I've included const in the declaration of s to indicate that count_spaces doesn't change the array that s represents. If s were not a string, the function would need a second argument specifying the length of the array. Since s is a string, however, count_spaces can determine where it ends by testing for the null character.

Many C programmers wouldn't write count_spaces as we have. Instead, they'd use a pointer to keep track of the current position within the string. As we saw in Section 12.2, this technique is always available for processing arrays, but it proves to be especially convenient for working with strings.

Let's rewrite the count_spaces function using pointer arithmetic instead of array subscripting. We'll eliminate the variable i and use s itself to keep track of our position in the string. By incrementing s repeatedly, count_spaces can step through each character in the string. Here's our new version of the function:

```
int count_spaces(const char *s)
{
  int count = 0;

  for (; *s != '\0'; s++)
    if (*s == ' ')
      count++;
  return count;
}
```

Note that const doesn't prevent count_spaces from modifying s; it's there to prevent the function from modifying what s points to. And since s is a copy of the pointer that's passed to count_spaces, incrementing s doesn't affect the original pointer.

The count_spaces example raises some questions about how to write string functions:

- *Is it better to use array operations or pointer operations to access the characters in a string?* We're free to use whichever is more convenient; we can even mix the two. In the second version of count_spaces, treating s as a pointer simplifies the function slightly by removing the need for the variable i. Traditionally, C programmers lean toward using pointer operations for processing strings.
- *Should a string parameter be declared as an array or as a pointer?* The two versions of count_spaces illustrate the options: the first version declares s to be an array; the second declares s to be a pointer. Actually, there's no difference between the two declarations—recall from Section 12.3 that the compiler treats an array parameter as though it had been declared as a pointer.
- *Does the form of the parameter* (s[] *or* *s) *affect what can be supplied as an argument?* No. When count_spaces is called, the argument could be an array name, a pointer variable, or a string literal—count_spaces can't tell the difference.