Now the program echoes and counts every character, including the spaces. Input still is buffered, so it still is possible to type more input than what eventually reaches the program.

If you are familiar with C, this program may strike you as terribly wrong! The `cin.get(ch)` call places a value in the `ch` variable, which means it alters the value of the variable. In C, you must pass the address of a variable to a function if you want to change the value of that variable. But the call to `cin.get()` in Listing 5.16 passes `ch`, not `&ch`. In C, code like this won't work. In C++ it can, provided that the function declares the argument as a `reference`. This is a feature type new to C++. The `iostream` header file declares the argument to `cin.get(ch)` as a `reference` type, so this function can alter the value of its argument. We get to the details in Chapter 8, "Adventures in Functions." Meanwhile, the C mavens among you can relax—ordinarily, argument passing in C++ works just as it does in C. For `cin.get(ch)`, however, it doesn't.

**Which `cin.get()`?**

Chapter 4 uses this code:

```cpp
char name[ArSize];
...
cout << "Enter your name:\n";
cin.get(name, ArSize).get();
```

The last line is equivalent to two consecutive function calls:

```cpp
cin.get(name, ArSize);
cin.get();
```

One version of `cin.get()` takes two arguments: the array name, which is the address of the string (technically, type `char*`), and `ArSize`, which is an integer of type `int`. (Recall that the name of an array is the address of its first element, so the name of a character array is type `char*`.) Then, the program uses `cin.get()` with no arguments. And, most recently, we've used `cin.get()` this way:

```cpp
char ch;
cin.get(ch);
```