

ELEG 1043

Computer Applications in Engineering





Chapter 3: Assignment, Formatting, and Interactive Input

C++ FOR ENGINEERS
AND SCIENTISTS

Acknowledgement

- Some of the slides or images are from various sources. The copyright of those materials belongs to their original owners.

Objectives

In this chapter, you will learn about:

- Assignment operations
- Formatting numbers for program output
- Using mathematical library functions
- Program input using the **cin** object
- Symbolic constants
- A case study involving acid rain
- Common programming errors

Objectives

In this chapter, you will learn about:

- Assignment operations
- Formatting numbers for program output
- Using mathematical library functions
- Program input using the `cin` object
- Symbolic constants
- A case study involving acid rain
- Common programming errors

Objectives

In this chapter, you will learn about:

- Assignment operations
- Formatting numbers for program output
- Using mathematical library functions
- Program input using the **cin** object
- Symbolic constants
- A case study involving acid rain
- Common programming errors

Using Mathematical Library Functions

- C++ has **preprogrammed** mathematical functions that can be included in a program
- You must include the **cmath** header file:

#include <cmath>

- Math functions require **one or more arguments** as input, but will return **only one value**
- All functions are overloaded, and can be used with integer and real arguments

Using Mathematical Library Functions (continued)

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

The diagram shows the syntax of a function call: `function-name (data passed to the function);`. A bracket under `function-name` points to the text "This identifies the called function". Another bracket under `(data passed to the function);` points to the text "This passes data to the function".

`function-name` (data passed to the function);

This identifies the called function This passes data to the function

Figure 3.10 Using and passing data to a function

- Function calls can be nested
 - Example: **`sqrt(sin(abs(theta)))`**

Using Mathematical Library Functions (continued)



Program 3.9

```
#include <iostream>    // this line can be placed second instead of first
#include <cmath>        // this line can be placed first instead of second
using namespace std;

int main()
{
    int height;
    double time;

    height = 800;
    time = sqrt(2 * height / 32.2);
    cout << "It will take " << time << " seconds to fall "
         << height << " feet.\n";

    return 0;
}
```

Program Input Using **cin**

- **cin Object:** Allows **data entry** to a running program
- Use of the **cin** object causes the program to **wait for input from the keyboard**
- When keyboard entry is complete, the program **resumes** execution, **using the entered data**
- An output statement preceding the **cin** object statement provides a **prompt** to the user

Program Input Using **cin** (continued)



Program 3.12

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

int main()
{
    double num1, num2, product;

    cout << "Please type in a number: ";
    cin >> num1;
    cout << "Please type in another number: ";
    cin >> num2;
    product = num1 * num2;
    cout << num1 << " times " << num2 << " is " << product << endl;

    return 0;
}
```

Program Input Using `cin` (continued)

- `cin` can accept **multiple input values** to be **stored** in **different variables**
- Multiple numeric input values must be separated by spaces

Example:

```
cin >> num1 >> num2
```

with keyboard entry: `0.052 245.79`

Program Input Using **cin** (continued)



Program 3.13

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

int main()
{
    int num1, num2, num3;
    double average;

    cout << "Enter three integer numbers: ";
    cin  >> num1 >> num2 >> num3;
    average = (num1 + num2 + num3) / 3.0;
    cout << "The average of the numbers is " << average << endl;

    return 0;
}
```

Program Input Using `cin` (continued)

- **User-input validation:** The process of ensuring that data entered by the user **matches the expected data type**

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
```

```
}
```

A Closer Look: Programming Errors

- Program errors may be detected in four ways:
 - **Before** a program is compiled (desk checking)
 - While it is being **compiled** (compile-time errors)
 - While it is being **run** (run-time errors)
 - While **examining the output** after completion
- Errors may be:
 - Syntax errors
 - **typos** in the source code
 - Logic errors
 - **often difficult** to detect and difficult to find the source

Common Programming Errors

- Failure to **declare or initialize variables** before use
- Failure to include the **preprocessor statement** when using a C++ preprogrammed library
 - **#include "stdafx.h"**
- Passing the **incorrect number** or type of arguments to a function
- Applying **increment or decrement operator** to an expression instead of an individual variable
 - **++ (a + b) , -- (b + 3)**

Common Programming Errors (continued)

- Failure to separate all variables passed to `cin` with the extraction symbol `>>`
 - `int a = 0, b = 1;`
 - `cin>>ab;`
- Failure to test thoroughly

Summary

- Expression: A sequence of one or more operands separated by operators
- Assignment operator: =
- Increment operator: ++
- Decrement operator: --

Summary (continued)

- Use `#include <cmath>` for math functions
- Arguments to a function must be passed in the **proper number, type, and order**
- `cin` object provides data input from a keyboard; program is suspended until the input arrives
- Constants are named values **that do not change**