The difficulty with this kind of error is that the code is syntactically correct, so the compiler won't tag it as an error. (However, years and years of C and C++ programmers making this error eventually has led many compilers to issue a warning asking if that's what you really meant to do.)

Caution

Don't use = to compare for equality; use ==.

Like C, C++ grants you more freedom than most programming languages. This comes at the cost of requiring greater responsibility on your part. Nothing but your own good planning prevents a program from going beyond the bounds of a standard C++ array. However, with C++ classes, you can design a protected array type that prevents this sort of nonsense. Chapter 13, "Class Inheritance," provides an example. In the meantime, you should build the protection into your programs when you need it. For example, the loop should have included a test that kept it from going past the last member. That's true even for the "good" loop. If all the scores had been 20s, it, too, would have exceeded the array bounds. In short, the loop needed to test the values of the array and the array index. Chapter 6, "Branching Statements and Logical Operators," shows you how to use logical operators to combine two such tests into a single condition.

Comparing Strings

Suppose you want to see if a string in a character array is the word mate. If word is the array name, the following test might not do what you think:

word == "mate"

Remember that the name of an array is a synonym for its address. Similarly, a quoted string constant is a synonym for its address. Thus, the preceding relational expression doesn't test to see whether the strings are the same—it checks to see whether they are stored at the same address. The answer to that is no, even if the two strings have the same characters.

Because C++ handles strings as addresses, you get little satisfaction if you try to use the