Nick Alvarez

CS202.1101

Project 6

The purpose of this program is to demonstrate how polymorphism works. Similar to the last project, it was just the creation of header and source files that was necessary. Especially nice was the fact the previous project’s files could be reused with some slight modifications.

And again, designing the program was like filling in the blanks, although even simpler this time because there were less moving parts per se, and only some methods needed to be modified. Other things were simply removed as they were no longer necessary within the scope of this project.

Since the last project worked fine, and the files from that one were used for this one, it almost compiled on the first go. In fact, the only errors being thrown were due to the provided proj6.cpp file, as the function setLLA changed to SetLLA and move to Move. Changing the functions in the classes solved the issue.

I see no issues as far as design goes. Some functions were called and that is not reflected in the tests because there was no call to output anything, so one must take solace in the back of their mind that they worked.

**Constructor Tests**

Testing Derived Default ctor

Vehicle: Default-ctor

Car: Default-ctor

This creates a Car object, which calls the Vehicle constructor and Car constructor, as Car is derived from Vehicle, and requires m\_lla, even though it is uninitialized.

Testing Derived Parametrized ctor

Vehicle: Parameterized-ctor

Car: Parameterized-ctor

A Car object is created with provided values for m\_lla, which within Car’s parameterized constructor goes to Vehicle’s parameterized constructor and then initializes the rest of the Car object.

Testing Derived Copy ctor

Vehicle: Default-ctor

Car: Copy-ctor

The Car object is copied, which in order to create an object, the Vehicle default constructor is called and then the Car copy constructor has somewhere to put all the new member data.

Testing Derived Assignment operator

Car: Assignment

The first Car object is set equal in every way to another car object. No new object is created, so the Vehicle constructor is not called.

**Polymorphism Tests**

Testing VIRTUAL Move Function for DERIVED Class Objects

Car: DRIVE to destination, with throttle @ 75

The Move function is called, which calls Vehicle’s SetLLA method and updates the location of the car. Then it calls Car’s drive function, setting Car’s m\_throttle to 75.

Testing Insertion operator<< Overload for BASE Class Objects

Car: Throttle: 0 @ [39.54, 119.82, 4500]

This tests the insertion operator from the Vehicle class. Since Car is derived from Vehicle, it can use this function to call serialize and output Car information.

**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

A Vehicle type pointer calls Vehicle’s pure virtual function Move. As seen in the output, this virtual function with a Vehicle pointer ends up calling Car’s Move function, as polymorphism intends it to. This process is repeated 3 times for an array of Vehicle pointers.

Testing Insertion operator<< Overload for Base Class Pointers

Car: Throttle: 75 @ [37.77, 122.42, 52]

Car: Throttle: 75 @ [37.77, 122.42, 52]

Car: Throttle: 75 @ [37.77, 122.42, 52]

The insertion operator is tested using an array of Vehicle pointers that point to Car objects. Because a base class pointer can call derived class methods, it outputs the Car object information just like a Car object was next to << and not a Vehicle pointer.

**Tests Done**

Car: Dtor

Vehicle: Dtor

Car: Dtor

Vehicle: Dtor

Car: Dtor

Vehicle: Dtor

Vehicle #1: Dtor

Since the program is ending, it calls all the destructors. Each Vehicle and Car object is eliminated before as part of the exit code 0.