arrays. Although structure variables resemble arrays in that both can hold several data items, structure variables behave like basic, single-valued variables when it comes to functions. That is, unlike an array, a structure ties its data into a single entity that will be treated as a unit. Recall that you can assign one structure to another. Similarly, you can pass structures by value, just as you do with ordinary variables. In that case, the function works with a copy of the original structure. Also, a function can return a structure. There's no funny business like the name of an array being the address of its first element. The name of a structure is simply the name of the structure, and if you want its address, you have to use the & address operator.

The most direct way to program by using structures is to treat them as you would treat the basic types; that is, pass them as arguments and use them, if necessary, as return values. However, there is one disadvantage to passing structures by value. If the structure is large, the space and effort involved in making a copy of a structure can increase memory requirements and slow the system down. For those reasons (and because, at first, C didn't allow the passing of structures by value), many C programmers prefer passing the address of a structure and then using a pointer to access the structure contents. C++ provides a third alternative, called passing by reference, that we discuss in Chapter 8. We examine the other two choices now, beginning with passing and returning entire structures.

**Passing and Returning Structures**

Passing structures by value makes the most sense when the structure is relatively compact, so let's develop a couple of examples along those lines. The first example deals with travel time (not to be confused with time travel). Some maps will tell you that it is three hours, 50 minutes, from Thunder Falls to Bingo City and one hour, 25 minutes, from Bingo City to Grotesquo. You can use a structure to represent such times, using one member for the hour value and a second member for the minute value. Adding two times is a little tricky because you might have to transfer some of the minutes to the hours part. For example, the two preceding times sum to four hours, 75 minutes, which should be converted to five hours, 15 minutes. Let's develop a structure to represent a time value and then a function that takes two such structures as arguments and returns a structure that represents their sum.

Defining the structure is simple:

```c
struct travel_time
{
```