This has the unfortunate effect of returning a reference to a temporary variable (newguy) that passes from existence as soon as the function terminates. (This chapter discusses the persistence of various kinds of variables later, in the section on storage classes.) Similarly, you should avoid returning pointers to such temporary variables.

**When to Use Reference Arguments**

There are two main reasons for using reference arguments:

- To allow you to alter a data object in the calling function
- To speed up a program by passing a reference instead of an entire data object

The second reason is most important for larger data objects, such as structures and class objects. These two reasons are the same reasons one might have for using a pointer argument. This makes sense, for reference arguments are really just a different interface for pointer-based code. So, when should you use a reference? Use a pointer? Pass by value? Here are some guidelines.

A function uses passed data without modifying it:

- If the data object is small, such as a built-in data type or a small structure, pass it by value.
- If the data object is an array, use a pointer because that's your only choice. Make the pointer a pointer to const.
- If the data object is a good-sized structure, use a const pointer or a const reference to increase program efficiency. You save the time and space needed to copy a structure or a class design. Make the pointer or reference const.
- If the data object is a class object, use a const reference. The semantics of class design often require using a reference, which is the main reason why C++ added this feature. Thus, the standard way to pass class object arguments is by reference.

A function modifies data in the calling function:

- If the data object is a built-in data type, use a pointer. If you spot code like `fixit(&x)`, where x is an int, it's pretty clear that this function intends to modify x.