Nick Alvarez

CS202.1101

Project 5

The purpose of this program is to demonstrate how inherited classes work. Given a project file, all that was left to do was create class header and source files to accompany the main file and each other.

Designing this program was fairly simple to do. In a way, it was like filling in the blanks. The project documentation listed off what each class should have and it was up to me how to implement each element so that it would work with the provided file and the other class. Most of the functions were fairly basic but some required a special touch.

There were indeed problems with the program. The first one was accessing the getIdgen() function from within the main file. As it turned out, the function itself was not static so it needed a member to call it. Making it static solved the issue. Once that was solved the program compiled soon after. The issues then came from the output. In some cases, words were being outputted where they shouldn't be, or not matching the specification in the assignment. It turns out it was a combination of *this and the insertion operator. Drawing inspiration from Occam's Razor, I simplified the debug outputs within each constructor and method, solving the issue. My final problem came from the VINs. In the beginning, I didn't realize what they were for, which was creating unique objects. Once I found that out, the issue came with them not being incremented, so there were multiple Vehicle and Car #0s. My initial solution was to assign m_vin to whatever getIdgen() returned, and increment the counter before each return statement. In execution, the numbers would go up even without a constructor being called, which I realized was due to the function calls within the main file. An alternate method was devised: create a setVin() function

that would increment the s_idgen member and then call getIdgen() and return the integer. That way, it would only be incremented for new objects.

As far as design goes, the program is fairly well off. One change I would consider is implementing more failsafes so there is never a s_idgen and parameterized constructor conflict when initializing m_vin. Additionally, there is likely a better way to create VINs without a setVin() function, but since it worked well, I let it be.

Base Tests

```
Testing Base Default ctor
Vehicle #1: Default-ctor
```

This creates a Vehicle object and initializes it to default values. S_idgen is incremented.

```
Testing Base insertion operator
Vehicle #1 @ [0, 0, 0]
```

This tests outputting a Vehicle object with << which lists the VIN and location.

```
Testing Base Parametrized ctor
Vehicle #99: Parameterized-ctor
Vehicle #99 @ [39.54, 119.82, 4500]
```

This creates a Vehicle object with the specified VIN and location, then prints it out using the insertion operator.

```
Testing Base Copy ctor
Vehicle #2: Copy-ctor
Vehicle #2 @ [39.54, 119.82, 4500]
```

This copies information from one Vehicle object into a new Vehicle object, also incrementing the s_idgen counter for m_vin. Then it outputs the object.

```
Testing Base Assignment operator
Vehicle #1: Assignment
Vehicle #1 @ [39.54, 119.82, 4500]
```

This tests setting two objects equal to each other with the = operator. It then prints out the object info.

```
Testing Base Move Function
Vehicle #1: CANNOT MOVE - I DON'T KNOW HOW
```

The vehicle is moved to a new location by changing the m_lla member.

Derived Tests

```
Testing Derived Default ctor
Vehicle #3: Default-ctor
Car #3: Default-ctor
```

Since Car is derived from Vehicle, any Car object created also calls Vehicle's default constructor, providing a VIN and location to the new Car object. Then, the m_throttle and m_plates members are initialized to default values. Since it is Vehicle's default constructor that provides the VIN, setVin() is called again which increments s_idgen.

```
Testing Derived insertion operator
Car #3 Plates: , Throttle: 0 @ [0, 0, 0]
```

This prints out info about the object with <<. Notice that m_plates has no value so it comes up empty, as it was not initialized.

```
Testing Derived Parametrized ctor
Vehicle #999: Parameterized-ctor
Car #999: Parameterized-ctor
Car #999 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500]
```

A new Car object is being created based on passed in values. So, the parameterized constructor for Vehicle and Car is called which initializes Vehicle's members (m_vin and m_lla) to the specified values, then Car's members (m_plates and m_throttle). Then the object is outputted using Car's insertion operator.

```
Testing Derived Copy ctor
Vehicle #4: Default-ctor
Car #4: Copy-ctor
Car #4 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500]
```

Similar to the above output, the copy constructor copies Vehicle's members to the new object then copies Car's members. It outputs the object and increments s_idgen from the setVin() method.

```
Testing Derived Assignment operator
Car #3: Assignment
Car #3 Plates: Gandalf, Throttle: 0 @ [39.54, 119.82, 4500]
```

The assignment operator does not need to change the VIN, and all objects can be changed using the set methods. So, no Vehicle assignment operator is called because it is between two Car objects.

```
Testing Derived Move Function
Car #3: DRIVE to destination, with throttle @ 75
```

This function simply changes the location of the car by changing m_lla, and then sets the throttle to 75 with the drive method.

Tests Done

```
Car #4: Dtor
Vehicle #4: Dtor
Car #999: Dtor
Vehicle #999: Dtor
Car #3: Dtor
Vehicle #3: Dtor
Vehicle #2: Dtor
Vehicle #99: 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.