CS 341 – Fall 2024
Assignment #2 – Master-Card
Due: 9/27/2024

This assignment is going to build upon your existing knowledge of C++ and help reinforce the topics of Inheritance, Polymorphism, Pointers, and Memory Management that we have discussed in lecture. This project is going to simulate a Deck of Cards and ultimately allow you to play a card game with them.

This assignment will be broken into distinct phases (five (5) to be specific) in order to help you progress and track your progress. Please make sure that you complete each phase in-turn before moving on to the next (these phases are for YOUR benefit so that way you are not biting off more than you can chew).

Additionally, for this project we are going to focus on good Software Engineering techniques and methods to accomplish our task. I have provided you the basis for the assignment on Canvas (`Files` → `Assignments` → `A2`). You are responsible for downloading the associated files and modifying them as necessary. **NOTE:** You may NOT change any of the existing code (e.g., you cannot change a method to return a void when it was originally written to return a string). Make sure to review any feedback from Assignment #1, with respect to proper programming practices, before moving on to Assignment #2.

Development Process:

Phase I -- (9/18/2024)

For the first phase of this assignment you are asked to complete the implementation for our Card class. You can find the declaration of this Class described in `card.h`. You will need to complete the source/implementation (`card.cpp`) to accomplish this. Once you have completed the Card class, you can test your implementation by creating an array of 52 Cards in a `driver.cpp` (don't forget your `makefile`!). Print out the contents of that array into the console – an example of this is shown below:

```
Ace of Spades
Two of Spades
Three of Spades
Four of Spades
Five of Spades
Six of Spades
Seven of Spades
Eight of Spades
Nine of Spades
Ten of Spades
Jack of Spades
Queen of Spades
King of Spades
Ace of Hearts
Two of Hearts
Three of Hearts
Four of Hearts
Five of Hearts
...
```

If you have successfully completed this – you are ready to move on to Phase II.

Phase II – (9/20/2024)

For the second phase of this assignment, you are asked to complete the implementation for our Deck class (`standardDeck.h`). This class will be responsible for simulating a standard deck of 52 cards (see the results of Phase I). To accomplish this we are going to be using the Heap rather than the Stack when it comes to managing our memory allocation for our Card array. You will need to create a Deck object that contains 52 Card objects (an array). Using your `driver.cpp` you can now test your newly created `StandardDeck` object to iterate through your `Cards`.

Don't forget to handle deletion of your dynamically created objects (e.g., `delete` keyword & destructor `~`). No memory leaks!!!! In order to check for leaks in your code you can use the `valgrind` tool.

```
valgrind --log-file=valgrind.txt Cards.exe
```

The output for this phase should display the contents of the deck (e.g., Phase I) along with the following:

```
...
Is the deck empty? 0
Number of Cards: 52
15th Card: Three of Hearts
```

If you have successfully completed this – you are ready to move on to Phase III.

Phase III – (9/22/2024)

For the third phase of this assignment you are tasked with adding four (4) new public methods to the `StandardDeck` class and their associated functionality in the implementation file:

- `bool addCard(Card c);`
  - This method will return TRUE if the Card `c` can be added to the **end** of the Deck, otherwise it will return FALSE.
- `void shuffle();`
  - Using random sort from CS 248 perform at-least three (3) x the number of cards in the deck swaps.
- `bool mergeDecks(StandardDeck &, bool);`
  - This method should take Cards from an input `StandardDeck` and merge them with the current `StandardDeck`. The Boolean input parameter indicates whether or not we want to then shuffle (Default = `FALSE`). The output in this case should return `TRUE` on successful addition.
- `void initializeDeck();`
  - This method will be responsible for populating the `StandardDeck` `Card` array with the appropriate 52 cards.

We can now move our Card creation code into our `initializeDeck()` method and update our default Constructor accordingly. In our `driver.cpp` we can also now test the `shuffle` and `mergeDecks` methods. An example of output for this phase is shown below:

```
Is the deck empty? 0
Number of Cards: 52
15th Card: Ace of Hearts
```

If you have successfully completed this – you are ready to move on to Phase IV.

Phase IV – (9/25/2024)

For the forth phase of this assignment, you are tasked with creating a simulation for a simplification of the game of War. In order to accomplish this phase, you will need to add a `dealCard` method to our Standard Deck class:

- `Card dealCard();`
    - This method will return the top most (the Card with the highest index value) Card on the Deck.

You will also need to modify your `driver.cpp` to facilitate the "playing" of the game. You may also need some helper functions along the way…<hint, hint>

The game of Simplified War can be described as follows and consists of two (2) players:

1.) Your Standard Deck will be built with 52 cards.
2.) At the start of the game, each Player is dealt half of a deck of cards, which represents their army.
3.) Each player alternates (e.g., p1, p2, p1, etc.) placing a card on top of a central pile called the battleground.
4.) If the most recently played card matches the face of the card beneath it, that player takes all of the cards in the battleground and adds them to their army.
    a. For example, if Player 1 played a Jack of Spades on top of a Jack of Hearts, Player 1 would collect all of the cards in the battleground and add them to their army.
5.) The player who gained the reinforcements then plays the next card to start a new battleground.
6.) Play continues in this way until only one player has cards left in their deck.
7.) That player wins the round with a score equal to the number of cards in their current army.
8.) In each round, Player 1 always plays the first card.

Your simulation should provide the following output:

- Your program must simulate the game of War fifty (50) times and return the following information to the user:
    - Which player won more, and how many times they won.
    - The average score of Player 1.
    - The average score of Player 2.

An example of this output is shown below:

```
Player 1 was the champion with 26 victories versus Player 2.

Player 1 Average Score: 25
Player 2 Average Score: 24
```

Phase V

For the fifth phase of this assignment, you are tasked with extending the project to allow for non-standard decks in addition to the traditional 52-card standard deck. This will necessitate that you leverage Inheritance as well as Polymorphism in the creation of a Base class.

In this phase, you will be creating a new class called `Deck` and modify your existing classes, `StandardDeck` and `NonStandardDeck`, inherit from it. The `NonStandardDeck` will have all the same qualities as a `StandardDeck` except it will read-in a variable deck size from a text file (`deck.txt`) along with the suit and face of the cards. Non-standard decks are also not bound by size limit of 52 cards, instead they can be of any size! You must modify your `StandardDeck` class to only allow for a maximum size of 52 cards. The format of this text file (`deck.txt`) is as follows:

```
3
0 0
3 2
4 12
```

Where the first line is the size of the deck and all subsequent lines are listed by `SUIT` and `FACE` values respectively.

You will need to utilize the `virtual` keyword for your `initializeDeck()` method and make it pure virtual.

```
virtual void initializeDeck() = 0;
```

Finally, you should apply your Simplified War simulation to your `NonStandardDeck` class applying the same criteria for the simulation as in Phase IV.

An example of output for this phase is shown below:

```
Player 1 was the champion with 26 victories versus Player 2.

Player 1 Average Score: 25
Player 2 Average Score: 24
```

Once you have completed this task successfully you have successfully finished the assignment!

Submission:

This is an individual assignment and will test your ability to program using OO concepts in C++ - each student is expected to submit their own work in their course GitHub repository. All assignments must be submitted on Butler GitHub (`github.butler.edu`) in the course repository (see Assignment #1).

The directory structure of the repository must contain the following files – please note the naming:

- **driver.cpp**
- **card.h**
- **card.cpp**
- **deck.h**
- **deck.cpp**
- **nonStandardDeck.h**
- **nonStandardDeck.cpp**
- **standardDeck.h**
- **standardDeck.cpp**
- **deck.txt**
- **makefile**

It is **STRONGLY** recommended that you commit your changes after the completion of each phase – this will not only serve as a "checkpoint" for your progress but will also allow me to award partial credit in the case that a specific phase is not completed successfully.

** Don't forget: Each source file (`.h`/`.cpp`) **must** include the Honor Pledge and digital signature. **