Assignment vs. Initialization

Before we talk about constructors, look at these two statements:

```
string a = "Bob", b;       // initialization
b = "Bill";                // assignment
```

1. Two `string` objects are created and initialized on line one; `a` is initialized using the C-String `"Bob"`, and `b` is initialized to the empty `string` by running the default constructor.
2. The `string` object `b` is destroyed (its destructor is run), a new `string` object is initialized with `"Bill"`, and that new `string` object replaces the `string` object originally held by `b`.

The variable `b` is first initialized, then destroyed, then assigned. **This is inefficient**.

Assignment in a Constructor

The body of the constructor is executed **after** the data members have been initialized. You may use **assignment** to place a new value into these data members. For primitive types, the cost of doing this is negligible, but for object types, such assignments mean that **data members are constructed twice**—once at initialization and once at assignement. Here's an example. (The implementation is inline to shorten the code.)

```
class Person
{
public:
  Person(const string& name) { m_name = name; }
private:
    string m_name;
};
```

When you write `Person p("Fred")`, the `m_name` data member first calls the default constructor to create an empty `string` object. Then, in the body of the constructor, the default-constructed `string` is destroyed when assigning `name` to `m_name`. **This is inefficient**, and you want to avoid it.