The message reminds you that C++ interprets the declaration `const double ar[]` to mean `const double *ar`. Thus, the declaration really says that `ar` points to a constant value. We'll discuss this in detail when we finish with the current example. Meanwhile, here is the code for the `show_array()` function:

```
void show_array(const double ar[], int n)
{
    using namespace std;
    for (int i = 0; i < n; i++)
    {
         cout << "Property #" << (i + 1) << ": $";
         cout << ar[i] << endl;
    }
}
```

Modifying the Array

The third operation for the array in this example is multiplying each element by the same revaluation factor. You need to pass three arguments to the function: the factor, the array, and the number of elements. No return value is needed, so the function can look like this:

```
void revalue(double r, double ar[], int n)
{
      for (int i = 0; i < n; i++)
            ar[i] *= r;
}
```

Because this function is supposed to alter the array values, you don't use `const` when you declare `ar`.

Putting the Pieces Together

Now that you've defined a data type in terms of how it's stored (an array) and how it's used (three functions), you can put together a program that uses the design. Because you've already built all the array-handling tools, you've greatly simplified programming `main()`. The program does check to see if the user responds to the prompt for a revaluation factor with a number. In this case, rather than stopping if the user fails to comply, the program uses a loop to ask the user to do the right thing. Most of the remaining programming work consists of having `main()` call the functions you've just developed. Listing 7.7 shows the result. It places a `using` directive in just those functions that use the `iostream` facilities.

Listing 7.7 **arrfun3.cpp**

```
// arrfun3.cpp -- array functions and const
#include <iostream>
const int Max = 5;
```

```
// function prototypes
int fill_array(double ar[], int limit);
void show_array(const double ar[], int n);  // don't change data
void revalue(double r, double ar[], int n);

int main()
{
    using namespace std;
    double properties[Max];

    int size = fill_array(properties, Max);
    show_array(properties, size);
    if (size > 0)
    {
        cout << "Enter revaluation factor: ";
        double factor;
        while (!(cin >> factor))    // bad input
        {
            cin.clear();
            while (cin.get() != '\n')
                continue;
           cout << "Bad input; Please enter a number: ";
        }
        revalue(factor, properties, size);
        show_array(properties, size);
    }
    cout << "Done.\n";
    cin.get();
    cin.get();
    return 0;
}

int fill_array(double ar[], int limit)
{
    using namespace std;
    double temp;
    int i;
    for (i = 0; i < limit; i++)
    {
        cout << "Enter value #" << (i + 1) << ": ";
        cin >> temp;
        if (!cin)    // bad input
        {
            cin.clear();
            while (cin.get() != '\n')
                continue;
           cout << "Bad input; input process terminated.\n";
```

```
            break;
        }
        else if (temp < 0)     // signal to terminate
            break;
        ar[i] = temp;
    }
    return i;
}

// the following function can use, but not alter,
// the array whose address is ar
void show_array(const double ar[], int n)
{
    using namespace std;
    for (int i = 0; i < n; i++)
    {
        cout << "Property #" << (i + 1) << ": $";
        cout << ar[i] << endl;
    }
}

// multiplies each element of ar[] by r
void revalue(double r, double ar[], int n)
{
    for (int i = 0; i < n; i++)
        ar[i] *= r;
}
```

Here are two sample runs of the program in Listing 7.7:

```
Enter value #1: 100000
Enter value #2: 80000
Enter value #3: 222000
Enter value #4: 240000
Enter value #5: 118000
Property #1: $100000
Property #2: $80000
Property #3: $222000
Property #4: $240000
Property #5: $118000
Enter revaluation factor: 0.8
Property #1: $80000
Property #2: $64000
Property #3: $177600
Property #4: $192000
Property #5: $94400
Done.
```