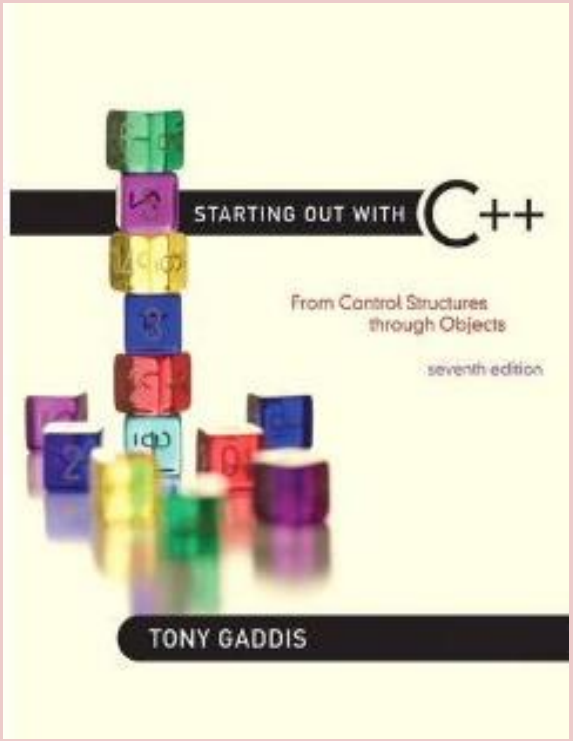
# Software Design & Programming I




*Starting Out with C++ (From Control Structures through Objects) 7th Edition*
*Written by: Tony Gaddis*
*Pearson - Addison Wesley*
*ISBN: 13-978-0-132-57625-3*

# Chapter 2
# (Part II)
# Introduction to C++

# The char Data Type (Sample Program)

**Program 2-11**

```
1  // This program demonstrates the close relationship between
2  // characters and integers.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     char letter;
9
10    letter = 65;
11    cout << letter << endl;
12
13    letter = 66;
14    cout << letter << endl;
15    return 0;
16 }
```

**Program Output**

```
A
B
```

# Character and String Constants

**Program 2-12**

```
1  // This program uses character constants.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     char letter;
8
9     letter = 'A';
10    cout << letter << endl;
11
12    letter = 'B';
13    cout << letter << endl;
14    return 0;
15 }
```

**Program Output**

```
A
B
```

# The char Data Type

Program 2-12 assigns character constants to the variable letter. Anytime a program works with a character, it internally works with the code used to represent that character, so this program is still assigning the values 65 and 66 to letter.

Character constants can only hold a single character. To store a series of characters in a constant we need a string constant. In the following example, 'H' is a character constant and "Hello" is a string constant. Notice that a character constant is enclosed in single quotation marks whereas a string constant is enclosed in double quotation marks.

**cout << ‘H’ << endl;**

**cout << “Hello” << endl**;

# The char Data Type

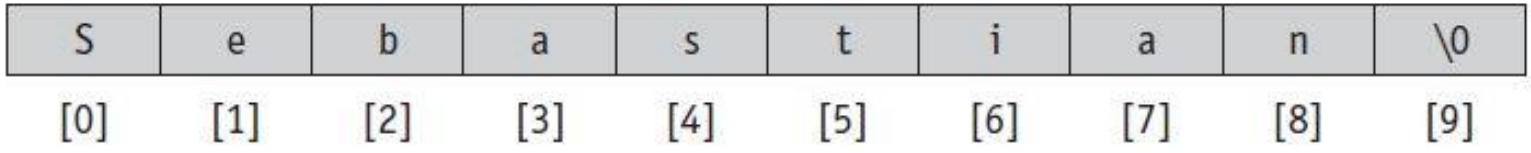
Strings, which allow a series of characters to be stored in consecutive memory locations, can be virtually any length. This means that there must be some way for the program to know how long the string is. **In C++ this is done by appending an extra byte to the end of string constants.** In this last byte, the number 0 is stored. It is called the null terminator or null character and marks the end of the string.

Don’t confuse the null terminator with the character '0'. If you look at Appendix A you will see that the character '0' has ASCII code 48, whereas the null terminator has ASCII code 0. If you want to print the character 0 on the screen, you use ASCII code 48. If you want to mark the end of a string, you use ASCII code 0.

# The char Data Type

Let's look at an example of how a string constant is stored in memory. Figure 2-5 depicts the way the string "Sebastian" would be stored.

**Figure 2-5**




First, notice the quotation marks are not stored with the string. They are simply a way of marking the beginning and end of the string in your source code. Second, notice the very last byte of the string. It contains the null terminator, which is represented by the \0 character. The addition of this last byte means that although the string is 9 characters long, it occupies 10 bytes of memory.

# The char Data Type

The null terminator doesn't print on the screen when you display a string, but nevertheless, it is there silently doing its job.

**NOTE: C++ automatically places the null terminator at the end of string constants.**

# The char Data Type

Let’s compare the way a string and a char are stored. Suppose you have the constants 'A‘ and "A" in a program.

**Figure 2-6**


As you can see, 'A' is a 1-byte element and "A" is a 2-byte element. Since characters are really stored as ASCII codes, Figure 2-7 shows what is actually being stored in memory.

**Figure 2-7**

# The char Data Type

Because a char variable can only hold a single character, it can be assigned the character 'A', but not the string "A".

```
char letterOne = 'A'; // This will work.
char letterTwo = "A"; // This will NOT work!
```

You have learned that some strings look like a single character but really aren’t. It is also possible to have a character that looks like a string. A good example is the newline character, \n. Although it is represented by two characters, a slash and an n, it is internally represented as one character. In fact, all escape sequences, internally, are just 1 byte.

# The char Data Type

Program 2-13 shows the use of \n as a character constant, enclosed in single quotation marks.

**Program 2-13**

```
1  // This program uses character constants.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     char letter;
8
9     letter = 'A';
10    cout << letter << '\n';
11
12    letter = 'B';
13    cout << letter << '\n';
14    return 0;
15 }
```

**Program Output**

```
A
B
```

# The C++ string Class

Standard C++ provides a special data type for storing and working with strings.

Because a char variable can store only one character in its memory location, another data type is needed for a variable able to hold an entire string. While C++ does not have a built-in data type able to do this, Standard C++ provides something called the string class that allows the programmer to create a string type variable.

# Using the string Class

The first step in using the string class is to #include the string header file. This is accomplished with the following preprocessor directive:

**#include <string>**

The next step is to define a string type variable, called a string object. Defining a string object is similar to defining a variable of a primitive type. For example, the following statement defines a string object named movieTitle.

**string movieTitle;**

# Using the string Class

You can assign a string literal to movieTitle with the assignment operator:

```
movieTitle = "Wheels of Fury";
```

The contents of movieTitle can be displayed on the screen with cout, as shown in the next statement:

```
cout << "My favorite movie is " << movieTitle << endl;
```

# Using the string Class

**Program 2-14**

```
1  // This program demonstrates the string class.
2  #include <iostream>
3  #include <string>              // Required for the string class.
4  using namespace std;
5
6  int main()
7  {
8     string movieTitle;
9
10    movieTitle = "Wheels of Fury";
11    cout << "My favorite movie is " << movieTitle << endl;
12    return 0;
13 }
```

**Program Output**

```
My favorite movie is Wheels of Fury
```

As you can see, working with string objects is similar to working with variables of other types.

# Floating-Point Data Types

Floating-point data types are used to define variables that can hold real numbers.

Whole numbers are not adequate for many jobs. If you are writing a program that works with dollar amounts or precise measurements, you need a data type that allows fractional values. In programming terms, these are called floating-point numbers.

Internally, floating-point numbers are stored in a manner similar to scientific notation. Take the number 47,281.97. In scientific notation this number is $4.728197 \times 10^4$. ($10^4$ is equal to 10,000, and $4.728197 \times 10{,}000$ is 47,281.97.) The first part of the number, 4.728197, is called the mantissa. The mantissa is multiplied by a power of 10.

# Floating-Point Data Types

**Table 2-7** Floating-Point Representations

| Decimal Notation | Scientific Notation | E Notation |
|---|---|---|
| 247.91 | $2.4791 \times 10^2$ | 2.4791E2 |
| 0.00072 | $7.2 \times 10^{-4}$ | 7.2E–4 |
| 2,900,000 | $2.9 \times 10^6$ | 2.9E6 |

In C++ there are three data types that can represent floating-point numbers. They are

float

double

long double .

# Floating-Point Data Types

The float data type is considered single precision. The double data type is usually twice as big as float, so it is considered double precision. As you've probably guessed, the long double is intended to be larger than the double. The exact sizes of these data types is dependent on the computer you are using. The only guarantees are

- A double is at least as big as a float.
- A long double is at least as big as a double.

**Table 2-8** Floating-Point Data Types on PCs

| Data Type | Key Word | Size | Range | Significant Digits |
|---|---|---|---|---|
| Single precision | `float` | 4 bytes | Numbers between ±3.4E-38 and ±3.4E38 | 7 |
| Double precision | `double` | 8 bytes | Numbers between ±1.7E-308 and ±1.7E308 | 16 |
| Long double precision | `long double` | 8 bytes* | Numbers between ±1.7E-308 and ±1.7E308 | 16 |

# Floating-Point Data Types

You will notice there are no unsigned floating-point data types. On all machines, variables of the float, double, and long double data type can store both positive and negative numbers.

**Program 2-15**

```
1  // This program uses two floating-point data types, float and double.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     float distance = 1.496E8;          // in kilometers
8     double mass = 1.989E30;            // in kilograms
9
10    cout << "The Sun is " << distance << " kilometers away.\n";
11    cout << "The Sun\'s mass is " << mass << " kilograms.\n";
12    return 0;
13 }
```

**Program Output**

```
The Sun is 1.496e+008 kilometers away.
The Sun's mass is 1.989e+030 kilograms.
```

# Floating-Point Constants

Floating-point constants may be expressed in a variety of ways. As shown in Program 2-15, E notation is one method. When you are writing numbers that are extremely large or extremely small, this will probably be the easiest way. E notation numbers may be expressed with an uppercase E or a lowercase e. Notice in the source code the constants were written as 1.496E8 and 1.989E30, but the program printed them as 1.496e+008 and 1.989e+030. The uppercase E and lowercase e are equivalent. The plus sign in front of the exponent is also optional.

You can also express floating-point constants in decimal notation. The constant 1.496E8 could have been written as 149600000.0

# Floating-Point Constants

Obviously the E notation is more convenient for lengthy numbers; but for numbers like 47.39, decimal notation is preferable to 4.739E1.

All of the following floating-point constants are equivalent:

**1.496E8**

**1.496e8**

**1.496E+8**

**1.496e+8**

**149600000.0**

# Floating-Point Constants

Floating-point constants are normally stored in memory as doubles. Just in case you need to force a constant to be stored as a float, you can append the letter F or f to the end of it. For example, the following constants would be stored as float numbers:

**1.2F**

**45.907f**

Because floating-point constants are normally stored in memory as doubles, some compilers issue a warning message when you assign a floating-point constant to a float variable.

# Floating-Point Constants

For example, assuming num is a float, the following statement might cause the compiler to generate a warning message:

**num = 14.725;**

You can suppress the error message by appending the f suffix to the floating-point constant, as shown here:

**num = 14.725f;**

If you want to force a value to be stored as a long double, append an L to it, as shown here:

**1034.56L**

The compiler won't confuse this with a long integer because of the decimal point.

# Assigning Floating-Point Values to Integer Variables

When a floating-point value is assigned to an integer variable, the fractional part of the value (the part after the decimal point) is discarded. This occurs because an integer variable cannot hold any value containing decimals. For example, look at the following code.

```
int number;
number = 7.8;       // Assigns 7 to number
```

This code attempts to assign the floating-point value 7.8 to the integer variable number. Because this is not possible, the value 7 is assigned to number, and the fractional part is discarded. When part of a value is discarded in this manner, the value is said to be truncated.

# Assigning Floating-Point Values to Integer Variables

Assigning a floating-point variable to an integer variable has the same effect. For example, look at the following code.

```
int intVar;
double doubleVar = 7.8;
intVar = doubleVar;        // Assigns 7 to intVar
                           // doubleVar remains 7.8
```

Floating-point variables can hold a much larger range of values than integer variables can. If a floating-point value is stored in an integer variable, and the whole part of the value (the part before the decimal point) is too large for the integer variable, an invalid value will be stored in the integer variable.

# The modulus Operator

```
#include <iostream>

using namespace std;

int main()
{
    int num;
    cin >> num;
    // num % 2 computes the remainder when num is divided by 2
    if ( num % 2 == 0 )
    {
        cout << num << " is even ";
    }

    return 0;
}
```

**PROGRAM #9**

**DUE: THURSDAY, MARCH 21, 2013**

Without using the modulus operation, write a program to determine if an inputted number is even or odd.

# The bool Data Type

Boolean variables are set to either true or false.

Expressions that have a true or false value are called Boolean expressions, named in honor of English mathematician George Boole (1815–1864).

The bool data type allows you to create variables that hold true or false values. Program 2-16 demonstrates the definition and use of a bool variable. Although it appears that it is storing the words true and false, it is actually an integer variable that stores 0 for false and 1 for true, as you can see from the program output.

# The bool Data Type

**Program 2-16**

```
1  // This program uses Boolean variables.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     bool boolValue;
8
9     boolValue = true;
10    cout << boolValue << endl;
11
12    boolValue = false;
13    cout << boolValue << endl;
14    return 0;
15 }
```

**Program Output**

```
1
0
```

# Determining the Size of a Data Type

The sizeof operator may be used to determine the size of a data type on any system.

A special operator called sizeof will report the number of bytes of memory used by any data type or variable. Program 2-17 illustrates its use. The first line that uses the operator is line 9.

**cout << "The size of an integer is " << sizeof(int);**

The name of the data type or variable is placed inside the parentheses that follow the operator. The operator “returns” the number of bytes used by that item. This operator can be used anywhere you can use an unsigned integer, including in mathematical operations.

# The sizeof() Command

**Program 2-17**

```
1  // This program displays the size of various data types.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     long double apple;
8
9     cout << "The size of an integer is " << sizeof(int);
10    cout << " bytes.\n";
11    cout << "The size of a long integer is " << sizeof(long);
12    cout << " bytes.\n";
13    cout << "An apple can be eaten in " << sizeof(apple);
14    cout << " bytes!\n";
15    return 0;
16 }
```

**Program Output**

```
The size of an integer is 4 bytes.
The size of a long integer is 4 bytes.
An apple can be eaten in 8 bytes!
```

# More on Variable Assignments and Initialization

An assignment operation assigns, or copies, a value into a variable. When a value is assigned to a variable as part of the variable's definition, it is called an initialization.

A value is stored in a variable with an *assignment statement*. For example, the following statement copies the value 12 into the variable unitsSold.

**unitsSold = 12;**

The = symbol, as you recall, is called the assignment operator. Operators perform operations on data. The data that operators work with are called operands. The assignment operator has two operands. In the previous statement, the operands are unitsSold and 12.

# More on Variable Assignments and Initialization

It is important to remember that in an assignment statement, C++ requires the name of the variable receiving the assignment to appear on the left side of the operator. The following statement is incorrect.

**12 = unitsSold; // Incorrect!**

It is possible to assign values to variables when they are defined. This is called initialization. When multiple variables are defined in the same statement, it is possible to initialize some of them without having to initialize all of them.

# Assignment Statement Example

**Program 2-18**

```
1  // This program shows variable initialization.
2  #include <iostream>
3  #include <string>
4  using namespace std;
5
6  int main()
7  {
8     string month = "February";      // month is initialized to "February"
9     int year,                       // year is not initialized
10        days = 28;                   // days is initialized to 28
11
12    year = 2007;                    // Now year is assigned a value
13
14    cout << "In "   << year << " " << month
15         << " had " << days << " days.\n";
16
17    return 0;
18 }
```

**Program Output**

```
In 2007 February had 28 days.
```

# Scope

A variable's scope is the part of the program that has access to the variable.

Every variable has a scope. The scope of a variable is the part of the program where it may be used. The rules that define a variable's scope are complex, and we will just introduce the concept here. Later in the book we will cover this topic in more depth.

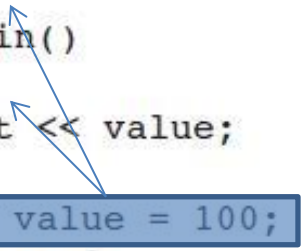
The first rule of scope is that a variable cannot be used in any part of the program before it

is defined.

# Sample Program

**Program 2-19**

```
1  // This program can't find its variable.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     cout << value;     // ERROR! value has not been defined yet!
8
9     int value = 100;
10    return 0;
11 }
```



The program will not work because line 7 attempts to send the contents of the variable value to cout before the variable is defined. The compiler reads a program from top to bottom. If it encounters a statement that uses a variable before the variable is defined, an error will result.

# Arithmetic Operators

There are many operators for manipulating numeric values and performing arithmetic operations.

C++ provides many operators for manipulating data. Generally, there are three types of operators: unary, binary, and ternary. These terms reflect the number of operands an operator requires.

Unary operators only require a single operand. For example, consider the following expression: −5

Of course, we understand this represents the value negative five. The constant 5 is preceded by the minus sign. The minus sign, when used this way, is called the negation operator. Since it only requires one operand, it is a unary operator.

# Arithmetic Operators

Binary operators work with two operands. Ternary operators, as you may have guessed, require three operands.

## Binary Operators

**Table 2-9** Fundamental Arithmetic Operators

| Operator | Meaning | Example |
|---|---|---|
| + | Addition | `total = cost + tax;` |
| – | Subtraction | `cost = total - tax;` |
| * | Multiplication | `tax = cost * rate;` |
| / | Division | `salePrice = original / 2;` |
| % | Modulus | `remainder = value % 3;` |

```
total = 4 + 8;          // total is assigned the value 12
candyBars = 8 – 3       // candyBars is assigned the value 5
points = 3 * 7          // points is assigned the value 21
```

# Arithmetic Operators

The division operator works differently depending on whether its operands are integer or floating-point numbers. When both numbers are integers, the division operator performs integer division. This means that the result is always an integer. If there is any remainder, it is discarded.

```
fullBoxes = 26 / 8;                    // fullBoxes is assigned the value 3
```

The variable fullBoxes is assigned the value 3 because 8 goes into 26 three whole times with a remainder of 2. The remainder is discarded.

# Arithmetic Operators

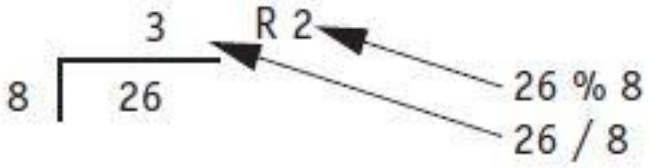
If you want the division operator to perform regular division, you must make sure at least one of the operands is a floating-point number.

```
boxes = 26.0 / 8;          // boxes is assigned the value 3.25
```

The modulus operator computes the remainder of doing an integer divide.

```
leftOver = 26 % 8;         // leftOver is assigned the value 2
```

**Figure 2-8** Integer Divide and Modulus Operations



# Assignment Statement Example

**Program 2-20**

```
1  // This program calculates hourly wages, including overtime.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     double basePayRate     = 18.25,   // Base pay rate
8            overtimePayRate = 27.38,   // Overtime pay rate
9            regularHours    = 40.0,    // Regular hours worked}
10           overtimeHours   = 10,      // Overtime hours worked
11           regularWages,              // Computed regular wages
12           overtimeWages,             // Computed overtime wages
13           totalWages;                // Computed total wages
14
15    // Calculate regular wages
16    regularWages = basePayRate * regularHours;
17
18    // Calculate overtime wages
19    overtimeWages = overtimePayRate * overtimeHours;
20
21    // Calculate total wages
22    totalWages = regularWages + overtimeWages;
23
24    // Display total wages
25    cout << "Wages for this week are $" << totalWages << endl;
26    return 0;
27 }
```

**Program Output**

```
Wages for this week are $1003.8
```

# Program Assignment #10

**Due date: March 21, 2013**

Referring to Program 2-20, write a program that will calculate a person's total commission. The person sold 32 items with a commission of $12.50 for each item sold. Plus $15.23 commission per 12 items sold over.

**Output should appear as:**

**Total commission for this week is $999.99**

NOTE: The '9's in the sample output represent the numeric value of the output.

# Comments

Comments are notes of explanation that document lines or sections of a program.

Comments are part of the program, but the compiler ignores them. They are intended for people who may be reading the source code.

It is crucial, however, that you develop the habit of thoroughly annotating your code with descriptive comments. It might take extra time now, but it will almost certainly save time in the future.

## Single Line Comments

As was illustrated in previous programs, you simply place two forward slashes (//) where you want the comment to begin. The compiler ignores everything from that point to the end of the line.

## Multi-Line Comments

Multi-line comments start with /* (a forward slash followed by an asterisk) and end with */ (an asterisk followed by a forward slash). Everything between these markers is ignored.

```
1 /*
2     PROGRAM: PAYROLL.CPP
3     Written by Herbert Dorfmann
4     This program calculates company payroll
5     Last modified: 8/20/2006
6 */
7 #include <iostream>
```

# Multi-Line Comments

Unlike a comment started with //, a multi-line comment can span several lines. This makes it more convenient to write large blocks of comments because you do not have to mark every line. On the other hand, the multi-line comment is inconvenient for writing single line comments because you must type both a beginning and ending comment symbol.

Many programmers use a combination of single line comments and multi-line comments.

When using multi-line comments:

- Be careful not to reverse the beginning symbol with the ending symbol.
- Be sure not to forget the ending symbol.

## Focus on Software Engineering: Programming Style

Programming style refers to the way a programmer uses identifiers, spaces, tabs, blank lines, and punctuation characters to visually arrange a program's source code. These are some, but not all, of the elements of programming style.

When the compiler reads a program it processes it as one long stream of characters. The compiler is not influenced by whether each statement is on a separate line, or whether spaces separate operators from operands. Humans, on the other hand, find it difficult to read programs that aren't written in a visually pleasing manner.

# Focus on Software Engineering: Programming Style

**Program 2-22**

```
1 #include <iostream>
2 using namespace std;int main(){double shares=220.0;double
3 avgPrice=14.67;cout
4 <<"There were "<<shares<<" shares sold at $"<<avgPrice<<
5 " per share.\n";return 0;}
```

**Program Output**

```
There were 220 shares sold at $14.67 per share.
```

Although the program is syntactically correct (it doesn't violate any rules of C++), it is difficult to read.

# Focus on Software Engineering: Programming Style

**Program 2-23**

```
1  // This program is visually arranged to make it readable.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7     double shares = 220.0;
8     double avgPrice = 14.67;
9
10    cout << "There were " << shares << " shares sold at $";
11    cout << avgPrice << " per share.\n";
12    return 0;
13 }
```

**SAME PROGRAM, BUT WRITTEN DIFFERENTLY**

**Program Output**

```
There were 220 shares sold at $14.67 per share.
```

Although you are free to develop your own style, you should adhere to common programming practices. By doing so, you will write programs that visually make sense to other programmers and that minimize the likelihood of errors.

# Focus on Software Engineering: Programming Style

Another aspect of programming style is how to handle statements that are too long to fit on one line. Because C++ is a free-flowing language, it is usually possible to spread a statement over several lines. For example, here is a cout statement that uses four lines:

```
cout << "The fahrenheit temperature is "
<< fahrenheit
<< " and the celsius temperature is "
<< celsius << endl;
```

This statement works just as if it were typed on one line. You have already seen variable definitions treated similarly:

```
int fahrenheit,
celsius,
kelvin;
```