By-Value or By-Reference

Let's look at that last example again:

```
if (equals(d1, d2)) cout << "equal" << endl;
```

Here, the two arguments, `d1` and `d2`, are **passed by value**, which means that the parameter variables `lhs` and `rhs` are initialized by **making a copy** of the entire `Date` structure when calling the function.

In this particular case, the **cost** (time and memory) of making that copy is not very high; but, if the structure had more data members, calling this function **could be very expensive**. For structure, class and library types, we can **avoid that cost** by:

- Using `const` **reference** if the function **should not** modify the argument.
- Use **non-**const **reference** if the **intent is to modify** the actual argument.

Given these guidelines, a **more correct** version of `equals()` would look like this:

```
bool equals(const Date& lhs, const Date& rhs)
{
    return lhs.month == rhs.month &&
        lhs.day == rhs.day && lhs.year == rhs.year;
}
```

In general, never pass a class or structure type by value to a function.

> *This is a fundamental difference in the way that Java and C++ object types work. In Java and C#, objects have **reference semantics**—the object variables do not contain the actual object members. In C++, objects have **value semantics**; the actual data members are stored inside the object variables.*