CST8219 – C++ Programming
Lab 5

Introduction:
The goal of this lab is to practice operator overloading. You will use your previous Vehicle class as a starting point and add traditional C++ operators.

Reference:
Week 5 Powerpoint materials on Brightspace. There are many reference websites at the end of the powerpoint slides.

Steps:

1. Use your project from lab 2. Open git bash and navigate to your project on your hard drive. In git bash, type **git branch** to see that you are on the **week4** branch. This means that your current project is the result of applying all the commits in time. Type the command: **git branch Week5.** Type **git checkout Week5** so that git will now add commits to the Week5 path instead. Make sure that you are on your week5 branch by typing: **git branch**. There should be a star beside Week5

2. Implement the assignment operator: operator=(Vehicle &)

   This should make the numWheels and numDoors equal to those of the object that are being passed in. It is similar to a copy constructor, only now you are not initializing memory. Don't forget that the return type of the function should be Vehicle& so that you can write something like:

   veh1 = veh2 = veh3 = veh4;

   This makes veh3 = veh4, but then returns veh4 so that it is used for the next assignment: veh2 = veh4;
   Also, veh4 shouldn't change, so make the Vehicle parameter to be a **const** object.

3. Implement the comparison operator: **operator==(const Vehicle &)**
   This function should return true if the numDoors and numWheels are both equal to the parameter's variables.

4. Implement the inequality operator: **operator!=(const Vehicle&).** This should just return the negation of the comparison operator of step #3.

5. Implement the prefix and postfix increment and decrement operators:
   - ***a.*** operator++()
   - ***b.*** operator++(int i)
   - ***c.*** operator--( )
   - ***d.*** operator--(int i)

For a Vehicle, you should implement increment so that all of the variables are incremented by 1. So Vehicle(2, 2) would be incremented to: Vehicle(3, 3)
Don't forget that postfix increment returns a copy of the original values before incrementing. So

Vehicle veh(4, 2); veh++; would make veh = (5, 3), but return Vehicle(4, 2)

6. Implement the output operator: ostream& operator<<(ostream&, const Vehicle &v) so that it outputs the variables to the output stream that is passed in. It should work like the printVehicle() that you have already implemented.
   You should declare the output operator as a friend of the Vehicle class and directly access the private variables of the Vehicle class when writing them to the stream.

7. Change your main function so that it looks like this:

```
int main(int argc, char **argv)
{
        Vehicle original;
        Vehicle copy(original); // copy constructor by reference

        cout << "Original is: " << original << " copy is: " << copy << endl;

        cout << "Increment original: " << original++ << endl;
        cout << "Increment copy:" << ++copy<< endl;

        cout << "Decrement original:" << --original << endl;
        cout << "Decrement copy:" << copy-- << endl;

        // should be true :
        cout << "Compare equality 1: " << (original == copy) << endl;

        //should be false:
        cout << "Compare equality 2: " << (--original == ++copy) << endl;

        //should be true:
        cout << "Compare inequality: " << (original != copy) << endl;

//This should make original = copy, and then return a Vehicle for output:
        cout << "Assignment operator: " << (original = copy) << endl;
        return 0;
}
```

8. Once you are finished, use git bash to commit your work to the Week5 branch:
   git commit -am "Finished Week 5"

Create a zip file containing everything in your week2 directory and submit it on Brightspace. Make sure it includes **week2.cpp**, **Vehicle.h**, **Vehicle.cpp**, **CMakeLists.txt**, the **.git** folder.

Marks: (total of 10)

The output operator<<( ) writes the Vehicle to the stream	+1
The output operator<<() is declared as a friend of Vehicle	+1
The prefix and postfix increment work properly:	+2
The prefix and postfix decrement work properly:	+2
The equality operator== works properly:	+1
The inequality operator!= works properly:	+1
The assignment operator=( ) works properly:	+1
The operator function signatures are in the header file, and the function implementations are in the .cpp file	+1