CSE 20212 Fundamentals of Computing II
Spring 2016

Lab handout for Week of February 8

Objectives

1. Practice with the Standard Template Library (Chapter 22)
2. Implement an impractical sort and a popular card game
3. Have fun!

Part 1: Simple Card Deck object (in-lab activity)

1. Report to lab **on time**. Attendance will be taken at the scheduled lab time.

2. Develop a C++ class named CardDeck that stores a deck of cards as an internal STL deque of *n* integers (to practice data hiding).

3. The non-default constructor should receive a value *n* and places the integers from 0 to *n* - 1 into the deque using **push_back**. Note that no explicit memory allocation is required when using a deque.

4. Please specify a "fall back" value for this non-default constructor so n is assumed to be 52 (aka CardDeck(int = 52); as a prototype)

5. Write a public member function **getSize**() that returns the size of the current deck using the **size()** member function of the deque.

6. Provide a utility function named **`shuffle`** that performs a shuffle using the **random_shuffle** algorithm of the STL. Note that **srand**() must be called to change the random number seed for the shuffle. For now, we recommend you simply place this **srand()** call in a very simple driver program that instantiates a CardDeck.

7. Add a public member called **`inOrder()`**, that returns 1 if the elements are in non-decreasing order (i.e., A[*i*] <= A[*i* + 1] for all cards in the deck), 0 otherwise.

8. Check out with a TA when you have completed your in-lab portion. If you have time, you can proceed to Part II.

Part 2: Augmenting our simple Card Deck object and learning about iterators

1. Overload the output operator such that it displays the elements of the card array with each element separated with ", " and ending with an end-of-line character. You must use an iterator to receive full credit on this problem (refer to rubric). Examples of what would qualify include ostream iterators/copy (see listexamp.cpp from the 2/8 notes) or deque<int> iterators (see vecexample.cpp from the 2/8 notes).

2. Develop an alternative function called printReverse() that displays the cards like your overloaded output operator, but in reverse order. Your solution must use either reverse iterators (see 2/9 vecexample.cpp on the online notes) or the reverse algorithm in the STL to receive full credit (refer to rubric). Note the reverse algorithm requires including <algorithm> and takes iterators as input … ask us if this is unclear.

3. Submit a test program for grading that initializes an array of ten cards (i.e., *n* = 10), ensures they are in order, runs the printReverse() function, and uses the output operator to print the deck before and after a shuffle.

Part 3: Taking the next step: sorting and games

1. Suppose you have a monkey, a deck of playing cards, a large spoon, a wooden tub, and a silly hat at home (NOTE: no animals will be hurt during this exercise and wooden tub is optional). A monkey-driven sort would involve the monkey throwing the cards in the tub, him/her stirring them with the spoon to ensure mixing, picking them up and then you checking if they are in order. Because monkeys are probably rare on campus, simulate "monkey sort" using your CardDeck class developed in Part 1 and specifically the shuffle functions and inOrder() member functions. You must also report number of shuffles until ordered on 7, 8, and 9 cards, 3 times each (9 total values, see rubric for details). HINT: on student00.cse.nd.edu (and other places we will grade), 342,189 shuffles are required for to resort a deck with 9 cards when you use the default random number seed (i.e., no call to **srand()**). Submitting a program that does the monkey sort (and report) three times for each deck size, or a program that just does it once, are both fine; however, you must record nine values in your report.

2. Blackjack is a simple and popular card game. A player competes with a dealer with the goal of coming as close to 21 as possible without going over. Players that exceed 21 lose; if the final value is less than 21, the dealer takes cards up until 17 or higher. If the player's value is higher, they win; otherwise they lose. Implement a simple BlackJack player using CardDeck as a foundation as follows: Aces are always 11, players have only two options (hit or stand), and dealers always stop when they reach 17 or higher. You will need to implement a **Deal()** public member function (or similar) to get cards from your deck for the game (and additional functions as necessary). Keep track of how many times a player wins and the dealer wins in your game, and ask the user if they want to continue playing after each game. When there are fewer than 15 cards, open a "new" deck and shuffle it before continuing the game. HINTs: You can use the modulus (%) operator to convert cards from 0-51 to 0-12, and you can use new/delete to ensure a constructor runs or a deconstructor runs for your decks.

Coder challenge! (optional)

Option 1: Code up a simulation for the children's game "War". First, a 52-card deck is shuffled and divvied evenly between players. Next, each person places a single card for the other to see. Whoever has the highest card wins both cards. If there is a tie, "war" takes place; each player places 3 cards face down and one card face up. The highest showing card takes the combined pile. The "war" procedure is repeated until one player runs out of cards. This implies that if a war is declared and one of the players does not have 3 cards, they lose. Finally, aces can be either high or low depending on personal preference.

Up to $8 coder dollars will be awarded if a working War simulator is implemented using the core CardDeck in lab this week. The output of the program should be a line by line occurrence of what cards were played for each player, who wins, and the size of the deck of each player. The game ends when one player has all 52 cards.

Option 2: Develop an AI for a BlackJack against a computer dealer. For simplicity, assume all bets are $5 and that the bank can go negative if you lose. Also, assume there is one deck (that minimizes house advantage). Up to $8 coder dollars will be awarded based on your implementation.

NOTE: Either Option1 or Option 2 must be chosen. You cannot do both.