SWE20004
Technical Software Development

Lecture 2
Problem Solving Using C++

Outline

- Modular programs
- Programming style
- Data types
- Arithmetic operations
- Variables and declaration statements
- Common programming errors

Introduction to C++

- **Modular program:** A program consisting of interrelated segments (or **modules)** arranged in a logical and understandable form
  - Easy to develop, correct, and modify
- Modules in C++ can be classes or functions

Introduction to C++ (continued)

- **Function:** Usually (but not always) accepts input(s), processes the input(s), and produces output(s)
  - A function's processing is encapsulated and hidden within the function

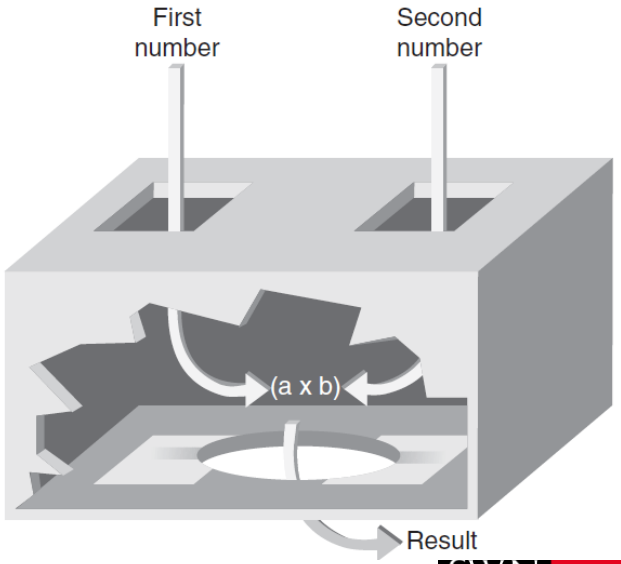


Figure 2.2 A multiplying function

Introduction to C++ (continued)

- **Class:** Contains both data and functions used to manipulate the data
  - It represents a real or imaginary entity
  - Acts as a template used to create individual instances (also called objects) of the entity it represents

Introduction to C++ (continued)

- **Identifier:** A name given to an element of the language, such as a class, function or a variable
  - Rules for forming identifier names:
    - First character must be a letter or underscore
    - Only letters, digits, or underscores may follow the initial letter (no blanks allowed)
    - Maximum length of an identifier = 1024 characters
    - Keywords/Reserved words cannot be used as identifiers

Introduction to C++ (continued)

- **Keyword**: A reserved name that represents a built-in object or function of the language

auto	delete	goto	public	this
break	do	if	register	template
case	double	inline	return	typedef
catch	else	int	short	union
char	enum	long	signed	unsigned
class	extern	new	sizeof	virtual
const	float	overload	static	void
continue	for	private	struct	volatile
default	friend	protected	switch	while

Table 2.1: Keywords in C++

Introduction to C++ (continued)

- Examples of valid C++ identifiers (using CamelCase convention):

```
degToRad intersect    addNums
slope    bessell    multTwo
findMax    density
```

- Examples of invalid C++ identifiers:

```
1AB3    (begins with a number)
E*6     (contains a special character)
while   (this is a keyword)
```

Introduction to C++ (continued)

- Function names
  - Require a set of parentheses at the end
  - Can use mixed upper and lower case
  - Should be meaningful, or be a **mnemonic**
- Examples of function names:

```
easy()  c3po() r2d2()
theForce()
```

- Note that C++ is a case-sensitive language therefore `easy()` `Easy()` `EASY()` are three distinct function names

The `main()` Function

- Overall structure of a C++ program contains one function named `main()`, called the **driver function**
- All other functions are invoked from `main()`

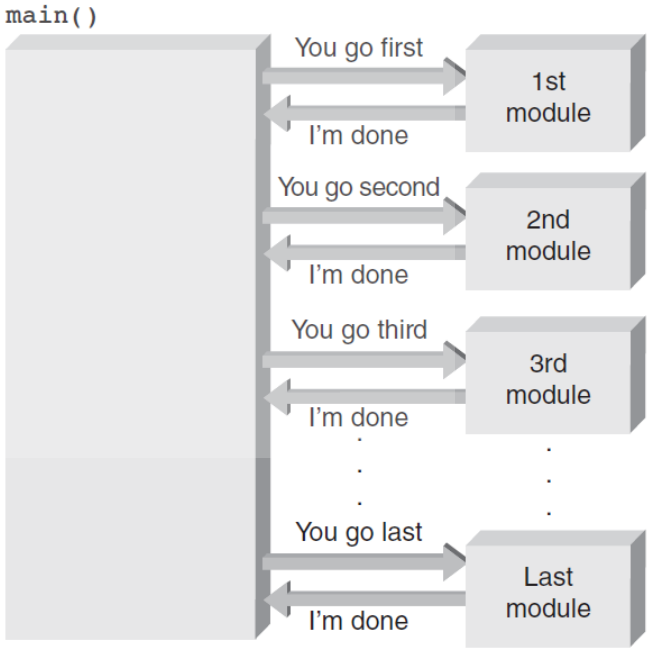
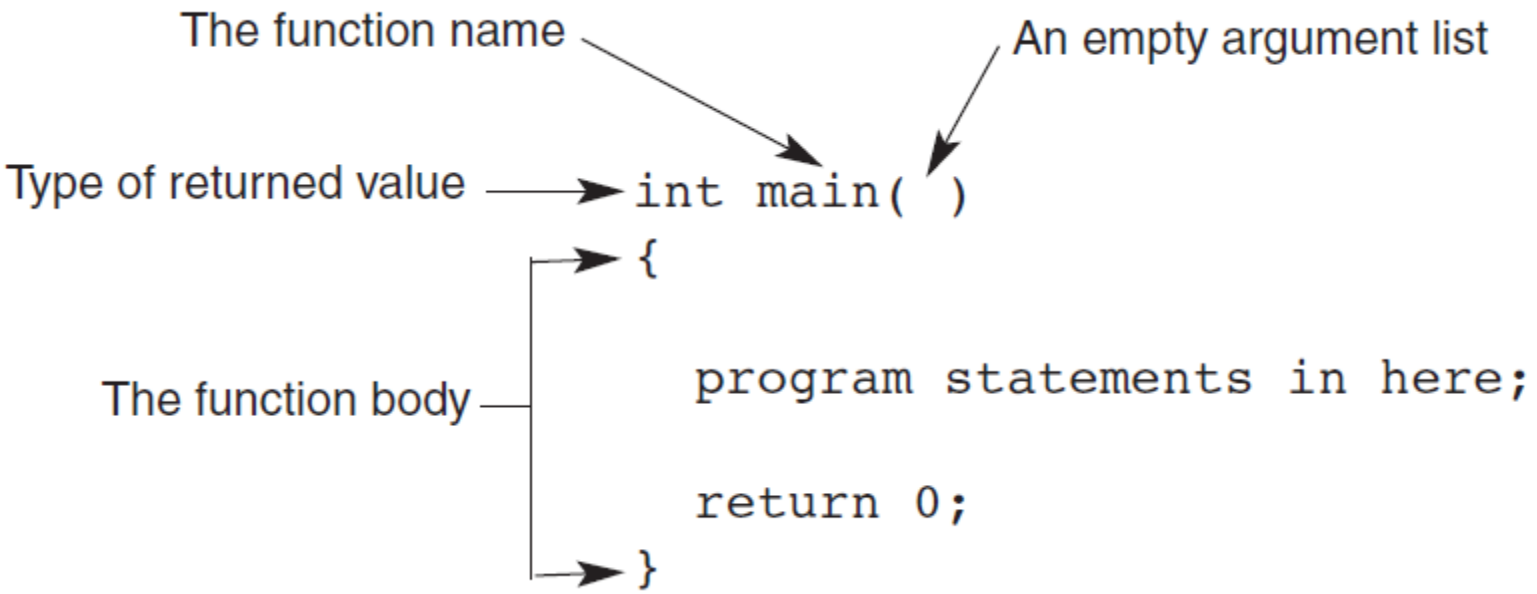


Figure 2.3 The `main()` function directs all other functions.

The `main()` Function (continued)



The `cout` Object

- **`cout`** object: An output object that sends data to a standard output display device (the computer monitor or the screen)

Program 2.1

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

int main()
{
   cout << "Hello there world!";

   return 0;
}
```

The `cout` Object (continued)

- Preprocessor command: Starts with a #
  - Causes an action before the source code is compiled into machine code
- **`#include <file name>`**: Causes the contents of the named header file to be inserted into the source code where the statement is
- C++ provides a standard library with many pre-written classes that can be included
- **Header files**: Files included at the head (top) of a C++ program

The `cout` Object (continued)

- **`#include<iostream>`** makes **`cout`** available for the program to use for producing output
- **`using namespace <namespace name>:`** Indicates where header file is located
  - Namespaces qualify a name
    - A function name in your class can be the same as one used in a standard library class

The `cout` Object (continued)

- **Escape sequence:** One or more characters preceded by a backslash, \

Program 2.3

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

int main()
{
   cout << "Computers everywheren\n as far as\n\nI can see";

   return 0;
}
```

strings

- **String:** Short for “string of characters”. Consists of a combination of letters, numbers, and special characters enclosed in double quotes (delimiter)
- 2 types in C++:
  - C-strings (null-terminated char arrays)
    - created automatically every time you type something in " "
  - C++ string objects
    - Growable, shrinkable, know their length, immune to buffer overflows, indexable (like arrays).
    - Support many memory-safe functions for comparing, copying, searching.

strings

- In practice it is convenient to:
  - use C-strings when the text is static / hard-coded (not stored in a variable)
  - use C++ strings where the text is used as data
  - use C++ strings when text is to be processed (searched, changed, copied)
  - C-strings can be used to initialise C++ strings

string

```
#include <string>
```

- A C++ ***string*** is a pre-defined class
  - starts with lower-case ***s***
- Unlike c-strings, a C++ string
  - is empty when created `string s1, s2;`
  - knows it's own length `s1.length();`
  - can be assigned safely using =
    - `s1 = "text"; s1 = s2;`
  - can be compared correctly using == `if (s1 == s2)`
  - can be appended using +=
    - `s1 += s2; s1 += "c-string";`
  - can be indexed like a c-string `s1[3];` `s1.at(3)`
  - For more information http://www.cplusplus.com/reference/string/string/

strings

- Examples:

```
/* just output a c-string to the screen */
cout << "This is a c-string" << endl;

/* create a C++ string and initialise it with a C-String */
string myString("This is a C++ string");
cout << myString << endl; //show it

/* call the string's length() and substr() functions */
cout << "myString is " << myString.length()
       << " characters long." << endl;
cout << myString.substr(0, 8) << "C++" << endl;
```

Comments

- **Comments:** Explanatory remarks in the source code added by the programmer
- **Single line (inline) comment:** Begins with **//** and continues to the end of the line
    - Example:

```
// this program displays a message
#include <iostream>
 using namespace std;

  int main ()
  {
       cout << "Hello there world!"; //displays text
      return 0;
  }
```

Comments (continued)

- **Block (multiline) comments:** comments that span across two or more lines
  - Begin with **/*** and end with ***/**
  - Example:

```
/* This is a block comment that
spans
across three lines */
```

Data Types

- **Data type:** A set of values and the operations that can be applied to these values
- Two fundamental C++ data groupings:
  - **Class data type**: Created by the programmer by defining a class
  - **Built-in data type** (primitive type): Part of the C++ compiler

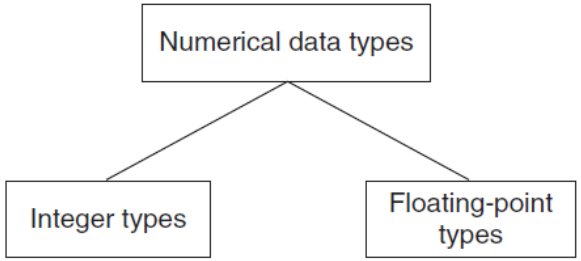


Figure 2.5 Built-in data types

Data Types (continued)

Built-in Data Type	Operations
Integer	`+`, `-`, `*`, `/`, `%`, `=`, `==`, `!=`, `<=`, `>=`, `sizeof()`, and bit operations (see Chapter 15, available online)
Floating point	`+`, `-`, `*`, `/`, `=`, `==`, `!=`, `<=`, `>=`, `sizeof()`

Table 2.2 Built-In Data Type Operations

Data Types (continued)

- **Literal (constant):** An actual value
  - Examples:

```
3.6          //numeric literal
"Hello"      //string literal
```

- **Integer:** A whole number
- C++ has nine built-in integer data types
  - Each provides different amounts of storage (compiler dependent)

Integer Data Types

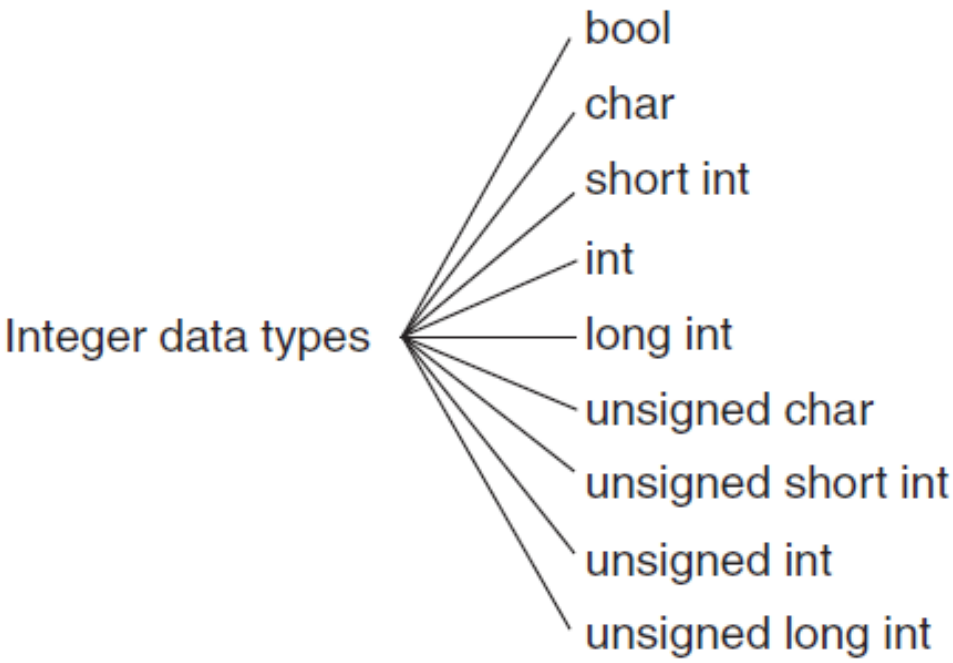


Figure 2.6 C++ integer data types

Integer Data Types (continued)

- **`int`** data type: Whole numbers (integers), optionally with plus (**`+`**) or minus (**`–`**) sign
  - Example: **`2, -5`**
- **`char`** data type: Individual character; any letter, digit, or special character enclosed in single quotes
  - Example: **`'A'`**
  - Character values are usually stored in **ASCII code**

Integer Data Types (continued)

- **`bool`** data type: Represents Boolean (logical) data
  - Restricted to two values: **true** or **false** (non-zero or zero values)
  - Useful when a program must examine a condition and take a prescribed course of action, based on whether the condition is true or false

Determining Storage Size

- A unique feature of C++ is that you can see where and how values are stored
  - **`sizeof()`** operator provides the number of bytes used to store values of the data type named in the parenthesis
  - Values returned by **`sizeof()`** are compiler dependent

Determining Storage Size (continued)

Program 2.5

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

int main()
{
  cout << "\nData Type   Bytes"
       << "\n---------   -----"
       << "\nint           " << sizeof(int)
       << "\nchar          " << sizeof(char)
       << "\nbool          " << sizeof(bool)
       << '\n';

  return 0;
}
```

Signed and Unsigned Data Types (continued)

Name of Data Type	Storage Size	Range of Values
`char`	1	256 characters
`bool`	1	true (considered as any positive value) and false (which is a 0)
`short int`	2	-32,768 to +32,767
`unsigned short int`	2	0 to 65,535
`int`	4	-2,147,483,648 to +2,147,483,647
`unsigned int`	4	0 to 4,294,967,295
`long int`	4	-2,147,483,648 to +2,147,483,647
`unsigned long int`	4	0 to 4,294,967,295

Table 2.5 Integer Data Type Storage

Floating-Point Types

- **Floating-point number** (real number): Zero or any positive or negative number containing a decimal point
  - Examples: `+10.625` `5.` `-6.2`
  - No special characters are allowed
  - Three floating-point data types in C++:
    - `float` (single precision)
    - `double` (double precision)
    - `long double`

Floating-Point Types (continued)

Type	Storage	Absolute Range of Values (+ and -)
`float`	4 bytes	$1.4012984643248 1707 \times 10^{-45}$ to $3.4028234663852 8860 \times 10^{38}$
`double` and `long double`	8 bytes	$4.9406564584124 6544 \times 10^{-324}$ to $1.7976931348623 1570 \times 10^{308}$

Table 2.6 Floating-Point Data Types

Floating-Point Types (continued)

- `float literal`: Append an `f` or `F` to the number
- `long double literal`: Append an `l` or `L` to the number
  - Examples:

```
9.234      // a double literal
9.234F     // a float literal
9.234L     // a long double literal
```

Arithmetic Operations (continued)

Operation	Operator
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulus division	%

Arithmetic Operations (continued)

Program 2.6

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

int main()
{
   cout << "15.0 plus 2.0 equals "       << (15.0 + 2.0) << endl
        << "15.0 minus 2.0 equals "      << (15.0 - 2.0) << endl
        << "15.0 times 2.0 equals "      << (15.0 * 2.0) << endl
        << "15.0 divided by 2.0 equals " << (15.0 / 2.0) << endl;

   return 0;
}
```

Expression Types

- **Expression:** Any combination of operators and operands that can be evaluated to yield a value
- If all operands are the same data type, the expression is named by the data type used (integer expression, floating-point expression, etc.)
- **Mixed-mode expression:** Contains integer and floating-point operands
  - Yields a double-precision value

Integer Division

- Integer division: Yields an integer result
  - Any fractional remainders are dropped (truncated)
  - Example: `15/2` yields `7`
- Modulus (remainder) operator: Returns only the remainder
  - Example: `9 % 4` yields `1`

Operator Precedence and Associativity (continued)

- Expressions with multiple operators are evaluated by using the rules of precedence of operators:
  - All negations occur first
  - Multiplication, division, and modulus are next, from left to right
  - Addition and subtraction are last, from left to right

Variables and Declaration Statements

- **Memory address:** All integer, float-point, and other values used in a program are stored and retrieved from the computer's memory
- Each memory location has a unique address

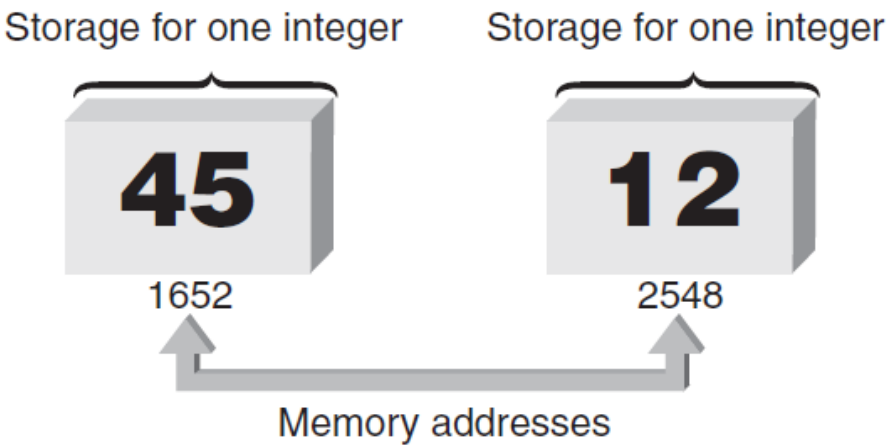


Figure 2.8 Enough storage for two integers

Variables and Declaration Statements (continued)

- **Variable:** Symbolic identifier for a memory address where data can be held
- Use identifier naming rules for variable names

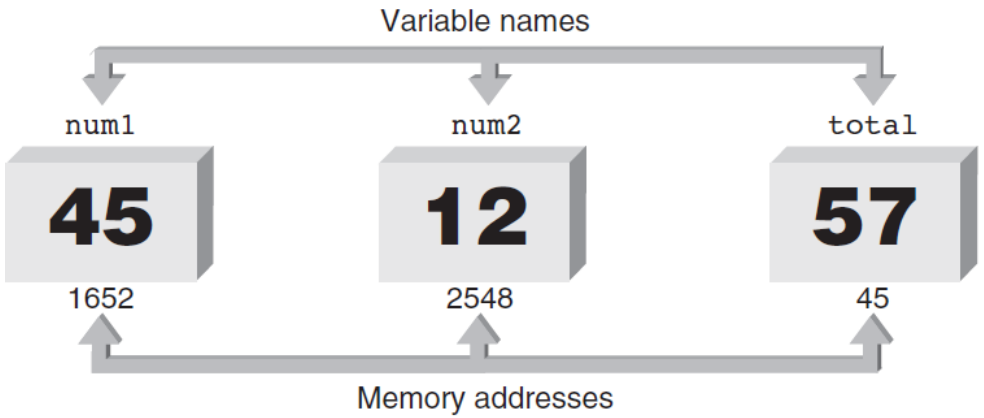


Figure 2.9 Naming storage locations

Variables and Declaration Statements (continued)

- **Assignment statement:** Used to store a value into a variable
- Value of the expression on the right side of the assignment operator is assigned to the memory location of the variable on the left side of the assignment operator
  - Examples:

```
num1 = 45;
num2 = 12;
total = num1 + num2;
```

Variables and Declaration Statements (continued)

- **Declaration statement:** Specifies the data type and identifier of a variable; sets up the memory location
  - Syntax: *`dataType variableName;`*
- Data type is any valid C++ data type referred to previously (primitive or class type)
  - Example: `int sum;`
- Declarations may be used anywhere in a function
  - Usually grouped after the opening brace

Variables and Declaration Statements (continued)

- **Character variables:** Declared using the **`char`** keyword
- Multiple variables of the same data type can be declared in a single declaration statement
  - Example:

```
double grade1, grade2, total, average;
```

- Variables can be initialized (given a value) in a declaration
  - Example:

```
double grade1 = 87.0
```

- A variable must be declared before it is used

Variables and Declaration Statements (continued)

Program 2.7a

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

int main()
{
   double grade1 = 85.5;
   double grade2 = 97.0;
   double total, average;

   total = grade1 + grade2;
   average = total/2.0; // divide the total by 2.0
   cout << "The average grade is " << average << endl;

   return 0;
}
```

Memory Allocation

- **Definition statement**: A declaration that defines how much memory is needed for data storage
- Three items associated with each variable:
  - Data type
  - Actual value stored in the variable (its contents)
  - Memory address of the variable
- Address operator (**&**) provides the variable's address

Memory Allocation (continued)

- Declaring a variable causes memory to be allocated based on the data type

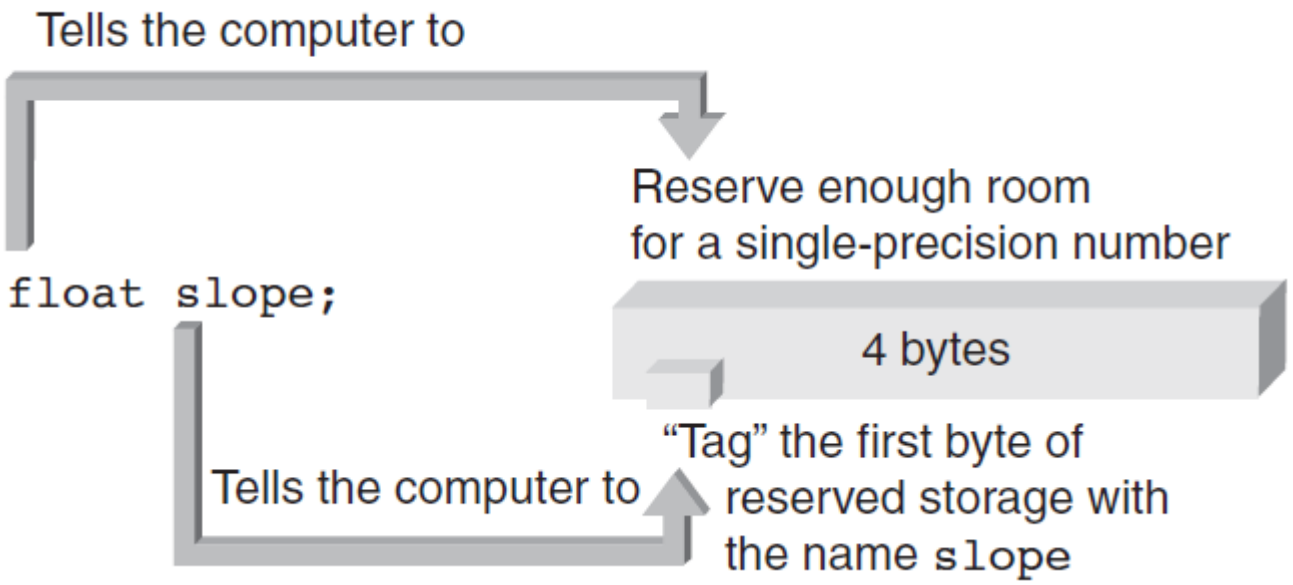


Figure 2.10b Defining the floating-point variable named `slope`

Memory Allocation (continued)

Program 2.10

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

int main()
{
   int num;

   num = 22;
   cout << "The value stored in num is " << num << endl;
   cout << "The address of num = " << &num << endl;

   return 0;
}
```

Symbolic Constants

- **Symbolic constant:** Constant value that is declared with an identifier using the **`const`** keyword
- A constant's value may not be changed

  Example:

```
const int MAXNUM = 100;
```

- Good programming places statements in appropriate order

Symbolic Constants (continued)

- Proper placement of statements:

```
preprocessor directives
int main()
{
        symbolic constants
        main function declarations

        other executable statements
        return value
}
```

Using Mathematical Library Functions (continued)

Function Name	Description	Returned Value
`abs(a)`	Absolute value	Same data type as argument
`pow(a1,a2)`	`a1` raised to the `a2` power	Same data type as argument `a1`
`sqrt(a)`	Square root of a real number	Double-precision
`sin(a)`	Sine of `a` (`a` in radians)	Double
`cos(a)`	Cosine of `a` (`a` in radians)	Double
`tan(a)`	Tangent of `a` (`a` in radians)	Double
`log(a)`	Natural logarithm of `a`	Double
`log10(a)`	Common log (base 10) of `a`	Double
`exp(a)`	`e` raised to the `a` power	Double

Table 3.5 Common C++ Functions

Using Mathematical Library Functions (continued)

- To use a math function, give its name and pass the input arguments within parentheses
- Expressions that can be evaluated to a value can be passed as arguments

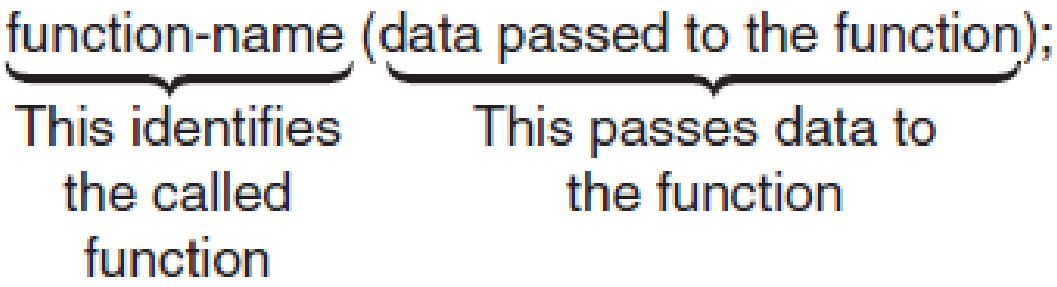


Figure 3.10 Using and passing data to a function

- Function calls can be nested. Eg. **sqrt(sin(abs(angle)))**

A Case Study: Radar Speed Trap

A highway-patrol speed-detection radar emits a beam of microwaves at a frequency designated as f_e. The beam is reflected off an approaching car, and the radar unit picks up and analyzes the reflected beam's frequency, f_r. The reflected beam's frequency is shifted slightly from f_e to f_r because of the car's motion. The relationship between the speed of the car, v, in miles per hour (mph), and the two microwave frequencies is

$$v = (6.685 \times 10^8)\left(\frac{f_r - f_e}{f_r + f_e}\right) mph$$

A Case Study: Radar Speed Trap

- Step 1: Analyze the Problem
  - Understand the desired outputs
  - Determine the required inputs
- Step 2: Develop a Solution
  - Determine the algorithms to be used
  - Use top-down approach to design
- Step 3: Code the Solution
- Step 4: Test and Correct the Program

A Case Study: Radar Speed Trap (continued)

- Analyze the Problem
  - Output: Speed of the car (speed)
  - Inputs: Emitted frequency (fe) and received frequency (fr)
- Develop a Solution
  - Algorithm:
    - Assign values to fe and fr
    - Calculate and display speed using formula

A Case Study: Radar Speed Trap (continued)

Program 2.11

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

int main()
{
  double speed, fe, fr;

  fe = 2e10;
  fr = 2.0000004e10;

  speed = 6.685e8 * (fr - fe) / (fr + fe);
  cout << "The speed is " << speed << " miles/hour " << endl;

  return 0;
}
```

A Case Study: Radar Speed Trap (continued)

- Test and Correct the Program
  - Verify that the calculation and displayed value agree with the hand calculation
  - Use the program with different values of received frequencies
  - Use a wide range of valid and invalid input values to test the behaviour of the program
  - Present your testing using screen-shots

Common Programming Errors

- Omitting the parentheses after **`main()`**
- Omitting or incorrectly typing the opening brace, {, or the closing brace, }, that signifies the start and end of a function body
- Misspelling the name of an object or function
- Forgetting to enclose a string sent to **`cout`** with quotation marks
- Omitting a semicolon at end of a statement

Common Programming Errors (continued)

- Adding a semicolon at end of **`#include`** statement
- Missing a semicolon at the end of **`using namespace std`** statement
- Missing **`\n`** to indicate new line
- Substituting letter O for zero and vice versa
- Failing to declare all variables
- Storing an incorrect data type into a variable
- Attempting to use a variable with no value
- Dividing integer values incorrectly
- Mixing data types in the same expression