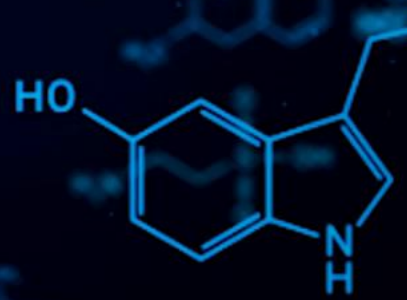
# Introduction to C/C++ Programming

CS5008/CS5009



# Objectives

- Discover what a compiler is and what it does
- Examine a C++ program
- Explore how a C++ program is processed
- Learn what an algorithm is and explore problem-solving techniques
- Become aware of structured design and object-oriented design programming methodologies
- Become aware of Standard C++, ANSI/ISO Standard C++, and C++11

# Introduction

- Without software, the computer is useless
- Software is developed with programming languages
  - C++ is a programming language
- C++ suited for a wide variety of programming tasks

# Elements of a Computer System

- Hardware
- CPU
- Main memory
- Secondary storage
- Input/Output devices
- Software

# Hardware

- CPU
- Main memory: RAM
- Input/output devices
- Secondary storage

# Central Processing Unit and Main Memory

- Central processing unit
  - Brain of the computer
  - Most expensive piece of hardware
  - Carries out arithmetic and logical operations

# Central Processing Unit and Main Memory (cont'd.)

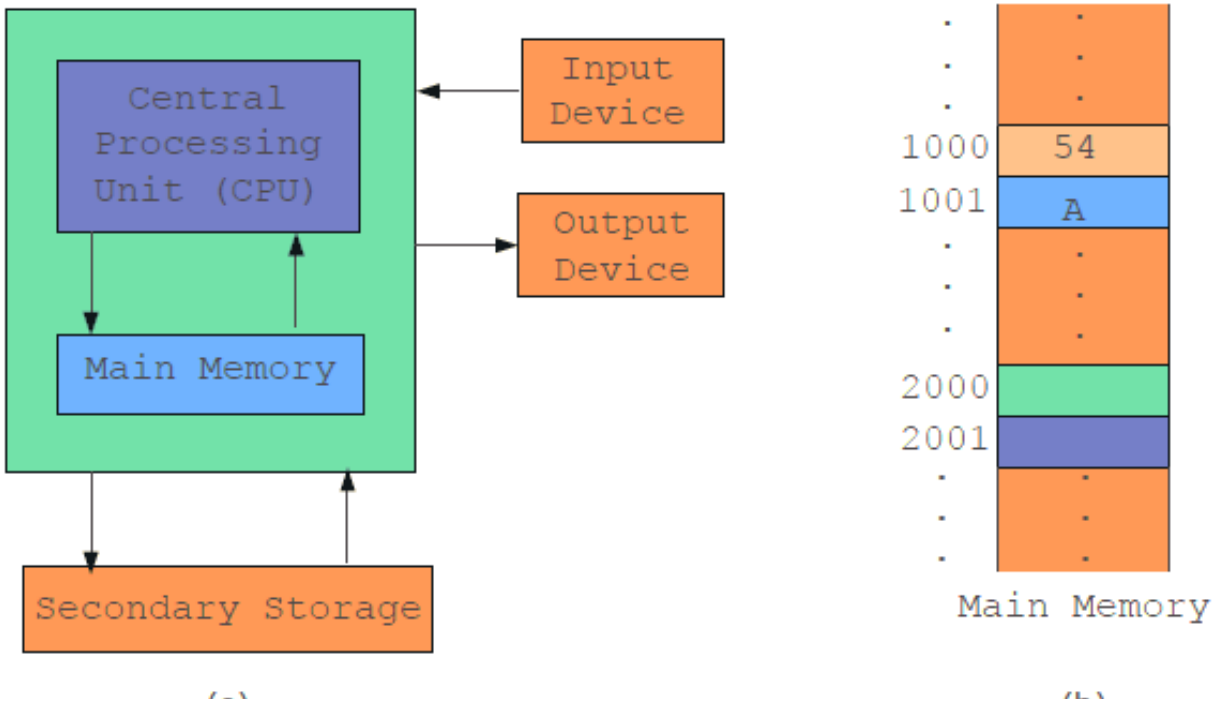



FIGURE 1-1 Hardware components of a computer and main memory

# Central Processing Unit and Main Memory (cont'd.)

- Random access memory
  - Directly connected to the CPU
- All programs must be loaded into main memory before they can be executed
- All data must be brought into main memory before it can be manipulated
- When computer power is turned off, everything in main memory is lost

# Central Processing Unit and Main Memory (cont'd.)

- Main memory is an ordered sequence of memory cells
  - Each cell has a unique location in main memory, called the address of the cell
- Each cell can contain either a programming instruction or data

# Secondary Storage

- Secondary storage: device that stores information permanently
- Examples of secondary storage:
  - Hard disks
  - Flash drives
  - Floppy disks
  - Zip disks
  - CD-ROMs
  - Tapes

# Input/Output Devices

- Input devices feed data and programs into computers
  - Keyboard
  - Mouse
  - Secondary storage
- Output devices display results
  - Monitor
  - Printer
  - Secondary storage

# Software

- Software: programs that do specific tasks
- System programs control the computer
  - Operating system monitors the overall activity of the computer and provides services such as:
    - Memory management
    - Input/output activities
    - Storage management
- Application programs perform a specific task
  - Word processors
  - Spreadsheets
  - Games

# The Language of a Computer

- Analog signals: continuous wave forms
- Digital signals: sequences of 0s and 1s
- Machine language: language of a computer; a sequence of 0s and 1s
- Binary digit (bit): the digit 0 or 1
- Binary code (binary number): a sequence of 0s and 1s

# The Language of a Computer (cont'd.)

- Byte:
  - A sequence of eight bits
- Kilobyte (KB): $2^{10}$ bytes = 1024 bytes
- ASCII (American Standard Code for Information Interchange)
  - 128 characters
  - A is encoded as 1000001 (66th character)
  - 3 is encoded as 0110011

# The Language of a Computer (cont'd.)

**TABLE 1-1** Binary Units

| Unit | Symbol | Bits/Bytes |
|---|---|---|
| Byte | | 8 bits |
| Kilobyte | KB | $2^{10}$ bytes = 1024 bytes |
| Megabyte | MB | 1024 KB = $2^{10}$ KB = $2^{20}$ bytes = 1,048,576 bytes |
| Gigabyte | GB | 1024 MB = $2^{10}$ MB = $2^{30}$ bytes = 1,073,741,824 bytes |
| Terabyte | TB | 1024 GB = $2^{10}$ GB = $2^{40}$ bytes = 1,099,511,627,776 bytes |
| Petabyte | PB | 1024 TB = $2^{10}$ TB = $2^{50}$ bytes = 1,125,899,906,842,624 bytes |
| Exabyte | EB | 1024 PB = $2^{10}$ PB = $2^{60}$ bytes = 1,152,921,504,606,846,976 bytes |
| Zettabyte | ZB | 1024 EB = $2^{10}$ EB = $2^{70}$ bytes = 1,180,591,620,717,411,303,424 bytes |

# The Evolution of Programming Languages

- Early computers were programmed in machine language
- To calculate `wages = rate * hours` in machine language:

```
100100 010001     //Load

100110 010010     //Multiply

100010 010011     //Store
```

# The Evolution of Programming Languages (cont'd.)

- Assembly language instructions are mnemonic
- Assembler: translates a program written in assembly language into machine language
- Using assembly language instructions, `wages = rate * hours` can be written as:

```
LOAD   rate
MULT   hour
STOR   wages
```

# The Evolution of Programming Languages (cont'd.)

- High-level languages include Basic, FORTRAN, COBOL, Pascal, C, C++, C#, and Java
- Compiler: translates a program written in a high-level language into machine language
- The equation `wages = rate • hours` can be written in C++ as:

```
wages = rate * hours;
```

# Processing a C++ Program

```
#include <iostream>
using namespace std;
int main()
{
    cout << "My first C++ program." << endl;
   return 0;
}
```

**Sample Run**:

```
My first C++ program.
```

# Processing a C++ Program (cont'd.)

- To execute a C++ program:
  - Use an editor to create a source program in C++
  - Preprocessor directives begin with # and are processed by the preprocessor
  - Use the compiler to:
    - Check that the program obeys the language rules
    - Translate into machine language (object program)

# Processing a C++ Program (cont'd.)

- To execute a C++ program (cont'd.):
  - Linker:
    - Combines object program with other programs provided by the SDK to create executable code
    - Library: contains prewritten code you can use
  - Loader:
    - Loads executable program into main memory
  - The last step is to execute the program
- Some IDEs do all this with a Build or Rebuild command

# Processing a C++ Program (cont'd.)

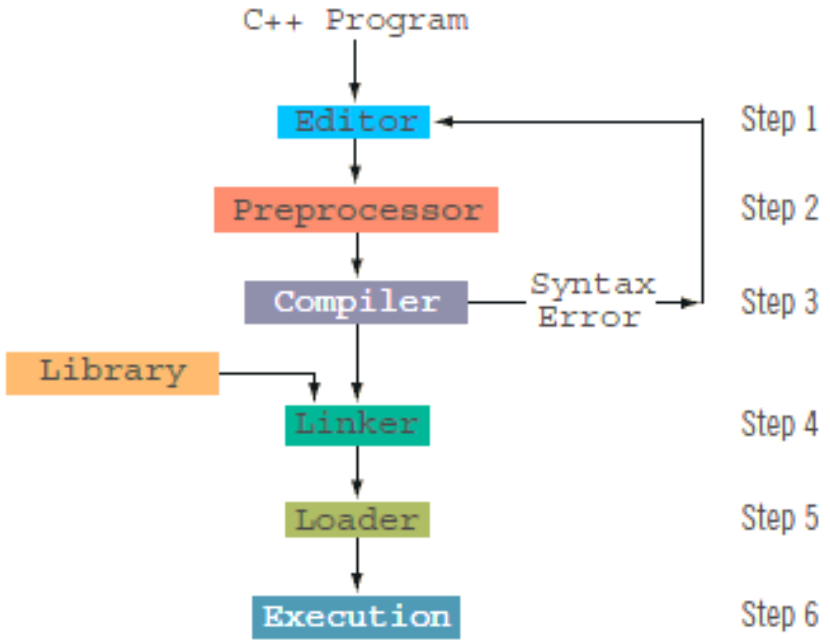



**FIGURE 1-2** Processing a C++ program

# Programming with the Problem Analysis–Coding–Execution Cycle

- Algorithm:
  - Step-by-step problem-solving process
  - Solution achieved in finite amount of time
- Programming is a process of problem solving

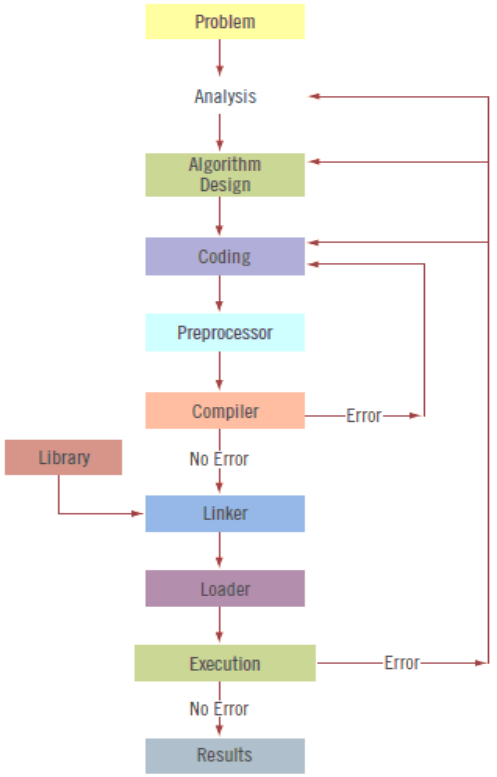



**FIGURE 1-3** Problem analysis–coding–execution cycle

# The Problem Analysis–Coding–Execution Cycle (cont'd.)

- Step 1: Analyze the problem
  - Outline the problem and its requirements
  - Design steps (algorithm) to solve the problem
- Step 2: Implement the algorithm
  - Implement the algorithm in code
  - Verify that the algorithm works
- Step 3: Maintain
  - Use and modify the program if the problem domain changes

# The Problem Analysis–Coding–Execution Cycle (cont'd.)

- Thoroughly understand the problem and all requirements
  - Does program require user interaction?
  - Does program manipulate data?
  - What is the output?
- If the problem is complex, divide it into subproblems
  - Analyze and design algorithms for each subproblem
- Check the correctness of algorithm
  - Can test using sample data
  - Some mathematical analysis might be required

# The Problem Analysis–Coding–Execution Cycle (cont'd.)

- Once the algorithm is designed and correctness verified
  - Write the equivalent code in high-level language
- Enter the program using text editor

# The Problem Analysis–Coding–Execution Cycle (cont'd.)

- Run code through compiler
- If compiler generates errors
  - Look at code and remove errors
  - Run code again through compiler
- If there are no syntax errors
  - Compiler generates equivalent machine code
- Linker links machine code with system resources

# The Problem Analysis–Coding–Execution Cycle (cont'd.)

- Once compiled and linked, loader can place program into main memory for execution
- The final step is to execute the program
- Compiler guarantees that the program follows the rules of the language
  - Does not guarantee that the program will run correctly

# Programming Methodologies

- Two popular approaches to programming design
  - Structured
  - Object-oriented

# Structured Programming

- Structured design:
  - Dividing a problem into smaller subproblems
- Structured programming:
  - Implementing a structured design
- The structured design approach is also called:
  - Top-down (or bottom-up) design
  - Stepwise refinement
  - Modular programming

# Object-Oriented Programming

- Object-oriented design (OOD)
  - Identify components called objects
  - Determine how objects interact with each other
- Specify relevant data and possible operations to be performed on that data
- Each object consists of data and operations on that data

# Object-Oriented Programming (cont'd.)

- An object combines data and operations on the data into a single unit
- A programming language that implements OOD is called an object-oriented programming (OOP) language
- Must learn how to represent data in computer memory, how to manipulate data, and how to implement operations

# A Quick Look at a C++ Program

```
#include <iostream>

using namespace std;

int main()
{
    double length;
    double width;
    double area;
    double perimeter;

    cout << "Program to compute and output the perimeter and "
         << "area of a rectangle." << endl;

    length = 6.0;
    width = 4.0;
    perimeter = 2 * (length + width);
    area = length * width;

    cout << "Length = " << length << endl;
    cout << "Width =  " << width << endl;
    cout << "Perimeter = " << perimeter << endl;
    cout << "Area = " << area << endl;

    return 0;
}
```

# (cont'd.)

- Sample run:

```
Program to compute and output the perimeter and area of a rectangle.
Length = 6
Width =  4
Perimeter = 20
Area = 24
```

# (cont'd.)

```
//*************************************************************
// Given the length and width of a rectangle, this C++ program
// computes and outputs the perimeter and area of the rectangle.
//*************************************************************

#include <iostream>

using namespace std;

int main()
{
    double length;
    double width;
    double area;
    double perimeter;

    cout << "Program to compute and output the perimeter and "
         << "area of a rectangle." << endl;

    length = 6.0;
```

Comments

Variable declarations. A statement such as `double length;` instructs the system to allocate memory space and name it `length`.

Assignment statement. This statement instructs the system to store `6.0` in the memory space `length`.

# A Quick Look at a C++ Program(cont'd.)

```
    width = 4.0;
    perimeter = 2 * (length + width);

    area = length * width;

    cout << "Length = " << length << endl;
    cout << "Width =  " << width << endl;
    cout << "Perimeter = " << perimeter << endl;
    cout << "Area = " << area << endl;

    return 0;
}
```

Assignment statement. This statement instructs the system to evaluate the expression `length * width` and store the result in the memory space `area`.

Output statements. An output statement instructs the system to display results.

**FIGURE 2-1** Various parts of a C++ program

# A Quick Look at a C++ Program(cont'd.)

- Variable: a memory location whose contents can be changed


**Figure 2-2 Memory allocation**


**Figure 2-3 Memory spaces after the statement** length = 6.0; **executes**

# The Basics of a C++ Program

- Function (or subprogram): collection of statements; when executed, accomplishes something
  - May be predefined or standard
- Syntax rules: rules that specify which statements (instructions) are legal or valid
- Semantic rules: determine the meaning of the instructions
- Programming language: a set of rules, symbols, and special words

# Comments

- Comments are for the reader, not the compiler
- Two types:
  - Single line: begin with //

```
// This is a C++ program.
// Welcome to C++ Programming.
```

  - Multiple line: enclosed between /* and */

```
/*
   You can include comments that can
   occupy several lines.
*/
```

# Special Symbols

- Token: the smallest individual unit of a program written in any language
- C++ tokens include special symbols, word symbols, and identifiers
- Special symbols in C++ include:

```
+     -     *     /
.     ;     ?     ,
<=    !=    ==    >=
```

# Reserved Words (Keywords)

- Reserved word symbols (or keywords):
  - Cannot be redefined within program
  - Cannot be used for anything other than their intended use

  Examples:
    - `int`
    - `float`
    - `double`
    - `char`
    - `const`
    - `void`
    - `return`

# Identifiers

- Identifier: the name of something that appears in a program
  - Consists of letters, digits, and the underscore character (_)
  - Must begin with a letter or underscore
- C++ is case sensitive
  - `NUMBER` is not the same as `number`
- Two predefined identifiers are `cout` and `cin`
- Unlike reserved words, predefined identifiers may be redefined, but it is not a good idea

# Identifiers (cont'd.)

- Legal identifiers in C++:
  - `first`
  - `conversion`
  - `payRate`

TABLE 2-1 Examples of Illegal Identifiers

| Illegal Identifier | Description |
|---|---|
| `employee Salary` | There can be no space between `employee` and `Salary`. |
| `Hello!` | The exclamation mark cannot be used in an identifier. |
| `one+two` | The symbol + cannot be used in an identifier. |
| `2nd` | An identifier cannot begin with a digit. |

# Whitespaces

- Every C++ program contains whitespaces
  - Include blanks, tabs, and newline characters
- Used to separate special symbols, reserved words, and identifiers
- Proper utilization of whitespaces is important
  - Can be used to make the program more readable

# Data Types

- Data type: set of values together with a set of operations
- C++ data types fall into three categories:
  - Simple data type
  - Structured data type
  - Pointers

# Simple Data Types

- Three categories of simple data
  - Integral: integers (numbers without a decimal)
    - Can be further categorized:
      - `char, short, int, long, bool, unsigned char, unsigned short, unsigned int, unsigned long`
  - Floating-point: decimal numbers
  - Enumeration type: user-defined data type

# Simple Data Types (cont'd.)

TABLE 2-2 Values and Memory Allocation for Simple Data Types

| Data Type | Values | Storage (in bytes) |
|---|---|---|
| `int` | –2147483648 (= $-2^{31}$) to 2147483647 (= $2^{31} - 1$) | 4 |
| `bool` | `true` and `false` | 1 |
| `char` | –128 (= $-2^7$) to 127 (= $2^7 - 1$) | 1 |
| `long long` | –9223372036854775808 ($-2^{63}$) to 9223372036854775807 ($2^{63} - 1$) | 64 |

- Different compilers may allow different ranges of values

# Floating-Point Data Types (cont'd.)

- Maximum number of significant digits (decimal places) for `float` values: 6 or 7
- Maximum number of significant digits for `double`: 15
- Precision: maximum number of significant digits
  - Float values are called single precision
  - Double values are called double precision

# Type Conversion (Casting)

- Implicit type coercion: when value of one type is automatically changed to another type
- Cast operator: provides explicit type conversion

```
static_cast<dataTypeName>(expression)
```

# Type Conversion (cont'd.)

**EXAMPLE 2-9**

| Expression | Evaluates to |
|---|---|
| `static_cast<int>(7.9)` | 7 |
| `static_cast<int>(3.3)` | 3 |
| `static_cast<double>(25)` | 25.0 |
| `static_cast<double>(5+3)` | = `static_cast<double>(8)` = 8.0 |
| `static_cast<double>(15)/2` | = 15.0 / 2<br>(because `static_cast<double>(15)` = 15.0)<br>= 15.0 / 2.0 = 7.5 |
| `static_cast<double>(15/2)` | = `static_cast<double>(7)` (because 15 / 2 = 7)<br>= 7.0 |
| `static_cast<int>(7.8 +`<br>`static_cast<double>(15)/2)` | = `static_cast<int>(7.8+7.5)`<br>= `static_cast<int>(15.3)`<br>= 15 |
| `static_cast<int>(7.8 +`<br>`static_cast<double>(15/2))` | = `static_cast<int>(7.8 + 7.0)`<br>= `static_cast<int>(14.8)`<br>= 14 |

# `string` Type

- Programmer-defined type supplied in ANSI/ISO Standard C++ library
- Sequence of zero or more characters enclosed in double quotation marks
- Null (or empty): a string with no characters
- Each character has a relative position in the string
  - Position of first character is 0
- Length of a string is number of characters in it
  - Example: length of `"William Jacob"` is 13

# Variables, Assignment Statements, and Input Statements

- Data must be loaded into main memory before it can be manipulated
- Storing data in memory is a two-step process:
  - Instruct computer to allocate memory
  - Include statements to put data into memory

# Allocating Memory with Constants and Variables

- Named constant: memory location whose content can't change during execution

```
const dataType identifier = value;
```

- Syntax to declare a named constant:

- In C++, `const` is a reserved word

**EXAMPLE 2-11**

Consider the following C++ statements:

```
const double CONVERSION = 2.54;
const int NO_OF_STUDENTS = 20;
const char BLANK = ' ';
```

# Allocating Memory with Constants and Variables (cont'd.)

- Variable: memory location whose content may change during execution
- Syntax

```
dataType identifier, identifier, . . .;
```

**EXAMPLE 2-12**

Consider the following statements:

```
double amountDue;
int counter;
char ch;
int x, y;
string name;
```

# Assignment Statement (cont'd.)

**EXAMPLE 2-13**

Suppose you have the following variable declarations:

```
int num1, num2;
double sale;
char first;
string str;
```

Now consider the following assignment statements:

```
num1 = 4;
num2 = 4 * 5 - 11;
sale = 0.02 * 1000;
first = 'D';
str = "It is a sunny day.";
```

# Input (Read) Statement

- `cin` is used with >> to gather input

```
cin >> variable >> variable ...;
```

- This is called an input (read) statement
- The stream extraction operator is >>
- For example, if miles is a double variable

```
cin >> miles;
```

  - Causes computer to get a value of type `double` and places it in the variable `miles`

**EXAMPLE 2-17**

```
#include <iostream>

using namespace std;

int main()
{
    int feet;
    int inches;

    cout << "Enter two integers separated by one or more spaces: ";
    cin >> feet >> inches;
    cout << endl;

    cout << "Feet = " << feet << endl;
    cout << "Inches = " << inches << endl;

    return 0;
}
```

**Sample Run:** In this sample run, the user input is shaded.

```
Enter two integers separated by one or more spaces: 23 7

Feet = 23
Inches = 7
```

# Output

- The syntax of `cout` and << is:

```
cout << expression or manipulator << expression or manipulator...;
```

  - Called an output statement
- The stream insertion operator is <<
- Expression evaluated and its value is printed at the current cursor position on the screen

# Output (cont'd.)

- A manipulator is used to format the output
  - Example: `endl` causes insertion point to move to beginning of next line

**EXAMPLE 2-21**

Consider the following statements. The output is shown to the right of each statement.

| | Statement | Output |
|---|---|---|
| 1 | `cout << 29 / 4 << endl;` | `7` |
| 2 | `cout << "Hello there." << endl;` | `Hello there.` |
| 3 | `cout << 12 << endl;` | `12` |
| 4 | `cout << "4 + 7" << endl;` | `4 + 7` |
| 5 | `cout << 4 + 7 << endl;` | `11` |
| 6 | `cout << 'A' << endl;` | `A` |
| 7 | `cout << "4 + 7 = " << 4 + 7 << endl;` | `4 + 7 = 11` |
| 8 | `cout << 2 + 3 * 5 << endl;` | `17` |
| 9 | `cout << "Hello \nthere." << endl;` | `Hello`<br>`there.` |

# Output (cont'd.)

TABLE 2-4 Commonly Used Escape Sequences

| | Escape Sequence | Description |
|---|---|---|
| `\n` | Newline | Cursor moves to the beginning of the next line |
| `\t` | Tab | Cursor moves to the next tab stop |
| `\b` | Backspace | Cursor moves one space to the left |
| `\r` | Return | Cursor moves to the beginning of the current line (not the next line) |
| `\\` | Backslash | Backslash is printed |
| `\'` | Single quotation | Single quotation mark is printed |
| `\"` | Double quotation | Double quotation mark is printed |

# Preprocessor Directives

- C++ has a small number of operations
- Many functions and symbols needed to run a C++ program are provided as collection of libraries
- Every library has a name and is referred to by a header file
- Preprocessor directives are commands supplied to the preprocessor program
- All preprocessor commands begin with #
- No semicolon at the end of these commands

# Preprocessor Directives (cont'd.)

- Syntax to include a header file:

```
#include <headerFileName>
```

- For example:

```
#include <iostream>
```

  - Causes the preprocessor to include the header file `iostream` in the program
- Preprocessor commands are processed before the program goes through the compiler

# namespace and Using cin and cout in a Program

- cin and cout are declared in the header file iostream, but within std namespace
- To use cin and cout in a program, use the following two statements:

```
#include <iostream>

using namespace std;
```

# Using the `string` Data Type in a Program

- To use the `string` type, you need to access its definition from the header file `string`
- Include the following preprocessor directive:

```
#include   <string>
```

# Reference


- C++ Programming: Program Design Including Data Structures, Seventh Edition