# EECE.3220: Data Structures

Spring 2017

Programming Assignment #5: Game of War

Due **Wednesday, 4/26/17**, 11:59:59 PM

## 1. Introduction

This assignment provides an introduction to working with queue objects. You will revisit the card and deck classes written in Program 3, modifying them (if necessary) 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:

- ***prog5_main.cpp:*** Source file containing your main function.
- ***Card5.h / Card5.cpp:*** Header/source files containing the `Card` class definition and member function definitions.
    - ***Note:*** The '5' in the file name is intended to differentiate this file from your Program 3 submission. You may simply copy the contents of your Program 3 `Card` files to these files, either as a starting point or as the full `Card` definition.
- ***Deck.h / Deck.cpp:*** Source file containing definitions of `Deck` member functions.
    - ***Note:*** You may use your `DeckOfCards` class from Program 3 as a starting point, but I would recommend reorganizing that class as a queue of `Card` objects and adding functions only to support the operations necessary for this game. More details are provided in Section 4: Hints.

Submit all five files by uploading them to your Dropbox folder. Ensure your file names match the names specified above. Do not place these files in a subdirectory—place them directly in your shared folder. Failure to meet this specification will reduce your grade, as described in the program grading guidelines.

***NOTE:*** Students who wish to use my `Card`/`DeckOfCards` solutions as starting points for this assignment may e-mail me directly. *Note that anyone requesting these solutions will be ineligible both for (1) extra credit points, and (2) regrade requests on Program 3. You may submit a regrade request for Program 3 prior to requesting these solutions and then use them in Program 5.*

# 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()`.

**Output:** After prompting for and reading the 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.

**EXTRA CREDIT:** You may earn up to **5 points** of extra credit for rewriting your `Card` definition from Program 3 as follows:

- Replace the `printCard()` function with an overloaded output operator.
- Write overloaded comparison operators to allow you to compare `Card` objects by rank. You'll need at least the == operator and one of the < or > operators.

# 4. Hints

While you likely can write this program using your Program 3 solutions, I'd rewrite your `Deck` class as follows:

- Use an array-based queue—refer to Lectures 22 and 23 for more details. You don't need a linked queue because you know the exact size of the 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
    - This deck can represent the initial 52-card deck that is dealt into the two player decks
    - 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.
- While some of the Program 3 functions are still useful, others may need to be modified or replaced. My `Deck` class contains functions to do the following:
  - Construct an empty deck *(not a deck containing all 52 cards)*
  - Indicate if the deck is empty
  - Deal (remove) the top card of the deck and return it
  - Add a card to the bottom of the deck
  - Return the number of cards currently in the deck
  - Fill the deck by generating all 52 unique cards, then shuffling them
    - I call this function only once—before dealing 26 cards to each player near the start of the program

Also, outside of the extra credit, you shouldn't need to do anything special with the `Card` class. In particular, remember the default assignment operator does a member-by-member copy between objects of the same type, so as long as you don't dynamically allocate anything inside a `Card`, that default operator is sufficient for any assignments you may need to perform with those objects.

## 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
[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!!!
```