CS 223 – Wednesday, August 28th, 2013

Classes and Object Oriented Programming in C++

Suggested Reading:

Moo Chap 9

Moo, Operators Pg. 209

Weiss, Ch. 1, Section 4

Fundamentals of Good Software

- Scalable
- Extensible
- Maintainable
- All inputs give desired output (Basically, it's not buggy).

Flaws of Procedural Languages

- Global memory can leak like crazy
- Orphaned data
- Too many chefs
  - Too many functions trying to mess with the same data, leading to collisions
- Hard to debug

Protecting Data

- Encapsulate-able
- Control of access
- Clean up after yourself

Odd Ownership

Something that happens in the global memory model. You can end up doing things where you coerce types, breaking things in some really odd ways. This is a *highly* uncool way to roll, and since everything's all global, you could for example link in some code that accidentally coerces types, screwing up your data. Pretty much, it's bad news bears.

Classes! - The cure for what ails ye!

Classes encapsulate data, though when we say data we mean more than just values and variables. Classes can encapsulate functions, as well as other classes (turtles all the way down).

Additionally, classes control access to this data, allowing you to set access to that data as public, private, or protected (we didn't cover protected in this lecture). Private data can only be accessed by code that operates within the scope of that class. Public data can be accessed by things outside of the class, and though they're quit important, you've got to be careful with them.

> **Part of CS223 is the following oath:**
>
> > **I (your name here) do solemnly swear that I will give my life to serve and protect my data in times of conflict, foreign and domestic. I will never leave my data publicly accessible, and I will always build the appropriate methods for accessing and manipulating that data.**

So, having taken this oath, what's all this about *"building the appropriate methods for accessing and manipulating that data?"* Well, that relies on this next section!

Getters and Setters

- Getters and setters are the public classes that allow you to `get` and `set` data that is contained within a class. These would be things like `get()` and `set()`.

Overloading

- In C++ it's possible to "overload" a method by providing multiple methods with the same names but different arguments (and thus different ways of managing this different data.)

Name Mangling

This'll be touched on just briefly, but compilers will generally modify the compiled names of functions for their own reasons. Though there are no doubt good reasons for doing so (from the compiler standpoint), but it can cause some problems. For example, when linking a some code in, certain function/method names need to be maintained so that outside code can properly access them. There are ways (not listed here) to get the compiler to honor some or all written names when compiling the actual programs.

Constructor

The constructor is the method that is called when the object is instantiated. It properly rearranges all data, allocating memory or doing whatever is needed to make things pretty.

The *Big Three*

When you define a new class, the compiler automatically creates the "Big Three" methods. These methods are found on every class in C++, though they don't always act exactly the same way since they're often re-written by the class authors to better handle a particular classes data.

Indeed, one reason they are known as *"The Big Three"* is because conventionally, if you have to manually implement one, then you should probably implement all three.

The Big Three are as following:

- Destructor
- Copy Constructor
- The Assignment Operator (the " = " sign)

Destructor

The destructor deletes the object. It gets called when one of two things happens:

- The class goes out of scope.
- The destructor/delete method is called on the object

By default the destructor method is pretty “stupid” and won’t do things like properly free allocated memory that is just referenced via pointer in the class. Because of this, it’s a very good idea to write your own destructor method to properly handle the cleanup of your data.

Copy Constructor

This is method can be called to actually copy all data from one instance of a class to a **new** instance of a class. However, like the default destructor, the default copy constructor can be quite “dumb.” For example, the default copy constructor only performs a “shallow” copy, where it directly copies a pointer to a new pointer, so that two different variables in two different objects may point to the same data in memory. This is another method where it’s prudent to write your own version to better handle your particular needs.

Also, it should be noted that the copy constructor is called whenever you pass by value, as well as when you return by value.

The Assignment Operator

The assignment operator, a.k.a the equals sign (=) works very similarly to the copy constructor in that it copies data from one instance of a class to another instance of a class. However, unlike the copy constructor, assignment is used to copy data between **existing** instances of classes. Because of this, it must be able to handle more cases than the straight copy constructor.

It’s probably best to write your own version.