Introduction to Programming

LESSON 1

If you're new to programming, you might be intimidated by code and flowcharts. You might even wonder how you'll ever understand them. This lesson offers some basic ideas and programming examples with which you can follow along. You'll soon understand the logic behind software programs.

In your first assignment, you'll read an introduction to computers and programming. In the second assignment, you'll learn the steps in creating a program. In the third assignment, you'll learn about modules and the best practices for creating a software program.

OBJECTIVES

When you complete this lesson, you'll be able to

- Describe basic programming concepts and steps
- Declare variables
- Describe *documentation*
- Discuss the importance of *structure*
- Create modules
- Pass arguments
- Describe *global variables* and *constants*

This study guide is designed to break down the material into easy-to-understand chunks so that you aren't intimidated by the material. Follow the directions outlined in the assignments, read the assigned text, and answer the questions to get the most from this course. You'll soon understand the basics of programming.

ASSIGNMENT 1: AN OVERVIEW OF COMPUTERS AND PROGRAMMING

Read this assignment, and then read pages 1–26 in your textbook. When you're comfortable with the material presented in the text, answer the review questions in the textbook and complete the assigned exercises to gauge your progress. If there are any areas with which you don't feel comfortable, reread those sections in the textbook before returning to the study guide. When you feel you've mastered the material in Chapter 1, move on to the next assignment.

> *Note:* Make sure to answer the checkpoints in the textbook as you read along. Check your answers in Appendix D, which is located on a PDF file on the CD included at the back of your textbook.

Computers are a big part of people's lives today, but what makes them capable of doing such powerful tasks? The answer is that they're programmed to perform *tasks,* or what someone tells them to do. A *program* is a set of instructions for the computer to follow; programs are often referred to as *software.* Someone who creates programs or software is known as a *programmer* or *software developer.*

Hardware refers to the physical parts of a computer. You'll read examples of different types of hardware, such as memory, a central processing unit (CPU), storage devices, input devices, and output devices.

A computer needs both hardware and software to perform its four major functions: input, processing, output, and storage. Programmers tell the computer what to do by using programming languages, such as Visual Basic and C#. The programming language must follow certain rules, called *syntax,* for it to be understood. The commands that tell the computer what to do also must follow a certain sequence, or *logic,* so the computer can process the request.

Computers store data in *bytes,* which are tiny storage locations. Each byte is divided into eight smaller units called *bits.* These bits act like switches because they're either "on" or "off." When bits represent numbers, or *binary numbers,*

they're assigned the number 0 when off and 1 when on. You'll read more about how numbers and characters are stored in this chapter.

Machines understand *machine language,* which is difficult for people to communicate in. Therefore, program languages that instruct computers on what to do have been created. Types of programming languages include C, C++, Java, JavaScript, and Visual Basic, as well as several more. These are known as *high-level languages.* Programmers write statements known as *source code*. High-level languages need to be translated into machine language. A program that handles this translation is called a *compiler.*

After programs are created, they're store on a storage device, such as a disk drive. When the CPU runs, or *executes,* a program, it follows three steps:

1. Fetch
2. Decode
3. Execute

Lastly, you'll read about types of software. *System software* manages computer operations and includes operating systems and software development tools. *Application software* is used for tasks, which include word processing, e-mail, and spreadsheets.

Now, answer the review questions at the end of Chapter 1, then complete Exercises 1–4 on page 26 of your textbook. Log in to your **Student Area** at **http://www.pennfoster.edu.** Go to **My Courses** and look for the **Solutions** link associated with this course. After you complete the review questions and exercises, move on to the next assignment.

ASSIGNMENT 2: INPUT, PROCESSING, AND OUTPUT

Read this assignment, and then read pages 27–78 in your textbook. When you're comfortable with the material presented in the text, answer the review questions in the textbook and complete the assigned exercises to gauge your progress. If there are any areas with which you don't feel comfortable, reread those sections in the textbook before returning to the study guide. When you feel you've mastered the material in Chapter 2, move on to the next assignment.

Note: Make sure to answer the checkpoints in the textbook as you read along. Check your answers in Appendix D, which is located on a PDF file on the CD included at the back of your textbook.

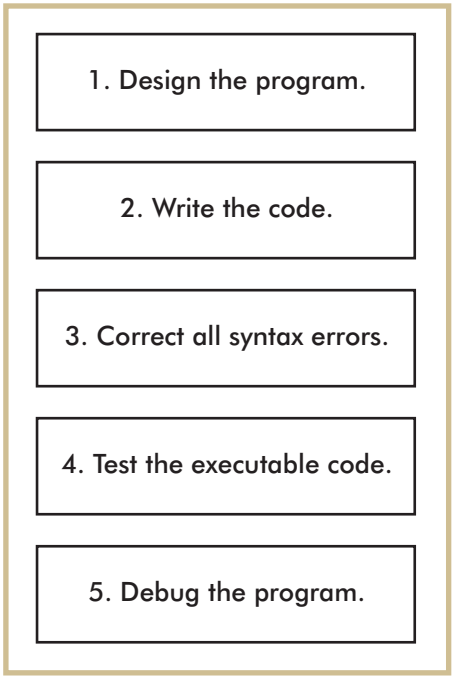

FIGURE 1—Steps of Program Creation

Programmers perform five steps when creating a program (Figure 1).

Syntax is the programming language's rules. *Debugging* refers to fixing any errors in the code so the program will work correctly. You'll read about each of these steps in greater detail in this chapter.

When planning the logic of a program, programmers will often use flowcharts or pseudocode. A *flowchart* is a pictorial representation of the steps in a program. *Pseudocode* is an English-like representation of the steps. See Appendix B and Appendix C in your textbook for more information on flowcharts and pseudocode.

Computer programs work by following these three steps:

1. Receive input (data)
2. Perform a process
3. Produce output (such as displaying information or performing a task)

Program code can become quite complicated. When a collection of program statements becomes difficult to read, it's called *spaghetti code* because it's like following a noodle in a bowl of spaghetti. Therefore, keep your code simple by using *struc-*

tures. A *sequence* is one kind of structure (Figure 2). You can stack structures on each other at entrance or exit points. Putting one structure within another is called *nesting*.

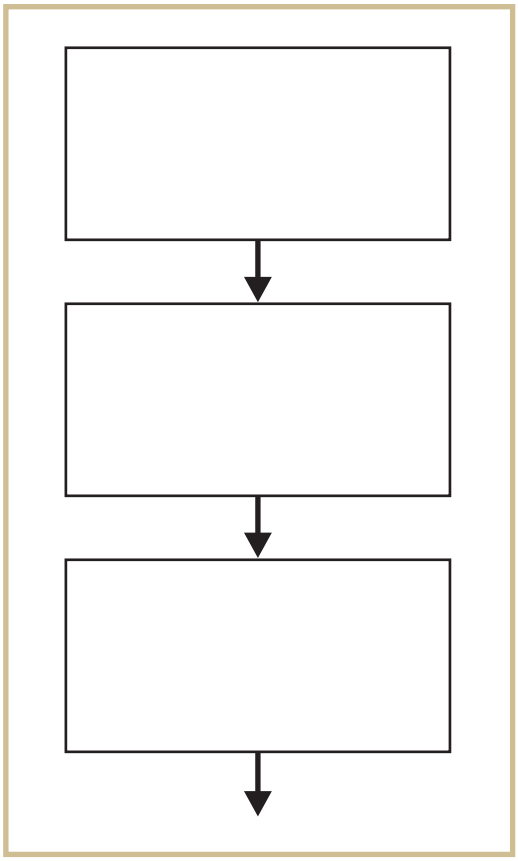

FIGURE 2—A Sequence

Programmers create variables that stand for different values. The name of a variable should be no more than one word. Programmers end programs by creating a *decision*, which is represented in a flowchart by a diamond. If the flowchart runs onto another page, they use a *connector*. When assigning values to variables, programmers often use the equal sign (=).

Many programs require some sort of calculation to be performed, such as addition or multiplication. In this chapter, you'll read about these and other common math operations, as well as the order of operations.

In most programs, variables must be *declared* before they can be used. This means that you must name the memory location and specify the type of data. *Variable names* are usually nouns, such as `payRate`, and *types* are the type of data the variable will hold (numbers or letters, for example).

Two types of documentation are generated during the programming process. *Internal program documentation* is for the programmer and includes program comments that identify the logic of the program. *External documentation* is for the people who use your programs—the end users—who will need documentation to know how to use your program. This type of documentation is also called *user documentation.*

Now, answer the review questions at the end of Chapter 2. In addition, complete the following exercises at the end of the chapter:

- Algorithm Workbench 1, 5, and 11
- Debugging Exercises 1 and 2
- Programming Exercises 1 and 7

Log in to the **Student Area** at **http://www.pennfoster.edu.** Go to **My Courses** and look for the **Solutions** link associated with this course. After you complete the review questions and exercises, move on to the next assignment.

ASSIGNMENT 3: MODULES

Read this assignment, and then read pages 79–119 in your textbook. When you're comfortable with the material presented in the text, answer the review questions in the textbook and complete the assigned exercises to gauge your progress. If there are any areas with which you don't feel comfortable, reread those sections in the textbook before returning to the study guide. When you feel you've mastered the material in Chapter 3, move on to the next assignment.

Note: Make sure to answer the checkpoints in the textbook as you read along. Check your answers in Appendix D, which is located on a PDF file on the CD included at the back of your textbook.

Programmers usually break down a program into separate units called *modules*. *Modularization,* or breaking programs into modules, is beneficial because it

- Uses simpler code
- Promotes code reuse
- Enables better testing
- Allows multiple programmers to work on a program

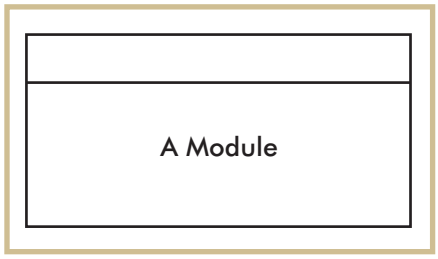

FIGURE 3—In a flowchart, modules are depicted by a rectangle with a bar across the top.

When naming a module, be sure to give it a one-word name that identifies it. Module names are often verbs, such as `calculatePayment`. A separate flowchart is created for each module. The module is *called* by placing the module's name in a rectangle with bars across the sides (Figure 3). Note that modules can also call other modules.

To keep track of modules, you can create a *hierarchy chart,* which shows the relationship between modules and demonstrates which modules call which other modules (Figure 4).

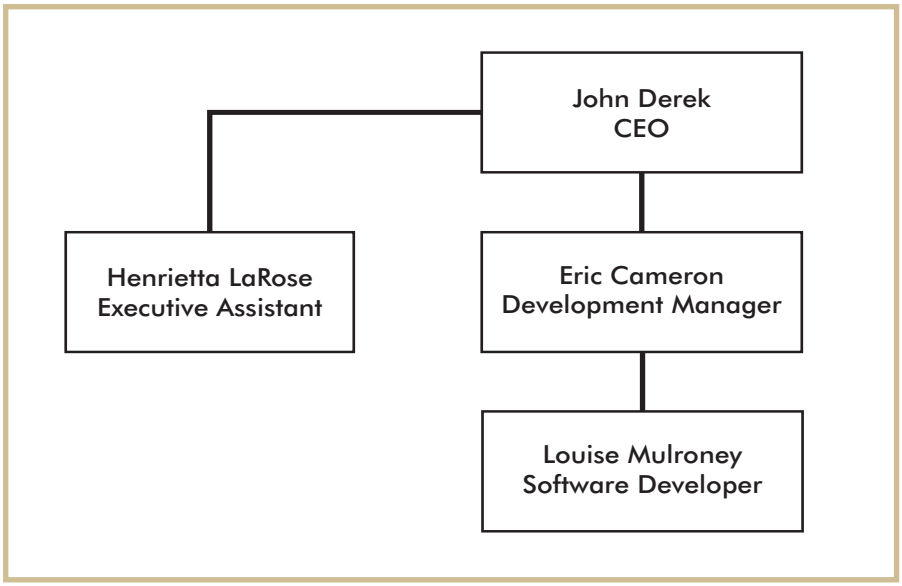

FIGURE 4—A hierarchy chart shows the relationships between modules just, as a company's organizational chart shows the relationships within the company or department.

Local variables are those declared inside modules. *Global variables* are those that can be accessed by all modules in a program; they're declared outside modules.

Arguments are pieces of data sent into modules. *Parameters* are variables that receive arguments passed into modules. You'll read about various ways to pass arguments into modules, such as passing arguments by value and by reference.

When you create a program, you should develop a plan before you start so that you create a sound program design. One way to facilitate sound program design is to store your programming components in separate files so that you can easily find, read, and reuse segments. Another good practice is to choose suitable names for variables and modules. The following list contains some tips for naming variables and modules:

- Use meaningful names.
- Use names that are easily pronounced.
- Choose abbreviations that are easily understood by others.
- Avoid using numbers.
- Use both upper- and lowercase letters *(camel case),* if possible, in multiword names, or use dashes or underscores to separate words.
- Use verbs in status names.

To create programs with good design, you should also divide your line breaks consistently. Note that shorter lines are easier to read and follow. In addition, consider using temporary variables when you have a long statement so that you can organize multiple variables.

Finally, you should use constants instead of numbers in your programs when possible. For example, suppose you have a value for an employee's pay rate. Preferably, you should store the value as `payRate` rather than `10` (for $10 an hour), because the employee's pay rate might change if he or she receives a raise.

Now, answer the review questions at the end of Chapter 3. In addition, complete the following exercises at the end of the chapter:

- Algorithm Workbench 1 and 2
- Debugging Exercises 1 and 4
- Programming Exercise 1

Check your answers by logging in to the **Student Area** at **http://www.pennfoster.edu.** Go to **My Courses** and look for the **Solutions** link associated with this course.

After completing the review questions and exercises, complete the first graded project. After you complete the graded project, move on to Lesson 2.

Lesson 1 Graded Project

Project Number: 41882700

INSTALLING VISIO

You must use Microsoft Visio to create flowcharts in this course. If you've never worked with Visio before, visit the Getting Started Guide at **http://visio.microsoft.com.**

You can download a free 60-day trial version by following these steps:

1. Visit **http://visio.microsoft.com** and click **Download Free Visio Trial.**
2. Follow the on-screen prompts to download the trail.
3. Open Visio from the Start Menu. It may appear in the Microsoft Office menu.
4. Activate your trial version after you open Visio.

Note: The system requirements for the Visio trial are Windows 7, Windows Vista, Windows XP with Service Pack (SP) 2, Windows Server 2003 with SP1, or another later operating system.

This graded project will test your knowledge of what you've learned in this course so far. If you've read all the material in each assignment and completed the checkpoints, review questions, and assigned exercises in the text, you shouldn't have any problems successfully completing these tasks.

Complete Exercise 6 on page 118 by creating both

- Pseudocode
- A flowchart

Scenario

A personal trainer asks you to create a program to calculate and displays client's body mass index (BMI). Use the information in Exercise 6 on page 118 to create this program by

1. Creating the pseudocode for this program
2. Creating a flowchart for this program

Hint: Use Visio or Microsoft Word's Drawing toolbar.

Graded Project

Submitting Your Project

Follow this procedure to submit your assignment online:

1. On your computer, save a revised and corrected version of your project. Make sure to include the project number (41882700) to identify the project.
2. Go to **http://www.takeexamsonline.com** and log in.
3. Go to **My Courses.**
4. Click on **Take Exam** next to the lesson on which you're working
5. Enter your e-mail address in the box provided. (*Note:* This information is required for online submission.)
6. Attach your file or files as follows:
   - Click on the **Browse** box.
   - Locate the file you wish to attach.
   - Double-click on the file.
   - Click on **Upload File.**
7. Click on **Submit Files.**

Grading

Your instructor will grade your project as follows:

■ Used the lessons learned in this course to create the pseudocode for this program	25 points
■ Created the modules for an effective program	25 points
■ Created a flowchart for this program using lessons learned in this course	25 points
■ Used the appropriate flowchart shapes to create a flowchart	25 points
TOTAL POINTS	100 points