Using `const` in this fashion means you can use `bird` to access the string but not to change it. Chapter 7 takes up the topic of `const` pointers in greater detail. Finally, the pointer `ps` remains uninitialize, so it doesn’t point to any string. (This, you recall, usually is a bad idea, and this example is no exception.)

Next, the program illustrates that you can use the array name `animal` and the pointer `bird` equivalently with `cout`. Both, after all, are the addresses of strings, and `cout` displays the two strings ("bear" and "wren") stored at those addresses. If you activate the code that makes the error of attempting to display `ps`, you might get a blank line, you might get garbage displayed, and you might get a program crash. Creating an uninitialized pointer is a bit like distributing a blank signed check; you lack control over how it will be used.

For input, the situation is a bit different. It's safe to use the array `animal` for input as long as the input is short enough to fit into the array. It would not be proper to use `bird` for input, however:

- Some compilers treat string literals as read-only constants, leading to a runtime error if you try to write new data over them. That string literals be constant is the mandated behavior in C++, but not all compilers have made that change from older behavior yet.

- Some compilers use just one copy of a string literal to represent all occurrences of that literal in a program.

Let's amplify the second point. C++ doesn't guarantee that string literals are stored uniquely. That is, if you use a string literal "wren" several times in the program, the compiler might store several copies of the string or just one copy. If it does the latter, then setting `bird` to point to one "wren" makes it point to the only copy of that string. Reading a value into one string could affect what you thought was an independent string elsewhere. In any case, because the `bird` pointer is declared as `const`, the compiler prevents any attempt to change the contents of the location pointed to by `bird`.

Worse yet is trying to read information into the location to which `ps` points. Because `ps` is not initialized, you don't know where the information will wind up. It might even overwrite information already in memory. Fortunately, it's easy to avoid these problems—just use a sufficiently large `char` array to receive input. Don't use string constants to receive input or uninitialized pointers to receive input.

**Caution**