

Characters, C-Strings, and More About the `string` Class

TOPICS

- | | |
|---|--|
| 10.1 Character Testing | 10.6 Focus on Software Engineering: Writing Your Own C-String-Handling Functions |
| 10.2 Character Case Conversion | |
| 10.3 C-Strings | |
| 10.4 Library Functions for Working with C-Strings | 10.7 More About the C++ <code>string</code> Class |
| 10.5 C-String/Numeric Conversion Functions | 10.8 Focus on Problem Solving and Program Design: A Case Study |

10.1 Character Testing

CONCEPT: The C++ library provides several functions for testing characters. To use these functions you must include the `cctype` header file.

The C++ library provides several functions that allow you to test the value of a character. These functions test a single `char` argument and return either `true` or `false`.^{*} For example, the following program segment uses the `isupper` function to determine whether the character passed as an argument is an uppercase letter. If it is, the function returns `true`. Otherwise, it returns `false`.

```
char letter = 'a';
if (isupper(letter))
    cout << "Letter is uppercase.\n";
else
    cout << "Letter is lowercase.\n";
```

Because the variable `letter`, in this example, contains a lowercase character, `isupper` returns `false`. The `if` statement will cause the message “Letter is lowercase” to be displayed.

^{*} These functions actually return an `int` value. The return value is nonzero to indicate `true`, or zero to indicate `false`.

Table 10-1 lists several character-testing functions. Each of these is prototyped in the `cctype` header file, so be sure to include that file when using the functions.

Table 10-1

Character Function	Description
<code>isalpha</code>	Returns true (a nonzero number) if the argument is a letter of the alphabet. Returns 0 if the argument is not a letter.
<code>isalnum</code>	Returns true (a nonzero number) if the argument is a letter of the alphabet or a digit. Otherwise it returns 0.
<code>isdigit</code>	Returns true (a nonzero number) if the argument is a digit from 0 through 9. Otherwise it returns 0.
<code>islower</code>	Returns true (a nonzero number) if the argument is a lowercase letter. Otherwise, it returns 0.
<code>isprint</code>	Returns true (a nonzero number) if the argument is a printable character (including a space). Returns 0 otherwise.
<code>ispunct</code>	Returns true (a nonzero number) if the argument is a printable character other than a digit, letter, or space. Returns 0 otherwise.
<code>isupper</code>	Returns true (a nonzero number) if the argument is an uppercase letter. Otherwise, it returns 0.
<code>isspace</code>	Returns true (a nonzero number) if the argument is a whitespace character. Whitespace characters are any of the following: space ' ' vertical tab '\v' newline '\n' tab '\t' Otherwise, it returns 0.

Program 10-1 uses several of the functions shown in Table 10-1. It asks the user to input a character and then displays various messages, depending upon the return value of each function.

Program 10-1

```
1 // This program demonstrates some character-testing functions.
2 #include <iostream>
3 #include <cctype>
4 using namespace std;
5
6 int main()
7 {
8     char input;
9
10    cout << "Enter any character: ";
11    cin.get(input);
12    cout << "The character you entered is: " << input << endl;
```

```

13     if (isalpha(input))
14         cout << "That's an alphabetic character.\n";
15     if (isdigit(input))
16         cout << "That's a numeric digit.\n";
17     if (islower(input))
18         cout << "The letter you entered is lowercase.\n";
19     if (isupper(input))
20         cout << "The letter you entered is uppercase.\n";
21     if (isspace(input))
22         cout << "That's a whitespace character.\n";
23     return 0;
24 }

```

Program Output with Example Input Shown in Bold

Enter any character: **A [Enter]**
 The character you entered is: A
 That's an alphabetic character.
 The letter you entered is uppercase.

Program Output with Different Example Input Shown in Bold

Enter any character: **7 [Enter]**
 The character you entered is: 7
 That's a numeric digit.

Program 10-2 shows a more practical application of the character testing functions. It tests a seven-character customer number to determine whether it is in the proper format.

Program 10-2

```

1  // This program tests a customer number to determine whether
2  // it is in the proper format.
3  #include <iostream>
4  #include <cctype>
5  using namespace std;
6
7  // Function prototype
8  bool testNum(char [], int);
9
10 int main()
11 {
12     const int SIZE = 8;    // Array size
13     char customer[SIZE];   // To hold a customer number
14
15     // Get the customer number.
16     cout << "Enter a customer number in the form ";
17     cout << "LLLNNNN\n";
18     cout << "(LLL = letters and NNNN = numbers): ";
19     cin.getline(customer, SIZE);

```

(program continues)

Program 10-2 (continued)

```

20
21     // Determine whether it is valid.
22     if (testNum(customer, SIZE))
23         cout << "That's a valid customer number.\n";
24     else
25     {
26         cout << "That is not the proper format of the ";
27         cout << "customer number.\nHere is an example:\n";
28         cout << "    ABC1234\n";
29     }
30     return 0;
31 }
32
33 //*****
34 // Definition of function testNum. *
35 // This function determines whether the custNum parameter *
36 // holds a valid customer number. The size parameter is *
37 // the size of the custNum array. *
38 //*****
39
40 bool testNum(char custNum[], int size)
41 {
42     int count; // Loop counter
43
44     // Test the first three characters for alphabetic letters.
45     for (count = 0; count < 3; count++)
46     {
47         if (!isalpha(custNum[count]))
48             return false;
49     }
50
51     // Test the remaining characters for numeric digits.
52     for (count = 3; count < size - 1; count++)
53     {
54         if (!isdigit(custNum[count]))
55             return false;
56     }
57     return true;
58 }

```

Program Output with Example Input Shown in Bold

Enter a customer number in the form LLLNNNN
 (LLL = letters and NNNN = numbers): **RQS4567** [Enter]
 That's a valid customer number.

Program Output with Different Example Input Shown in Bold

Enter a customer number in the form LLLNNNN
 (LLL = letters and NNNN = numbers): **AX467T9** [Enter]
 That is not the proper format of the customer number.
 Here is an example:
 ABC1234

In this program, the customer number is expected to consist of three alphabetic letters followed by four numeric digits. The `testNum` function accepts an array argument and tests the first three characters with the following loop in lines 45 through 49:

```
for (count = 0; count < 3; count++)
{
    if (!isalpha(custNum[count]))
        return false;
}
```

The `isalpha` function returns `true` if its argument is an alphabetic character. The `!` operator is used in the `if` statement to determine whether the tested character is NOT alphabetic. If this is so for any of the first three characters, the function `testNum` returns `false`. Likewise, the next four characters are tested to determine whether they are numeric digits with the following loop in lines 52 through 56:

```
for (count = 3; count < size - 1; count++)
{
    if (!isdigit(custNum[count]))
        return false;
}
```

The `isdigit` function returns `true` if its argument is the character representation of any of the digits 0 through 9. Once again, the `!` operator is used to determine whether the tested character is *not* a digit. If this is so for any of the last four characters, the function `testNum` returns `false`. If the customer number is in the proper format, the function will cycle through both the loops without returning `false`. In that case, the last line in the function is the `return true` statement, which indicates the customer number is valid.

10.2 Character Case Conversion

CONCEPT: The C++ library offers functions for converting a character to upper- or lowercase.

The C++ library provides two functions, `toupper` and `tolower`, for converting the case of a character. The functions are described in Table 10-2. (These functions are prototyped in the header file `cctype`, so be sure to include it.)

Table 10-2

Function	Description
<code>toupper</code>	Returns the uppercase equivalent of its argument.
<code>tolower</code>	Returns the lowercase equivalent of its argument.

Each of the functions in Table 10-2 accepts a single character argument. If the argument is a lowercase letter, the `toupper` function returns its uppercase equivalent. For example, the following statement will display the character **A** on the screen:

```
cout << toupper('a');
```

If the argument is already an uppercase letter, `toupper` returns it unchanged. The following statement causes the character `Z` to be displayed:

```
cout << toupper('Z');
```

Any nonletter argument passed to `toupper` is returned as it is. Each of the following statements display `toupper`'s argument without any change:

```
cout << toupper('*'); // Displays *
cout << toupper('&'); // Displays &
cout << toupper('%'); // Displays %
```

`toupper` and `tolower` don't actually cause the character argument to change, they simply return the upper- or lowercase equivalent of the argument. For example, in the following program segment, the variable `letter` is set to the value `'A'`. The `tolower` function returns the character `'a'`, but `letter` still contains `'A'`.

```
char letter = 'A';
cout << tolower(letter) << endl;
cout << letter << endl;
```

These statements will cause the following to be displayed:

```
a
A
```

Program 10-3 demonstrates the `toupper` function in an input validation loop.

Program 10-3

```
1 // This program calculates the area of a circle. It asks the user
2 // if he or she wishes to continue. A loop that demonstrates the
3 // toupper function repeats until the user enters 'y', 'Y',
4 // 'n', or 'N'.
5 #include <iostream>
6 #include <cctype>
7 #include <iomanip>
8 using namespace std;
9
10 int main()
11 {
12     const double PI = 3.14159; // Constant for pi
13     double radius;             // The circle's radius
14     char goAgain;               // To hold Y or N
15
16     cout << "This program calculates the area of a circle.\n";
17     cout << fixed << setprecision(2);
18
19     do
20     {
21         // Get the radius and display the area.
22         cout << "Enter the circle's radius: ";
23         cin >> radius;
24         cout << "The area is " << (PI * radius * radius);
25         cout << endl;
```

```

26
27         // Does the user want to do this again?
28         cout << "Calculate another? (Y or N) ";
29         cin >> goAgain;
30
31         // Validate the input.
32         while (toupper(goAgain) != 'Y' && toupper(goAgain) != 'N')
33         {
34             cout << "Please enter Y or N: ";
35             cin >> goAgain;
36         }
37
38     } while (toupper(goAgain) == 'Y');
39     return 0;
40 }

```

Program Output with Example Input Shown in Bold

This program calculates the area of a circle.

Enter the circle's radius: **10** [Enter]

The area is 314.16

Calculate another? (Y or N) **b** [Enter]

Please enter Y or N: **y** [Enter]

Enter the circle's radius: **1** [Enter]

The area is 3.14

Calculate another? (Y or N) **n** [Enter]

In lines 28 and 29 the user is prompted to enter either Y or N to indicate whether he or she wants to calculate another area. We don't want the program to be so picky that it accepts only uppercase Y or uppercase N. Lowercase y or lowercase n are also acceptable. The input validation loop must be written to reject anything except 'Y', 'y', 'N', or 'n'. One way to do this would be to test the goAgain variable in four relational expressions, as shown here:

```

while (goAgain != 'Y' && goAgain != 'y' &&
      goAgain != 'N' && goAgain != 'n')

```

Although there is nothing wrong with this code, we could use the toupper function to get the uppercase equivalent of goAgain and make only two comparisons. This is the approach taken in line 32:

```

while (toupper(goAgain) != 'Y' && toupper(goAgain) != 'N')

```

Another approach would have been to use the tolower function to get the lowercase equivalent of goAgain. Here is an example:

```

while (tolower(goAgain) != 'y' && tolower(goAgain) != 'n')

```

Either approach will yield the same results.



Checkpoint

- 10.1 Write a short description of each of the following functions:

```
isalpha
isalnum
isdigit
islower
isprint
ispunct
isupper
isspace
toupper
tolower
```

- 10.2 Write a statement that will convert the contents of the `char` variable `big` to lowercase. The converted value should be assigned to the variable `little`.
- 10.3 Write an `if` statement that will display the word “digit” if the variable `ch` contains a numeric digit. Otherwise, it should display “Not a digit.”
- 10.4 What is the output of the following statement?

```
cout << toupper(tolower('A'));
```

- 10.5 Write a loop that asks the user “Do you want to repeat the program or quit? (R/Q)”. The loop should repeat until the user has entered an R or Q (either uppercase or lowercase).

10.3 C-Strings

CONCEPT: In C++, a C-string is a sequence of characters stored in consecutive memory locations, terminated by a null character.

String is a generic term that describes any consecutive sequence of characters. A word, a sentence, a person’s name, and the title of a song are all strings. In the C++ language, there are two primary ways that strings are stored in memory: as `string` objects, or as C-strings. You have already been introduced to the `string` class, and by now, you have written several programs that use `string` objects. In this section, we will use C-strings, which are an alternative method for storing and working with strings.

A *C-string* is a string whose characters are stored in consecutive memory locations and are followed by a null character, or null terminator. Recall from Chapter 2 that a null character or null terminator is a byte holding the ASCII code 0. Strings that are stored this way are called C-strings because this is the way strings are handled in the C programming language.

In C++, all string literals are stored in memory as C-strings. Recall that a string literal (or string constant) is the literal representation of a string in a program. In C++, string literals are enclosed in double quotation marks, such as:

```
"Bailey"
```


Figure 10-1 illustrates how the string literal "Bailey" is stored in memory, as a C-string.

Figure 10-1

B	a	i	l	e	y	\0
---	---	---	---	---	---	----



NOTE: Remember that \0 ("slash zero") is the escape sequence representing the null terminator. It stands for the ASCII code 0.

The purpose of the null terminator is to mark the end of the C-string. Without it, there would be no way for a program to know the length of a C-string.

More About String Literals

A string literal or string constant is enclosed in a set of double quotation marks (" "). For example, here are five string literals:

```
"Have a nice day."
"What is your name?"
"John Smith"
"Please enter your age:"
"Part Number 45Q1789"
```

All of a program's string literals are stored in memory as C-strings, with the null terminator automatically appended. For example, look at Program 10-4.

Program 10-4

```
1 // This program contains string literals.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     char again;
8
9     do
10    {
11        cout << "C++ programming is great fun!" << endl;
12        cout << "Do you want to see the message again? ";
13        cin >> again;
14    } while (again == 'Y' || again == 'y');
15    return 0;
16 }
```

This program contains two string literals:

```
"C++ programming is great fun!"
"Do you want to see the message again? "
```

The first string occupies 30 bytes of memory (including the null terminator), and the second string occupies 39 bytes. They appear in memory in the following forms:

C	+	+		p	r	o	g	r	a	m	m	i	n	g		i	s		g	r	e	a	t		f	u	n	!	\0
D	o		y	o	u		w	a	n	t		t	o		s	e	e		t	h	e		m	e	s	s	a	g	
e		a	g	a	i	n	?		\0																				

It's important to realize that a string literal has its own storage location, just like a variable or an array. When a string literal appears in a statement, it's actually its memory address that C++ uses. Look at the following example:

```
cout << "Do you want to see the message again? ";
```

In this statement, the memory address of the string literal “Do you want to see the message again?” is passed to the `cout` object. `cout` displays the consecutive characters found at this address. It stops displaying the characters when a null terminator is encountered.

C-Strings Stored in Arrays

The C programming language does not provide a `string` class like the one that C++ provides. In the C language, all strings are treated as C-strings. When a C programmer wants to store a string in memory, he or she has to create a `char` array that is large enough to hold the string, plus one extra element for the null character.

You might be wondering why this should matter to anyone learning C++. You need to know about C-strings for the following reasons:

- The `string` class has not always existed in the C++ language. Several years ago, C++ stored strings as C-strings. As a professional programmer, you might encounter older C++ code (known as *legacy code*) that uses C-strings.
- Some of the C++ library functions work only with C-strings. For example, when you use a file stream object to open a file, the `open` member function accepts a C-string argument for the filename.
- In the workplace, it is not unusual for C++ programmers to work with specialized libraries that are written in C. Any strings that C libraries work with will be C-strings.

As previously mentioned, if you want to store a C-string in memory, you have to define a `char` array that is large enough to hold the string, plus one extra element for the null character. Here is an example:

```
const int SIZE = 21;
char name[SIZE];
```

This code defines a `char` array that has 21 elements, so it is big enough to hold a C-string that is no more than 20 characters long.

You can initialize a `char` array with a string literal, as shown here:

```
const int SIZE = 21;
char name[SIZE] = "Jasmine";
```

After this code executes, the `name` array will be created with 21 elements. The first 8 elements will be initialized with the characters 'J', 'a', 's', 'm', 'i', 'n', 'e', and '\0'. The null character is automatically added as the last character. You can implicitly size a `char` array by initializing it with a string literal, as shown here:

```
char name[] = "Jasmine";
```

After this code executes, the `name` array will be created with 8 elements, initialized with the characters 'J', 'a', 's', 'm', 'i', 'n', 'e', and '\0'.

C-string input can be performed by the `cin` object. For example, the following code allows the user to enter a string (with no whitespace characters) into the `name` array:

```
const int SIZE = 21;
char name[SIZE];
cin >> name;
```

Recall from Chapter 7 that an array name with no brackets and no subscript is converted into the beginning address of the array. In the previous statement, `name` indicates the address in memory where the string is to be stored. Of course, `cin` has no way of knowing that `name` has 21 elements. If the user enters a string of 30 characters, `cin` will write past the end of the array. This can be prevented by using `cin`'s `getline` member function. Assume the following array has been defined in a program:

```
const int SIZE = 80;
char line[SIZE];
```

The following statement uses `cin`'s `getline` member function to get a line of input (including whitespace characters) and store it in the `line` array:

```
cin.getline(line, SIZE);
```

The first argument tells `getline` where to store the string input. This statement indicates the starting address of the `line` array as the storage location for the string. The second argument indicates the maximum length of the string, including the null terminator. In this example, the `SIZE` constant is equal to 80, so `cin` will read 79 characters, or until the user presses the **[Enter]** key, whichever comes first. `cin` will automatically append the null terminator to the end of the string.

Once a string is stored in an array, it can be processed using standard subscript notation. For example, Program 10-5 displays a string stored in an array. It uses a loop to display each character in the array until the null terminator is encountered.

Program 10-5

```
1 // This program displays a string stored in a char array.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     const int SIZE = 80; // Array size
8     char line[SIZE];     // To hold a line of input
9     int count = 0;       // Loop counter variable
10
11     // Get a line of input.
12     cout << "Enter a sentence of no more than "
13          << (SIZE - 1) << " characters:\n";
14     cin.getline(line, SIZE);
15
```

(program continues)

Program 10-5 *(continued)*

```

16     // Display the input one character at a time.
17     cout << "The sentence you entered is:\n";
18     while (line[count] != '\0')
19     {
20         cout << line[count];
21         count++;
22     }
23     return 0;
24 }

```

Program Output with Example Input Shown in Bold

Enter a sentence of no more than 79 characters:

C++ is challenging but fun! [Enter]

The sentence you entered is:

C++ is challenging but fun!

10.4 Library Functions for Working with C-Strings

CONCEPT: The C++ library has numerous functions for handling C-strings. These functions perform various tests and manipulations and require that the `cstring` header file be included.

The `strlen` Function

Because C-strings are stored in arrays, working with them is quite different than working with `string` objects. Fortunately, the C++ library provides many functions for manipulating and testing C-strings. These functions all require the `cstring` header file to be included, as shown here:

```
#include <cstring>
```

For instance, the following code segment uses the `strlen` function to determine the length of the string stored in the `name` array:

```

char name[] = "Thomas Edison";
int length;
length = strlen(name);

```

The `strlen` function accepts a pointer to a C-string as its argument. It returns the length of the string, which is the number of characters up to, but not including, the null terminator. As a result, the variable `length` will have the number 13 stored in it. The length of a string isn't to be confused with the size of the array holding it. Remember, the only information being passed to `strlen` is the beginning address of a C-string. It doesn't know where the array ends, so it looks for the null terminator to indicate the end of the string.

When using a C-string handling function, you must pass one or more C-strings as arguments. This means passing the address of the C-string, which may be accomplished by using any of the following as arguments:

- The name of the array holding the C-string
- A pointer variable that holds the address of the C-string
- A literal string

Anytime a literal string is used as an argument to a function, the address of the literal string is passed. Here is an example of the `strlen` function being used with such an argument:

```
length = strlen("Thomas Edison");
```

The `strcat` Function

The `strcat` function accepts two pointers to C-strings as its arguments. The function *concatenates*, or appends one string to another. The following code shows an example of its use:

```
const int SIZE = 13;
char string1[SIZE] = "Hello ";
char string2[] = "World!";

cout << string1 << endl;
cout << string2 << endl;
strcat(string1, string2);
cout << string1 << endl;
```

These statements will cause the following output:

```
Hello
World!
Hello World!
```

The `strcat` function copies the contents of `string2` to the end of `string1`. In this example, `string1` contains the string “Hello ” before the call to `strcat`. After the call, it contains the string “Hello World!”. Figure 10-2 shows the contents of both arrays before and after the function call.

Figure 10-2

Before the call to `strcat (string1, string2):`

`string1`

H	e	l	l	o		\0						
---	---	---	---	---	--	----	--	--	--	--	--	--

`string2`

W	o	r	l	d	!	\0
---	---	---	---	---	---	----

After the call to `strcat (string1, string2):`

`string1`

H	e	l	l	o		W	o	r	l	d	!	\0
---	---	---	---	---	--	---	---	---	---	---	---	----

`string2`

W	o	r	l	d	!	\0
---	---	---	---	---	---	----

Notice the last character in `string1` (before the null terminator) is a space. The `strcat` function doesn't insert a space, so it's the programmer's responsibility to make sure one is already there, if needed. It's also the programmer's responsibility to make sure the array holding `string1` is large enough to hold `string1` plus `string2` plus a null terminator.

Here is a program segment that uses the `sizeof` operator to test an array's size before `strcat` is called:

```
if (sizeof(string1) >= (strlen(string1) + strlen(string2) + 1))
    strcat(string1, string2);
else
    cout << "String1 is not large enough for both strings.\n";
```



WARNING! If the array holding the first string isn't large enough to hold both strings, `strcat` will overflow the boundaries of the array.

The `strcpy` Function

Recall from Chapter 7 that one array cannot be assigned to another with the `=` operator. Each individual element must be assigned, usually inside a loop. The `strcpy` function can be used to copy one string to another. Here is an example of its use:

```
const int SIZE = 13;
char name[SIZE];
strcpy(name, "Albert Einstein");
```

The `strcpy` function's two arguments are C-string addresses. The contents of the second argument are copied to the memory location specified by the first argument, including the null terminator. (The first argument usually references an array.) In this example, the `strcpy` function will copy the string "Albert Einstein" to the `name` array.

If anything is already stored in the location referenced by the first argument, it is overwritten, as shown in the following program segment:

```
const int SIZE = 10;
char string1[SIZE] = "Hello", string2[SIZE] = "World!";
cout << string1 << endl;
cout << string2 << endl;
strcpy(string1, string2);
cout << string1 << endl;
cout << string2 << endl;
```

Here is the output:

```
Hello
World!
World!
World!
```



WARNING! Being true to C++'s nature, `strcpy` performs no bounds checking. The array specified by the first argument will be overflowed if it isn't large enough to hold the string specified by the second argument.

The `strncat` and `strncpy` Functions

Because the `strcat` and `strcpy` functions can potentially overwrite the bounds of an array, they make it possible to write unsafe code. As an alternative, you should use `strncat` and `strncpy` whenever possible.

The `strncat` function works like `strcat`, except it takes a third argument specifying the maximum number of characters from the second string to append to the first. Here is an example call to `strncat`:

```
strncat(string1, string2, 10);
```

When this statement executes, `strncat` will append no more than 10 characters from `string2` to `string1`. The following code shows an example of calculating the maximum number of characters that can be appended to an array.

```
1  int maxChars;
2  const int SIZE_1 = 17;
3  const int SIZE_2 = 18;
4
5  char string1[SIZE_1] = "Welcome ";
6  char string2[SIZE_2] = "to North Carolina";
7
8  cout << string1 << endl;
9  cout << string2 << endl;
10 maxChars = sizeof(string1) - (strlen(string1) + 1);
11 strncat(string1, string2, maxChars);
12 cout << string1 << endl;
```

The statement in line 10 calculates the number of empty elements in `string1`. It does this by subtracting the length of the string stored in the array plus 1 for the null terminator. This code will cause the following output:

```
Welcome
to North Carolina
Welcome to North
```

The `strncpy` function allows you to copy a specified number of characters from a string to a destination. Calling `strncpy` is similar to calling `strcpy`, except you pass a third argument specifying the maximum number of characters from the second string to copy to the first. Here is an example call to `strncpy`:

```
strncpy(string1, string2, 5);
```

When this statement executes, `strncpy` will copy no more than five characters from `string2` to `string1`. However, if the specified number of characters is less than or equal to the length of `string2`, a null terminator is not appended to `string1`. If the specified number of characters is greater than the length of `string2`, then `string1` is padded with null terminators, up to the specified number of characters. The following code shows an example using the `strncpy` function.

```
1  int maxChars;
2  const int SIZE = 11;
3
4  char string1[SIZE];
5  char string2[] = "I love C++ programming!";
6
```

```

7  maxChars = sizeof(string1) - 1;
8  strncpy(string1, string2, maxChars);
9  // Put the null terminator at the end.
10 string1[maxChars] = '\0';
11 cout << string1 << endl;

```

Notice that a statement was written in line 10 to put the null terminator at the end of `string1`. This is because `maxChars` was less than the length of `string2`, and `strncpy` did not automatically place a null terminator there.

The `strstr` Function

The `strstr` function searches for a string inside of a string. For instance, it could be used to search for the string “seven” inside the larger string “Four score and seven years ago.” The function’s first argument is the string to be searched, and the second argument is the string to look for. If the function finds the second string inside the first, it returns the address of the occurrence of the second string within the first string. Otherwise it returns `nullptr` (the address 0). Here is an example:

```

char arr[] = "Four score and seven years ago";
char *strPtr = nullptr;
cout << arr << endl;
strPtr = strstr(arr, "seven"); // search for "seven"
cout << strPtr << endl;

```

In this code, `strstr` will locate the string “seven” inside the string “Four score and seven years ago.” It will return the address of the first character in “seven” which will be stored in the pointer variable `strPtr`. If run as part of a complete program, this segment will display the following:

```

Four score and seven years ago
seven years ago

```



NOTE: The `nullptr` key word was introduced in C++ 11. If you are using a previous version of the C++ language, use the constant `NULL` instead.

The `strstr` function can be useful in any program that must locate data inside one or more strings. Program 10-6, for example, stores a list of product numbers and descriptions in an array of C-strings. It allows the user to look up a product description by entering all or part of its product number.

Program 10-6

```

1  // This program uses the strstr function to search an array.
2  #include <iostream>
3  #include <cstring>    // For strstr
4  using namespace std;
5
6  int main()
7  {
8      // Constants for array lengths
9      const int NUM_PRODS = 5;    // Number of products
10     const int LENGTH = 27;      // String length
11

```



```

12      // Array of products
13      char products[NUM_PRODS][LENGTH] =
14          { "TV327 31-inch Television",
15            "CD257 CD Player",
16            "TA677 Answering Machine",
17            "CS109 Car Stereo",
18            "PC955 Personal Computer" };
19
20      char lookUp[LENGTH];      // To hold user's input
21      char *strPtr = nullptr;    // To point to the found product
22      int index;                // Loop counter
23
24      // Prompt the user for a product number.
25      cout << "\tProduct Database\n\n";
26      cout << "Enter a product number to search for: ";
27      cin.getline(lookUp, LENGTH);
28
29      // Search the array for a matching substring
30      for (index = 0; index < NUM_PRODS; index++)
31      {
32          strPtr = strstr(products[index], lookUp);
33          if (strPtr != nullptr)
34              break;
35      }
36
37      // If a matching substring was found, display the product info.
38      if (strPtr != nullptr)
39          cout << products[index] << endl;
40      else
41          cout << "No matching product was found.\n";
42
43      return 0;
44  }

```

Program Output with Example Input Shown in Bold

```

Product Database
Enter a product to search for: CS [Enter]
CS109 Car Stereo

```

Program Output with Different Example Input Shown in Bold

```

Product Database
Enter a product to search for: AB [Enter]
No matching product was found.

```

Table 10-3 summarizes the string-handling functions discussed here, as well as the `strcmp` function that was discussed in Chapter 4. (All the functions listed require the `cstring` header file.)

In Program 10-6, the `for` loop in lines 30 through 35 cycles through each C-string in the array calling the following statement:

```
strPtr = strstr(prods[index], lookUp);
```

The `strstr` function searches the string referenced by `prods[index]` for the name entered by the user, which is stored in `lookUp`. If `lookUp` is found inside `prods[index]`, the function returns its address. In that case, the following `if` statement causes the `for` loop to terminate:

```
if (strPtr != nullptr)
    break;
```

Outside the loop, the following `if else` statement in lines 38 through 41 determines whether the string entered by the user was found in the array. If not, it informs the user that no matching product was found. Otherwise, the product number and description are displayed:

```
if (strPtr == nullptr)
    cout << "No matching product was found.\n";
else
    cout << prods[index] << endl;
```

The `strcmp` Function

Because C-strings are stored in `char` arrays, you cannot use the relational operators to compare two C-strings. To compare C-strings, you should use the library function `strcmp`. This function takes two C-strings as arguments and returns an integer that indicates how the two strings compare to each other. Here is the function's prototype:

```
int strcmp(char *string1, char *string2);
```

The function takes two C-strings as parameters (actually, pointers to C-strings) and returns an integer result. The value of the result is set accordingly:

- The result is *zero* if the two strings are *equal* on a character-by-character basis
- The result is *negative* if `string1` comes *before* `string2` in alphabetical order
- The result is *positive* if `string1` comes *after* `string2` in alphabetical order

Here is an example of the use of `strcmp` to determine if two strings are equal:

```
if (strcmp(string1, string2) == 0)
    cout << "The strings are equal.\n";
else
    cout << "The strings are not equal.\n";
```

Program 10-7 shows a complete example.

Program 10-7

```
1 // This program tests two C-strings for equality
2 // using the strcmp function.
3 #include <iostream>
4 #include <string>
5 using namespace std;
6
7 int main()
8 {
9     // Two arrays for two strings.
10    const int LENGTH = 40;
11    char firstString[LENGTH], secondString[LENGTH];
12
```

```

13     // Read two strings.
14     cout << "Enter a string: ";
15     cin.getline(firstString, LENGTH);
16     cout << "Enter another string: ";
17     cin.getline(secondString, LENGTH);
18
19     // Compare the strings for equality with strcmp.
20     if (strcmp(firstString, secondString) == 0)
21         cout << "You entered the same string twice.\n";
22     else
23         cout << "The strings are not the same.\n";
24
25     return 0;
26 }

```

Program Output with Example Input Shown in Bold

```

Enter a string: Alfonso [Enter]
Enter another string: Alfonso [Enter]
You entered the same string twice.

```

The `strcmp` function is case sensitive when it compares strings. If the user enters “Dog” and “dog” in Program 10-7, it will report they are not the same. Most compilers provide nonstandard versions of `strcmp` that perform case-insensitive comparisons. For instance, some compilers provide a function named `stricmp` that works identically to `strcmp` except the case of the characters is ignored.

Program 10-8 is a more practical example of how `strcmp` can be used. It asks the user to enter the part number of the stereo they wish to purchase. The part number contains digits, letters, and a hyphen, so it must be stored as a string. Once the user enters the part number, the program displays the price of the stereo.

Program 10-8

```

1  // This program uses strcmp to compare the string entered
2  // by the user with the valid stereo part numbers.
3  #include <iostream>
4  #include <cstring>
5  #include <iomanip>
6  using namespace std;
7
8  int main()
9  {
10     // Price of parts.
11     const double A_PRICE = 249.0,
12                B_PRICE = 299.0;
13
14     // Character array for part number.
15     const int PART_LENGTH = 8;
16     char partNum[PART_LENGTH];
17

```

(program continues)

Program 10-8 *(continued)*

```

18      // Instruct the user to enter a part number.
19      cout << "The stereo part numbers are:\n"
20           << "\tBoom Box, part number S147-29A\n"
21           << "\tShelf Model, part number S147-29B\n"
22           << "Enter the part number of the stereo you\n"
23           << "wish to purchase: ";
24
25      // Read a part number of at most 8 characters.
26      cin >> partNum;
27
28      // Determine what user entered using strcmp
29      // and print its price.
30      cout << showpoint << fixed << setprecision(2);
31      if (strcmp(partNum, "S147-29A") == 0)
32          cout << "The price is $" << A_PRICE << endl;
33      else if (strcmp(partNum, "S147-29B") == 0)
34          cout << "The price is $" << B_PRICE << endl;
35      else
36          cout << partNum << " is not a valid part number.\n";
37      return 0;
38  }

```

Program Output with Example Input Shown in Bold

```

The stereo part numbers are:
    Boom Box, part number S147-29A
    Shelf Model, part number S147-29B
Enter the part number of the stereo you
wish to purchase: S147-29B [Enter]
The price is $299.00

```

Using ! with strcmp

Some programmers prefer to use the logical NOT operator with `strcmp` when testing strings for equality. Because 0 is considered logically false, the `!` operator converts that value to true. The expression `!strcmp(string1, string2)` returns true when both strings are the same, and false when they are different. The two following statements perform the same operation:

```

if (strcmp(firstString, secondString) == 0)
if (!strcmp(firstString, secondString))

```

Sorting Strings

Programs are frequently written to print alphabetically sorted lists of items. For example, consider a department store computer system that keeps customers' names and addresses in a file. The names do not appear in the file alphabetically but in the order the operator entered them. If a list were to be printed in this order, it would be very difficult to locate any specific name. The list would have to be sorted before it was printed.

Because the value returned by `strcmp` is based on the relative alphabetic order of the two strings being compared, it can be used in programs that sort strings. Program 10-9 asks the user to enter two names, which are then printed in alphabetic order.

Program 10-9

```

1 // This program uses the return value of strcmp to
2 // alphabetically sort two strings entered by the user.
3 #include <iostream>
4 #include <string>
5 using namespace std;
6
7 int main()
8 {
9     // Two arrays to hold two strings.
10    const int NAME_LENGTH = 30;
11    char name1[NAME_LENGTH], name2[NAME_LENGTH];
12
13    // Read two strings.
14    cout << "Enter a name (last name first): ";
15    cin.getline(name1, NAME_LENGTH);
16    cout << "Enter another name: ";
17    cin.getline(name2, NAME_LENGTH);
18
19    // Print the two strings in alphabetical order.
20    cout << "Here are the names sorted alphabetically:\n";
21    if (strcmp(name1, name2) < 0)
22        cout << name1 << endl << name2 << endl;
23    else if (strcmp(name1, name2) > 0)
24        cout << name2 << endl << name1 << endl;
25    else
26        cout << "You entered the same name twice!\n";
27
28    return 0;
29 }

```

Program Output with Example Input Shown in Bold

```

Enter a name (last name first): Smith, Richard [Enter]
Enter another name: Jones, John [Enter]
Here are the names sorted alphabetically:
Jones, John
Smith, Richard

```

Table 10-3 provides a summary of the C-string handling functions that we have discussed. All of the functions listed require the `cstring` header file.

Table 10-3

Function	Description
<code>strlen</code>	Accepts a C-string or a pointer to a C-string as an argument. Returns the length of the C-string (not including the null terminator.) <i>Example Usage:</i> <code>len = strlen(name);</code>
<code>strcat</code>	Accepts two C-strings or pointers to two C-strings as arguments. The function appends the contents of the second string to the first C-string. (The first string is altered, the second string is left unchanged.) <i>Example Usage:</i> <code>strcat(string1, string2);</code>

(table continues)

Table 10-3 (continued)

Function	Description
<code>strcpy</code>	Accepts two C-strings or pointers to two C-strings as arguments. The function copies the second C-string to the first C-string. The second C-string is left unchanged. <i>Example Usage:</i> <code>strcpy(string1, string2);</code>
<code>strncat</code>	Accepts two C-strings or pointers to two C-strings, and an integer argument. The third argument, an integer, indicates the maximum number of characters to copy from the second C-string to the first C-string. <i>Example Usage:</i> <code>strncat(string1, string2, n);</code>
<code>strncpy</code>	Accepts two C-strings or pointers to two C-strings, and an integer argument. The third argument, an integer, indicates the maximum number of characters to copy from the second C-string to the first C-string. If <code>n</code> is less than the length of <code>string2</code> , the null terminator is not automatically appended to <code>string1</code> . If <code>n</code> is greater than the length of <code>string2</code> , <code>string1</code> is padded with <code>'\0'</code> characters. <i>Example Usage:</i> <code>strncpy(string1, string2, n);</code>
<code>strcmp</code>	Accepts two C-strings or pointers to two C-strings arguments. If <code>string1</code> and <code>string2</code> are the same, this function returns 0. If <code>string2</code> is alphabetically greater than <code>string1</code> , it returns a negative number. If <code>string2</code> is alphabetically less than <code>string1</code> , it returns a positive number. <i>Example Usage:</i> <code>if (strcmp(string1, string2))</code>
<code>strstr</code>	Accepts two C-strings or pointers to two C-strings as arguments. Searches for the first occurrence of <code>string2</code> in <code>string1</code> . If an occurrence of <code>string2</code> is found, the function returns a pointer to it. Otherwise, it returns <code>nullptr</code> (address 0). <i>Example Usage:</i> <code>cout << strstr(string1, string2);</code>



Checkpoint

10.6 Write a short description of each of the following functions:

```
strlen
strcat
strcpy
strncat
strncpy
strcmp
strstr
```

10.7 What will the following program segment display?

```
char dog[] = "Fido";
cout << strlen(dog) << endl;
```

10.8 What will the following program segment display?

```
char string1[16] = "Have a ";
char string2[9] = "nice day";
strcat(string1, string2);
cout << string1 << endl;
cout << string2 << endl;
```

- 10.9 Write a statement that will copy the string “Beethoven” to the array `composer`.
- 10.10 When complete, the following program skeleton will search for the string “windy” in the array `place`. If `place` contains “windy” the program will display the message “windy found.” Otherwise it will display “windy not found.”

```
#include <iostream>
// include any other necessary header files
using namespace std;

int main()
{
    char place[] = "The Windy City";
    // Complete the program. It should search the array place
    // for the string "Windy" and display the message "Windy
    // found" if it finds the string. Otherwise, it should
    // display the message "Windy not found."
    return 0;
}
```

10.5 C-String/Numeric Conversion Functions

CONCEPT: The C++ library provides functions for converting a C-string representation of a number to a numeric data type and vice versa.

There is a great difference between a number that is stored as a string and one stored as a numeric value. The string “26792” isn’t actually a number, but a series of ASCII codes representing the individual digits of the number. It uses six bytes of memory (including the null terminator). Because it isn’t an actual number, it’s not possible to perform mathematical operations with it, unless it is first converted to a numeric value.

Several functions exist in the C++ library for converting C-string representations of numbers into numeric values. Table 10-4 shows some of these. Note that all of these functions require the `cstdlib` header file.

Table 10-4

Function	Description
<code>atoi</code>	Accepts a C-string as an argument. The function converts the C-string to an integer and returns that value. <i>Example Usage:</i> <code>int num = atoi("4569");</code>
<code>atol</code>	Accepts a C-string as an argument. The function converts the C-string to a long integer and returns that value. <i>Example Usage:</i> <code>long lnum = atol("500000");</code>
<code>atof</code>	Accepts a C-string as an argument. The function converts the C-string to a double and returns that value. <i>Example Usage:</i> <code>double fnum = atof("3.14159");</code>

The `atoi` Function

The `atoi` function converts a string to an integer. It accepts a C-string argument and returns the converted integer value. Here is an example of how to use it:

```
int num;
num = atoi("1000");
```

In these statements, `atoi` converts the string “1000” into the integer 1000. Once the variable `num` is assigned this value, it can be used in mathematical operations or any task requiring a numeric value.

The `atol` Function

The `atol` function works just like `atoi`, except the return value is a long integer. Here is an example:

```
long bigNum;
bigNum = atol("500000");
```

The `atof` Function

The `atof` function accepts a C-string argument and converts it to a double. The numeric double value is returned, as shown here:

```
double num;
num = atof("12.67");
```

Although the `atof` function returns a double, you can still use it to convert a C-string to a float. For example, look at the following code.

```
float x;
x = atof("3.4");
```

The `atof` function converts the string “3.4” to the double value 3.4. Because 3.4 is within the range of a `float`, it can be stored in a `float` variable without the loss of data.



NOTE: If a string that cannot be converted to a numeric value is passed to any of these functions, the function’s behavior is undefined by C++. Many compilers, however, will perform the conversion process until an invalid character is encountered. For example, `atoi("123x5")` might return the integer 123. It is possible that these functions will return 0 if they cannot successfully convert their argument.

The `to_string` Function

11

C++ 11 introduces a function named `to_string` that converts a numeric value to a `string` object. There are nine overloaded versions of the `to_string` function listed in Table 10-5. Note that the `to_string` function requires the `string` header file to be included.

Each version of the function accepts an argument of a numeric data type and returns the value of that argument converted to a `string` object. Here is an example:

```
int number = 99;
string output = to_string(number);
```


Table 10-5

Function	Description
<code>to_string(int value);</code>	Accepts an <code>int</code> argument and returns that argument converted to a <code>string</code> object.
<code>to_string(long value);</code>	Accepts a <code>long</code> argument and returns that argument converted to a <code>string</code> object.
<code>to_string(long long value);</code>	Accepts a <code>long long</code> argument and returns that argument converted to a <code>string</code> object.
<code>to_string(unsigned value);</code>	Accepts an <code>unsigned</code> argument and returns that argument converted to a <code>string</code> object.
<code>to_string(unsigned long value);</code>	Accepts an <code>unsigned long</code> argument and returns that argument converted to a <code>string</code> object.
<code>to_string(unsigned long long value);</code>	Accepts an <code>unsigned long long</code> argument and returns that argument converted to a <code>string</code> object.
<code>to_string(float value);</code>	Accepts a <code>float</code> argument and returns that argument converted to a <code>string</code> object.
<code>to_string(double value);</code>	Accepts a <code>double</code> argument and returns that argument converted to a <code>string</code> object.
<code>to_string(long double value);</code>	Accepts a <code>long double</code> argument and returns that argument converted to a <code>string</code> object.

The first statement initializes the `number` variable (an `int`) to the value 99. In the second statement, the `number` variable is passed as an argument to the `to_string` function. The `to_string` function returns the value “99”, which is assigned to the output variable.

Here is another example:

```
double number = 3.14159;
cout << to_string(number) << endl;
```

The first statement initializes the `number` variable (a `double`) to the value 3.14159. In the second statement, the `number` variable is passed as an argument to the `to_string` function, and the value “3.14159” is returned from the function and displayed on the screen.

Now let’s look at Program 10-10, which uses a string-to-number conversion function, `atoi`. It allows the user to enter a series of values, or the letters `Q` or `q` to quit. The average of the numbers is then calculated and displayed.

Program 10-10

```
1 // This program demonstrates the strcmp and atoi functions.
2 #include <iostream>
3 #include <cctype>           // For tolower
4 #include <cstring>          // For strcmp
5 #include <cstdlib>          // For atoi
6 using namespace std;
7
```

(program continues)

Program 10-10 (continued)

```

8  int main()
9  {
10     const int SIZE = 20;    // Array size
11     char input[SIZE];      // To hold user input
12     int total = 0;          // Accumulator
13     int count = 0;          // Loop counter
14     double average;        // To hold the average of numbers
15
16     // Get the first number.
17     cout << "This program will average a series of numbers.\n";
18     cout << "Enter the first number or Q to quit: ";
19     cin.getline(input, SIZE);
20
21     // Process the number and subsequent numbers.
22     while (tolower(input[0]) != 'q')
23     {
24         total += atoi(input); // Keep a running total
25         count++;             // Count the numbers entered
26         // Get the next number.
27         cout << "Enter the next number or Q to quit: ";
28         cin.getline(input, SIZE);
29     }
30
31     // If any numbers were entered, display their average.
32     if (count != 0)
33     {
34         average = static_cast<double>(total) / count;
35         cout << "Average: " << average << endl;
36     }
37     return 0;
38 }

```

Program Output with Example Input Shown in Bold

```

This program will average a series of numbers.
Enter the first number or Q to quit: 74 [Enter]
Enter the next number or Q to quit: 98 [Enter]
Enter the next number or Q to quit: 23 [Enter]
Enter the next number or Q to quit: 54 [Enter]
Enter the next number or Q to quit: Q [Enter]
Average: 62.25

```

In line 22, the following while statement uses the `tolower` function to determine whether the first character entered by the user is “q” or “Q”.

```
while (tolower(input[0]) != 'q')
```

If the user hasn’t entered ‘Q’ or ‘q’ the loop performs an iteration. The following statement, in line 24, uses `atoi` to convert the string in `input` to an integer and adds its value to `total`:

```
total += atoi(input); // Keep a running total
```

The counter is updated in line 25 and then the user is asked for the next number. When all the numbers are entered, the user terminates the loop by entering 'Q' or 'q'. If one or more numbers are entered, their average is displayed.

The string-to-numeric conversion functions can also help with a common input problem. Recall from Chapter 3 that using `cin >>` and then calling `cin.get` causes problems because the `>>` operator leaves the newline character in the keyboard buffer. When the `cin.get` function executes, the first character it sees in the keyboard buffer is the newline character, so it reads no further.

The same problem exists when a program uses `cin >>` and then calls `cin.getline` to read a line of input. For example, look at the following code. (Assume `idNumber` is an `int` and `name` is a `char` array.)

```
1 // Get the user's ID number.
2 cout << "What is your ID number? ";
3 cin >> idNumber;
4
5 // Get the user's name.
6 cout << "What is your name? ";
7 cin.getline(name, NAME_SIZE);
```

Let's say the user enters 25 and presses Enter when the `cin >>` statement in line 3 executes. The value 25 will be stored in `idNumber`, and the newline character will be left in the keyboard buffer. When the `cin.getline` function is called in line 7, the first character it sees in the keyboard buffer is the newline character, so it reads no further. It will appear that the statement in line 7 was skipped.

One work-around that we have used in this book is to call `cin.ignore` to skip over the newline character just before calling `cin.getline`. Another approach is to use `cin.getline` to read all of a program's input, including numbers. When numeric input is needed, it is read into a `char` array as a string and then converted to the appropriate numeric data type. Because you aren't mixing `cin >>` with `cin.getline`, the problem of the remaining newline character doesn't exist. Program 10-11 shows an example.

Program 10-11

```
1 // This program demonstrates how the getline function can
2 // be used for all of a program's input.
3 #include <iostream>
4 #include <cstdlib>
5 #include <iomanip>
6 using namespace std;
7
8 int main()
9 {
10     const int INPUT_SIZE = 81; // Size of input array
11     const int NAME_SIZE = 30; // Size of name array
12     char input[INPUT_SIZE]; // To hold a line of input
13     char name[NAME_SIZE]; // To hold a name
14     int idNumber; // To hold an ID number
15     int age; // To hold an age
16     double income; // To hold income
```

(program continues)

Program 10-11 (continued)

```

17
18     // Get the user's ID number.
19     cout << "What is your ID number? ";
20     cin.getline(input, INPUT_SIZE); // Read as a string
21     idNumber = atoi(input);         // Convert to int
22
23     // Get the user's name. No conversion necessary.
24     cout << "What is your name? ";
25     cin.getline(name, NAME_SIZE);
26
27     // Get the user's age.
28     cout << "How old are you? ";
29     cin.getline(input, INPUT_SIZE); // Read as a string
30     age = atoi(input);              // Convert to int
31
32     // Get the user's income.
33     cout << "What is your annual income? ";
34     cin.getline(input, INPUT_SIZE); // Read as a string
35     income = atof(input);           // Convert to double
36
37     // Show the resulting data.
38     cout << setprecision(2) << fixed << showpoint;
39     cout << "Your name is " << name
40           << ", you are " << age
41           << " years old,\nand you make $"
42           << income << " per year.\n";
43
44     return 0;
45 }

```

Program Output with Example Input Shown in Bold

```

What is your ID number? 1234 [Enter]
What is your name? Janice Smith [Enter]
How old are you? 25 [Enter]
What is your annual income? 60000 [Enter]
Your name is Janice Smith, you are 25 years old,
and you make $60000.00 per year.

```

**Checkpoint**

10.11 Write a short description of each of the following functions:

```

atoi
atol
atof
itoa

```

10.12 Write a statement that will convert the string “10” to an integer and store the result in the variable num.

- 10.13 Write a statement that will convert the string "100000" to a long and store the result in the variable num.
- 10.14 Write a statement that will convert the string "7.2389" to a double and store the result in the variable num.
- 10.15 Write a statement that will convert the integer 127 to a string, stored in base-10 notation in the array value.

10.6 Focus on Software Engineering: Writing Your Own C-String-Handling Functions

CONCEPT: You can design your own specialized functions for manipulating strings.

By being able to pass arrays as arguments, you can write your own functions for processing C-strings. For example, Program 10-12 uses a function to copy a C-string from one array to another.



VideoNote
Writing a
C-String-
Handling
Function

Program 10-12

```

1  // This program uses a function to copy a C-string into an array.
2  #include <iostream>
3  using namespace std;
4
5  void stringCopy(char [], char []); // Function prototype
6
7  int main()
8  {
9      const int LENGTH = 30; // Size of the arrays
10     char first[LENGTH];    // To hold the user's input
11     char second[LENGTH];   // To hold the copy
12
13     // Get a string from the user and store in first.
14     cout << "Enter a string with no more than "
15          << (LENGTH - 1) << " characters:\n";
16     cin.getline(first, LENGTH);
17
18     // Copy the contents of first to second.
19     stringCopy(first, second);
20
21     // Display the copy.
22     cout << "The string you entered is:\n" << second << endl;
23     return 0;
24 }
25
26 //*****
27 // Definition of the stringCopy function. *
28 // This function copies the C-string in string1 to string2. *
29 //*****

```

(program continues)

Program 10-12 *(continued)*

```

30
31 void stringCopy(char string1[], char string2[])
32 {
33     int index = 0; // Loop counter
34
35     // Step through string1, copying each element to
36     // string2. Stop when the null character is encountered.
37     while (string1[index] != '\0')
38     {
39         string2[index] = string1[index];
40         index++;
41     }
42
43     // Place a null character in string2.
44     string2[index] = '\0';
45 }

```

Program Output with Example Input Shown in Bold

Enter a string with no more than 29 characters:

Thank goodness it's Friday! [Enter]

The string you entered is:

Thank goodness it's Friday!

Notice the function `stringCopy` does not accept an argument indicating the size of the arrays. It simply copies the characters from `string1` into `string2` until it encounters a null terminator in `string1`. When the null terminator is found, the loop has reached the end of the C-string. The last statement in the function assigns a null terminator (the `'\0'` character) to the end of `string2`, so it is properly terminated.



WARNING! Because the `stringCopy` function doesn't know the size of the second array, it's the programmer's responsibility to make sure the second array is large enough to hold the string in the first array.

Program 10-13 uses another C-string-handling function: `nameSlice`. The program asks the user to enter his or her first and last names, separated by a space. The function searches the string for the space and replaces it with a null terminator. In effect, this “cuts” the last name off of the string.

Program 10-13

```

1 // This program uses the function nameSlice to cut the last
2 // name off of a string that contains the user's first and
3 // last names.
4 #include <iostream>
5 using namespace std;

```

```

6
7 void nameSlice(char []); // Function prototype
8
9 int main()
10 {
11     const int SIZE = 41; // Array size
12     char name[SIZE];      // To hold the user's name
13
14     cout << "Enter your first and last names, separated ";
15     cout << "by a space:\n";
16     cin.getline(name, SIZE);
17     nameSlice(name);
18     cout << "Your first name is: " << name << endl;
19     return 0;
20 }
21
22 //*****
23 // Definition of function nameSlice. This function accepts a      *
24 // character array as its argument. It scans the array looking *
25 // for a space. When it finds one, it replaces it with a null *
26 // terminator.                                                    *
27 //*****
28
29 void nameSlice(char userName[])
30 {
31     int count = 0; // Loop counter
32
33     // Locate the first space, or the null terminator if there
34     // are no spaces.
35     while (userName[count] != ' ' && userName[count] != '\0')
36         count++;
37
38     // If a space was found, replace it with a null terminator.
39     if (userName[count] == ' ')
40         userName[count] = '\0';
41 }

```

Program Output with Example Input Shown in Bold

Enter your first and last names, separated by a space:

Jimmy Jones [Enter]

Your first name is: Jimmy

The following loop in lines 35 and 36 starts at the first character in the array and scans the string searching for either a space or a null terminator:

```

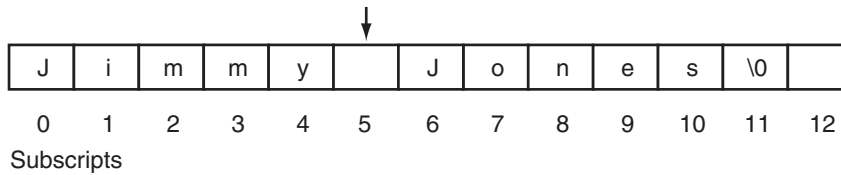
while (userName[count] != ' ' && userName[count] != '\0')
    count++;

```

If the character in `userName[count]` isn't a space or the null terminator, `count` is incremented, and the next character is examined. With the example input "Jimmy Jones," the loop finds the space separating "Jimmy" and "Jones" at `userName[5]`. When the loop stops, `count` is set to 5. This is illustrated in Figure 10-3.

Figure 10-3

The loop stops when `count` reaches 5 because `userName[5]` contains a space



J	i	m	m	y		J	o	n	e	s	\0	
0	1	2	3	4	5	6	7	8	9	10	11	12

Subscripts



NOTE: The loop will also stop if it encounters a null terminator. This is so it will not go beyond the boundary of the array if the user didn't enter a space.

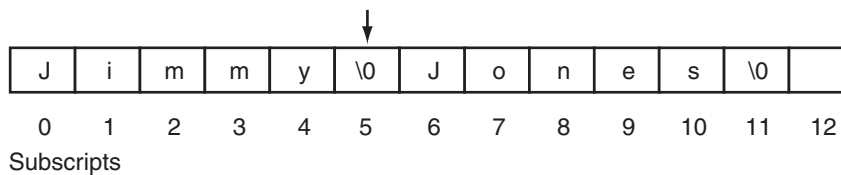
Once the loop has finished, `userName[count]` will either contain a space or a null terminator. If it contains a space, the following `if` statement, in lines 39 and 40, replaces it with a null terminator:

```
if (userName[count] == ' ')
    userName[count] = '\0';
```

This is illustrated in Figure 10-4.

Figure 10-4

The space is replaced with a null terminator. This now becomes the end of the string.



J	i	m	m	y	\0	J	o	n	e	s	\0	
0	1	2	3	4	5	6	7	8	9	10	11	12

Subscripts

The new null terminator now becomes the end of the string.

Using Pointers to Pass C-String Arguments

Pointers are extremely useful for writing functions that process C-strings. If the starting address of a string is passed into a pointer parameter variable, it can be assumed that all the characters, from that address up to the byte that holds the null terminator, are part of the string. (It isn't necessary to know the length of the array that holds the string.)

Program 10-14 demonstrates a function, `countChars`, that uses a pointer to count the number of times a specific character appears in a C-string.

Program 10-14

```
1 // This program demonstrates a function, countChars, that counts
2 // the number of times a specific character appears in a string.
3 #include <iostream>
4 using namespace std;
5
```



```

6  int countChars(char *, char); // Function prototype
7
8  int main()
9  {
10     const int SIZE = 51;      // Array size
11     char userString[SIZE];    // To hold a string
12     char letter;              // The character to count
13
14     // Get a string from the user.
15     cout << "Enter a string (up to 50 characters): ";
16     cin.getline(userString, SIZE);
17
18     // Choose a character whose occurrences within the string will be counted.
19     cout << "Enter a character and I will tell you how many\n";
20     cout << "times it appears in the string: ";
21     cin >> letter;
22
23     // Display the number of times the character appears.
24     cout << letter << " appears ";
25     cout << countChars(userString, letter) << " times.\n";
26     return 0;
27 }
28
29 //*****
30 // Definition of countChars. The parameter strPtr is a pointer *
31 // that points to a string. The parameter Ch is a character that *
32 // the function searches for in the string. The function returns *
33 // the number of times the character appears in the string.      *
34 //*****
35
36 int countChars(char *strPtr, char ch)
37 {
38     int times = 0; // Number of times ch appears in the string
39
40     // Step through the string counting occurrences of ch.
41     while (*strPtr != '\0')
42     {
43         if (*strPtr == ch) // If the current character equals ch...
44             times++;       // ... increment the counter
45         strPtr++;          // Go to the next char in the string.
46     }
47
48     return times;
49 }

```

Program Output with Example Input Shown in Bold

```

Enter a string (up to 50 characters): Starting Out with C++ [Enter]
Enter a character and I will tell you how many
times it appears in the string: t [Enter]
t appears 4 times.

```

In the function `countChars`, `strPtr` points to the C-string that is to be searched and `ch` contains the character to look for. The `while` loop in lines 41 through 46 repeats as long as the character that `strPtr` points to is not the null terminator:

```
while (*strPtr != '\0')
```

Inside the loop, the `if` statement in line 43 compares the character that `strPtr` points to with the character in `ch`:

```
if (*strPtr == ch)
```

If the two are equal, the variable `times` is incremented in line 44. (`times` keeps a running total of the number of times the character appears.) The last statement in the loop is

```
strPtr++;
```

This statement increments the address in `strPtr`. This causes `strPtr` to point to the next character in the string. Then, the loop starts over. When `strPtr` finally reaches the null terminator, the loop terminates, and the function returns the value in `times`.

For another example, see the String Manipulation Case Study, available for download from the book's companion Web site at www.pearsonhighered.com/gaddis.



Checkpoint

10.16 What is the output of the following program?

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

// Function Prototype
void mess(char []);

int main()
{
    char stuff[] = "Tom Talbert Tried Trains";
    cout << stuff << endl;
    mess(stuff);
    cout << stuff << endl;
    return 0;
}

// Definition of function mess
void mess(char str[])
{
    int step = 0;

    while (str[step] != '\0')
    {
        if (str[step] == 'T')
            str[step] = 'D';
        step++;
    }
}
```

10.7 More About the C++ string Class

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



VideoNote
**More About
the string
Class**

The `string` class is an abstract data type. This means it is not a built-in, primitive data type like `int` or `char`. Instead, it is a programmer-defined data type that accompanies the C++ language. It provides many capabilities that make storing and working with strings easy and intuitive.

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>
```

Now you are ready to define 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;
```

You assign a `string` value to the `movieTitle` object with the assignment operator, as shown in the following statement.

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

The contents of `movieTitle` is displayed on the screen with the `cout` object, as shown in the next statement:

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

Program 10-15 is a complete program that demonstrates the statements shown above.

Program 10-15

```
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. For example, Program 10-16 demonstrates how you can use `cin` to read a value from the keyboard into a `string` object.

Program 10-16

```

1  // This program demonstrates how cin can read a string into
2  // a string class object.
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9      string name;
10
11      cout << "What is your name? ";
12      cin >> name;
13      cout << "Good morning " << name << endl;
14      return 0;
15  }
```

Program Output with Example Input Shown in Bold

```

What is your name? Peggy [Enter]
Good morning Peggy
```

Reading a Line of Input into a `string` Object

If you want to read a line of input (with spaces) into a `string` object, use the `getline()` function. Here is an example:

```

string name;
cout << "What is your name? ";
getline(cin, name);
```

The `getline()` function's first argument is the name of a stream object you wish to read the input from. The function call above passes the `cin` object to `getline()`, so the function reads a line of input from the keyboard. The second argument is the name of a `string` object. This is where `getline()` stores the input that it reads.

Comparing and Sorting `string` Objects

There is no need to use a function such as `strcmp` to compare `string` objects. You may use the `<`, `>`, `<=`, `>=`, `==`, and `!=` relational operators. For example, assume the following definitions exist in a program:

```

string set1 = "ABC";
string set2 = "XYZ";
```

The object `set1` is considered less than the object `set2` because the characters "ABC" alphabetically precede the characters "XYZ." So, the following `if` statement will cause the message "set1 is less than set2" to be displayed on the screen.

```

if (set1 < set2)
    cout << "set1 is less than set2.\n";
```

Relational operators perform comparisons on `string` objects in a fashion similar to the way the `strcmp` function compares C-strings. One by one, each character in the first operand is compared with the character in the corresponding position in the second operand. If all the characters in both strings match, the two strings are equal. Other relationships can be determined if two characters in corresponding positions do not match. The first operand is less than the second operand if the mismatched character in the first operand is less than its counterpart in the second operand. Likewise, the first operand is greater than the second operand if the mismatched character in the first operand is greater than its counterpart in the second operand.

For example, assume a program has the following definitions:

```
string name1 = "Mary";
string name2 = "Mark";
```

The value in `name1`, “Mary,” is greater than the value in `name2`, “Mark.” This is because the “y” in “Mary” has a greater ASCII value than the “k” in “Mark.”

`string` objects can also be compared to C-strings with relational operators. Assuming `str` is a `string` object, all of the following are valid relational expressions:

```
str > "Joseph"
"Kimberly" < str
str == "William"
```

Program 10-17 demonstrates `string` objects and relational operators.

Program 10-17

```
1 // This program uses the == operator to compare the string entered
2 // by the user with the valid stereo part numbers.
3 #include <iostream>
4 #include <iomanip>
5 #include <string>
6 using namespace std;
7
8 int main()
9 {
10     const double APRICE = 249.0;    // Price for part A
11     const double BPRICE = 299.0;    // Price for part B
12     string partNum;                 // Part number
13
14     cout << "The stereo part numbers are:\n";
15     cout << "\tBoom Box, part number S147-29A\n";
16     cout << "\tShelf Model, part number S147-29B\n";
17     cout << "Enter the part number of the stereo you\n";
18     cout << "wish to purchase: ";
19     cin >> partNum;
20     cout << fixed << showpoint << setprecision(2);
21
22     if (partNum == "S147-29A")
23         cout << "The price is $" << APRICE << endl;
24     else if (partNum == "S147-29B")
25         cout << "The price is $" << BPRICE << endl;
26     else
27         cout << partNum << " is not a valid part number.\n";
28     return 0;
29 }
```

(program output continues)

Program 10-17 *(continued)***Program Output with Example Input Shown in Bold**

The stereo part numbers are:
 Boom Box, part number S147-29A
 Shelf Model, part number S147-29B
 Enter the part number of the stereo you
 wish to purchase: **S147-29A [Enter]**
 The price is \$249.00

You may also use relational operators to sort string objects. Program 10-18 demonstrates this.

Program 10-18

```

1  // This program uses relational operators to alphabetically
2  // sort two strings entered by the user.
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main ()
8  {
9      string name1, name2;
10
11      // Get a name.
12      cout << "Enter a name (last name first): ";
13      getline(cin, name1);
14
15      // Get another name.
16      cout << "Enter another name: ";
17      getline(cin, name2);
18
19      // Display them in alphabetical order.
20      cout << "Here are the names sorted alphabetically:\n";
21      if (name1 < name2)
22          cout << name1 << endl << name2 << endl;
23      else if (name1 > name2)
24          cout << name2 << endl << name1 << endl;
25      else
26          cout << "You entered the same name twice!\n";
27      return 0;
28  }
```

Program Output with Example Input Shown in Bold

Enter a name (last name first): **Smith, Richard [Enter]**
 Enter another name: **Jones, John [Enter]**
 Here are the names sorted alphabetically:
 Jones, John
 Smith, Richard

Other Ways to Define string Objects

There are a variety of ways to initialize a `string` object when you define it. Table 10-6 shows several example definitions and describes each. Program 10-19 demonstrates a `string` object initialized with the string “William Smith.”

Program 10-19

```

1 // This program initializes a string object.
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 int main()
7 {
8     string greeting;
9     string name("William Smith");
10
11     greeting = "Hello ";
12     cout << greeting << name << endl;
13     return 0;
14 }
```

Program Output

Hello William Smith

Table 10-6

Definition	Description
<code>string address;</code>	Defines an empty <code>string</code> object named <code>address</code> .
<code>string name("William Smith");</code>	Defines a <code>string</code> object named <code>name</code> , initialized with “William Smith.”
<code>string person1(person2);</code>	Defines a <code>string</code> object named <code>person1</code> , which is a copy of <code>person2</code> . <code>person2</code> may be either a <code>string</code> object or character array.
<code>string set1(set2, 5);</code>	Defines a <code>string</code> object named <code>set1</code> , which is initialized to the first five characters in the character array <code>set2</code> .
<code>string lineFull('z', 10);</code>	Defines a <code>string</code> object named <code>lineFull</code> initialized with 10 'z' characters.
<code>string firstName(fullName, 0, 7);</code>	Defines a <code>string</code> object named <code>firstName</code> , initialized with a substring of the <code>string</code> <code>fullName</code> . The substring is seven characters long, beginning at position 0.

Notice in Program 10-19 the use of the `=` operator to assign a value to the `string` object (line 11). The `string` class supports several operators, which are described in Table 10-7.

Table 10-7

Supported Operator	Description
>>	Extracts characters from a stream and inserts them into the string. Characters are copied until a whitespace or the end of the string is encountered.
<<	Inserts the string into a stream.
=	Assigns the string on the right to the string object on the left.
+=	Appends a copy of the string on the right to the string object on the left.
+	Returns a string that is the concatenation of the two string operands.
[]	Implements array-subscript notation, as in name[x]. A reference to the character in the x position is returned.
Relational Operators	Each of the relational operators is implemented: < > <= >= == !=

Program 10-20 demonstrates some of the string operators.

Program 10-20

```
1 // This program demonstrates the C++ string class.
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 int main ()
7 {
8     // Define three string objects.
9     string str1, str2, str3;
10
11     // Assign values to all three.
12     str1 = "ABC";
13     str2 = "DEF";
14     str3 = str1 + str2;
15
16     // Display all three.
17     cout << str1 << endl;
18     cout << str2 << endl;
19     cout << str3 << endl;
20
21     // Concatenate a string onto str3 and display it.
22     str3 += "GHI";
23     cout << str3 << endl;
24     return 0;
25 }
```

Program Output

ABC
DEF
ABCDEF
ABCDEFghi

Using string Class Member Functions

The `string` class also has member functions. For example, the `length` member function returns the length of the string stored in the object. The value is returned as an unsigned integer.

Assume the following `string` object definition exists in a program:

```
string town = "Charleston";
```

The following statement in the same program would assign the value 10 to the variable `x`.

```
x = town.length();
```

Program 10-21 further demonstrates the `length` member function.

Program 10-21

```
1 // This program demonstrates a string
2 // object's length member function.
3 #include <iostream>
4 #include <string>
5 using namespace std;
6
7 int main ()
8 {
9     string town;
10
11     cout << "Where do you live? ";
12     cin >> town;
13     cout << "Your town's name has " << town.length() ;
14     cout << " characters\n";
15     return 0;
16 }
```

Program Output with Example Input Shown in Bold

```
Where do you live? Jacksonville [Enter]
Your town's name has 12 characters
```

The `size` function also returns the length of the string. It is demonstrated in the `for` loop in Program 10-22.

Program 10-22

```
1 // This program demonstrates the C++ string class.
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 int main()
7 {
8     // Define three string objects.
9     string str1, str2, str3;
10
11     // Assign values to all three.
```

(program continues)

Program 10-22 (continued)

```
12     str1 = "ABC";
13     str2 = "DEF";
14     str3 = str1 + str2;
15
16     // Use subscripts to display str3 one character
17     // at a time.
18     for (int x = 0; x < str3.size(); x++)
19         cout << str3[x];
20     cout << endl;
21
22     // Compare str1 with str2.
23     if (str1 < str2)
24         cout << "str1 is less than str2\n";
25     else
26         cout << "str1 is not less than str2\n";
27     return 0;
28 }
```

Program Output

ABCDEF
str1 is less than str2

Table 10-8 lists many of the string class member functions and their overloaded variations. In the examples, assume mystring is the name of a string object.

Table 10-8

Member Function Example	Description
mystring.append(n, 'z');	Appends n copies of 'z' to mystring.
mystring.append(str);	Appends str to mystring. str can be a string object or character array.
mystring.append(str, n);	The first n characters of the character array str are appended to mystring.
mystring.append(str, x, n);	n number of characters from str, starting at position x, are appended to mystring. If mystring is too small, the function will copy as many characters as possible.
mystring.assign(n, 'z');	Assigns n copies of 'z' to mystring.
mystring.assign(str);	Assigns str to mystring. str can be a string object or character array.
mystring.assign(str, n);	The first n characters of the character array str are assigned to mystring.
mystring.assign(str, x, n);	n number of characters from str, starting at position x, are assigned to mystring. If mystring is too small, the function will copy as many characters as possible.
mystring.at(x);	Returns the character at position x in the string.
mystring.back();	Returns the last character in the string. (This member function was introduced in C++ 11.)



Table 10-8 (Continued)

Member Function Example	Description
<code>mystring.begin();</code>	Returns an iterator pointing to the first character in the <code>string</code> . (For more information on iterators, see Chapter 16.)
<code>mystring.c_str();</code>	Converts the contents of <code>mystring</code> to a C-string, and returns a pointer to the C-string.
<code>mystring.capacity();</code>	Returns the size of the storage allocated for the <code>string</code> .
<code>mystring.clear();</code>	Clears the <code>string</code> by deleting all the characters stored in it.
<code>mystring.compare(str);</code>	Performs a comparison like the <code>strcmp</code> function (see Chapter 4), with the same return values. <code>str</code> can be a <code>string</code> object or a character array.
<code>mystring.compare(x, n, str);</code>	Compares <code>mystring</code> and <code>str</code> , starting at position <code>x</code> , and continuing for <code>n</code> characters. The return value is like <code>strcmp</code> . <code>str</code> can be a <code>string</code> object or character array.
<code>mystring.copy(str, x, n);</code>	Copies the character array <code>str</code> to <code>mystring</code> , beginning at position <code>x</code> , for <code>n</code> characters. If <code>mystring</code> is too small, the function will copy as many characters as possible.
<code>mystring.empty();</code>	Returns true if <code>mystring</code> is empty.
<code>mystring.end();</code>	Returns an iterator pointing to the last character of the <code>string</code> in <code>mystring</code> . (For more information on iterators, see Chapter 16.)
<code>mystring.erase(x, n);</code>	Erases <code>n</code> characters from <code>mystring</code> , beginning at position <code>x</code> .
<code>mystring.find(str, x);</code>	Returns the first position at or beyond position <code>x</code> where the <code>string</code> <code>str</code> is found in <code>mystring</code> . <code>str</code> may be either a <code>string</code> object or a character array.
<code>mystring.find('z', x);</code>	Returns the first position at or beyond position <code>x</code> where 'z' is found in <code>mystring</code> .
11 <code>mystring.front();</code>	Returns the first character in the <code>string</code> . (This member function was introduced in C++ 11.)
<code>mystring.insert(x, n, 'z');</code>	Inserts 'z' <code>n</code> times into <code>mystring</code> at position <code>x</code> .
<code>mystring.insert(x, str);</code>	Inserts a copy of <code>str</code> into <code>mystring</code> , beginning at position <code>x</code> . <code>str</code> may be either a <code>string</code> object or a character array.
<code>mystring.length();</code>	Returns the length of the <code>string</code> in <code>mystring</code> .
<code>mystring.replace(x, n, str);</code>	Replaces the <code>n</code> characters in <code>mystring</code> beginning at position <code>x</code> with the characters in <code>string</code> object <code>str</code> .
<code>mystring.resize(n, 'z');</code>	Changes the size of the allocation in <code>mystring</code> to <code>n</code> . If <code>n</code> is less than the current size of the <code>string</code> , the <code>string</code> is truncated to <code>n</code> characters. If <code>n</code> is greater, the <code>string</code> is expanded and 'z' is appended at the end enough times to fill the new spaces.
<code>mystring.size();</code>	Returns the length of the <code>string</code> in <code>mystring</code> .
<code>mystring.substr(x, n);</code>	Returns a copy of a substring. The substring is <code>n</code> characters long and begins at position <code>x</code> of <code>mystring</code> .
<code>mystring.swap(str);</code>	Swaps the contents of <code>mystring</code> with <code>str</code> .

10.8 Focus on Problem Solving and Program Design: A Case Study

As a programmer for the Home Software Company, you are asked to develop a function named `dollarFormat` that inserts commas and a \$ sign at the appropriate locations in a `string` object containing an unformatted dollar amount. As an argument, the function should accept a reference to a `string` object. You may assume the `string` object contains a value such as 1084567.89. The function should modify the `string` object so it contains a formatted dollar amount, such as \$1,084,567.89.

The code for the `dollarFormat` function follows.

```
void dollarFormat(string &currency)
{
    int dp;
    dp = currency.find('.');    // Find decimal point
    if (dp > 3)                // Insert commas
    {
        for (int x = dp - 3; x > 0; x -= 3)
            currency.insert(x, ",");
    }
    currency.insert(0, "$");    // Insert dollar sign
}
```

The function defines an `int` variable named `dp`. This variable is used to hold the position of the unformatted number's decimal point. This is accomplished with the statement:

```
dp = currency.find('.');
```

The `string` class's `find` member function returns the position number in the string where the `'.'` character is found. An `if` statement determines if the number has more than three numbers preceding the decimal point:

```
if (dp > 3)
```

If the decimal point is at a position greater than 3, then the function inserts commas in the string with the following loop:

```
for (int x = dp - 3; x > 0; x -= 3)
    currency.insert(x, ",");
```

Finally, a \$ symbol is inserted at position 0 (the first character in the `string`).

Program 10-23 demonstrates the function.

Program 10-23

```
1 // This program lets the user enter a number. The
2 // dollarFormat function formats the number as
3 // a dollar amount.
4 #include <iostream>
5 #include <string>
6 using namespace std;
7
```

```

 8 // Function prototype
 9 void dollarFormat(string &);
10
11 int main ()
12 {
13     string input;
14
15     // Get the dollar amount from the user.
16     cout << "Enter a dollar amount in the form nnnnn.nn : ";
17     cin >> input;
18     dollarFormat(input);
19     cout << "Here is the amount formatted:\n";
20     cout << input << endl;
21     return 0;
22 }
23
24 //*****
25 // Definition of the dollarFormat function. This function *
26 // accepts a string reference object, which is assumed *
27 // to hold a number with a decimal point. The function *
28 // formats the number as a dollar amount with commas and *
29 // a $ symbol. *
30 //*****
31
32 void dollarFormat(string &currency)
33 {
34     int dp;
35
36     dp = currency.find('.'); // Find decimal point
37     if (dp > 3) // Insert commas
38     {
39         for (int x = dp - 3; x > 0; x -= 3)
40             currency.insert(x, ",");
41     }
42     currency.insert(0, "$"); // Insert dollar sign
43 }

```

Program Output with Example Input Shown in Bold

```

Enter a dollar amount in the form nnnnn.nn: 1084567.89 [Enter]
Here is the amount formatted:
$1,084,567.89

```

Review Questions and Exercises

Short Answer

1. What header file must you include in a program using character testing functions such as `isalpha` and `isdigit`?
2. What header file must you include in a program using the character conversion functions `toupper` and `tolower`?

3. Assume `c` is a `char` variable. What value does `c` hold after each of the following statements executes?

<i>Statement</i>	<i>Contents of <code>c</code></i>
<code>c = toupper('a');</code>	_____
<code>c = toupper('B');</code>	_____
<code>c = tolower('D');</code>	_____
<code>c = toupper('e');</code>	_____

4. Look at the following code. What value will be stored in `s` after the code executes?

```
char name[10];
int s;
strcpy(name, "Jimmy");
s = strlen(name);
```

5. What header file must you include in a program using string functions such as `strlen` and `strcpy`?
6. What header file must you include in a program using string/numeric conversion functions such as `atoi` and `atof`?
7. What header file must you include in a program using `string` class objects?
8. How do you compare `string` class objects?

Fill-in-the-Blank

9. The _____ function returns true if the character argument is uppercase.
10. The _____ function returns true if the character argument is a letter of the alphabet.
11. The _____ function returns true if the character argument is a digit.
12. The _____ function returns true if the character argument is a whitespace character.
13. The _____ function returns the uppercase equivalent of its character argument.
14. The _____ function returns the lowercase equivalent of its character argument.
15. The _____ file must be included in a program that uses character testing functions.
16. The _____ function returns the length of a string.
17. To _____ two strings means to append one string to the other.
18. The _____ function concatenates two strings.
19. The _____ function copies one string to another.
20. The _____ function searches for a string inside of another one.
21. The _____ function compares two strings.
22. The _____ function copies, at most, n number of characters from one string to another.
23. The _____ function returns the value of a string converted to an integer.
24. The _____ function returns the value of a string converted to a long integer.
25. The _____ function returns the value of a string converted to a float.
26. The _____ function converts an integer to a string.

Algorithm Workbench

27. The following `if` statement determines whether choice is equal to 'Y' or 'y'.

```
if (choice == 'Y' || choice == 'y')
```

Simplify this statement by using either the `toupper` or `tolower` function.

28. Assume `input` is a `char` array holding a C-string. Write code that counts the number of elements in the array that contain an alphabetic character.
29. Look at the following array definition.

```
char str[10];
```

Assume that `name` is also a `char` array, and it holds a C-string. Write code that copies the contents of `name` to `str` if the C-string in `name` is not too big to fit in `str`.

30. Look at the following statements.

```
char str[] = "237.89";  
double value;
```

Write a statement that converts the string in `str` to a double and stores the result in `value`.

31. Write a function that accepts a pointer to a C-string as its argument. The function should count the number of times the character 'w' occurs in the argument and return that number.
32. Assume that `str1` and `str2` are string class objects. Write code that displays "They are the same!" if the two objects contain the same string.

True or False

33. T F Character testing functions, such as `isupper`, accept strings as arguments and test each character in the string.
34. T F If `toupper`'s argument is already uppercase, it is returned as is, with no changes.
35. T F If `tolower`'s argument is already lowercase, it will be inadvertently converted to uppercase.
36. T F The `strlen` function returns the size of the array containing a string.
37. T F If the starting address of a C-string is passed into a pointer parameter, it can be assumed that all the characters, from that address up to the byte that holds the null terminator, are part of the string.
38. T F C-string handling functions accept as arguments pointers to strings (array names or pointer variables), or literal strings.
39. T F The `strcat` function checks to make sure the first string is large enough to hold both strings before performing the concatenation.
40. T F The `strcpy` function will overwrite the contents of its first string argument.
41. T F The `strcpy` function performs no bounds checking on the first argument.
42. T F There is no difference between "847" and 847.

Find the Errors

Each of the following programs or program segments has errors. Find as many as you can.

- ```
43. char str[] = "Stop";
 if (isupper(str) == "STOP")
 exit(0);

44. char numeric[5];
 int x = 123;
 numeric = atoi(x);

45. char string1[] = "Billy";
 char string2[] = " Bob Jones";
 strcat(string1, string2);

46. char x = 'a', y = 'a';
 if (strcmp(x, y) == 0)
 exit(0);
```

## Programming Challenges

### 1. String Length

Write a function that returns an integer and accepts a pointer to a C-string as an argument. The function should count the number of characters in the string and return that number. Demonstrate the function in a simple program that asks the user to input a string, passes it to the function, and then displays the function's return value.

### 2. Backward String

Write a function that accepts a pointer to a C-string as an argument and displays its contents backward. For instance, if the string argument is "Gravity" the function should display "yrtivraG". Demonstrate the function in a program that asks the user to input a string and then passes it to the function.

### 3. Word Counter

Write a function that accepts a pointer to a C-string as an argument and returns the number of words contained in the string. For instance, if the string argument is "Four score and seven years ago" the function should return the number 6. Demonstrate the function in a program that asks the user to input a string and then passes it to the function. The number of words in the string should be displayed on the screen. *Optional Exercise:* Write an overloaded version of this function that accepts a string class object as its argument.

### 4. Average Number of Letters

Modify the program you wrote for Problem 3 (Word Counter), so it also displays the average number of letters in each word.

### 5. Sentence Capitalizer

Write a function that accepts a pointer to a C-string as an argument and capitalizes the first character of each sentence in the string. For instance, if the string argument is "hello. my name is Joe. what is your name?" the function should manipulate the string so it contains "Hello. My name is Joe. What is your name?"



VideoNote  
Solving the  
Backward  
String  
Problem



Demonstrate the function in a program that asks the user to input a string and then passes it to the function. The modified string should be displayed on the screen. *Optional Exercise:* Write an overloaded version of this function that accepts a string class object as its argument.

## 6. Vowels and Consonants

Write a function that accepts a pointer to a C-string as its argument. The function should count the number of vowels appearing in the string and return that number.

Write another function that accepts a pointer to a C-string as its argument. This function should count the number of consonants appearing in the string and return that number.

Demonstrate these two functions in a program that performs the following steps:

1. The user is asked to enter a string.
2. The program displays the following menu:
  - A) Count the number of vowels in the string
  - B) Count the number of consonants in the string
  - C) Count both the vowels and consonants in the string
  - D) Enter another string
  - E) Exit the program
3. The program performs the operation selected by the user and repeats until the user selects E to exit the program.

## 7. Name Arranger

Write a program that asks for the user's first, middle, and last names. The names should be stored in three different character arrays. The program should then store, in a fourth array, the name arranged in the following manner: the last name followed by a comma and a space, followed by the first name and a space, followed by the middle name. For example, if the user entered "Carol Lynn Smith", it should store "Smith, Carol Lynn" in the fourth array. Display the contents of the fourth array on the screen.

## 8. Sum of Digits in a String

Write a program that asks the user to enter a series of single digit numbers with nothing separating them. Read the input as a C-string or a `string` object. The program should display the sum of all the single-digit numbers in the string. For example, if the user enters 2514, the program should display 12, which is the sum of 2, 5, 1, and 4. The program should also display the highest and lowest digits in the string.

## 9. Most Frequent Character

Write a function that accepts either a pointer to a C-string, or a `string` object, as its argument. The function should return the character that appears most frequently in the string. Demonstrate the function in a complete program.

## 10. replaceSubstring Function

Write a function named `replaceSubstring`. The function should accept three C-string or `string` object arguments. Let's call them *string1*, *string2*, and *string3*. It should search *string1* for all occurrences of *string2*. When it finds an occurrence of

`string2`, it should replace it with `string3`. For example, suppose the three arguments have the following values:

```
string1: "the dog jumped over the fence"
string2: "the"
string3: "that"
```

With these three arguments, the function would return a `string` object with the value "that dog jumped over that fence." Demonstrate the function in a complete program.

### 11. Case Manipulator

Write a program with three functions: `upper`, `lower`, and `reverse`. The `upper` function should accept a pointer to a C-string as an argument. It should step through each character in the string, converting it to uppercase. The `lower` function, too, should accept a pointer to a C-string as an argument. It should step through each character in the string, converting it to lowercase. Like `upper` and `lower`, `reverse` should also accept a pointer to a string. As it steps through the string, it should test each character to determine whether it is upper- or lowercase. If a character is uppercase, it should be converted to lowercase. Likewise, if a character is lowercase, it should be converted to uppercase.

Test the functions by asking for a string in function `main`, then passing it to them in the following order: `reverse`, `lower`, and `upper`.

### 12. Password Verifier

Imagine you are developing a software package that requires users to enter their own passwords. Your software requires that users' passwords meet the following criteria:

- The password should be at least six characters long.
- The password should contain at least one uppercase and at least one lowercase letter.
- The password should have at least one digit.

Write a program that asks for a password and then verifies that it meets the stated criteria. If it doesn't, the program should display a message telling the user why.

### 13. Date Printer

Write a program that reads a string from the user containing a date in the form `mm/dd/yyyy`. It should print the date in the form `March 12, 2014`.

### 14. Word Separator

Write a program that accepts as input a sentence in which all of the words are run together, but the first character of each word is uppercase. Convert the sentence to a string in which the words are separated by spaces and only the first word starts with an uppercase letter. For example the string `"StopAndSmellTheRoses."` would be converted to `"Stop and smell the roses."`

### 15. Character Analysis

If you have downloaded this book's source code from the companion Web site, you will find a file named `text.txt` in the Chapter 10 folder. (The companion Web site is at [www.pearsonhighered.com/gaddis](http://www.pearsonhighered.com/gaddis).) Write a program that reads the file's contents and determines the following:

- The number of uppercase letters in the file
- The number of lowercase letters in the file
- The number of digits in the file

## 16. Pig Latin

Write a program that reads a sentence as input and converts each word to “Pig Latin.” In one version, to convert a word to Pig Latin you remove the first letter and place that letter at the end of the word. Then you append the string “ay” to the word. Here is an example:

English:        I SLEPT MOST OF THE NIGHT  
Pig Latin:     IAY LEPTSAY OSTMAY FOAY HETAY IGHNTAY

## 17. Morse Code Converter

Morse code is a code where each letter of the English alphabet, each digit, and various punctuation characters are represented by a series of dots and dashes. Table 10-9 shows part of the code.

Write a program that asks the user to enter a string, and then converts that string to Morse code.

**Table 10-9** Morse Code

| Character     | Code         | Character | Code   | Character | Code  | Character | Code  |
|---------------|--------------|-----------|--------|-----------|-------|-----------|-------|
| space         | <i>space</i> | 6         | -....  | G         | ---.  | Q         | ---.- |
| comma         | --...--      | 7         | --...- | H         | ....  | R         | .-.-  |
| period        | .-.-.-       | 8         | ----.  | I         | ..    | S         | ...-  |
| question mark | ..-.-.-      | 9         | -----  | J         | .---- | T         | -     |
| 0             | -----        | A         | .-     | K         | -.-   | U         | ..-   |
| 1             | .-----       | B         | -...   | L         | .-... | V         | ...-- |
| 2             | ..----       | C         | -.-.   | M         | --    | W         | .---  |
| 3             | ...---       | D         | -..    | N         | -..   | X         | -.-.- |
| 4             | ....-        | E         | .      | O         | ---   | Y         | -.--  |
| 5             | .....        | F         | ..-.   | P         | .-.-. | Z         | ---.. |

## 18. Phone Number List

Write a program that has an array of at least 10 `string` objects that hold people’s names and phone numbers. You may make up your own strings, or use the following:

```
"Alejandra Cruz, 555-1223"
"Joe Looney, 555-0097"
"Geri Palmer, 555-8787"
"Li Chen, 555-1212"
"Holly Gaddis, 555-8878"
"Sam Wiggins, 555-0998"
"Bob Kain, 555-8712"
"Tim Haynes, 555-7676"
"Warren Gaddis, 555-9037"
"Jean James, 555-4939"
"Ron Palmer, 555-2783"
```

The program should ask the user to enter a name or partial name to search for in the array. Any entries in the array that match the string entered should be displayed. For example, if the user enters “Palmer” the program should display the following names from the list:

```
Geri Palmer, 555-8787
Ron Palmer, 555-2783
```

### 19. Check Writer

Write a program that displays a simulated paycheck. The program should ask the user to enter the date, the payee’s name, and the amount of the check (up to \$10,000). It should then display a simulated check with the dollar amount spelled out, as shown here:

Date: 11/24/2014

Pay to the Order of: John Phillips \$1920.85

One thousand nine hundred twenty and 85 cents

Be sure to format the numeric value of the check in fixed-point notation with two decimal places of precision. Be sure the decimal place always displays, even when the number is zero or has no fractional part. Use either C-strings or `string` class objects in this program.

*Input Validation: Do not accept negative dollar amounts, or amounts over \$10,000.*