❷ *UNDERSTANDING SIMPLE PROGRAM LOGIC*

- A programmer's job involves writing instructions (such as those in the doubling program in the preceding section), but a professional programmer usually does not just sit down at a computer keyboard and start typing. The **program development cycle** can be broken down into at least seven steps:

  1. Understand the problem.
  2. Plan the logic.
  3. Code the program.
  4. Use a compiler or interpreter to translate the program into machine language.
  5. Test the program.
  6. Put the program into production.
  7. Maintain the program.

- ❶ ***Understanding the Problem***: Professional computer programmers write programs to satisfy the needs of others, called **users** or **end users**.
  - Examples of end users include a Human Resources department that needs a printed list of all employees, a Billing department that wants a list of clients who are 30 or more days overdue on their payments, and an Order department that needs a Web site to provide buyers with an online shopping cart.
  - Because programmers are providing a service to these users, programmers must first understand what the users want. When a program runs, you usually think of the logic as a cycle of input-processing-output operations, but when you plan a program, you think of the output first.
  - After you understand what the desired result is, you can plan the input and processing steps to achieve it.
    - Suppose the director of Human Resources says to a programmer, "*Our department needs a list of all employees who have been here over five years, because we want to invite them to a special thank-you dinner*."
    - On the surface, this seems like a simple request. An experienced programmer, however, will know that the request is incomplete. For example, you might not know the answers to the following questions about which employees to include:
      - Does the director want a list of full-time employees only, or a list of full- and part-time employees together?
      - Does she want to include people who have worked for the company on a month-tomonth contractual basis over the past five years, or only regular, permanent employees?
      - Do the listed employees need to have worked for the organization for five years as of today, as of the date of the dinner, or as of some other cutoff date?
      - What about an employee who worked three years, took a two-year leave of absence, and has been back for three years?

- The programmer cannot make any of these decisions; the user (in this case, the Human Resources director) must address these questions.
- More decisions still might be required. For example:
  - What data should be included for each listed employee?
  - Should the list contain both first and last names? Social Security numbers? Phone numbers? Addresses?
  - Should the list be in alphabetical order? Employee ID number order? Length-of-service order? Some other order?
  - Should the employees be grouped by any criteria, such as department number or years of service?

- Several pieces of documentation are often provided to help the programmer understand the problem. **Documentation** consists of all the supporting paperwork for a program; it might include items such as original requests for the program from users, sample output, and descriptions of the data items available for input.
- Understanding the problem might be even more difficult if you are writing an app that you hope to market for mobile devices. Business developers are usually approached by a user with a need, but successful developers of mobile apps often try to identify needs that users aren't even aware of yet. For example, no one knew they wanted to play Angry Birds or leave messages on Facebook before those applications were developed.
- Mobile app developers also must consider a wider variety of user skills than programmers who develop applications that are used internally in a corporation. Mobile app developers must make sure their programs work with a range of screen sizes and hardware specifications because software competition is intense and the hardware changes quickly.
- Fully understanding the problem may be one of the most difficult aspects of programming. On any job, the description of what the user needs may be vague—worse yet, users may not really know what they want, and users who think they know frequently change their minds after seeing sample output.

- ❷ ***Planning the Logic***: The heart of the programming process lies in planning the program's logic. During this phase of the process, the programmer plans the steps of the program, deciding what steps to include and how to order them.
  - You can plan the solution to a problem in many ways. The two most common planning tools are **flowcharts** and **pseudocode**. Both tools involve writing the steps of the program in English.
  - Programmers refer to planning a program as "developing an algorithm." An **algorithm** (see the end of the section for the meaning) is the sequence of steps or rules you follow to solve a problem.
  - In addition to flowcharts and pseudocode, programmers use a variety of other tools to help in program development.
    - One such tool is an **IPO chart**, which delineates input, processing, and output tasks.

- Some object-oriented programmers also use **TOE charts**, which list **t**asks, **o**bjects, and **e**vents.
- **Storyboards** and **UML** are frequently used in interactive, object-oriented applications.

- The programmer shouldn't worry about the syntax of any particular language during the planning stage, but should focus on figuring out what sequence of events will lead from the available input to the desired output. Planning the logic includes thinking carefully about all the possible data values a program might encounter and how you want the program to handle each scenario.

- ❸ ***Coding the Program***: After the logic is developed, only then can the programmer write the source code for a program. Hundreds of programming languages are available. Programmers choose particular languages because some have built-in capabilities that make them more efficient than others at handling certain types of operations. Despite their differences, programming languages are quite alike in their basic capabilities—each can handle input operations, arithmetic processing, output operations, and other standard functions. The logic developed to solve a programming problem can be executed using any number of languages. Only after choosing a language must the programmer be concerned with proper punctuation and the correct spelling of commands—in other words, using the correct syntax.

- ❹ ***Use a Compiler or Interpreter to Translate the Program into Machine Language***: Even though there are many programming languages, each computer knows only one language—its machine language, which consists of 1s and 0s.
  - Computers understand machine language because they are made up of thousands of tiny electrical switches, each of which can be set in either the on or off state, which is represented by a 1 or 0, respectively.
  - Languages like Java or Visual Basic are available for programmers because someone has written a translator program (a compiler or interpreter) that changes the programmer's English-like **high-level programming language** into the **low-level machine language** that the computer understands.
  - When you learn the syntax of a programming language, the commands work on any machine on which the language software has been installed. However, your commands then are translated to machine language, which differs in various computer makes and models.
  - If you write a programming statement incorrectly, the translator program doesn't know how to proceed and issues an error message identifying a syntax error. Although making errors is never desirable, syntax errors are not a programmer's deepest concern, because the compiler or interpreter catches every syntax error and displays a message that notifies you of the problem. The computer will not execute a program that contains even one syntax error.
  - Typically, a programmer develops logic, writes the code, and compiles the program, receiving a list of syntax errors. The programmer then corrects the syntax errors and compiles the program again.

- Correcting the first set of errors frequently reveals new errors that originally were not apparent to the compiler.

- ❺ ***Testing the Program***: A program that is free of syntax errors is not necessarily free of logical errors.
  - A logical error results when you use syntactically correct statements but the program doe not produce the intended results.
  - Once a program is free of syntax errors, the programmer can test it—that is, execute it with some sample data to see whether the results are logically correct.
  - The process of finding and correcting program errors is called **debugging**. You debug a program by testing it using many sets of data.
  - Selecting test data is somewhat of an art in itself, and it should be done carefully
- ❻ ***Putting the Program into Production***: Once the program is thoroughly tested and debugged, it is ready for the organization to use. Putting the program into production might mean simply running the program once, if it was written to satisfy a user's request for a special list. However, the process might take months if the program will be run on a regular basis, or if it is one of a large system of programs being developed. Perhaps data-entry people must be trained to prepare the input for the new program, users must be trained to understand the output, or existing data in the company must be changed to an entirely new format to accommodate this program. **Conversion**, the entire set of actions an organization must take to switch over to using a new program or set of programs, can sometimes take months or years to accomplish.
- ❼ ***Maintaining the Program***: After programs are put into production, making necessary changes is called **maintenance**.
  - Maintenance can be required for many reasons: for example, because new tax rates are legislated, the format of an input file is altered, or the end user requires additional information not included in the original output specifications.
  - Frequently, your first programming job will require maintaining previously written programs. When you maintain the programs others have written, you will appreciate the effort the original programmer put into writing clear code, using reasonable variable names, and documenting his or her work.
  - When you make changes to existing programs, you repeat the development cycle. That is, you must understand the changes, then plan, code, translate, and test them before putting them into production. If a substantial number of program changes are required, the original program might be retired, and the program development cycle might be started for a new program.