Object Oriented Programming in C++

Welcome to Part II of the class. Part I was on memory management – when and where are objects created? When and how are they destroyed? What are good strategies for managing heap memory? Part III will be on templates. Part II, however, is on object oriented programming in C++. Since you have already spent three semesters doing object oriented programming in Java, the big question is: how is OO programming different in C++ than in Java?

Let’s begin by talking about the programming assignment. PA4 is complex. If full of tiny little rules. None of the rules are complex by themselves, but there are lots of them. The art of programming is the art of managing complexity, as in PA4.

In my opinion, there are only two fundamental skills to programming. One is to take a complex problem and break it down into a sequence of simple steps. Most of you are pretty good at this, or you wouldn’t be in this class. But the second and equally important skill is the ability to collect these little steps into self-contained units that can be conceptually replaced by an abstraction. This is the skill of design, and without it programmers would be overwhelmed by complexity. We can all only hold seven plus or minus two things at a time in our short-term memory (more or less), so we can’t treat every line of code as its own item. We have to build up abstractions precisely so that we can forget the details. It’s the only way to handle the complexity.

So with that in mind, I have three hints to steer you in the right direction (i.e. toward abstraction):

1. Never have a method longer than 15 lines of (C++) code. That includes lines that only have a bracket, that includes the function header, etc. 15 lines isn’t much, and that’s the point. Keep your methods short.

2. Have lots of methods (sort of follows from #1 above). But keep them clean, meaning:
   a. Give then meaningful names that capture what they do. Names are the single most important form of documentation, because they are how functions are called. Also, if you can’t think of an appropriate name, then you don’t have an abstract idea of what the method does. That’s a red flag that you might want to change how you divided your function up.
   b. Most of them should be accessors. They can read object data, but most of them shouldn’t change it. Similarly, most of them should not side effect their arguments
   c. For example, my code has a method for every definition in PA4, a method for every Porter stemming step, and still more methods after that…

3. Break your program into object classes. You have multiple data types supporting different operations (files, strings, documents, frequencies…). Figure out how to encapsulate this data and the associated operations into classes. For example, I have 3 classes (Document, Lexeme, and Stemmer). Arguably I should have more. Your classes may be different. But if you only have one (or god forbid, none), you are probably going to struggle.

OK, let’s get back to object oriented programming in C++. What is the difference between a C++ object and a Java object? Well, they are pretty much the same, but let’s review. Why do we use objects? What is the primary feature of an object? **An object encapsulates code and data.**
This is fundamentally why we use objects. If I want to manipulate complex numbers, I don’t want complex number code spread all over my application. I want to group all the complex number code and associate it with the complex number values.

Note that data in this context is synonymous with state.

What’s the biggest difference between Java objects and C++ objects? In Java, everything HAS to be part of an object. In C++, objects are optional.

Most of the time, objects in C++ are a good idea. In most C++ applications, almost all the code is in the form of methods.

When do you not want to use objects? This is an interesting question. It’s partly a question about decomposition. In the programming assignments, I have a lexeme class that contains strings. But I do not have a character class or a syllable class. Why not?

I could have a character class. It could store the char, and have methods to tell me whether it is punctuation (our definition), alphabetic or numeric. One criterion for an object is that it supports unique operations.

Would it make sense outside of the context of a lexeme? Probably not.

How about complexity? Is that a reason to make something an object or not? (Not as much as you might think.)

Any other reasons?

Note that this discussion applies equally well to C++ or Java. Both are object oriented languages, and objects have to be defined.

But C++ also gives you the option of functions outside of methods. When is this a good idea?

What if I am not manipulating any state? What if I am computing a pure function (i.e., only manipulating the arguments)? Why do I need an object?

In fact, if the function doesn’t manipulate state, why do I need to create an instance of the state? Why would I pass a hidden argument? Isn’t that a burden?

So one instance where you might consider a function is when there are no side-effects.

No, this isn’t the same as an accessor. An accessor uses state, it just doesn’t change it. A pure function only accesses its arguments.

Of course, if all the arguments are of the same type (or there is a single argument), then it might as well be a method of that type. But if there are arguments of different types, none of which are changed, and no state is needed, then functions may be appropriate.

By the same argument, however, you will mostly use methods. C++ is not that different from Java.

Another difference between objects in C++ and Java lies in the notion of an inheritance hierarchy.
Let’s review again. What is inheritance? If class B extends class A, what does that mean?

Inheritance is another word for union. If B extends A, then B contains everything in A, plus whatever is added to B. It’s that simple.

In fact, it’s even implemented that way. When an instance of B is allocated, the first element in B is an instance of A. This is why a pointer to B is also a pointer to A.

This insight (inheritance == union) is the basis for decisions about inclusion versus inheritance.

For example, in the programming assignments you might use `std::string` to store the sequence of characters in a lexeme. Should you create an object that inherits `vector`, or includes it?

Include it. Because otherwise it would have to support append (and what would the type of the appended string be?), among other operations.

I get inherit/use questions all the time. It seems to be a confusion inherited from previous classes. But the criterion is pretty clear. Do you want everything in the parent class, or do you just want to use a subset of abilities from the parent class? Can you substitute a child for the parent in the parent’s operations? In other words, is polymorphism the goal? If not, use the class. If yes, inherit it.

Inheritance is union. Inheritance is also mandatory in Java. Every class inherits from something. All classes form a single hierarchy. This makes classes heavy; all classes inherit stuff.

Inheritance in C++ is optional. In assignment #4, your data class probably didn’t inherit anything.

This is simpler; it means you know exactly everything in your image class. It is also lighter weight. C++ class instances are smaller and easier to initialize.

The difference has subtle impacts. In Java, the class hierarchy is a single, large tree. In C++, it is a disjoint forest of very small trees.

What does this mean? Well, in Java you can have a vector of objects, because all instances are objects. In C++ you can have a vector of int, or a vector of images, or... but you can’t have a vector of arbitrary objects.

Unless you want to. There is nothing to stop you from declaring an ‘object’ class, and then making all of your objects inherit it. Then you can have a vector of objects. (Well, pointers to objects – we will get to slicing soon.)

But the object classes in the STL library won’t be part of this hierarchy, or classes you get from colleagues…. In short, you would have to rebuild the Java hierarchy in C++.