////////////////////////////////

///// Constructor Tests /////

////////////////////////////////

Testing Derived Default ctor

Vehicle: Default-ctor

Car: Default-ctor

* The program is calling for the Car class, which is the derived class, and in which the default constructor is called. Because there was no parameter in the calling of the vehicle class, “Car c1”, the default constructor was used. Upon successfully calling the constructor both “Vehicle: Default-ctor” and “Car: Default ctor” were displayed from the constructor as a sign that it was correctly called and executed. And the reason for both the Vehicle and Car ctor being called is because the constructor was derived from the car class.

Testing Derived Parametrized ctor

Vehicle: Default-ctor

Car: Parametrized-ctor

* Similarly to the default constructor, the parameterized constructor was called here, from the program calling “ Car c\_rno(lla\_rno)”. The intent of the parameterized constructor was to create a new object based on the values that were passed in the parameters. The parameters that was passed were the LLA of Reno, and it later determined that my function was able to work correctly aside from printing “Car: Parametrized-ctor”

Testing Derived Copy ctor

Vehicle: Default-ctor

Car: Copy-ctor

* The copy constructor was called here with the intent to create a new vehicle car and initialize them with values from another car object. It was called so that object “c\_rno”, which is the LLA of Reno, can be copied over into “c\_cpy” which is then later stored into “c1”. It is clear that the function was able to correctly work because the “Vehicle: Default-ctor” and “Car: Copy-ctor” were displayed into terminal.

Testing Derived Assignment operator

Car : Assignment

* The assignment operator was tested here which was intended to assign “c1” with the data from “c\_cpy”. The function was successfully called and printed out “Car: Assignment” which is a sign that the function was called correctly and run.

/////////////////////////////////

///// Polymorphism Tests /////

/////////////////////////////////

Testing VIRTUAL Move Function for DERIVED Class Objects

Car: DRIVE to destination, with throttle @ 75

* Proj6.cpp is checking whether the derived class was able to override the base class move function. The move function’s purpose is to take in a new LLA location to move the Car object there. When called, it produces a debug output: "Car: DRIVE to destination, with throttle @ 75 " , then calls Drive with an argument value of 75 and finally updates m\_lla with the passed float data values.

Testing Insertion operator<< Overload for BASE Class Objects

Car @[39.54,119.82,4500]

* The program is testing the insertion operator, which in other words is calling for the information of “c\_rno” to be displayed. The correct LLA was displayed which further confirms the ability of my functions to correctly execute.

///////////////////////////////////////////////////

///// Polymorphic Base Class Pointer Tests /////

///////////////////////////////////////////////////

Testing VIRTUAL Move Function on Base Class Pointers

Car: DRIVE to destination, with throttle @ 75

Car: DRIVE to destination, with throttle @ 75

Car: DRIVE to destination, with throttle @ 75

* The move function was intended to move the LLA member to a new address that is provided in the method’s parameters. However, it is correctly being called upon as “Car: DRIVE to destination, with throttle @ 75” is displayed as instructed. Proj6.cpp calls for this to happen with “vehicles\_array[i]->Move( lla\_new )” which in turn assigns all of “vehicle\_array” values into the ones in “lla\_new”.

Testing Insertion operator<< Overload for Base Class Pointers

Car @[37.77,122.42,52]

Car @[37.77,122.42,52]

Car @[37.77,122.42,52]

* Here the results from “Testing VIRTUAL Move Function on Base Class Pointers” are displayed. As explained from above, all the values of “vehicle\_array” are assigned new values from “lla\_new” and here they are printed out into terminal, which my functions were able to correctly display.

////////////////////////////

///// Tests Done /////

////////////////////////////

Car: Dtor

Vehicle: Dtor

Car: Dtor

Vehicle: Dtor

Car: Dtor

Vehicle: Dtor

* All the destructors are posted here because destructors are ran at the end of the program. It follows the sequence of the constructors from the base first, then the derived class, then the destructors and finally the derived destructor.