Lab 5: Using C++ Standard Library Containers, Algorithms, and Iterators
CS-240: Data Structures
Tuesday October 14, 2014

Goal

The goal of Lab 5 is to explore and become familiar with the Standard Library containers algorithms, and iterators, and to begin to select data structures based on the requirements of a problem or program specification. You will implement a simple program that carries out a game of *War* between two participants, displaying the game's progress as it proceeds, and announcing the winner at the end. You will use the resulting code for this lab in Program 2, so build robust complete C++ classes. You will receive details about Program 2 during next week's lab period.

Background

War is a very simple 2-player card game that requires no strategy, but lots of patience. It works as follows. A single deck of cards is shuffled and dealt out to the two players, 26 cards each. Each player holds her hand of cards in a single ordered pile, without looking at them. Each player then *battles* by turning over a single card from the top of her hand of cards, and comparing its value with the other player's card. Whichever player has the higher value card wins the battle, and gets to add both cards to the bottom of her pile. (Aces are high, 2's are low.) The players then turn over their next top cards, for the next battle, and continue.

If the two players turn over cards with the same value, they have a *war* wherein they take the next three cards off their deck and put them in the middle, then turn over and compare their 4th cards. Whichever player has the highest value on this 4th card wins the war and gets to keep all ten cards. A second ties causes players to place three more cards in the middle and again compare values of the next card. This continues until one player wins the war. If one player does not have enough (four) cards to participate in a war, she puts as many cards as she can in the middle, and saves and compares her last card. (If she loses that war, she loses the game; if she wins, she wins all the cards she put in, plus all the cards the other player put in, and stays alive.) For our purposes, if a player ties in a battle or war with her very last card, she loses the game.

Play continues until someone (the winner) has all 52 cards in the deck, or until the players have battled 10,000 times, whichever comes first.

Specifications

Implement a C++ program that uses Standard Library containers to store playing cards and to simulate a single game of war between two players. Implement a C++ class for a single playing card, and create a deck of cards by *selecting from the Standard Library* a container to store 52 instances of your class. Shuffle the deck with an algorithm from the Standard Library, and deal 26 cards to each player. Have the players battle and exchange cards according to the rules of the game, until one player holds all the cards, or until the players have battled 10,000 times, whichever comes first. (There could be cases where the game continues with players exchanging cards back and forth in a deterministic pattern; we would like to end the game if that happens.)

Your program should read in the names of the two game participants from a file, and write out the results of every battle into a log file with a *very specific format*. Both the Player File (the input file) and the Log File (the output file) will be specified as command line arguments to your program, which should be named *war.exe*. That is, users invoke your program as follows:

```
war.exe <player_file> <log_file>
```

The Player File contains the names of two participants, one name on each of the first two lines. You should count the participants and ensure that there are exactly two. If not, your program should print an appropriate and informative error message, and exit.[1]

For example, a valid Player File might look like this:

```
Jane
John
```

Upon reading a valid Player File, your program should then carry out a single game of war between the two participants named in that file. The program should create and write the results into the Log File with entries formatted *exactly* as follows *(capital letters, indenting, spaces, colons, etc., all matter!):*

```
Jane vs. John
   Battle 1: Jane (JD) defeated John (2H): Jane 27, John 25
   Battle 2: John (AC) defeated Jane (5S): Jane 26, John 26
   ...
   Battle 21: John (5C) defeated Jane (2S): Jane 33, John 19
   Battle 22: John (4H) tied Jane (4C)
   War 1: Jane (QC) defeated John (10D): Jane 38, John 14
   ...
   Battle 120: Jane (KH) defeated John (QD): Jane 52, John 0
Jane DEFEATED John in 120 Battles and 13 Wars
```

The two-character values in parentheses indicate playing cards (e.g. "QD" for "Queen of Diamonds").

Design

Please write your own C++ `PlayingCard` class to hold the card value and suit as data members. Implement (i) the comparison operators (at least the one you use to determine the winners of battles and wars), (ii) the insertion operator (<<) to print playing cards in the requested format (e.g. "KC" for the King of Clubs), both to a file and to `cout`, and (iii) any other member functions (including constructors) that you need for the class.

You will need other classes, containers, functions, and algorithms to implement this program. Pick them from the Standard Library carefully! Use the STL Programmer's Guide or the textbook to explore the possibilities. It is very important to be able to select appropriate functions from a library. Examine the options for containers and algorithms that do what you need them to do. In particular, you will need to hold a deck of cards, shuffle it[2], and deal it out to two players who will initially hold 26 cards each (in their own containers). Then you will need to take cards out of one end of each player's hand and add them back onto the other end of the battle winner's hand. You will need to track and print the number of battles and wars that occur during the game. Your design matters for this program; pick the right tools for the jobs, so that your code is both straightforward and efficient. Don't do anything that code from the STL could do for you unless you have good reason to.

Submission

Please follow the Lab Submission Conventions described in Lab 1, to produce an appropriately named "tar ball" for Lab 5. Everything should unpack for us into a single directory named **Lastname_Firstname_Lab5**. Submit your code to Blackboard.

[1] Having two names in a file seems like overkill (why not just put them on the command line?). But Program 2 will have you read in more than two participants, for a War Tournament, so we might as well read in the names in the same way that you will eventually get them for Program 2.

[2] Hint: the Standard Library contains a "`random_shuffle()`" algorithm, but be aware that the `random_shuffle()` algorithm does not work with every container.