Property #1: $240000
Property #2: $100800
Property #3: $192000
Done.

Program Notes

We've already discussed the important programming details, so let's reflect on the process. We began by thinking about the data type and designed appropriate functions to handle the data. Then, we assembled these functions into a program. This sometimes is called bottom-up programming, because the design process moves from the component parts to the whole. This approach is well suited to OOP, which concentrates on data representation and manipulation first. Traditional procedural programming, on the other hand, leans toward top-down programming, in which you develop a modular grand design first and then turn your attention to the details. Both methods are useful, and both lead to modular programs.

Functions Using Array Ranges

As you've seen, C++ functions that process arrays need to be informed about the kind of data in the array, the location of the beginning of the array, and the number of elements in it. The traditional C/C++ approach to functions that process arrays is to pass a pointer to the start of the array as one argument and to pass the size of the array as a second argument. (The pointer tells the function both where to find the array and the kind of data in it.) That gives the function the information it needs to find all the data.

There is another approach to giving the function the information it needs, and that is to specify a range of elements. This can be done by passing two pointers: one identifying the start of the array and one identifying the end of the array. The C++ Standard Template Library (presented in Chapter 16, "The String Class and the Standard Template Library"), for example, generalizes the range approach. The STL approach uses the concept of "one past the end" to indicate the extent. That is, in the case of an array, the argument identifying the end of the array would be a pointer to the location just after the last element. For example, suppose we have this declaration:

double elbuod[20];