CSCI205 CS3: Data Structures and Algorithms
Lab Assignment: Due Friday 9/11/2020

All assignments in this class need to be pushed to a private git hub repository. I should be added as a collaborator to this repository. Please send me the url and I will grade your work by cloning the repo. Each week I will pull and grade.

Task One: This assignment is a continuation of the Fraction class presented in Chapter 1 of Runestone

1. Implement the methods ***getNumerator*** and ***getDenominator***

2. Modify the constructor for the Fraction class so that GCD is used to reduce fractions immediately. Notice that this means the + function no longer needs to reduce. Make the necessary modifications.

3. Overload the following operators: **{-, *, /, >, >=, <, <=, !=}**

4. Modify the constructor for the Fraction class so that it checks to make sure that the numerator and denominator are both integers. If either is not an integer the constructor should raise an exception.

5. Modify the constructor to allow the user to pass a negative denominator so that all of the operators continue to work properly.

6. Overload **+=** keeping in mind that denominators can be negative.

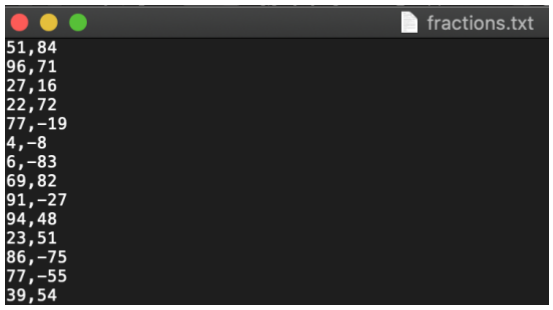
In a separate source code file (the application) define **main** to accomplish the following

https://www.learncpp.com/cpp-tutorial/programs-with-multiple-code-files/

1. Define a vector of Fraction objects

2. Open and read the contents of **fractions.txt**. This file is organized into rows containing numerators and denominators separated by commas.

   https://www.tutorialspoint.com/read-data-from-a-text-file-using-cplusplus



```
51,84
96,71
27,16
22,72
77,-19
4,-8
6,-83
69,82
91,-27
94,48
23,51
86,-75
77,-55
39,54
```

3. For each row in the file construct a Fraction object and add it to the vector. You will need to research splitting a string in C++:

   http://www.martinbroadhurst.com/how-to-split-a-string-in-c.html

4. Write the function **Fraction find_largest(vector<Fraction>)** this function will return the largest fraction. Use your overloaded comparison operators for this task.

5. Write the function **void print(vector<Fraction>)** this function will print the fractions to the screen 5 fractions per line.

6. Demonstrate that your overloaded operators function properly by choosing Fractions from the vector and demonstrating **{+, -, /, *}**

Task Two: Simple **War** Card Game. Practice with separate source code files. Have your class definition stored in a **.h** file and your implementation stored in a **.cpp** file.

You are responsible for the proper OOD in this assignment. Use your best judgement and adhere to proper OOD guidelines. I will be grading structure on this assignment.

1. Define a class to represent a playing card (you design). Use proper encapsulation and include appropriate getter and setter methods.
2. Define a class to represent a deck of cards (you design). Include methods to shuffle and deal a single card.
3. Define a class to represent a hand (you design). Include methods to add a card to the ***back of hand*** and a method to select the first card from the ***front of the hand***.
4. In a separate source code file define main to accomplish the following.

   a. Create and shuffle a deck
   b. Create two hands
   c. Alternating between the hands, deal all 52 cards. Each hand should have 26 cards.
   d. Play an automated game of **War.** Just let the game spin until it is finished
   e. Announce the winner and show how many “rounds” it took to win.
   f. Simple rules follow.

The objective of the game is to win all of the cards.

The deck is divided evenly among the players. In unison, each player reveals the top card of their deck—this is a "battle"—and the player with the higher card takes both of the cards played and moves them to the bottom of their deck. Aces are high, and suits are ignored.

If the two cards played are of equal value, then there is a "war". Both players place the next card of their pile face down (some variants have three face down cards) and then another card face-up. The owner of the higher face-up card wins the war and adds all the cards on the table to the bottom of their deck. If the face-up cards are again equal then the battle repeats with another set of face-down/up cards. This repeats until one player's face-up card is higher than their opponent's.

Submission: Your repo should be structured in the following way

- A directory for each assignment
- If the assignment is comprised of multiple tasks include a sub-directory for each task
- Push the appropriate code to the appropriate directory.
- An example structure for this assignment would be

```
week_one
    |-> fraction
          |-> Fraction.cpp
          |-> main.cpp
    |-> war
          |-> Card.h
          |-> Card.cpp
          |-> Deck.h
          |-> Deck.cpp
          |-> Hand.h
          |-> Hand.cpp
          |-> main.cpp
```