Objectives

The objectives of this lab are to cover the following:

- C-Strings
- C-Strings Stored in Arrays
- Library Functions for Working with C-Strings
- C-String/Numeric Conversion Functions
- Your Own C-String-Handling Functions

C-Strings

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.

C-string is a string whose characters are stored in consecutive memory locations and are followed by a null character, or **null terminator**. **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.

Illustrates how the string literal **"Bailey"** is stored in memory, as a C-string.

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.

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.
- 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;
```

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.

```
// This program displays a string stored in a char array.
 #include <iostream>
 using namespace std;

 int main()
 {
 const int SIZE = 80; // Array size
 char line[SIZE]; // To hold a line of input
 int count = 0; // Loop counter variable

 // Get a line of input.
 cout << "Enter a sentence of no more than "<< (SIZE - 1) << " characters:\n";
 cin.getline(line, SIZE);

// Display the input one character at a time.
 cout << "The sentence you entered is:\n";
 while (line[count] != '\0')
 {
 cout << line[count];
 count++;
  }
 return 0;
 }
```

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!

Library Functions for Working with C-Strings

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.

Function	Description
`strlen`	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.) *Example Usage:* `len = strlen(name);`
`strcat`	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.) *Example Usage:* `strcat(string1, string2);`
`strcpy`	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. *Example Usage:* `strcpy(string1, string2);`
`strncat`	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. *Example Usage:* `strncat(string1, string2, n);`
`strncpy`	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 `n` is less than the length of `string2`, the null terminator is not automatically appended to `string1`. If `n` is greater than the length of `string2`, `string1` is padded with '\0' characters. *Example Usage:* `strncpy(string1, string2, n);`
`strcmp`	Accepts two C-strings or pointers to two C-strings arguments. If `string1` and `string2` are the same, this function returns 0. If `string2` is alphabetically greater than `string1`, it returns a negative number. If `string2` is alphabetically less than `string1`, it returns a positive number. *Example Usage:* `if (strcmp(string1, string2))`
`strstr`	Accepts two C-strings or pointers to two C-strings as arguments. Searches for the first occurrence of `string2` in `string1`. If an occurrence of `string2` is found, the function returns a pointer to it. Otherwise, it returns `nullptr` (address 0). *Example Usage:* `cout << strstr(string1, string2);`

C-String/Numeric Conversion Functions

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.

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

Your Own C-String-Handling Functions

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 uses a function to copy a C-string from one array to another.

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

 void stringCopy(char [], char []); // Function prototype

 int main()
 {
 const int LENGTH = 30; // Size of the arrays
 char first[LENGTH]; // To hold the user's input
 char second[LENGTH]; // To hold the copy

 // Get a string from the user and store in first.
 cout << "Enter a string with no more than "
 << (LENGTH - 1) << " characters:\n";
cin.getline(first, LENGTH);

 // Copy the contents of first to second.
stringCopy(first, second);

 // Display the copy.
 cout << "The string you entered is:\n" << second << endl;
 return 0;
}



void stringCopy(char string1[], char string2[])
 {
 int index = 0; // Loop counter

 // Step through string1, copying each element to
 // string2. Stop when the null character is encountered.
 while (string1[index] != '\0')
 {
 string2[index] = string1[index];
 index++;
 }

 // Place a null character in string2.
 string2[index] = '\0';
 }
```

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!