

Just Enough Programming Logic & Design



Chapter 1

An Overview of Computers and Logic



Objectives

After completing this chapter, you will be able to:

- Explain computer components and operations
- Discuss the steps involved in the programming process
- Use pseudocode statements and flowchart symbols
- Use and name variables and constants
- Explain data types and declare variables
- End a program by using sentinel values
- Discuss the evolution of programming techniques

Understanding Computer Components and Operations

- Components of a computer system
 - **Hardware**
 - Equipment or devices associated with a computer
 - **Software**
 - Computer instructions
 - Tells hardware what to do
 - **Programs**: instruction sets written by programmers
 - **Programming**: writing computer instructions

Understanding Computer Components and Operations (cont'd)

- **Application software**
 - Programs applied to a task
 - Includes word processing programs, spreadsheets, payroll and inventory programs, and even games
- **System software**
 - Programs used to manage your computer
 - Includes operating systems such as Windows, Linux or UNIX

Understanding Computer Components and Operations (cont'd)

- **Input**
 - Hardware devices, such as keyboards and mice, perform input operations
 - **Data**, or facts, enter the computer system through input devices
- **Processing**
 - Processing data items may involve organizing them, checking them for accuracy, or performing mathematical operations on them

Understanding Computer Components and Operations (cont'd)

- **Processing** continued
 - The hardware that performs these tasks is the **central processing unit**, or **CPU**
- **Output**
 - Sending information resulting from processing to a printer or monitor so people can view, interpret, and use the results
 - Can be stored on disks or flash media for later retrieval

Understanding Computer Components and Operations (cont'd)

- Computer instructions are written in a computer **programming language**
- Computer instructions you write are **program code**, also called **source code**
- **Coding a program** is when you write a program
- **Syntax** are the rules governing the word usage and punctuation of a programming language

Understanding Computer Components and Operations (cont'd)

- Each programming language uses a piece of software to translate the specific programming language statements into **object code**
- Object code is the computers on/off circuitry language, or **machine language**
- Machine language is represented as a series of 0s and 1s, also called **binary form**
- The language translation software is called a **compiler** or **interpreter**

Understanding Computer Components and Operations (cont'd)

- When a program's instructions are carried out, the program **runs** or **executes**
- A program that is free of syntax errors can be executed, but it might not produce the correct results
- For a program to work, the programmer must give the instructions to the computer in a specific sequence called the **logic** of the computer program

Understanding Computer Components and Operations (cont'd)

- **Logic errors** include instructions that are out of sequence, missing instructions, and instructions not part of the procedure
- Using an otherwise correct statement that does not make sense in the current context is called a **semantic error**
- **Logical errors** are much more difficult to locate than syntax errors

Understanding Computer Components and Operations (cont'd)

- To use a computer program, it must first be loaded into the computer's memory
- **Memory** is the internal storage in a computer
 - Also called **main memory** or **random access memory (RAM)**
 - Internal memory is temporary and **volatile** – its contents are lost when the computer loses power
 - Programmers often refer to memory locations, or addresses using hexadecimal notation



Understanding the Programming Process

- The programmer's job can be broken down into seven development steps
 - Understanding the problem
 - Planning the logic
 - Coding the program
 - Using software (a compiler or interpreter) to translate the program into machine language
 - Testing the program
 - Putting the program into production
 - Maintaining the program

Understanding the Programming Process (cont'd)

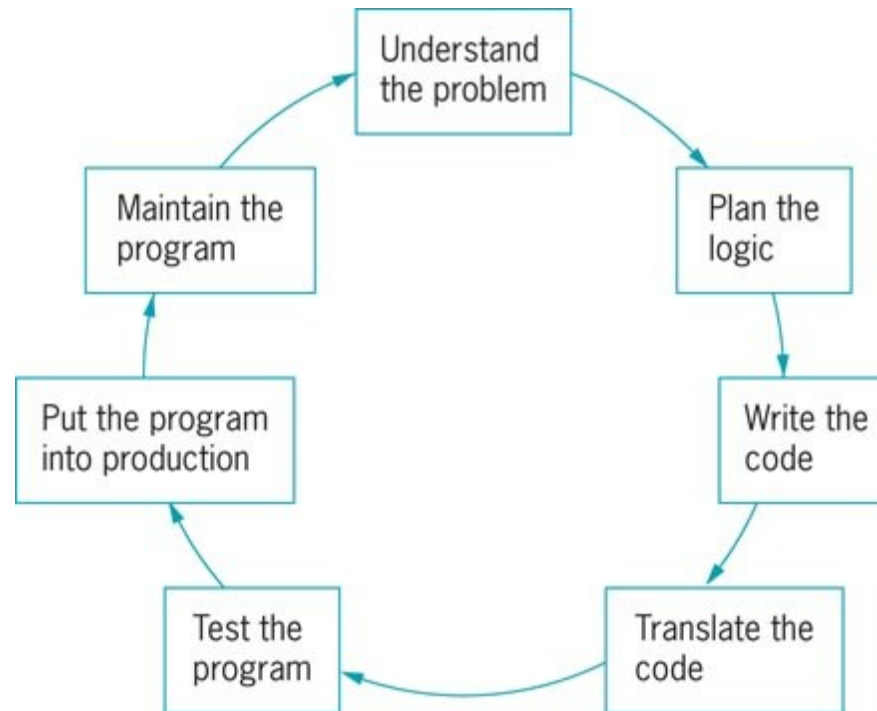


Figure 1.1 The program development cycle



Understanding the Problem

- Professional computer programmers write programs to satisfy the needs of others
- Really 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, or the users may not even really know what they want
 - Users frequently change their minds after seeing sample output
- A good programmer is often part counselor, part detective



Planning the Logic

- The heart of the programming process lies in planning the program's logic
 - During this phase of the programming process, the programmer plans the steps of the program, deciding what steps to include and how to order them
 - An **algorithm** is the sequence of steps needed to solve any problem
- The two most common planning tools are flowcharts and pseudocode



Planning the Logic (cont'd)

- Programmer should define the sequence of events that will lead to the desired output
- Planning a program's logic includes:
 - Thinking carefully about all the possible data values a program might encounter
 - How you want the program to handle each scenario
- The process of walking through a program's logic on paper before you actually write the program is called **desk-checking**



Coding the Program

- Only when the programmer has developed the logic of a program, can he or she code the program
 - In one of more than 400 programming languages
- Programmers choose a particular language because some languages have built-in capabilities that make them more efficient than others at handling certain types of operations

Using Software to Translate the Program into Machine Language

- Even though there are many programming languages, each computer is built knowing only one language – its machine language
 - Machine language consists of many 1s and 0s
- A translator program (a compiler or interpreter) changes the English-like **high-level programming language** in which the programmer writes into the **low-level machine language** that the computer understands

Using Software to Translate the Program into Machine Language (cont'd)

- A computer program must be free of syntax errors before you can execute it
- Typically, a programmer develops a program's logic, writes the code, and then attempts to compile or interpret the program using language-interpreting software



Testing the Program

- A program that is free of syntax errors is not necessarily free of logical errors
- Selecting test data is something of an art in itself, and it should be done carefully
- Many companies do not know that their software has a problem until an unusual circumstance occurs



Putting the Program into Production

- Once the program is tested adequately, it is ready for the organization to use
- Putting the program into production might mean simply running the program once
- The process might take months if the program will be run on a regular basis
- **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



Maintaining the Program

- After programs are put into production, changes may be required
- **Maintenance** is the process of updating programs
- When maintaining programs others have written, programmers appreciate the effort the original programmer put into writing clear code, using reasonable identifiers for values, and documenting his or her work



Maintaining the Program (cont'd)

- Changes to existing programs repeat the development cycle
 - Understand the changes
 - Plan
 - Code
 - Translate
 - Test
 - Put into production
 - Maintenance

Using Pseudocode Statements and Flowchart Symbols

- **Pseudocode** is an English-like representation of the logical steps it takes to solve a problem
- Example: Pseudocode for program doubling a number

```
start
  input originalNumber
  compute calculatedAnswer as originalNumber times 2
  output calculatedAnswer
stop
```

- A **flowchart** is a pictorial representation of the same thing

Using Pseudocode Statements and Flowchart Symbols (cont'd)

- When creating a **flowchart**, draw geometric shapes around individual statements and connect them with arrows
- **Input symbol**
 - Parallelogram
 - Indicates input operation

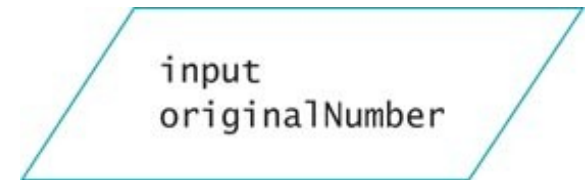
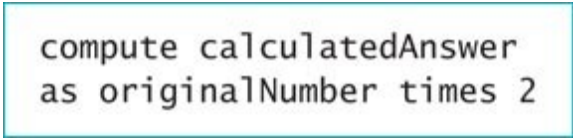


Figure 1.2 Input symbol

Using Pseudocode Statements and Flowchart Symbols (cont'd)

- **Processing symbol**
 - Rectangle
 - Processing statement



```
compute calculatedAnswer  
as originalNumber times 2
```

Figure 1.3 Processing symbol

- **Output symbol**
 - Parallelogram
 - Often called **I/O symbol** or **input/output symbol**
 - Indicates output operation



```
output  
calculatedAnswer
```

Figure 1.4 Output symbol

Using Pseudocode Statements and Flowchart Symbols (cont'd)

- To show the correct sequence of these statements, use arrows, or **flowlines**, to connect the steps
- To be complete, a flowchart should include two more elements:
 - **Terminal symbols**, or start/stop symbols, at each end

Using Pseudocode Statements and Flowchart Symbols (cont'd)

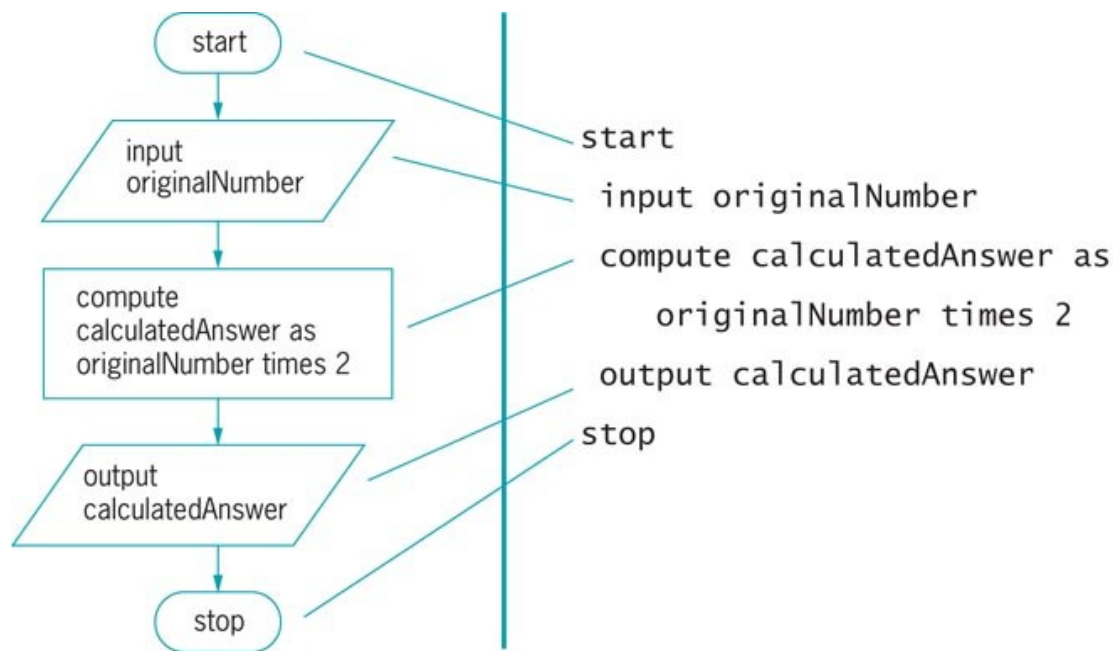


Figure 1.5 Flowchart and pseudocode of a program that doubles a number

The Advantages of Repetition

- Writing a number-doubling computer program is worth the effort only if a user has many numbers to double in a limited amount of time

```
start
  input originalNumber
  compute calculatedAnswer as originalNumber times 2
  output calculatedAnswer
  input originalNumber
  compute calculatedAnswer as originalNumber times 2
  output calculatedAnswer
  input originalNumber
  compute calculatedAnswer as originalNumber times 2
  output calculatedAnswer
  ...and so on for 9,997 more times
```

Don't Do It

You would never want to write such a repetitious list of instructions.

Figure 1.6 Inefficient pseudocode for program that doubles 10,000 numbers



Using and Naming Variables and Constants

- **Variables** are named memory locations whose contents can vary over time
- A variable name is also called an **identifier**
- Different languages put different limits on the length of identifiers, although in general, newer languages allow longer names
- Every language has its own rules for naming variables

Using and Naming Variables and Constants (cont'd)

- Variable name formats:
 - **Camel casing** has a “hump” in the middle – hourlyWage
 - **Pascal casing** is where the first letter of each name is upper case – HourlyWage
 - Names must be one word
 - Names should have some appropriate meaning
- When designing logic, do not be concerned with the syntax of any particular computer language



Assigning Values to Variables

- Assigning values to variables:
 - **Assignment statements**
 - `calculatedAnswer = originalNumber * 2`
 - The **assignment operator** is the equal sign
 - A **named constant** can be assigned a value only once
 - A **magic number** is an unnamed constant, like 0.08, whose meaning is not immediately apparent



Performing Arithmetic Operations

- Most programming languages use the following standard arithmetic operators:
 - + (plus sign)—addition
 - – (minus sign)—subtraction
 - * (asterisk)—multiplication
 - / (slash)—division
- Every operator follows **rules of precedence** that dictate the order in which operations in the same statement are carried out

Understanding Data Types and Declaring Variables

- Computers deal with two basic types of data: numeric and text (or string)
- A specific numeric value is often called a **numeric constant** (or a **literal numeric constant** or an **unnamed numeric constant**) because it does not change
- A specific text value, or string of characters, enclosed in quotation marks is a **string constant**

Understanding Data Types and Declaring Variables (cont'd)

- A variable's **data type** describes the kind of values the variable can hold, how much memory the value occupies, and the types of operations that can be performed with the data stored there
- A **numeric variable** is one that can have mathematical operations performed on it
 - Can hold digits, and usually can hold a decimal point and a sign indicating positive or negative

Understanding Data Types and Declaring Variables (cont'd)

- A **string variable** is a separate type of variable that can hold letters of the alphabet and other special characters such as punctuation marks
- To **declare the variable** is to tell the computer what type of variable to expect
- Variables must always be declared before the first time they are used in a program

Understanding Data Types and Declaring Variables (cont'd)

- After a variable is declared, you can assign a value to it, send it to output, or perform any operations that are allowed for its data type
- Variables are **initialized** when you assign initial values to them
 - Example:

```
string heading = "Employee Report"  
num countOfEmployees = 0
```

Understanding Data Types and Declaring Variables (cont'd)

- In some languages uninitialized variables are assigned a default value
- More commonly, uninitialized variables have an unknown, or **garbage value**, until a valid assignment is made



Ending a Program by Using Sentinel Values

- **Infinite loop:** repeating flow of logic with no end
- **Sentinel value:** predetermined value that means “Stop the program!”
- **Dummy value:** preselected value that stops the program
- Testing a value is also called **making a decision**
- **Decision symbol:** a diamond shape in a flowchart that represents a decision

Ending a Program by Using Sentinel Values (cont'd)

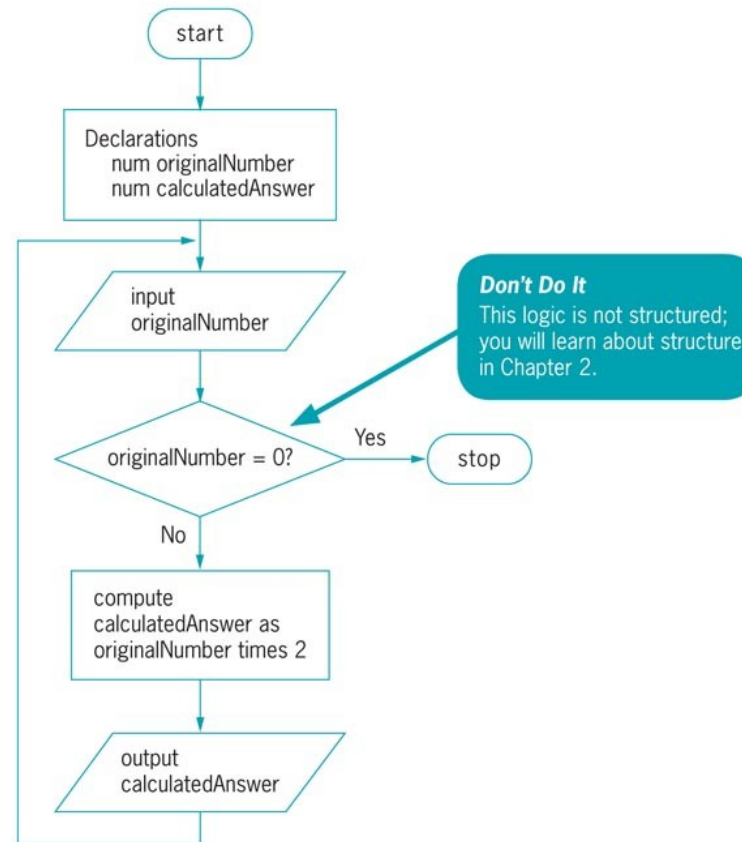



Figure 1.8 Flowchart of number-doubling program with sentinel value of 0



Ending a Program by Using Sentinel Values (cont'd)

- Many programming languages use the term **eof** (for “end of file”) to refer to a marker that automatically acts as a sentinel

Understanding the Evolution of Programming Techniques

- People have been writing modern computer programs since the 1940s
- The oldest programming languages required programmers to work with memory addresses and to memorize awkward codes associated with machine languages
- Newer programming languages look much more like natural language and are easier for programmers to use

Understanding the Evolution of Programming Techniques (cont'd)

- Modern programs use **modularity** - the ability to build programs from smaller segments
- Techniques used to develop programs:
 - **Procedural programming** focuses on the procedures that programmers create by breaking down processes into manageable tasks
 - **Object-oriented programming** focuses on objects and describes their attributes and behaviors



Summary

- Hardware and software are the two major components of any computer
- The programmer's job can be broken down into seven development steps
- Pseudocode and flowcharts are used to plan the logic for a solution
- Variables are named memory locations, whose contents can vary
- Two major techniques used to develop programs:
 - Procedural programming
 - Object-oriented programming