makes a program easier to read and follow.

Another thing you can do with structures is create a structure with no type name. You do this by omitting a tag name while simultaneously defining a structure form and a variable:

```c
struct    // no tag
{
    int x;  // 2 members
    int y;
}  position;  // a structure variable
```

This creates one structure variable called `position`. You can access its members with the membership operator, as in `position.x`, but there is no general name for the type. You subsequently can't create other variables of the same type. This book won't be using this limited form of structure.

Aside from the fact a C++ program can use the structure tag as a type name, C structures have all the features we've discussed so far for C++ structures. But C++ structures go further. Unlike C structures, for example, C++ structures can have member functions in addition to member variables. But these more advanced features most typically are used with classes rather than structures, so we'll discuss them when we cover classes.

**Arrays of Structures**

The `inflatable` structure contains an array (the `name` array). It's also possible to create arrays whose elements are structures. The technique is exactly the same as for creating arrays of the fundamental types. For example, to create an array of 100 `inflatable` structures, do the following:

```c
inflatable gifts[100];  // array of 100 inflatable structures
```

This makes `gifts` an array of `inflatable`s. Hence each element of the array, such as `gifts[0]` or `gifts[99]`, is an `inflatable` object and can be used with the membership operator:

```c
cin >> gifts[0].volume;       // use volume member of first struct
cout << gifts[99].price << endl;  // display price member of last struct
```