CSCI 3110 (Barbosa S18)
Project 2: Fill 'er up!

Due: **Fri Feb 9 by 11:59 PM** – may be turned in until Feb 16 by 11:59 PM with reduced points (per Project guidance found in the course syllabus).

Assignment ID: **proj2**
File(s) to be submitted: **gaspump.h, gaspump.cpp, proj2.cpp**

Objectives: 1) Define classes; 2) Split interface and implementation into files; 3) Instantiate objects; 4) Use objects in a client or driver function.

Project description:
In this assignment you will create a class representing a gasoline pump. The pump will maintain a running total of the amount of fuel dispensed and revenues collected. The driver program will simulate fuel demand for a number of vehicles.

Requirements:

1. Your program must be split into three files, the requirements for which are specified below:

   a) The class interface, or declaration, file

   - Must be named g***aspump.h***
   - Must contain #include guards for GASPUMP_H
   - Will have these private members
     i. A string containing the type of gas the gas pump holds
     ii. A double representing the amount of fuel on hand (in gallons)
     iii. A double representing the fuel pump's capacity (in gallons)
     iv. A double that holds the price per gallon
     v. A double to represent the amount of fuel dispensed per purchase
     vi. A double representing the total amount of fuel dispensed (all purchases)
     vii. A double that stores the total amount of money collected (all purchases)
     viii. A bool that indicates whether the next customer should be turned away as fuel is replenished
     ix. A function that replenishes the fuel pump's tank when it runs out of fuel
   - Must have these public members
     i. A single constructor with three parameters: a standard string (std::string) that holds the type of gasoline in the pump, a double representing the maximum capacity of the tank in gallons, and a double for the price per gallon.
     ii. An ***inline*** accessor function that returns a standard string for the pump's fuel type
     iii. An ***inline*** accessor function that returns the total amount of fuel dispensed (all purchases)
     iv. An ***inline*** accessor function that returns the total amount of revenue collected (all purchases)
     v. A ***void*** function named ***dispenseFuel***, that controls the fueling of vehicles

   b) A class implementation, or definition, file

   - Must be named ***gaspump.cpp***
   - Will have the implementation for the following functions
     i. Constructor – This function has three parameters that map directly to data members; it should also initialize the fuel quantity on hand to the maximum tank capacity (i.e., start at full capacity)

ii. Replenish – This function should simply reset the fuel level to the maximum capacity, and output a message to that effect, as shown in the sample output below

iii. *dispenseFuel* – This function is the heart of the class, and controls the fueling of a single vehicle. It accepts a double argument that indicates the number of gallons requested by the vehicle. It should begin by setting the number of decimal places for output to 2, with the following statements:

```
std::cout << std::fixed;
std::cout.precision(2);
```

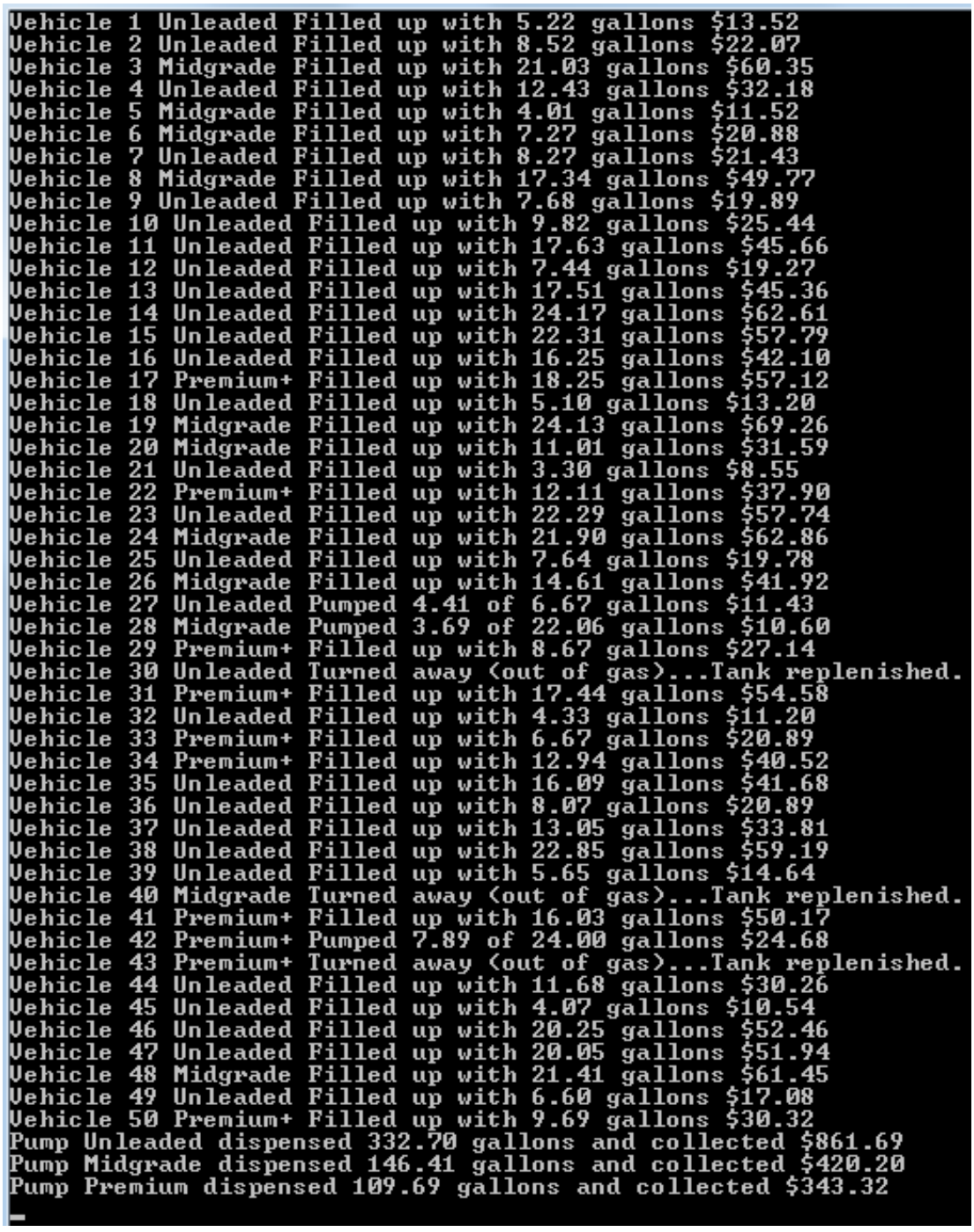
It will also

A. Display all of the message shown in the example output exactly as shown (verbatim)
B. Ensure that it only dispenses fuel if the tank is not being filled (in which case the customer is turned away)
C. Ensure that the pump dispenses no more than the amount of fuel on hand. If the vehicle's fuel demand exceeds the amount of fuel on hand, the vehicle gets all of the fuel on hand, and the next customer is turned away as the fuel is replenished. The amount desired and amount actually dispensed should be output as shown.
D. Keep track of fuel dispensed
E. Keep track of revenue collected (gallons * price per gallon)

c) A driver, or client, file

- Must be named ***proj2.cpp***
- Will instantiate 3 objects of the GasPump class: named *unleaded* (initialize to "Unleaded", 200 gallon capacity, and 2.59 per gallon), *midgrade* (initialize to "Midgrade", 125 gallon capacity, and 2.87 per gallon), and *premium* (initialize to "Premium**+**", 100 gallon capacity, and 3.13 per gallon). Note **+** after Premium.
- Declare a pointer to a GasPump object (will be used to select which pump a vehicle will use)
- Set the random number generator seed to 1000 : *srand(1000);*
- Create a for loop to simulate 50 vehicles; for each iteration
  - Determine which pump the vehicle will fuel from
    - Set the pointer to the *unleaded* pump (i.e. assume that the vehicle wants unleaded gas)
    - Draw a random number: *int rnd = rand();*
    - Take the remainder (i.e., mod 7) of rnd, and if it is 4 or 5, set the pointer to midgrade, if it is 6 set it to premium – otherwise leave it at unleaded. Do not reassign *rnd* (i.e., *rnd=rnd%7;*).
    - Note: The above establishes a probability distribution where, after infinitely many draws, unleaded is picked ~ 57% of the time (4 out of 7), midgrade around 29% of the time (2 out of 7), and premium about 14% of the time (1 out of 7). Be sure you understand how/why.
  - Determine the amount of fuel the vehicle requires
    - Use the same number drawn above to get the amount of fuel to be dispensed, through the following statement: *reqFuel = 3+ (((double)rnd) / RAND_MAX) * 22;*
    - The above establishes a fuel demand for the vehicle, between 3 and 25 gallons
  - Use the pointer to invoke the *dispenseFuel* function for the appropriate pump
  - Output "Vehicle X " as shown in the screenshot below (remaining output is from the pump object)
- Upon exiting the loop, output the total fuel dispensed and total revenue collected for each pump, as shown

2. The format of your output MUST MATCH EXACTLY the output in this specification. This means the verbiage, line spacing, character spacing (do not use tabs – only a single space between items). Based on the hardware/software you're using, you may or may not get the exact same random number sequence. However your output should be correct for the values drawn. An example screenshot is shown below:



3. Test your program - Use different capacity initialization and choose different random number seeds.

4. Code comments - Add the following comments to your code:

   - A section at the top of the source file(s) with the following identifying information:

     Your Name
     CSCI 3110-00X (your section #)
     Project #X
     Due: mm/dd/yy

   - Below your name add comments in each file that gives an overview of the program or class.
   - Place a one or two line comment above each function that summarizes the workings of the function.