x + 3.0 = 5.0; // nonsensical

What happens if you try a function call like refcube(x + 3.0)? In contemporary C++, that's an error, and some compilers will tell you so. Others give you a warning along the following lines:

Warning: Temporary used for parameter 'ra' in call to refcube(double &)

The reason for this milder response is that C++, in its early years, did allow you to pass expressions to a reference variable. In some cases, it still does. What happens is this:
Because x + 3.0 is not a type double variable, the program creates a temporary, nameless variable, initializing it to the value of the expression x + 3.0. Then, ra becomes a reference to that temporary variable. Let's take a closer look at temporary variables and see when they are and are not created.

**Temporary Variables, Reference Arguments, and const**

C++ can generate a temporary variable if the actual argument doesn't match a reference argument. Currently, C++ permits this only if the argument is a const reference, but this is a new restriction. Let's look at the cases in which C++ does generate temporary variables and see why the restriction to a const reference makes sense.

First, when is a temporary variable created? Provided that the reference parameter is a const, the compiler generates a temporary variable in two kinds of situations:

- The actual argument is the correct type, but isn't an Lvalue
- The actual argument is of the wrong type, but of a type that can be converted to the correct type

An argument that's an Lvalue is a data object that can be referenced. For example, a variable, an array element, a structure member, a reference, and a dereferenced pointer are Lvalues. Non-Lvalues include literal constants and expressions with multiple terms. For example, suppose we redefine refcube() so that it has a constant reference argument:

double refcube(const double &ra)
{
    return ra * ra * ra;