Chapter 2:
C++ Fundamentals

Starting Out with Games and Graphics in C++

Second Edition

by Tony Gaddis

2.1 The Parts of a C++ Program

```cpp
#include <iostream> // include directive
using namespace std; // which namespace to use

int main() // beginning of function named main
{
    // beginning of block for main
    return 0; // send 0 to operating system

    // end of block for main
}
```
2.2 Displaying Screen Output

• The cout Statement:
  – Displays output on the computer screen
  – You use the stream insertion operator << to send output to cout:

    cout << "Hello world";

• Can be used to send more than one item to cout:

    cout << "Hello " << "world";
    Or:

    cout << "Hello ";
    cout << "world";
2.2 Displaying Screen Output

• This produces one line of output:

    ```cpp
    cout << "Programming is ";
    cout << "fun!";
    ```

• The `endl` Manipulator:
  - You can use the `endl` manipulator to start a new line of output. This will produce two lines of output:

    ```cpp
    cout << "Programming is" << endl;
    cout << "fun!";
    ```
2.2 Displaying Screen Output

```cpp
cout << "Programming is" << endl;
cout << "fun!";
```

- You do NOT put quotation marks around `endl`.
- The last character in `endl` is a lowercase L, not the number 1.

`endl` — This is a lowercase L
2.2 Displaying Screen Output

• The newline escape sequence:
  – You can also use the \n escape sequence to start a new line of output. This will produce two lines of output:

    cout << "Programming is\n";
    cout << "fun!";

    Notice that the \n is INSIDE the string.
2.3 More About the `#include` Directive

- Inserts the contents of another file into the program
- This is a preprocessor directive, not part of C++ language
- `#include` lines are not seen by the compiler
- Do not place a semicolon at end of `#include` line

2.4 A First Look at Variables

- **Variable**: a storage location in memory
  - Has a name and a type of data it can hold
  - Must be declared before it can be used:

    ```
    int item;
    ```
2.4 A First Look at Variables

• **Data Type:**
  A variable’s data type indicates the type of data that the variable will hold.

• **Variable Name:**
  A variable name identifies a variable in the program code. When naming a variable, you should always choose a meaningful name that indicates what the variable is used for.

  A variable name should represent the purpose of the variable. For example:

  `itemsOrdered`

  The purpose of this variable is to hold the number of items ordered.
• Variable Naming Rules:
  – The first character of an identifier must be an alphabetic character or and underscore (_),
  – After the first character you may use alphabetic characters, numbers, or underscore characters.
  – Upper- and lowercase characters are distinct
2.4 A First Look at Variables

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>VALID?</th>
<th>REASON IF INVALID</th>
</tr>
</thead>
<tbody>
<tr>
<td>totalSales</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>total_Sales</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>total.Sales</td>
<td>No</td>
<td>Cannot contain .</td>
</tr>
<tr>
<td>4thQtrSales</td>
<td>No</td>
<td>Cannot begin with digit</td>
</tr>
<tr>
<td>totalSale$</td>
<td>No</td>
<td>Cannot contain $</td>
</tr>
</tbody>
</table>

- The **int**, **float**, and **double** data types
  - Most of the time, you will use the **int** data type to store integers and the **float** and **double** data type to store real numbers.
2.4 A First Look at Variables

• An assignment statement uses the = operator to store a value in a variable.

\[ \text{item} = 12; \]

• This statement assigns the value 12 to the \text{item} variable.

2.4 A First Look at Variables

• You can display the value of a variable with a cout statement:

\[
\begin{align*}
\text{int speed;} \\
\text{speed} & \text{ = 60;} \\
\text{cout} & \text{ << speed} \text{ << endl;} \\
\end{align*}
\]
2.4 A First Look at Variables

- The variable receiving the value must appear on the left side of the = operator.
- This will NOT work:

// ERROR!
60 = speed;

2.4 A First Look at Variables

- Declaring Variables Inside a Function
  - Variables that are declared inside a function are known as local variables.
  - A local variable belongs to the function in which it is declared, and only statements inside that function can access the variable.
2.4 A First Look at Variables

• Declaring Variables Before Using Them
  – A variable declaration causes a variable to be created in memory.
  – A variable's declaration must appear before any other statements in the function that use the variable.

• Variable Initialization:
  – To initialize a variable means to assign it a value when it is defined:

    int length = 12;

  – Can initialize some or all variables:

    int length = 12, width = 5, area;
2.4 A First Look at Variables

• Numeric Literals
  – A numeric literal is a number that is written into a program’s code.

```java
int speed;
speed = 60;
```

• Numeric Literals
  – **int literals**: If a numeric literal is written as an integer (without a decimal point), and fits within the range of an int, it is treated as an int.
  – **double literals**: If a numeric literal is written with a decimal point, and fits within the range of a double, it is treated as a double.
  – You can assign int literals to int variables, but you cannot assign double literals to int variables.
  – You can assign either int or double literals to double variables.
2.4 A First Look at Variables

• Variables hold only one value at a time.

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

int main()
{
    double price = 29.95;
    cout << price << endl;
    price = 12.95;
    cout << price << endl;
    return 0;
}
```

Program Output
29.95
12.95

2.4 A First Look at Variables

• **Scope:**
  
  - The part of the program in which the variable can be accessed
  
  - A local variable’s scope begins at the variable’s declaration and ends at the end of the function in which the variable is declared.
2.4 A First Look at Variables

- **Scope:**
  - A variable cannot be used before it is declared.
  - A local variable cannot be accessed by code outside the function in which it is declared.
  - You cannot declare two variables with the same name in the same scope.

2.5 Reading Keyboard Output

- **The cin Statement**
  - Like cout, requires iostream file
  - Used to read input from keyboard
  - Input is retrieved from cin with `>>`
  - Input is stored in one or more variables
2.5 Reading Keyboard Output

#include <iostream>
using namespace std;

int main()
{
    int age;
    cin >> age;
    cout << "What is your age? " << age << " years old!" << endl;
    return 0;
}

Program Output with Example Input Shown in Bold
What is your age? 25 [Enter]
I would never have guessed that you are 25 years old!

2.5 Reading Keyboard Output

#include <iostream>
using namespace std;

int main()
{
    double payRate;
    int hours;
    cin >> payRate;
    cout << "Enter the number of hours worked: " << hours << endl;
    cin >> hours;
    cout << "Here are the values that you entered: " << endl;
    cout << "Hourly pay rate: " << payRate << endl;
    cout << "Hours worked: " << hours << endl;
    return 0;
}

Program Output with Example Input Shown in Bold
Enter your hourly pay rate: 25 [Enter]
Enter the number of hours worked: 40 [Enter]
Here are the values that you entered:
Hourly pay rate: 25
Hours worked: 40
2.5 Reading Keyboard Output

• Prompting the User
  – Getting keyboard input from the user is normally a two-step process:
  1. Display a prompt on the screen.
  2. Read a value from the keyboard.

2.6 Comments, Blank Lines, and Indentation

• Comments
  – Used to document parts of the program
  – Intended for persons reading the source code of the program:
    • Indicate the purpose of the program
    • Describe the use of variables
    • Explain complex sections of code
  – Are ignored by the compiler
2.6 Comments, Blank Lines, and Indentation

• Single-Line Comments

Begin with // through to the end of line:

```c
int length = 12;  // length in inches
int width = 15;   // width in inches
int area;        // calculated area

// calculate rectangle area
area = length * width;
```
2.6 Comments, Blank Lines, and Indentation

- Multiline Comments
  - Begin with /*
  end with */
  - Can span multiple lines:
    /* this is a multi-line comment */
  - Can begin and end on the same line:
    int area;   /* calculated area */

2.7 Performing Calculations and Working with Numbers

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>OPERATION</th>
<th>EXAMPLE</th>
<th>VALUE OF x</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>x = 7 + 3;</td>
<td>10</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>x = 7 - 3;</td>
<td>4</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>x = 7 * 3;</td>
<td>21</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>x = 7 / 3;</td>
<td>2</td>
</tr>
<tr>
<td>%</td>
<td>Remainder, or modulus</td>
<td>x = 7 % 3;</td>
<td>1</td>
</tr>
</tbody>
</table>
2.7 Performing Calculations and Working with Numbers

• **Order of Operations:**
  1. Perform any operations that are enclosed in parentheses.
  2. Perform any multiplications, divisions, or modulus operations as they appear from left to right.
  3. Perform any additions or subtractions as they appear from left to right.

```
outcome = 12 + 6 / 3;
outcome = 12 + 2
outcome = 14
```
2.7 Performing Calculations and Working with Numbers

- Order of Operations

<table>
<thead>
<tr>
<th>Table 2-5 Some expressions and their values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression</td>
</tr>
<tr>
<td>-------------------------------</td>
</tr>
<tr>
<td>$5 + 2 \times 4$</td>
</tr>
<tr>
<td>$10 / 2 - 3$</td>
</tr>
<tr>
<td>$8 + 12 \times 2 - 4$</td>
</tr>
<tr>
<td>$6 - 3 \times 2 + 7 - 1$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2-6 More expressions and their values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression</td>
</tr>
<tr>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>$(5 + 2) \times 4$</td>
</tr>
<tr>
<td>$10 \div (5 - 3)$</td>
</tr>
<tr>
<td>$8 + 12 \times (6 - 2)$</td>
</tr>
<tr>
<td>$(6 - 3) \times (2 + 7) / 3$</td>
</tr>
</tbody>
</table>
2.7 Performing Calculations and Working with Numbers

- **The Division Operator (\(/\))**
- Performs integer division if both operands are integers
  
  ```cpp
  cout << 13 / 5;    // displays 2
  cout << 91 / 7;    // displays 13
  ```
- If either operand is floating point, the result is floating point
  
  ```cpp
  cout << 13 / 5.0;  // displays 2.6
  cout << 91.0 / 7;  // displays 13.0
  ```

- **Remainder, or Modulus Operator (\(\%\))**
- Computes the remainder resulting from integer division
  
  ```cpp
  cout << 13 % 5;    // displays 3
  ```
- \(\%\) requires integers for both operands
  
  ```cpp
  cout << 13 % 5.0; // error
  ```
Look at the following statement:

```
sum = sum + 1;
```

This adds 1 to the variable `sum`. 
2.7 Performing Calculations and Working with Numbers

• Other Similar Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>What It Does</th>
<th>Value of ( x ) After the Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = x + 4; )</td>
<td>Adds 4 to ( x )</td>
<td>10</td>
</tr>
<tr>
<td>( x = x - 3; )</td>
<td>Subtracts 3 from ( x )</td>
<td>3</td>
</tr>
<tr>
<td>( x = x * 10; )</td>
<td>Multiplies ( x ) by 10</td>
<td>60</td>
</tr>
<tr>
<td>( x = x / 2; )</td>
<td>Divides ( x ) by 2</td>
<td>3</td>
</tr>
<tr>
<td>( x = x % 4 )</td>
<td>Makes ( x ) the remainder of ( x / 4 )</td>
<td>2</td>
</tr>
</tbody>
</table>

The combined assignment operators are made to perform these types of operations.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example Usage</th>
<th>Equivalence</th>
</tr>
</thead>
<tbody>
<tr>
<td>( += )</td>
<td>( x += 5; )</td>
<td>( x = x + 5; )</td>
</tr>
<tr>
<td>( -= )</td>
<td>( y -= 2; )</td>
<td>( y = y - 2; )</td>
</tr>
<tr>
<td>( *= )</td>
<td>( z *= 10; )</td>
<td>( z = z * 10; )</td>
</tr>
<tr>
<td>( /= )</td>
<td>( a /= b; )</td>
<td>( a = a / b; )</td>
</tr>
<tr>
<td>( %= )</td>
<td>( c %= 3; )</td>
<td>( c = c % 3; )</td>
</tr>
</tbody>
</table>
2.7 Performing Calculations and Working with Numbers

• Mixed-Type Expressions
  – When an operation is performed on two int values, the result will be an int.
  – When an operation is performed on two double values, the result will be a double.
  – When an operation is performed on an int and a double, the int value will be temporarily converted to a double, and the result of the operation will be a double.

• Type Cast Expressions
  – Used for manual data type conversion:

    ```
    double dval = 3.7;
    int ival = static_cast<int>(dval);
    ```

  – Useful for floating point division using ints:

    ```
    int a = 10, b = 2;
    double m;
    m = static_cast<double>a / b;
    ```
2.8 Named Constants

- **Named constant**: a name that represents a value that cannot be changed during the program’s execution.
- Used for representing constant values with descriptive names:
  ```
  const double TAX_RATE = 0.0675;
  const int NUM_STATES = 50;
  ```
- Often named in uppercase letters
2.9 Math Functions in the Standard Library

- Require `cmath` header file
- Most of the functions take `double` as input, return a `double`
- Example: The `pow` function raises a number to a power:

\[
\text{area} = \text{pow}(4.0, 2.0);
\]
2.9 Math Functions in the Standard Library

area = pow(4.0, 2.0);

1st argument

2nd argument

---

Program 2.17 (PowFunction.cpp)

1 // This program demonstrates the pow function.
2 #include <iostream>
3 #include <cmath>
4 using namespace std;
5
6 int main()
7 {
8    const double PI = 3.14159;
9
10    // Get the circle's radius.
11    cout << "Enter the circle's radius: ";
12    cin >> radius;
13
14    // Calculate and display the circle's area.
15    area = PI * pow(radius, 2.0);
16    cout << "The circle's area is: " << area << endl;
17    return 0;
18 }

Program Output with Example Input Shown in Bold

Enter the circle's radius: 10 [Enter]
The circle's area is: 314.159
2.9 Math Functions in the Standard Library

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acos(x)</td>
<td>Returns the arc cosine of x, in radians. The argument and the return value are doubles.</td>
</tr>
<tr>
<td>asin(x)</td>
<td>Returns the arc sine of x, in radians. The argument and the return value are doubles.</td>
</tr>
<tr>
<td>atan(x)</td>
<td>Returns the arc tangent of x, in radians. The argument and the return value are doubles.</td>
</tr>
<tr>
<td>sin(x)</td>
<td>Returns the sine of x in radians. The argument and the return value are doubles.</td>
</tr>
<tr>
<td>sqrt(x)</td>
<td>Returns the square root of x. The argument and the return value are doubles.</td>
</tr>
<tr>
<td>tan(x)</td>
<td>Returns the tangent of x in radians. The argument and the return value are doubles.</td>
</tr>
</tbody>
</table>

See Table 2-8 (page 75) for more.

2.10 Working with Strings

• The **string class requires this** `#include` directive:
  ```cpp
  #include <string>
  ```

• Can declare **string objects in programs**:  
  ```cpp
  string firstName, lastName;
  ```

• Can receive values with assignment operator:
  ```cpp
  firstName = "George";
  lastName = "Washington";
  ```

• Can be displayed via **cout**
  ```cpp
  cout << firstName << " " << lastName;
  ```
2.10 Working with Strings

Program 2-18

```
1 // This program demonstrates the string class.
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 int main()
7 {
8     string movieTitle = "Wheels of Fury";
9
10    cout << "My favorite movie is " << movieTitle << endl;
11    return 0;
12 }
```

Program Output

My favorite movie is Wheels of Fury

Program 2-19

```
1 // This program demonstrates string input.
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 int main()
7 {
8     string name;
9
10    // Get the user's name.
11    cout << "Enter your name: ";
12    cin >> name;
13
14    // Display the user's name.
15    cout << "Your name is: 
16    << name << endl;
17    return 0;
18 }
```

Program Output with Example Input Shown in Bold

Enter your name: Liza [Enter]
Your name is: Liza
2.10 Working with Strings

- Use the concatenation operator (+) to join multiple strings:

```java
string str1 = "Hello ";
string str2;
string str3 = "World";
string str4 = "People";
str2 = str1 + str3;  // str2 now holds "Hello World"
str1 = str1 + str4;  // str1 now holds "Hello People"
```

2.11 The char Data Type

- Used to hold a single character:
  ```java
  char letter;
  ```

- Character literals are enclosed in single-quotiation marks:
  ```java
  letter = 'g';
  ```
2.11 The `char` Data Type

- You cannot store string literals in a char variable:
  ```
  letter = "g";  // ERROR!
  ```

This is a string literal. It cannot be assigned to a char variable.