

the next section, we discuss three popular methods used to develop the logic plan: flowcharts, pseudocode, and top-down charts. These tools help the programmer break a problem into a sequence of small tasks the computer can perform to solve the problem. Planning also involves using representative data to test the logic of the algorithm by hand to ensure that it is correct.

3. Design the interface: Select the objects (text boxes, buttons, etc.).

Determine how the input will be obtained and how the output will be displayed. Then create objects to receive the input and display the output. Also, create appropriate buttons and menus to allow the user to control the program.

4. Code: Translate the algorithm into a programming language.

Coding is the technical word for writing the program. During this stage, the program is written in Visual Basic and entered into the computer. The programmer uses the algorithm devised in Step 2 along with a knowledge of Visual Basic.

5. Test and debug: Locate and remove any errors in the program.

Testing is the process of finding errors in a program, and **debugging** is the process of correcting errors that are found. (An error in a program is called a **bug**.) As the program is typed, Visual Basic points out certain kinds of program errors. Other kinds of errors will be detected by Visual Basic when the program is executed; however, many errors due to typing mistakes, flaws in the algorithm, or incorrect use of the Visual Basic language rules can be uncovered and corrected only by careful detective work. An example of such an error would be using addition when multiplication was the proper operation.

6. Complete the documentation: Organize all the material that describes the program.

Documentation is intended to allow another person, or the programmer at a later date, to understand the program. Internal documentation (comments) consists of statements in the program that are not executed but point out the purposes of various parts of the program. Documentation might also consist of a detailed description of what the program does and how to use it (for instance, what type of input is expected). For commercial programs, documentation includes an instruction manual and on-line help. Other types of documentation are the flowchart, pseudocode, and hierarchy chart that were used to construct the program. Although documentation is listed as the last step in the program development cycle, it should take place as the program is being coded.

1.4 Programming Tools

This section discusses some specific algorithms and describes three tools used to convert algorithms into computer programs: flowcharts, pseudocode, and hierarchy charts.

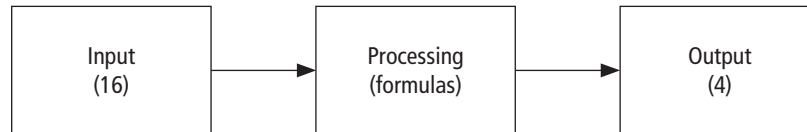
You use algorithms every day to make decisions and perform tasks. For instance, whenever you mail a letter, you must decide how much postage to put on the envelope. One rule of thumb is to use one stamp for every five sheets of paper or fraction thereof. Suppose a friend asks you to determine the number of stamps to place on an envelope. The following algorithm will accomplish the task.

1. Request the number of sheets of paper; call it Sheets. (input)
2. Divide Sheets by 5. (processing)
3. Round the quotient up to the next highest whole number; call it Stamps. (processing)
4. Reply with the number Stamps. (output)

The preceding algorithm takes the number of sheets (Sheets) as input, processes the data, and produces the number of stamps needed (Stamps) as output. We can test the algorithm for a letter with 16 sheets of paper.

1. Request the number of sheets of paper; Sheets = 16.
2. Dividing 5 into 16 gives 3.2.
3. Rounding 3.2 up to 4 gives Stamps = 4.
4. Reply with the answer, 4 stamps.

This problem-solving example can be pictured by



Of the program design tools available, three popular ones are the following:





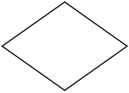


Flowcharts: Graphically depict the logical steps to carry out a task and show how the steps relate to each other.

Pseudocode: Uses English-like phrases with some Visual Basic terms to outline the task.

Hierarchy charts: Show how the different parts of a program relate to each other.

■ Flowcharts

A flowchart consists of special geometric symbols connected by arrows. Within each symbol is a phrase presenting the activity at that step. The shape of the symbol indicates the type of operation that is to occur. For instance, the parallelogram denotes input or output. The arrows connecting the symbols, called **flowlines**, show the progression in which the steps take place. Flowcharts should “flow” from the top of the page to the bottom. Although the symbols used in flowcharts are standardized, no standards exist for the amount of detail required within each symbol.

Symbol	Name	Meaning
	Flowline	Used to connect symbols and indicate the flow of logic.
	Terminal	Used to represent the beginning (Start) or the end (End) of a task.
	Input/Output	Used for input and output operations, such as reading and displaying. The data to be read or displayed are described inside.
	Processing	Used for arithmetic and data-manipulation operations. The instructions are listed inside the symbol.
	Decision	Used for any logic or comparison operations. Unlike the input/output and processing symbols, which have one entry and one exit flowline, the decision symbol has one entry and two exit paths. The path chosen depends on whether the answer to a question is “yes” or “no.”
	Connector	Used to join different flowlines.
	Annotation	Used to provide additional information about another flowchart symbol.

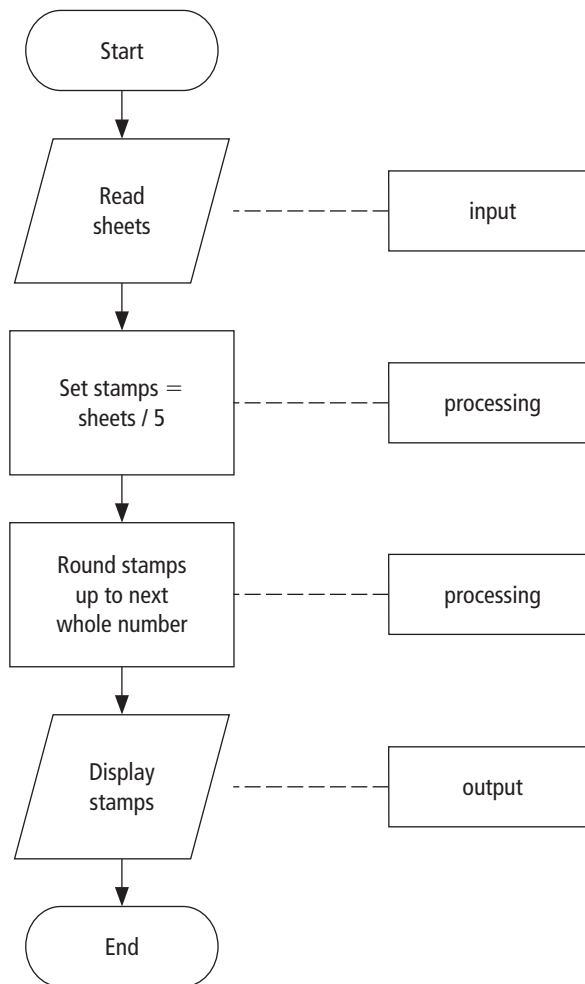


FIGURE 1.4 Flowchart for the postage-stamp problem.

The table of the flowchart symbols shown on the previous page has been adopted by the American National Standards Institute (ANSI). Figure 1.4 shows the flowchart for the postage-stamp problem.

The main advantage of using a flowchart to plan a task is that it provides a pictorial representation of the task, which makes the logic easier to follow. We can clearly see every step and how each is connected to the next. The major disadvantage is that when a program is very large, the flowcharts may continue for many pages, making them difficult to follow and modify.

■ Pseudocode

Pseudocode is an abbreviated plain English version of actual computer code (hence, *pseudocode*). The geometric symbols used in flowcharts are replaced by English-like statements that outline the process. As a result, pseudocode looks more like computer code than does a flowchart. Pseudocode allows the programmer to focus on the steps required to solve a problem rather than on how to use the computer language. The programmer can describe the algorithm in Visual Basic-like form without being restricted by the rules of Visual Basic. When the pseudocode is completed, it can be easily translated into the Visual Basic language.

The following is pseudocode for the postage-stamp problem:

Program: Determine the proper number of stamps for a letter	
Read Sheets	(input)
Set the number of stamps to $\text{Sheets} / 5$	(processing)
Round the number of stamps up to the next whole number	(processing)
Display the number of stamps	(output)

Pseudocode has several advantages. It is compact and probably will not extend for many pages as flowcharts commonly do. Also, the plan looks like the code to be written and so is preferred by many programmers.

■ Hierarchy Chart

The last programming tool we'll discuss is the **hierarchy chart**, which shows the overall program structure. Hierarchy charts are also called structure charts, HIPO (Hierarchy plus Input-Process-Output) charts, top-down charts, or VTOC (Visual Table of Contents) charts. All these names refer to planning diagrams that are similar to a company's organization chart.

Hierarchy charts depict the organization of a program but omit the specific processing logic. They describe what each part, or **module**, of the program does and they show how the modules relate to each other. The details on how the modules work, however, are omitted. The chart is read from top to bottom and from left to right. Each module may be subdivided into a succession of submodules that branch out under it. Typically, after the activities in the succession of submodules are carried out, the module to the right of the original module is considered. A quick glance at the hierarchy chart reveals each task performed in the program and where it is performed. Figure 1.5 shows a hierarchy chart for the postage-stamp problem.

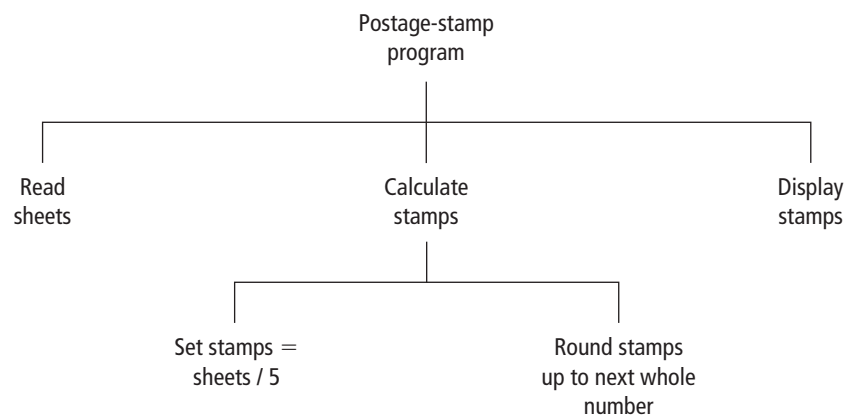


FIGURE 1.5 Hierarchy chart for the postage-stamp problem.

The main benefit of hierarchy charts is in the initial planning of a program. We break down the major parts of a program so we can see what must be done in general. From this point, we can then refine each module into more detailed plans using flowcharts or pseudocode. This process is called the **divide-and-conquer** method.

■ Decision Structure

The postage-stamp problem was solved by a series of instructions to read data, perform calculations, and display results. Each step was in a sequence; that is, we moved from one line to the next without skipping over any lines. This kind of structure is called a **sequence structure**. Many problems, however, require a decision to determine whether a series of instructions should be executed. If the answer to a question is “yes,” then one group of instructions is

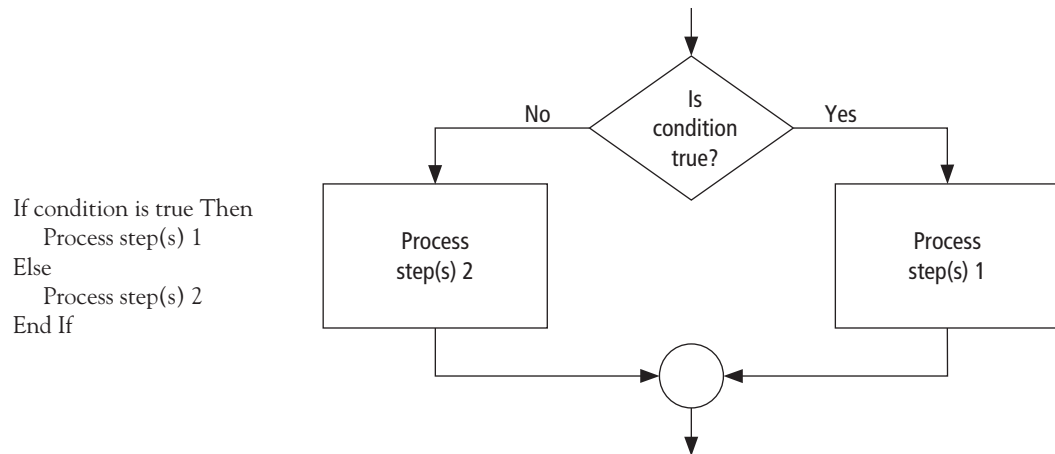


FIGURE 1.6 Pseudocode and flowchart for a decision structure.

executed. If the answer is “no,” then another is executed. This structure is called a **decision structure**. Figure 1.6 contains the pseudocode and flowchart for a decision structure.

Sequence and decision structures are both used to solve the following problem.

■ Direction of Numbered NYC Streets Algorithm

Problem: Given a street number of a one-way street in New York, decide the direction of the street, either eastbound or westbound.

Discussion: There is a simple rule to tell the direction of a one-way street in New York: Even-numbered streets run eastbound.

Input: Street number.

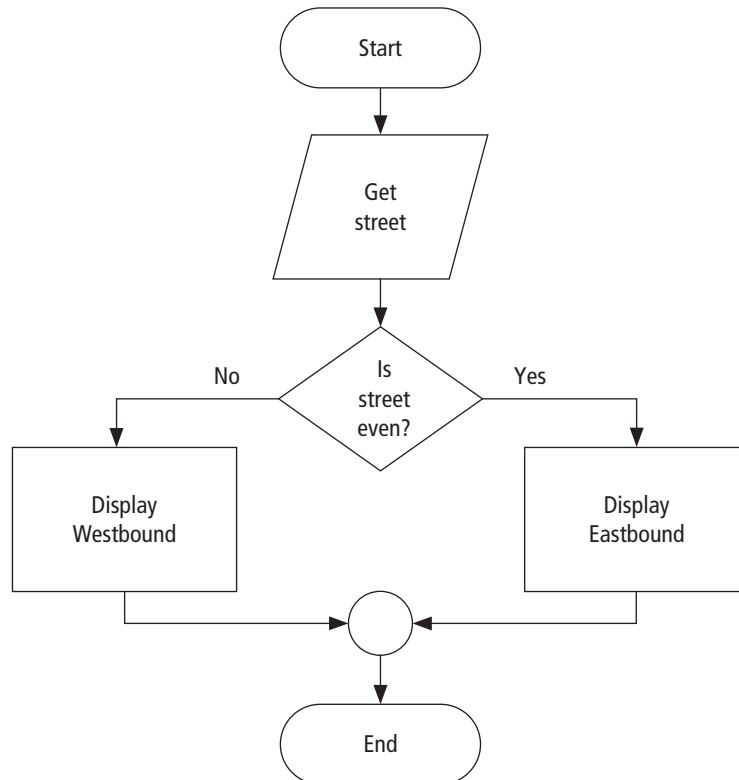


FIGURE 1.7 Flowchart for the numbered New York City streets problem.

Program: Determine the direction of a numbered NYC street.
 Get street
 If street is even Then
 Display Eastbound
 Else
 Display Westbound
 End If

FIGURE 1.8 Pseudocode for the numbered New York City streets problem.

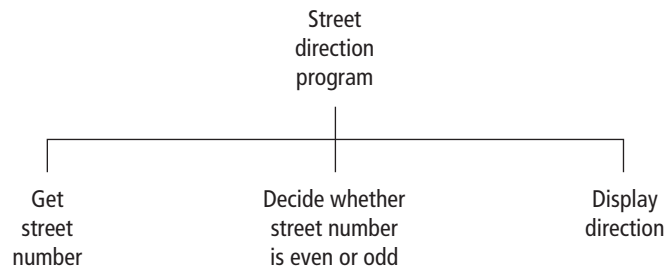


FIGURE 1.9 Hierarchy chart for the numbered New York City streets problem.

Processing: Decide if the street number is divisible by 2.

Output: “Eastbound” or “Westbound”.

Figures 1.7 through 1.9 show the flowchart, pseudocode, and hierarchy chart for the numbered New York City streets problem.

■ Repetition Structure

A programming structure that executes instructions many times is called a **repetition structure** or a **loop structure**. Loop structures need a test (or condition) to tell when the loop should end. Without an exit condition, the loop would repeat endlessly (an infinite loop). One way to control the number of times a loop repeats (often referred to as the number of passes or iterations) is to check a condition before each pass through the loop and continue executing the loop as long as the condition is true. See Fig. 1.10. The solution of the next problem requires a repetition structure.

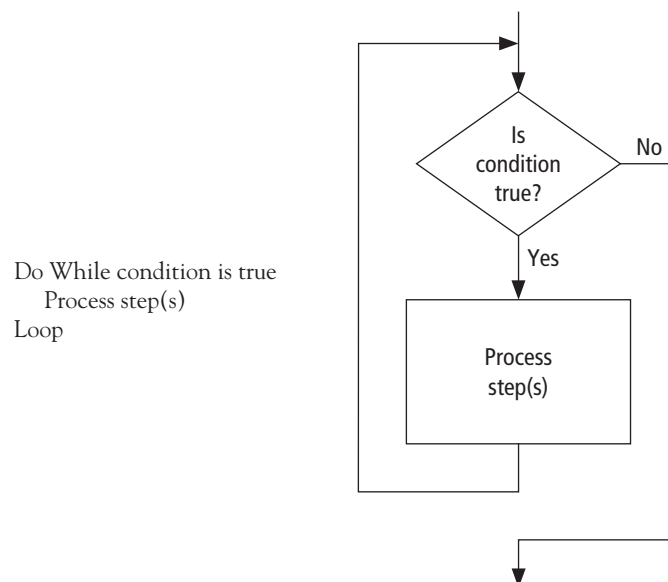


FIGURE 1.10 Pseudocode and flowchart for a loop.

■ Class Average Algorithm

Problem: Calculate and report the average grade for a class.

Discussion: The average grade equals the sum of all grades divided by the number of students. We need a loop to read and then add (accumulate) the grades for each student in the class. Inside the loop, we also need to total (count) the number of students in the class. See Figs. 1.11 to 1.13.

Input: Student grades.

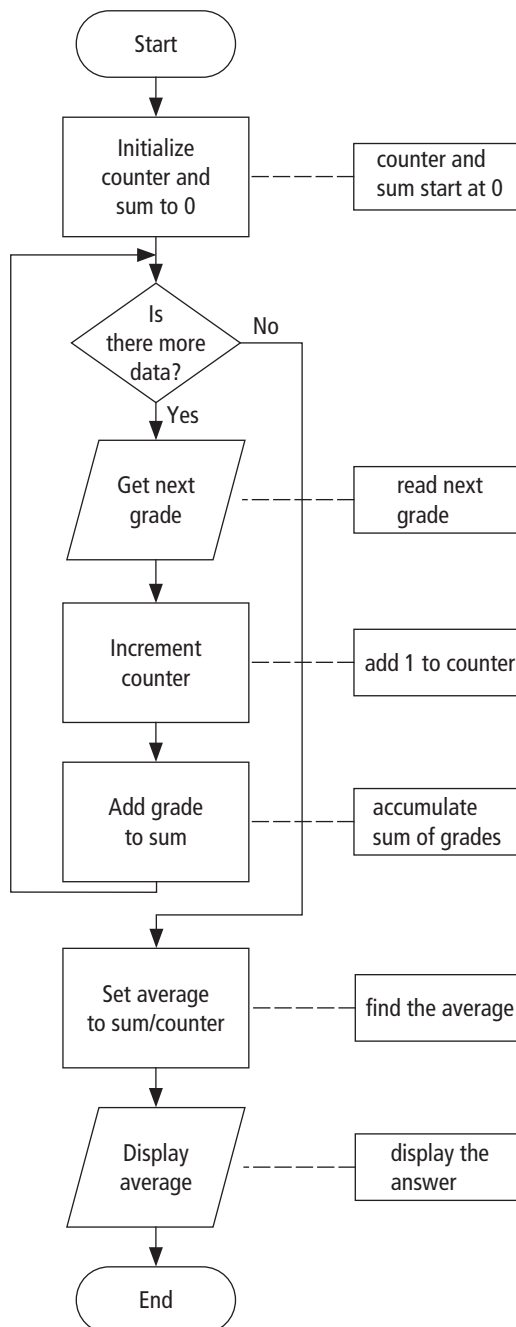


FIGURE 1.11 Flowchart for the class average problem.

Program: Calculate and report the average grade of a class.
 Initialize Counter and Sum to 0
 Do While there is more data
 Get the next Grade
 Increment the Counter
 Add the Grade to the Sum
 Loop
 Compute Average = Sum/Counter
 Display Average

FIGURE 1.12 Pseudocode for the class average problem.

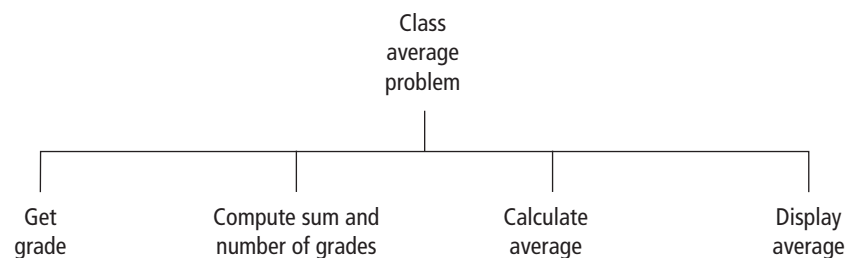


FIGURE 1.13 Hierarchy chart for the class average problem.

Processing: Find the sum of the grades; count the number of students; calculate average grade = sum of grades / number of students.

Output: Average grade.

■ Comments

1. Tracing a flowchart is like playing a board game. We begin at the Start symbol and proceed from symbol to symbol until we reach the End symbol. At any time, we will be at just one symbol. In a board game, the path taken depends on the result of spinning a spinner or rolling a pair of dice. The path taken through a flowchart depends on the input.
2. The algorithm should be tested at the flowchart stage before being coded into a program. Different data should be used as input, and the output checked. This process is known as **desk checking**. The test data should include nonstandard data as well as typical data.
3. Flowcharts, pseudocode, and hierarchy charts are universal problem-solving tools. They can be used to construct programs in any computer language, not just Visual Basic.
4. Flowcharts are used throughout this text to provide a visualization of the flow of certain programming tasks and Visual Basic control structures. Major examples of pseudocode and hierarchy charts appear in the case studies.
5. Flowcharts are time consuming to write and difficult to update. For this reason, professional programmers are more likely to favor pseudocode and hierarchy charts. Because flowcharts so clearly illustrate the logical flow of programming techniques, however, they are a valuable tool in the education of programmers.



6. There are many styles of pseudocode. Some programmers use an outline form, whereas others use a form that looks almost like a programming language. The pseudocode appearing in the case studies of this text focuses on the primary tasks to be performed by the program and leaves many of the routine details to be completed during the coding process. Several Visual Basic keywords, such as If, Else, Do, and While, are used extensively in the pseudocode appearing in this text.