Kobe Maristela

Christos Papachristos

CS 202

10/10/2018

Project 5 Documentation

The main purpose of this program is to test my ability to create and use inheritance with C++ classes as well as to review my knowledge on how to manipulate classes with multiple constructors, static members/functions and operators. My design on this program is to reuse code written from vehicle inside car. Most of my problems that I encountered in coding the project was mostly initializing the constant variables using initializer list. Also, I ran into problems with the overloading operators in which it took me time on figuring out how to fix them. If given more time, I would fix the output in my derived tests in, which the vehicle constructor is being outputted in my derived tests because I used it to initialize my car.

Starting off in Base Tests, “Testing Base Default ctor” outputs invokes the default constructor, since no parameters are passed in, the result should be 0 since this starts off the count. Next, “Testing Base insertion operator” tests the insertion operator in which its outputs are the values from the default constructor. Next, “Testing Base Parametrized ctor” tests the parametrized constructor in which values are getting passed in creating a new object with the values passed in (Vehicle #99 @ [39.54, 119.82, 4500], Base idgen: 100). Next, “Testing Base Copy ctor” tests the copy constructor in which an object of the same type is copied (Vehicle #100 @ [39.54, 119.82, 4500], Base idgen: 101). Next, “Testing Base Assignment operator” tests the assignment operator in which the value of vehicle 99 is assigned to vehicle 1(Vehicle #0 @ [39.54, 119.82, 4500], Base idgen: 101). Next, “Testing Base Move Function” tests the move function, however, this function only works through the Car object since vehicle cannot move (Vehicle #0: CANNOT MOVE - I DON'T KNOW HOW).

Following the Base tests comes the Derived Tests, “Testing Derived Default ctor” outputs invoke the default constructor, since no parameters are passed in, the result should be 0 since this starts off the count. Next, “Testing Derived insertion operator” tests the insertion operator in which its outputs are the values from the default constructor(Car #101 Plates: , Throttle: 0 @ [0, 0, 0], Derived idgen: 102). Next, “Testing Derived Parametrized ctor” tests the parametrized constructor in which values are getting passed in creating a new object with the values passed in (Car #999 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500], Derived idgen: 1000). Next, “Testing Derived Copy ctor” tests the copy constructor in which an object of the same type is copied (Car #1000 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500], Derived idgen: 1001). Next, “Testing Derived Assignment operator” tests the assignment operator in which the value of vehicle 99 is assigned to vehicle 1(Car #101 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500], Derived idgen: 1001). Finally, “Testing Derived Move Function” invokes the drive function in which it moves car (Car #101: DRIVE to destination with throttle @ 75).

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

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

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

Testing Base Default ctor

Vehicle #0: Default-ctor 🡪 **invokes the default constructor; initializes values to 0**

Testing Base insertion operator

Vehicle #0 @ [0, 0, 0] 🡪 **outputs the member of the object.**

Base idgen: 1 🡪 **keeps count of the cars, so no two cars have same vin number**

Testing Base Parametrized ctor

Vehicle #99: Parametrized-ctor

Vehicle #99 @ [39.54, 119.82, 4500] 🡪 **initializes the member variables based on the values passed in**

Base idgen: 100 🡪 **keeps count of the cars, so no two cars have same vin number**

Testing Base Copy ctor

Vehicle #100: Copy-ctor

Vehicle #100 @ [39.54, 119.82, 4500] 🡪 **copies the object to another object of the same type**

Base idgen: 101 🡪 **keeps count of the cars, so no two cars have same vin number**

Testing Base Assignment operator

Vehicle #0: Assignment

Vehicle #0 @ [39.54, 119.82, 4500] 🡪 **assigns one object to another**

Base idgen: 101 🡪 **keeps count of the cars, so no two cars have same vin number**

Testing Base Move Function

Vehicle #0: CANNOT MOVE - I DON'T KNOW HOW 🡪 **Vehicle cannot drive; car can**

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

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

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

**FUNCTIONS SAME AS BASE TESTS**

Testing Derived Default ctor

Vehicle #101: Default-ctor

Car #101: Default-ctor

Testing Derived insertion operator

Car #101 Plates: , Throttle: 0 @ [0, 0, 0]

Derived idgen: 102

Testing Derived Parametrized ctor

Vehicle #999: Parametrized-ctor

Car #999: Parametrized-ctor

Car #999 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500]

Derived idgen: 1000

Testing Derived Copy ctor

Vehicle #1000: Default-ctor

Car #1000: Copy-ctor

Car #1000 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500]

Derived idgen: 1001

Testing Derived Assignment operator

Car #101: Assignment

Car #101 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500]

Derived idgen: 1001

Testing Derived Move Function

Car #101: DRIVE to destination with throttle @ 75 🡪 **invokes drive function to drive the car**

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

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

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

**Destroys the objects to free up space in memory**

Car #1000: DtorVehicle #1000: Dtor

Car #999: DtorVehicle #999: Dtor

Car #101: DtorVehicle #101: Dtor

Vehicle #100: Dtor

Vehicle #99: Dtor

Vehicle #0: Dtor