CSE 20212 Fundamentals of Computing II
Spring 2017

Lab handout for Week of January 30

Objectives

1. Review basic sequence containers by implementing a card deck object.
2. Implement simple polymorphic behavior 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, dynamically allocated array 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 internal array. New should be called appropriately in the constructor.

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). NOTE: This makes a default constructor redundant and your program will not work with a default one.

5. Write a public member function getSize() that returns the size of the current deck, stored as a separate private data member in your class.

6. Provide a utility function named shuffle that performs a shuffle using Knuth’s algorithm discussed in class. Note that srand() must be called to change the random number seed for the shuffle; do so in your main function.

7. 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.

8. Submit a test program for grading that initializes an array of ten cards (i.e., n = 10), and then uses the output operator to print the deck before and after a shuffle.

9. If you have time, you can proceed to Part II.

Part 2: Introducing yourself to inheritance and polymorphism

1. Develop an abstract base class for a collection of objects that have diverse behaviors. This can be characters in a video game (e.g., Pokemon), characters in TV/movies, or any other “is a “ hierarchy you are interested in.

2. Generate at least two derived classes that inherit from your base class with an
action (catch phrase / action / attack). Make that action a virtual function. For example, Terminator and Rebel could be derived classes of MovieChar and contain a simple action “catchphrase” that would display to the user either “I’ll be back” or “May the force be with you” based on which type of object.

3. Create a simple driver program that instantiates at least 6 derived objects and uses polymorphism to invoke derived class behavior through a base class pointer.

**Part 3: Making another simple game**

Blackjack is a simple and popular card game. Unlike other card games, though, the odds can shift based on prior cards… and Ed Thorp used his mathematical analysis for this game to become the first ‘quant’ (and a very successful one). If you are interested, NPR has a nice (and very recent) podcast you can look into:

http://www.npr.org/sections/money/2017/01/20/510810752/episode-749-professor-blackjack

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 easily check and reallocate cards in your deck.

**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 Option 1 or Option 2 must be chosen. You cannot do both.