Fundamentals of Computer Programming

1. An Overview of Computers and Programming

Objectives

- Computer systems
- Simple program logic
- The program development cycle
- Pseudocode statements and flowchart symbols
- Programming and user environments
- The evolution of programming models

1.1 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**
  - Computer instructions
  - Tells the hardware what to do
  - **Programs**
    - Instructions written by programmers

Understanding Computer Systems

  - Application software such as word processing, spreadsheets, payroll and inventory, even games
  - System software such as operating systems like Windows, Linux, or UNIX
- Computer hardware and software accomplish three major operations
  - **Input**
    - **Data items** such as text, numbers, images, and sound
  - **Processing**
    - Calculations and comparisons performed by the **central processing unit** (**CPU**)

Understanding Computer Systems

  - **Output**
    - Resulting information that is sent to a printer, a monitor, or storage devices after processing
- **Programming language**
  - Used to write computer instructions
  - Examples
    - Visual Basic, C#, C++, or Java
- **Syntax**
  - Rules governing word usage and punctuation

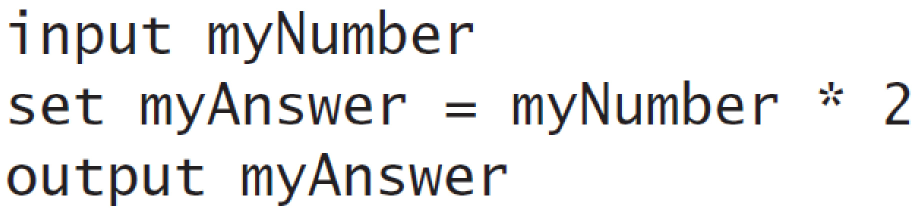
Understanding Computer Systems

- **Computer memory**
  - Computer's temporary, internal storage – **random access memory** (**RAM**)
  - **Volatile** memory – lost when the power is off
- Permanent storage devices
  - **Nonvolatile** memory
- **Compiler** or **interpreter**
  - Translates source code into **machine language** (**binary language**) statements called **object code**
  - Checks for syntax errors

1.2 Understanding Simple Program Logic

- Program **executes** or **runs**
  - Input will be accepted, some processing will occur, and results will be output
- Programs with syntax errors cannot execute
- **Logical errors**
  - Errors in program logic produce incorrect output
- **Logic** of the computer program
  - Sequence of specific instructions in specific order
- **Variable**
  - Named memory location whose value can vary

Understanding Simple Program Logic

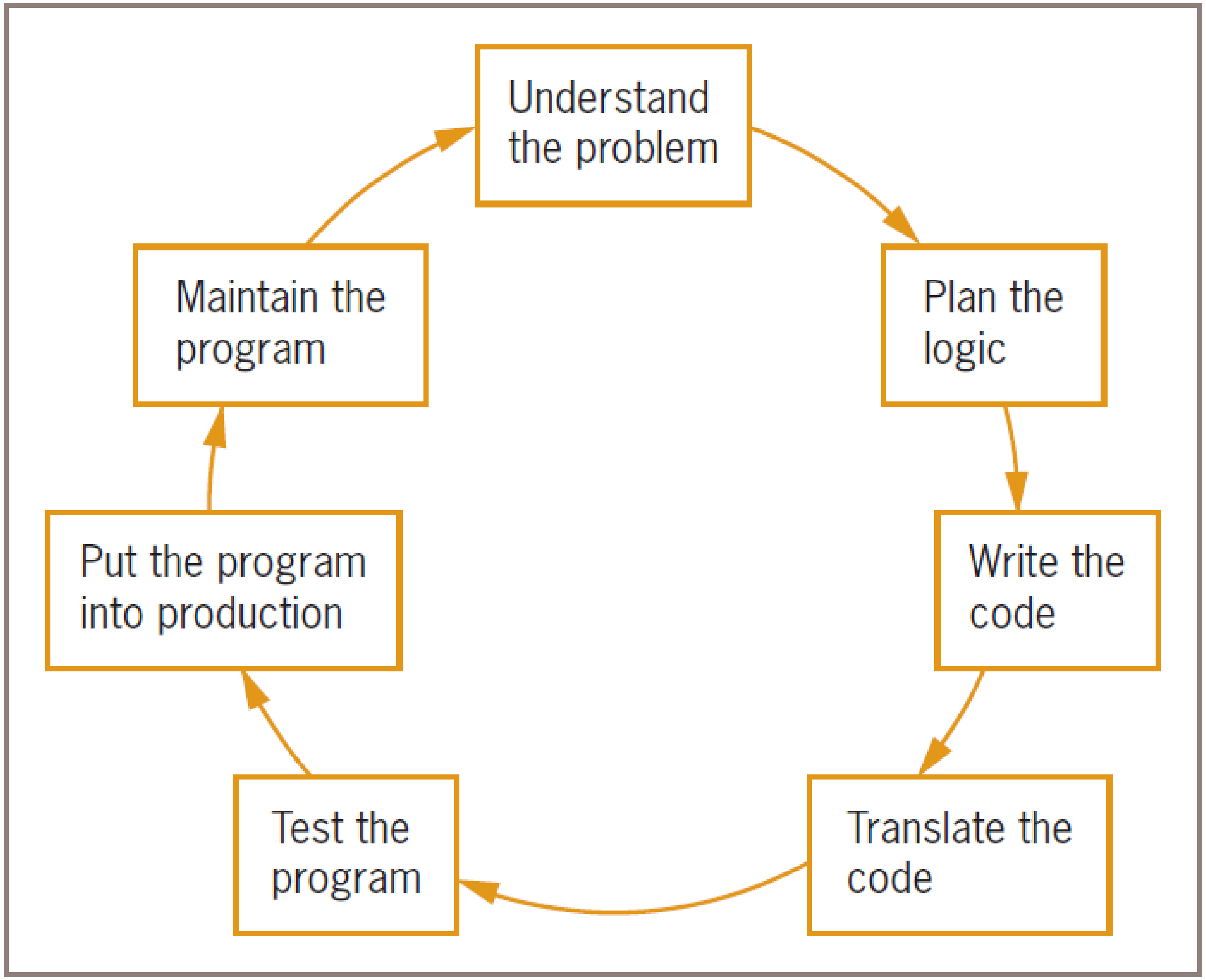


```
input myNumber
set myAnswer = myNumber * 2
output myAnswer
```

1.3 The Program Development Cycle

- Understand the problem
- Plan the logic
- Code the program
- Use software (a compiler or interpreter) to translate the program into machine language
- Test the program
- Put the program into production
- Maintain the program

The Program Development Cycle



Understand the problem

Plan the logic

Write the code

Translate the code

Test the program

Put the program into production

Maintain the program

Understanding the Problem

- One of the most difficult aspects of programming
- **Users** or **end users**
  - People for whom a program is written
- **Documentation**
  - Supporting paperwork for a program

Planning the Logic

- Developing an **algorithm**
- Heart of the programming process
- Most common planning tools
  - Flowcharts
  - Pseudocode
- **Desk-checking**
  - Walking through a program’s logic on paper before you actually write the program

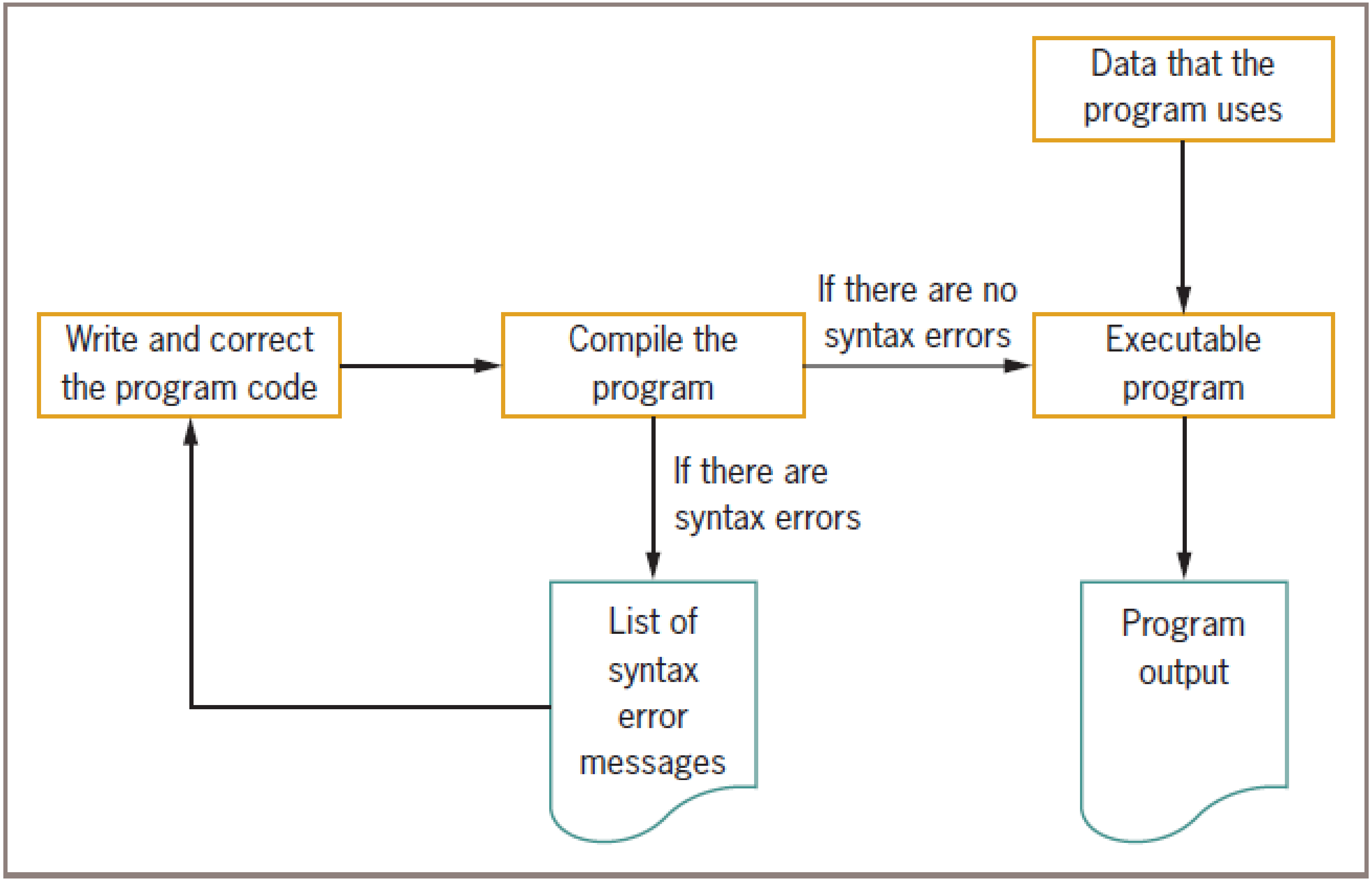
Coding the Program

- Hundreds of programming languages available
  - Choose based on features
  - Similar in their basic capabilities
- Easier than the planning step

Using Software to Translate the Program into Machine Language

- Translator program
  - Compiler 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

Creating an Executable Program



Data that the program uses

Write and correct the program code

Compile the program

If there are no syntax errors

Executable program

If there are syntax errors

List of syntax error messages

Program output

Testing the Program

- Logical error
  - Results when a syntactically correct statement, but the wrong one for the current context, is used
- Test
  - Execute the program with some sample data to see whether the results are logically correct
- **Debugging** is the process of finding and correcting program errors
- Programs should be tested with many sets of data

Putting the Program into Production

- Process depends on program’s purpose
  - May take several months
- **Conversion**
  - The entire set of actions an organization must take to switch over to using a new program or set of programs

Maintaining the Program

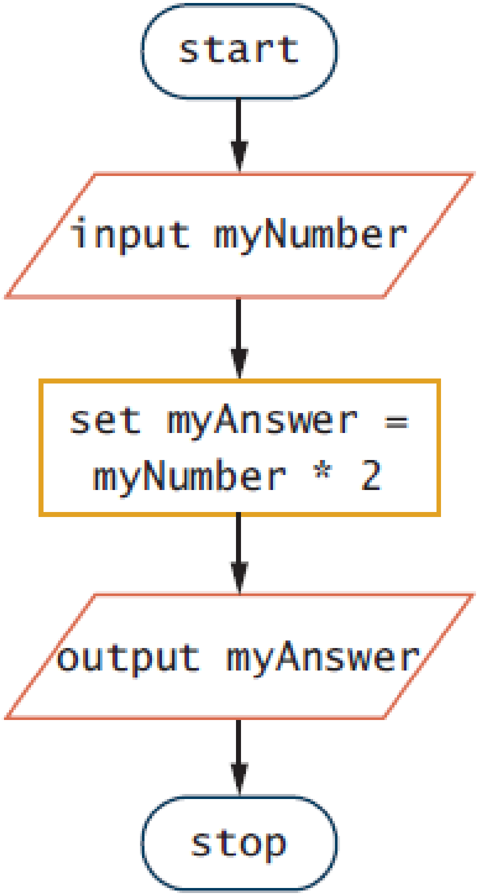
- **Maintenance**
  - Making changes after the program is put into production
- Common first programming job
  - Maintaining previously written programs
- Make changes to existing programs
  - Repeat the development cycle

1.4 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

Flowchart and Pseudocode of Program that Doubles a Number

Flowchart



Pseudocode

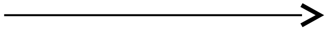
```
start
   input myNumber
   set myAnswer = myNumber * 2
   output myAnswer
stop
```

Writing Pseudocode

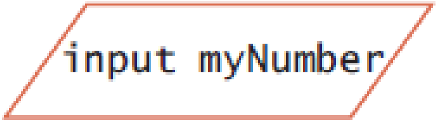
- Programmers preface their pseudocode with a beginning statement like **`start`** and end it with a terminating statement like **`stop`**
- Flexible planning tool

Drawing Flowcharts

- Create a flowchart
  - Draw geometric shapes that contain the individual statements
  - Connect shapes with arrows

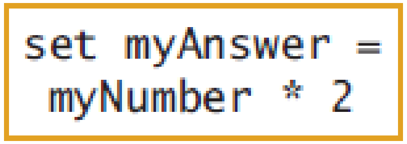


- **Input symbol**
  - Indicates input operation
  - Parallelogram



```
input myNumber
```

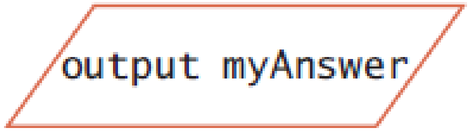
- **Processing symbol**
  - Contains processing statements such as arithmetic
  - Rectangle



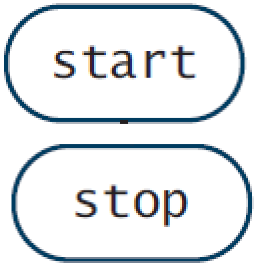
```
set myAnswer =
 myNumber * 2
```

Drawing Flowcharts

- **Output symbol**
  - Represents output statements
  - Parallelogram



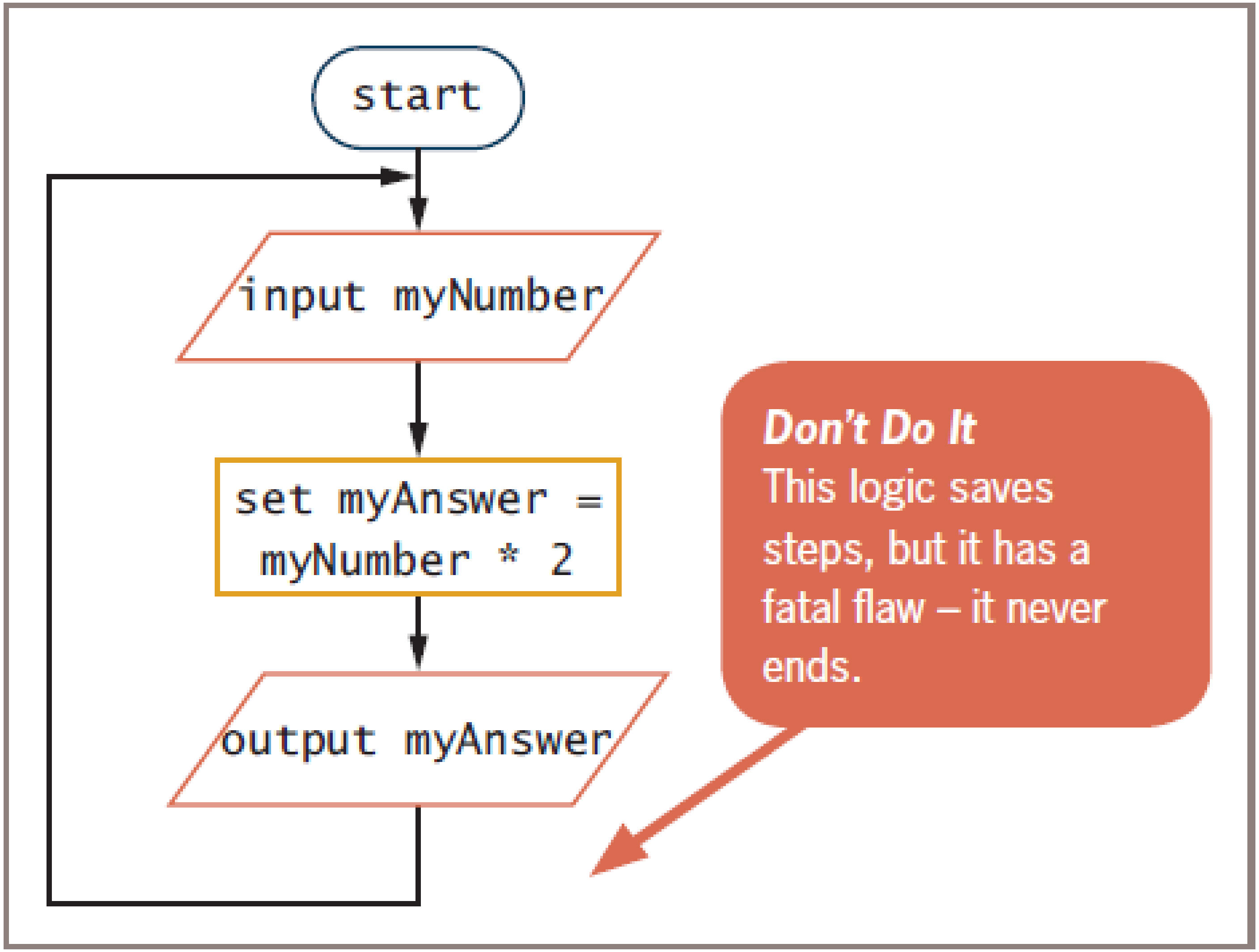
- **Flowlines**
  - Arrows that connect steps
- **Terminal symbols**
  - Start/stop symbols
  - Shaped like a racetrack
  - Also called lozenges



Repeating Instructions

- Program above only works for one number
- Not feasible to run the program over and over 10,000 times
- Not feasible to add 10,000 lines of code to a program
- Create a **loop** (repetition of a series of steps) instead
- Avoid an **infinite loop** (repeating flow of logic that never ends)

Flowchart of iInfinite Number-Doubling Program

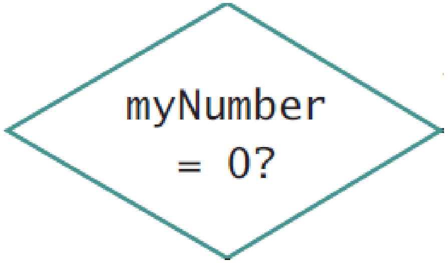


Don't Do It

This logic saves steps, but it has a fatal flaw – it never ends.

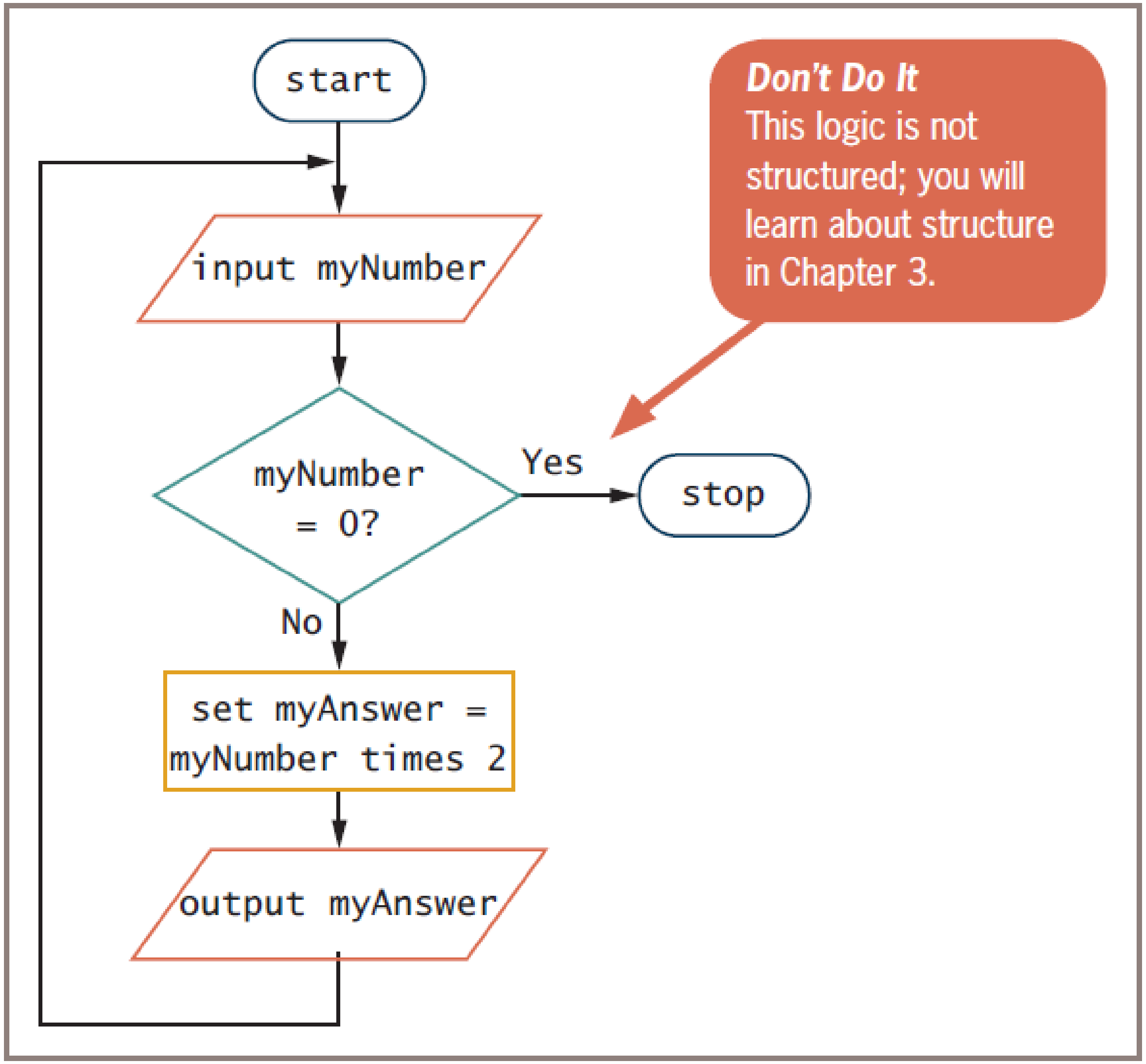
Using a Sentinel Value to End a Program

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



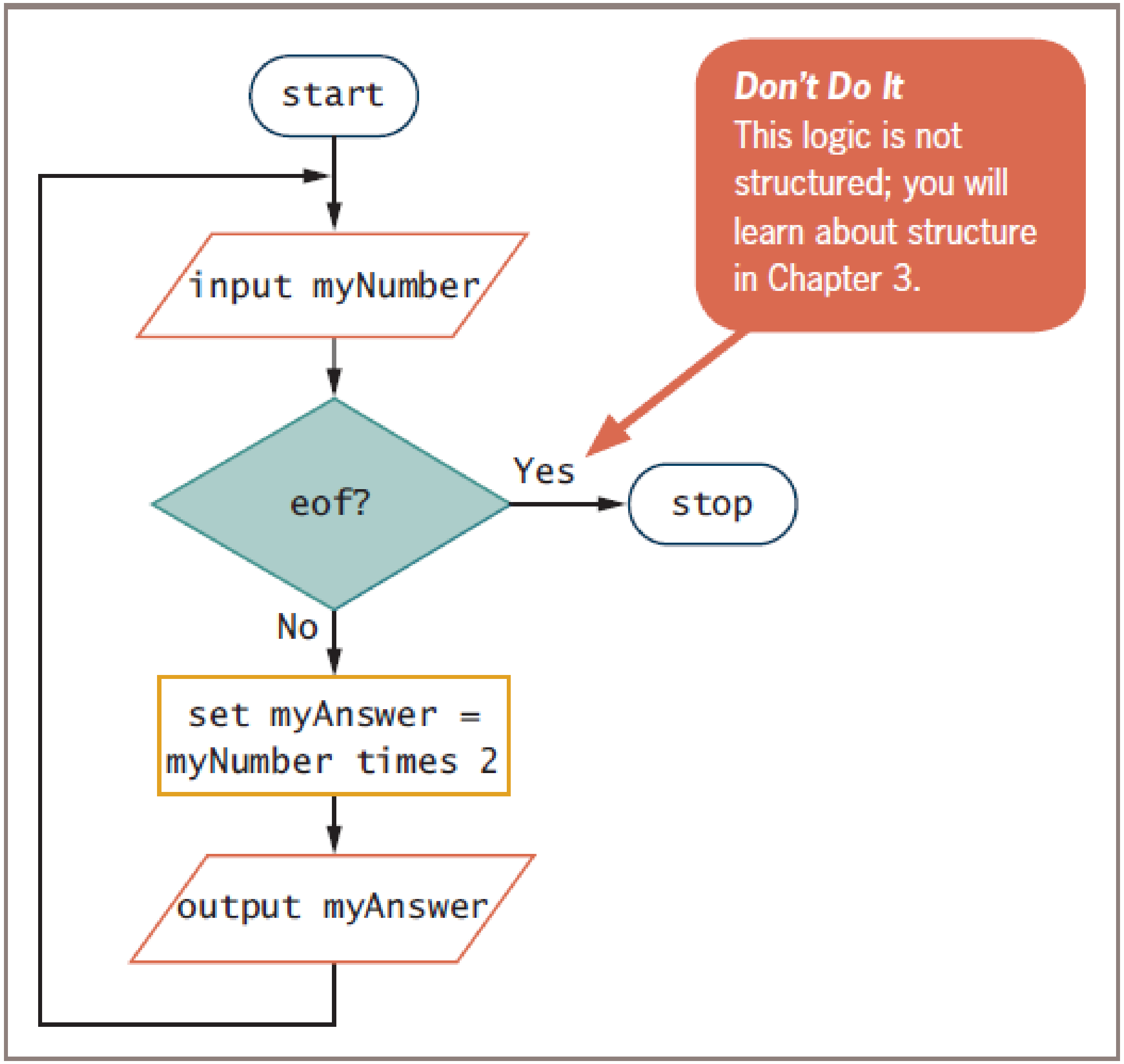
- **Dummy value**
  - Data-entry value that the user will never need
  - **Sentinel value**
- **eof** ("end of file")
  - Marker at the end of a file that automatically acts as a sentinel

Flowchart of Number-Doubling Program with Sentinel Value of 0



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

Flowchart Using eof



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

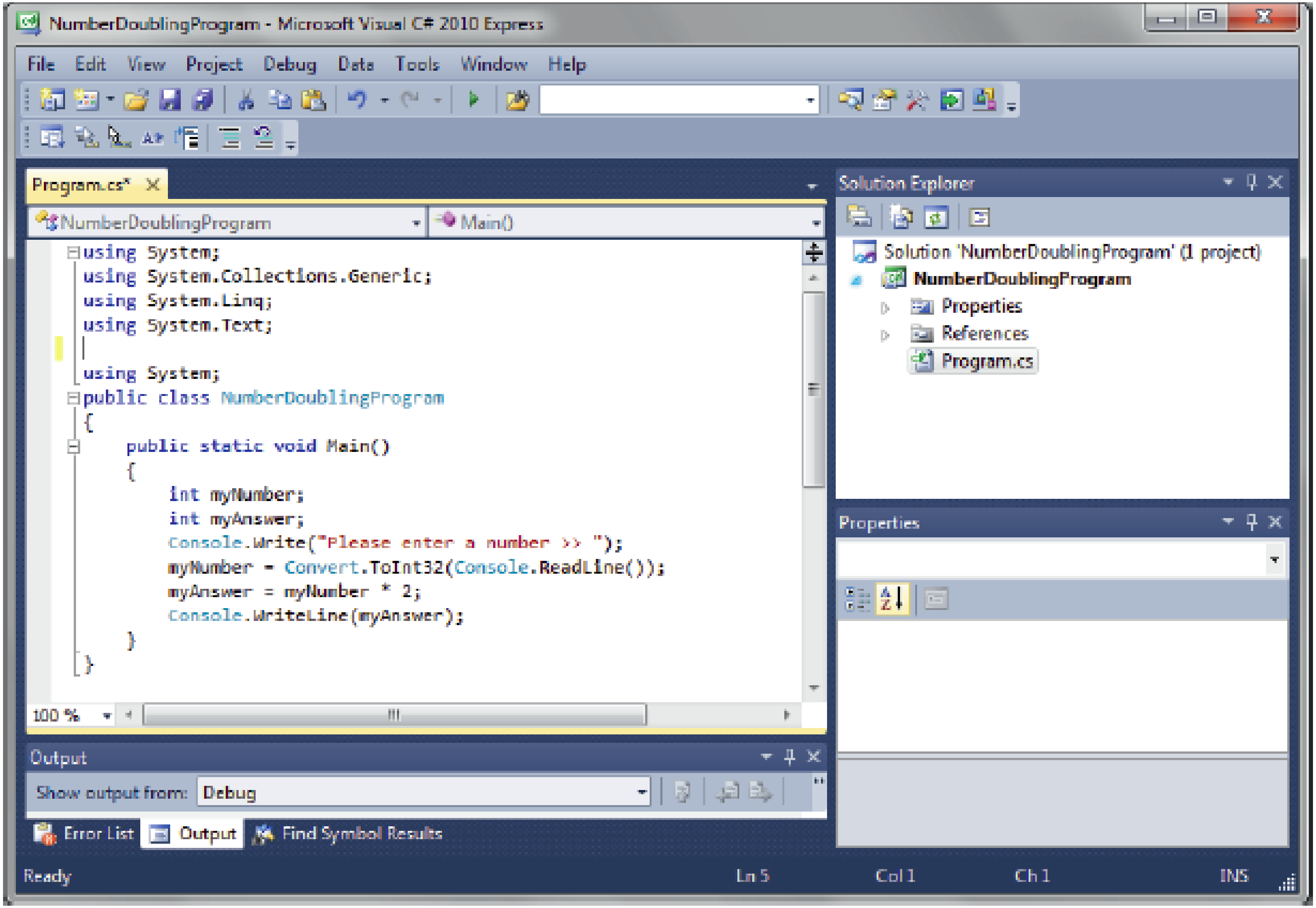
1.5 Understanding Programming and User Environments

- Many options for programming and user environments
  - Planning
    - Flowchart
    - Pseudocode
  - Coding
    - Text editors
  - Executing
    - Input from keyboard, mouse, microphone
  - Outputting
    - Text, images, sound

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, a compiler, and other programming tools

A C# Number-Doubling Program in Visual Studio



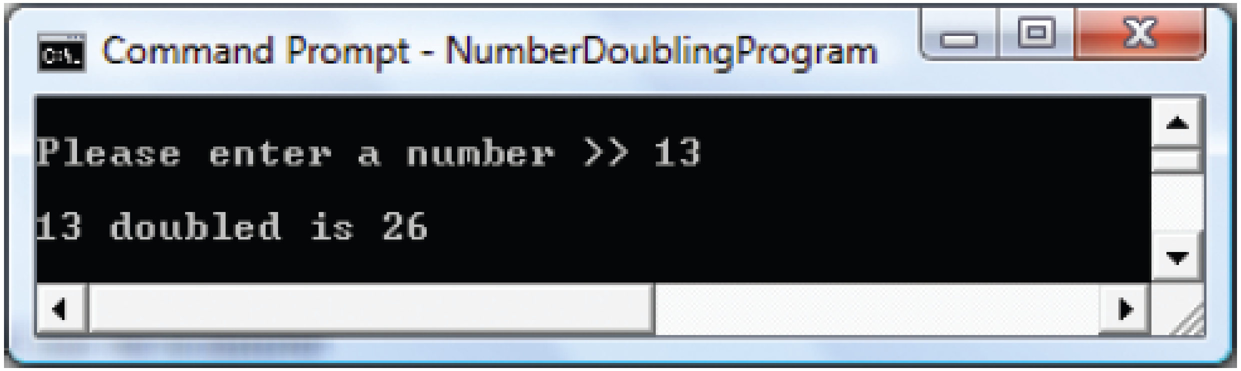
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System;
public class NumberDoublingProgram
{
    public static void Main()
    {
        int myNumber;
        int myAnswer;
        Console.Write("Please enter a number >> ");
        myNumber = Convert.ToInt32(Console.ReadLine());
        myAnswer = myNumber * 2;
        Console.WriteLine(myAnswer);
    }
}
```

Understanding User Environments

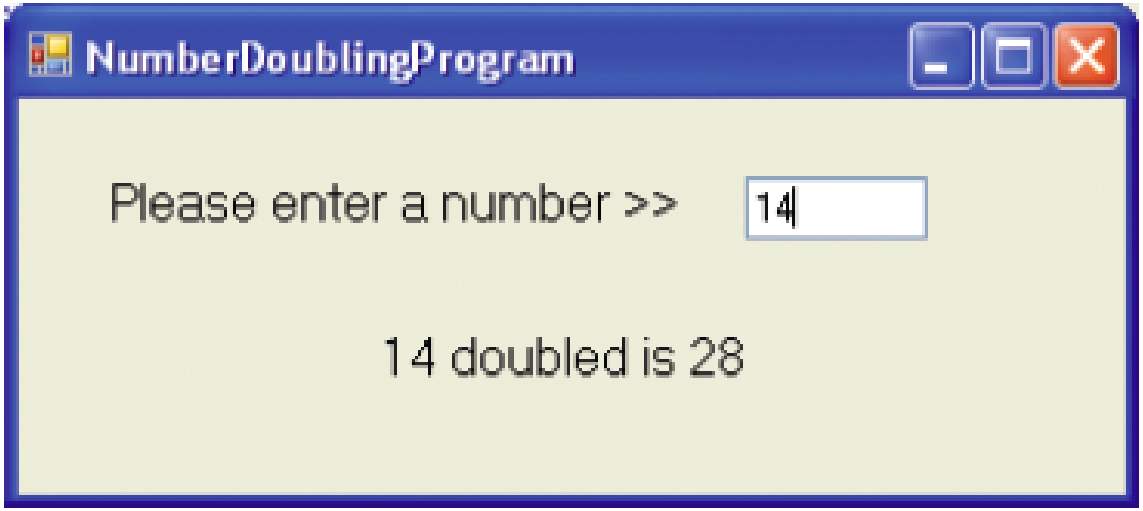
- **Command line**
  - Location on your computer screen where you type text entries to communicate with the computer's operating system
- **Graphical user interface** (**GUI**)
  - Allows users to interact with a program in a graphical environment

Executing a Number-Doubling Program in a Command-Line Environment



```
Please enter a number >> 13
13 doubled is 26
```

Executing a Number-Doubling Program in a GUI Environment



```
Please enter a number >>   14

14 doubled is 28
```

1.6 Understanding the Evolution of Programming Models

- People have been writing modern computer programs since the 1940s
- Newer programming languages
  - Look much more like natural language
  - Are easier to use
  - Create self-contained modules or program segments that can be pieced together in a variety of ways

Understanding the Evolution of Programming Models

- 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
  - This text
    - Focuses on procedural programming techniques

1.7 Summary

- Hardware and software accomplish input, processing, and output
- Logic must be developed correctly
- Logical errors are much more difficult to locate than syntax errors
- Use flowcharts, pseudocode charts to plan the logic
- Avoid infinite loops by testing for a sentinel value
- Use a text editor or an IDE to enter your program statements

1.8 Programming Exercises

1. Draw a flowchart or write pseudocode to represent the logic of a program that allows the user to enter a value. The program divides the value by 2 and outputs the result.
2. Draw a flowchart or write pseudocode to represent the logic of a program that allows the user to enter a value for one edge of a cube. The program calculates the surface area of one side of the cube, the surface area of the cube, and its volume. The program outputs all the results.

Programming Exercises

3. Draw a flowchart or write pseudocode to represent the logic of a program that allows the user to enter values for a salesperson's base salary, total sales, and commission rate. The program computes and outputs the salesperson's pay by adding the base salary to the product of the total sales and commission rate.

4. A mobile phone app allows a user to press a button that starts a timer that counts seconds. When the user presses the button again, the timer stops. Draw a flowchart or write pseudocode that accepts the elapsed time in seconds and displays the value in minutes and seconds. For example, if the elapsed time was 130 seconds, the output would be 2 minutes and 10 seconds.