Chapter 1
An Overview of Computers and Programming
Objectives

In this chapter, you will learn about:

• Computer systems
• Simple program logic
• The steps involved in the program development cycle
• Pseudocode statements and flowchart symbols
• Using a sentinel value to end a program
• Programming and user environments
• The evolution of programming models
Understanding Computer Systems

• **Computer system**
  – Combination of all the components required to process and store data using a computer

• **Hardware**
  – Equipment associated with a computer

• **Software**
  – System & Application
  – Computer instructions that tell the hardware what to do
  – **Programs**
    • Software written in a language to perform a particular task
Understanding Computer Systems (continued)

• **Programming**
  – writing complete programs
  – writing portions of a program (modules)

• Computer hardware and software accomplish three major operations: [Information Processing Cycle]
  
  – **Input**
    • Data items enter computer
  
  – **Process**
    • By central processing unit (CPU)
  
  – **Output**
  
  – **Store**
• **Programming language**
  - Use to write computer instructions
  - Examples:
    - Visual Basic, C#, C++, Java, Ada, Python, Ruby

• **Syntax**
  - Rules governing the construction of valid statements in a language
  - Conventions

• **Computer memory**
  - Computer’s temporary, internal storage
  - Volatile
Understanding Computer Systems (continued)

• Permanent storage devices
  – Non-volatile storage

• Translator
  – Compiler and/or an **interpreter**
  – Translates program code into **machine language** (**binary language**)
  – Checks for syntax errors
  – Many modern languages use both a compiler and an interpreter

• Program **executes** or **runs**
  – Input will be accepted, some processing will occur, and results will be output
Understanding Simple Program Logic

• Program with **syntax errors** cannot execute
• Program with **logic errors** can execute, but...
  – Errors in program logic produce incorrect output as a result

• **Logic** of the computer program
  – Sequence of specific instructions in specific order

• **Variable**  [fundamental concept in program design]
  – Named memory location whose value can vary

• **Syntax & Semantics**
Understanding the Program Development Cycle

- **Program development cycle**
  - Understand the problem
  - Plan the logic
  - Code the program
  - Translate the program into machine language using software (a compiler and/or interpreter)
  - Test the program
  - Deploy the program (make available for use)
  - Maintain the program

- Detailed information follows...
Understanding the **Program Development Cycle** (continued)

![Diagram of the program development cycle](image)

**Figure 1-1** The program development cycle
Understanding the Problem

• One of the most difficult aspects of programming

• Users (end users)
  – People for whom program is written

• Documentation
  – Supporting paperwork for a program
    • flowchart / pseudocode
    • hierarchy chart (aka structure chart or VTOC)
    • screen / printer spacing chart
    • end user instructions
Plan the Logic

• Heart of the programming process

• Most common logic planning tools
  – Flowcharts
  – Pseudocode
  – hierarchy chart

• Desk-checking
  – Walking through a program’s logic on paper before you actually write the program
Code the Program

• Hundreds of programming languages are available
  – Choose based on:
    • features
    • organizational requirements

• Most languages are similar in their basic capabilities

• Easier than planning step (not necessarily so for new programming students...)

Programming Logic & Design, Seventh Edition
Using Software to **Translate** the Program into Machine Language

- **Translator program**
  - Compiler and/or interpreter
  - Changes the programmer’s English-like **high-level** programming language into the **low-level** machine language

- **Syntax error**
  - Misuse of a language’s grammar rules
  - Programmer corrects listed syntax errors
  - Might need to recompile the code several times
    - misspelled variable names
    - unmatched curly braces
Languages / File Types

- **Source language**
  - Java, C++, Visual Basic, etc.
  - file types (extensions):
    - java
    - cpp
    - vb

- **Compiled language (destination language)**
  - other high-level language (cross compiler)
  - machine language
  - virtual machine language (intermediate language)
    - Java class file (.class)
    - MSIL (Microsoft Intermediate Language)
  - files types (extensions):
    - class
    - msil
    - obj
    - exe
Using Software to Translate the Program into Machine Language (continued)

Figure 1-2 Creating an executable program
Test the Program

• **Logical error**
  – Use a syntactically correct statement but use the wrong one for the current context

• **Run-time error**
  – Program ends abnormally when the user runs the program (sometimes or every time)

• **Test Data**
  – Execute the program with some sample test data to see whether the results are logically correct
Deploy the Program
Make the Program Available for Use

• Process depends on program’s purpose
  – May take several months

• Conversion
  – Entire set of actions an organization must take to switch over to using a new program or set of programs
Maintain the Program

• **Maintenance**
  – Making changes after program is put into production

• **Common first programming job**
  – Maintaining previously written programs

• **Make changes to existing programs**
  – Repeat the development cycle
Using Pseudocode Statements and Flowchart Symbols

• **Pseudocode**
  – English-like representation of the logical steps it takes to solve a problem

• **Flowchart**
  – Pictorial representation of the logical steps it takes to solve a problem
Writing Pseudocode

- Pseudocode representation of a number-doubling problem

```plaintext
start
  input myNumber
  set myAnswer = myNumber * 2
  output myAnswer
stop
```
Writing Pseudocode (continued)

• Programmers preface their pseudocode with a beginning statement like \textit{start} and end it with a terminating statement like \textit{stop}

• \textbf{Flexible} because it is a planning tool

• English-like

• Doesn’t require any software/hardware
Drawing a Flowchart

• Create a flowchart
  – Draw geometric shapes that contain an individual action
  – Connect shapes with arrows

• Input symbol
  – Indicates input operation
  – Parallelogram

• Processing symbol
  – Processing statements such as arithmetic
  – Rectangle

• Connector symbol
  – Used to connect flowlines
  – small circle
• **Output symbol**
  - Represents output statements
  - Parallelogram

• **Flowlines**
  - Lines and Arrows that connect steps

• **Terminal symbols**
  - Start/stop symbols
  - Shaped like a racetrack
  - Also called **lozenge** or **capsule**
Figure 1-6  Flowchart and pseudocode of program that doubles a number
Repeating Instructions

• **Loop**
  – Repeats a series of steps
  – referred to as *looping, repetition, and iteration* (synonyms)

• **Infinite loop**
  – Repeating flow of logic with no end (repeat forever)
Repeating Instructions (continued)

**Figure 1-8** Flowchart of infinite number-doubling program
Using a **Sentinel Value** to End a Program

- **Making a decision**
  - Testing a value
  - **Decision symbol**
    - Diamond shape

- **Dummy value**
  - Data-entry value that the user will never need
  - **Sentinel value**

- **eof** (“end of file”)
  - Marker at the end of a file that automatically acts as a sentinel
Using a Sentinel Value to End a Program (continued)

Figure 1-9 Flowchart of number-doubling program with sentinel value of 0
Using a Sentinel Value to End a Program (continued)

Figure

Don’t Do It
This logic is not structured; you will learn about structure in Chapter 3.

- **start**
- **input myNumber**
- **eof?**
  - **Yes** → **stop**
  - **No**
    - **set myAnswer = myNumber times 2**
    - **output myAnswer**
Understanding Programming and User Environments

• Many options for programming and user environments:
  – simple text editor such as Notepad
  – “Smart Editor” such as Brief or ConTEXT
  – IDE (Integrated Development Environment) such as jGRASP or Visual Studio or Eclipse
Understanding Programming Environments

• Use a keyboard to type program statements into an editor
  – Plain **text editor**
    • Similar to a word processor but without as many features
  – Text editor that is part of an **integrated development environment (IDE)**
    • Software package that provides an editor, compiler, and other programming tools
Figure 1-12 A C# number-doubling program in Visual Studio
Understanding User Environments

• **Command line**
  – Location on your computer screen at which you type text entries to communicate with the computer’s operating system

• **Graphical user interface (GUI)**
  – Allows users to interact with a program in a graphical environment
Understanding User Environments (continued)

**Figure 1-13** Executing a number-doubling program in a command-line environment
Understanding User Environments (continued)

Figure 1-14 Executing a number-doubling program in a GUI environment
Understanding the Evolution of Programming Models

• People have been writing computer programs since the 1940s

• Newer programming languages
  – Look much more like natural language
  – Easier to use
  – Create self-contained modules or program segments that can be pieced together in a variety of ways
Major models or paradigms used by programmers

- **Procedural programming**
  - Focuses on the procedures that programmers create

- **Object-oriented programming**
  - Focuses on objects, or “things,” and describes their features (or attributes) and their behaviors

- Major difference
  - Focus the programmer takes during the earliest planning stages of a project
Summary

• Computer programming
  – Requires specific syntax
  – Must develop correct logic

• Programmer’s job
  – Understanding the problem, planning the logic, coding the program, translating the program into machine language, testing the program, putting the program into production, and maintaining it

• Procedural and object-oriented programmers approach problems differently