University of Sadat City
Faculty of Computers and Artificial Intelligence (FCAI)

# Fundamentals of Programming (CS101)

## Lecture 1

*Presented By:*

**Dr. Heba Askr**

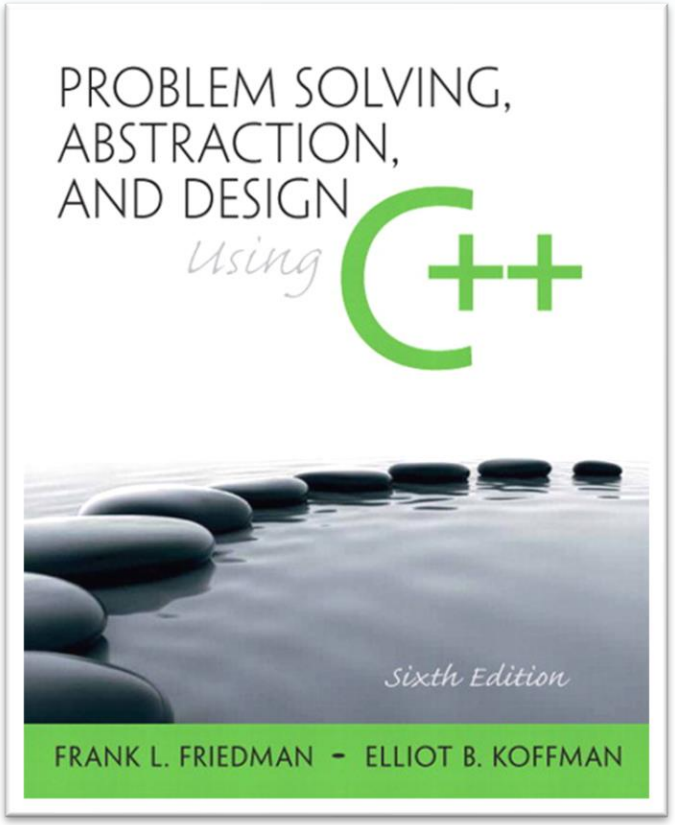
**Second Term 2024-2025**

# Course Objectives

The main objective of this course is to provide students with **computer programming**, **programming languages** and generations, **programming life cycle**, **programming errors, problem solving techniques**, what is **algorithm**, algorithm representation (**Pseudo code**), sequential operations, conditional operations, iterative operations, what is **flowchart**, flowchart notations, program construction, constants, **variable** declarations, simple **data types**, **input statement, output statement**, formatting output, arithmetic expressions, control structures, conditions, selection structures, **repetition and loop statements**, **library functions**, **user-defined functions**, **function arguments**, **array declarations**, array subscripts, array operations, **array sorting** and searching, **multidimensional arrays**.

# Course Resources

1. Problem Solving, Abstraction, and Design using C++.
2. Handouts when necessary.
3. Lecture Notes.
4. Internet resources when necessary.
5. Computer Labs for Practical work.



# Course Assessment

| Assessment | weight |
|---|---|
| Mid term exam | 20% |
| Quizzes | 10% |
| Practical exam and attendance | 20% |
| Final exam | 50% |

# Chapter 1: Introduction to Computers, Problem Solving, and Programming

**Problem Solving,**
**Abstraction, and Design using C++ 6e**

**by Frank L. Friedman and Elliot B. Koffman**

# Categories of Computers

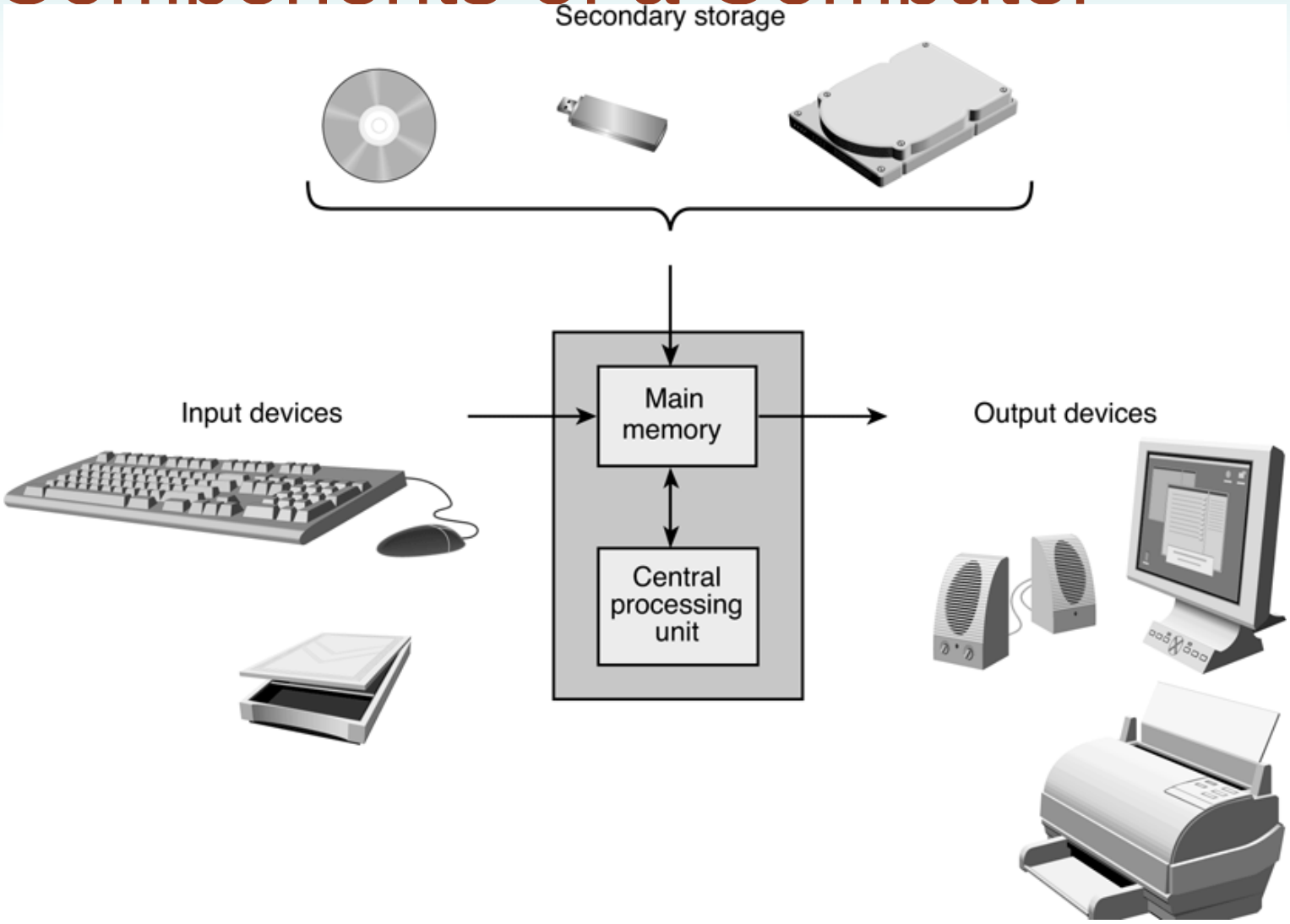
- Supercomputers: High-performance systems for complex computations.
- Mainframe: Large systems for centralized data processing in enterprises.
- Minicomputers: Smaller than mainframes, used in organizations for tasks like production and research.
- Microcomputers(Personal Computers): Desktops and laptops for individual use.
- Embedded Systems: Specialized systems within larger electronic devices.

# Categories of Computers

**Examples of Embedded Systems:**

- **Home Appliances**: Washing machines, microwaves, and smart refrigerators.
- **Automotive**: Engine control units (ECUs), anti-lock braking systems (ABS), and infotainment systems.
- **Healthcare Devices**: Pacemakers, blood pressure monitors, and ventilators.
- **Consumer Electronics**: Smart TVs, cameras, and gaming consoles.
- **Industrial Equipment**: Automated assembly lines and robotics.
- **IoT Devices**: Smart thermostats, home assistants, and wearable fitness trackers.

# Figure 1.3
# Components of a Computer



# Computer Hardware

- CPU - central processing unit
  - Where decisions are made, computations are performed, and input/output requests are delegated
- Main Memory
  - Stores information being processed by the CPU
- Secondary Memory (Mass Storage)
  - Stores data and programs

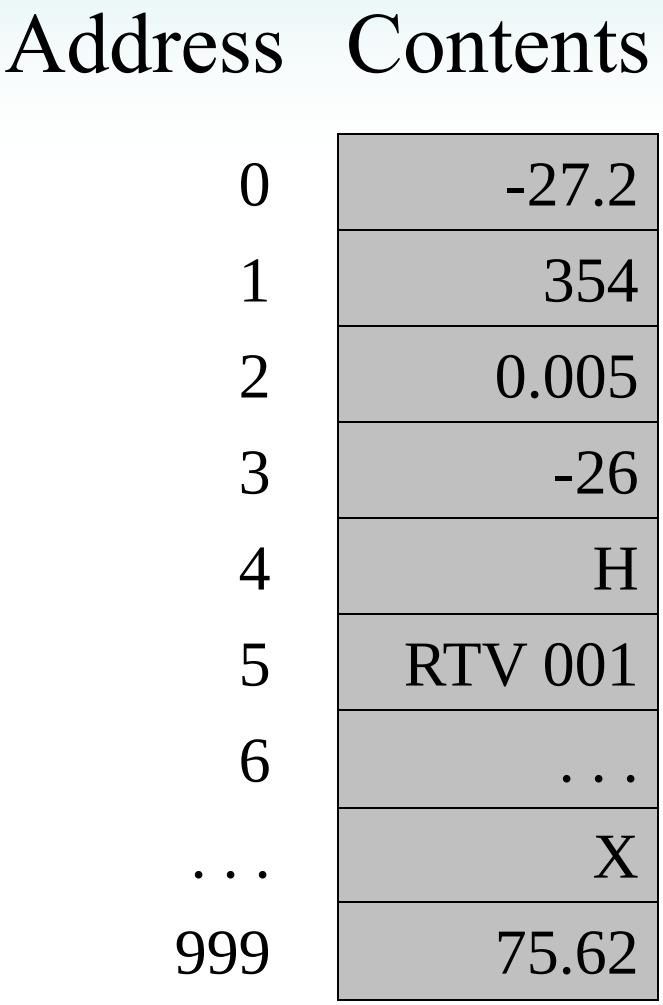
# Computer Hardware

- Input devices
  - Allow people to supply information to computers
- Output devices
  - Allow people to receive information from computers
- Network connection
  - Modems
  - Ethernet interface

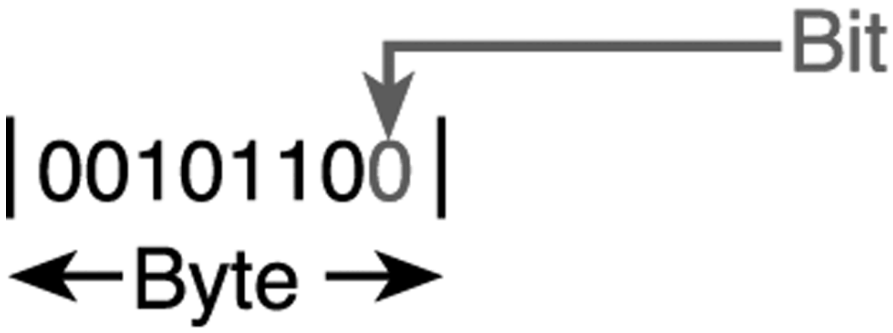
# Memory

- Stores
  - programs
    - operating system
    - applications
  - data
- Types
  - RAM - volatile
  - ROM
- Composed of **bits**, which are combined into **bytes**

# Memory Cells



# Figure 1.5
# Relationship Between a Byte and a Bit

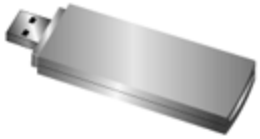


# Secondary Memory

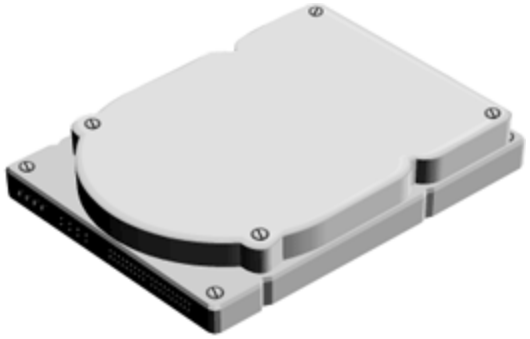
- Semi permanent data-storage capability
  - Magnetic
    - Hard disk
    - Floppy disk
    - Tape
  - Non-magnetic
    - CD or DVD
    - memory stick, flash drive
- Secondary memory usually has much more storage capacity than main memory

# Figure 1.6 Secondary Storage Media

CD



Flash drive



Hard disk

# CPU

- "Brains" of the computer
  - Arithmetic calculations are performed using the Arithmetic/Logical Unit or ALU
  - Control unit decodes and executes instructions
  - Registers hold information and instructions for CPU to process
- Arithmetic operations are performed using binary number system

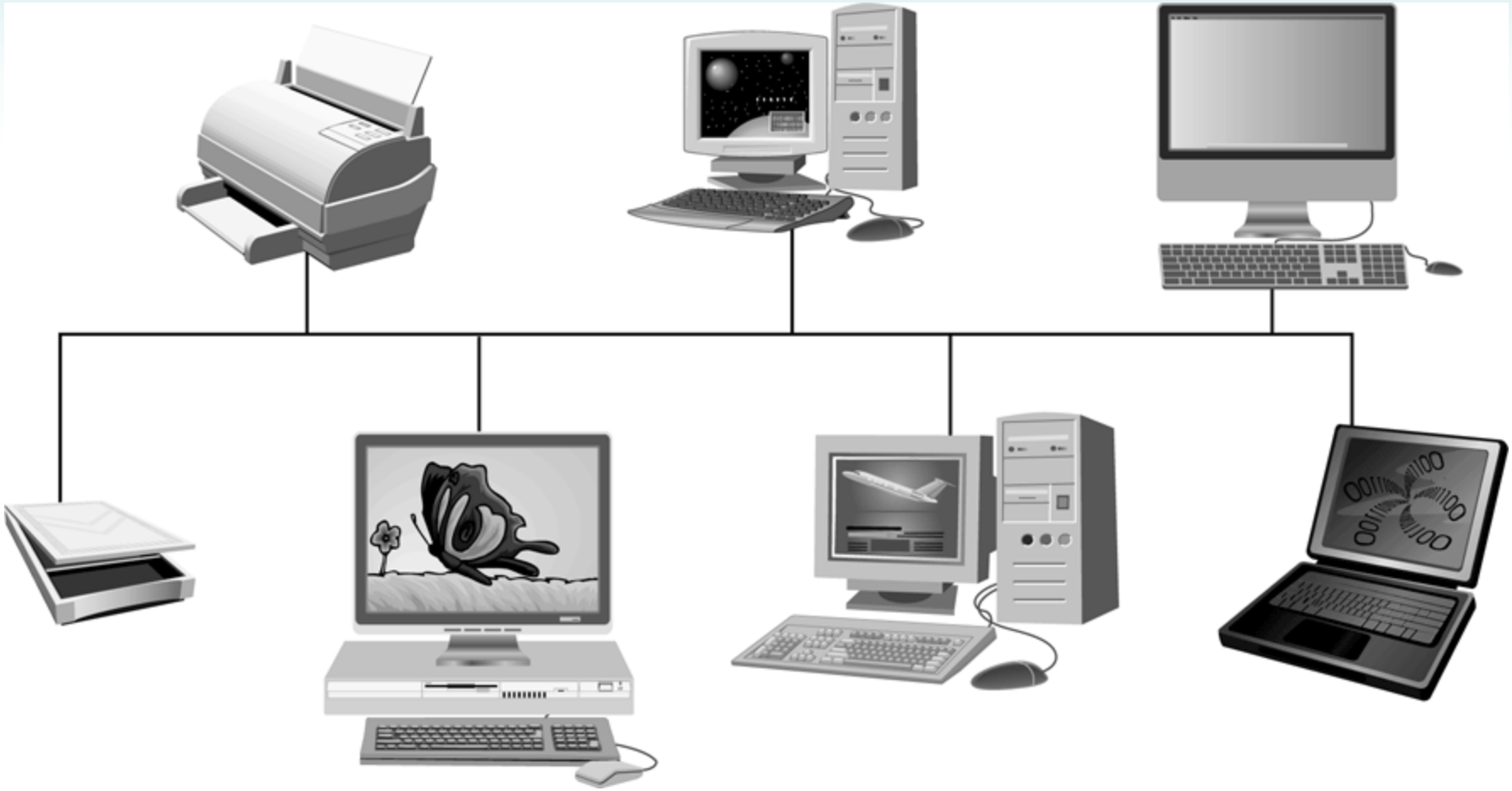
# Input / Output Devices

- Accessories that allow computer to perform specific tasks
  - Receiving information for processing
  - Return the results of processing
- Common input and output devices
  - Keyboard Joystick Scanner
  - Printer Monitor Speaker

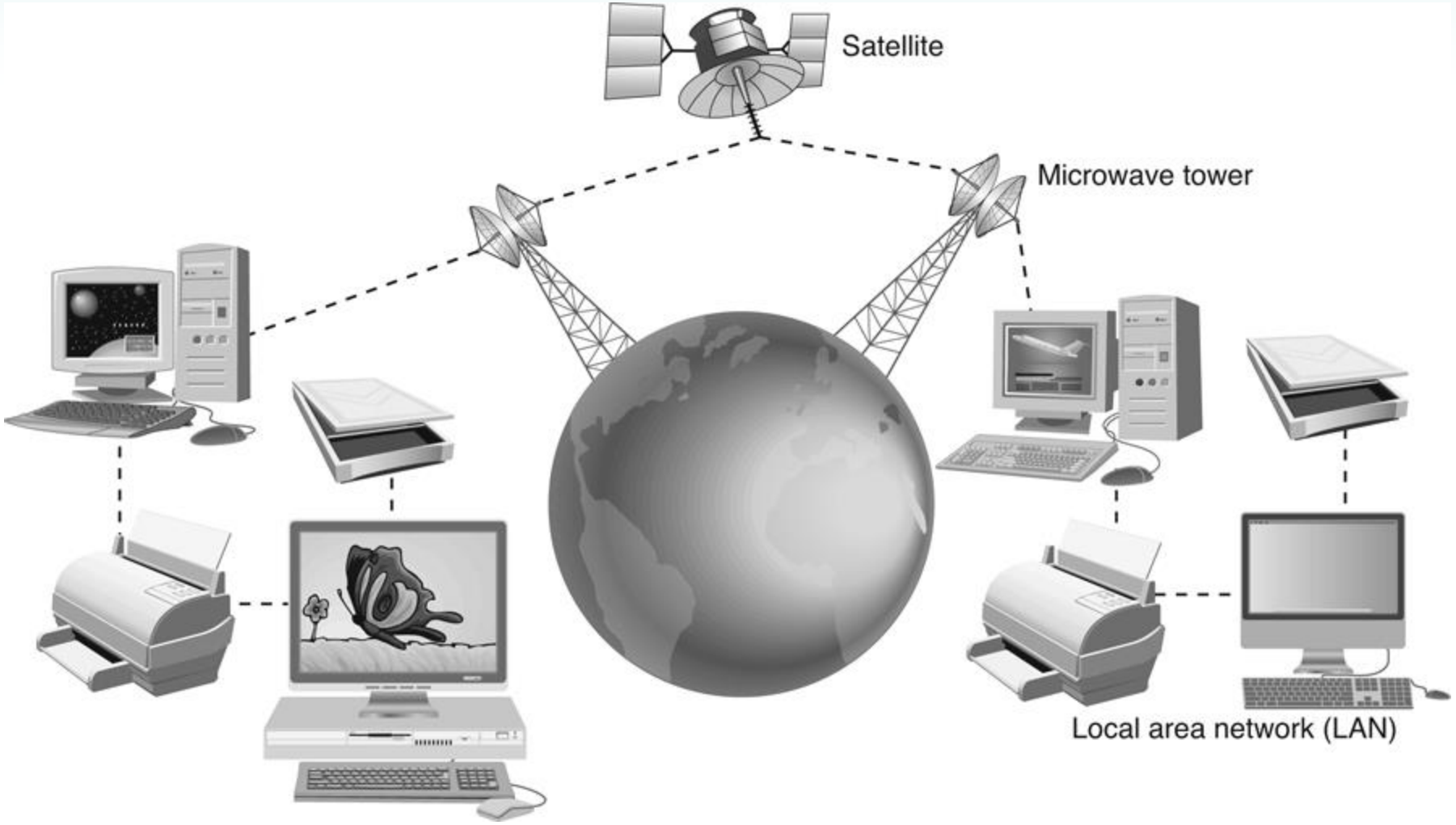
# Computer Networks

- Allows multiple computers to connect to share resources and/or data
- LAN - Local area network
  - Organizational
- WAN - Wide area network
  - Internet
- Requires additional hardware
  - modem
  - network interface

# Figure 1.7 Local Area Network



# Figure 1.8
# A Wide Area Network with Satellite Relays of Microwave Signals



# World Wide Web

- Introduced 1989
- Developed by CERN
  - European Laboratory for Particle Physics
- **Web browser**
  - MS Edge
  - Netscape
  - IE
  - Google chrome

# 1.3 Computer Software

- **Operating system**
- Other system software
  - utilities
  - programming language systems
- **Applications**

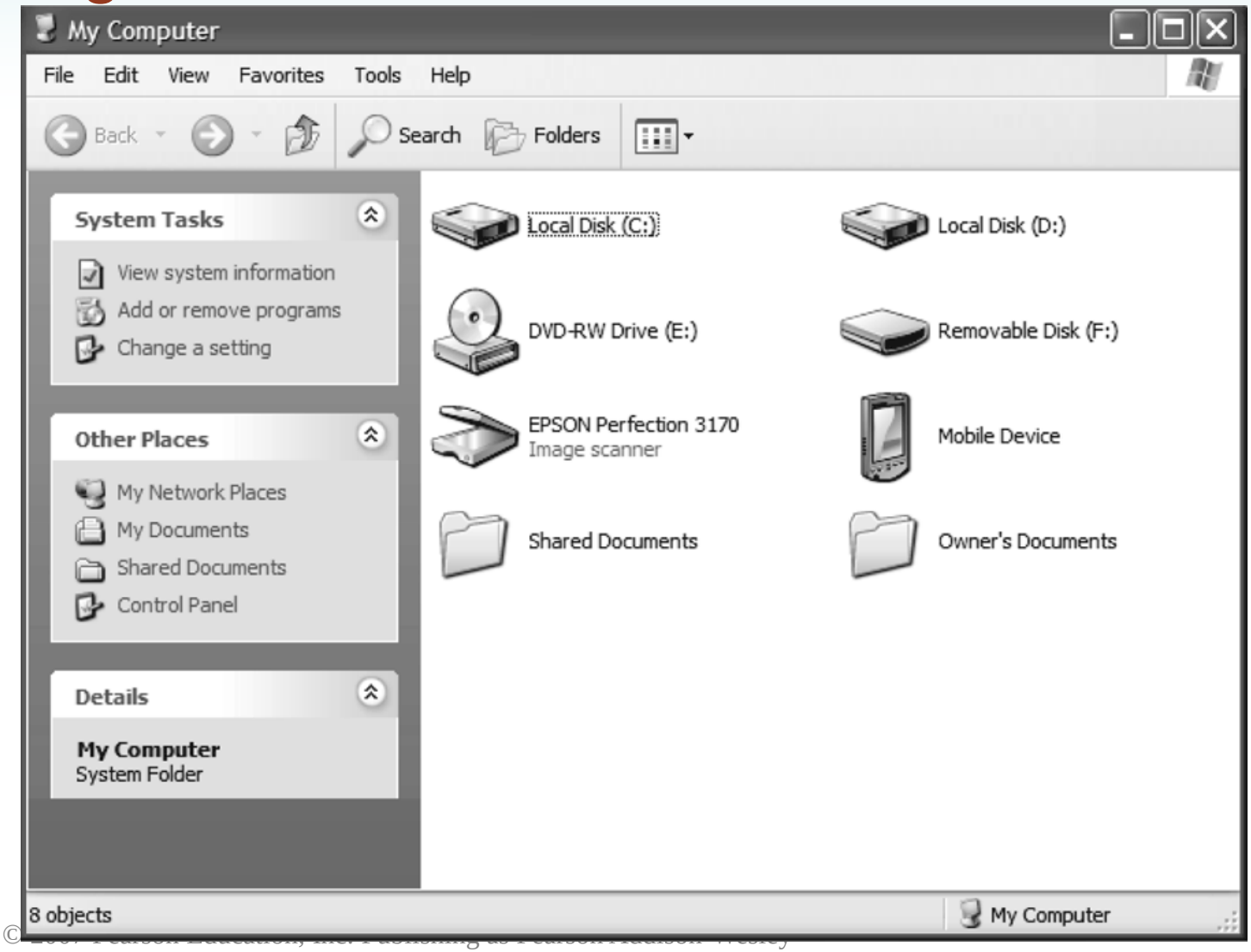
# Operating System

- E.g. Windows®, Unix®
- Controls
  - **the interaction of system with the user**
  - hardware interactions
- Part is usually stored on ROM, rest on hard drive
  - This arrangement requires **booting the system**

# Some OS Responsibilities

- Communicating with the user; receiving user commands
- Managing allocation of memory, processor time, file system, and other resources
- Collecting input from keyboard, mouse, etc.
- Conveying output to screen, printer, etc.
- Writing data to secondary storage devices

# Figure 1.10
# Accessing Secondary Storage Devices through Windows



# Application Software

- Common application software
  - Word processors
  - Desktop publishing programs
  - Spreadsheets
  - Presentation managers
  - Drawing programs

# Programming Languages

- **Machine Language**
  - Most fundamental language of the computer
  - Unique for each processor type
  - **Binary 0s and 1s that specify what to do**
    - **0010 0000 0000 0100**
    - **1000 0000 0000 0101**
    - **0011 0000 0000 0110**

# High - Level Languages

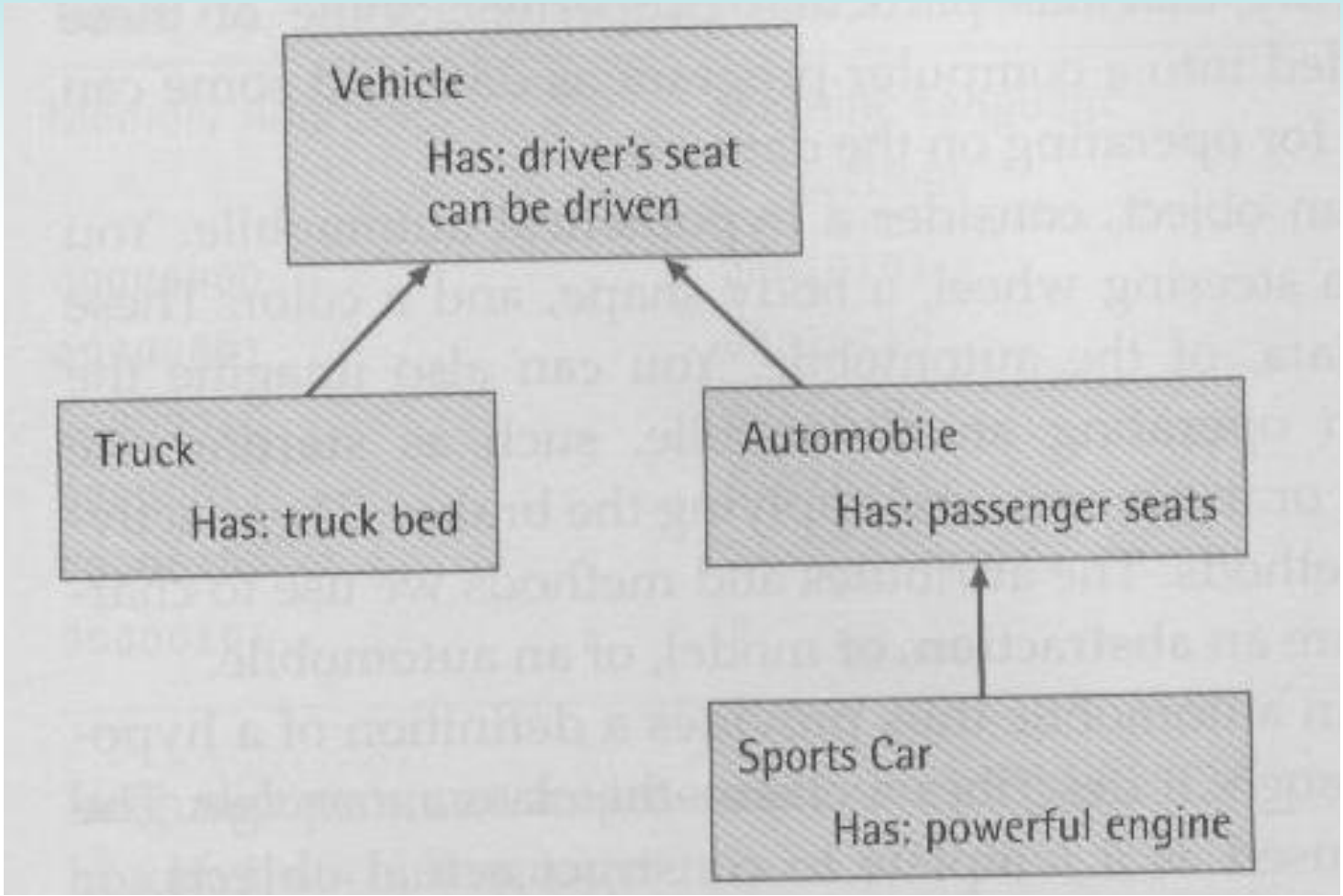
- Resemble human language
  - C++, C, Pascal, FORTRAN, Ada

```
a = a + b;
```

- More compact and human understandable than machine language
- Must be translated into machine language

# Object Oriented Programming

- C++ derived from C by Bjarne Stroustrup
- Popular because of reuse
  - Classes
  - Objects
  - Methods
- Organized in a Hierarchy
  - Super Classes
  - Sub Classes



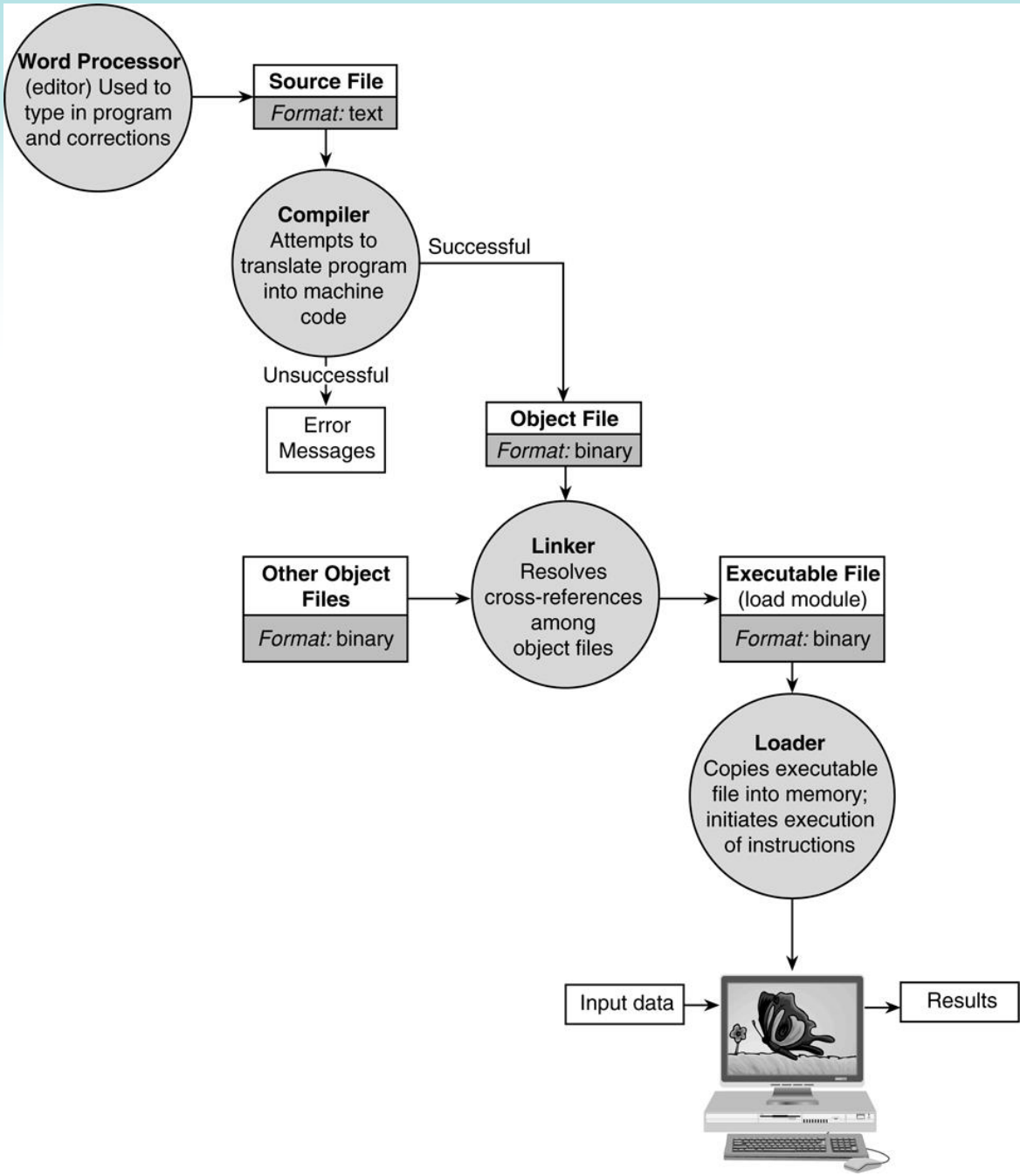
# 1.4 Processing a High-Level Language Program

- Set of programs used to develop software
- A key component is a translator
  - Compiler
- Examples
  - g++, Borland C++®, Microsoft Visual C++®
- Other programs needed
  - Editor
  - Linker/Loader

# Processing a Program

- Editor used to enter the program
  - Like minimal word processor
  - Creates **source** program file
- Compiler translates the source program
  - Displays syntax errors
  - Creates (usually) temporary **object code** file
- Linker/Loader to combine object file with other object files and execute program
  - Creates final **executable** program

# Figure 1.11 Entering, Translating, and Running a High-Level Language Program



# 1.5 Software Development Method

1. Problem Analysis
   - Identify data objects
   - Determine Input / Output data
   - Constraints on the problem
2. Design
   - Decompose into smaller problems
   - Top-down design (divide and conquer)
   - Develop Algorithm (Desk check)
     - Algorithm refinement

# Software Development Method

3. Implementation
   - Converting the algorithm into programming language
4. Testing
   - Verify the program meets requirements
   - System and Unit test
5. Maintenance
   - All programs undergo change over time

# 1.6 Applying the Software Development Method

- Case Study: Converting Miles to Kilometers
  - **Problem** Your summer surveying job requires you to study some maps that give distances in kilometers and some that use miles. You and your coworkers prefer to deal in metric measurements. Write a program that performs the necessary conversion.

# Data Requirements

- Problem Input

  miles *distance in miles*

- Problem Output

  kms *the distance in kilometers*

- Relevant Formula

  1 mile = 1.609 kilometers

# Design

- Formulate the algorithm that solves the problem.
- **Algorithm**
  1. Get the distance in miles.
  2. Convert the distance to kilometers.
  3. Display the distance in kilometers.
- Algorithm Refinement

  2.1 The distance in kilometers is 1.609 the distance in miles

# Listing 1.2 Miles to kilometers

```
// miles.cpp
// Converts distance in miles to kilometers.

#include <iostream>
using namespace std;

int main()                                  // start of main function
{
      const float KM_PER_MILE = 1.609;      // 1.609 km in a mile
      float miles,                          // input: distance in miles
            kms;                            // output: distance in kilometers

      // Get the distance in miles.
      cout << "Enter the distance in miles: ";
      cin >> miles;


      // Convert the distance to kilometers.
      kms = KM_PER_MILE * miles;

      // Display the distance in kilometers.
      cout << "The distance in kilometers is " << kms << endl;

      return 0;                                 //Exit the main function
}
```

```
Enter the distance in miles: 10.0
The distance in kilometers is 16.09
```

# Implementation

```
#include <iostream>
using namespace std;
int main( )
{
    const float KM_PER_MILE = 1.609;
    float miles, kms;
    cout << “Enter the distance in miles: “;
    cin >> miles;
    kms = KM_PER_MILE * miles;
    cout << “The distance in kilometers is “ << kms << endl;
    return 0;
}
```

# Testing

- Test with input data for which you can easily determine the expected results
- E.g.

  10 miles should convert to 16.09 kilometers

University of Sadat City
Faculty of Computers and Artificial Intelligence (FCAI)

# Fundamentals of Programming (CS101)

## Lecture 2

*Presented By:*

**Dr. Heba Askr**

**Second Term 2024-2025**

# Chapter 2:
# Overview of C++

## Problem Solving,
## Abstraction, and Design using C++ 6e

by Frank L. Friedman and Elliot B. Koffman

# 2.1 C++ Language Elements

- Comments
- Compiler directives
- Function **main**
- Declaration statements
- Executable statements

# Comments

- // symbols indicate a line comment - apply to just the rest of the line
- Block comments start with /* and end with */ - apply to as many lines as you like
- Used to describe the code in English or provide non-code information
- E.g. to include the name of the program or the author's name

# Listing 2.1 Converting miles to kilometers

```
// miles.cpp
// Converts distance in miles to kilometers.

#include <iostream>
using namespace std;

int main()          // start of main function
{

      const float KM_PER_MILE = 1.609;  // 1.609 km in a mile
      float miles,                      // input: distance in miles
            kms;                        // output: distance in kilometers

      // Get the distance in miles.
      cout << "Enter the distance in miles: ";
      cin >> miles;

      // Convert the distance to kilometers.

      kms = KM_PER_MILE * miles;

      // Display the distance in kilometers.

      cout << "The distance in kilometers is " << kms << endl;

      return 0;                         // Exit main function

}
```

# #include <*filename*>

- Compiler directive
- Includes previously written code from a **library** into your program
- E.g.

  #include <iostream>

  has operators for performing input and output within the program
- Libraries allow for code reuse

# using namespace std;

- Indicates to compiler that this program uses objects defined by a standard namespace called **std.** (the standard C++ library functions and objects, such as cout, cin, string, vector, etc.)>

- Without using namespace std;, you would have to prefix standard library components with std::.

  For example: std::cout << "Hello, World!" << std::endl;

- Ends with a semicolon
- Follows #include directives in the code
- Must appear in all programs

# Function **main**

```
int main ( )
{
        // function body
}
```

# Function **main**

- Exactly one **main** function per program
- A function is a collection of related statements that perform a specific operation
- **int** indicates the return type of the function
- **( )** indicates no special information passed to the function by the operating system

# Types of Statements

- **Declaration statements** - describe the data the function needs:

```
const float KM_PER_MILE = 1.609;
float miles,
      kms;
```

- **Executable statements** - specify the actions the program will take:

```
cout << "Enter the distance in miles: ";
cin >> miles;
```

# 2.2 Reserved Words (Keywords)

- Have special meaning in C++
- Cannot be used for other purposes

# **Table 2.1** Reserved words in Listing 2.1

| Reserved Words | Meaning |
|---|---|
| `const` | Constant; indicates a data element whose value cannot change |
| `float` | Floating point; indicates that a data item is a real number |
| `include` | Preprocessor directive; used to insert a library file |
| `int` | Integer; indicates that the main function returns an integer value |
| `namespace` | Region where program elements are defined |
| `return` | Causes a return from a function to the unit that activates it |
| `using` | Indicates that a program is using elements from a particular namespace |

# Identifiers

- Names for data and objects to be manipulated by the program
- Must begin with a letter or underscore (not recommended)
- Consist only of letters, digits, and underscores
- Cannot be reserved word
- Upper and lower case significant

# Identifiers

| Identifier | Use |
|---|---|
| cin | C++ name for standard input stream |
| cout | C++ name for standard output stream |
| km | Data element for storing distance in kilometers |
| KM_PER_MILE | Conversion constant |
| miles | Data element for storing distance in miles |
| std | C++ name for the standard namespace |

# 2.3 Data Types

- Defines a set of values and operations that can be performed on those values
- **integers**
  - positive and negative whole numbers,

    e.g. 5, -52, 343222
  - short, int, long

# Data Types (con't)

- **Floating point (real)**
  - number has two parts, integral and fractional
  - e.g. 2.5, 3.66666666, -.000034, 5.0
  - float, double, long double
  - stored internally in binary as **exponent**
  - 10.0 and 10 are stored differently in memory

# Data Types (con't)

- **Boolean**
  - named for George Boole
  - represent conditional values
  - values: true and false

# Data Types (con't)

- **Characters**
  - represent individual character values
    E.g. `'A' 'a' '2' '*' '"' ' '`
  - stored in 1 byte of memory
  - special characters: escape sequences
    E.g. `'\n' '\b' '\r' '\t' '\''`

# string Class

- Strings not built-in, but come from library
- Classes extend C++
- string literal enclosed in double quotes

  E.g.: “Enter speed: “ “ABC” “B” “true”
  “1234”
- #include <string>
  - for using string identifiers, but not needed for literals

# string Class Cont..

**For String Identifiers (Variables)**

If you declare string variables like:

```cpp
#include <string>
using namespace std;

int main() {
    string name = "John"; // Uses std::string
    return 0;
}
```

**For String Literals (Not Needed)**

If you are just using **string literals**, you **do not** need `<string>`.

Example:

```cpp
int main() {
    cout << "Hello, World!"; // "Hello, World!" is a string literal
    return 0;
}
```

# Variable Declarations

- Set aside memory with a specific name for the data and define its values and operations
- The value can change during execution
- *type identifier-list;*
- E.g.:

```
float x, y;
int me, you;
float week = 40.0;
string flower = “rose”;
```

# Constant Declarations

- Memory cells whose values cannot change once set
- const *type constant-identifier* = *value*;
- E.g.:

```
const  float  KM_PER_MILE = 1.609;
```

- Often identified by using all upper-case name

# **Listing 2.2** Printing a welcoming message

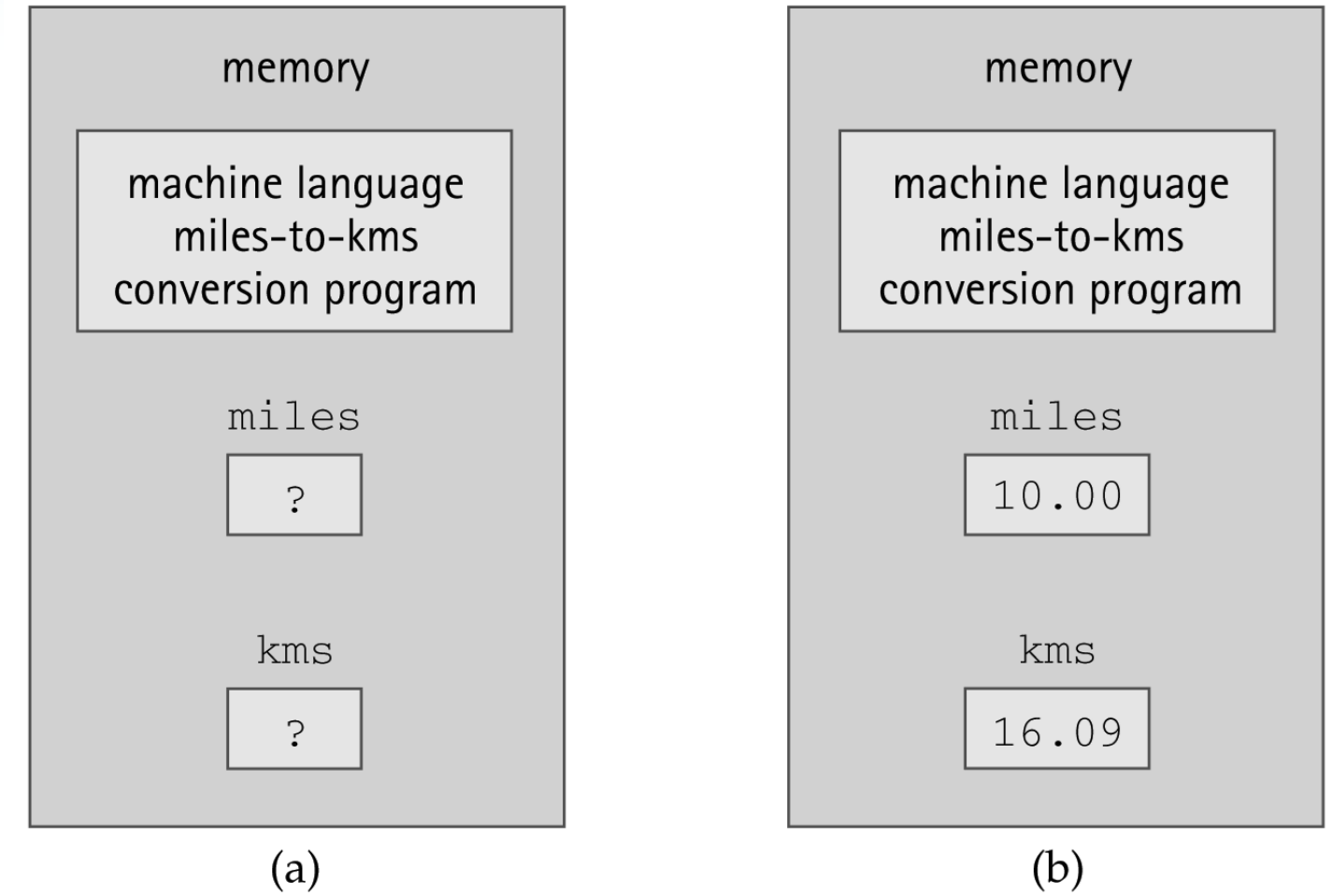
```
// File: hello.cpp
// Displays a user's name

#include <iostream>
#include <string>
using namespace std;

int main()
{
   char letter1, letter2;  // input and output: first two initials
   string lastName;        // input and output: last name

   // Enter letters and print message.
   cout << "Enter 2 initials and a last name: ";
   cin >> letter1 >> letter2 >> lastName;
   cout << "Hello " << letter1 << ". " << letter2 << ". "
        << lastName << "! ";
   cout << "We hope you enjoy studying C++." << endl;

   return 0;
}
```

```
Enter 2 initials and a last name: EB Koffman
Hello E. B. Koffman! We hope you enjoy studying C++.
```

# 2.4 Executable Statements

- Assignment statements
- Input statements
- Program output
- The **return** statement

# **Figure 2.2** Memory (a) before and (b) after execution of a program



# Assignment Statements

- *variable* = *expression*;

- E.g.:

```
kms = KM_PER_MILE * miles;
```

# Input Statements

- Obtain data for program to use - different each time program executes
- Standard stream library **iostream**
- **cin** - name of stream associated with standard input device (keyboard by default)
- **Extraction** operator (>>)
- E.g.:

```
cin >> miles;
cin >> age >> firstInitial;
```

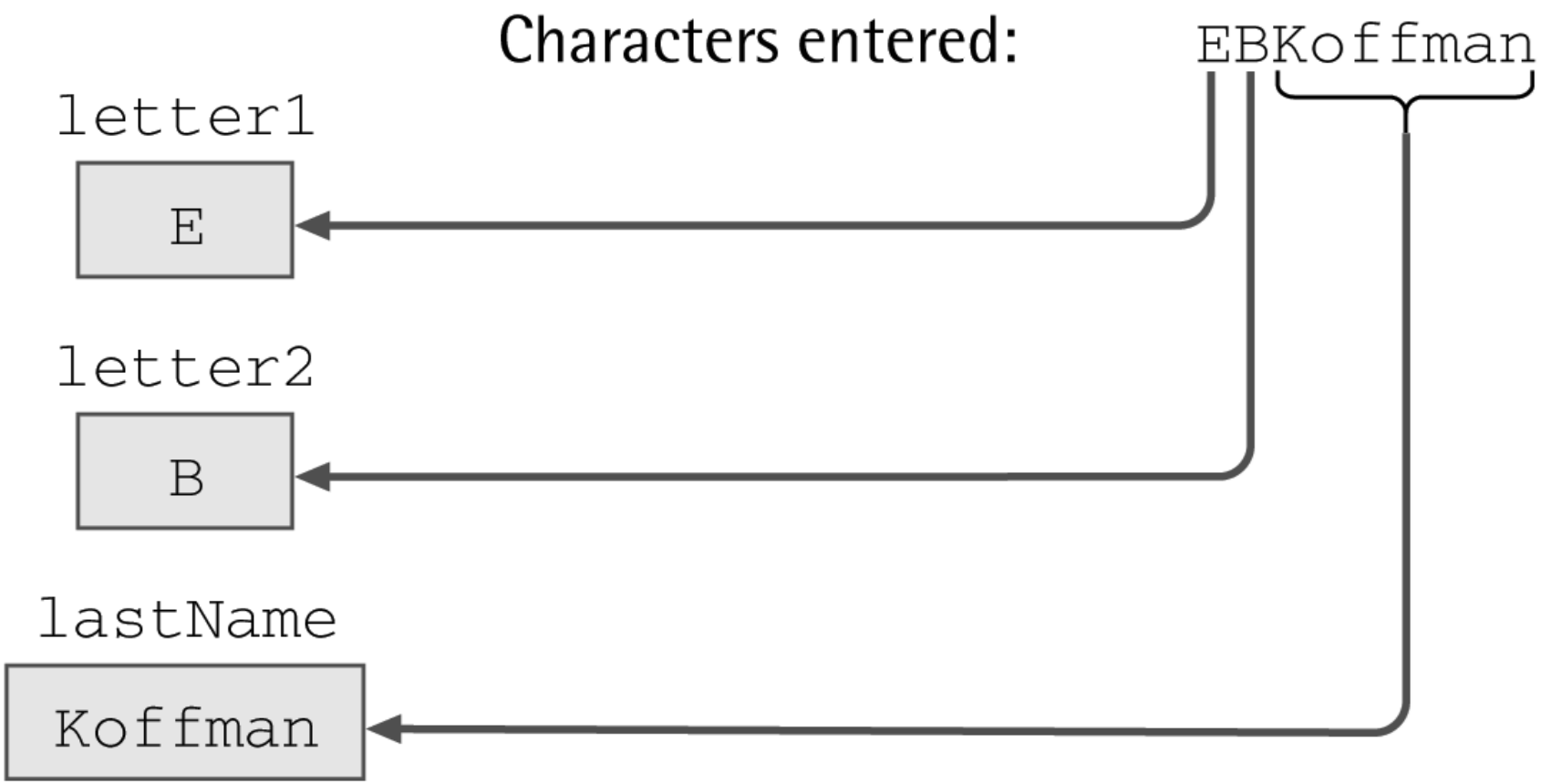
# In Listing 2.2:

```
cin >> letter1 >> letter2 >> lastname;
```

has the effect:

# Figure 2.6 Effect of

```
cin >> letter1 >> letter2 >> lastname;
```



# Program Output

- Used to display results of program
- Also standard stream library **iostream**
- **cout -** name of stream associated with standard output device (monitor by default)
- **Insertion** operator (<<) for each element

    cout << *data-element*;

# Program Output

- cout statement can be broken across lines
- Strings cannot be broken across lines
- Prompt messages used to inform program user to enter data
- Screen cursor is a moving marker indicating the position of where the next character will be displayed
- endl (or '\n') causes a new line in output

# Output Example

```
cout << "The distance in kilometers is "
     << kms << endl;
```

If variable kms has value 16.09, the output is:

The distance in kilometers is 16.09

# The **return** Statement

- Last line of main function is typically

  return 0;

- This transfers control from the program to the operating system, indicating that no error has occurred

# **Listing 2.3** General Form of a C++ Program

```
compiler directives
using namespace std;

int main()
{
   declaration statements
   executable statements
}
```

# Program Style

- **Use of spacing**
  - one statement per line
  - blanks after comma, around operators
  - in between some lines of code for readability
- **Use of comments**
  - header section
  - document algorithm steps
  - describe difficult code

# Program Style

- **Naming conventions for identifiers**
  - Variables in all lower case, with initial capital letter for additional words. No underscore.
  - Constants in all upper case, with underscores between words.

# 2.6 Arithmetic Expressions

- int data type

  + - * / %

- Integer division examples - result is integer

  15 / 3 = 5

  15 / 2 = 7

  0 / 15 = 0

  15 / 0 undefined

# Modulus for Integers

- Used *only* with integers
- Yields remainder - the result is integer
- Examples:

  7 % 2 = 1

  299 % 100 = 99

  49 % 5 = 4

  15 % 0 undefined

  15 % -7  system dependent

# Mixed-type Expressions

- E.g.: 4.6 / 2 evaluates to 2.3
- Rule: *when an integer and a floating point operand are combined by an operator, the integer gets converted to the floating point type*
- **Caveat**: this rule is dependent on operator precedence rules

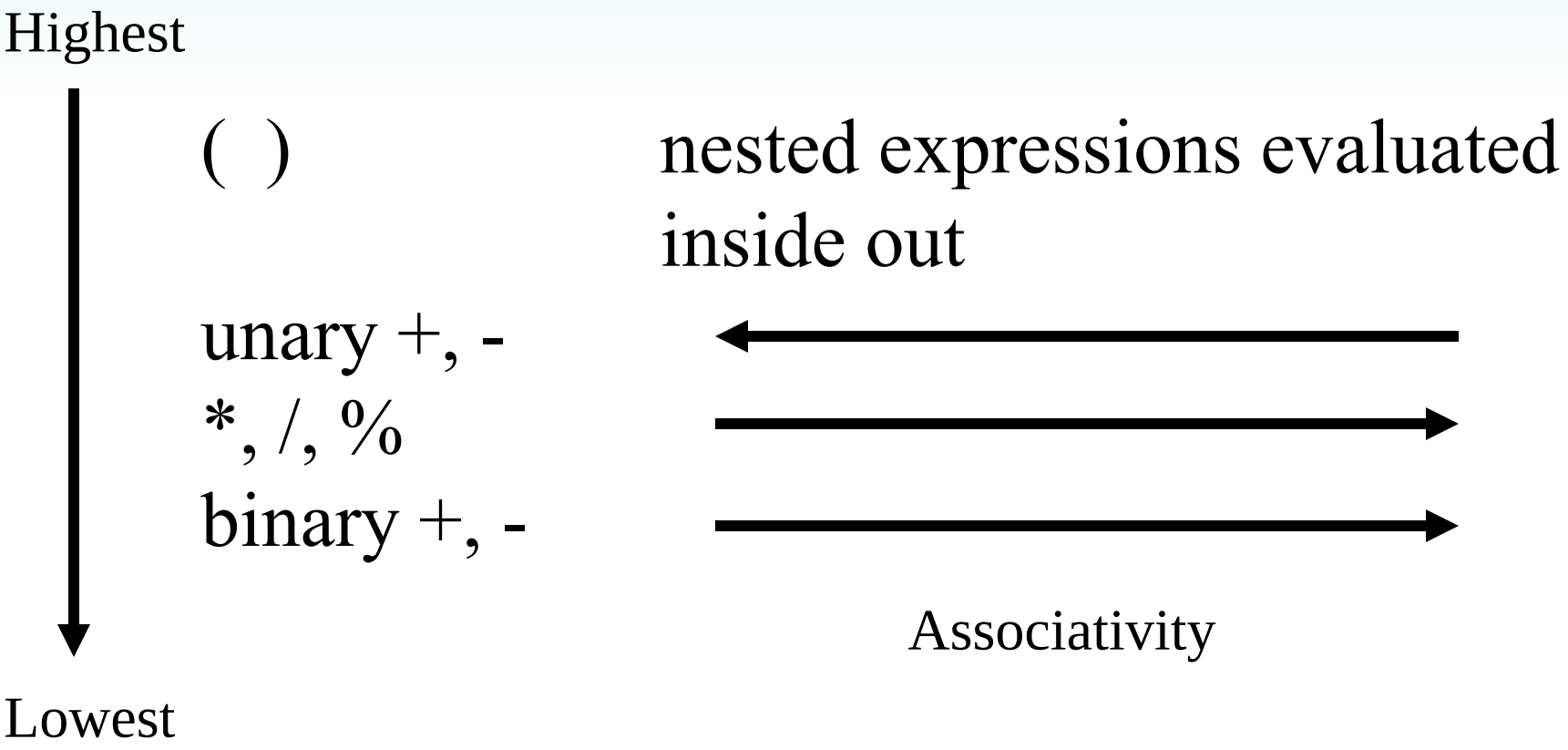
# Mixed-type Assignments

- If the variable on left side of assignment is of different type than the type of the evaluated expression on the right side of =, the result of the expression must be converted to the appropriate type

# Mixed-type Assignment Examples

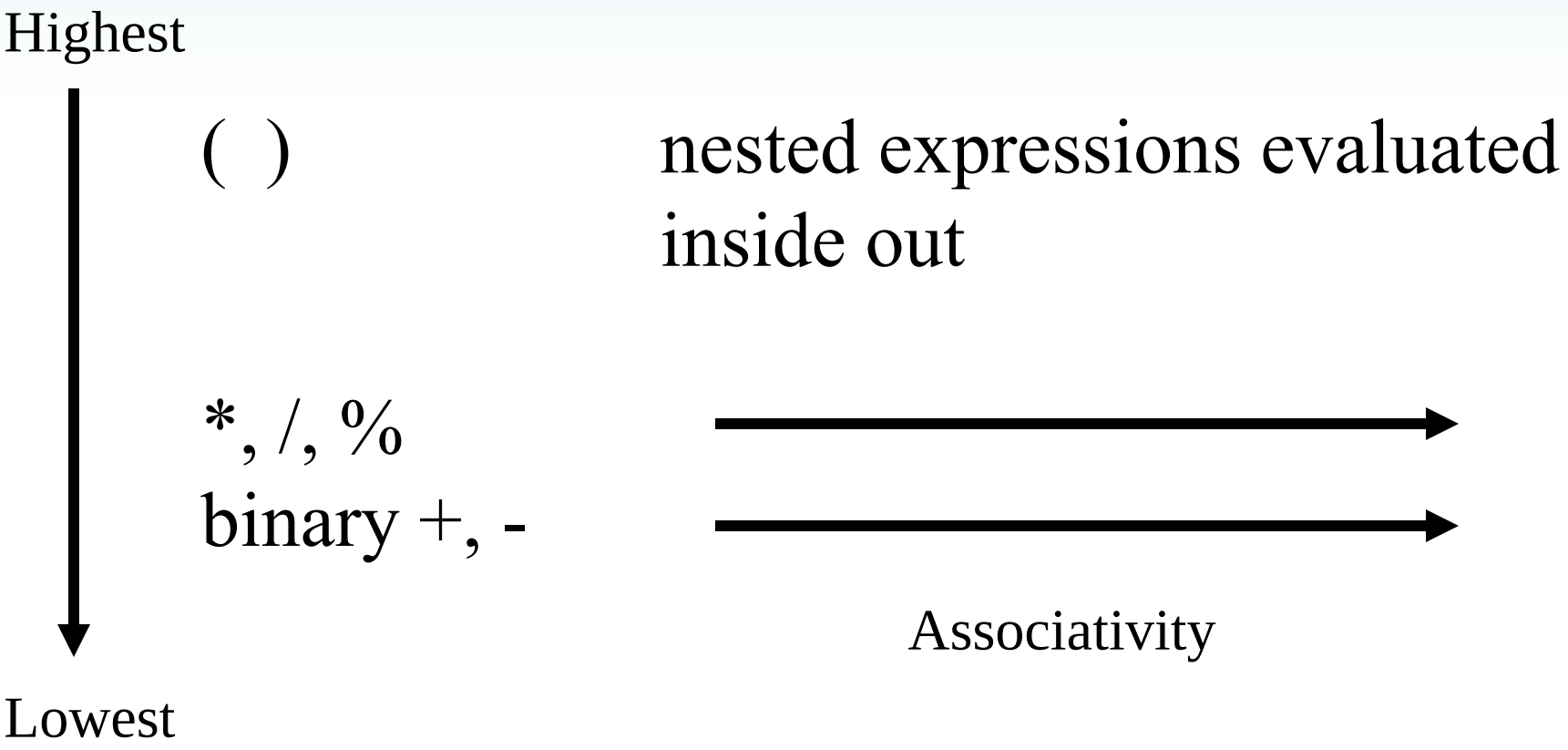
```
float a, b, x;
int m, n;
a=10;              // result is 10.0 stored in a
b = 3.5;
m=5;
n = 10;
x = m / n;         // result is 0 assigned to x
m = b * 3;         // result is 10 assigned to m
```

# Order of Operator Precedence




**Warning**: watch out for the types of operands and the type of the result from evaluating each operand!

# Order of Operator Precedence

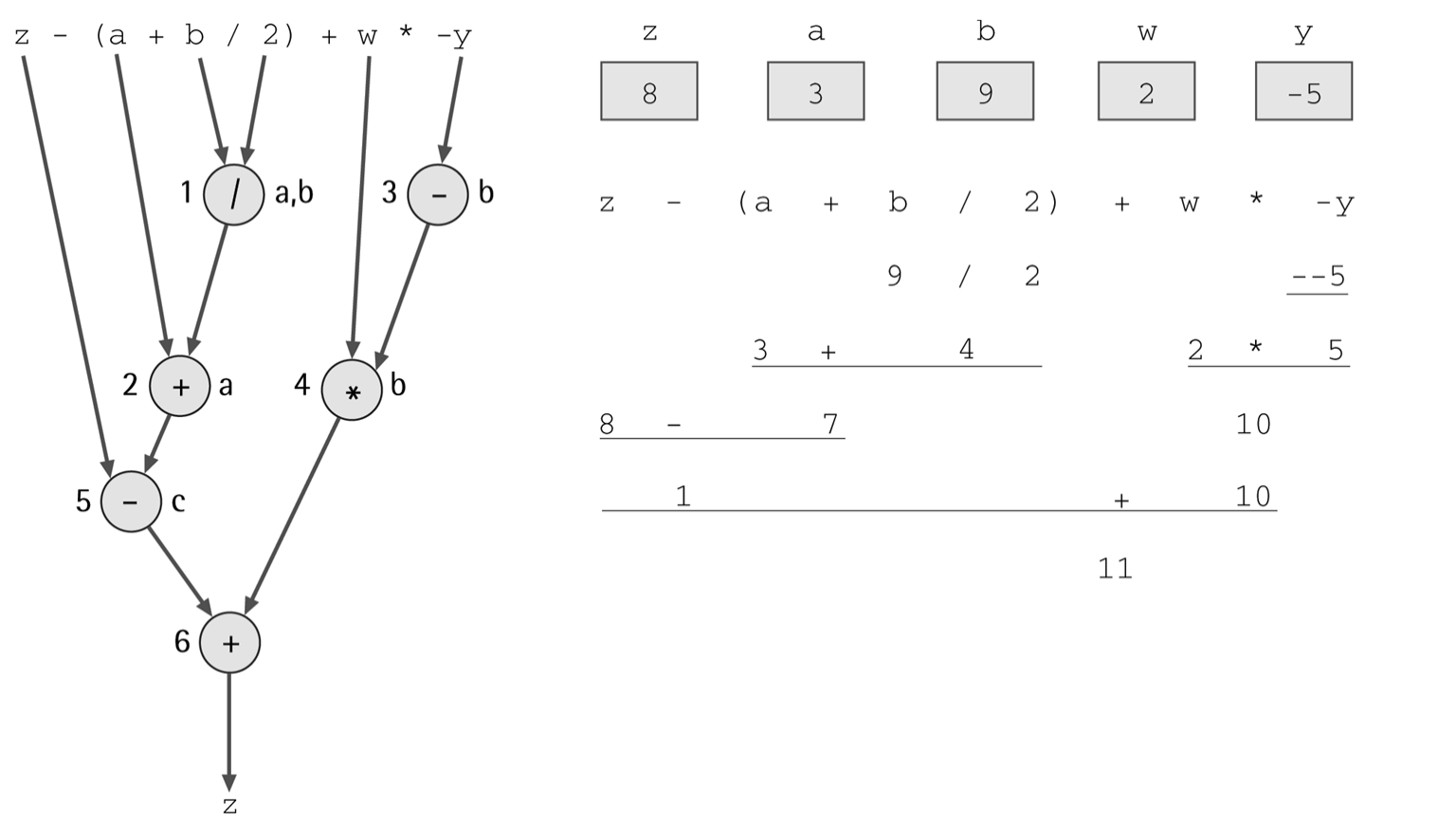



**Warning**: watch out for the types of operands and the type of the result from evaluating each operand!

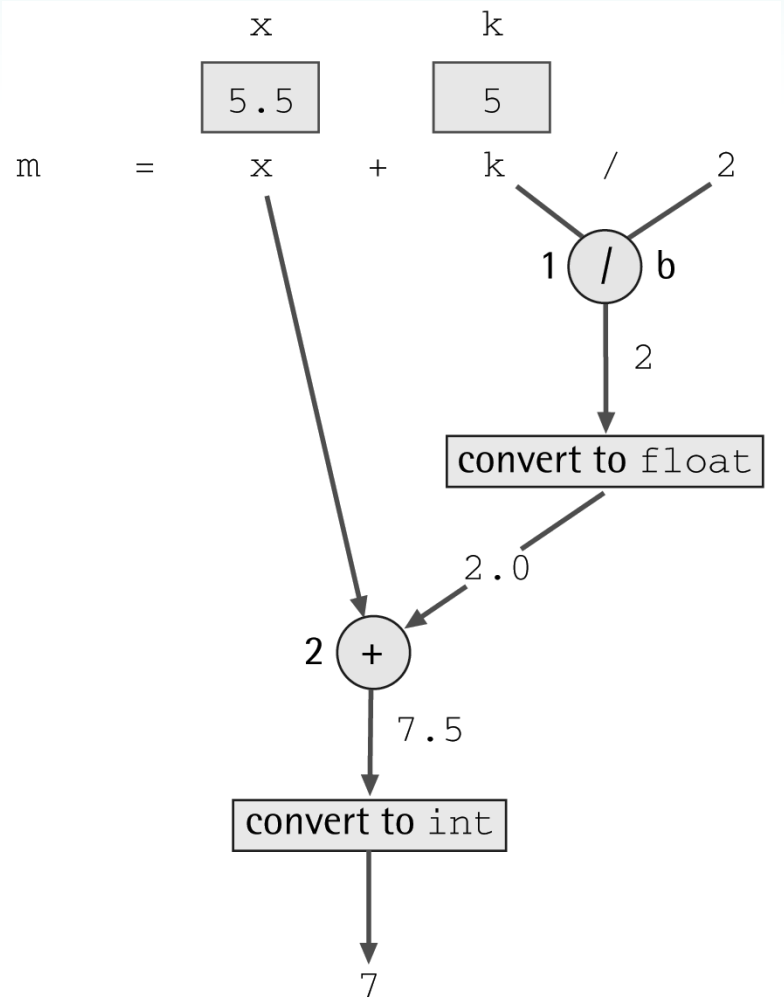
# Step-by-Step Expression Evaluation

```
area    =      PI    *   radius   *   radius
           3.14159         2.0          2.0
                  6.28318
                                12.56636
```

# Figure 2.10 Evaluation for z - (a +b / 2) + w * -y;



# Figure 2.11 Evaluation tree for m = x + k / 2:



# Mathematical Formulas in C++

- a = bc  *not valid C++ syntax*

  Must use * operator

  a = b * c;

- m = $\frac{y - b}{x - a}$

  Must use ( ) and /

  m = (y - b) / (x - a);

# Listing 2.4 Value of a coin collection

```
// File: coins.cpp
// Determines the value of a coin collection
#include <iostream>
#include <string>
using namespace std;
int main()
{
   string name;      // input: niece's first name
   int pennies;      // input: count of pennies
   int nickels;      // input: count of nickels
   int dollars;      // output: value of coins in dollars
   int change;       // output: value of coins in cents
   int totalCents;   // total cents represented
```

# Listing 2.4 Value of a coin collection (continued)

```
// Read in your niece's first name.
   cout << "Enter your first name: ";
   cin >> name;

// Read in the count of nickels and pennies.
   cout << "Enter the number of nickels: ";
   cin >> nickels;
   cout << "Enter the number of pennies: ";
   cin >> pennies;
```

# Listing 2.4 Value of a coin collection (continued)

```
// Compute the total value in cents.
   totalCents = 5 * nickels + pennies;

   // Find the value in dollars and change.
   dollars = totalCents / 100;    // integer division
   change = totalCents % 100;

// Display the value in dollars and change.
   cout << "Good work " << name << '!' << endl;
   cout << "Your collection is worth "
        << dollars << " dollars and "
        << change << " cents." << endl;
   return 0;
}
```

# coins.cpp Sample Execution

Enter your first name: Sally

Enter the number of nickels: 30

Enter the number of pennies: 77

Good work Sally!

Your collection is worth 2 dollars and 27 cents.

# 2.7 Interactive Mode, Batch Mode, and Data Files

- **Interactive mode** - input from user via keyboard
- **Batch mode** - input via a file, no user interaction
- Input/output redirection can be used for batch mode

# Input Redirection

- Requires a file already containing all input data before the program is executed
- At the time the program is executed, the input file is specified
- E.g. in UNIX

  metric < mydata

- Executes the program metric using the file mydata for input
- Echo printing often used with batch input

# Output Redirection

- Sends ALL output to a file instead of to the display monitor
- Not typically used with interactive input mode, since even prompt messages will be sent to the output file
- E.g. in UNIX (not typical)

  metric > myoutput

- Both input AND output redirection

  metric < mydata > myoutput

# 2.8 Common Programming Errors

- Syntax
  - a grammatical error in the formation of a program statement
  - detected by the compiler
  - prevents translation of source code into object code, so no execution possible
  - messages are compiler dependent, so you must learn how *your* compiler identifiers errors

# Miles-to-kms program with syntax error – missing “

```
// Miles.cpp
// Converts distance in miles to kilometers.

#include <iostream>     // class for stream input/output
using namespace std;    // use the standard namespace

int main()               // start of main function
{
      const float km_per_mile = 1.609;  // 1.609 km in a mile
      float miles,       // input: distance in miles
            kms;    // output: distance in kilometers

      // Get the distance in miles.
      cout << "Enter the distance in miles: ;  // missing quote
      cin >> miles;

      // Convert the distance to kilometers.
      kms = km_per_mile * miles;

      // Display the distance in kilometers.
      cout << "The distance in kilometers is " << kms << endl;

      return 0;
}
```

# Syntax Error display

```
[C++ Error] miles.cpp(12): E2141 Declaration syntax error.
[C++ Error] miles.cpp(15): E2380 Unterminated string or
character constant.
[C++ Error] miles.cpp(16): E2379 Statement missing ;.
[C++ Error] miles.cpp(19): E2451 Undefined symbol 'kms'.
[C++ Warning] miles.cpp(25): W8080 'miles' is declared but
never used.
[C++ Warning] miles.cpp(25): W8004 'KM_PER_MILE' is
assigned a value that is never used.
```

# Common Programming Errors

- **Run-time errors**
  - detected and displayed during execution of a program
  - usually fatal - halts execution of code

# Common Programming Errors

- **Undetected errors**
  - program runs to completion, but results are incorrect
  - often the result of input of the wrong type being entered

# Common Programming Errors

- **Logic errors**
  - caused by a faulty algorithm
  - often difficult to locate
  - can result in either a run-time or undetected error
  - system cannot identify - up to programmer to locate
  - vital to verify program output to ensure correct results


University of Sadat City
Faculty of Computers and Artificial Intelligence (FCAI)


# Fundamentals of Programming (CS101)

## Lecture 3

***Presented By:***

**Dr. Heba Askr**

**Second Term 2024-2025**

# Chapter 3:
# Top-Down Design with Functions and Classes

**Problem Solving,**
**Abstraction, and Design using C++ 6e**

**by Frank L. Friedman and Elliot B. Koffman**

# 3.1 Building Programs with Existing Information

- Analysis and design phases provide much information to help plan and complete a program
- Can start with data requirements to develop constant and variable declarations
- Use the **algorithm** as a first step in coding executable statements

# *Case Study*: Finding the Area and Circumference of a Circle

- Problem statement

  Get the radius of a circle. Compute and display the circle's area and circumference.
- Analysis
  - input is the circle's radius
  - need calculation for the area
  - need calculation for the circumference

# *Case Study*: Data Requirements

- Problem Constant

  PI = 3.14159

- Problem input

  float radius  // radius of a circle

- Problem output

  float area  // area of a circle

  float circum  // circumference of a circle

# *Case Study*: Formulas

- Area of a circle = $\pi \times \text{radius}^2$
- Circumference of a circle = $2 \times \pi \times \text{radius}$

# *Case Study*: Design - Algorithm

1. Get the circle radius
2. Compute the area of circle
3. Compute the circumference of circle
4. Display area and circumference

# *Case Study*: Design - Algorithm

1. Get the circle radius
2. Compute the area of circle
   2.1 Assign PI * radius * radius to area
3. Compute the circumference of circle
   3.1 Assign 2 * PI * radius to circum
4. Display area and circumference

# **Listing 3.2** Outline of area and circumference program

```
// Computes and displays the area and circumference of a circle

int main()
{
   const float PI = 3.14159;
   float radius;           // input: radius of circle
   float area;             // output: area of circle
   float circum;           // output: circumference of circle

   // Get the circle radius.

   // Compute area of circle.
       // Assign PI * radius * radius to area.

   // Compute circumference of circle.
       // Assign 2 * PI * radius to circum.

   // Display area and circumference.

   return 0;
}
```

# **Listing 3.3** Finding the area and circumference of a circle

```
// File: circle.cpp
// Computes and displays the area and circumference of a circle

#include <iostream>
using namespace std;

int main()
{
   const float PI = 3.14159;
   float radius;           // input: radius of circle
   float area;             // output: area of circle
   float circum;           // output: circumference of circle

   // Get the circle radius.
   cout << "Enter the circle radius: ";
   cin >> radius;
```

# **Listing 3.3** Finding the area and circumference of a circle (continued)

```
   // Compute area of circle.
   area = PI * radius * radius;

   // Compute circumference of circle.
   circum = 2 * PI * radius;

   // Display area and circumference.
   cout << "The area of the circle is " << area << endl;
   cout << "The circumference of the circle is " << circum << endl;

   return 0;
}
```

```
Enter the circle radius: 5.0
The area of the circle is 78.539749
The circumference of the circle is 31.415901
```

# *Case Study*: Testing

- Radius of 5.0
- Should get area of 78.539…
- Should get circumference of 31.415...

# *Case Study*: Weight of Flat Washers

- Problem statement

  You work for a hardware company that manufactures flat washers. To estimate shipping costs, you company needs a program that computes the weight of a specified quantity of flat washers.

# *Case Study*: Weight of Flat Washers

- Analysis
  - flat washer is like a small donut
  - need to know rim area, thickness, density
  - rim area will be computed from knowing the washer's outer and inner diameters.

# *Case Study*: Data Requirements

- Problem Constant

  PI = 3.14159

- Problem inputs

```
float holeDiameter      // diameter of hole
float edgeDiameter      // diameter of outer edge
float thickness         // thickness of washer
float density           // density of material used
float quantity          // number of washers made
```

# *Case Study*: Data Requirements

- Problem output

```
float weight      // weight of batch of washers
```

- Program Variables

```
float holeRadius      // radius of hole
float edgeRadius      // radius of outer edge
float rimArea         // area of rim
float unitWeight      // weight of 1 washer
```

# *Case Study*: Formulas

- Area of circle = $\pi \times \text{radius}^2$
- Radius of circle = diameter / 2
- Rim area = area of outer circle - area of hole
- Unit weight = rim area × thickness × density

# Listing 3.4

Washer program

```
// File: washers.cpp
// Computes the weight of a batch of flat washers.

#include <iostream>
using namespace std;

int main()
{
      const float PI = 3.14159;
      float holeDiameter;  // input - diameter of hole
      float edgeDiameter;  // input - diameter of outer edge
      float thickness;     // input - thickness of washer
      float density;       // input - density of material used
      float quantity;      // input - number of washers made
      float weight;        // output - weight of washer batch
      float holeRadius;    // radius of hole
      float edgeRadius;    // radius of outer edge
      float rimArea;       // area of rim
      float unitWeight;    // weight of 1 washer

      // Get the inner diameter, outer diameter, and thickness.
      cout << "Inner diameter in centimeters: ";
      cin >> holeDiameter;
      cout << "Outer diameter in centimeters: ";
      cin >> edgeDiameter;
      cout << "Thickness in centimeters: ";
      cin >> thickness;

      // Get the material density and quantity manufactured.
      cout << "Material density in grams per cubic centimeter: ";
      cin >> density;
      cout << "Quantity in batch: ";
      cin >> quantity;
```

# Listing 3.4 Washer program (continued)

```
    // Compute the rim area.
    holeRadius = holeDiameter / 2.0;
    edgeRadius = edgeDiameter / 2.0;
    rimArea =  PI * edgeRadius * edgeRadius -
               PI * holeRadius * holeRadius;

    // Compute the weight of a flat washer.
    unitWeight = rimArea * thickness * density;

    // Compute the weight of the batch of washers.
    weight = unitWeight * quantity;

        // Display the weight of the batch of washers.
        cout << "The expected weight of the batch is "
             << weight << " grams." << endl;
        return 0;
}
```

```
Inner diameter in centimeters: 1.2
Outer diameter in centimeters: 2.4
Thickness in centimeters: 0.1
Material density in grams per cubic centimeter: 7.87
Quantity in batch: 1000
The expected weight of the batch is 2670.23 grams.
```

# *Case Study*: Testing

Input data

- inner diameter of 1.2
- outer diameter of 2.4
- thickness of 0.1
- material density of 7.87
- quantity in batch of 1000

Should produce

- expected weight of batch of 2670.23 grams

# 3.2 Library Functions

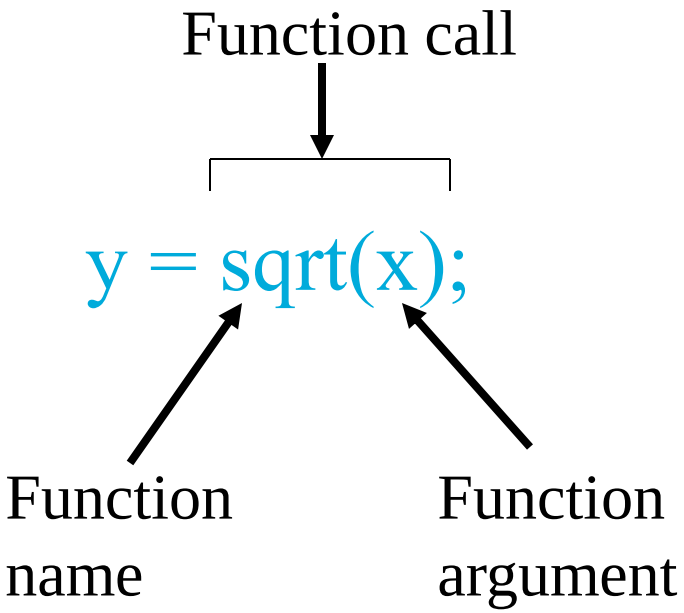
- **Goals of software engineering**
  - reliable code
  - accomplished by code reuse

- C++ promotes code reuse with predefined classes and functions in the standard library

# C++ cmath Library

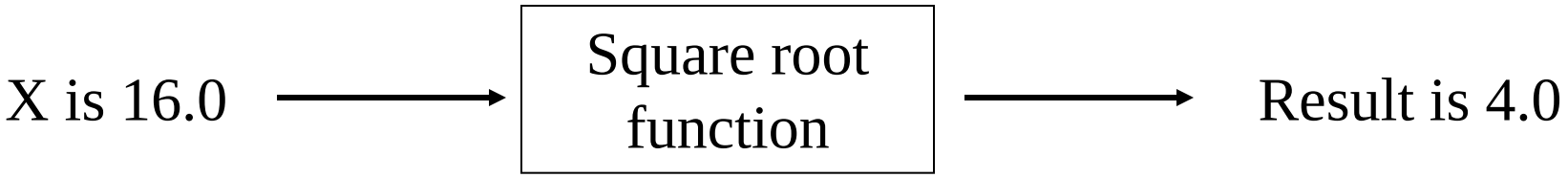
- Typical mathematical functions

  e.g. sqrt, sin, cos, log
- Function use in an assignment statement



# Example: sqrt Function

**Function sqrt as a "black box"**



# **Listing 3.5** Illustration of the use of the C++ **`sqrt`** function

```
// File: squareRoot.cpp
// Performs three square root computations

#include <cmath>                // sqrt function
#include <iostream>             // i/o functions
using namespace std;

int main()
{
   float first;              // input: one of two data values
   float second;             // input: second of two data values
   float answer;             // output: a square root value

   // Get first number and display its square root.
   cout << "Enter the first number: ";
   cin >> first;
   answer = sqrt(first);
   cout << "The square root of the first number is "
        << answer << endl;
```

# **Listing 3.5** Illustration of the use of the C++ **`sqrt`** function (continued)

```
    // Get second number and display its square root.
    cout << "Enter the second number: ";
    cin >> second;
    answer = sqrt(second);
    cout << "The square root of the second number is "
         << answer << endl;

    // Display the square root of the sum of first and second.
    answer = sqrt(first + second);
    cout << "The square root of the sum of both numbers is "
         << answer << endl;
    return 0;
}
```

```
Enter the first number: 9
The square root of the first number is 3
Enter the second number: 25
The square root of the second number is 5
The square root of the sum of both numbers is 5.83095
```

# Table 3.1 Some Mathematical Library Functions

| Function | Standard Library | Purpose: Example | Argument(s) | Result |
|---|---|---|---|---|
| `abs(x)` | `<cstdlib>` | Returns the absolute value of its integer argument: if `x` is `-5`, `abs(x)` is `5` | `int` | `int` |
| `ceil(x)` | `<cmath>` | Returns the smallest integral value that is not less than `x`: if `x` is `45.23`, `ceil(x)` is `46.0` | `double` | `double` |
| `cos(x)` | `<cmath>` | Returns the cosine of angle `x`: if `x` is `0.0`, `cos(x)` is `1.0` | `double` (radians) | `double` |
| `exp(x)` | `<cmath>` | Returns $e^x$ where $e$ = `2.71828...`: if `x` is `1.0`, `exp(x)` is `2.71828` | `double` | `double` |
| `fabs(x)` | `<cmath>` | Returns the absolute value of its type `double` argument: if `x` is `-8.432`, `fabs(x)` is `8.432` | `double` | `double` |
| `floor(x)` | `<cmath>` | Returns the largest integral value that is not greater than `x`: if `x` is `45.23`, `floor(x)` is `45.0` | `double` | `double` |

# Table 3.1 Some Mathematical Library Functions (continued)

| | | | | |
|---|---|---|---|---|
| `log(x)` | `<cmath>` | Returns the natural logarithm of `x` for `x > 0.0`: if `x` is `2.71828`, `log(x)` is `1.0` | `double` | `double` |
| `log10(x)` | `<cmath>` | Returns the base-10 logarithm of `x` for `x > 0.0`: if `x` is `100.0`, `log10(x)` is `2.0` | `double` | `double` |
| `pow(x, y)` | `<cmath>` | Returns $x^y$. If `x` is negative, `y` must be integral: if `x` is `0.16` and `y` is `0.5`, `pow(x, y)` is `0.4` | `double, double` | `double` |
| `sin(x)` | `<cmath>` | Returns the sine of angle `x`: if `x` is `1.5708`, `sin(x)` is `1.0` | `double` (radians) | `double` |
| `sqrt(x)` | `<cmath>` | Returns the non-negative square root of `x` ($\sqrt{x}$) for `x ≥ 0.0`: if `x` is `2.25`, `sqrt(x)` is `1.5` | `double` | `double` |
| `tan(x)` | `<cmath>` | Returns the tangent of angle `x`: if `x` is `0.0`, `tan(x)` is `0.0` | `double` (radians) | `double` |

# Looking Ahead

- **Could write own functions**
  - findArea( r ) returns area of circle of radius r
  - findCircum( r ) returns circumference of circle of radius r
- Program to compute area and circumference

```
area = findArea(radius);
circum = findCircum(radius);
```

- Washers program

```
rimArea = findArea(edgeRadius) -
                findArea(holeRadius);
```

# 3.3 Top-Down Design and Structure Charts

- **Top-down design**
  - process to break down complex problem into smaller, simpler subproblems
  - similar to development of an algorithm
- **Structure chart**
  - graphical representation of relationship of subproblems

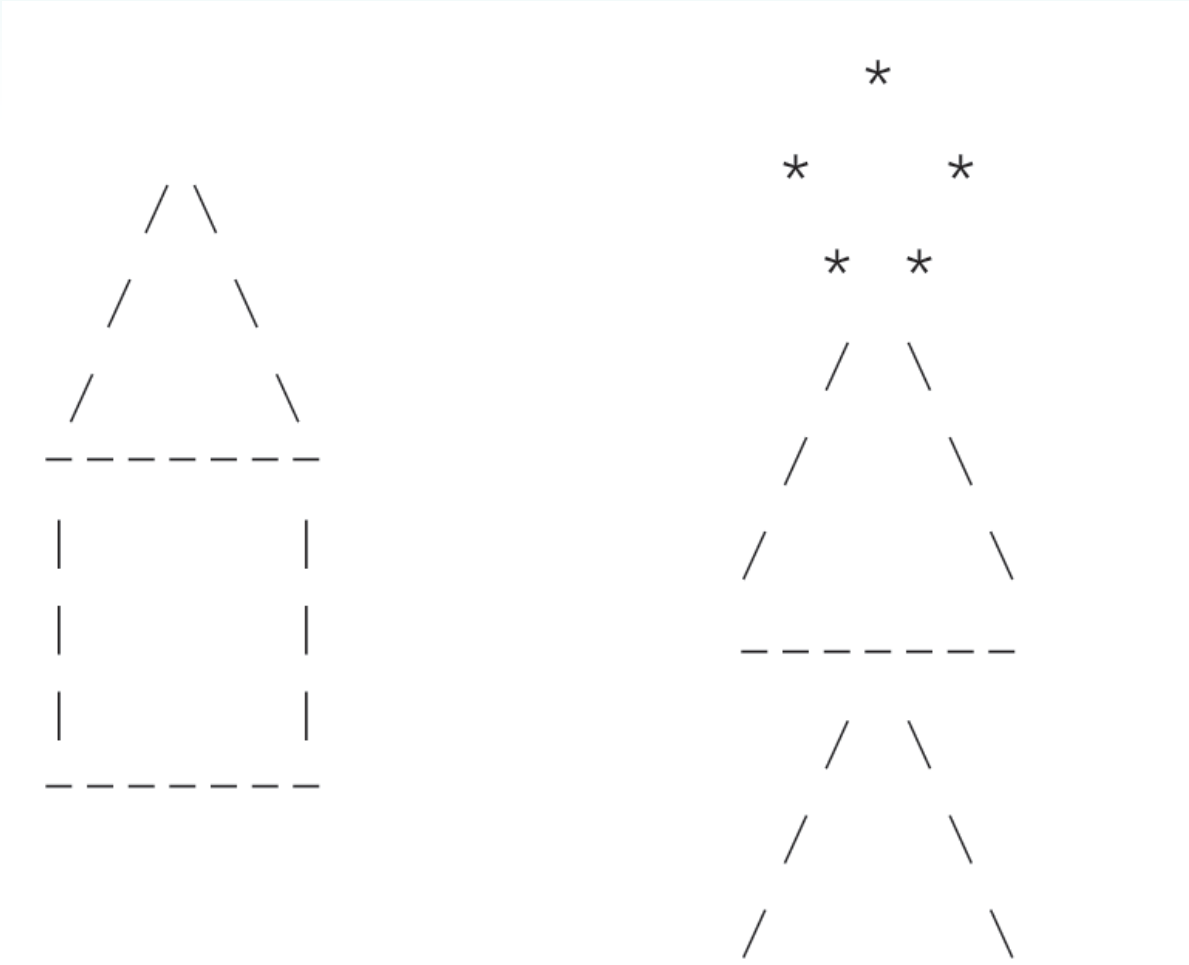
# *Case Study*: Simple Figures

- Problem statement

  Draw some simple diagrams on the screen, e.g. a house and a female stick figure.

- Analysis
  - house is formed by displaying a triangle without its base, on top of a rectangle
  - stick figure consists of a circular shape, a triangle, and a triangle without its base.
  - 4 basic shapes: circle, base line, parallel lines, intersecting lines

# Figure 3.4 House and stick figure



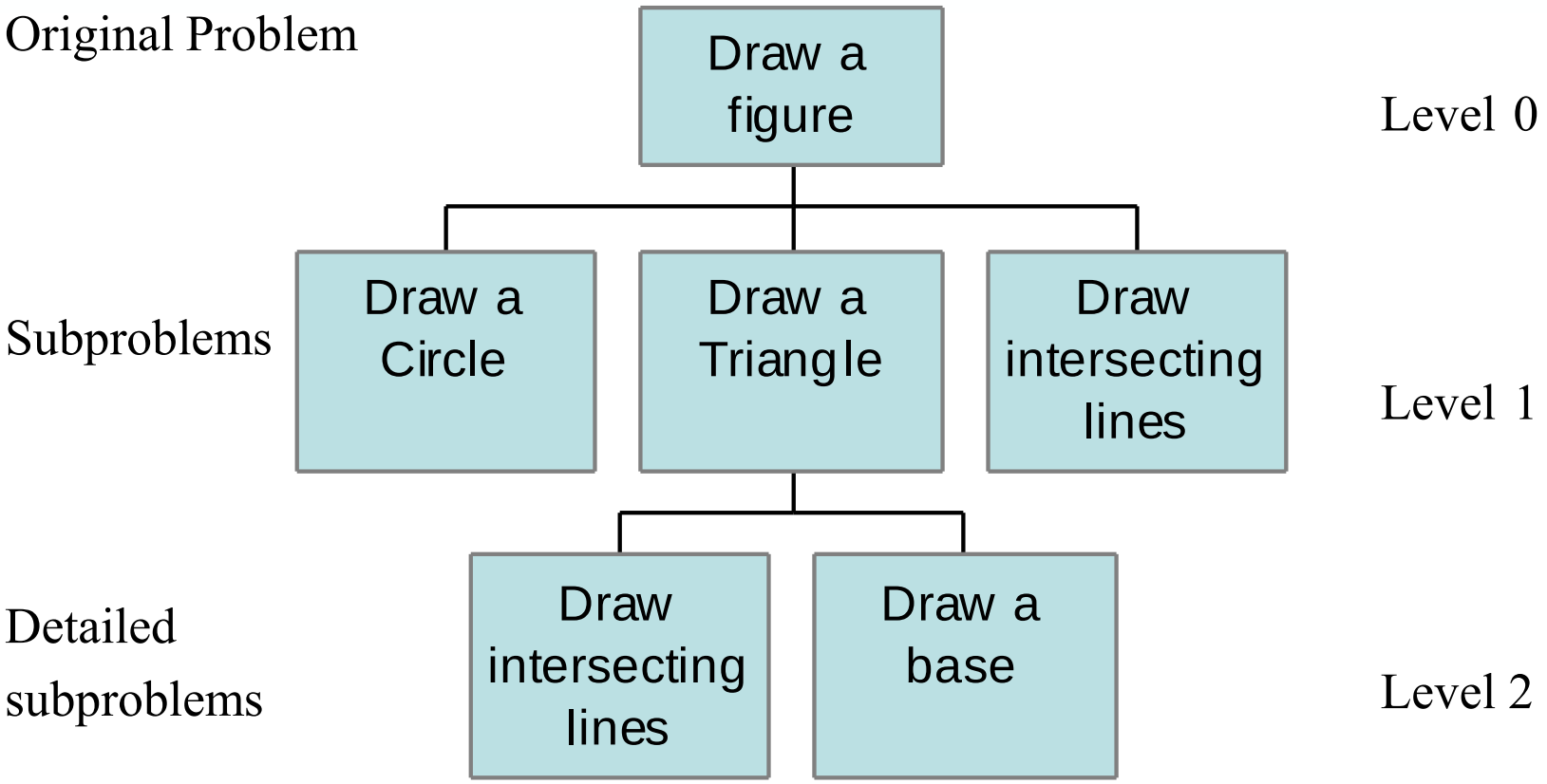
# *Case Study*: Design - Algorithm

- (no real data involved, so skip Data Requirements)
- For stick figure:
  1. Draw a circle.
  2. Draw a triangle.
  3. Draw intersecting lines.

# *Case Study*: Design - Algorithm

- (no real data involved, so skip Data Requirements)
- For stick figure:
  1. Draw a circle.
  2. Draw a triangle.
     - 2.1 Draw intersecting lines.
     - 2.2 Draw a base line.
  3. Draw intersecting lines.

# *Case Study*: Structure Chart



# 3.4 Functions without Arguments

- Functions important part of top-down design
- main( ) is a function called by the OS
- Form of call: *fname( );*
- Example: drawCircle( );
- Interpretation: the function *fname* is activated. After *fname* finishes execution, the program statement that follows the function call will be executed next.

# Some Notes on Functions

- Don't need to know details about how a function is implemented to know how to call.
- E.g.

```
y = sqrt(x);      // don't know how sqrt implemented
```

- *Do* know how function is used (called).
- Empty parentheses indicate no arguments (more on this later).

# Function Prototype

- Declares the function to the compiler
- Appears before function `main`.
- Tells compiler the function's type, its name, and information about arguments.

# Function Prototype (no Arguments)

- Form: *ftype fname( );*
- Example: void skipThree( );
- Interpretation: identifier *fname* is declared to be the name of a function. The identifier *ftype* specifies the data type of the function result. *Ftype* of void indicates the function does not return a value.

# Function Definitions

- Specifies the function's operations
- **Function header** similar to function prototype
- **Function body** contains declarations for **local variables** and constants and executable statements
- Prototypes precede `main` function (after #include directives)
- Function definitions follow `main` function

# Function Definition (no arguments)

- Syntax: *ftype fname( )*

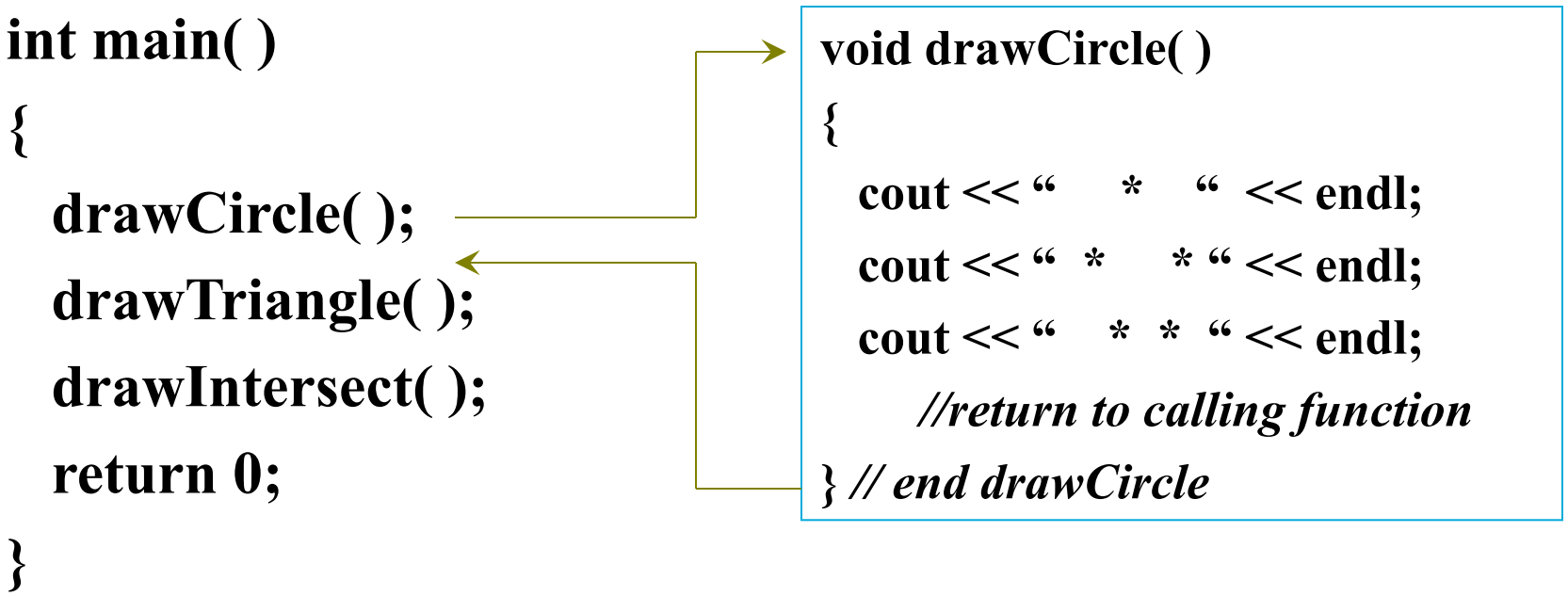
```
{
        local declarations
        executable statements
}
```

# Function Definition (no arguments)

- Example:

```
// Displays block-letter H
void printH( )
{
  cout << "**  **" << endl;
  cout << "**  **" << endl;
  cout << "******" << endl;
  cout << "******" << endl;
  cout << "**  **" << endl;
  cout << "**  **" << endl;
}     // end printH
```

# Order of Execution



# Notes on Function Execution

- Execution always begins at first statement of `main` function
- When a function is called
  - space is set aside for function variables
  - statements of function are executed
  - control returns to statement following call
  - function ends with space released

# Function Advantages

- Team assignments on large project
- Simplify tasks
- Each function is a separate unit
- Top-down approach
- Reuse (e.g. drawTriangle)
- **Procedural abstraction**
- **Information hiding**

# Displaying User Instructions

- Simple functions (no arguments, no return value) are of limited use
- Useful for displaying instructions to user
- E.g.

```
void instruct( );          // prototype
…
instruct( );               //function call
```

# Figure 3.10 Function **instruct**

```
// Displays instructions to user of area/circumference program

void instruct ()
{
   cout << "This program computes the area and " << endl;
   cout << "circumference of a circle. " << endl << endl;
   cout << "To use this program, enter the radius of the "
        << endl;
    cout << "circle after the prompt" << endl;
    cout << "Enter the circle radius: " << endl << endl;
    cout << "The circumference will be computed in the same"
        << endl;
    cout << "units of measurement as the radius. The area "
        << endl;
    cout << "will be computed in the same units squared."
        << endl << endl;
}
```

# Functions with Input Arguments

- Arguments make functions versatile
- E.g.:

```
rimArea = findArea(edgeRadius) -
                    findArea(holeRadius);
```

# `void` Functions with Input Arguments

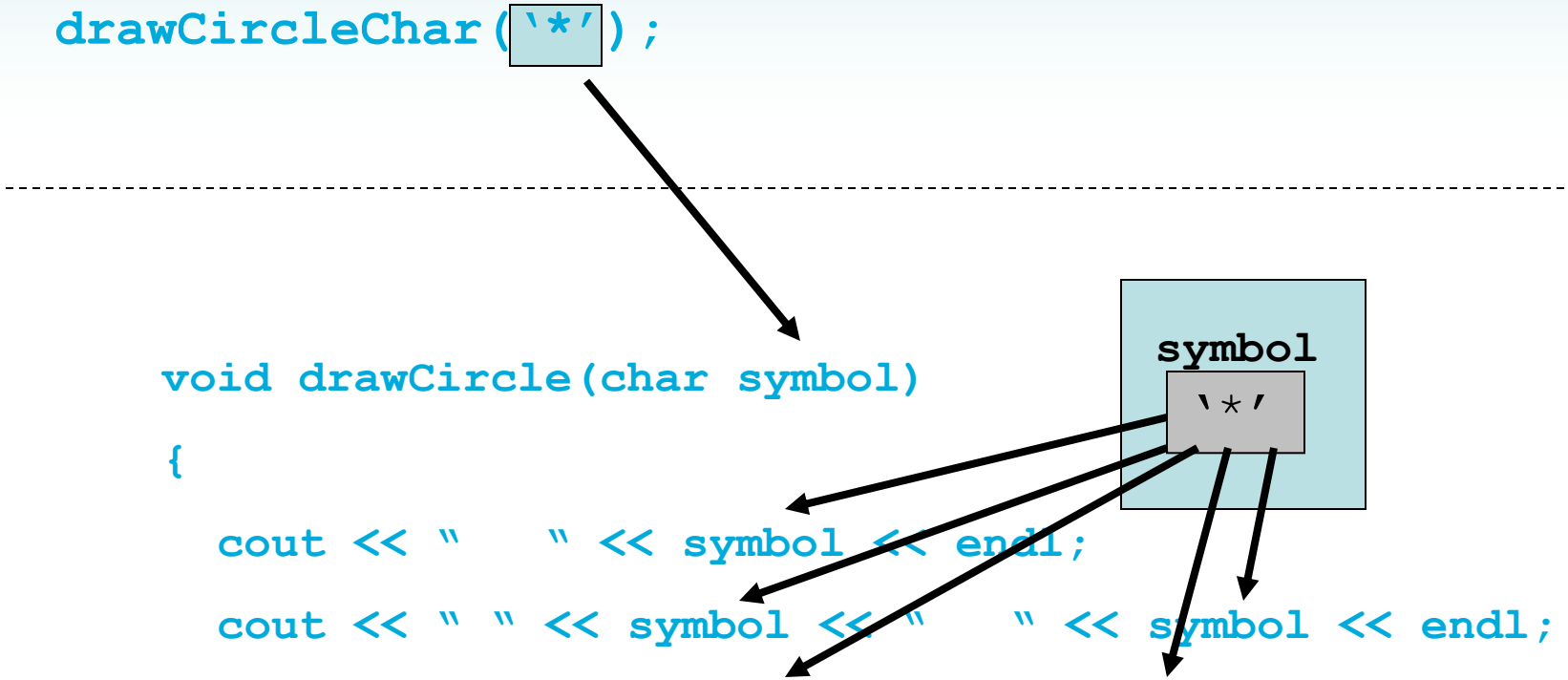
- Give data to function to use
- Don't expect function to return any result(s)
- Call format:

  ***fname (actual-argument-list);***

- E.g.:

```
drawCircleChar('*');
```

```
drawCircleChar('*');
```




```
void drawCircle(char symbol)
{
  cout << "   " << symbol << endl;
  cout << " " << symbol << "   " << symbol << endl;
  cout << "  " << symbol << " " << symbol << endl;
} // end drawCircle
```

# Functions with Arguments and a Single Result

- Functions that return a result must have a return statement:

Form: ***return expression;***

Example: return x * y;

```
#include <iostream>
#include <cmath>
using namespace std;
const float PI = 3.14159;
float findCircum(float);
float findArea(float);
int main( )
{
  float radius = 10.0;
  float circum;
  float area;
  circum = findCircum(radius);
  area = findArea(radius);
  cout << "Area is " << area << endl;
  cout << "Circumference is " << circum << endl;
  return 0;
}
```

Function declaration or prototype

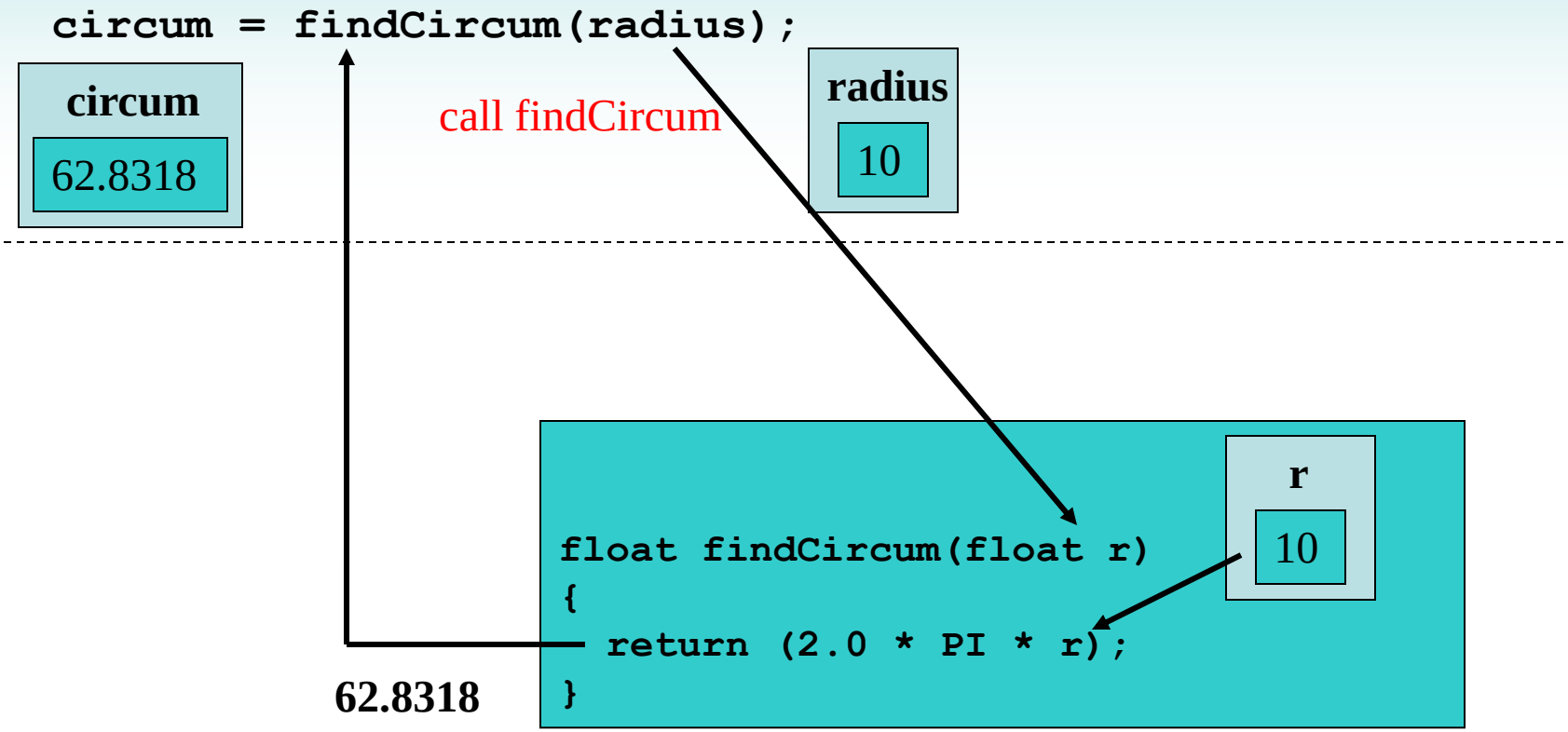
Function calling

# Figure 3.12 Functions `findCircum` and `findArea`

```
// Computes the circumference of a circle with radius r
// Pre: r is defined and is > 0.
//        PI is a constant.
// Post: returns circumference
float findCircum(float r)
{
  return (2.0 * PI * r);
}
// Computes the area of a circle with radius r
// Pre: r is defined and is > 0.
//        PI is a constant.
// Post: returns area
float findArea(float r)
{
  return (PI * pow(r,2));
}
```

Function definition

Function definition



# Function Definition (Input Arguments with One Result)

- **Syntax:**

*// function interface comment*

*ftype fname(formal-parameter-declaration-list)*

*{*

*local variable declarations*

*executable statements*

*}*

# Function Definition (Input Arguments with One Result)

- **Example:**

```
// Finds the cube of its argument.
// Pre: n is defined.
int cube(int n)
{
   return (n * n * n);
}
```

# Function Prototype (With Parameters)

- **Form:**

  *ftype fname(formal-parameter-type-list);*

- **Example:**

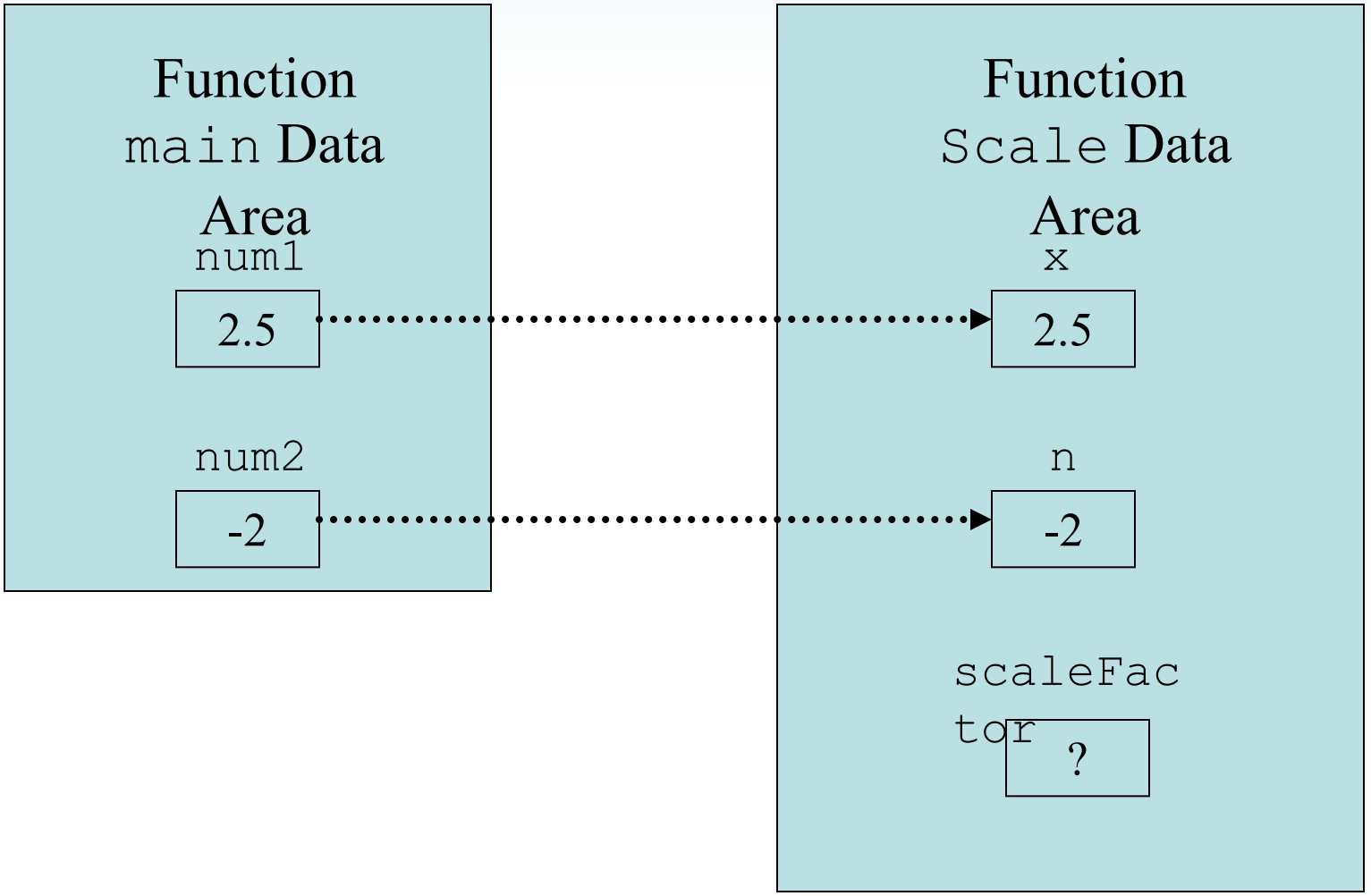
  int cube(int);

# Function Data Area

- Each time function is executed
  - an area of memory is allocated for storage of the function's data (formal parameters and local variables)
  - it is created empty with each call to the function
- When the function terminates
  - the data area is lost

# Data Areas After Call

## scale(num1, num2);

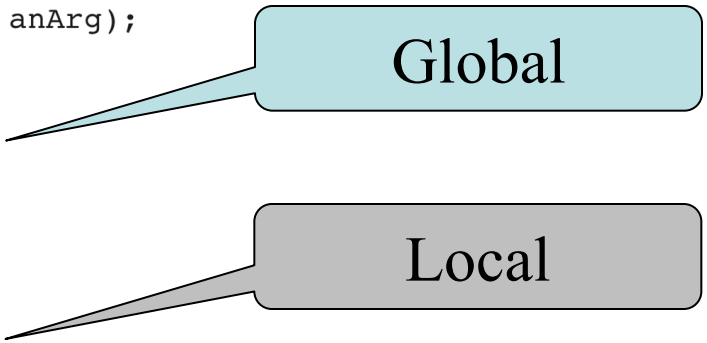


# Scope of Names

- **Scope -** where a particular meaning of a name is visible or can be referenced
- **Local -** can be referred to only within the one function
  - applies to
    - formal argument names
    - constants and variables declared within the function
- **Global -** can be referred to within all functions
  - useful for constants
  - must be used with care

# Listing 3.15 Outline of program for studying scope of names

```
void one(int anArg, double second);     // prototype 1

int funTwo(int one, char anArg);

const int MAX = 950;
const int LIMIT = 200;

int main()
{
   int localVar;
   . . .
} // end main

void one(int anArg, double second)      // header 1
{
   int oneLocal;                        // local 1
   . . .
} // end one

int funTwo(int one, char anArg)         // header 2
{
   int localVar;                        // local 2
   . . .
} // end funTwo
```



# Using Class `string`

- **Data abstraction**
  - facilitated in C++ through class feature
  - enables programmers to define new types
  - defines a set a values and a set of operations

# Accessing the String Library

- Must include the appropriate library

```
#include <string>
```

# String Objects

- Attributes include
  - character sequence it stores
  - length
- Common operations

  << >> = +

# Listing 3.16 Illustrating string operations

```
// File: stringOperations.cpp
// Illustrates string operations

#include <iostream>
#include <string>
using namespace std;

int main()
{
   string firstName, lastName;   // inputs - first and
                                 //          last names
   string wholeName;             // output - whole name
   string greeting = "Hello ";   // output - a greeting
                                 //          string

   // Read first and last names.
   cout << "Enter your first name: ";
   cin >> firstName;
   cout << "Enter your last name: ";
   cin >> lastName;
```

# Listing 3.16 Illustrating string operations (continued)

```
   // Join names in whole name
   wholeName = firstName + " " + lastName;

   // Display results
   cout << greeting << wholeName << '!' << endl;
   cout << "You have " << (wholeName.length() - 1)
        << " letters in your name." << endl;

   // Display initials
   cout << "Your initials are " << firstName.at(0)
        << lastName.at(0) << endl;

   return 0;
}
```

```
Enter your first name: Caryn
Enter your last name: Jackson
Hello Caryn Jackson!
You have 12 letters in your name.
Your initials are CJ
```

# Declaring `string` Objects

- General declarations format:

  *type identifier*$_1$, *identifier*$_2$, . . .;

- E.g.:

```
string firstName, lastName;
string wholeName;
string greeting = "Hello ";
```

# Reading and Displaying Strings

- Extraction operator is defined for strings

  cin >> firstName;  // reads up to blank or return

- Insertion operator is defined for strings

  cout << greetings << wholeName << '!' << endl;

- Can read string with blanks

  getline(cin, lastName, ;\n');

# String Assignment and Concatenation

- + puts strings together (concatenates)
- E.g.:

  wholeName = firstName + " " + lastName;

- Note need for space between names

# Operator Overloading

- Note use of +
  - usually means *addition* for two numeric values
  - has been redefined (overloaded) to also mean concatenation when applied to two strings
- Useful when defining new types

# Accessing String Operations

- **Member functions** *length* and *at*
- These functions can be called using **dot notation**
- Applies the identified operation to the named object
- E.g.:

  wholeName.length( )

  returns the length of the string currently stored in the variable wholeName

# Dot Notation

- Syntax: object.function-call
- Ex: firstName.at(0)

this returns the character in firstName that is at position 0 (start numbering characters from left, beginning at 0).

# Additional Member Functions

- Searching for a string

  wholeName.find(firstName)

- Inserting characters into a string

  wholeName.insert(0, “Ms.”);

- Deleting portion of a string

  wholeName.erase(10,4);

- Assign a substring to a `string` object

  title.assign(wholeName, 0 3);

# Introduction to Graphics

- In normal computer output (called **text mode**), we use cout to display lines of characters to the standard output device, or console.
- Now we discuss another mode of output (called **graphics mode**) that enables you to draw pictures or graphical patterns on your computer screen.

# Windows and Pixels

- In text mode, you don't pay much attention to the position of each line of characters displayed on the screen. In graphics programming, you control the location in a window of each line or shape that you draw. Consequently, you must know your window size and how to reference the individual picture elements (called **pixels**) in a window.
- You can visualize a window as an X-Y grid of pixels. If your window has the dimensions 400 x 300. The pixel at the top-left corner has X-Y coordinates (0, 0), and the pixel at the bottom-right corner has X-Y coordinates (399, 299).
- Notice the pixels in the Y-direction are numbered differently from how we are accustomed. The Y-coordinate values increase as we move down the screen. In a normal X-Y coordinate system, the point (0, 0) is at the bottom-left corner, not the top-left corner.

# Initializing a Window

- A graphics program is a sequence of statements that call graphics functions to do the work.
- Functions getmaxwidth and getmaxheight return the position of the last pixel in the X- and Y-directions on your computer screen. The statements

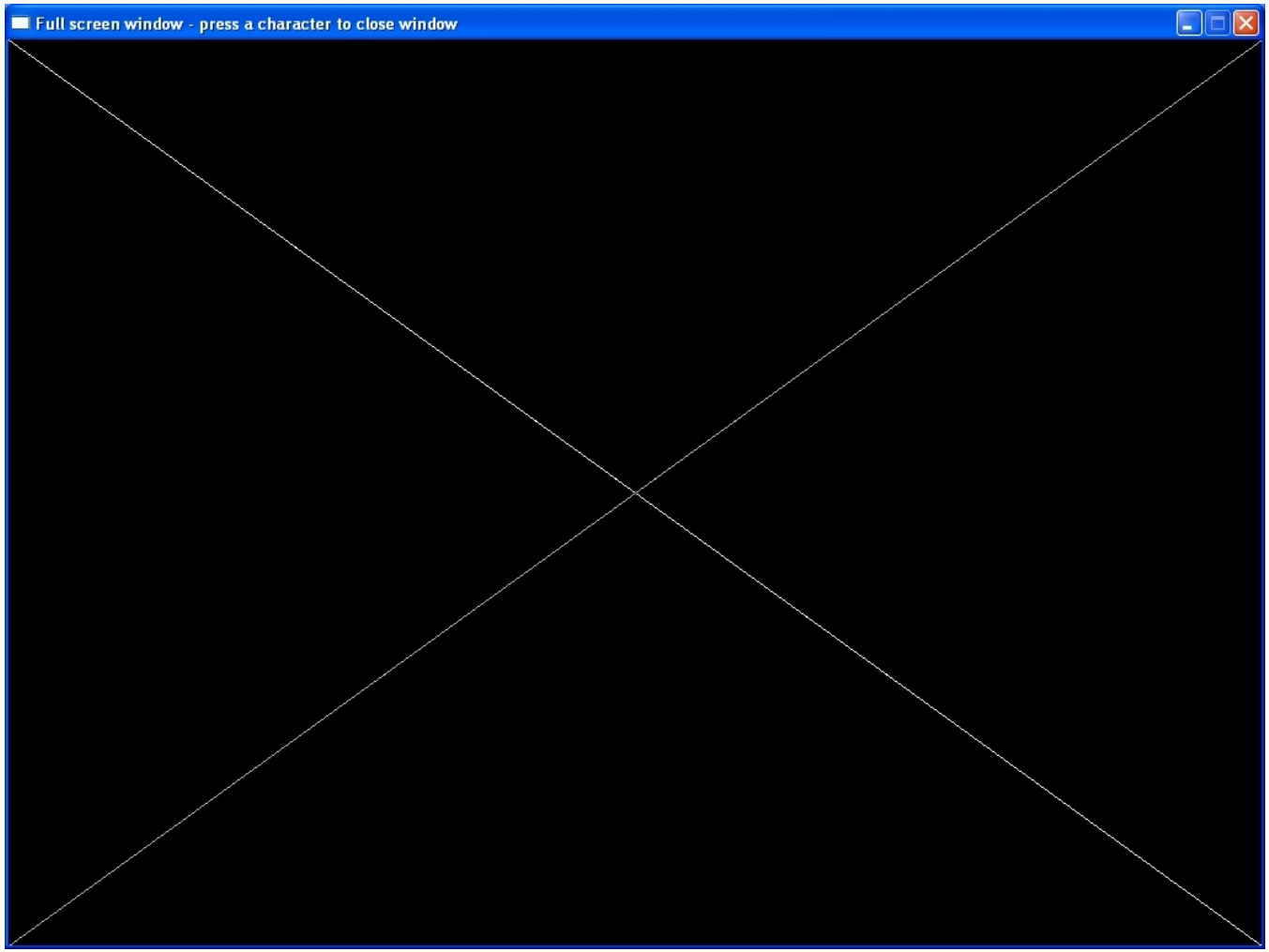
```
bigx = getmaxwidth();          // get largest x-coordinate.
bigy = getmaxheight();         // get largest y-coordinate.
initwindow(bigx, bigy,
    "Full screen window - press a character to close window");
```

pop up a window of maximum width and height with the third argument as the window label (see Fig. 3.13). The window position is set by the optional 4th and 5th arguments. If they are missing, the top-left corner of the window is at (0, 0).

# Listing 3.17 Draw intersecting lines

```
#include <graphics.h>
#include <iostream>

using namespace std;

int main()
{
   int bigx;  // largest x-coordinate
   int bigy;  // largest y-coordinate

   bigx = getmaxwidth();        // get largest x-coordinate
   bigy = getmaxheight();       // get largest y-coordinate
   initwindow(bigx, bigy,
              "Full screen window - press a key to close");

   // Draw intersecting lines
   line(0, 0, bigx, bigy);  // Draw white line from (0, 0) to (bigx, bigy)
   setcolor(LIGHTGRAY);     // Change color to gray
   line(bigx, 0, 0, bigy);  // Draw gray line from (bigx, 0) to (0, bigy)

   // Close screen when ready
   getch();           // pause until user presses a key
   closegraph();      // close the window

   // Display window size in console
   cout << "Window size is " << bigx << " X " << bigy << endl;

   return 0;
}
```

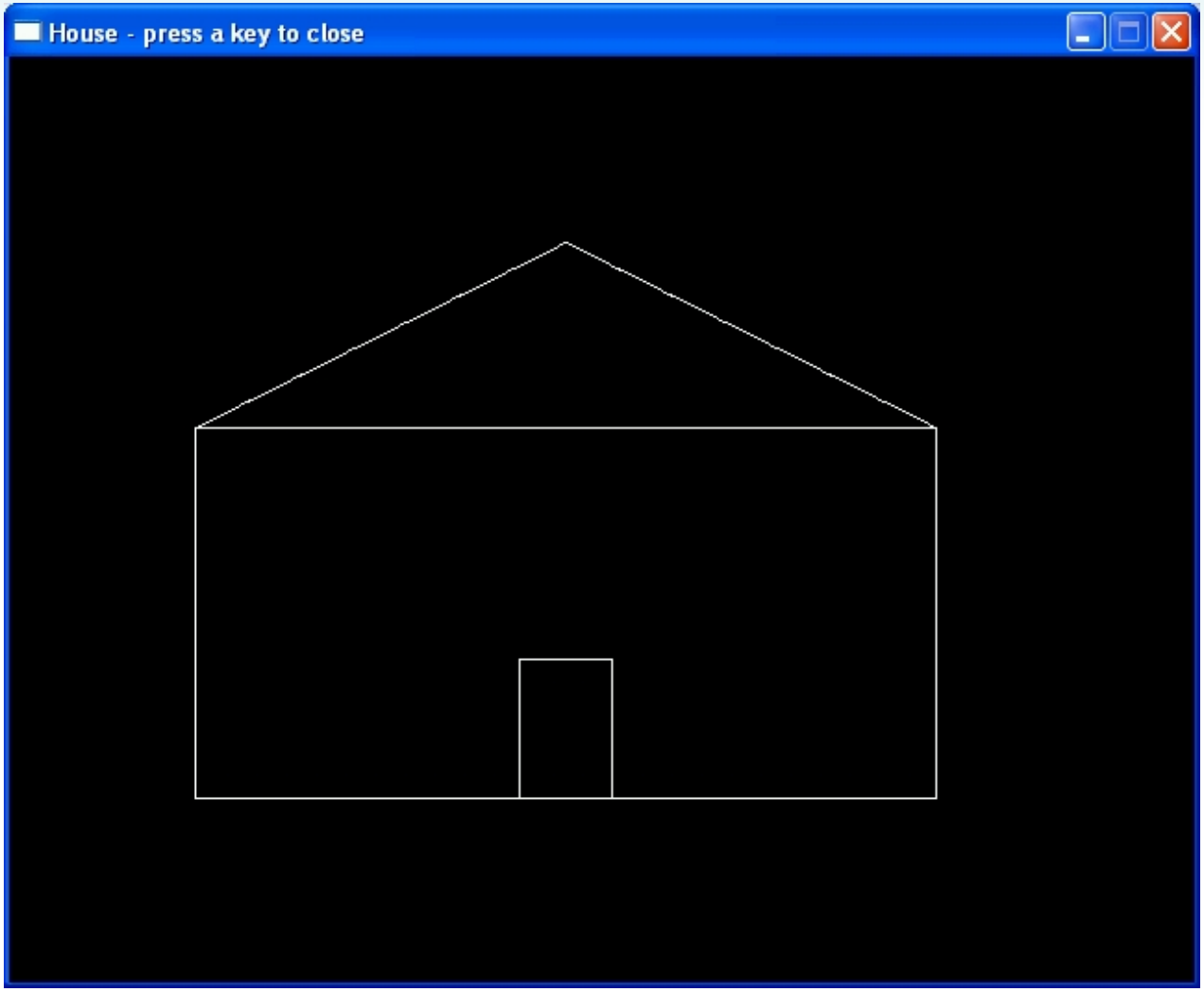
# Fig. 3.13 Intersecting Lines



# Listing 3.18 Draw a House

```
#include <graphics.h>

int main()
{
   // Define corners of house
   int x1 = 100; int y1 = 200;          // top-left corner
   int x2 = 300; int y2 = 100;          // roof peak
   int x3 = 500; int y3 = 200;          // top-right corner
   int x4 = 500; int y4 = 400;          // bottom-right corner
   int x5 = 325; int y5 = 400;          // bottom-right corner of door
   int x6 = 275; int y6 = 325;          // top-left corner of door

   initwindow(640, 500, "House - press a key to close", 100, 50);
   // draw roof
   line(x1, y1, x2, y2);  // Draw line from (x1, y1) to (x2, y2)
   line(x2, y2, x3, y3);  // Draw line from (x2, y2) to (x3, y3)

   // draw rest of house.
   rectangle(x1, y1, x4, y4);

   // draw door.
   rectangle(x5, y5, x6, y6);

   getch();            // pause until user presses a key
   closegraph();       // close the window
   return 0;
}
```

# Figure 3.14 House



# Listing 3.19 Draw Happy Face

```
#include <graphics.h>

int main()
{
   int midX, midY,                          // coordinates of center point
       leftEyeX, rightEyeX, eyeY,           // eye center points
       noseX, noseY,                        // nose center point
       headRadius,                          // head radius
       eyeNoseRadius,                       // eye/nose radius
       smileRadius,                         // smile radius
       stepX, stepY;                        // x and y increments

   initwindow(500, 400, "Happy Face - press key to close", 200, 150);

   // draw head.
   midX = getmaxx() / 2;                    // center head in x-direction
   midY = getmaxy() / 2;                    // center head in y-direction
   headRadius = getmaxy() / 4;              // head will fill half the window
   circle(midX, midY, headRadius);          // draw head.
```

# Listing 3.19 cont'd.

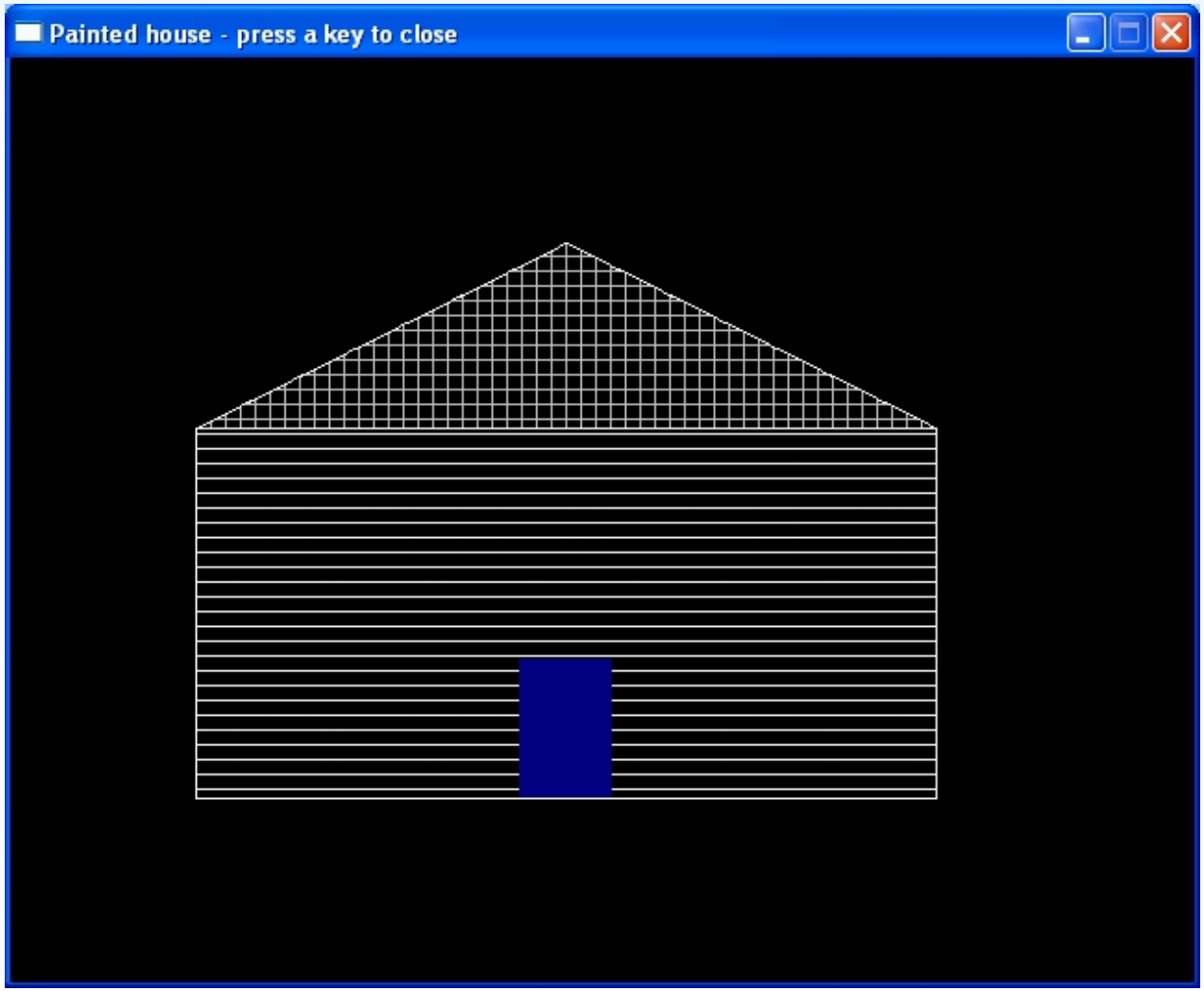
```
    // draw eyes.
    stepX = headRadius / 4;                    // x-offset for eyes
    stepY = stepX;                             // y-offset for eyes and nose
    leftEyeX = midX - stepX;                   // x-coordinate for left eye
    rightEyeX = midX + stepX;                  // x-coordinate for right eye
    eyeY = midY - stepY;                       // y-coordinate for both eyes
    eyeNoseRadius = headRadius / 10;
    circle(leftEyeX,  eyeY, eyeNoseRadius);    // draw left eye.
    circle(rightEyeX, eyeY, eyeNoseRadius);    // draw right eye.

    // draw nose.
    noseX = midX;                              // nose is centered in x direction.
    noseY = midY + stepY;
    circle(noseX, noseY, eyeNoseRadius);

    // draw smile.
    smileRadius = int(0.75 * headRadius + 0.5);   // 3/4 of head radius
    arc(midX, midY, 210, 330, smileRadius);

    getch();
    closegraph();
    return 0;
}
```

# Figure 3.15 Happy face

# Listing 3.20 Paint a House

```
#include <graphics.h>

int main()
{
   // Insert same code as in Listing 3.18 (Draw a house)

   ...

   // draw rest of house.
   rectangle(x1, y1, x4, y4);

    // Paint the house
   setfillstyle(HATCH_FILL, LIGHTGRAY);
   floodfill(x2, y2 + 10,  WHITE);      // Paint the roof
   setfillstyle(LINE_FILL, WHITE);
   floodfill(x2, y1 + 10, WHITE);       // Paint the house

   setfillstyle(SOLID_FILL, BLUE);
   bar(x5, y5, x6, y6);                 // Draw blue door

   getch();
   closegraph();
   return 0;
}
```

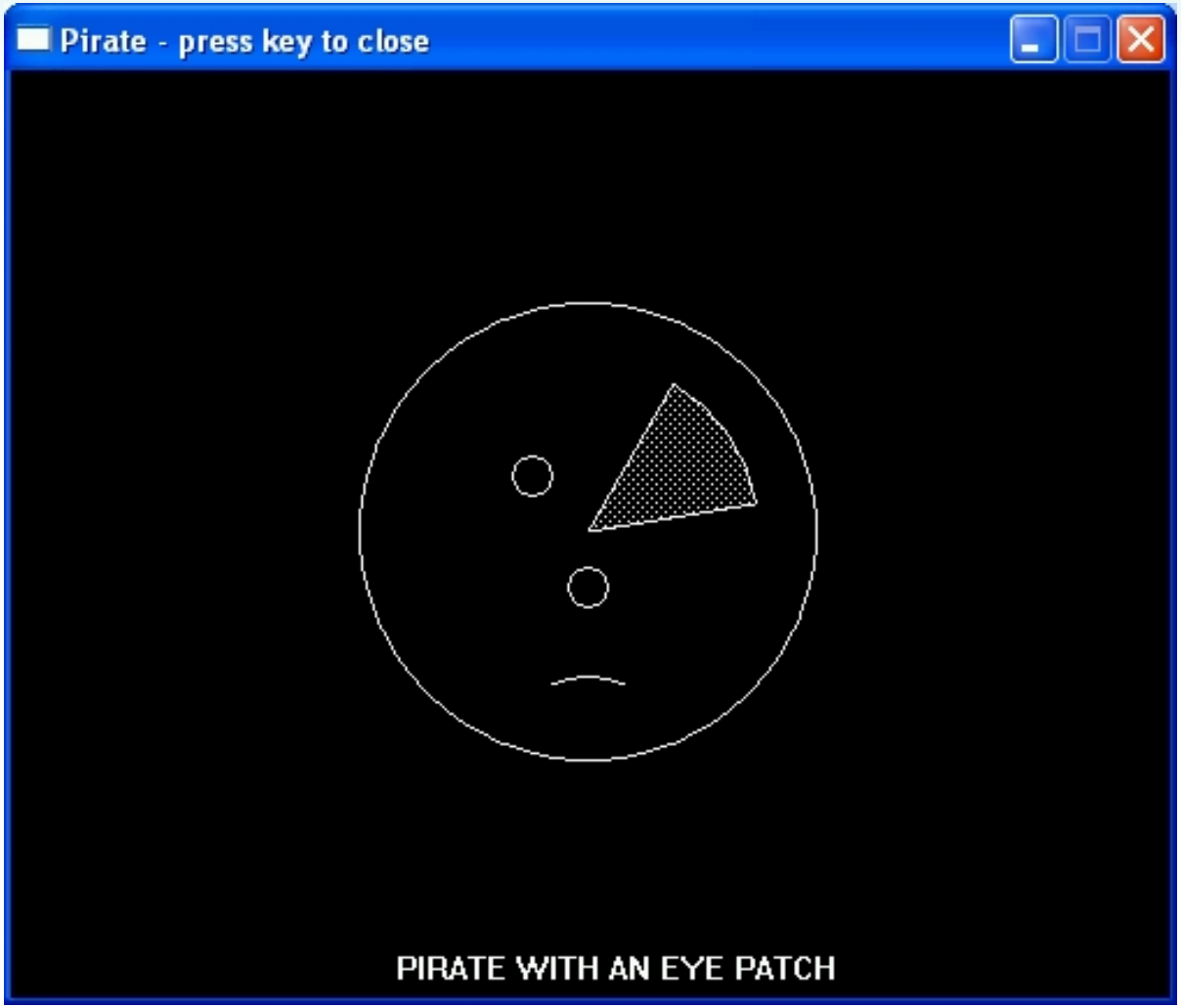
# Figure 3.16 Painted House



# Listing 3.21 Draw Pirate

```
#include <graphics.h>

int main()
{
   // Insert same code as in Listing 3.19 (Draw Happy Face)
   . . .

   // Draw nose
   noseX = midX;                              // nose is centered in x direction
   noseY = midY + stepY;
   circle(noseX, noseY, eyeNoseRadius);

   // Draw frown
   smileRadius = int(0.75 * headRadius + 0.5);    // 3/4 of head radius
   arc(midX, midY + headRadius, 65, 115, smileRadius / 2);  // Draw frown

   setfillstyle(CLOSE_DOT_FILL, WHITE);
   pieslice(midX, midY, 10, 60, smileRadius);    // Draw eye patch

   outtextxy(getmaxx() / 3, getmaxy() - 20, "PIRATE WITH AN EYE PATCH");

   getch();
   closegraph();
   return 0;
}
```

# Fig. 3.17 Pirate with eye patch



Chapter 1

**Short Answer:**

3. What is software?

Answer: The programs or instructions for the computer.

4. Name an input device.

Answer: Keyboard OR mouse OR other

5. What part of the computer performs operations on data?

Answer: The central processing unit.

6. Every memory cell must have a(n) _____.

Answer: Address

7. What are the memory locations inside the CPU called?

Answer: Registers

8. What is the secondary storage device that is fixed inside a personal computer called?

Answer: A hard disk/drive

9. What is the program that allows you to access the World Wide Web called?

Answer: A Web browser

10. What is UNIX an example of?

Answer: An operating system

11. What is the process of loading the computer's operating system into memory called?

Answer: Booting the computer

12. What does GUI stand for?

Answer: Graphical user interface

13. In object-oriented programming what are the two kinds of properties that an object has?

Answer: Data and methods

14. If tree and oak are in a hierarchy of classes, tree would be the _____.

Answer: Superclass

15. What is the purpose of a compiler?

Answer: It translates programs into machine language.

16. What program combines all the files needed for a complete program to run into one file?

Answer: Linker

17. What is the file containing the program called after it has been translated into machine language?

Answer: Object file.

18. How many steps are there in the software development method?

Answer: 6

19. What will be identified in the analysis step?

Answer: Inputs, outputs and formulas.

20. What is a virus?

Answer: A program that copies itself throughout the computer's disk memory.

**Multiple Choice:**

1. The first electronic computer was completed:

a) before 1950
b) in the 1950s
c) in the 1960s
d) in the 1970s

Answer: A

2. Which of the following is not a category of computers:

a) microcomputer
b) mainframe
c) maxicomputer
d) supercomputer

Answer: C

3. Which of the following is not a hardware component?

a) Main memory
b) Central processing unit
c) Operating system
d) Keyboard

Answer: C

4. How many bits are in a byte?

a) 4
b) 8
c) 16
d) 256

Answer: B

5. The instructions that a computer needs to get running when switched on are stored in:

a) Main memory
b) ROM
c) RAM
d) Secondary memory

Answer: B

6. Which of the following is **not** a secondary storage device:

a) Hard drive
b) Floppy disk
c) CD
d) ROM

Answer: D

7. Which of the following is **not** an input/output (I/O) device:

a) Zip drive
b) Monitor
c) Keyboard
d) Printer

Answer: A

8. What kind of software allocates memory, processor time and other resources?

a) GUI
b) Application software
c) Compiler
d) Operating system

Answer: D

9. Which of the following is an operating system?

a) UNIX
b) Microsoft Word
c) Assembly Language
d) C++

Answer: A

10. Which of the following is an application program?

a) Microsoft Windows
b) WordPerfect
c) C++
d) Java

Answer: B

11. Which of the following would most likely be a class (the others would be objects of a class):

a) Lassie
b) my house
c) George Washington
d) flower

Answer: D

12. Which would be the bottom class if the following classes were in a hierarchy?

a) mammal
b) cat
c) tiger
d) animal

Answer: C

13. Which of the following is not a step in the software development method?

a) testing
b) analysis
c) compiling
d) design

Answer: C

14. Which step of the software development method is the one in which the program is written in C++ code?

a) Analysis
b) Design
c) Implementation
d) Testing

Answer: C

15. Which of the following is an ethical activity?

a) hacking
b) plagiarism
c) software piracy
d) algorithm refinement

Answer: D

**True/False:**

1. The first electronic computer was very small.

Answer: False

2. A byte is bigger than a bit.

Answer: True

3. When a value is stored in a memory cell, the value that was there previously is lost.

Answer: True

4. When you are typing a paper the characters you type are stored in RAM.

Answer: True

5. When the computer is turned off the contents of main memory are stored.

Answer: False

6. Memory cells can store data or instructions.

Answer: True

7. The computer user can store data in ROM.

Answer: False

8. The CPU can process data on secondary storage devices.

Answer: False

9. A network that serves users over a large geographic area is called a LAN.

Answer: False.

10. A house class is a subclass of a building class.

Answer: True

11. A loader combines code from different programs into one file.

Answer: False

12. In the testing phase of the software development method you know your program is correct if it runs correctly once.

Answer: False

13. The person who maintains the program is not always the same as the person who wrote it.

Answer: True.

14. The C++ word cout is used to display information on the screen.

Answer: True

15. It is ethical for programmers to copy software without permission.

Answer: False

Chapter 2

**Short Answer:**

1. What symbol indicates that the following line is a comment?

Answer: // (or /*)

2. What are the two kinds of statements inside a function?

Answer: Declaration statements and executable statements

3. What does the following statement do?

```
cout << "This is good.";
```

Answer: It displays the words

```
This is good
```

on the screen

4. What C++ word is used to read information in from the keyboard?

Answer: cin

5. What is the name for a word that has a specific meaning in C++?

Answer: Reserved word (or keyword)

6. What is an identifier?

Answer: A name for the data elements used in a program.

7. What would be the best choice for the data type for a person's middle initial?

Answer: char

8. What would be the best choice for the data type for the distance between two houses?

Answer: float (or double or long double)

9. Write the following number in normal decimal notation:

```
734E-4
```

Answer: 0.0734

10. What is the escape sequence for a linefeed?

Answer: '\n'

11. What is the following kind of statement called?

```
int counter;
```

Answer: A variable declaration.

12. If the variables name and color contain these values:

| name | color |
|---|---|
| "Elmo" | "red" |

write a statement **using these variables** that will output:
Elmo, the puppet, is red!

Answer: cout << name << ", the puppet, is " << color << "!";

13. What is a prompt?

Answer: An output statement that informs the user what to enter.

14. What is the output of the following code segment if the data entered are 3, 2 and 4?

```
int x, y, z;
cout << "Enter three numbers: ";
cin >> x >> y >> z;
y = x + z;
x = z + 1;
cout << x << " " << y<< " " << z;
```

Answer: 5 7 4

15. Why do programmers use comments?

Answer: To help others read and understand the program.

16. What is the result of the following arithmetic expression?

$5 + 7/3$

Answer: 7

17. What is the result of the following arithmetic expression?

$7 - 2 * (4 + 1)$

Answer: -3

18. Write the following statement in C++:

$a = 2bc^3$

Answer: a = 2 * b * c * c * c

19. Write the following statement in C++:

$$x = \frac{3y}{5 - z}$$

Answer: x = (3 * y)/(5 – z) OR x = 3 * y / (5 – z)

20. If data is being read from a file for a program in batch mode, a prompt is replaced with what?

Answer: An echo.

**Multiple Choice:**

1. iostream is part of:

a) the comment section
b) the C++ standard library
c) the compiler
d) the main function

Answer: B

2. Which of the following lines must end with a semicolon (;)?

a) #include <iostream>
b) int main ( )
c) // *a comment*
d) using namespace std

Answer: D

3. Which of the following is a compiler directive:

a) #include <iostream>
b) using namespace std;
c) int main ( )
d) return 0;

Answer: A

4. **main** is a:

a) compiler directive
b) comment

c) function
d) namespace

Answer: C

5. Which of the following is **not** a C++ reserved word?

a) float
b) number
c) const
d) using

Answer: B

6. Which of the following could be used as an identifier?

a) first_name
b) 5stars
c) small number
d) yes&no

Answer: A

7. Which of the following is **not** a valid identifier?

a) Pig3
b) I_HAVE_A_DOG
c) help!
d) _answer

Answer: C

8. Which of the following would be the best choice for a pay rate of a student?

a) payrate
b) PAYRATE
c) p
d) pay_rate

Answer: D

9. What would be the best choice for the data type for a person's take home pay?

a) int
b) float
c) char

d) string

Answer: B

10. What would be the best choice for the data type for the number of students in a class?

a) int
b) float
c) bool
d) char

Answer: A

11) What would be the best choice of data type for a person's address?

a) int
b) float
c) char
d) string

Answer: D

12) Which of the following statements is correct?

a) float num1; num2;
b) int day, night;
c) int blue = 5.0;
d) string black = 'white';

Answer: B

13) What is the value of the following arithmetic expression if the variables have these values:

| p | s | x |
|---|---|---|
| | 5 | 20 | 6 |

p + 3 * x – s

a) -37
b) 3
c) 28
d) 59

Answer: B

14) What is the value of the following arithmetic expression if the variables have these values?

| a | b | c |
|---|---|---|
| 7 | 3 | 10 |

```
a + b % c
```

a) 0
b) 7
c) 7.3
d) 10

Answer: D

15) What kind of error is the following:

```
float height = 6.0,
```

a) syntax
b) logic
c) run-time
d) undetected

Answer: A

**True/False:**

1. The computer will carry out the instructions that follow the symbol //

Answer: False

2. A program must have a main function.

Answer: True

3. The following is an example of a declaration statement:

```
cout << "Enter a number: ";
```

Answer: False

4. An identifier must start with a letter or an underscore.

Answer: True

5. It is best to use very short identifiers.

Answer: False

6. In the statement:

```
cout << "Hello!"
```

"Hello!" is called a string literal.

Answer: True

7. There is no limit on the size of the numbers that can be stored in the **int** data type.

Answer: False

8. 76.45e-2 is a valid value for a float data type.

Answer: True

9. There are only two possible values for the **bool** data type.

Answer: True

10. All data types take up the same amount of storage.

Answer: False

11. It is good program style to put spaces between words and symbols.

Answer: True

12. A C++ statement cannot extend over more than one line.

Answer: False

13. In C++ addition is always evaluated before subtraction.

Answer: False

14. The value of 3/7 is 0.

Answer: True

15. >> is used for output.

Answer: False

Chapter 3

**Short Answer:**

1. Why is PI a const in the program that finds the area and circumference of a circle?

Answer: So that it cannot be changed during the run of the program.

2. What is the library function that finds the square root of a number sent to it?

Answer: sqrt

3. What is the value of y after the following code executes:

```
float y = 3.4 + sqrt (25.0);
```

Answer: 8.4

4. What is the library function that finds the absolute value of an integer?

Answer: abs

5. What is top-down design?

Answer: A method of solving a problem by breaking it into smaller parts.

6. What is a void function?

Answer: A function that does not return a value.

7. What is the output of this code, given the function definition that follows:

```
doSomething( );
doSomething( );          // code in main

void doSomething( ) {                    // function definition
        cout << "Hi";
        cout << "Bye";
}
```

Answer: HiByeHiBye

8. Why do we use functions?

Answer: To break the program into smaller pieces.
OR
So that parts of our code may be reused.

9. In the function call

```
doesSomething (7, x, p);
```

what are 7, x and p called?

Answer: arguments

10. In the function definition

```
float aValue (int r, int s) {……
```

what are r and s called?

Answer: parameters

11. In the function prototype

```
int calculation (float p);
```

int is called _____?

Answer: the return type

12. What is the output of the following code given the function definition that follows:

```
cout << myFunction (7, 5, 2);     // code in main

int myFunction (int a, int b, int c)       // function definition
{
        int p = a + 2 * b + c;
        return p + 3;
}
```

Answer: 22

13. The program that you write to test a function is called a(n) _____.

Answer: driver

14. The region in a program where a particular name is visible or can be referenced is called the name's _____.

Answer: scope

15. What function would you use if you wanted to read a string that might contain a blank?

Answer: getline

16. What would be the output of the following lines of code?

```
string one = "red", two = "blue";
string three = two + "and" + one;
```

```
cout << three;
```

Answer: blueandred

17. Functions like length, at and erase are called _____ of the string class.

Answer: member functions

18. What line of code must you put in a program if you want to use the money class?

Answer: #include "money.h"

19. What is the only type of function that does not have to end with a return statement?

Answer: a void function

20. What must a call to a member function always be preceded by?

Answer: an object name (and a dot)

**Multiple Choice:**

1. What is the value of z after the following code executes:

```
float x, y, z;
x =9.0;
y = 16.0;
z= sqrt ( x + y);
```

a) 5.0
b) 7.0
c) 337.0
d) 625.0

Answer: A

2. What is the value of x after execution of the following code?

```
int x, y, z;
y = 3;
z = 4;
x = pow (z, y);
```

a) 7
b) 12
c) 64
d) 81

Answer: C

3. What is the value of x after the following code is executed?

```
y = 3.7;
x = floor (y);
```

a) 3
b) 3.7
c) 4
d) 7

Answer: A

4. A structure chart shows:

a) the syntax of a statement
b) the output of a program
c) the relationships among the subproblems of a problem
d) stick figures on the screen

Answer: C

5. Which of the following is a valid function prototype?

a) float some function ( );
b) void nothing;
c) int (int thing);
d) void something ( );

Answer: D

6. What is the output of this line of code, given the following definition?

```
display (7, 3);                    // code in main

void display (int x, int y) {              // function definition
        cout << y << " and " << x;
}
```

a) 7 and 3
b) 3 and 7
c) 10
d) 7 3

Answer: B

7. What is the output of this line of code, given the following definition?

```
cout << aNumber (2, 5);        // code in main

int aNumber (int x, int y) {    // function definition
        return (2 * x - y);
}
```

a) -6
b) -1
c) 6
d) 8

Answer: B

8. What is the value of num after the following code is executed, given the function definition that follows:

```
float num = findIt(5.0, 2);              // code in main

float findIt (float one, int two)        // function definition
{
        one = one + two;
        return (one / two);
}
```

a) 0.5
b) 2.5
c) 3
d) 3.5

Answer: D

9. What is the output of the following line of code, given the function definition that follows:

```
cout << calc (1, 2) + calc (2,3);        // code in main

int calc (int x, int y)                  // function definition
{
        x = x + 1;
        return x % y;
}
```

a) 0
b) 1
c) 2
d) none of the above

Answer: A

10. Which of the following function calls is valid for the function prototype?

```
int calculation (int m, char p, float q);    // function prototype
```

a) cout << calculation (3, "yes", 2.0);
b) cout << calculation (5, p, 4.5);
c) cout << calculation (5, 's');
d) cout << calculation (10, 'r', 7.0);

Answer: D

11. Which of the following function calls is valid for the function prototype?

```
void workItOut (int p, float q);    // function prototype
```

a) cout << workItOut (2, 3.0);
b) workItOut (2, 3);
c) num = workItOut (5, 2.5);
d) workItOut ("p", 3.7);

Answer: B

12. Which of the following would be the best function prototype for a function that calculates the volume of a box?

a) void volume (int, int, int);
b) int volume (float, float, float);
c) float volume (int, int, int);
d) float volume (float, float, float);

Answer: D

13. What would be the output of the following lines of code?

```
string word = "somebody";
cout << word.at(3);
```

a) m
b) e
c) r
d) d

Answer: B

14. What would be the value of x after the following code has executed?

```
string words = "How are you?";
int x = words.length( );
```

a) 9
b) 10
c) 11
d) 12

Answer: D

15. What is the value of phrase after the following code executes?

```
string phrase = "A nice day", exes = "xxxx";
phrase.replace(2, 3, exes);
```

a) "A xxxxce day"
b) "Axxxxice day"
c) "A xxxxe day"
d) "Axxxxce day"

Answer: C

**True/False:**

1. To use the sqrt function in a program you should include the cmath library.

Answer: True

2. The function floor (x) rounds its argument x to the nearest integer

Answer: False

3. Function prototypes are placed at the end of a program.

Answer: False

4. Void functions return an integer.

Answer: False

5. One reason for using functions is called procedural abstraction.

Answer: True

6. Void functions may have arguments.

Answer: True

7. Functions are executed in the order in which they are defined.

Answer: False

8. A precondition is a condition that must be true before the function is called.

Answer: True

9. The values in the function data area are kept after the function ends.

Answer: False

10. In the following program outline the name var1 is visible (can be referenced) in the function fun.

```
int fun (int a, int b);

int main ()
{
        int var1;
        …
}

int fun (int a, int b)
{
        …
}
```

Answer: False

11. In the following program outline the name var1 is visible (can be referenced) in the function fun.

```
int fun (int a, int b);

const int var1 = 5;

int main ()
{
        …
}

int fun (int a, int b)
{
        int c;
        …
}
```

Answer: True

12. The string class is part of the C++ standard library.

Answer: True

13. The file money.h is called the implementation file.

Answer: False

14. A function prototype must be terminated with a semicolon.

Answer: True

15. It is best to wait until the whole program is written before testing your functions.

Answer: False

**With My Best Wishes**
**Dr. Heba Askr**