Moving Ahead – Odometer Calculations

CSE 1325 – Spring 2017 – Homework #2
Due Tuesday, January 31 at 8:00 am

A common form of program is the “filter” - data flows into the program, and a transformed version of the data flows out of the program. This is analogous to, though implemented differently from, C++ streams (good 'ole cout and cin). A good example is the bash command line

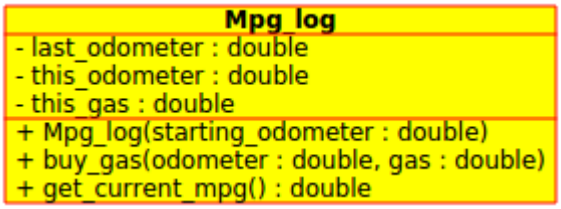
```
cat unsorted_names.txt | sort | uniq > unique_sorted_names.txt
```

The text file “unsorted_names.txt” is sent into the “sort “program, where it is (well) sorted by line; that output flows in to the “uniq” program, when any duplicate lines are removed, leaving only lines that are (well) unique; and then *that* output is written to the text file “unique_sorted_names.txt”. Filters can be chained together like this in ad hoc fashion to solve a lot of problems quickly – one of the advantages of the command line interface.[1]

http://stackoverflow.com/questions/9834086/what-is-a-simple-explanation-for-how-pipes-work-in-bash offers some good discussion of pipes and redirection of standard input and output, which is what we're using with C++ stream operators cin and cout, respectively.

In this assignment, you'll begin writing a filter in which gallons of gas purchased and the odometer reading at the time of purchase flows in, and out will flow miles per gallon statistics. This is useful for detecting when your engine needs a tune up, or to convince your family, financial advisor, or significant other that you should buy a more efficient (or better, an electric :-)) car.[2]

The Math



This program **shall** include a C++ class named Mpg_log and a main() function with no parameters. The main function shall instance Mpg_log and use the resulting object for maintaining odometer and gas data and for calculating mpg. The **required** class design is shown in UML.

Miles per gallon (MPG) is calculated by dividing the number of gallons pumped into the tank by the number of miles traveled. So if your odometer when you filled up the tank last week was 100,000 miles, and today you filled the tank with 10.0 gallons and the odometer reads 100,300 miles, your gas mileage was (100,300 – 100,000) / 10.0 = 30.0 MPG. (Note we don't care how much gas we bought last week – it doesn't matter, since that was gas we burned *prior* to our 100,000 odometer reading.)

If next week, you buy 5.0 gallons of gas and the odometer reads 100,400, your gas mileage for the week was (100,400 – 100,300) / 5.0 = 20.0 MPG.

Incidentally, your gas mileage *for both weeks* is (100,400 – 100,000) / (10.0 + 5.0) = 26.7 MPG. This is NOT the same as the average gas mileage, which would be (30.0 + 20.0) / 2 = 25 MPG – that's not mathematically valid or particularly useful. MPG is always “miles traveled / gallons used”.

---

1 Yes, graphical interfaces have many advantages, too. We'll get to them after the first exam...

2 Of course, mathematically you can rarely justify replacing a car on the basis of “saving money”. I didn't say this would provide a *good* argument; just an argument.

Given

You will be provided a spreadsheet called Mileage_Log.csv with a sample log of odometer readings and gasoline purchases, including additional data that you don't need. You may use the spreadsheet to copy and paste or to rekey test data into your program (the expected values are in column E to help you with testing), or you may create your own data.

Grading

- **Full Credit** – You will create an object-oriented program, implementing and instancing the Mpg_log class specified above, that will first show a prompt (think cout <<) and then read an initial odometer value from the user (think cin >>).

  Next, the program will enter a loop in which it prompts for and collects subsequent odometer readings and associated gallons of gas purchased, updates the Mpg_log object instanced above, and then prints the MPG value calculated by the object since the last time you bought gas.

  Entering an odometer reading of 0.0 exits the program.

  You will use git in a local repository to manage iterations of your program as you develop it, with a minimum of 3 commits over time.

  In the full_credit directory, you will provide a file called mileage.cpp, a screenshot named mileage.png of your interactive testing session similar to mine below, and a screenshot named mileage-git.png of your git log similar to mine below.

```
ricegf@pluto:~/dev/cpp/P2$ g++ -w mileage.cpp
ricegf@pluto:~/dev/cpp/P2$ ./a.out
Initial odometer: 36381.1
Odometer: 36545.2
Gallons: 4.5
This mpg: 36.4667
Odometer: 36712.9
Gallons: 8.2
This mpg: 20.4512
Odometer: 36845.7
Gallons: 4.4
This mpg: 30.1818
Odometer:
```

```
ricegf@pluto:~/dev/cpp/201701/P2$ git log --oneline
bd49cbd Fix narrowing conversion bug
7c16e1c migrated to iostreams and std
3341649 original version from 201608
ricegf@pluto:~/dev/cpp/201701/P2$
```

- **Bonus** – In addition to calculating the MPG since the last fill up, you will also calculate and print to the console the MPG *since the first odometer reading entered*. Thus, you are printing out two calculated values: This MPG and Total MPG. Column F of Mileage_Log.csv provides expected values for testing this version of the program.

  In the bonus directory, you will provide your source code in a file named mileage1.cpp, a screenshot of testing named mileage1.png, a screenshot of your git log named mileage1-git.png, and a screenshot or image of your class design in UML, including both private and public data and methods, named mileage1-uml.png.

```
ricegf@pluto:~/dev/cpp/201608/P2$ g++ -std=c++11 mileage1.cpp
ricegf@pluto:~/dev/cpp/201608/P2$ ./a.out
Initial odometer: 36381.1
Odometer: 36545.2
Gallons: 4.5
This mpg: 36.4889
Ave  mpg: 36.4889
Odometer: 36712.9
Gallons: 8.2
This mpg: 20.4512
Ave  mpg: 26.1339
Odometer: 36845.7
Gallons: 4.4
This mpg: 30.1818
Ave  mpg: 27.1754
Odometer:
```

  Note that you will need additional private data and public method(s) for the Bonus level. *Designing the class for the bonus levels is part of the requirements.*

- **Extreme Bonus** – Using the Mpg_log class from the Bonus level without modification for your solution, create a program that accepts a stream of data in CSV format from standard input (that's cin), and prints a stream of data in CSV format to standard output (that's cout) that includes the original unmodified data but with columns added for This MPG and Total MPG[3].

  Use test.csv to test this program, and visually compare the output to Mileage_Log.csv to verify correct operation.

  Give some thought to handling exceptions that may be caused by bad input, perhaps by ignoring the bad line of data and printing a warning to standard error, or by throwing an exception and aborting.

  In the xb directory, you will provide your source code in a file named mileage2.cpp, a screenshot of testing (using a command line such as "./a.out <test.csv") named mileage2.png, and a screenshot of your git log named mileage2-git.png. The UML didn't change.

3 Hey, look – it's a *filter*!