

Chapter 1

An Overview of Computer Programming

At a Glance

Instructor's Manual Table of Contents

- Overview
- Objectives
- Teaching Tips
- Quick Quizzes
- Class Discussion Topics
- Additional Projects
- Additional Resources
- Key Terms

Lecture Notes

Overview

General programming concepts are introduced in Chapter 1. This material provides a foundation for the remaining chapters in the book. Several topics are covered, beginning with a review of computer components, both hardware and software, and their operations. This is followed by a general discussion of program logic. Next, the chapter reviews the evolution of programming models and introduces object-oriented programming, followed by an introduction to the steps in the programming process. A presentation on flowcharts and pseudocode statements will be featured along with an explanation of how these tools greatly improve the planning stage. The chapter concludes with material describing programming tools and user environments.

Chapter Objectives

In this chapter, your students will learn about:

- Computer components and operations
- Simple program logic
- The evolution of programming models
- The steps in the programming process
- Pseudocode and flowcharts
- Program comments
- Programming and user environments

Teaching Tips

Understanding Computer Components and Operations

1. Outline the two major components of any computer system: hardware and software. Provide students with a brief introduction to each component.

Teaching Tip	Remind students that software can be classified as application software or system software. Application software comprises all of the programs you apply to a task: word-processing programs, spreadsheets, payroll and inventory programs, and even games. System software comprises the programs that you use to manage your computer, including operating systems such as Windows or UNIX, and other utility programs not used directly by end-users.
---------------------	--

2. Outline the three major operations computer hardware and software accomplish (input, processing, and output).

3. Discuss the input operation in more detail. Explain the purpose of input devices. Provide examples of input devices (e.g. mice, keyboards), and explain how they differ.
4. Discuss the processing operation in more detail and the three tasks involved. Explain the piece of hardware that performs the sorting of processing tasks (central processing unit (CPU)).
5. Discuss the output operation in more detail. Explain the use of output devices to view, interpret, and use the results. Describe common data storage devices such as disks and flash media.

Teaching Tip	Be sure students understand the difference between data and information. Data includes text, numerical information, or other information that is processed by a computer. However, many computer professionals reserve the term <i>information</i> for data that has been processed.
---------------------	--

6. Explain how computer instructions are written using a computer programming language. Provide examples of various types of languages such as Visual Basic, C#, C++, or Java. Point out that some programmers work exclusively in one language, while others know several languages and use the one that seems most appropriate for the given task.
7. Discuss programming language syntax (e.g. correct spelling and punctuation). Explain that the syntax must be perfect for a computer to interpret the instructions.
8. Explain how computer memory (RAM) stores a program and how it is usually desirable to store the program on a permanent storage device so that it can be loaded into memory later.
9. Explain that when a program runs, or executes, it carries out its instructions.
10. Discuss machine language and the need to translate programming languages into the computer's on/off circuitry language.
11. Discuss the purpose of compilers and interpreters, and explain the differences between them. Compilers create and store machine instructions, and these instructions are reusable. Interpreters interpret the programming language at run-time and must do so each time the program runs. Briefly introduce the topic of scripting languages, and explain that they use interpreters.

Teaching Tip	Find a list of links to "Free Compilers and Interpreters for Programming Languages" at the following Web site: www.thefreecountry.com/compilers/index.shtml .
---------------------	--

Understanding Simple Program Logic

1. Explain the difference between syntax and logic errors, emphasizing that the compiler or interpreter identifies syntax errors; however, logic errors are found when the program runs and may be difficult to identify.
2. Explain the concept of programming logic. Use the cake-mixing example on Page 4 to clarify the concept. Point out that programs may only be executed or run after the program is compiled successfully without syntax errors.
3. Use the program instruction example on Page 5 that takes a number (an input step), doubles it (processing), and provides the answer (output) to demonstrate writing code using statements resembling the English language. Discuss the three computer operations available in the example, and stress that they are independent of hardware.

**Teaching
Tip**

See the “How Stuff Works” Web site for more information on Understanding Computer Components and Operations: <http://computer.howstuffworks.com>.

Understanding the Evolution of Programming Models

1. Discuss characteristics of the oldest programming languages. These languages required programmers to work with memory addresses and to memorize awkward codes associated with machine languages. Point out that the earliest language dates back to the 1940s.
2. Discuss characteristics of the modern programming languages. For example, they look much more like natural language and are easier for programmers to use.
3. Point out the reasons for the ease of use in modern programming languages. For example, they allow programmers to give meaningful names to memory locations instead of using awkward memory addresses. Newer programming languages also allow the creation of self-contained modules or program segments that may be pieced together in a variety of ways.
4. Describe the two major program development techniques (procedural programming and object-oriented programming, or OOP). Identify the focus of each, and highlight that the differences are in the earliest planning stages of a project.
5. Discuss the two application types originally used with object-oriented programming: computer simulations and graphical user interfaces (GUI). Briefly explain each type.

Teaching Tip	Alan Turing proposed the basis for most modern software in 1935. Visit the following Web site for more information on Alan Turing: www.alanturing.net .
Teaching Tip	To see “The Evolution of Object-Oriented Languages” article, visit the following Web site: www.developer.com/design/article.php/10925_3493761_1 .
Teaching Tip	To see an interesting timeline of some of the key events of components and object-oriented programming in the last 50 years, see “History-making components: Tracing the roots of components from OOP through WS” at the following Web site: www.ibm.com/developerworks/webservices/library/co-tmlne .

Quick Quiz 1

1. What are the three major operations accomplished by hardware and software?
Answer: input, processing, and output
2. True or False: Syntax does not have to be perfect in order for the computer to execute the program in a production environment.
Answer: False
3. Two major program development techniques are ____ and ____.
Answer: procedural programming, object-oriented programming or OOP
4. Which of the following best describes a syntax error?
 - A. Syntax errors are found during user data input processing.
 - B. Syntax errors are identified by the compiler or interpreter.
 - C. Syntax errors are found when the program is run.
 - D. Syntax errors may be identified by the hardware device driver code.Answer: B
5. Which of the following best describes a logic error?
 - A. Logic errors are found during user data input processing.
 - B. Logic errors are identified by the compiler or interpreter.
 - C. Logic errors are found when the program is run.
 - D. Logic errors may be identified by the hardware device driver code.Answer: C

6. When you use a(n) ____, an entire program is translated before it can execute.

Answer: compiler

7. When you use a(n) ____, each instruction is translated prior to execution.

Answer: interpreter

Understanding the Steps in the Programming Process

1. Introduce the three steps in developing an object-oriented programming application system. First, the program or system is analyzed. Second, the overall design of the system is created. The third step is writing the program. Explain that writing the program has several substeps: planning the logic, coding the program, translating the code into machine language, and testing. Note that there are other tasks associated with putting the program into production and maintaining it.

Analyzing the Program or System

1. Describe the purpose of programming, which is to meet the needs of the user. Emphasize that thoroughly understanding a problem may be one of the most difficult aspects of programming for several reasons.
 - Vague description of what the user needs
 - Users may not even really know what they want
 - Users who think they know what they want frequently change their minds after seeing sample output

Teaching Tip	Read the following article on “Recommended Requirements Gathering Practices:” http://www.clearspecs.com/downloads/ClearSpecs58V01_Recommended_Reqs_Gathering_Practices_Young.pdf
---------------------	--

Designing the Program or System

1. Describe how to design a system using an object-oriented approach, starting by envisioning the objects that will be needed, their attributes (characteristics), and their relationships to each other. Mention that this process is called object-oriented design (OOD).
2. Provide examples of entities, attributes, and relationships in a product ordering system as described on Page 9.
3. Briefly introduce the term class— a general category that describes entities. Note that classes that already exist can be reused or modified.

Writing and Testing Programs

1. Discuss the task of writing the program. Explain that this task involves four steps: developing the program's logic, coding the program, translating it into machine language, and testing the program.
2. Explain that planning the program's logic is the heart of the programming process. The programmer must decide which steps to include and how to order them.
3. Briefly touch on the two most common planning tools: flowcharts and pseudocode. Explain the common feature of both tools (writing the program steps in English). These will be covered more thoroughly later in the chapter.
4. Point out that the programmer is not concerned about programming language syntax at this point.
5. Define the term desk-checking, which is reviewing a program's logic on paper.

Teaching Tip

Inform students that programmers often refer to planning a program as “developing an algorithm.” An algorithm is the sequence of steps necessary to solve any problem.

Coding the Program

1. Explain that a programmer codes the statements needed using a programming language. Briefly name some common object-oriented programming languages. Discuss their similarity, namely that each language can handle creating objects and establishing communication between them.
2. Point out that syntax (e.g. correct spelling and punctuation) becomes a concern only after the programming language is selected.
3. Explain that sometimes programmers can successfully combine the planning and the actual instruction writing (coding) of the program into one step. Use the example of writing a postcard to a friend, as described on Page 11, to illustrate that this applies to simple programs.
4. Point out that planning the program is most often more challenging than coding it.

Using Software to Translate the Program into Machine Language

1. Review with students that there are many programming languages, but each computer knows only one language: its machine language, which consists of many 1s and 0s.

2. Discuss how translator programs (compilers and interpreters) change high-level programming language that resembles the English language, into low-level machine language that the computer understands. Point out that the compiler or interpreter identifies any syntax errors in the code. Note that all syntax errors must be corrected before the program becomes executable.
3. Use Figure 1-1 on Page 11 to explain the cycle of syntax error detection and correction.

Testing the Program

1. Explain that testing for logical errors occurs after the program is free of syntax errors. Point out that a program that is free of syntax errors is not necessarily free of logical errors.
2. Explain that testing usually requires entering sample data to see whether the results are logically correct. Point out that selecting a proper set of test data is critical to thoroughly test the program.
3. Return to the number-doubling example presented on Page 12, discussing the choice of appropriate sample data to test the program and the different types of errors that could be discovered.

After the Program is Written and Tested

1. Explain that once the program has been tested adequately, it is ready for the organization to use.
2. Discuss additional tasks that may be necessary:
 - Preparing manuals
 - Training users
 - Converting existing data to a format that is usable by the new system
3. Describe that conversion is the entire set of actions an organization must take to switch over to using a new program or set of programs, and it can sometimes take months or years to accomplish.
4. Explain that maintenance is the process of making required changes after a program has been put into production. Point out that maintenance is necessary for many reasons, including fixing errors, updating values, changing the format of input data, and adding functionality. Provide some examples of these, found on Page 13.

Using Pseudocode and Flowcharts

1. Introduce the use of pseudocode and flowcharts as tools to plan programs. Explain that pseudocode is an English-like representation of program steps and that a flowchart is a pictorial representation of the same thing.

Writing Pseudocode

1. Show the example of pseudocode on Page 14. Discuss how pseudocode might look very much like a programming language, except without the strict rules of syntax. Note that starting and ending statements are usually used.

Drawing Flowcharts

1. Explain what a flowchart is and how it is used. Note that when you create a flowchart, you write program steps in geometric shapes that you connect with arrows (flowlines) to show the order and flow of the program logic.
2. Discuss the five types of flowchart symbols and their use. Use the flowcharts in Figure 1-2 on Page 16 and Figure 1-3 on Page 17 as examples. Note that top-to-bottom or left-to-right flow is preferred.
3. Note that there are several software packages that have flowcharting tools, and some, such as Visio, are designed specifically for creating flowcharts.

Teaching Tip	Use a simple example, such as basic payroll processing, to create a flowchart showing the input, checking the number of hours worked, multiplying hours worked times the pay rate, and producing the output of the pay amount. Then use the same example using pseudocode.
---------------------	--

Understanding Program Comments

1. Explain the use of program comments (nonexecuting statements) to create documentation. Note that double-forward slashes are used in several popular programming languages to precede comments. Show the example of the annotation symbol in Figure 1-4 on Page 18, used on flowcharts for comments.

Understanding Programming and User Environments

1. Discuss the different programming environments programmers can use during the planning step. Explain that flowcharts can be written by hand or by using software, and pseudocode can be typed into a text editor or in an integrated development environment (IDE).

Understanding Programming Environments

1. Describe some advantages of using an IDE.
 - Uses colors to display various language components
 - Highlights syntax errors visually
 - Automatic statement completion
 - Can step through program execution to find errors

One disadvantage is that storing a program in an IDE requires more storage space than a text editor.

Understanding User Environments

1. Explain that a user might execute the same program using a command line or a graphical user interface (GUI) but that the programming process is the same regardless of the user environment.

Quick Quiz 2

1. True or False: In object-oriented design, the programmer must first envision and create the objects that a program will manipulate.

Answer: True

2. As discussed in this lecture, the heart of the programming process is:
 - A. analyzing the system
 - B. designing the system
 - C. planning the logic
 - D. coding the program
 - E. translating the program code

Answer: C

3. Compilers or interpreters are also known as:
 - A. translucent programs
 - B. transistor programs
 - C. translator programs
 - D. transgression programs

Answer: C

4. Two tools that programmers use to plan the logic of a solution to a programming problem are ____ and ____.

Answer: flowcharts, pseudocode

5. True or False: Two forward slashes are used to precede comments in many programming languages.

Answer: True

6. A text editor and a(n) ____ are two examples of programming environments.

Answer: integrated development environment or IDE

7. A(n) ____ and ____ are two examples of user environments.

Answer: command line, GUI

Class Discussion Topics

1. Have students discuss the key features of the object-oriented design programming process.
2. Have students discuss how different input devices vary in the types of data that they can directly provide to the computer system. For example, numeric keypads provide only numeric input; a stylus provides tap-selection or handwriting input. Discuss the impact on programming with these devices as input or output media.
3. Have students discuss the differences between pseudocode and flowcharts, and why one might be more suited to a given project or programmer. For example, a flowchart can help visualize program flow in a large program with many decision points but may be unnecessary for a simple program. Also, some programmers naturally think visually, and others prefer more language-based code.
4. Discuss some of the costs involved in creating a large, complex application, and discuss why good initial design and reusable components are important. How do they contribute to the return on investment (ROI) for the business? How does good design help in reducing the amount of maintenance required and in simplifying that maintenance?

Additional Projects

1. Have students create a list of Web sites that specifically relate to good techniques in object-oriented programming. Compile these lists into a master list to be distributed to the class.
2. Have students research and create a list of six object-oriented programming languages available for use today. Students should create a table with the following information: name of programming language, most current version, supporting vendor or organization, cost, compiler or interpreter location (separate or included with language), and a brief product description. They should then write a paragraph describing the research experience and a paragraph indicating the language they would use from their list, with reasons to back up their choice.
3. Have students create a flowchart to represent an everyday task, such as washing a car. With an open-ended instruction such as this, they will realize the importance of the requirements gathering step. Remind them to use the appropriate flowchart symbols.

Additional Resources

1. Pseudocode standards:
www.csc.calpoly.edu/~jdalbey/SWE/pdl_std.html
2. Sun JAVA site:
<http://java.sun.com>
3. Sun JAVA Tutorial: Lesson: Object-Oriented Programming Concepts:
<http://java.sun.com/docs/books/tutorial/java/concepts>
4. Microsoft Visual Basic Developer Center:
<http://msdn2.microsoft.com/en-us/vbasic/default.aspx>
5. Microsoft Visual C# Developer Center:
<http://msdn2.microsoft.com/en-us/vcsharp/default.aspx>
6. A High-Level Introduction to C Programming:
<http://computer.howstuffworks.com/c.htm>
7. An Introduction to Object-Oriented Programming Using C++ – online tutorial:
<http://gd.tuwien.ac.at/languages/c/c++oop-pmueller/>

Key Terms

- An **algorithm** is the sequence of steps necessary to solve any problem.
- An **annotation symbol** is a flowchart symbol that is used to hold comments; it is represented by a three-sided box connected with a dashed line to the step it explains.
- **Application software** comprises all the programs you apply to a task.
- **Attributes** are the characteristics of entities.
- The **attributes of an object** are the features it “has.”
- The **behaviors of an object** are the things it “does.”
- **Binary language** consists of 1s and 0s; it is machine language.
- **Black box tests** are software tests in which the tester does not know how the software works internally but verifies that correct output is derived from various input values.
- The **central processing unit**, or **CPU**, is the hardware component that processes data.
- A **class** is a general category of objects.
- **Coding the program** is the act of writing program instructions.
- A **command line** is a location on your computer screen at which you type text entries to communicate with the computer’s operating system.
- A **compiler** translates a high-level language into machine language and tells you if you have used a programming language incorrectly. A compiler translates an entire program at once.
- **Computer memory** is a computer’s temporary, internal storage.
- **Computer simulations** attempt to mimic real-world activities so that their processes can be improved or so that users can better understand how the real-world processes operate.

- **Conventions** are standards of format and style that are selected for consistency, while acknowledging that other customs might be used by others and be equally as correct.
- **Conversion** is the entire set of actions an organization must take to switch over to using a new program or set of programs.
- **Data items** include all the text, numbers, and other information that are processed by a computer.
- **Data modeling** is the act of identifying all the objects you want to manipulate and how they relate to each other.
- A **decision symbol** in a flowchart holds a question that allows program logic to follow divergent paths and is represented by a diamond.
- **Desk-checking** is the process of walking through a program's logic on paper without using a computer.
- **Entity types** describe the broad categories of data items in a system.
- To **execute** a program is to carry out a program's instructions.
- **Executable statements** are the statements that carry out a program's actions.
- A **flowchart** is a pictorial representation of the logical steps it takes to solve a problem.
- **Flowlines** are the arrows in a flowchart that show the sequence of steps carried out.
- A **graphical user interface**, or **GUI** (pronounced "gooey"), allows users to interact with a program in a graphical environment.
- **Hardware** is the set of physical devices associated with a computer.
- **High-level** describes programming languages that resemble the English language.
- **Information** is data that has been processed and is ready for output.
- **Input** is the process of entering data into a system using hardware devices such as keyboards and mice.
- An **input symbol** in a flowchart contains an input statement and is represented by a parallelogram.
- An **input/output symbol**, or **I/O symbol**, is a parallelogram used to diagram both input and output operations.
- An **integrated development environment (IDE)** is a software package that provides an editor, compiler, and other programming tools.
- An **interpreter** translates a high-level language into machine language and tells you if you have used a programming language incorrectly. An interpreter translates a program one instruction at a time.
- The **logic** of a computer program is developed when you give instructions to the computer in a specific sequence, without leaving any instructions out or adding extraneous instructions.
- **Logical errors** occur when incorrect instructions are performed or when instructions are performed in the wrong order.
- **Low-level** describes languages that more closely reflect computer circuitry than high-level languages; the lowest-level language is the set of 1s and 0s that a computer understands.
- **Machine language** is a computer's on/off circuitry language.
- **Maintenance** is the act of making changes to programs that are already finished and in production.
- An **object** is an entity used in a program.
- **Object code** is machine language statements that have been translated from source code.
- **Object-oriented analysis**, or **OOA**, is the process of analyzing a system using an object-oriented approach.

- Taking an **object-oriented approach** to a problem means defining the objects needed to accomplish a task, and developing the objects so that each maintains its own data and carries out tasks when another object requests them.
- **Object-oriented design**, or **OOD**, is the process of designing a system using an object-oriented approach.
- **Object-oriented programming (OOP)** is a technique that focuses on objects, or “things.” OOP describes the objects’ features, or attributes, and their behaviors.
- **Output** is the process of extracting information from a system through hardware such as a monitor or printer so that people can view, interpret, and use the results.
- An **output symbol** in a flowchart contains an output statement and is represented by a parallelogram.
- **Procedural programming** is a technique that focuses on procedures that programmers create to manipulate data.
- **Processing** data items may involve organizing them, checking them for accuracy, or performing mathematical operations on them.
- A **processing symbol** in a flowchart contains a processing statement and is represented by a rectangle.
- **Program code** is written computer instructions.
- **Program comments** are nonexecuting statements that you add to a program for the purpose of documentation.
- **Programming** is the act of writing software instructions.
- **Programming languages**, such as Visual Basic, C#, C++, and Java, are used to write programs.
- **Programs** are sets of executable instructions written by programmers.
- **Pseudocode** is a representation of the logical steps it takes to solve a problem compiled in a manner that closely resembles the English language.
- **Random access memory (RAM)** is internal computer memory.
- **Relationships** describe how entities communicate with and react to each other.
- To **run** a program is to carry out a program’s instructions.
- **Scripting languages** (also called **scripting programming languages** or **script languages**) such as Python, Lua, Perl, and PHP are used to write programs that are typed directly from a keyboard. Scripting languages are stored as text rather than as binary executable files.
- **Semantic errors** are logical program errors.
- **Software** is the set of instructions written by programmers that tell the computer what to do; software is a set of computer programs.
- **Software testers** are professionals who test programs for correctness.
- **Source code** is the set of statements you write in a programming language before they are translated to object code.
- The **state of an object** is made up of its attributes’ values.
- **Storage devices** are hardware, such as disks or flash media, on which you can store data.
- The **syntax** of a language consists of its rules.
- A **syntax error** is an error in language or grammar.
- **System software** comprises the programs that you use to manage your computer, including operating systems such as Windows, UNIX, or other utility programs not directly used by end users.
- A **terminal symbol** in a flowchart marks the beginning or end of a flowchart segment, method, or program, and is represented by a lozenge.

- To **test a program** is to execute it on a computer to determine if the output is correct.
- A **text editor** is a program that you use to create simple text files; it is similar to a word processor, but without as many features.
- **Users** (or **end users**) are the people or entities for whom programs are written, and who will benefit from using them.
- **White box tests** are a type of software test in which the tester understands how the software works internally.