Chapter 1

General Problem-Solving Concepts

Overview

Problem Solving in Everyday Life
Types of Problems
Problem Solving with Computers
Difficulties with Problem Solving

Objectives

When you have finished this chapter, you should be able to:

1. Describe the difference between heuristic and algorithmic solutions to problems.
2. List and describe the six problem-solving steps to solve a problem that has an algorithmic solution.
3. Use the six problem-solving steps to solve a problem.

Problem Solving in Everyday Life

People make decisions every day to solve problems that affect their lives. The problems may be as unimportant as what to watch on television or as important as choosing a new profession. If a bad decision is made, time and resources are wasted, so it's important that people know how to make decisions well. There are six steps to follow to ensure the best decision. These six steps in problem solving include the following:

Six Steps of Problem Solving

1. *Identify the problem.* The first step toward solving a problem is to identify the problem. In a classroom situation, most problems have been identified for you and given to you in the form of written assignments or problems out of a book.

However, when you are doing problem solving outside the classroom, you need to make sure you identify the problem before you start solving it. If you don't know what the problem is, you cannot solve it.

2. *Understand the problem.* You must understand what is involved in the problem before you can continue toward the solution. This includes understanding the knowledge base of the person or machine for whom you are solving the problem. If you are setting up a solution for a person, then you must know what that person knows. A different set of instructions might have to be used depending on this knowledge base. For example, you would use a more detailed set of instructions to tell someone how to find a restaurant in your city if he has a limited knowledge of the city than if he knows it well. When you are working with a computer, its knowledge base is the limited instructions the computer can understand in the particular language or application you are using to solve the problem. Knowing the knowledge base is very important since you cannot use any instructions outside this base. You also must know your own knowledge base. You cannot solve a problem if you do not know the subject. For example, to solve a problem involving calculus, you must know calculus; to solve a problem involving accounting, you must know accounting. You must be able to communicate with your client and be able to understand what is involved in solving the problem.
3. *Identify alternative ways to solve the problem.* This list should be as complete as possible. You might want to talk to other people to find other solutions than those you have identified. Alternative solutions must be acceptable ones. You could go from Denver to Los Angeles by way of New York, but this would probably not be an acceptable solution to your travel needs.
4. *Select the best way to solve the problem from the list of alternative solutions.* In this step, you need to identify and evaluate the pros and cons of each possible solution before selecting the best one. In order to do this, you need to select criteria for the evaluation. These criteria will serve as the guidelines for evaluating each solution.
5. *List instructions that enable you to solve the problem using the selected solution.* These numbered, step-by-step instructions must fall within the knowledge base set up in step 2. No instruction can be used unless the individual or the machine can understand it. This can be very limiting, especially when working with computers.
6. *Evaluate the solution.* To evaluate or test a solution means to check its result to see if it is correct, and to see if it satisfies the needs of the person(s) with the problem. (When a person needs a piece of furniture to sleep on, buying her a cot may be a *correct* solution, but it may not be very satisfactory.) If the result is either incorrect or unsatisfactory, then the problem solver must review the list of instructions to see that they are correct or start the process all over again.

If any of these six steps are not completed well, the results may be less than desired.

People solve problems daily at home, or work, or wherever they go. Problems at home include such things as what to cook for dinner, which movie to see this evening, which car to buy, or how to sell the house. At work, the problems might involve dealing with fellow employees, work policies, management, or customers. The better the decisions an employee can make, the more valuable that person will be to the company. In each case, the six steps in problem solving can be followed. Most people use them without even knowing it.

Take the problem of what to do this evening.

1. Identify the problem. How do the individuals wish to spend the evening?
2. Understand the problem. With this simple problem, also, the knowledge base of the participants must be considered. The only solutions that should be selected are ones that everyone involved would know how to do. You probably would not select as a possible solution playing a game of chess if the participants did not know how to play.
3. Identify alternatives.
   a. Watch television.
   b. Invite friends over.
   c. Play video games.
   d. Go to the movies.
   e. Play miniature golf.
   f. Go to the amusement park.
   g. Go to a friend's party.

   The list is complete only when you can think of no more alternatives.
4. Select the best way to solve the problem.
   a. Weed out alternatives that are not acceptable, such as those that cost too much money or do not interest one of the individuals involved.
   b. Specify the pros and cons of each remaining alternative.
   c. Weigh the pros and cons to make the final decision. This solution will be the best alternative if all the other steps were completed well.
5. Prepare a list of steps (instructions) that will result in a fun evening.
6. Evaluate the solution. Are we having fun yet? If nobody is having fun, then the planner needs to review the steps to have a fun evening to see whether anything can be changed, if not then the process must start again.

By going through these steps, the problem solver can be assured that he has arrived at the best possible solution and will achieve the desired results.

Types of Problems

algorithmic solution

algorithm

heuristic solution

Problems do not always have straightforward solutions. Some problems, such as balancing a checkbook or baking a cake, can be solved with a series of actions. These solutions are called **algorithmic solutions**. Once the alternatives have been eliminated, for example, and once one has chosen the best among several methods of balancing the checkbook, the solution can be reached by completing the actions in steps. These steps are called the **algorithm**. The solutions of other problems, such as how to buy the best stock or whether to expand the company, are not so straightforward. These solutions require reasoning built on knowledge and experience, and a process of trial and error. Solutions that cannot be reached through a direct set of steps are called **heuristic solutions**.

The problem solver can use the six steps for both algorithmic and heuristic solutions. However, in step 6, evaluating the solution, the correctness and appropriateness of heuristic solutions are far less certain. It's easy to tell if your completed checkbook balance is correct and satisfactory, but it's hard to tell if you have bought the best stock. With heuristic solutions, the problem solver will often need to follow the six steps more than once, carefully evaluating each possible solution before deciding which is best.

answer at all. If the instructions are not properly sequenced, the computer will, nevertheless, execute them in the order given, and the result will be wrong.

syntax
bug
debugging

The meaning of an instruction is essentially the same in any computer language or application. The differences between instructions from one language and another are in how they are set up. **Syntax** refers to the rules governing the computer operating system, the language, and the application. An error is called a **bug.** A bug must be found and corrected, a process called **debugging.** Many bugs are a result of syntax errors, but some are logic errors. You can find and correct most logic errors during the problem-solving process. You will find and correct syntax errors when you enter your program into the computer. All syntax errors must be corrected before you execute and test your program.

Although a set of instructions must be in a correct order to lead to the correct result, there may be several "correct" orders, just as there are several routes leading from New York to San Francisco. Whether the routes are equally efficient doesn't affect the result. When efficiency is important, then and only then should each route be examined and kept or discarded. People employ different logic and different ways of thinking. Two programmers may develop equally good solutions to a problem, but the solutions may look entirely different. There is nothing wrong with this. Computers are exact machines, but the people working with them are not.

Organizing the Solution

problem analysis chart
structure chart
interactivity chart
IPO chart
algorithms
flowchart
pseudocode
coupling diagram
data dictionary
UML

Certain organizational tools will help you learn to solve problems on the computer. The tools used in this book and illustrated in this chapter include the **problem analysis chart,** which shows a beginning analysis of the problem; the **structure chart** or **interactivity chart,** which shows the overall layout or structure of the solution; the **IPO chart,** which shows the input, the processing, and the output; the **algorithms,** which show the sequence of instructions comprising the solution; and the **flowcharts,** which are graphic representations of the algorithms and **pseudocode,** which represents a language like solution. A **coupling diagram** and **Data Dictionary** are presented in Chapter 4. The coupling diagram shows the relationship between the modules and the data needed for the modules. The Data Dictionary lists all date variable names and their definitions. **UML (Unified Modeling Language)** is a basic tool when using Object Oriented Programming stucture. An introduction to UML is presented in this chapter, followed by usage in Chapters 15 and 16. To analyze a problem and set up the most efficient solution, a programmer organizes the solution by using all or some of these tools. When the programmer does not use these tools during the problem-solving process, the solution takes longer to program, and the final program is less efficient, lacks readability, and increases programmer frustration.

Analyzing the Problem

To organize a solution, the programmer first has to understand and analyze the requirements of the problem. A good way to analyze a problem is to separate it into four parts, shown in the problem analysis chart (PAC) in Figure 3.1:

1. The given data
2. The required results
3. The processing that is required in the problem
4. A list of solution alternatives

Given Data	Required Results
Section 1: Data given in the problem or provided by the user. These can be known values or general names for data, such as price, quantity, and so forth.	Section 2: Requirements for the output reports. This includes the information needed and the format required.
Processing Required	Solution Alternatives
Section 3: List of processing required. This includes equations or other types of processing, such as sorting, searching, and so forth.	Section 4: List of ideas for the solution of the problem.

Figure 3.1 Problem Analysis Chart

(A blank problem analysis chart for student use appears in Appendix D, Figure D.1). As a problem unfolds (in a textbook or in the real world), the programmer can use this form to sort it out. The PAC allows the problem solver to eliminate the words and glean only the facts from the problem. Data, constants and variables, would be entered in the Given Data section. Variable data are the input values. Requirements for the output reports would be entered under Required Results. Any equations or other processing requirements would be listed in the Processing Required section. Finally, the programmer would write any other ideas that spring to mind concerning the solution in the Solution Alternatives section.

The following problem illustrates the use of the problem-solving tools: Calculate the gross pay of an employee given the hours worked and the rate of pay. The gross pay is calculated by multiplying the hours worked by the rate of pay.

Figure 3.2 shows how the problem solver would fill in the PAC for this problem. The hours worked and the pay rate are the data given and are put into the Given Data box. The gross pay is what needs to be calculated and given to the user; therefore it is put into the Required Results box. The formula used is

$$GrossPay = Hours * PayRate$$

It is put into the Processing Required box. The Solution Alternatives are as follows:

1. Define the hours worked and the pay rate as constants.
2. Define the hours worked and the pay rate as input values.

Given Data	Required Results
Hours Pay Rate	Gross Pay
Processing Required	Solution Alternatives
GrossPay = *Hours* * *PayRate*	1. Define the hours worked and pay rate as constants. *2. Define the hours worked and pay rate as input values.

Figure 3.2 Problem Analysis Chart for the Payroll Problem

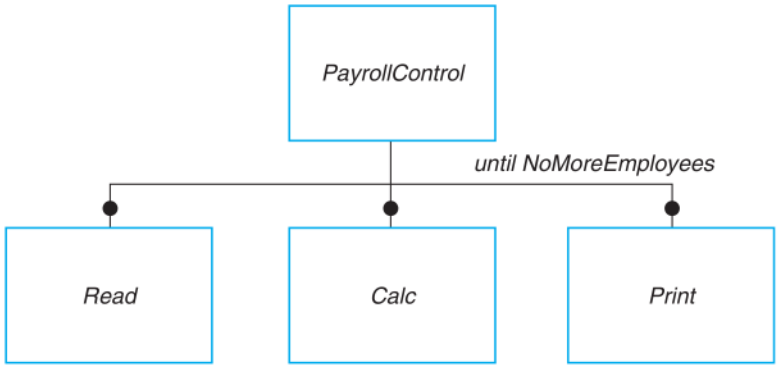


Figure 3.5 The Interactivity Chart for the Payroll Problem

3. A read module (Read) to enter data needed in the solution.
4. Two modules (Salaried and Hourly) to enter data needed for two different types of employees.
5. A calculation module (Calculate) to process the data for each employee.
6. A print module (Print) to print the processed information.
7. A wrap-up module (WrapUp) to process any data that should be done only once in the solution and at the end of the solution.

These are some of the standard modules. In the next several chapters you might start with four modules. The control module, the read module, the calculation module, and the print module are basic to most programs. From there, add other modules as needed along with the addition of processing symbols showing which modules are duplicates, which are part of a decision, and which are repeated in a loop. In Figure 3.4, the Calculate module is duplicated in two places as indicated by the shaded in corner; the Salaried and Hourly modules are part of a decision as indicated by the diamond; and the Read module, the Calculate module, and the Print module are repeated until the NoMoreEmpolyees (when there are no more employees to process) as indicated by the filled-in circle. These concepts will be discussed further in Chapter 4.

Figure 3.5 shows the interactivity chart for the payroll problem. The top rectangle is the *Control* module, which processes the *Read,* the *Calc,* and the *Print* modules.

Developing the IPO Chart

The IPO (input-processing-output) chart extends and organizes the information in the problem analysis chart. It shows in more detail what data items are input, what processing takes place on that data, and what information will be the end result, the output (see Figure 3.6). The IPO chart also shows where in the solution the processing takes place. The output is the

Input	Processing	Module Reference	Output
All input data (from Section 1 of the problem analysis chart)	All processing in steps (from Sections 3 and 4 of the problem analysis chart)	Module reference from the interactivity chart	All output requirements (from Sections 1 and 2 of the problem analysis chart)

Figure 3.6 The IPO Chart

Input	Processing	Module Reference	Output
Hours Worked Pay Rate	1. Enter Hours Worked 2. Enter Pay Rate 3. Calculate Pay 4. Print Pay 5. End	*Read* *Read* *Calc* *Print* *PayRollControl*	Gross pay

Figure 3.7 The IPO Chart for the Payroll Problem

first to be completed in an IPO chart. The next is the input, and the last is the processing. This may sound backwards, but it is really very logical. An analogy to completing an IPO chart would be planning a trip. The first thing you need to know is the destination—San Francisco, Paris, New York. This information is the output. Then you can plan what to pack, your means of transportation, and all the other essentials for starting out—the input. Finally, you plan the route—the processing. How can you know what to pack or what route to follow if you don't know where you're going?

The IPO chart has four sections: the Input, the Processing, the Module Reference, and the Output. The Input section contains all input data from the problem analysis chart. The input includes all given data from Section 1 on the problem analysis chart. The Processing section contains all processing, evident and implied, from Sections 3 and 4 of the problem analysis chart. The Module Reference section contains the number from the interactivity chart of the module in which each step in the processing is to be completed. The Output section includes all required output as designated by the problem and/or the user; it comes from Section 2, along with the needed input items from Section 1, of the problem analysis chart. (An IPO chart for student use appears in Appendix D, Figure D.3.)

To illustrate how to use an IPO chart, Figure 3.7 shows one for the payroll problem. The input, the processing, and the output are taken from Figure 3.2, the problem analysis chart for the payroll problem. The module references are taken from Figure 3.5, the interactivity chart for this problem. They show which module will perform each step in the processing. Notice also that these steps are numbered in the order to be processed.

Writing the Algorithms

Algorithms

After using the structure chart and the IPO chart, the next step in organizing a solution is for the programmer to develop sets of instructions for the computer, called *algorithms* (see Figure 3.8). To complete all of the algorithms needed to solve a problem, the programmer writes a separate set of instructions for each module in the structure

Control *Module*	*Name of Module (list of parameters)*
1. *Instruction*	1. *Instruction*
2. *Instruction*	2. *Instruction*
3. ..	3. ..
4. ..	4. ..
..	..
—. end	—, exit

Figure 3.8 The Form of an Algorithm

chart. To be understood by the computer, the instructions have to be written according to certain rules that will be explained in future chapters throughout this book. Setting up the algorithms is probably the hardest part of problem solving on the computer. *The instructions cannot assume anything, cannot skip steps, must be executable one step at a time, and must be complete.* The modules are taken from the interactivity chart, and the processing is taken from the IPO chart. The number of instructions in an algorithm is determined by the way the programmer chooses to solve the problem. The algorithms pull the interactivity chart and the IPO chart together to give a logical step-by-step solution. Note in Figure 3.8, the *Control* module uses an *End* since this is the end of the processing. The other modules use *Exit* because the processing continues.

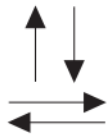
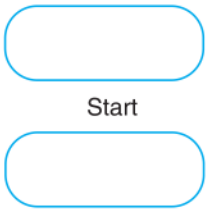
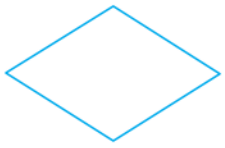
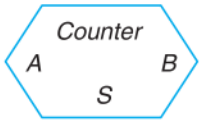
Flowchart Symbol	Explanation
 Flowlines	Flowlines are indicated by straight lines with optional arrows to show the direction of data flow. The arrowhead is necessary when the flow direction might be in doubt. Flowlines are used to connect blocks by exiting from one and entering another.
 Start End/Stop/Exit	Flattened ellipses indicate the start and the end of a module. An ellipse uses the name of the module at the start. The end is indicated by the word *end* or *stop* for the top or *Control* module and the word *exit* for all other modules. A start has no flowlines entering it and only one exiting it; an end or exit has one flowline entering it but none exiting it.
 Processing	The rectangle indicates a processing block, for such things as calculations, opening and closing files, and so forth. A processing block has one entrance and one exit.
 I/O	The parallelogram indicates input to and output from the computer memory. An input/output (I/O) block has one entrance and only one exit.
 Decision	The diamond indicates a decision. It has one entrance and two and only two exits from the block. One exit is the action when the resultant is *True* and the other exit is the action when the resultant is *False*.

Figure 3.9 Flowchart Symbols (*continued on page 50*)

Flowchart Symbol	Explanation
 Process Module	Rectangles with lines down each side indicate the process of modules.They have one entrance and only one exit.
 Automatic-Counter Loop	The polygon indicates a loop with a counter. The counter starts with *A* (the beginning value) and is incremented by *S* (the incrementor value) until the counter is greater than *B* (the ending value). *Counter* is a variable. *A*, *B*, and *S* may be constants, variables, or expressions.
 On-Page Connectors*  Off-Page Connectors*	Flowchart sections can be connected with two different symbols. The circle connects sections on the same page, and the home base plate connects flowcharts from page to page. Inside these two symbols the programmer writes letters or numbers. The on-page connector uses letters inside the circle to indicate where the adjoining connector is located. An *A* connects to an *A*, a *B* to a *B*, etc. The off-page connectors use the page number where the next part or the previous part of the flowchart is located. This allows the reader to easily follow the flowchart. On- and off-page connectors will have either an entrance or an exit.

* These connectors should be used as little as possible. They should be used to enhance readability. Overuse decreases readability and produces a cluttered effect.

Figure 3.9 Flowchart Symbols (*continued from page 49*)

There are four algorithms, flowcharts, and pseudocode for the payroll problem because there are four different modules in the structure chart. Figure 3.10 shows how the flowcharts, algorithms, and pseudocode correspond for the payroll problem. Figure 3.11 shows the order of processing from module to module.

Flowchart

Drawing the Flowcharts

From the algorithms the programmer develops the flowcharts, graphic representations of the algorithms. The algorithms and the flowcharts are the final steps in organizing a solution. Using them, the programmer can test the solution for bugs and go on to code the problem into a computer language for entry into the computer. A flowchart will show errors in logic not readily visible in the other charts. Also, a set of data can be tested easily using a flowchart.

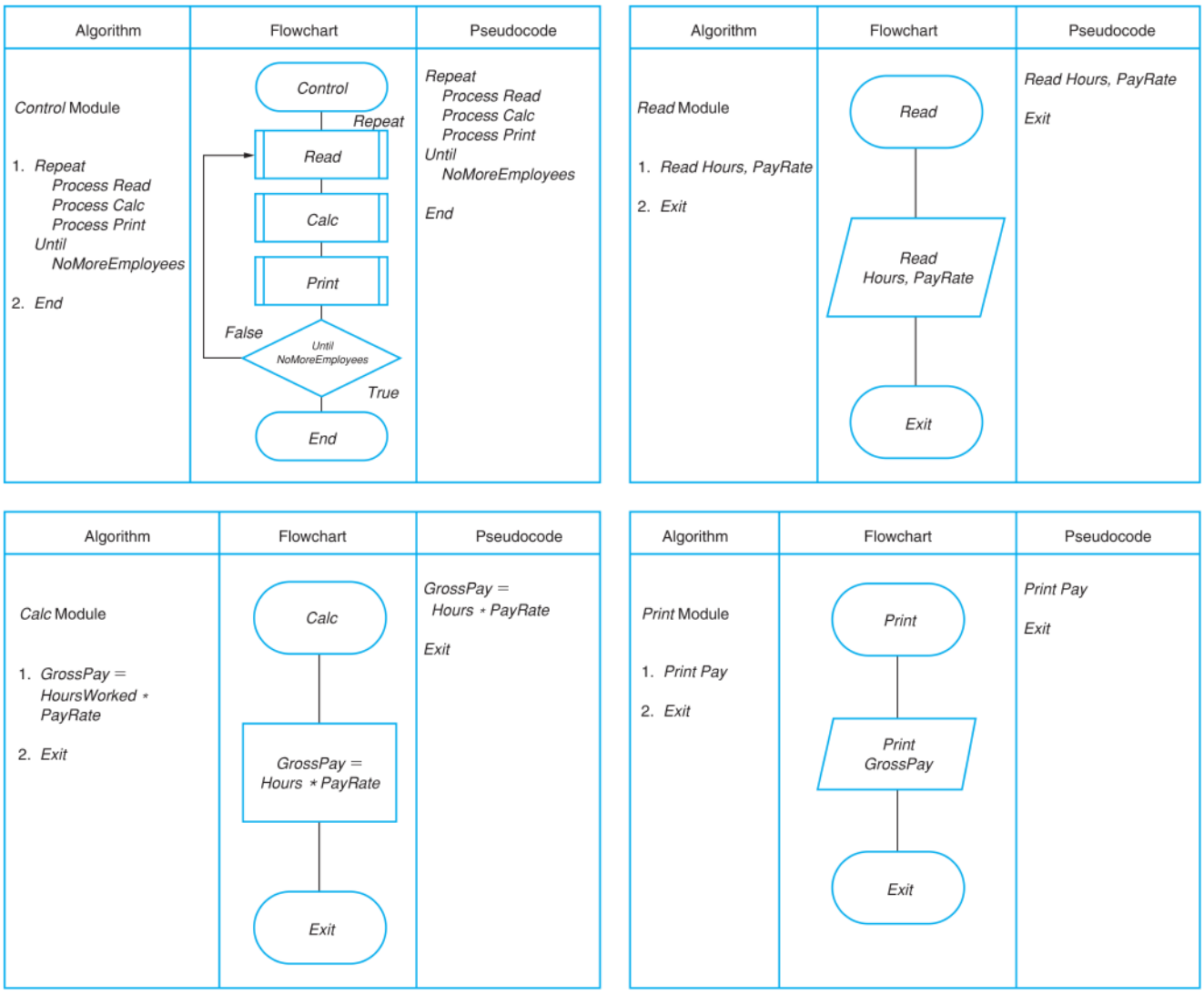


Figure 3.10 The Algorithms and Flowcharts for the Payroll Problem

There are flowchart symbols for use with various types of processing. Figure 3.9 shows and explains some general flowchart symbols. There are many specific symbols, such as printer output, monitor output, and so forth, that are used in systems flowcharts.

A flowchart shows the flow of the processing from the beginning to the end of a solution. Each block in a flowchart represents one instruction from an algorithm. *Flowlines* indicate the direction of the data flow. Most blocks have one or more *entrances,* flowlines directing the flow of the data into the block. Most blocks have only one *exit,* flowlines directing the data out of the block, since, in most cases, data can flow to only one other block. The exception to this rule is a block representing a *decision instruction,* an instruction that enables the computer to make one of two choices. A decision block has two exits, one for each choice.

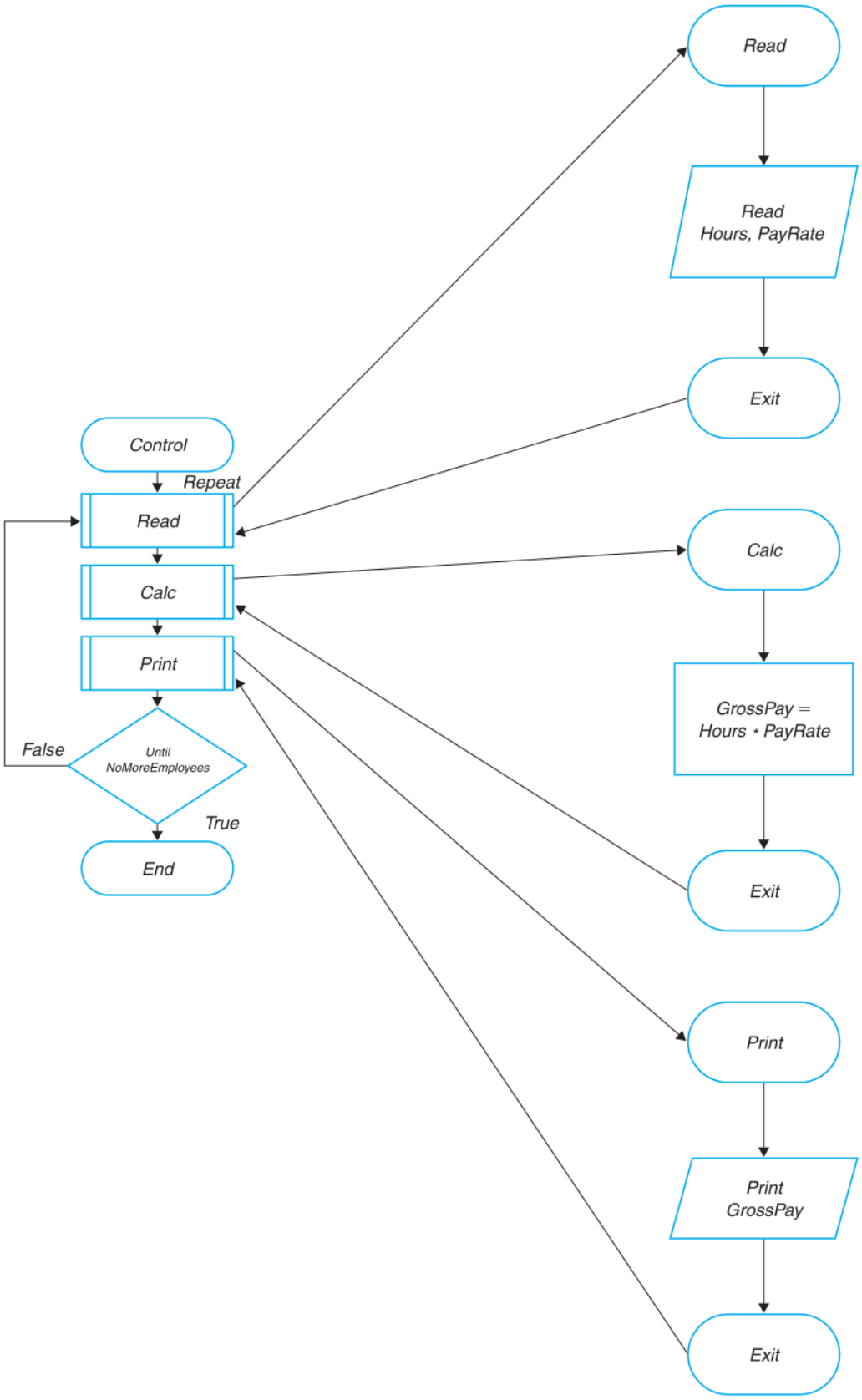


Figure 3.11 Order of Execution of Instructions

Besides the decision instructions, Figure 3.9 refers to another type of instruction that you will learn to use later, but it does deserve a brief explanation here. A *loop* enables the computer to perform a task repeatedly during the processing of a solution. An *automatic-counter loop* is a type of loop that enables the computer to count in increments of a given value; the increment can be 1, 2, 5, or whatever. That given value is the *incrementor* referred to in Figure 3.9.

Rules for Drawing Flowcharts

The following are some of the rules for drawing flowcharts:

1. You should write the instructions inside the blocks.
2. If there is something you need to remember, you can write a note beside a block. Test values of variables also can be placed beside flowchart blocks. This makes the flowchart an **annotated flowchart.**
3. A flowchart always starts at the top of the page and flows to the bottom. If you need more than one page for a flowchart, start another column on the same page or go on to another page. On- and off-page connectors are used to connect parts of the flowchart. A flowchart should not flow up, or sideways, or all over the page.
4. Use a computer program, or a template and a straightedge, to draw the flowchart. When you use a computer program or a template, the symbols are the same size and your flowchart will be neater and easier to read.
5. Make the blocks big enough to write instructions so they can be easily read.
6. Put the module number and name from the interactivity chart in the upper right-hand corner of the page for quick reference to the correct module.
7. Have plenty of paper on hand since the final copy of the flowchart normally will not be the first draft.
8. Use a pencil with a large eraser.

annotated flowchart

It is helpful to keep the algorithm and the flowchart of a module on the same page. Figure 3.12 shows a form (also in Appendix D, Figure D.4) you can use to develop your algorithms and flowcharts. Notice the last four sections of the form. The first is the Annotation section. Here you put notes about the algorithms and the flowcharts including test data, information about variables, things to remember, and so forth. The Test section is used to test the solution with sample data. The other two sections, Internal Documentation and External Documentation, are for notes to refer to later when you are completing the internal and external documentation. The last four sections of Figure 3.12 are optional but helpful in many cases.

Pseudocode

Pseudocode is similar to the algorithm without the numbers and somewhat condensed. The pseudocode for the Payroll problem is given on the right hand side of the flowcharts. It closely follows the algorithm, but is characteristically closer to what you would write in a computer language. In this book, both the algorithm and the pseudocode will be given with developed problems. Your instructor may choose to use one or the other.

Internal and External Documentation

internal documentation
external documentation

Internal documentation consists of remarks written with the instructions to explain what is being done in the program. **External documentation** is made up of the manuals or help menus written about the solution. Documentation of your program is very important.