CS 350

# Week 1

## Lecture 1

### Constructors/Destructors

* Scoping
* When an automatic variable is declared (implicitly allocating, not saying “new”), when going out of scope, the destructor is called
* When explicitly allocating a variable (using the “new” keyword, used with pointers), when going out of scope, the destructor is not called – you have to explicitly use “delete” to call the destructor
* Inheritance
* Have to have the car shell before the internals can go in: Base constructor 🡪 Derived constructor
* Have to remove the internals of the car before removing the car shell: Derived destructor 🡪 Base destructor

### Static and Dynamic Scoping

* By default, C++ is statically scoped
* When variables are declared in a static scope, they are set – even if an object of type A is declared as a type B, the object will use type A methods
* A static variable means that there is one instance of the variable for all instances of all objects of that class
* The value will be the same for all instances of that object
* If there is a virtual method, it is dynamically scoped
* When variables are declared in a dynamic/virtual scope, they can change – if an object of type A is declared as a type B, the object will use type B methods

### Arrays

* If an array of object A is declared of size 3, then 3 objects of type A are created
* If an automatic/implicit variable is declared, the destructor is called 3 times
* If an explicit variable is declared, the destructor is not called

### Increment

* Pre-increment (++a)
* Takes the value and increments it before you use it
* Post-increment (a++)
* Takes the value and increments it after you use it
* Example
* int a=5;
* Question1(++a, ++a)
* It can be either 7 6 or 6 7 depending on if your processor compiles left-to-right or right-to-left
* C++ does not define as part of its specification the order in which you should compute the parameters of a function
* Hard Example
* for (int i = 0; i < 3; ++i)

num += num++;

* The answer is 0
* Each loop has the previous value of num, 0 in this case, copied back to num, so the num++ is ignored

## Lecture 2

### Child Class and Constructors

* If you have an object that extends another class, when an object is created of that class, it calls the base constructor and that class’s constructor

### Pass by Reference and Pass by Value

* Java is Pass by Value Reference
* Passes a copy of the reference of the object
* If you pass a string to a function and change the string, when you are outside of the function, the string is still its original value

### Interface and Inheritance

* If a class implements an interface but also inherits from a base class, the object of that class will use its own method, rather than the method of the base class
* Java does not support multiple inheritance, but it does support multiple interfaces

### Static vs. Dynamic in Java

* Java uses dynamic scoping, but if a method uses the “static” keyword, you are saying to scope it statically
* You can reference a static method in a different method without creating an object of the class as long as both are static

### null in Java

* When null is used in Java, the lowest on the hierarchy is the one executed
* For example, if you pass null to a class that has a method that executes with a string and a method that executes with an object, the string method would be called
* String extends Object in Java

### Try/Catch/Finally

* Finally is always executed at the end of a try/catch, and places whatever is in the finally on top of the stack
* Therefore, an answer can change based on what is in the finally

### Private and Public

* If you have a private variable, you need an accessor method to access it
* If you try to access the variable directly, you will get an error

### Comparing Strings with == in Java

* When declaring Strings with the same value, Java will point to the previous object with that value instead of making a new object of the same value

### final in Java

* If you attempt to change a final variable in Java, errors will be thrown

### this Keyword in java

* Refers to the current class, not the method