Project 5: Blind Man’s Bluff

(Due October 24 at 3:30pm)

Learning Objectives:

- Write C++ programs in a modern software development using the CLion IDE and version control with Github repositories,
- Implement two classes to satisfy the requirements of an existing driver,
- Use abstraction to hide the implementation of a class from its interface,
- Write constructors, destructors, and overload the assignment operator for a class that uses dynamic memory management, and
- Write unit tests for class methods.

Blind Man’s Bluff

The card game Blind Man’s Bluff usually works as follows: two players are each dealt a card. Without looking at the card, each player places the card on their forehead face out so their opponent can see the value. Then the players make a bet on whether or not they think their card is higher or lower than their opponent’s. The game is typically played with a standard French-style 52-card deck, which consists of cards each with one of 13 ranks (2, 3, …, 9, 10, Jack, Queen, King, Ace) and one of 4 suits (Clubs, Diamonds, Hearts, Spades), and some total ordering on the cards.

In this project, we will simulate Blind Man’s Bluff as a one player game, where the user is dealt a card, shown their opponent’s card, and then asked to guess if their card is higher or lower. A simple way to simulate the game is to compute two random numbers for each turn. There is a file named “main.cpp” in your assignment repository that implements this approach.

You will implement a more realistic simulation of Blind Man’s Bluff: your code will track a deck of 52 cards, where at each turn, two new cards are dealt from the deck and then placed in a discard pile. The player can choose to play up to 26 turns, when the deck has been exhausted. (The player can also count cards played throughout the game to make more informed decisions as the deck gets smaller.)

In addition to updating main.cpp to implement this more realistic simulation, you will implement two classes called Deck and Card. The files for implementing these classes are also already in the repository.

The requirements for the Card class are as follows:

- includes the default constructor and at least one alternate constructor,
- includes at least one overloaded comparison operator (such as < or >), using the following ordering rules:
    - the rank is more important than the suit,
    - Aces are low and Queens are high, so that A < 2 < … < 10 < J < K < Q,
    - suits are ordered alphabetically, so that Clubs < Diamonds < Hearts < Spades.
- includes one member function that returns a string specifying the value of the card in the format of “Jack of Hearts”,
- does **not** include any setters or getters (mutators or accessors) in order to keep the implementation of the class hidden from users.

The requirements for the Deck class are as follows:

- have these 3 private members:
    - Card* cards (pointer to array of cards)
    - int arraySize (physical size of array)
    - int cardsLeft (logical size of array)
- implements the “big three”: copy constructor, copy assignment operator, and destructor,
- includes a member function that shuffles the cards in the deck by performing cardsLeft^2 swaps of two randomly chosen cards in the deck,
- includes a member function that removes a card from the deck and returns its value,
- includes a member function that adds a card to the deck (or returns false if there is no physical space in the deck).

The requirements for the driver:

- asks the user to play again after each turn; the game ends when the player chooses not to play or the deck is empty,
- tracks the numbers of user’s wins and losses (correct and incorrect guesses) and report them when the game ends,
- does **not** need to include unit tests for Card and Deck member functions in its final version, but writing unit tests in the driver as you implement the classes is highly recommended!

Project 4 Invitation Link: https://classroom.github.com/a/X2lKWBkZ

Implementation Notes:
Any program that does not compile without errors (warnings okay) will not be graded and be given an automatic 0 points for that part.

Comments and Style:
Although there will be no formal policy on commenting and style, the reader should be able to easily follow the main purpose of the code. Each block of code, including each function, that

does something significant must be commented. The variable names should be easily recognizable and acronyms should be avoided if possible.

Project Submission:
You will develop your program for this project with CLion connected to a remote Github repository given to you. Points will be deducted for an incorrect filenames and no credit will be given for late submissions.

Pledged Work Policy:
Assignments in Computer Science courses may be specified as "pledged work" assignments by the professor of the course. When an assignment is specified as "pledged work" the only aid that the student may seek is from either the course professor or TAs (including CS Center tutors) that the professor has explicitly specified. On "pledged work" assignments the student may not use the services of a tutor.

You may discuss only <u>basic C++ syntax and general computer science concepts</u> with everyone else. Any other communications of the project (e.g., giving your code to someone else or seeing someone else's code) are strictly prohibited except with the professor and TAs of the course. Your code and your implementation of the project must be the product of your own work.