Structure Definitions

Here is the C++ definition for a `Date` user-defined structure type:

```
#include <string>
struct Date
{
    std::string month;
    int day;
    int year;
};
```

Unlike a variable definition, a structure definition does not create an object in memory. Instead, it **defines or specifies a new type** which contains **three data members**. (You should not call them fields—ala Java— since the term *field* has a slightly different meaning in C++).

The structure name (`Date`) is formally known as the **structure tag**. As mentioned earlier, structure members **do not** all **need** to be of the same type. In `Date`, `month` is of type `string`, while `day` and `year` are of type `int`.

Nested Structures

A structure member may be another type of structure. This is called a **nested structure**. For instance, a person has a name and a birthday. We have a `Date` type, so we can use it as part of a `Person` definition.

```
struct Person
{
    std::string name;
    Date birthday;
};
```

Structure definitions are normally found **inside header files**. Thus, all library members (such as the `std::string month`), must be **fully qualified**. It is **illegal** to include the same type definition multiple times, even if the definitions are exactly the same. Protect against this by using **header guards** (which are not shown here).

> *Don't forget the semicolon appearing after the final brace. If you leave it off, you're likely to see a misleading error message pointing to a different area of your code.*