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

///// Base Tests /////

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

Testing Base Default ctor

Vehicle #0: Default-ctor

* The program is calling for the Vehicle class, which is the base class, and in which the default constructor is called. Because there was no parameter in the calling of the vehicle class, “Vehicle v1”, the default constructor was used. Upon successfully calling the constructor, “Vehicle #0: Default-ctor” was displayed from the constructor as a sign that it was correctly called and executed.

Testing Base insertion operator

Vehicle #0@[1.4013e-45,0,0]

Base idgen: 1

* The program is testing the insertion operator, which in other words is calling for the information of v1. In this case the numbers given out were nonsensical because they were not initialized. Following, the baseidgen was incremented and displayed.

Testing Base Parametrized ctor

Vehicle #1: Parametrized-ctor

Vehicle #1@[1.4013e-45,0,7.08495e-25]

Base idgen: 2

* Similarly to the default constructor, the parameterized constructor was called here. The intent of the parameterized constructor was to create a new object based on the values that were passed in the parameters. However, my program was unable to input the parameters correctly and resulted in data that wasn’t initialized to be printed.

Testing Base Copy ctor

Vehicle #2: Copy-ctor

Vehicle #2@[1.4013e-45,0,7.08495e-25]

Base idgen: 3

* The copy constructor was called here with the intent to create a new vehicle object and initialize them with values from another vehicle object. The address of the vehicle object was passed in so the copy constructor can take it and copy over the data into a new m\_lla that was initialized. Then ends with “Vehicle #2: Copy-ctor” to indicate that it was successfully called and run.

Testing Base Assignment operator

Vehicle #0: Assignment

Vehicle #0@[1.4013e-45,0,7.08495e-25]

Base idgen: 3

* The assignment operator was tested here which was intended to assign “v1” with the data from “v99\_cpy”. The function was successfully called and printed out “Vehicle #0: Assignment” which is a sign that the function was run. However, I am unsure if my program was able to execute as intended because from the beginning my program was unable to correctly receive and insert the data.

Testing Base Move Function

Vehicle #0: CANNOT MOVE - I DON'T KNOW HOW

* The move function was intended to move the LLA member to a new address that is provided in the method’s parameters. It is assumed that the data was moved to the new address as my code was written to go through the m\_lla array and assign it into the new address. However, I am unable to check without writing new line of code because proj5.cpp does not print the new location of the LLA data. However, it is correctly being called upon as “Vehicle #0: CANNOT MOVE - I DON'T KNOW HOW” is displayed as instructed.

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

///// Derived Tests /////

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

Testing Derived Default ctor

Vehicle #3: Default-ctor

Car #3: Default-ctor

* The derived tests are from the car class which it inherits through the vehicle class public. As tested in the “Base tests” section, proj5.cpp calls for the a new Car class object to be created through the default constructor. In which the constructor sets m\_throttle to 0.

Testing Derived insertion operator

Car #3 Plates:ï¿½ï¿½, Throttle: 0 @ [0, 0, 7.08453e-25]

Derived idgen: 4

* Again similar to the “Base tests” the insertion operator here serves the same purpose, however it is testing to see if it can correctly insert and display the idgen for the the car class. The results are similar in which numbers that were not intended are displayed because of the lack of any values for which it was to be initialized with.

Testing Derived Parametrized ctor

Vehicle #4: Default-ctor

Car #4: Parametrized-ctor

Car #4 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500]

Derived idgen: 5

* This parameterized constructor is called and passes the plate, vin, and the LLA. My constructor was able to intake and insert the passed values. From then proj5.cpp calls for the “c999” to be printed into terminal which is “Car #4 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500]” which leads me to believe that my constructor was able to work as intended. Then the idgen is incremented and displayed.

Testing Derived Copy ctor

Vehicle #5: Default-ctor

Car #5: Copy-ctor

Car #5 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500]

Derived idgen: 6

* This copy constructor only differs from the vehicle copy constructor in which more values are being copied. The values being copied are the plates, lla, and the throttle. They are copied into the address that was provided by the parameters during the function call, which was “c999”. The constructor works as “Car #5: Copy-ctor Car #5 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500]” are both printed into the terminal.

Testing Derived Assignment operator

Car #3: Assignment

Car #3 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500]

Derived idgen: 6

* This assignment operator sets “c999\_cpy” into “c1” which was constructed using the default constructor. The information was able to correctly be copied over as shown from “Car #3 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500]” which is the information from “c1” as called upon in proj5.cpp by “cout << c1 << endl”. So it is safe to conclude that this information was correctly copied from “c999\_cpy” from the assignment operator

Testing Derived Move Function

Car #3: DRIVE to destination, with throttle @ 75

* The testing of this move function calls for several things to happen. One, it correctly is able to move the car “c1” to the new destination of LLA. Then the drive function is called within the move function in and sets the throttle to a new value of 75. Everything works correctly as intended by “Car #3: DRIVE to destination, with throttle @ 75”. The car does “move” to the new destination with the new throttle value of 75.

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

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

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

Car #5: Dtor

Vehicle #5: Dtor

Car #4: Dtor

Vehicle #4: Dtor

Car #3: Dtor

Vehicle #3: Dtor

Vehicle #2: Dtor

Vehicle #1: Dtor

Vehicle #0: 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.