

CSE 20212 Fundamentals of Computing II

Spring 2008

Lab handout for Week of February 4

Objectives

1. Learn about composition and how to return a result of the same class type
2. Develop a C++ container class and learn a basic data structure
3. Implement Monte Carlo simulations for some card games
4. Have fun!

Pre-lab assignment

1. (7 points) Problem 10.9 in D & D.
2. (5 points) Submit a C++ class implementation named `CardDeck` (`CardDeck.cpp` and `CardDeck.h`) that stores cards represented as a collection of n integers. The class should have private slots for an array containing integers and the total number of objects n . The default constructor should assign 52 to n and place the integers from 1 to 52 into the card array. A nondefault constructor that receives the value of n and places the integers from 1 to n into the card array is also required (hint: use a “new deck” function to place the integers as described above). Because you may not be able to assume the value of n , use dynamic memory allocation in the constructor (**new**) and perform the necessary “clean up” in a destructor you provide (**delete**). Finally, provide three utility methods. The first, named **shuffle()**, takes no arguments, and performs a Knuth shuffle on the elements of the card array as described in the accompanying handout. The second, named **inOrder()**, returns 1 if the elements are in non-decreasing order, 0 otherwise. Finally, provide a function called **print()** that displays the elements of the card array with each member separated with the string “, “. End with an end-of-line character.

Write a test program that initializes an array of ten cards (i.e., $n=10$) and prints the deck before and after a shuffle.

3. (3 points) The infinite monkey theorem states that a monkey hitting keys at random on a keyboard for an infinite amount of time will almost surely produce any work of literature, including the complete works of Shakespeare. For this problem, you will implement related monkey sort, also called bogosort. Suppose you have a monkey, a deck of playing cards, a large spoon, a wooden tub, and a silly hat (NOTE: no animals will be hurt during this exercise). Monkey sort entails having the monkey throw the cards in the tub, stir them with the spoon and pick them up. Your job is to check if they are in order. Repeat this procedure until ordered. Because wooden tubs and monkeys are probably rare on campus, simulate monkey sort using the class developed for part 2 (a silly hat, if you have access to one, is optional). Submit your monkey sort driver program and report number of shuffles until ordered on 6, 7, 8, 9 and 10 cards, 3 times each (15 total values).

In-lab activities

1. (1 point) Report to lab **on time**. Attendance will be taken at the scheduled lab time.
2. Implement functions that return (*i.e.*, deal) the top card on the deck and that insert a card into the bottom of the deck. This should respect the “first in, first out” rule and as such can be accomplished using a queue. Consider an array for our cards with one extra element. After initializing our deck, the top card is the first element in the array (*i.e.*, index 0) and the next open spot is the extra element (*i.e.*, n). Updating either value, which should be stored as private data members, can be done as shown in this example:

```
topCard = (topCard + 1) % (n + 1);
```

3. Celebrate! You have just done your first basic data structure in Fundamentals II in C style. Later, we will discuss how this can be done using the STL.
4. Implement a member function that receives another CardDeck object and places those cards on the bottom its deck. Call this function **addDeck**.
5. Code up a Monte Carlo simulator for the children’s game “War”. Two people play war as follows: a standard 52-card deck is shuffled and divvied evenly between players in an alternating fashion. To play, each person places a single card down face up. Whoever has the highest card places wins both cards. In the event of a tie, a “war” takes place; each player places 3 cards face down and one card face up. The highest card takes all played cards. The “war” procedure is repeated until either a higher card is played or one player runs out of cards. In our CSE20212 rules, an Ace is the highest card and whenever a player runs out of cards, including during a war, that player loses. Implement a conversion function that converts a card from 1 to 52 to 0 to 12 (hint: use the modulus operator with 13). You must use your revised CardDeck container class for the initial deck, the two player decks, and to keep track of the piles formed during wars. After a war, the addDeck member function must be used to collect the cards won.
6. (4 points) Flag down the lab TA and have them examine and check off your work.

Post-lab (due at the start of next week's lab)

Write and submit (in your dropbox) a lab report with the following sections.

- (5 points) Although War is a very simple game of random chance, something akin to genetic selection takes place; the probability for success is not uniform with respect to deck size. To test this, keep track of adds and deletes in a deck and use a simple new member function **size()** to return the current number of cards held. Keep track of the wins/losses/ties for 10,000 games for each value (1 ... 52) in the simulator (there will be two data points per play). Plot the % loss for all 52 values. Repeat for 1,000,000 games and plot. Submit both graphs with your report.
- (5 points) Functional code for the modified CardDeck class and “war” simulator.
- (5 points) A one-page (or so) description of the problem, your solution, and your own explanation of why the winning rate changes (or not) based on deck size.

Code is expected to be well-commented, and the narrative portions of your report must be written professionally (*i.e.*, complete sentences, correct grammar and punctuation, consistent tense and voice, formal tone).