Character Arrays and Strings

Even though C++ has no string variables, you can act as if C++ has them by using character arrays. The concept of arrays might be new to you, but this chapter explains how easy they are to declare and use. After you declare these arrays, they can hold character strings—just as if they were real string variables. This chapter includes

- Character arrays
- Comparison of character arrays and strings
- Examples of character arrays and strings

After you master this chapter, you are on your way to being able to manipulate almost every type of variable C++ offers. Manipulating characters and words is one feature that separates your computer from a powerful calculator; this capability gives computers true data-processing capabilities.

Character Arrays

A string literal can be stored in an array of characters.

Almost every type of data in C++ has a variable, but there is no variable for holding character strings. The authors of C++ realized that you need some way to store strings in variables, but instead of storing them in a string variable (as some languages such as BASIC or Pascal do) you must store them in an *array* of characters.

If you have never programmed before, an array might be new to you. An array is a list (sometimes called a *table*) of variables, and most programming languages allow the use of such lists. Suppose you had to keep track of the sales records of 100 salespeople. You could make up 100 variable names and assign a different salesperson's sales record to each one.

All those different variable names, however, are difficult to track. If you were to put them in an array of floating-point variables, you would have to keep track of only a single name (the array name) and reference each of the 100 values by a numeric subscript.

The last few chapters of this book cover array processing in more detail. However, to work with character string data in your early programs, you have to become familiar with the concept of *character arrays.*

Because a string is simply a list of one or more characters, a character array is the perfect place to hold strings of information. Suppose you want to keep track of a person's full name, age, and salary in variables. The age and salary are easy because there are variable types that can hold such data. The following code declares those two variables:

```
int age;
float salary;
```

You have no string variable to hold the name, but you can create an appropriate array of characters (which is actually one or more character variables in a row in memory) with the following declaration:

```
char name[15];
```

This reserves a character array. An array declaration always includes brackets (`[]`) that declare the space for the array. This array is 15 characters long. The array name is `name`. You also can assign a

value to the character array at the time you declare the array. The following declaration statement not only declares the character array, but also assigns the name "Michael Jones" at the same time:

Declare the character array called `name` *as 15 characters long, and assign* `Michael Jones` *to the array.*

```
char name[15]="Michael Jones";
```

Figure 5.1 shows what this array looks like in memory. Each of the 15 boxes of the array is called an *element.* Notice the null zero (the string-terminating character) at the end of the string. Notice also that the last character of the array contains no data. You filled only the first 14 elements of the array with the data and the data's null zero. The 15th element actually has a value in it—but whatever follows the string's null zero is not a concern.

Figure 5.1. A character array after being declared and assigned a string value.

You can access individual elements in an array, or you can access the array as a whole. This is the primary advantage of an array over the use of many differently named variables. You can assign values to the individual array elements by putting the elements' location, called a *subscript,* in brackets, as follows:

```
name[3]='k';
```

This overwrites the `h` in the name `Michael` with a `k`. The string now looks like the one in Figure 5.2.

Figure 5.2. The array contents (see Figure 5.1) after changing one of the elements.

All array subscripts begin at 0.

All array subscripts start at zero. Therefore, to overwrite the first element, you must use `0` as the subscript. Assigning `name[3]` (as is done in Figure 5.2) changes the value of the fourth element in the array.

You can print the entire string—or, more accurately, the entire array—with a single `cout` statement, as follows:

```
cout << name;
```

Notice when you print an array, you do not include brackets after the array name. You must be sure to reserve enough characters in the array to hold the entire string. The following line,

```
char name[5]="Michael Jones";
```

is incorrect because it reserves only five characters for the array, whereas the name and its null zero require 14 characters. However, C++ does give you an error message for this mistake (`illegal initialization`).

CAUTION: Always reserve enough array elements to hold the string, plus its null-terminating character. It is easy to forget the null character, but don't do it!

If your string contains 13 characters, it also must have a 14th for the null zero or it will never be treated like a string. To help eliminate this error, C++ gives you a shortcut. The following two character array statements are the same:

```
char horse[9]="Stallion";
```

and

```
char horse[]="Stallion";
```

If you assign a value to a character array at the same time you declare the array, C++ counts the string's length, adds one for the null zero, and reserves the array space for you.

If you do not assign a value to an array at the time it is declared, you cannot declare it with empty brackets. The following statement,

```
char people[];
```

does not reserve any space for the array called `people`. Because you did not assign a value to the array when you declared it, C++ assumes this array contains zero elements. Therefore, you have no room to put values in this array later. Most compilers generate an error if you attempt this.

Character Arrays Versus Strings

In the previous section, you saw how to put a string in a character array. Strings can exist in C++ only as string *literals,* or as stored information in character arrays. At this point, you have only to understand that strings must be stored in character arrays. As you read through this book and become more familiar with arrays and strings, however, you should become more comfortable with their use.

NOTE: Strings must be stored in character arrays, but not all character arrays contain strings.

Look at the two arrays shown in Figure 5.3. The first one, called `cara1`, is a character array, but it does not contain a string. Rather than a string, it contains a list of several characters. The second array, called `cara2`, contains a string because it has a null zero at its end.

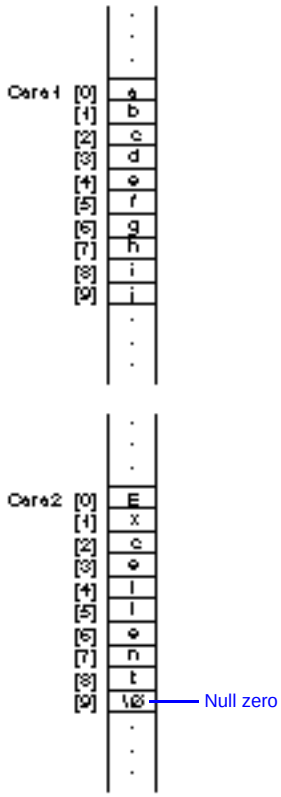


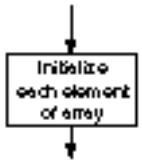
Figure 5.3. Two character arrays: `Cara1` contains characters, and `Cara2` contains a character string.

You could initialize these arrays with the following assignment statements.

Declare the array `cara1` *with 10 individual characters.*
Declare the array `cara2` *with the character string* `"Excellent"`.

```
char cara1[10]={'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i',
                'j'};
char cara2[10]="Excellent";
```

If you want to put only individual characters in an array, you must enclose the list of characters in braces, as shown. You could initialize `cara1` later in the program, using assignment statements, as the following code section does.




```
char cara1[10];
cara1[0]='a';
cara1[1]='b';
cara1[2]='c';
cara1[3]='d';
cara1[4]='e';
cara1[5]='f';
cara1[6]='g';
cara1[7]='h';
cara1[8]='i';
cara1[9]='j';   // Last element possible with subscript of nine.
```

Because the `cara1` character array does not contain a null zero, it does not contain a string of characters. It does contain characters that can be stored in the array—and used individually—but they cannot be treated in a program as if they were a string.

CAUTION: You cannot assign string values to character arrays in a regular assignment statement, except when you first declare the character arrays.

Because a character array is not a string variable (it can be used only to hold a string), it cannot go on the left side of an equal (=) sign. The program that follows is invalid:

```
#include <iostream.h>
main()
{
   char petname[20];      // Reserve space for the pet's name.
   petname = "Alfalfa";   // INVALID!
   cout << petname;       // The program will never get here.
   return;
}
```

Because the pet's name was not assigned *at the time the character array was declared,* it cannot be assigned a value later. The following is allowed, however, because you can assign values individually to a character array:

```
#include <iostream.h>
main()
{
   char petname[20];    // Reserve space for the pet's name.
   petname[0]='A';  // Assign values one element at a time.
   petname[1]='l';
   petname[2]='f';
   petname[3]='a';
   petname[4]='l';
   petname[5]='f';
   petname[6]='a';
   petname[7]='\0';  // Needed to ensure this is a string!
   cout <<petname;   // Now the pet's name prints properly.
   return;
}
```

The `petname` character array now holds a string because the last character is a null zero. How long is the string in `petname`? It is seven characters long because the length of a string never includes the null zero.

You cannot assign more than 20 characters to this array because its reserved space is only 20 characters. However, you can store any string of 19 (leaving one for the null zero) or fewer characters to the array. If you assign the `"Alfalfa"` string in the array as shown, and then assign a null zero to `petname[3]` as in:

```
petname[3]='\0';
```

the string in petname is now only three characters long. You have, in effect, shortened the string. There are still 20 characters reserved for petname, but the data inside it is the string "Alf" ending with a null zero.

There are many other ways to assign a value to a string. You can use the strcpy() function, for example. This is a built-in function that enables you to copy a string literal in a string. To copy the "Alfalfa" pet name into the petname array, you type:

```
strcpy(petname, "Alfalfa");  // Copies Alfalfa into the array.
```

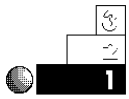
The strcpy() function puts string literals in string arrays.

The strcpy() ("string copy") function assumes that the first value in the parentheses is a character array name, and that the second value is a valid string literal or another character array that holds a string. You must be sure that the first character array in the parentheses is long enough (in number of reserved elements) to hold whatever string you copy into it.

NOTE: Place an #include <string.h> line before the main() function in programs that use strcpy() or any other built-in string functions mentioned in this book. Your compiler supplies the string.h file to help the strcpy() function work properly. The #include files such as iostream.h and string.h will be further explained as you progress through this book.

Other methods of initializing arrays are explored throughout the rest of this book.

Examples



1. Suppose you want to keep track of your aunt's name in a program so you can print it. If your aunt's name is Ruth Ann Cooper, you have to reserve at least 16 elements—15 to hold the name and one to hold the null character. The following statement properly reserves a character array to hold her name:

   ```
   char aunt_name[16];
   ```

2. If you want to put your aunt's name in the array at the same time you reserve the array space, you could do it like this:

```
char aunt_name[16]="Ruth Ann Cooper";
```

You could also leave out the array size and allow C++ to count the number needed:

```
char aunt_name[]="Ruth Ann Cooper";
```

3. Suppose you want to keep track of the names of three friends. The longest name is 20 characters (including the null zero). You simply have to reserve enough character-array space to hold each friend's name. The following code does the trick:

```
char friend1[20];
char friend2[20];
char friend3[20];
```

These array declarations should appear toward the top of the block, along with any integer, floating-point, or character variables you have to declare.

4. The next example asks the user for a first and last name. Use the `cin` operator (the opposite of `cout`) to retrieve data from the keyboard. Chapter 7, "Simple I/O," more fully explains the `cout` and `cin` operators. The program then prints the user's initials on-screen by printing the first character of each name in the array. The program must print each array's `0` subscript because the first subscript of any array begins at 0, not 1.

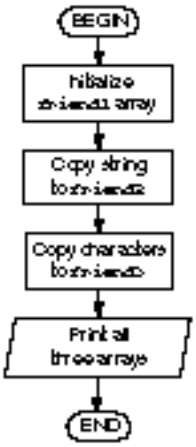
```
// Filename: C5INIT.CPP
// Print the user's initials.
#include <iostream.h>
main()
{
   char first[20];   // Holds the first name
   char last[20];    // Holds the last name

   cout << "What is your first name? \n";
   cin >> first;
```

```
   cout << "What is your last name? \n";
   cin >> last;

   // Print the initials
   cout << "Your initials are " << first[0] << " "
        << last[0];
   return 0;
}
```

5. The following program takes your three friends' character arrays and assigns them string values by using the three methods shown in this chapter. Notice the extra `#include` file used with the string function `strcpy()`.




```
// Filename: C5STR.CPP
// Store and initialize three character arrays for three
friends.

#include <iostream.h>
#include <string.h>
main()
{
   // Declare all arrays and initialize the first one.
   char friend1[20]="Jackie Paul Johnson";
   char friend2[20];
   char friend3[20];

// Use a function to initialize the second array.
   strcpy(friend2, "Julie L. Roberts");

   friend3[0]='A';  // Initialize the last,
   friend3[1]='d';  // an element at a time.
   friend3[2]='a';
   friend3[3]='m';
   friend3[4]=' ';
   friend3[5]='G';
   friend3[6]='.';
   friend3[7]=' ';
   friend3[8]='S';
   friend3[9]='m';
   friend3[10]='i';
```

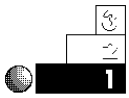
```
    friend3[11]='t';
    friend3[12]='h';
    friend3[13]='\0';

    // Print all three names.
    cout << friend1 << "\n";
    cout << friend2 << "\n";
    cout << friend3 << "\n";
    return 0;
}
```

The last method of initializing a character array with a string—one element at a time—is not used as often as the other methods.

Review Questions

The answers to the review questions are in Appendix B.



1. How would you declare a character array called `my_name` that holds the following string literal?

   `"This is C++"`

2. How long is the string in Question 1?
3. How many bytes of storage does the string in Question 1 take?

4. With what do all string literals end?
5. How many variables do the following statements declare, and what are their types?

```
char name[25];
char address[25];
```

6. True or false: The following statement assigns a string literal to a character array.

```
myname[]="Kim Langston";
```

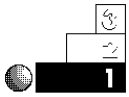
7. True or false: The following declaration puts a string in the character array called `city`.

```
char city[]={'M', 'i', 'a', 'm', 'i', '\0'};
```

8. True or false: The following declaration puts a string in the character array called `city`.

```
char city[]={'M', 'i', 'a', 'm', 'i'};
```

Review Exercises



1. Write the C++ code to store your weight, height (in feet), shoe size, and name with four variables. Declare the variables, then assign their values in the body of your program.

2. Rewrite the program in Exercise 1, adding proper `printf()` statements to print the values. Use appropriate messages (by printing string literals) to describe the printed values.

3. Write a program to store and print the names of your two favorite television programs. Store these programs in two character arrays. Initialize one of the strings (assign it the first program's name) at the time you declare the array. Initialize the second value in the body of the program with the `strcpy()` function.

4. Write a program that puts 10 different initials in 10 elements of a single character array. Do not store a null zero. Print the list backward, one initial on each line.

Summary

This has been a short, but powerful chapter. You learned about character arrays that hold strings. Even though C++ has no string variables, character arrays can hold string literals. After you put a string in a character array, you can print or manipulate it as if it were a string.

Starting with the next chapter, you begin to hone the C++ skills you are building. Chapter 6, “Preprocessor Directives,” introduces preprocessor directives, which are not actually part of the C++ language but help you work with your source code before your program is compiled.