EECE.3220: Data Structures

Spring 2019

Programming Assignment #4: Game of War

Due **Wednesday, 5/1/19**, 11:59:59 PM

1. Introduction

This assignment provides an introduction to working with queue objects. You will modify the array-based queue class we discussed in lecture to play the classic—and simple—card game of War.

This assignment was adapted from an assignment written by Professor Phil Viall at UMass Dartmouth for ECE 161: Foundations of Computer Engineering II.

2. Deliverables

You should submit five files for this assignment. Starter versions of each are available both from the course schedule page and the Blackboard assignment for this program:

- ***prog4_main.cpp:*** Source file containing your main function.
  - The starter file contains an outline of how the main function should behave. Feel free to modify this outline to fit your solution, if necessary.
- ***Card.h / Card.cpp:*** Header/source files containing the `Card` class definition and member function definitions.
  - These files are exactly what I used in my solution. While you can modify these files if you want to, you can use them without changing anything.
- ***Deck.h / Deck.cpp:*** Header/source files containing the `Deck` class definition and member function definitions.
  - The .h file contains a list of functions I believe you need for this program, and a suggested list of data members.
  - The .cpp file mostly contains empty functions that you must write. However, I've written the `fill()` function, which creates a "shuffled" deck by initializing every `Card` object randomly. Note that you'll have to change this function if you change the list of data members.
    - In particular, if you add a data member to track the number of cards, make sure that value is set to 52 at the end of the `fill()` function.
  - I strongly recommend using the array-based queue we covered in class as a reference for the `Deck` class. Some `Deck` functions are exactly the same as queue functions, while others can be similar functions modified to work for this specific program. More details are provided in Section 4: Hints.

Submit a single archive file (<u>.zip only</u>) containing <u>all .h and .cpp files</u> to the "Program 4" assignment on Blackboard.

3. Specifications

Rules: A full War ruleset can be found at https://www.pagat.com/war/war.html. Note that this site resolves "wars" slightly differently than described below.

A. The full 52-card deck is used. Suits are ignored but must be tracked to differentiate cards. Cards are ranked from high to low as follows: A K Q J T 9 8 7 6 5 4 3 2

B. The deck is shuffled and dealt so each player has a 26-card "deck." The object of the game is to win all the cards.

C. Each round of gameplay proceeds as follows:

- **i.** Each player turns his or her top card face up.
- **ii.** Whoever turned the card of higher rank wins both cards and places them at the bottom of their deck.
- **iii.** If the cards are of equal rank, a "war" starts.
  - **a.** In a war, each player deals three cards off the top of his or her deck, then turns a fourth card face up.
  - **b.** The higher of these last two cards wins the war, thus allowing the winning player to take all cards played in the war.
  - **c.** If the turned cards are equal, another round of war repeats—each player deals three more cards, then turns a fourth card face up to resolve the war.

D. If, at any point, a player has no cards and is therefore unable to play a card when required, then the other player wins the game. Note that it is possible for one player to play his last card, win that round, and continue playing the game.

Input: Your main program only needs to take a single input value—a seed value to be provided to the random number generator by calling the function `srand()` exactly once near the beginning of `main()`.

The `fill()` function in the `Deck` class, which I've written for you, is the only function that uses random numbers, so don't worry about the details of random number generation if you're unfamiliar with it.

Output: After prompting for and reading a seed value, your program should play out each turn of the game, printing the outcome on one or more lines. Each line should contain:

- The number of cards in each player's deck
- The card each player turns up at the start of the turn
- The outcome of the turn

If the turn involves a war, then the output should include the extra three cards dealt out of each player's hand. Treat the fourth card—the one that potentially resolves the war—as a new turn.

See Section 5: Test Cases for examples of turn-by-turn outputs.

4. Hints

Some notes on the `Deck` class:

- Use an array-based queue—refer to Lecture 27 for more details. You don't need to dynamically allocate anything because you know the exact size of a full deck.
- Allow deck size to be flexible, with cards removed from a deck no longer stored in that `Deck` object. My solution stores three `Deck` objects in `main()`:
  - The two player decks, which start empty and are then filled with 26 cards apiece
  - The cards placed on the table during each turn
    - I've declared this `Deck` and called it "table" in the main starter file
    - This `Deck` can represent the initial 52-card deck that is dealt into the two player decks (so it's empty once the players have their cards)
    - In each turn, this `Deck` starts empty, fills up as players remove cards from their `Deck`, and then empties into the appropriate player's `Deck` once one player wins a turn or a war.

And a couple of notes on using Card functions:

- The `compare()` function returns an integer based on the relationship between the calling object (the `Card` it's called on) and the `Card` you pass as an argument:
  - 0 if the cards are "equal" (same rank)
  - 1 if the calling object is "higher" (if the calling object beats the other card)
  - -1 if the calling object is "lower" (if the other card beats the calling object)

  So, given 3 cards: `c1` is the ace of hearts (`Ah`), `c2` is the 9 of clubs (`9c`), and `c3` is the 9 of diamonds (`9d`):
  - `c1.compare(c2)` returns 1 (`Ah` has a higher rank than `9c`)
  - `c2.compare(c3)` returns 0 (`9c` and `9d` have the same rank)
  - `c3.compare(c1)` returns -1 (`9d` has a lower rank than `Ah`)
- The `printCard()` function takes the output stream to which it prints as an argument
  - `c1.printCard(cout)` prints the contents of `c1` to the screen

More hints may be added as necessary!

5. Test Cases

Your output should closely match these test cases in terms of format and general functionality. Your program should behave the same each time you use the same seed, but it may not match my output *(which is missing almost 10 pages in areas shown in red. Games take many rounds, but thankfully not many compute cycles, to end).*

```
Enter seed: 1                            Remember, seed is only user input!
[A:26, B:26] A: As, B: Qh --> A wins!
[A:27, B:25] A: Ks, B: Ah --> B wins!
[A:26, B:26] A: 6h, B: 4c --> A wins!
[A:27, B:25] A: Qs, B: 5s --> A wins!
[A:28, B:24] A: 3s, B: Tc --> B wins!
[A:27, B:25] A: 5c, B: 2c --> A wins!
[A:28, B:24] A: 6d, B: 5h --> A wins!
[A:29, B:23] A: 9s, B: Jd --> B wins!
[A:28, B:24] A: 2h, B: 3c --> B wins!
[A:27, B:25] A: Kh, B: Jh --> A wins!
[A:28, B:24] A: 8c, B: 9d --> B wins!
[A:27, B:25] A: 4h, B: Js --> B wins!
[A:26, B:26] A: 7c, B: 5d --> A wins!
[A:27, B:25] A: 8h, B: Kd --> B wins!
[A:26, B:26] A: 3h, B: Qd --> B wins!
[A:25, B:27] A: Td, B: 9c --> A wins!
[A:26, B:26] A: 8d, B: Ad --> B wins!
[A:25, B:27] A: 9h, B: 7s --> A wins!
[A:26, B:26] A: Jc, B: 4s --> A wins!
[A:27, B:25] A: 2s, B: Ac --> B wins!
[A:26, B:26] A: 7h, B: Th --> B wins!
[A:25, B:27] A: 6c, B: 2d --> A wins!
[A:26, B:26] A: 4d, B: 3d --> A wins!
[A:27, B:25] A: 6s, B: Kc --> B wins!
[A:26, B:26] A: 8s, B: Qc --> B wins!
[A:25, B:27] A: 7d, B: Ts --> B wins!
[A:24, B:28] A: As, B: Ks --> A wins!
[A:25, B:27] A: Qh, B: Ah --> B wins!
[A:24, B:28] A: 6h, B: 3s --> A wins!
[A:25, B:27] A: 4c, B: Tc --> B wins!
[A:24, B:28] A: Qs, B: 9s --> A wins!
[A:25, B:27] A: 5s, B: Jd --> B wins!
[A:24, B:28] A: 5c, B: 2h --> A wins!
[A:25, B:27] A: 2c, B: 3c --> B wins!
[A:24, B:28] A: 6d, B: 8c --> B wins!
[A:23, B:29] A: 5h, B: 9d --> B wins!
[A:22, B:30] A: Kh, B: 4h --> A wins!
[A:23, B:29] A: Jh, B: Js --> WAR!! <A:7c, B:8h; A:5d, B:Kd; A:Td, B:3h>
[A:19, B:25] A: 9c, B: Qd --> B wins!
                  :                            "Missing" outcomes
[A:31, B:21] A: 8c, B: Jd --> B wins!
[A:30, B:22] A: 7d, B: 7s --> WAR!! <A:3h, B:9c; A:Qh, B:4s; A:5d, B:Qd>
[A:26, B:18] A: 8h, B: 8s --> WAR!! <A:3s, B:Ad; A:Qs, B:6c; A:7h, B:Th>
[A:22, B:14] A: Ac, B: 8d --> A wins!
                  :                            More "missing" outcomes
[A:50, B:2] A: Kc, B: 3d --> A wins!
[A:51, B:1] A: Qd, B: Jd --> A wins!
PLAYER A WINS!!!
```