Default Argument Rules

Here are the rules that determine the use of default arguments:

1. The default value appears **only in the function prototype**. If you repeat the default argument in the implementation file you will get a compiler error.
2. Parameters with defaults must **appear at the end of the parameter list** and cannot be followed by a parameter without a default. Here's a bad example:

   ```
   void badOrder(int a = 3, int b);    // how would you call this?
   ```

3. Default arguments are only used with value, **not reference** parameters. Here's another (bad) example:

   ```
   void badType(int& a = ????);    // what to set it to?
   ```

   Since a reference must refer to an `lvalue`, there is no way to specify which `lvalue` should be used when the function is called.

Prefer Overloading

Overloading is usually preferable to default arguments. Suppose for example, you wish to define a procedure named `setLocation()` which takes an `x` and a `y` coordinate as arguments.

You may write the prototype, **using default arguments**, like this:

```
void setLocation(double x = 0, double y = 0);
```

Now, the default location defaults to the origin `(0, 0)`. However, it **is possible** to call the function with **only one argument**, which is **confusing** to anyone reading the code. It is **better to just define a pair of overloaded** functions like this:

```
void setLocation(double x, double y);   // supply both
inline void setLocation() { setLocation(0, 0); }
```

The body of the second function, can just calls the first, passing `0`, `0` as the arguments. Since the function is very, very short, it can be marked `inline` which means it does not need to be defined inside the implementation file.