This accentuates the idea that the combination *ptr is a type int value. Many C++ programmers, on the other hand, use this form:

    int* ptr;

This emphasizes the idea that int* is a type, pointer-to-int. Where you put the spaces makes no difference to the compiler. Be aware, however, that the declaration

    int* p1, p2;

creates one pointer (p1) and one ordinary int (p2). You need an * for each pointer variable name.

**Remember**

In C++, the combination int * is a compound type, pointer-to-int.

You use the same syntax to declare pointers to other types:

    double * tax_ptr; // tax_ptr points to type double
    char * str;       // str points to type char

Because you declare tax_ptr as a pointer-to-double, the compiler knows that *tax_ptr is a type double value. That is, it knows that *tax_ptr represents a number stored in floating-point format that occupies (on most systems) eight bytes. A pointer variable is never simply a pointer. It always is a pointer to a specific type. tax_ptr is type pointer-to-double (or type double *) and str is type pointer-to-char (or char *). Although both are pointers, they are pointers of two different types. Like arrays, pointers are based upon other types.

Note that whereas tax_ptr and str point to data types of two different sizes, the two variables tax_ptr and str themselves typically are the same size. That is, the address of a char is the same size as the address of a double, much as 1016 might be the street address for a department store, whereas 1024 could be the street address of a small cottage. The size or value of an address doesn't really tell you anything about the size or kind of variable or building you find at that address. Usually, addresses require two or four