## Copy constructors

Constructors give us a way of initializing an object with one (or more) values. But consider how we would define a constructor which initializes an object with the value(s) of another object of the same type. For example:

```
Date d1 (1998,4,1); // INIT TO 1 APRIL
Date d2 (d1);       // INIT TO SAME DATE AS d1
Date d3 = d1;       // INIT TO SAME DATE AS d1
```

Such constructors are called “copy constructors”

*Note, however, that the following is not an example of a copy constructor but rather that of the assignment operator.*

*Date d1 (1998,4,1);*
*Date d2;*
*d2 = d1;*

There's a problem if we write the constructor in the "obvious" way:

```
class Date
{
public:
        Date(int year,int month,int day)
        { myYear = year; myMonth = month, myDay = day; }

        Date(Date d)
        { myYear = d.myYear; myMonth = d.myMonth; myDay = d.myDay; }

private:
        int myYear;
        int myMonth;
        int myDay;
};
```

---

## The problem

- In order to initialize **d2** we have to call the **`Date(Date d)`** constructor.
- In order to do that we need to make a temporary object (the parameter **d**) of type **`Date`**, with the same value as **`d1`**.
- When we create the temporary **d**, its constructor gets called and is passed **`d1`**.
- But now we're initializing a **`Date`** object (**d**) with another **`Date`** object (**`d1`**). So the copy constructor(**`Date(Date d)`**) gets called!
- In order to do that we need to make another temporary parameter (call this one **`d'`**) of type **`Date`**, with the same value as **d**.
- When we create the temporary **`d'`**, its constructor gets called and is passed **d**.
- But now we're initializing a **`Date`** object (**`d'`**) with another **`Date`** object (**d**). So the copy constructor (**`Date(Date d)`**) gets called!

- In order to do that we need to make another temporary parameter (call this one **`d''`**) of type **`Date`**, with the same value as **`d'`**.
- When we create the temporary **`d''`**, its constructorgets called and is passed **`d'`**.
- But now we're initializing a **`Date`** object (**`d''`**) with another **`Date`** object (**`d'`**). So the copy constructor (**`Date(Date d)`**) gets called!
- Et cetera, et cetera, et cetera....

## The solution

We must avoid *copying* the single **Date** parameter when we invoke the copy constructor. We can do that by passing the parameter *as a reference*

```
class Date
{
public:
        Date(int year,int month,int day); // AS BEFORE

        Date(Date& d)
        { myYear = d.myYear; myMonth = d.myMonth; myDay = d.myDay; }

        // ETC...
};

Date d1 (1998,4,1); // INIT TO 1 APRIL
Date d2 (d1);       // INIT TO SAME DATE AS d1
```

Now when we initialize **`d2`**, we actually pass a reference to the original **`d1`** into **`Date(Date& d)`**, instead of a copy. Since no copy is required, there are no recursive calls to the copy constructor.

Finally, it's almost always better to define the copy constructor like this:

```
class Date
{
public:
        Date(const Date& d)
        { myYear = d.myYear; myMonth = d.myMonth; myDay = d.myDay; }

        // ETC...
};
```

**Why?** Because we do not want to inadvertently change the values of the Date object being passed by reference.