Aggregate & Unsupported Operations

Structure types in the C programming language cannot automatically perform all of the common operations that the built-in types can, so we say that such derived types are **second-class types**. Operations that **work with the structure as a whole** are called **aggregate** operations.

Four **built-in aggregate operations** work in both C and in C++: assignment, initialization, passing parameters and returning structures. Given a `Date` variable:

- You can **assign** it to another variable, just as if it were an `int` or `double`.
- You can use it to **initialize** another variable.
- You can **pass it to a function** as an argument.
- You can **return it** from a function.

All of these are closely related to assignment. Here are some things **you can and cannot do** with structure variables:

```
Date d1{"February", 2, 1950}, d2;
d2 = d1;                // assignment OK
Date d3{d2};            // initialize OK
if (d1 == d2)           // NO aggregate comparison
    cout << d1 << endl; // NO aggregate I/O
d1++;                   // NO built-in arithmetic
```

As you can see:

- You **cannot compare two structures** using either equality or the relational operators. You must compare the individual data members instead.
- You **cannot automatically display** a structure variable using `cout`; you must access and print the individual data members.
- There is no **built-in arithmetic**.

It is, however, easy to turn each of these operations into an aggregate operation by simply **writing some functions**. We'll look at those shortly.