the flowchart loop back to box 2, we can go on to process the next crate. This addition to the flowchart in Figure 1.6 appears in Figure 1.7

Figure 1.7



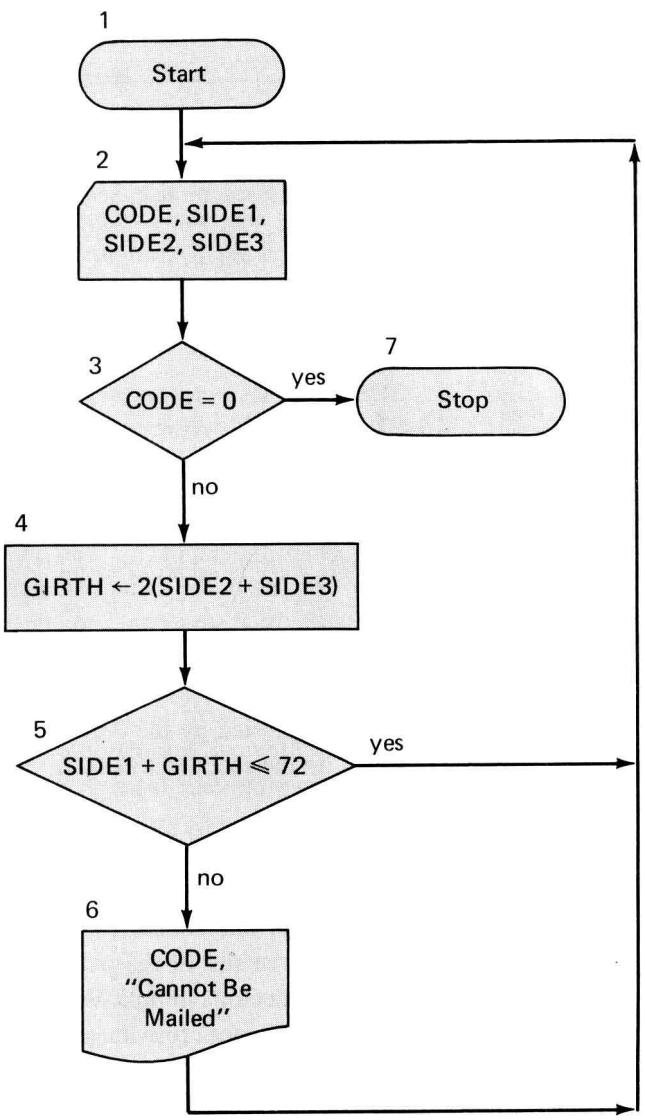
Let us now examine the flowchart in Figure 1.7. It seems to be complete, but there are several places where we could improve it. Consider the message which the computer writes. It does not say which crate cannot be mailed. Since each crate was numbered, we should write that number along with the message. To show in the flowchart that we want to write the specific words "cannot be mailed" they are enclosed in quotation marks. To show that we want to write the crate code number, we will include the word CODE in the flowchart write box. This means that the *value* of CODE will be written and not the letters CODE. If we wish to write the crate code number, first we must read it in along with the lengths of the sides.

A second improvement in the flowchart would be a way to stop the process. Our flowchart loops continually expecting a new card each time. At some point no cards will remain. What can we use to stop the loop in the flowchart? We could ask if the last card has been read, as in Figure 1.3. However, if we wish to convert this flowchart into a form which a computer can understand, there

is no way for us to tell the computer to check for the last data card. When a computer reads a card, it cannot tell if that card is the first one, the last one, or any other one. However, we can insert a special card at the end of the card deck. If this card contained a number unlike any of the numbers in the data deck, then we could look for this card. When we find it, we would know that we had come to the end of the card deck. This special card is called a **dummy data card,** because it is not part of the actual data deck, but it signals the end of the data card deck.

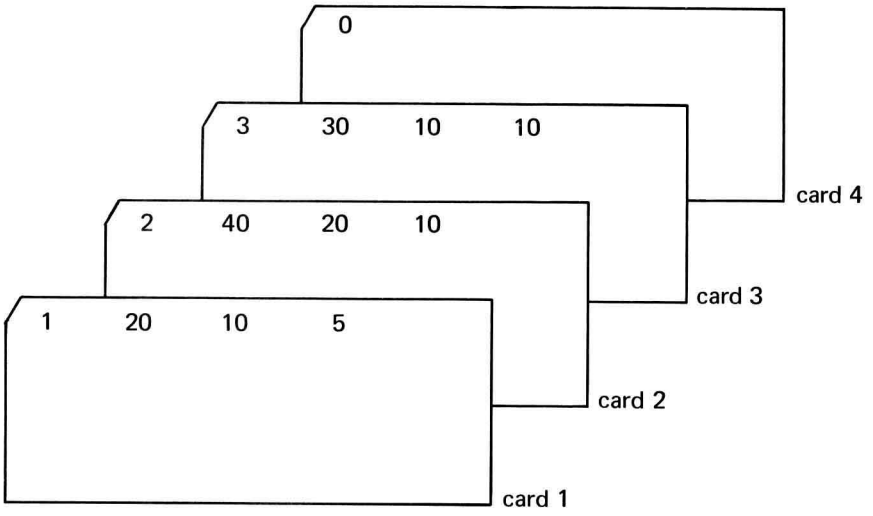
What kind of dummy data card could we use in this flowchart? Since the crates would be numbered 1, 2, 3, and so on, our dummy card could contain a zero value for CODE. The complete flowchart appears in Figure 1.8.

Figure 1.8



Exercise

Suppose the warehouse manager has the following four data cards:



Construct a trace diagram of the flowchart in Figure 1.8 using these data cards. Your trace table headings should look like the following:

Step No.	Flow-chart Box No.	Variables					Test		Output (Write)
		Code	Side1	Side2	Side3	Girth	Code=0	Side1+Girth≤72	

1.5 Summary of Flowchart Symbols

A flowchart is an effective way of representing the solution to a problem. If a flowchart is going to be useful, the author should draw it so that someone else can understand it. This means that everyone should use the same symbol to represent the same command. For example, the diamond asks a question and the oval says either to start or stop. If we saw a flowchart in which ovals represented questions and diamonds represented starting, we would have to relearn the meaning of these symbols. In this section we will discuss those symbols suggested by the American National Standards Institute (ANSI) which apply to the scope of this text.

Rules for the Use of Flowcharts

1. A flowchart should have a place to *start* and a place to *stop*.
2. A line should connect every symbol in a flowchart to another symbol. If the size of the page prohibits the connection of two symbols, use the connector symbol. We will give an example of this later.
3. Do not cross lines on a flowchart.
4. Normally the direction of movement from one flowchart box to another is from top to bottom and from left to right. Use arrows on the connecting lines to show that the direction is from bottom to top or from right to left. Arrows may be used on all connecting lines to point out explicitly the direction of the flow.
5. Sentences, symbols, or words may be used inside the flowchart boxes.
6. The arrow (←) denotes replacement.
7. Use quotation marks around a message that will be written out.