errors that inadvertently alter data.

- Using `const` allows a function to process both `const` and non-`const` actual arguments, while a function omitting `const` in the prototype only can accept non-`const` data.

- Using a `const` reference allows the function to generate and use a temporary variable appropriately.

You should declare formal reference arguments as `const` whenever it's appropriate to do so.

### Using References with a Structure

References work wonderfully with structures and classes, C++'s user-defined types. Indeed references were introduced primarily for use with these types, not for use with the basic built-in types.

The method for using a reference to a structure is the same as the method for using a reference to a basic variable; just use the `&` reference operator when declaring a structure parameter. The program in Listing 8.6 does exactly that. It also adds an interesting twist by having a function return a reference. This makes it possible to use a function invocation as an argument to a function. Well, that's true of any function with a return value. But it also makes it possible to assign a value to a function invocation, and that's possible only with a reference return type. We'll explain these points after showing the program's output. The program has a `use()` function that displays two members of a structure and increments a third member. Thus, the third member can keep track of how many times a particular structure has been handled by the `use()` function.

#### Listing 8.6 strtref.cpp

```cpp
// strtref.cpp -- using structure references
#include <iostream>
using namespace std;
```