Name: Soham Thaker SID: 011-748-159 Email: [sdthaker1@myseneca.ca](mailto:sdthaker1@myseneca.ca)

I learnt how does the inheritance works in OOP languages and how classes are inherited from a base class to derived class. When a class in a hierarchy is derived from a base class, it inherits all the public members of the base class if we add access modifier as ‘public’ while defining the derived class, for example, ”class DerivedClass : public BaseClass”. ‘public’ access modifiers are the most common in C++. Apart from that, there are two more access modifiers named as private which bars all the access and protected which limits access to derived classes only. If a member of a Base class is labelled as protected, then only the derived class can have access to it, barring the client code from accessing that member.

Apart from that, construction of classes in a hierarchy always happens in a way where the base class portion of the object is created first where memory is allocated to all the instance variables in the order of their declaration followed by the execution of the base class constructor and then the same process is followed for the construction of derived class as well. Whereas destruction happens in reverse order, where derived class’ destructor is called first followed by deallocation of all the derived class’ memory followed later by calling the base class’ destructor and deallocating all the base class’ memory.

Also, C++ provides us a way to pass the arguments received in a derived class’ constructor to the base class with the help of initializer list so that base part of derived instance gets initialised. This is the only place in an inherited class structure where we can invoke the base class’ constructor which would possibly accept some argument to initialize the base class part of derived instance with the oncoming arguments in derived class’ constructor. The syntax for a constructor definition for a derived class would be, “DerivedClass(paramater) : BaseClass(parameter)” where the initializer list will pass on the parameters to the base class constructor to initialize the base class data members.

Moreover, I learnt what shadowing a function means. Let us say we have 2 display functions one in base class and one in derived. If in main, an object of derived class is created, and we call the display on the derived class’ object, now we have shadowed the base class’ display function as this call to display will invoke derived class’ display function. In order to call the base class’ display function we must use scope resolution operator within the display of derived class(explained in next paragraph) or else write ‘derivedObject.BaseClass::display();’ in main which override the call to base class’ display().

When we say derived class can access all the public members of a base class, it means that we must override the call to any public member from base class by using a scope resolution operator because a function in a hierarchy can have same identifier and in order to call the right function we must use resolve the scope, where we inform the compiler that during compile time which function is supposed to be called when the program runs otherwise the function call would be shadowed with the current instance. We usually do this by calling “BaseClass::display()” from derived class’ member function definition(here display() is a function defined in base class).

Furthermore, I also learnt how to upcast a derived class’ object to a base class and call the custom I/O operators on that casted version of the object. A c style upcasting from a derived class to a base class is allowed when we do “(BaseClass)derivedObject”. This will convert derivedObject instance into a base class instance and thus we can access the all the public methods of base class.