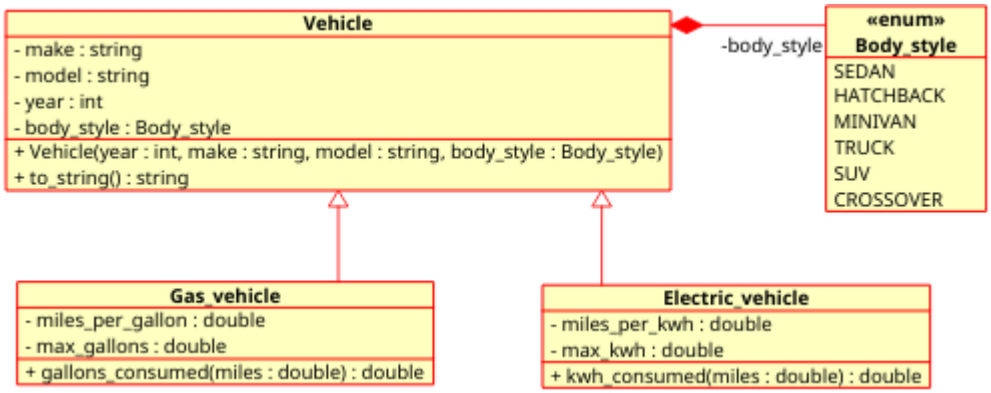
Up To Speed
Vehicles

CSE 1325 – Fall 2018 – Homework #4
Due Tuesday, September 18 at 8:00 am

Assignment Overview

Now that we're comfortable with simple classes, we'll mix in a little inheritance, flavored by an enum class, and (at the bonus level) some operator overloading and simple file I/O. The hardy souls among us should attempt the multiple inheritance at the extreme bonus level!

Full Credit: Using git (with *at least* 3 commits) and a Makefile:



- **Write the Vehicle base class, and Gas_Vehicle and Electric_Vehicle derived classes,** as shown to the right, based on an **enum class implementation of Body_style. Do NOT include "using namespace std;" anywhere in your files!** Use, e.g., "std::string" instead.
- **Write an application in file main.cpp** that iterates through a vector of EVs and a vector of gas vehicles, printing their year, make, model, and body style using Vehicle::to_string, and the fuel cost of driving 100 miles in each for a range of electricity and gasoline prices.

Deliver file CSE1325_04.zip to Blackboard. In **subdirectory "full_credit"**, include your full git repository, including your implementation of the Body_style enum class, the most recent .h and .cpp files for classes **Vehicle**, **Gas_vehicle**, and **Electric_vehicle, main.cpp, Makefile,** and any optional screenshots you believe will assist the graders.

Bonus: Using git (3+ additional commits) and a copy of the above code, **refactor main to read the vehicle data from a text file** by instancing ifstream (from lecture 7) using any file format you find convenient. For additional credit, overload the << operator for Vehicle and use it to print the vehicles.

In CSE1325_04.zip **subdirectory "bonus"**, include your git repository with the updated version of **main** and your **Vehicle class.**

Extreme Bonus: Using git and the Bonus code, create class Plugin_hybrid that inherits from *both* Gas_vehicle and Electric_vehicle. Add plug-in hybrids as a third vector in main, printing their costs for 100 miles of driving assuming electricity is consumed first. Update the UML class diagram as needed.

Notes

Use Git

You are REQUIRED to use git locally (not GitHub, unless you so choose[1]) while developing this code. You may elect to include a screenshot of your git commit log in case of problems, though you will no longer lose points if you don't. **Check the Notes in Homework #2 or Git in 5 Pages if you can't remember how.**

Use a Makefile

You are REQUIRED to use a simple Makefile that can build and execute your main program given the simple command "`make`" (or Tools → External Tools → Build in Gedit), and also support "make test" to build and run all of your regression tests. We covered how to create this step-by-step in Lecture 05, so these will no longer be routinely supplied to you.

Writing Body_style

Your first decision is how to represent the Body_style for a card. For this assignment, **use an enum CLASS** (NOT just an enum), to give you some experience with writing them before the exam (ahem). This is perhaps most conveniently placed in vehicle.h, in the global namespace.

As usual, you will need a way to convert the enun class variable back into a string. You may use a map (example in Lecture 06), which is good practice, or you may prefer an if / if else clause in Vehicle::to_string() or a dedicated Vehicle method instead.

Writing the Vehicle class

Write the vehicle class *interface* (no implementations) and store it in a file named **vehicle.h**. Don't forget your guards (#ifndef __VEHICLE_H, #define __VEHICLE_H, and #endif)! Once the interface is right, you may write the body.

- You'll need a constructor that accepts the year, make, model, and body style as parameters, and just stores them in the respective private variables.
- You'll need four getters, one for each of the private variables. *These are not shown on the UML diagram for conciseness.*
- You'll need a to_string() method for the main application. The standard to_string function should work with the year, and make and model are already strings but you'll need some way to convert a body style to a string.

Writing the Gas_vehicle class

Write the Gas_vehicle class interface as a publicly derived class from Vehicle, and store it in a file named **gas_vehicle.h**. You have two additional private variables, miles_per_gallon and max_gallons, and one additional method, gallons_consumed. Remember that all public and protected methods from Vehicle also are members of Gas_vehicle; this is automatic, and you need not mention them at all.

- You'll need one constructor, which accepts the same parameters as Vehicle::Vehicle and, in addition, miles_per_gallon and max_gallons. Storing the Vehicle values directly isn't possible, since they are private, but you can **delegate those to the Vehicle constructor**.

---

1 If you decide to use Github, please include a text file named GitHub.txt with a link to your public GitHub repository in CSE1325_02.zip.

- In Gas_vehicle's body, calculate gallons_consumed as miles / miles_per_gallon. **If gallons_consumed is greater than max_gallons, throw a runtime_error.**

Writing the Electric_vehicle class

Electric_vehicle is very similar to Gas_Vehicle, but with energy measured in kilowatt-hours of electricity in a battery rather than gallons of gasoline in a tank.

Writing the main application

The main application iterates through a vector of Gas_vehicle objects and then a vector of Electric_vehicle objects, printing out the vehicle information and the cost to travel 100 miles give gas prices per gallon of $2, $2.25, $2.50, $3, and $4, and electricity prices per kwh or $0.05, $0.08, $0.11, $0.13, and $0.15. (Current prices are around $2.25, and $0.08, btw.) See the example below. **Add any vehicles you like to your vectors**, but here are a few to get you started.

```
std::vector<Electric_vehicle> evs = {
    Electric_vehicle{2014, "Telsa", "Model S 85", Body_style::SEDAN, 3.12, 85},
    Electric_vehicle{2014, "Telsa", "Model 3 LR", Body_style::SEDAN, 4.13, 75},
    Electric_vehicle{2018, "GM", "Bolt", Body_style::HATCHBACK, 3.58, 60},
    Electric_vehicle{2018, "Nissan", "LEAF SL", Body_style::HATCHBACK, 3.88, 40},
};
std::vector<Gas_vehicle> ice = {
    Gas_vehicle{2017, "Toyota", "RAV4", Body_style::CROSSOVER, 26, 15.9},
    Gas_vehicle{2018, "Ford", "F-150", Body_style::TRUCK, 21, 36},
    Gas_vehicle{2018, "Nissan", "Rogue", Body_style::HATCHBACK, 29, 14.5},
    Gas_vehicle{2018, "Chrysler", "Pacifica", Body_style::MINIVAN, 22, 19},
};
```

```
ricegf@pluto:~/dev/cpp/201808/P4/full_credit$ ls
electric_vehicle.cpp  gas_vehicle.cpp  main.cpp  vehicle.cpp
electric_vehicle.h    gas_vehicle.h    Makefile  vehicle.h
ricegf@pluto:~/dev/cpp/201808/P4/full_credit$ make
g++ --std=c++17 -c main.cpp
g++ --std=c++17 -c vehicle.cpp
g++ --std=c++17 -c gas_vehicle.cpp
g++ --std=c++17 -c electric_vehicle.cpp
g++ --std=c++17 -o main main.o vehicle.o gas_vehicle.o electric_vehicle.o
ricegf@pluto:~/dev/cpp/201808/P4/full_credit$ ./main
2014 Telsa Model S 85 Sedan costs per 100 miles:
    at $0.05 per kwh -> $1.60256
    at $0.08 per kwh -> $2.5641
    at $0.11 per kwh -> $3.52564
    at $0.13 per kwh -> $4.16667
    at $0.15 per kwh -> $4.80769
2014 Telsa Model 3 LR Sedan costs per 100 miles:
    at $0.05 per kwh -> $1.21065
    at $0.08 per kwh -> $1.93705
```

Deliverables

In the full_credit directory, you will provide:

- At least 7 C++ source files named **vehicle.h and .cpp, gas_vehicle.h and .cpp, electric_vehicle.h and .cpp,** and **main.cpp,** and their associated git repository. How clever you would be to also have regression tests for your classes!
- A *required* make file named **Makefile** that builds your application by default ("make").
- *Optional* screenshots that illustrate your application or git repository history.

Bonus (may use some material from Lecture 07)

Continue to use git and a copy of the above code, **refactor your code to load vehicle data from a text file instead of embedding the data in your code.** Thus, you no longer need to rebuild to add another vehicle to the list.

Use ifstream, NOT fopen or bash redirection. You may hardcode the filename as a constant. The file format is up to you, but "ist >> year" is probably easier than "getline(ist, year_string); year = std::to_string(year_string);". Just saying.

Also, overload the Vehicle classes << operator, so that you may output the information for a Vehicle instances using:

```
std::cout << my_vehicle << std::endl;,
```

instead of

```
std::cout << my_vehicle.to_string() << std::endl;.
```

It's certainly fine for your << operator to call to_string, though, in which case it may not need to be a friend of Vehicle. (If you define << for Vehicle, will it also work for Gas_vehicle and Electric_vehicle? It should, yes, one of the conveniences of inheritance!)

Deliverables

In the bonus directory, you will provide the same files as the full_credit directory.

Extreme Bonus

Using the Bonus code with our usual array of tools, use multiple inheritance to create a Plugin_hybrid class that i**nherits from BOTH Gas_vehicle and Electric_vehicle**. (This is covered in lecture 07.) When calculating energy consumed, use all of the electricity first, then fall back to gasoline. Add a third vector to main, and add some plug-in hybrids such as the Chevy Volt and the Chrysler Pacifica Hybrid, and print those results along with the pure gasoline and pure electric vehicle results.

Update the UML class diagram and any tests you cleverly wrote as needed to provide a complete implementation.

Deliverables

In the extreme_bonus directory, you will provide the same files as in the bonus directory, as well as your **plugin_hybrid.h and .cpp** files and your updated **Makefile**.