character of the array is stored at `my_word + 1`. Using the dereferencing operator and this knowledge of how the array is stored, you can change the fourth character in the array using a statement like the one below.

```
*(my_word + 3) = 'n';  // has the same result as my_word[3] = 'n';
```

You can see that subscript notation makes for more readable code than the dereferencing operator. But as statements like the one above begin to make sense to you, you will begin to unlock the real power of C++. Let's add the statement to the program you entered in Exercise 10-4.

EXERCISE 10-5 CHANGING ARRAY CHARACTERS

1. Add the following lines to the bottom of the program on your screen.

```
*(my_word + 3) = 'n';  // has the same result as my_word[3] = 'n';
cout << my_word << '\n';
```

2. Run the program to see the effect of the new statements.
3. Add a statement that uses subscript notation to change the word to *sown*. Add an output statement to output the new word.
4. Run, save, and close.

SECTION 10.2 QUESTIONS

1. The name of a character array is what kind of pointer?
2. Why can't you assign a string to a character array using the assignment operator?
3. What is the name of the method that allows you to access individual characters of a character array using brackets ([])?
4. Given the character array declared as `char A[8] = "ABCDEFG";`, what character is returned by `A[2]`?
5. Using the same character array you used for question 4, what would be the resulting string if the following statement were executed?

```
A[1] = 'X';
```

PROBLEM 10.2.1

Write a program that declares a character array named `alphabet` and initializes the array to "ABCDEFGHIJKLMNOPQRSTUVWXYZ." In your program, include a loop that replaces one character of the array at a time with the lowercase letters a–z. Print the character array to the screen during each iteration of the loop. *Hint:* Remember that a character array is an array of integer values. Use ASCII values to make the changes. Save the source code file as *ALPHABET.CPP*.

Using enum

The *enum* keyword (short for enumerated) is a C++ feature that is often overlooked. It allows you to create your own simple data types for special purposes in your program. For example, you could create a data type called colors that allows only the values *red*, *green*, *blue*, and *yellow* as data. In this section, you will learn how enum works and how you can use it in your programs.

HOW TO USE enum

The enum keyword is easy to use. You simply create a type, give it a name, and tell the compiler what values your new data type will accept. Consider the statement below.

```
enum sizes {small, medium, large, jumbo};
```

The data type called `sizes` can have one of four values: *small*, *medium*, *large*, or *jumbo*. The next step is to declare a variable that uses `sizes` as a type. Let's declare two variables of the type `sizes`.

```
enum sizes drink_size, popcorn_size;
```

The variable `drink_size` and `popcorn_size` are of type `sizes` and can be assigned one of the four sizes defined in the `sizes` type.

EXERCISE 10-6 USING enum

1. Enter the program below. Save the source code as *ENUMTEST.CPP*.

```
#include <iostream.h> // necessary for cout command

int main()
{
  enum sizes {small, medium, large, jumbo};
  sizes drink_size, popcorn_size;

  drink_size = large;
  popcorn_size = jumbo;

  if (drink_size == large)
   { cout << "You could have a jumbo for another quarter.\n"; }

  if ((popcorn_size == jumbo) && (drink_size != jumbo))
   { cout << "You need more drink to wash down a jumbo popcorn.\n"; }
  return 0;
}
```

2. Run the program to see the output. Close the source code file.